Disclaimer and Legal Information
INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R) PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.
Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

This document contains information on products in the design phase of development.


* Other names and brands may be claimed as the property of others.

Copyright (C) 1996-2008, Intel Corporation. All rights reserved.

Portions Copyright (C) 2001, Hewlett-Packard Development Company, L.P.
# Table of Contents

Intel(R) C++ Compiler User and Reference Guides .................................................. 1  
Intel® C++ Compiler User and Reference Guides .................................................. 1  
Disclaimer and Legal Information ........................................................................ 1  
Welcome to the Intel® C++ Compiler ..................................................................... 2  
See Also .................................................................................................................. 3  
Conventions ............................................................................................................ 3  
   Notational Conventions ....................................................................................... 3  
Introduction to the Intel® C++ Compiler ............................................................... 6  
   Product Website and Support ........................................................................... 6  
   System Requirements ......................................................................................... 6  
   FLEXlm* Electronic Licensing .......................................................................... 6  
Related Publications ............................................................................................... 7  
   Associated Intel Documents ............................................................................. 7  
   Optimization and Vectorization Terminology and Technology ....................... 8  
   Additional Training on the Intel® C++ Compiler .............................................. 8  
Building Applications ............................................................................................ 9  
   Overview: Building Applications ...................................................................... 9  
   Getting Started ................................................................................................ 9  
   Building Applications with Eclipse* ............................................................... 15  
   Building Applications from the Command Line ............................................ 36  
   Using Preprocessor Options ............................................................................ 50  
   Modifying the Compilation Environment ....................................................... 53  
   Debugging ....................................................................................................... 59  
   Creating and Using Libraries ......................................................................... 62  
   gcc* Compatibility .......................................................................................... 73  
   Language Conformance .................................................................................. 83  
   Porting Applications ......................................................................................... 86  
   Error Handling ............................................................................................... 96  
   Reference ....................................................................................................... 109
# Table of Contents

Intel(R) C++ Compiler Pragmas............................................................... 1488
Intel(R) Math Library ........................................................................... 1537
Intel C++ Class Libraries .................................................................... 1581
C/C++ Language Extensions ................................................................. 1664
Disclaimer and Legal Information

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL(R) PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is
subject to change without notice. Do not finalize a design with this information. The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request. Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See http://www.intel.com/products/processor_number for details.

This document contains information on products in the design phase of development.


* Other names and brands may be claimed as the property of others.

Copyright (C) 1996-2008, Intel Corporation. All rights reserved.

Portions Copyright (C) 2001, Hewlett-Packard Development Company, L.P.

Welcome to the Intel® C++ Compiler

The Intel® C++ Compiler lets you build and optimize C/C++ applications for the Linux* OS (operating system). You can use the compiler on the command line or in the Eclipse* integrated development environment.
See Also
- Introduction
- Building Applications
- Compiler Options
- Optimizing Applications
- Floating-point Operations
- Compiler Reference
- Intrinsics Reference

For details on getting started with the Intel C++ Compiler, see:
- Getting Started
- Invoking the Compiler from the Command Line

Conventions
Information in this documentation applies to all supported operating systems and architectures unless otherwise specified.

This documentation uses the following conventions:

Notational Conventions

This type Indicates command-line or option arguments.

This type Indicates a code example.

This type Indicates what you type as input.

This type Indicates menu names, menu items, button names, dialog window names, and other user-interface items.

File>Open Menu names and menu items joined by a greater than (> ) sign indicate a sequence of actions. For example, "Click File>Open" indicates that in the File menu, click Open to perform this action.

{value | value} Indicates a choice of items or values. You can usually only choose one of the values in the braces.

[item] Indicates items that are optional.
item [, item]... Indicates that the item preceding the ellipsis (three dots) can be repeated.

Windows* OS These terms refer to all supported Microsoft* Windows* operating systems.

Windows operating system

Linux* OS These terms refer to all supported Linux* operating systems.

Linux operating system

Mac OS* X These terms refer to Intel®-based systems running the Mac OS* X operating system.

Microsoft Windows XP* An asterisk at the end of a word or name indicates it is a third-party product trademark.

compiler option This term refers to Windows* OS options, Linux* OS options, or MAC OS* X options that can be used on the compiler command line.

Conventions Used in Compiler Options

/option or -option A slash before an option name indicates the option is available on Windows OS. A dash before an option name indicates the option is available on Linux OS* and Mac OS* X systems. For example:

Windows option: /fast

Linux and Mac OS X option: -fast

Note: If an option is available on Windows* OS, Linux* OS, and Mac OS* X systems, no slash or dash appears in the general description of the option. The slash and dash
will only appear where the option syntax is described.

\[\text{/option:argument}\] Indicates that an option requires a argument (parameter).

or

\[\text{-option argument}\] For example, you must specify an argument for the following options:

Windows OS option: \(/\text{Qdiag-error-limit:}n\)

Linux OS and Mac OS X option: \(-\text{diag-error-limit }n\)

\[\text{/option:keyword}\] Indicates that an option requires one of the keyword values.

or

\[\text{-option keyword}\]

\[\text{/option[:keyword]}\] Indicates that the option can be used alone or with an optional keyword.

or

\[\text{-option [keyword]}\]

\[\text{option[n]}\] Indicates that the option can be used alone or with an optional value; for example, in \(/\text{Qunroll}[n]\) or \(-\text{unroll}[n]\), the \(n\) can be omitted or a valid value can be specified for \(n\).

\[\text{option[-]}\] Indicates that a trailing hyphen disables the option; for example, \(/\text{Qglobal_hoist-}\) disables the Windows OS option \(/\text{Qglobal_hoist}\).  

\[\text{[no]option or [no-]option}\] Indicates that "no" or "no-" preceding an option disables the option. For example:

In the Windows OS option \(/\text{[no]traceback}\), \(/\text{traceback}\) enables the option, while \(/\text{notraceback}\) disables it.

In the Linux OS and Mac OS X option \(-\text{[no-]}\text{global_hoist}\), \(-\text{global_hoist}\) enables the option, while \(-\text{no-global_hoist}\) disables it.

In some options, the "no" appears later in the option
name; for example, -fno-alias disables the -falias option.

Introduction to the Intel® C++ Compiler
The Intel® C++ Compiler can generate code for IA-32, Intel® 64, or IA-64 applications on any Intel®-based Linux* system. IA-32 applications (32-bit) can run on all Intel®-based Linux systems. Intel® 64 applications and IA-64 applications can only run on Intel® 64-based or IA-64-based Linux systems.

For more information about the compiler features and other components, see your Release Notes.

This documentation assumes that you are familiar with the C++ programming language and with your processor's architecture. You should also be familiar with the host computer's operating system.

Product Website and Support
For general information on support for Intel software products, visit the Intel web site http://developer.intel.com/software/products/

At this site, you will find comprehensive product information, including:

- Links to each product, where you will find technical information such as white papers and articles
- Links to user forums
- Links to news and events

To find technical support information, to register your product, or to contact Intel, please visit: http://www.intel.com/software/products/support/

For additional information, see the Technical Support section of your Release Notes.

System Requirements
For detailed information on system requirements, see the Release Notes.

FLEXlm* Electronic Licensing
The Intel® C++ Compiler uses Macrovision*'s FLEXlm* licensing technology. The compiler requires a valid license file in the licenses directory in the installation path. The default directory is /opt/intel/licenses.
License files have a file extension of `.lic`.
For information on how to install and use the Intel® License Manager for FLEXlm to configure a license server for systems using counted licenses, see *Using the Intel® License Manager for FLEXlm* (flex_ug.pdf).

### Related Publications

#### Associated Intel Documents

The following Intel documents provide additional information about the Intel® C++ Compiler, Intel® architecture, Intel® processors, or tools:

- *Using the Intel® License Manager for FLEXlm*
- *Intel® 64 and IA-32 Architectures Optimization Reference Manual*
- *Intel® Processor Identification with the CPUID Instruction*, Intel Corporation, doc. number 241618
- *IA-64 Architecture Assembler User's Guide*
- *IA-64 Architecture Assembly Language Reference Guide*
Most Intel documents can be found at the Intel web site
http://developer.intel.com/software/products/

### Optimization and Vectorization Terminology and Technology
The following documents provide details on basic optimization and vectorization terminology and technology:

- *Intel® Architecture Optimization Reference Manual*

### Additional Training on the Intel® C++ Compiler
For additional training on the Intel C++ Compiler, choose a course in the Intel® Software College - Course Catalog at http://shale.intel.com/SoftwareCollege/CourseCatalog.asp
For additional technical product information including white papers about Intel compilers, open the page associated with your product at http://developer.intel.com/software/products/

Building Applications

Overview: Building Applications
This section describes how to build your applications with the Intel® C++ Compiler. It includes information on using the compiler on the command line and how to use the compiler with supported integrated development environments. Also in this section are helpful topics on linking, debugging, libraries, compatibility, and language conformance.

Getting Started

Getting Started

You can invoke the Intel® C++ Compiler from a system command prompt with the icc or icpc command after setting the environment variables.

Getting Help

On the command line, you can execute icpc -help for a summary of command-line options.

Other Resources

For new features and known issues, see the Release Notes.
For general product information or information on support for Intel software products, visit the Intel web site: http://developer.intel.com/software/products/. At this site, you will find comprehensive product information, including:

- Links to each product, where you will find technical information such as white papers and articles
- Links to user forums
Compilation Phases

The Intel® C++ Compiler processes C and C++ language source files. By default, the compiler performs the compile and link phases of compilation and produces an executable file. The compiler also determines which compilation phases to perform based on the file name extension and the compilation options specified. The compiler passes object files and any unrecognized file names to the linker. The linker then determines whether the file is an object file or a library file.

Default Behavior of the Compiler

If you do not specify any options when you invoke the Intel® C++ Compiler, the compiler performs the following:

- produces an executable file
- invokes options specified in a configuration file first
- invokes options specified in the CL environment variable
- searches for header files in known locations
- sets 16 bytes as the strictest alignment constraint for structures
- displays error and warning messages
- uses ANSI with extensions
- performs standard optimizations
- on operating systems that support characters in Unicode* (multi-byte) format, the compiler will process file names containing these characters

Default Output Files

A default invocation of the compiler requires only a C or C++ file name argument, such as:

```
icpc x.cpp
```

You can compile more than one input file:
icpc x.cpp y.cpp z.cpp

This command does the following:

- Compiles and links three input source files.
- Produces one executable file, \texttt{a.out}, in the current directory.

**Using Compiler Options**

A compiler option is a case-sensitive, command-line expression used to change the compiler’s default operation. Compiler options are not required to compile your program, but they are extremely useful in helping you control different aspects of your application, such as:

- code generation
- optimization
- output file (type, name, location)
- linking properties
- size of the executable
- speed of the executable

See Option Categories below for a broader view of the option capabilities included with the Intel® C++ Compiler.

**Command-line Syntax**

When you specify compiler options on the command-line, the following syntax applies:

\texttt{icc [options] [@response\_file] file1 [file2...]}

where \texttt{options} represents zero or more compiler options

where \texttt{file} is any of the following:

- C or C++ source file (\texttt{.C .c .cc .cpp .cxx .c++ .i .ii})
- assembly file (\texttt{.s .S})
- object file (\texttt{.o})
- static library (\texttt{.a})

If you are compiling just C language sources, invoke the compiler with \texttt{icc}. You should invoke the compiler with \texttt{icpc} if you are compiling just C++ language sources or a combination of C and C++.
The optional `response_file` is a text file that lists compiler options you want to include during compilation. See Using Response Files.

The compiler reads command-line options from left to right. If your compilation includes competing options, then the compiler uses the one furthest to the right. In other words, "the last one wins." In this example:

```
icc -xP main.c file1.c -xW file2.c
```

the compiler sees `-xP` and `-xW` as two forms of the same option where only one form can be used. Since `-xW` is last (furthest to the right), it wins.

All options specified on the command line are used to compile each file. The compiler will NOT compile individual files with specific options as this example may suggest:

```
icc -O3 main.c file1.c -mp1 file2.c
```

It may seem that `main.c` and `file1.c` are compiled with `-O3`, and `file2.c` is compiled with the `-mp1` option. This is not the case. All files are compiled with both options.

A rare exception to this general rule is the `-x type` option:

```
icc -x c file1 -x c++ file2 -x assembler file3
```

where the `type` argument identifies each file type for the compiler.

**Default Operation**

The compiler invokes many options by default. For example, the `-O2` option is on by default for systems based on IA-32 architecture. In this simple example, the compiler includes `-O2` (and the other default options) in the compilation:

```
icc main.c
```

See the Compiler Options reference for the default status of each compiler option.

Each time you invoke the compiler, options listed in the corresponding configuration file (`icc.cfg` or `icpc.cfg`) override any competing default options. For example, if your `icc.cfg` file includes the `-O3` option, the compiler will use `-O3` rather than the default `-O2` option. Use the configuration file to list options you’d like the compiler to use for every compilation. See Using Configuration Files.
Finally, options used on the command-line override any competing options specified elsewhere (default options, options in the configuration file). If you specify `-O1` on the command line, this option setting would "win" over competing option defaults and competing options in configuration files, in addition to competing options in the CL environment variable.

Certain `#pragma` statements in your source code can override competing options specified on the command line. For example, if a function in your code is preceded by `#pragma optimize("", off)`, then optimization for that function is turned off, even though `-O2` optimization is on by default, `-O3` is listed in the configuration file, and `-O1` is specified on the command-line for the rest of the program.

**Using Options with Arguments**

Compiler options can be as simple as a single letter, such as `-E/E`. However, many options accept or require arguments. The `-O` option, for example, accepts a single-value argument that the compiler uses to determine the degree of optimization. Other options require at least one argument and can accept multiple arguments. For most options that accept arguments, the compiler will warn you if your option and argument are not recognized. If you specify `-O9`, for example, the compiler will issue a warning, ignore the unrecognized `-O9` option, and proceed with compilation.

While the `-O` option does not require an argument, there are other options that must include an argument. The `-I` option requires an argument that identifies the directory to add to the include file search path. If you use this option without an argument, the compiler will not finish compilation.

See the Compiler Options reference for a complete description of options and their supported arguments.

**Other Forms of Options**

You can toggle some options on or off by using the negation convention. For example, the `-complex-limited-range` option, and many others, include a
negation form, -no-complex-limited-range, to change the state of the option. Since this option is disabled by default, using -complex-limited-range on the command line would toggle it to the "ON" state.

Option Categories

When you invoke the Intel C++ Compiler and specify a compiler option, you have a wide range of choices to influence the compiler’s default operation. Intel compiler options typically correspond to one or more of the following categories:

- Advanced Optimization
- Code Generation
- Compatibility
- Component Control
- Data
- Deprecated
- Diagnostics
- Floating Point
- Help
- Inlining
- Interprocedural Optimizations (IPO)
- Language
- Linking/Linker
- Miscellaneous
- OpenMP and Parallel Processing
- Optimization
- Output
- Profile Guided Optimization (PGO)
- Preprocessor

To see which options are included in each category, invoke the compiler from the command line with the -help category option. For example:

```bash
icc -help codegen
```
will print to *stdout* the names and syntax of the options in the Code Generation category.

**Using Compiler Options in the Integrated Development Environment**

If you use the Intel compiler with the Eclipse integrated development environment (IDE) to build your applications, you can specify compiler options for the entire project or individual source files. The compiler integration with the IDE also lets you specify options on the command-line if you need an option that’s not included on the IDE Property Pages.

See [Building Applications with Eclipse](#)

**Saving Compiler Information in Your Executable**

If you want to save information about the compiler in your executable, use the -sox (Linux* OS and Mac* OS X) or /Qsox (Windows*) option. When you use this option, the following information is saved:

- compiler version number
- compiler options that were used to produce the executable

**On Linux OS and Mac OS X:**

To view the information stored in the object file, use the following command:

```
objdump -sj comment a.out
strings -a a.out |grep comment:
```

**On Windows OS:**

To view the linker directives stored in string format in the object file, use the following command:

```
link /dump /directives filename.obj
```

In the output, the `-comment` linker directive displays the compiler version information.

To search your executable for compiler information, use the following command:

```
findstr "Compiler" filename.exe
```

This searches for any strings that have the substring "Compiler" in them.

[Building Applications with Eclipse](#)
Overview: Eclipse Integration

Eclipse* is an open source software development project dedicated to providing a robust, full-featured, commercial-quality, industry platform for the development of highly integrated tools. It is an extensible, open-source integrated development environment (IDE).

The CDT project is dedicated to providing a fully functional C/C++ IDE for the Eclipse platform. CDT is layered on Eclipse and provides a C/C++ development environment perspective.

**Note**

Eclipse and CDT are not bundled with the Intel compiler. They must be obtained separately.

The Intel C++ Compiler for Linux* OS provides an integration (also known as an extension) to the Eclipse/CDT IDE that lets you develop, build, and run your Intel C/C++ projects in a visual, interactive environment.

Similarly, the Intel® Debugger for Linux* OS includes a debugger integration to Eclipse and CDT. This integration is also included with the compiler installation.

To use the compiler and the debugger integrations, add them to Eclipse using the Add an Extension Location feature. For more information, see Starting Eclipse.

This section includes the following topics:

- Starting Eclipse
- Creating a New Project
- Setting Properties
- Project Types and Makefiles

See Also

http://www.eclipse.org/ for further information about Eclipse
http://www.eclipse.org/cdt/ for further information about CDT

Multi-version Compiler Support
For Eclipse Executable, Shared Library, Static Library, and Makefile projects, you can select different versions of the Intel compiler to compile your Eclipse Intel project. Eclipse configurations (and the toolchain that the configuration is based on) are used to provide this support. Refer to the Release Notes for a list of the currently supported compiler versions by platform.

Select the version of the Intel compiler to build your project with. You do this by selecting the configuration associated with the desired version of the compiler prior to building your project. You can create the desired configurations for the versions of the compiler that you would like to build either when you first create an Intel project, or, later, through the Manage Configurations interface within the IDE, accessible via the project's properties.

To create configurations using the Manage Configurations interface:

1. Right click the project, select Properties > C/C++ Build>Settings
2. On the Configuration line, select the Manage configurations button and then New.

Within configurations, you can set distinct project properties, like compiler options, to be used with different versions of the Intel compiler and freely select and modify which version of the compiler with which to build by changing the active configuration. The active configuration is the configuration that is in effect when your application is built within the Eclipse IDE.

If you have multiple instances of the same major version of the compiler installed on your system (and, for an Eclipse Executable, Shared Library, or Static Library Project, a configuration with that major version is active) with different minor versions, the Eclipse IDE will, by default, use the compiler with its environment established, via execution of <install-dir>/bin/iccvars.*sh. If no compiler environment is established for such a project, then the most current compiler, that is, the one with the highest minor version number, will be used. For an Eclipse/CDT Makefile Project, the compiler environment must be established to enable successful invocation of the compiler, by default. Note also, that for any
project, you can set the compiler environment by specifying it within Eclipse. This compiler specification overrides any specification established outside of Eclipse.

**Starting Eclipse**

If Eclipse and the CDT are installed on your system, follow these steps to use the Intel® C++ Compiler with Eclipse:

1. Initialize the compiler environment by setting environment variables. You can do this by executing `iccvars.sh` (or `iccvars.csh`) with the 'source' command. For example, for a root installation to the default directory:
   ```bash
   source /opt/intel/Compiler/11.0/package_id/bin/iccvars.sh <arg>
   ```
   The script takes an argument specifying architecture:
   - `ia32`: Compiler and libraries for IA-32 architectures only
   - `intel64`: Compiler and libraries for Intel® 64 architectures only
   - `ia64`: Compiler and libraries for IA-64 architectures only

2. Be sure the `LANG` environment variable is set correctly:
   ```bash
   export LANG=en_US
   ```

3. Start Eclipse and indicate the JRE, for example:
   ```bash
   <eclipse-install-dir>/eclipse/eclipse -vm <jre-install-dir>/jrockit-R26.4.0-jrel.5.0_06/bin/java -vmargs -Xmx256m
   ```

To add the Intel C++ Compiler product extension to your Eclipse configuration, follow these steps from within Eclipse:

1. Open the **Product Configuration** page by selecting **Help > Software Updates > Manage Configuration**
2. Under **Available Tasks**, select **Add An Extension Location**. A directory browser will open.

3. Browse to select the appropriate Eclipse directory for the version of CDT you want to integrate with. Intel integrations are provided for CDT versions 4.0 and 5.0. For example, if you installed the compiler as root to the default directory and want to integrate with CDT version 4.0, browse to 

   /opt/intel/Compiler/11.0/package_id/eclipse_support/cdt4.0/eclipse.

4. When asked to restart Eclipse, select **Yes**. When Eclipse restarts, you will be able to create and work with CDT projects that use the Intel C++ compiler.

If you also installed the Intel Debugger (idb) product extension along with the idb Eclipse product extension and would like to use idb within Eclipse, you should add the idb product extension site to your Eclipse configuration in the same way. For example, if you installed idb as root to the default directory, the idb Eclipse product extension site will be located at

   /opt/intel/Compiler/11.0/package_id/idb/eclipse_support/cdt4.0/eclipse.

### Creating a Simple Project

**Creating a New Project**

To create a simple project, start Eclipse* and follow these steps:

1. From the Eclipse **File** menu, select **New > C Project**. The **C Project** wizard opens.

2. For **Project name**, type **hello_world**. Check the **Use default** box or specify a directory for your Executable C Project.

3. In the **Project Types** list, expand the **Executable** project type list using the arrow character and select **Hello World ANSI C Project**. In the **Toolchain** list, select **Intel(R) Toolchain for Executable on platform-id** where **platform-id** is either **IA-32** or **Intel® 64**.
Click Next.

4. The Basic Settings page allows you to specify template information, such as Author and Copyright notice, which will appear as a comment at the top of the generated source file. You can also specify the Hello world greeting string to be displayed by your hello_world program as well as a Source directory relative to the project where your generated source file will be created. When you are done filling in the desired fields, click Next.

5. The Select Configurations page opens, allowing you to specify the platforms and configurations for deployment. By default, a Debug and Release configuration will be created for the selected toolchain. Click Finish to complete creation of your new hello_world project.

6. If you are not currently in the C/C++ Development Perspective, you will see the Open Associated Perspective dialog box. In the C/C++
**Perspective**, click Yes to proceed. In the Project Explorer view, you should now see an entry for your hello_world project.

The next step is Adding a C Source File.

### Adding a C Source File

After Creating a New Project, you can add additional source files, then build and run your completed project.

If, however, you chose to create a project type of Empty project for the hello_world project, you would follow these steps to add a hello.c source file to the hello_world project.

1. Select the hello_world project in the Project Explorer view.
2. From the Eclipse* File menu, select New > Source File. Enter hello.c in the Source File text box of the New Source File dialog. Click Finish to add the file to the hello_world project.
3. In the Editor view, add your code for hello.c. When your code is complete, save your file using File > Save, then proceed to Building a Project.

### Building a Project

To build your project, make sure your hello_world project is selected in the Project Explorer view, then select Build all from the Eclipse* Project menu. See the Build results in the Console view.

```
**** Build of configuration Release for project hello_world ****
make -k all

Building file: ../main.c
Invoking: Compiler
icc -MMD -MP -MF"main.d" -MT"main.d" -c -o "main.o" "../main.c"
Finished building: ../main.c

Building target: hello_world
Invoking: Linker
icc -o"hello_world" ./main.o
Finished building target: hello_world
```
Running a Project

After **Building a Project**, you can run your project by following these steps:

1. Select the **hello_world** project in the **Project Explorer** view.
2. Select **Run > Run As > Local C/C++ Application**.
3. On the **Launch Debug Configuration Selection** dialog, select either the **Intel(R) Debugger** (if installed) or the GDB Debugger*, then click **OK**.
4. After the executable runs, the output of **hello.c** appears in the **Console** view.

Intel Error Parser

The Intel® C/C++ Error Parser (selected by default) lets you track compile-time errors in Eclipse*/CDT*. To confirm that the Intel® C/C++ Error Parser is active:

1. Select the **hello_world** project in the **Project Explorer** view.
2. Select **Project > Properties**.
3. In the **Properties** dialog box, select **C/C++ Build>Settings**.
4. Click the **Error Parsers** tab. The **Intel(R) C/C++ Error Parser** should already be selected. Do not check the **CDT Visual C Error Parser**.
5. Click **OK** to update your choices, if you have changed any settings, and close the dialog.

Using the Intel C/C++ Error Parser

If you introduce an error into your **hello.c** program, such as:

```c
#include <xstdio.h>
```

then compile **hello.c**, the error is reported in the **Problems** view and a ✗ marker appears in the source file at the line where the error was detected. This processing occurs automatically because the Intel C/C++ Error Parser detects and manages diagnostics generated by the Intel® C++ Compiler. You can
double-click on each error in the Problems view to visit the source line in the Editor view.

Correct the error, then rebuild your project.

Setting Properties

Setting Properties

The Intel® C++ Compiler integration with Eclipse*/CDT* lets you specify compiler, linker, and archiver options at the project and source file level.

Setting Options for Your Project

Follow these steps to set options for your project:

1. Select your project in the Project Explorer view.
2. From the Eclipse toolbar, select Project > Properties > C/C++ Build>Settings.
3. Under **Tool Settings**, click an option category for **C Compiler** or **Linker**.

![Tool Settings](image.png)

4. Set the option(s) you want to add to your project compilations, then open other categories if necessary. You can specify option settings independently on each configuration by changing the configuration selection in the **Configuration** drop-down box. If you are integrating with CDT5.0, you can set the same option across multiple configurations with one operation. Use the **Configuration** drop-down box to select this mode.

5. Click **OK** to complete your selections.

To reset properties to their default setting, click **Restore Defaults**. The **Restore Defaults** button appears on each property page, but the **Restore Defaults** action applies to ALL property pages.

**Setting Options for Source Files**

In addition to setting compiler options for your entire project, you can also specify options for any source file in your project.
1. Select a source file in your project.
2. Right-click on the source file, and select **Properties** from the context menu.
3. Select **C/C++ Build>Settings** to display the **Tool Settings** for changing compiler options.
4. Change options as described above.
5. In the **C/C++ Build** dialog box, you may choose **Exclude from build** if you do not want the source file included in the build.

![Active Resource configuration](image)

**Specifying Properties**

Some properties use check boxes, drop-down boxes, or other input mechanisms to specify a compiler option.

- Show Startup Banner (-V)
- Include Debug Information (-g)

**Optimization Level**
- Maximize Speed (-O2)

**Warning Level**
- Warnings and Errors (-w1)

Several options let you specify arguments. Click **New** to add an argument to the list. Enter a valid argument for the option, then click **OK**.
If you want to specify an option that is not available from the Properties dialog, use the Command Line category. Enter the command line options in the Additional Options text box just as you would enter them on the command line.

| Additional Options | -E -march=pentium4 |

See Also

Properties for Supported Options

Properties for Supported Options

The options listed in the following table are supported under the corresponding Option Category.

C and C++ Compiler Options

<table>
<thead>
<tr>
<th>Option Category</th>
<th>Use</th>
<th>Option Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Show startup banner</td>
<td>-V</td>
</tr>
<tr>
<td>General</td>
<td>Include debug information</td>
<td>-g</td>
</tr>
<tr>
<td>General</td>
<td>Optimization level</td>
<td>-O0</td>
</tr>
<tr>
<td>Optimization</td>
<td>(default for Debug Config.)</td>
<td>-O1</td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
<td>-O2</td>
</tr>
<tr>
<td>Optimization</td>
<td>(default for Release Config.)</td>
<td>-O3</td>
</tr>
<tr>
<td>Optimization</td>
<td>Warning level</td>
<td>-w</td>
</tr>
<tr>
<td>Optimization</td>
<td>Provide frame pointers</td>
<td>-fno-omit-frame-pointer</td>
</tr>
<tr>
<td>Optimization</td>
<td>Enable prefetch insertion optimization</td>
<td>-opt-prefetch[=n]</td>
</tr>
<tr>
<td>Option Category</td>
<td>Use</td>
<td>Option Name</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------</td>
<td>----------------------</td>
</tr>
<tr>
<td>Enable interprocedural</td>
<td>Enable interprocedural optimization for</td>
<td>(-ip)</td>
</tr>
<tr>
<td>optimization for single</td>
<td>single file compilation</td>
<td></td>
</tr>
<tr>
<td>Enable Whole Program</td>
<td>Enable Whole Program Optimization</td>
<td>(-ipo)</td>
</tr>
<tr>
<td>Optimization</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Disable intrinsic inline</td>
<td>Disable intrinsic inline expansion</td>
<td>(-nolib-inline)</td>
</tr>
<tr>
<td>expansion</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inline function expansion</td>
<td>Inline function expansion</td>
<td>(-inline-level)</td>
</tr>
<tr>
<td>Profile Guided Optimization</td>
<td>Profile Guided Optimization</td>
<td>(-prof-gen)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-prof-gen:srcpos)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-prof-use)</td>
</tr>
<tr>
<td>profile Directory</td>
<td>profile Directory</td>
<td>(-prof-dir)</td>
</tr>
<tr>
<td>Optimize for Intel</td>
<td>Optimize for Intel processor</td>
<td>(-mtune)</td>
</tr>
<tr>
<td>processor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop unroll count</td>
<td>Loop unroll count</td>
<td>(-unroll)</td>
</tr>
<tr>
<td>Parallelization</td>
<td>Parallelization</td>
<td>(-parallel)</td>
</tr>
<tr>
<td>Auto-parallelization</td>
<td>Auto-parallelization</td>
<td>(-par-threshold)</td>
</tr>
<tr>
<td>Threshold</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Precompiled</td>
<td>Precompiled</td>
<td>(-pch)</td>
</tr>
<tr>
<td></td>
<td>Automatic processing for precompiled</td>
<td></td>
</tr>
<tr>
<td></td>
<td>headers</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Precompiled headers file directory</td>
<td>(-pch-dir)</td>
</tr>
<tr>
<td>Preprocessor</td>
<td>gcc compatibility options</td>
<td>(-cxxlib)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-fabi-version)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(-gcc-version)</td>
</tr>
<tr>
<td>Option Category</td>
<td>Use</td>
<td>Option Name</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>Enable gcc predefined macros</td>
<td>-no-gcc</td>
<td></td>
</tr>
<tr>
<td>Non-standard gcc installation</td>
<td>-gcc-sys</td>
<td></td>
</tr>
<tr>
<td>Additional include directories</td>
<td>-I</td>
<td></td>
</tr>
<tr>
<td>Ignore standard include path</td>
<td>-X</td>
<td></td>
</tr>
<tr>
<td>Preprocessor definitions</td>
<td>-D</td>
<td></td>
</tr>
<tr>
<td>Undefined preprocessor</td>
<td>-U</td>
<td></td>
</tr>
<tr>
<td>definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Undefine all preprocessor</td>
<td>-A-</td>
<td></td>
</tr>
<tr>
<td>definitions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Enable use of ANSI aliasing rules in</td>
<td>-ansi-alias</td>
</tr>
<tr>
<td></td>
<td>optimizations</td>
<td></td>
</tr>
<tr>
<td>ANSI conformance</td>
<td>-ansi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-strict-ansi</td>
<td></td>
</tr>
<tr>
<td></td>
<td>-std</td>
<td></td>
</tr>
<tr>
<td>Recognize the restrict keyword</td>
<td>-restrict</td>
<td></td>
</tr>
<tr>
<td>Support multibyte characters</td>
<td>-multibyte-chars</td>
<td></td>
</tr>
<tr>
<td>in source</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Process OpenMP*</td>
<td>-openmp</td>
<td></td>
</tr>
<tr>
<td>directives</td>
<td>-openmp-stubs</td>
<td></td>
</tr>
<tr>
<td>Option Category</td>
<td>Use</td>
<td>Option Name</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>Compilation Diagnostics</td>
<td>Treat warnings as errors</td>
<td>-Werror</td>
</tr>
<tr>
<td></td>
<td>Allow usage messages</td>
<td>-Wcheck</td>
</tr>
<tr>
<td></td>
<td>&gt;Enable warnings for style guideline violations</td>
<td>-Weffc++</td>
</tr>
<tr>
<td></td>
<td>Set error limit</td>
<td>-diag-error-limit</td>
</tr>
<tr>
<td>Optimization Diagnostics</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Emit Optimization Diagnostics</td>
<td>-opt-report</td>
</tr>
<tr>
<td></td>
<td>Optimization Diagnostic Phase</td>
<td>-opt-report-phase</td>
</tr>
<tr>
<td></td>
<td>Optimization Diagnostic Routine</td>
<td>-opt-report-routine</td>
</tr>
<tr>
<td></td>
<td>OpenMP report</td>
<td>-openmp-report</td>
</tr>
<tr>
<td></td>
<td>Auto-parallelizer</td>
<td>-par-report</td>
</tr>
<tr>
<td></td>
<td>Vectorizer report</td>
<td>-vec-report</td>
</tr>
<tr>
<td>Disable Specific Diagnostics</td>
<td></td>
<td>-diag-disable</td>
</tr>
<tr>
<td></td>
<td>Emit Diagnostics To File</td>
<td>-diag-file=filespec</td>
</tr>
<tr>
<td></td>
<td>Level of Static Analysis</td>
<td>-diag-enable sv[n]</td>
</tr>
<tr>
<td></td>
<td>Analyze Include Files</td>
<td>-diag-enable sv-include</td>
</tr>
<tr>
<td>Data</td>
<td>Enable argument aliasing</td>
<td>-fargument-alias</td>
</tr>
<tr>
<td></td>
<td>Assume Restrict</td>
<td>-alias-const</td>
</tr>
<tr>
<td></td>
<td>Semantics for Const</td>
<td></td>
</tr>
<tr>
<td>Option Category</td>
<td>Use</td>
<td>Option Name</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td></td>
<td>Assume no aliasing in program</td>
<td><code>-fno-alias</code></td>
</tr>
<tr>
<td></td>
<td>Allow gprel addressing of common data variables</td>
<td><code>-fno-common</code></td>
</tr>
<tr>
<td></td>
<td>Allocate as many bytes as needed for enumerated types</td>
<td><code>-fshort-enums</code></td>
</tr>
<tr>
<td></td>
<td>Change default bitfield type to unsigned</td>
<td><code>-funsigned-bitfields</code></td>
</tr>
<tr>
<td></td>
<td>Change default char type to unsigned</td>
<td><code>-funsigned-char</code></td>
</tr>
<tr>
<td></td>
<td>Disable placement of zero-initialized variables in .bss - use .data</td>
<td><code>-no-bss-init</code></td>
</tr>
<tr>
<td></td>
<td>Default symbol visibility</td>
<td><code>-fvisibility</code></td>
</tr>
<tr>
<td></td>
<td>Structure member alignment</td>
<td><code>-Zp</code></td>
</tr>
<tr>
<td>Floating Point</td>
<td>Improve floating-point consistency</td>
<td><code>-mp</code></td>
</tr>
<tr>
<td></td>
<td>Round floating-point results</td>
<td><code>-fp-port</code></td>
</tr>
<tr>
<td></td>
<td>Limit Complex range</td>
<td><code>-complex-limited-range</code></td>
</tr>
<tr>
<td></td>
<td>Check floating-point stack</td>
<td><code>-fp-stack-check</code></td>
</tr>
<tr>
<td></td>
<td>Reliable floating-point exceptions model</td>
<td><code>-fp-model except</code></td>
</tr>
<tr>
<td>Option Category</td>
<td>Use</td>
<td>Option Name</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td></td>
<td>Floating-point speculation</td>
<td><code>-fp-speculation=safe</code></td>
</tr>
<tr>
<td></td>
<td>Floating-point model</td>
<td><code>-fp-model</code></td>
</tr>
<tr>
<td></td>
<td>Flush denormal results to zero</td>
<td><code>-ftz</code></td>
</tr>
<tr>
<td>Output Files</td>
<td>Generate assembler source file</td>
<td><code>-S</code></td>
</tr>
<tr>
<td></td>
<td>Generate position-independent code</td>
<td><code>-fpic</code></td>
</tr>
<tr>
<td>Code</td>
<td>Initialize local variables to NaN</td>
<td><code>-ftrapuv</code></td>
</tr>
<tr>
<td>Generation</td>
<td>Use Intel processor extensions</td>
<td><code>-ax</code></td>
</tr>
<tr>
<td></td>
<td>Require Intel processor extensions</td>
<td><code>-x</code></td>
</tr>
<tr>
<td>Runtime</td>
<td>Generate traceback information</td>
<td><code>-traceback</code></td>
</tr>
<tr>
<td></td>
<td>Check uninitialized variables</td>
<td><code>-check-uninit</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option Category</th>
<th>Use</th>
<th>Option Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libraries</td>
<td>Link with static libraries</td>
<td><code>-static</code></td>
</tr>
<tr>
<td></td>
<td>Link with dynamic libraries</td>
<td><code>-shared-intel</code></td>
</tr>
<tr>
<td></td>
<td>Use no C++ libraries</td>
<td><code>-no-cxxlib</code></td>
</tr>
<tr>
<td>Option Category</td>
<td>Use</td>
<td>Option Name</td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td></td>
<td>Use no system libraries</td>
<td><code>-nodefaultlibs</code></td>
</tr>
<tr>
<td></td>
<td>Additional libraries</td>
<td><code>-l</code></td>
</tr>
<tr>
<td></td>
<td>Search directory for libraries</td>
<td><code>-L</code></td>
</tr>
<tr>
<td>Optimization</td>
<td>Disable link-time IPO phase</td>
<td><code>no-ipo</code></td>
</tr>
</tbody>
</table>

**Makefiles**

**Project Types and Makefiles**

When you create a new Intel C project in Eclipse*/CDT*, you can select **Executable, Shared Library, or Static Library** projects or **Makefile** projects.
• Select Makefile Project if your project already includes a makefile.

• Use Executable, Shared Library, or Static Library Project to build a makefile using Intel compiler-specific options assigned from property pages.

Exporting Makefiles

If you created an Executable, Shared Library, or Static Library Project, you can use Eclipse* to build a makefile that includes Intel compiler options. See Setting Properties. When your project is complete, you can export your makefile and project source files to another directory, then build your project from the command line using make.

Exporting makefiles
To export your makefile:

1. Select your project in the Eclipse Project Explorer view.
2. From the Eclipse File menu, select Export to launch the Export Wizard.
3. On the Select dialog of the Export Wizard, select File system, then click Next.

![Export Wizard](image)

4. On the File system dialog, check both the helloworld and Release directories in the left-hand pane. Be sure all the project sources in the right-hand pane are also checked.

   **Note**

   You may deselect some files in the right-hand pane, such as the hello.o object file and helloworld executable. However, you must also select Create directory structure for files in the Options section to successfully create the export directory. This also applies to project files in the helloworld directory.

5. Use the Browse button to target the export to an existing directory. Eclipse can also create a new directory for full paths entered in the To
directory text box. If, for example, you specified /code/makefile as the export directory, Eclipse creates two new sub-directories:

- /code/makefile/helloworld
- /code/makefile/helloworld/Release

6. Click Finish to complete the export.

**Running make**

In a terminal window, change to the /cpp/hello_world/Release directory, then run make by typing:

```
make clean all
```

You should see the following output:
Building Applications from the Command Line

Invoking the Compiler from the Command Line

There are two necessary steps to invoke the Intel® C++ Compiler from the command line:

1. set the environment
2. invoke the compiler

Set the Environment Variables

Before you can operate the compiler, you must set the environment variables to specify locations for the various components. The Intel C++ Compiler installation includes shell scripts that you can "source" to set environment variables. With the default compiler installation, these scripts are:

<install-dir>/bin/iccvars.sh <arg>

or

<install-dir>/bin/iccvars.csh <arg>

The scripts take an argument specifying architecture:

- ia32: Compiler and libraries for IA-32 architectures only
- intel64: Compiler and libraries for Intel® 64 architectures only
- ia64: Compiler and libraries for IA-64 architectures only

To source an environment script, enter one of the following on the command line:

source <install-dir>/bin/iccvars.sh <arg>
or

```
source <install-dir>/bin/iccvars.csh <arg>
```

If you want the script to run automatically, add the same command to the end of your startup file.

**Sample .bash_profile entry for iccvars.sh:**

```
# set environment vars for Intel C++ compiler
source <install-dir>/bin/iccvars.sh ia32
```

With some Linux* distributions, if you source `iccvars.sh` from your `.bash_profile`, the location of `LIBRARY_PATH` may not be set as you would expect. It may be necessary to source `iccvars.sh` after starting your terminal session. This affects the Intel C++ compiler (icpc) only.

**Invoking the Compiler with icc or icpc**

You can invoke the Intel C++ Compiler on the command line with either `icc` or `icpc`.

- When you invoke the compiler with `icc`, the compiler builds C source files using C libraries and C include files. If you use `icc` with a C++ source file, it is compiled as a C++ file. Use `icc` to link C object files.
- When you invoke the compiler with `icpc` the compiler builds C++ source files using C++ libraries and C++ include files. If you use `icpc` with a C source file, it is compiled as a C++ file. Use `icpc` to link C++ object files.

**Command-line Syntax**

When you invoke the Intel C++ Compiler with `icc` or `icpc`, use the following syntax:

```
{icc|icpc} [options] file1 [file2 ...]
```

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>options</td>
<td>Indicates one or more command-line options. The compiler recognizes one or more letters preceded by a hyphen (-). This includes linker options.</td>
</tr>
<tr>
<td>Argument</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>file1,</td>
<td>Indicates one or more files to be processed by the compiler. You can specify more than one file. Use a space as a delimiter for multiple files.</td>
</tr>
<tr>
<td>file2 . . .</td>
<td></td>
</tr>
</tbody>
</table>

**Invoking the Compiler from the Command Line with make**

To run `make` from the command line using Intel® C++ Compiler, make sure that `/usr/bin` is in your path.

To use the Intel compiler, your makefile must include the setting `CC=icpc`. Use the same setting on the command line to instruct the makefile to use the Intel compiler. If your makefile is written for gcc, the GNU* C compiler, you will need to change those command line options not recognized by the Intel compiler. Then you can compile:

```
make -f my_makefile
```

**See Also**

[Modifying Your makefile](#)

**Passing Options to the Linker**

This topic describes the options that let you control and customize the linking with tools and libraries and define the output of the `ld` linker. See the `ld` man page for more information on the linker.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-L</code> directory</td>
<td>Instruct the linker to search <code>directory</code> for libraries.</td>
</tr>
<tr>
<td><code>-Qoption,tool,list</code></td>
<td>Passes an argument list to another program in the compilation sequence, such as the assembler or linker.</td>
</tr>
<tr>
<td><code>-shared</code></td>
<td>Instructs the compiler to build a Dynamic Shared Object (DSO) instead of an executable.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-shared-libgcc</code></td>
<td><code>-shared-libgcc</code> has the opposite effect of <code>-static-libgcc</code>. When it is used, the GNU standard libraries are linked in dynamically, allowing the user to override the static linking behavior when the <code>-static</code> option is used. Note: By default, all C++ standard and support libraries are linked dynamically.</td>
</tr>
<tr>
<td><code>-shared-intel</code></td>
<td>Specifies that all Intel-provided libraries should be linked dynamically.</td>
</tr>
<tr>
<td><code>-static</code></td>
<td>Causes the executable to link all libraries statically, as opposed to dynamically.</td>
</tr>
<tr>
<td></td>
<td>When <code>-static</code> is not used:</td>
</tr>
<tr>
<td></td>
<td>• <code>/lib/ld-linux.so.2</code> is linked in</td>
</tr>
<tr>
<td></td>
<td>• all other libs are linked dynamically</td>
</tr>
<tr>
<td></td>
<td>When <code>-static</code> is used:</td>
</tr>
<tr>
<td></td>
<td>• <code>/lib/ld-linux.so.2</code> is not linked in</td>
</tr>
<tr>
<td></td>
<td>• all other libs are linked statically</td>
</tr>
<tr>
<td><code>-static-libgcc</code></td>
<td>This option causes the GNU standard libraries to be linked in statically.</td>
</tr>
<tr>
<td><code>-Bstatic</code></td>
<td>This option is placed in the linker command line corresponding to its location on the user command line. This option is used to control the linking behavior of any library being passed in via the command line.</td>
</tr>
<tr>
<td><code>-Bdynamic</code></td>
<td>This option is placed in the linker command line corresponding to its location on the user command line. This option is used to control the linking behavior of any library being passed in via the command line.</td>
</tr>
<tr>
<td><code>-static-intel</code></td>
<td>This option causes Intel-provided libraries to be linked in</td>
</tr>
</tbody>
</table>
### Compiler Input Files

The Intel® C++ Compiler recognizes input files with the extensions listed in the following table:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Interpretation</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.c</td>
<td>C source file</td>
<td>Passed to compiler</td>
</tr>
<tr>
<td>file.C</td>
<td>C++ source file</td>
<td>Passed to compiler</td>
</tr>
<tr>
<td>file.CC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cpp</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.cxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.a</td>
<td>Library file</td>
<td>Passed to linker</td>
</tr>
<tr>
<td>file.so</td>
<td></td>
<td></td>
</tr>
<tr>
<td>file.i</td>
<td>Preprocessed file</td>
<td>Passed to stdout</td>
</tr>
<tr>
<td>file.o</td>
<td>Object file</td>
<td>Passed to linker</td>
</tr>
<tr>
<td>file.s</td>
<td>Assembly file</td>
<td>Passed to assembler</td>
</tr>
</tbody>
</table>

### Output Files

The Intel® C++ Compiler produces output files with the extensions listed in the following table:

- statically. It is the opposite of `-shared-intel`.

- `-Wl, optlist` This option passes a comma-separated list (`optlist`) of linker options to the linker.

- `-Xlinker val` This option passes a value (`val`), such as a linker option, an object, or a library, directly to the linker.
### File Name

<table>
<thead>
<tr>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file.i</td>
<td>Preprocessed file -- produced with the (-P) option.</td>
</tr>
<tr>
<td>file.o</td>
<td>Object file -- produced with the (-c) option.</td>
</tr>
<tr>
<td>file.s</td>
<td>Assembly language file -- produced with the (-S) option.</td>
</tr>
<tr>
<td>a.out</td>
<td>Executable file -- produced by default compilation.</td>
</tr>
</tbody>
</table>

**See Also**

- [Using Options for Preprocessing](#)
- [Specifying Object Files](#)
- [Specifying Assembly Files](#)
- [Specifying Executable Files](#)

### Specifying Compilation Output

#### Specifying Executable Files

You can use the \(-o\) option to specify the name of the executable file. In the following example, the compiler produces an executable file named `startup`.

```
icpc -ostartup prog1.cpp
```

**See Also**

- \(-o\) compiler option

#### Specifying Object Files

You can use the \(-c\) and \(-o\) options to specify an alternate name for an object file. In this example, the compiler generates an object file name `myobj.o`:

```
icpc -c -omyobj.o x.cpp
```

**See Also**

- \(-c\) compiler option
- \(-o\) compiler option

#### Specifying Assembly Files
You can use the -S and -o options to specify an alternate name for an assembly file. In this example, the compiler generates an assembly file named `myasm.s`:

```
icpc -S -omyasm.s x.cpp
```

**See Also**

- `-S` compiler option

**Specifying Alternate Tools and Paths**

Use the `-Qlocation` option to specify an alternate path for a tool. This option accepts two arguments using the following syntax:

```
-Qlocation,tool,path
```

where `tool` designates which compilation tool is associated with the alternate `path`.

<table>
<thead>
<tr>
<th>tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpp</td>
<td>Specifies the compiler front-end preprocessor.</td>
</tr>
<tr>
<td>c</td>
<td>Specifies the C++ compiler.</td>
</tr>
<tr>
<td>asm</td>
<td>Specifies the assembler.</td>
</tr>
<tr>
<td>link</td>
<td>Specifies the linker.</td>
</tr>
</tbody>
</table>

Use the `-Qoption` option to pass an option specified by `optlist` to a `tool`, where `optlist` is a comma-separated list of options. The syntax for this command is:

```
-Qoption,tool,optlist
```

where `tool` designates which compilation tool receives the `optlist`.

<table>
<thead>
<tr>
<th>tool</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cpp</td>
<td>Specifies the compiler front-end preprocessor.</td>
</tr>
<tr>
<td>c</td>
<td>Specifies the C++ compiler.</td>
</tr>
<tr>
<td>asm</td>
<td>Specifies the assembler.</td>
</tr>
<tr>
<td>link</td>
<td>Specifies the linker.</td>
</tr>
</tbody>
</table>
optlist indicates one or more valid argument strings for the designated program. If the argument is a command-line option, you must include the hyphen. If the argument contains a space or tab character, you must enclose the entire argument in quotation characters ("""). You must separate multiple arguments with commas.

Building Applications Using Intel Performance Headers

The Intel compiler provides a high performance implementation of specialized valarray operations for the C++ standard valarray container. The standard C++ valarray template consists of array or vector operations for high performance computing; these operations are designed to exploit high performance hardware features such as parallelism and achieve performance benefits. The Intel implementation of valarray, which requires installation of Intel® Integrated Performance Primitives (IPP), consists of a replacement header that provides a specialized high performance implementation for the following operators and types:

<table>
<thead>
<tr>
<th>operator</th>
<th>valarrays of Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>abs, acos, asin, atan, atan2, cos, cosh, exp, log, log10, pow, sin, sinh, sqrt, tan, tanh</td>
<td>float, double</td>
</tr>
<tr>
<td>addition, subtraction, division, multiplication</td>
<td>float, double</td>
</tr>
<tr>
<td>bitwise or, and, xor</td>
<td>(all unsigned) char, short, int</td>
</tr>
<tr>
<td>min, max, sum</td>
<td>signed or short/signed int, float, double</td>
</tr>
</tbody>
</table>

The following examples illustrate two instances of how to compile and link a program to include the Intel valarray replacement header file and link with IPP. Refer to the IPP documentation for details.

In the following examples:

- compiler_dir is the directory where the Intel C++ compiler is installed
• *ipp_include_dir* is the include directory for IPP
• *ipp_lib_dir* is the directory containing IPP libraries
• "merged" libraries means using a static library that contains all the cpu-specific variants of the library code

**Windows OS examples:**

The following command lines perform separate compile and link steps for a system based on IA-32 architecture, running Windows OS:

**DLL (dynamic):**

icl -Icompiler_dir/perf_headers/c++ -Iipp_include_dir -c source.cpp
icl source.obj /link /libpath:ipp_lib_dir ippcore.lib ipps.lib ippvm.lib

**Merged (static):**

icl -Icompiler_dir/perf_headers/c++ -Iipp_include_dir -c source.cpp
icl source.obj /link /libpath:ipp_lib_dir ippcorel.lib ippsmerged.lib ippvmmerged.lib

**Linux OS examples:**

The following command lines perform separate compile and link steps for a system based on Intel® 64 architecture, running Linux OS:

**so (dynamic):**

icpc -Icompiler_dir/perf_headers/c++ -Iipp_include_dir -c source.cpp
icpc source.o -Lipp_lib_dir -lippsem64t -lippvmem64t -lippcoreem64t

**Merged (static):**

icpc -Icompiler_dir/perf_headers/c++ -Iipp_include_dir -c source.cpp
icpc source.o -Lipp_lib_dir -lippsemmerged64t -lippsmergedem64t -lippvmmergedem64t -lippcoreem64t

**Note**

To use the static merged library, containing all cpu-specific optimized versions of the library code, you need to call the ippStaticInit function first, before any IPP calls. This ensures automatic dispatch to the correct version of the library code at runtime. If you don't call ippStaticInit first, the emerged library will use the generic instance of the code.

If you are using the dynamic version of the libraries, you do not need to call ippStaticInit.
Using Precompiled Header Files

The Intel® C++ Compiler supports precompiled header (PCH) files to significantly reduce compile times using the following options:

- `-pch`
- `-pch-dir dirname`
- `-pch-create filename`
- `-pch-use filename`

⚠️ Caution

Depending on how you organize the header files listed in your sources, these options may increase compile times. See Organizing Source Files to learn how to optimize compile times using the PCH options.

Using `-pch`

The `-pch` option directs the compiler to use appropriate PCH files. If none are available, they are created as `sourcefile.pchi`. This option supports multiple source files, such as the ones shown in Example 1:

Example 1 command line:
```
icpc -pch source1.cpp source2.cpp```

Example 1 output when `.pchi` files do not exist:

"source1.cpp": creating precompiled header file
"source1.pchi"
"source2.cpp": creating precompiled header file
"source2.pchi"

Example 1 output when `.pchi` files do exist:

"source1.cpp": using precompiled header file "source1.pchi"
"source2.cpp": using precompiled header file "source2.pchi"
The -pch option will use PCH files created from other sources if the headers files are the same. For example, if you compile `source1.cpp` using -pch, then `source1.pchi` is created. If you then compile `source2.cpp` using -pch, the compiler will use `source1.pchi` if it detects the same headers.

Using -pch-create

Use the -pch-create `filename` option if you want the compiler to create a PCH file called `filename`. Note the following regarding this option:

- The `filename` parameter must be specified.
- The `filename` parameter can be a full path name.
- The full path to `filename` must exist.
- The `.pchi` extension is not automatically appended to `filename`.
- This option cannot be used in the same compilation as -pch-use `filename`.
- The -pch-create `filename` option is supported for single source file compilations only.

Example 2 command line:
```
icpc -pch-create /pch/source32.pchi source.cpp
```
Example 2 output:
"source.cpp": creating precompiled header file
"/pch/source32.pchi"

Using -pch-use `filename`

This option directs the compiler to use the PCH file specified by `filename`. It cannot be used in the same compilation as -pch-create `filename`. The -pch-use `filename` option supports full path names and supports multiple source files when all source files use the same `.pchi` file.

Example 3 command line:
```
icpc -pch-use /pch/source32.pchi source.cpp
```
Example 3 output:
"source.cpp": using precompiled header file
/pch/source32.pchi

Using -pch-dir dirname

Use the -pch-dir dirname option to specify the path (dirname) to the PCH file.
You can use this option with -pch, -pch-create filename, and -pch-use filename.
Example 4 command line:
icpc -pch -pch-dir /pch/source32.cpp
Example 4 output:
"source32.cpp": creating precompiled header file
/pch/source32.pchi

Organizing Source Files

If many of your source files include a common set of header files, place the
common headers first, followed by the #pragma hdrstop directive. This
pragma instructs the compiler to stop generating PCH files. For example, if
source1.cpp, source2.cpp, and source3.cpp all include common.h, then
place #pragma hdrstop after common.h to optimize compile times.
#include "common.h"
#pragma hdrstop
#include "noncommon.h"

When you compile using the -pch option:
icpc -pch source1.cpp source2.cpp source3.cpp
the compiler will generate one PCH file for all three source files:
"source1.cpp": creating precompiled header file
"source1.pchi"
"source2.cpp": using precompiled header file "source1.pchi"
"source3.cpp": using precompiled header file "source1.pchi"
If you don't use #pragma hdrstop, a different PCH file is created for each
source file if different headers follow common.h, and the subsequent compile
times will be longer. `#pragma hdrstop` has no effect on compilations that do not use these PCH options.

See Also

- `pch` compiler option
- `pch-dir` compiler option
- `pch-create` compiler option
- `pch-use` compiler option

**Compiler Option Mapping Tool**

The Intel compiler's Option Mapping Tool provides an easy method to derive equivalent options between Windows* and Linux*. If you are a Windows-based application developer who is developing an application for Linux OS, you may want to know, for example, the Linux OS equivalent for the `/Oy-` option. Likewise, the Option Mapping Tool provides Windows OS equivalents for Intel compiler options supported on Linux OS. The Option Mapping Tool is not supported on Mac OS* X.

**Using the Compiler Option Mapping Tool**

You can start the Option Mapping Tool from the command line by:

- invoking the compiler and using the `-map-opts` option
- or, executing the tool directly

**Note**

Compiler options are mapped to their equivalent on the architecture you are using. It will not, for example, map an option that is specific to the IA-64 architecture to a like option available on the IA-32 architecture or Intel® 64 architecture.

**Calling the Option Mapping Tool with the Compiler**

If you use the compiler to execute the Option Mapping Tool, the following syntax applies:

```bash
<compiler command> <map-opts option> <compiler option(s)>
```
**Example:** Finding the Windows OS equivalent for `-fp`

```
icpc -map-opts -fp
```

**Example:** Finding the Windows OS equivalent for `-fp`

```
icpc -map-opts -fp
```

Intel(R) Compiler option mapping tool

mapping Linux options to Windows OS for C++

'`-map-opts' Linux option maps to
    --> '-Qmap-opts' option on Windows
    --> '-Qmap_opts' option on Windows

'`-fp' Linux option maps to
    --> '-Oy-' option on Windows

Output from the Option Mapping Tool also includes:

- option mapping information (not shown here) for options included in the compiler configuration file
- alternate forms of the option that are supported but may not be documented

When you call the Option Mapping Tool with the compiler, your source file is not compiled.

**Calling the Option Mapping Tool Directly**

Use the following syntax to execute the Option Mapping Tool directly from a command line environment where the full path to the `map_opts` executable is known (compiler `bin` directory):

```
map_opts [-nologo] -t<target OS> -l<language> -opts <compiler option(s)>
```

where values for:

- `<target OS> = {l|linux|w|windows}`
- `<language> = {f|fortran|c}`

**Example:** Finding the Windows OS equivalent for `-fp`

```
map_opts -tw -lc -opts -fp
```

Intel(R) Compiler option mapping tool

mapping Linux options to Windows for C++

'`-fp' Linux option maps to
    --> '-Oy-' option on Windows

**Open Source Tools**

This version of the Intel® C++ Compiler includes improved support for the following open source tools:
Intel® C++ Compiler User and Reference Guides

- **GNU Libtool** – a script that allows package developers to provide generic shared library support.
- **Valgrind** – a flexible system for debugging and profiling executables running on x86 processors.
- **GNU Automake** – a tool for automatically generating `Makefile.ins` from files called `Makefile.am`.

**See Also**

GNU Automake documentation – http://sources.redhat.com/automake/automake.html

**Using Preprocessor Options**

**About Preprocessor Options**

This section explains how preprocessor options are implemented in the Intel® C++ Compiler to perform preliminary operations on C and C++ source files. The preprocessor options are summarized in the following table:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-E</td>
<td>preprocess to stdout. #line directives included.</td>
</tr>
<tr>
<td>-P</td>
<td>preprocess to a file. #line directives omitted.</td>
</tr>
<tr>
<td>-EP</td>
<td>preprocess to stdout omitting #line directives.</td>
</tr>
<tr>
<td>-C</td>
<td>retain comments in intermediate file (use with -E or -P).</td>
</tr>
<tr>
<td>-D</td>
<td>define a macro.</td>
</tr>
<tr>
<td>-U</td>
<td>undefine a macro.</td>
</tr>
<tr>
<td>-I</td>
<td>add directory to include file search path.</td>
</tr>
<tr>
<td>-X</td>
<td>remove standard directories from include file search path.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>-H</td>
<td>print include file order.</td>
</tr>
<tr>
<td>-M</td>
<td>generate makefile dependency information.</td>
</tr>
</tbody>
</table>

See Also

- E compiler option
- P compiler option
- EP compiler option
- C compiler option
- D compiler option
- U compiler option
- I compiler option
- X compiler option
- H compiler option
- M compiler option

Using Options for Preprocessing

Use these options to preprocess your source files without compiling them. When using these options, only the preprocessing phase of compilation is activated.

Using -E

Use this option to preprocess to stdout. For example, to preprocess two source files and write them to stdout, enter the following command:

```
icpc -E prog1.cpp prog2.cpp
```

Using -P

Use this option to preprocess to a .i file omitting #line directives. For example, the following command creates two files named progl.i and progl2.i, which you can use as input to another compilation:

```
icpc -P prog1.cpp prog2.cpp
```
Existing files with the same name and extension are overwritten when you use this option.

Using -EP

Use this option to preprocess to stdout omitting #line directives.
```
icpc -EP prog1.cpp prog2.cpp
```

Using -C

Use this option to retain comments. In this example:
```
icpc -C -P prog1.cpp prog2.cpp
```
the compiler preserves comments in the prog1.i preprocessed file.

Option Summary

The following table summarizes the preprocessing options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Output Includes</th>
<th>Output #line Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td>-E</td>
<td>Yes</td>
<td>stdout</td>
</tr>
<tr>
<td>-P</td>
<td>No</td>
<td>.i file</td>
</tr>
<tr>
<td>-EP</td>
<td>No</td>
<td>stdout</td>
</tr>
<tr>
<td>-P -EP</td>
<td>No</td>
<td>.i file</td>
</tr>
</tbody>
</table>

See Also

-EP compiler option
-P compiler option
-EP compiler option
-C compiler option

Using Options to Define Macros

You can use compiler options to define or undefine predefined macros.

Using -D
Use this option to define a macro. For example, to define a macro called `SIZE` with the value 100 use the following command:

```
icpc -DSIZE=100 prog1.cpp
```

If you define a macro, but do not assign a value, the compiler defaults to 1 for the value of the macro.

**Using -U**

Use this option to undefine a macro. For example, this command:

```
icpc -Uia32 prog1.cpp
```

undefines the `ia32` predefined macro. If you attempt to undefine an ANSI C macro, the compiler will emit an error:

```
invalid macro undefinition: <name of macro>
```

**See Also**

- ANSI Standard Predefined Macros
- Additional Predefined Macros
- Modifying the Compilation Environment

**About Modifying the Compilation Environment**

To run the Intel® C++ Compiler, you need to start with the proper environment. The compiler includes scripts which set the environment for all necessary components. You can modify the compilation environment by specifying different settings for:

- Environment Variables
- Configuration Files
- Include Files

**See Also**

- Response Files

**Setting Environment Variables**

You can customize your system environment by specifying paths where the compiler searches for special files such as libraries, include files, and
configuration files. The Intel® C++ Compiler supports the environment variables listed in the following table:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GXX_INCLUDE</td>
<td>Specifies the location of the gcc headers. Set this variable only when the compiler cannot locate the gcc headers when using the <code>-gcc-name</code> option.</td>
</tr>
<tr>
<td>GXX_ROOT</td>
<td>Specifies the location of the gcc binaries. Set this variable only when the compiler cannot locate the gcc binaries when using the <code>-gcc-name</code> option.</td>
</tr>
<tr>
<td>IA32ROOT (IA-32 architecture and Intel® 64 architecture)</td>
<td>Points to the directories containing the include and library files for a non-standard installation structure.</td>
</tr>
<tr>
<td>IA64ROOT (IA-64 architecture based systems)</td>
<td>Points to the directories containing the include and library files for a non-standard installation structure.</td>
</tr>
<tr>
<td>ICCCFG</td>
<td>Specifies the configuration file for customizing compilations when invoking the compiler using <code>icc</code>.</td>
</tr>
<tr>
<td>ICPCCFG</td>
<td>Specifies the configuration file for customizing compilations when invoking the compiler using <code>icpc</code>.</td>
</tr>
<tr>
<td>INTEL_LICENSE_FILE</td>
<td>Specifies the location for the Intel license file.</td>
</tr>
<tr>
<td>KMP_ALL_THREADS</td>
<td>Limits the number of simultaneously executing threads in an OpenMP* program. If this limit is reached and another native operating system thread encounters OpenMP* API calls or constructs, then the program may abort with an error message. If this limit is reached at the time an OpenMP parallel region begins, a one-time</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>warning message may be generated indicating that the number of threads in the team was reduced, but the program will continue execution. This environment variable is only used for programs compiled with <code>-openmp</code> or <code>-openmp-profile</code>.</td>
<td></td>
</tr>
<tr>
<td>KMP_LIBRARY</td>
<td>Selects the OpenMP run-time library execution mode. The values for this variable are <code>serial</code>, <code>turnaround</code>, or <code>throughput</code> (default).</td>
</tr>
<tr>
<td>KMP_STACKSIZE</td>
<td>Sets the number of bytes to allocate for each OpenMP* thread to use as its private stack. Use the optional suffix b, k, m, g, or t, to specify bytes, kilobytes, megabytes, gigabytes, or terabytes. Note that this variable does not have any effect on native operating system threads created by the user program or the thread executing the sequential part of an OpenMP* program. Default: IA-32 architecture: 2m, Intel® 64 architecture: 4m, IA-64 architecture: 4m.</td>
</tr>
<tr>
<td>KMP_VERSION</td>
<td>Enables (1) or disables (0) the printing of OpenMP run-time library version information during program execution. Default: disabled.</td>
</tr>
<tr>
<td>LD_LIBRARY_PATH</td>
<td>Specifies the location for shared objects.</td>
</tr>
<tr>
<td>OMP_DYNAMIC</td>
<td>Enables (1) or disables (0) the dynamic adjustment of the number of threads. Default is 0 (disabled).</td>
</tr>
<tr>
<td>OMP_NESTED</td>
<td>Enables (1) or disables (0) nested parallelism. Default is 0 (nested parallelism disabled).</td>
</tr>
<tr>
<td>OMP_NUM_THREADS</td>
<td>Sets the maximum number of threads to use for</td>
</tr>
</tbody>
</table>
OpenMP* parallel regions if no other value is specified in the program itself. Default is the number of processors currently visible to the operating system on which the program is executed.

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PATH</td>
<td>Specifies the directories the system searches for binary executable files.</td>
</tr>
<tr>
<td>TMP</td>
<td>Specifies the location for temporary files. If none of these are specified, the compiler stores temporary files in /tmp.</td>
</tr>
</tbody>
</table>

**GNU* Environment Variables**

The Intel C++ Compiler also supports the GNU environment variables listed in the following table:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPATH</td>
<td>Path to include directory for C/C++ compilations.</td>
</tr>
<tr>
<td>C_INCLUDE_PATH</td>
<td>Path include directory for C compilations.</td>
</tr>
<tr>
<td>CPLUS_INCLUDE_PATH</td>
<td>Path include directory for C++ compilations.</td>
</tr>
<tr>
<td>DEPENDENCIES_OUTPUT</td>
<td>If this variable is set, its value specifies how to output dependencies for make based on the non-system header files processed by the compiler. System header files are ignored in the dependency output.</td>
</tr>
<tr>
<td>GCC_EXEC_PREFIX</td>
<td>This variable specifies alternative names for the linker (ld) and assembler (as).</td>
</tr>
<tr>
<td>LIBRARY_PATH</td>
<td>The value of LIBRARY_PATH is a colon-separated list</td>
</tr>
</tbody>
</table>
Environment Variable | Description
--- | ---
SUNPRO_DEPENDENCIES | This variable is the same as DEPENDENCIES_OUTPUT, except that system header files are not ignored.

Using Configuration Files

You can decrease the time you spend entering command-line options by using the configuration file to automate command-line entries. Add any valid command-line option to the configuration file. The compiler processes options in the configuration file in the order they appear followed by the command-line options that you specify when you invoke the compiler. Options in the configuration file are executed every time you run the compiler. If you have varying option requirements for different projects, use response files.

How to Use Configuration Files

The following example illustrates a basic configuration file. The text following the "#" character is recognized as a comment. The configuration file, icc.cfg and icpc.cfg, is located in the same directory as the compiler's executable file. You should modify the configuration file environment variable if you need to specify a different location for the configuration file.

```
# Sample configuration file.
-I/my_headers
```

In this example, the compiler reads the configuration file and invokes the -I option every time you run the compiler, along with any options you specify on the command line.

See Also

Environment Variables
Using Response Files
Specifying Include Files

The Intel® C++ Compiler searches the default system areas for include files and whatever is specified by the \(-I\) compiler option. The compiler searches directories for include files in the following order:

1. Directories specified by the \(-I\) option
2. Directories specified in the environment variables
3. Default include directory

Use the \(-x\) option to remove default directories from the include file search path.

For example, to direct the compiler to search the path \(/alt/include\) instead of the default path, do the following:

```
icpc -X -I/alt/include prog1.cpp
```

See Also

\(-I\) compiler option
\(-x\) compiler option
Environment Variables

Using Response Files

Use response files to specify options used during particular compilations. Response files are invoked as an option on the command line. Options in a response file are inserted in the command line at the point where the response file is invoked.

Sample Response Files

```
# response file: response1.txt
# compile with these options
-w0
# end of response1 file
# response file: response2.txt
# compile with these options
-00
# end of response2 file
```
Use response files to decrease the time spent entering command-line options and to ensure consistency by automating command-line entries. Use individual response files to maintain options for specific projects.

Any number of options or file names can be placed on a line in a response file. Several response files can be referenced in the same command line. The following example shows how to specify a response file on the command line:

```bash
icpc @response1.txt prog1.cpp @response2.txt prog2.cpp
```

**Note**

An "@" symbol must precede the name of the response file on the command line.

**See Also**

*Using Configuration Files*

*Debugging*

**Using the Debugger**

See the Intel® Debugger (IDB) documentation for complete information on using the Intel Debugger.

You can also use the GNU Debugger (gdb) to debug programs compiled with the Intel® C++ Compiler.

**Preparing for Debugging**

Use this option to direct the compiler to generate code to support symbolic debugging. For example:

```bash
icpc -g prog1.cpp
```

The compiler does not support the generation of debugging information in assembly files. When you specify this option, the resulting object file will contain debugging information, but the assembly file will not. This option changes the default optimization from -O2 to -O0.

**See Also**
Symbolic Debugging and Optimizations

When you use debugging options to generate code with debug symbols, the compiler disables $O_n$ optimizations. If you specify an $O_n$ option with debugging options, some of the generated debug information may be inaccurate as a side-effect of optimization.

The table below summarizes the effects of using the debugging options with the optimization options.

<table>
<thead>
<tr>
<th>Option(s)</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-g</td>
<td>Debugging information produced, -00 and -fp enabled.</td>
</tr>
<tr>
<td>-g -01</td>
<td>Debugging information produced, -01 optimizations enabled.</td>
</tr>
<tr>
<td>-g -02</td>
<td>Debugging information produced, -02 optimizations enabled.</td>
</tr>
<tr>
<td>-g -03 -fp</td>
<td>Debugging information produced, -03 optimizations and -fp are enabled.</td>
</tr>
</tbody>
</table>

See Also

- -g compiler option
- -fp compiler option
- -01, -02, -03 compiler options

Using Options for Debug Information

The Intel® C++ Compiler provides basic debugging information and new features for enhanced debugging of code. The basic debugging options are listed in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-debug all</td>
<td>These options are equivalent to -g. They turn on</td>
</tr>
<tr>
<td>-debug full</td>
<td>production of basic debug information. They are off by</td>
</tr>
</tbody>
</table>
The Intel C++ Compiler improves debuggability of optimized code through enhanced support for:

- **tracebacks**
- **variable locations**
- **breakpoints and stepping**

The options described in the following table control generation of enhanced debug information.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>default.</td>
<td>This option turns off production of debug information. This option is on by default.</td>
</tr>
<tr>
<td>-debug none</td>
<td>This option is on by default.</td>
</tr>
<tr>
<td>-debug [no]expr-source-pos</td>
<td>This option controls whether source position information at the statement level of granularity is generated.</td>
</tr>
<tr>
<td>-debug inline-debug-info</td>
<td>This option produces enhanced debug information for inlined code. It provides more information to debuggers for function call traceback.</td>
</tr>
<tr>
<td>-debug semantic-stepping</td>
<td>This option generates information useful for breakpoints and stepping. It tells the debugger to stop only at machine instructions that achieve the final effect of a source statement.</td>
</tr>
<tr>
<td>-debug variable-locations</td>
<td>This option produces additional debug information for scalar local variables using a feature of the DWARF object module format known as &quot;location lists.&quot; The runtime locations of local scalar variables are specified more accurately using this feature, i.e. whether at a given position in the code, a variable value is found in memory or a machine register.</td>
</tr>
</tbody>
</table>
Option | Description
---|---
-`debug extended` | This option turns on the following `-debug` options:
- `-debug semantic-stepping`
- `-debug variable-locations`
It also specifies that column numbers should appear in the line information.

**Note**

When the compiler needs to choose between optimization and quality of debug information, optimization is given priority.

**Creating and Using Libraries**

**Overview: Using Libraries**

The Intel® C++ Compiler uses the GNU* C Library and the Standard C++ Library. These libraries are documented at the following URLs:


**Default Libraries**

The following libraries are supplied with the Intel® C++ Compiler:

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>libguide.a</code></td>
<td>For OpenMP* implementation</td>
</tr>
<tr>
<td><code>libguide.so</code></td>
<td></td>
</tr>
<tr>
<td><code>libguide_stats.a</code></td>
<td>OpenMP static library for the parallelizer tool with performance statistics and profile information</td>
</tr>
<tr>
<td><code>libguide_stats.so</code></td>
<td></td>
</tr>
<tr>
<td><code>libompstub.a</code></td>
<td>Library that resolves references to OpenMP subroutines when OpenMP is not in use</td>
</tr>
<tr>
<td>Library</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>libsvml.a</td>
<td>Short vector math library</td>
</tr>
<tr>
<td>libsvml.so</td>
<td></td>
</tr>
<tr>
<td>libirc.a</td>
<td>Intel support library for PGO and CPU dispatch</td>
</tr>
<tr>
<td>libirc_s.a</td>
<td></td>
</tr>
<tr>
<td>libintlc.so.5</td>
<td>Intel support library for PGO and CPU dispatch</td>
</tr>
<tr>
<td>libimf.a</td>
<td>Intel math library</td>
</tr>
<tr>
<td>libimf.so</td>
<td></td>
</tr>
<tr>
<td>libimf.so</td>
<td></td>
</tr>
<tr>
<td>libcxaguard.a</td>
<td>Used for interoperability support with the -cxxlib option.</td>
</tr>
<tr>
<td>libcxaguard.so</td>
<td></td>
</tr>
<tr>
<td>libcxaguard.so.5</td>
<td>See gcc Interoperability.</td>
</tr>
</tbody>
</table>

When you invoke the -cxxlib option, libcprts is replaced with libstdc++ from the gcc* distribution (3.2 or newer)

⚠️ **Caution**

The Linux* OS system libraries and the compiler libraries are not built with the -align option. Therefore, if you compile with the -align option and make a call to a compiler distributed or system library, and have long long, double, or long double types in your interface, you will get the wrong answer due to the difference in alignment. Any code built with -align cannot make calls to libraries that use these types in their interfaces unless they are built with -align (in which case they will not work without -align).

**Math Libraries**

The Intel math library, libimf.a, contains optimized versions of math functions found in the standard C run-time library. The functions in libimf.a are optimized for program execution speed on Intel processors. The Intel math library is linked by default.
Managing Libraries

During compilation, the compiler reads the `LIBRARY_PATH` environment variable for static libraries it needs to link when building the executable. At runtime, the executable will link against dynamic libraries referenced in the `LD_LIBRARY_PATH` environment variable.

Modifying `LIBRARY_PATH`

If you want to add a directory, `/libs` for example, to the `LIBRARY_PATH`, you can do either of the following:

- command line: `prompt> export LIBRARY_PATH=/libs:$LIBRARY_PATH`
- startup file: `export LIBRARY_PATH=/libs:$LIBRARY_PATH`

To compile `file.cpp` and link it with the library `mylib.a`, enter the following command:

```
icpc file.cpp mylib.a```

The compiler passes file names to the linker in the following order:

1. the object file
2. any objects or libraries specified on the command line, in a response file, or in a configuration file
3. the Intel® Math Library, `libimf.a`

Creating Libraries

Libraries are simply an indexed collection of object files that are included as needed in a linked program. Combining object files into a library makes it easy to distribute your code without disclosing the source. It also reduces the number of command-line entries needed to compile your project.
Static Libraries

Executables generated using static libraries are no different than executables generated from individual source or object files. Static libraries are not required at runtime, so you do not need to include them when you distribute your executable. At compile time, linking to a static library is generally faster than linking to individual source files.

To build a static library on Linux OS:

1. use the `-c` option to generate object files from the source files:
   ```bash
   icpc -c my_source1.cpp my_source2.cpp my_source3.cpp
   ```
2. use the GNU tool `ar` to create the library file from the object files:
   ```bash
   ar rc my_lib.a my_source1.o my_source2.o my_source3.o
   ```
3. compile and link your project with your new library:
   ```bash
   icpc main.cpp my_lib.a
   ```

   If your library file and source files are in different directories, use the `–L` option to indicate where your library is located:
   ```bash
   icpc -L/cpp/libs main.cpp my_lib.a
   ```

To build a static library on Mac OS X:

1. use the following command line to generate object files and create the library file:
   ```bash
   icpc -fpic -o mylib.a -staticlib my_source1.cpp my_source2.cpp my_source3.cpp
   ```
2. compile and link your project with your new library:
   ```bash
   icpc main.cpp my_lib.a
   ```

   If your library file and source files are in different directories, use the `–L` option to indicate where your library is located:
   ```bash
   icpc -L/cpp/libs main.cpp my_lib.a
   ```

If you are using Interprocedural Optimization, see Creating a Library from IPO Objects using `xiar`. 
Shared Libraries

Shared libraries, also referred to as dynamic libraries or Dynamic Shared Objects (DSO), are linked differently than static libraries. At compile time, the linker insures that all the necessary symbols are either linked into the executable, or can be linked at runtime from the shared library. Executables compiled from shared libraries are smaller, but the shared libraries must be included with the executable to function correctly. When multiple programs use the same shared library, only one copy of the library is required in memory.

To build a shared library on Linux* OS:

1. use the -fPIC and -c options to generate object files from the source files:
   
   ```
   icpc -fPIC -c my_source1.cpp my_source2.cpp my_source3.cpp
   ```

2. use the -shared option to create the library file from the object files:
   
   ```
   icpc -shared -o my_lib.so my_source1.o my_source2.o my_source3.o
   ```

3. compile and link your project with your new library:
   
   ```
   icpc main.cpp my_lib.so
   ```

To build a shared library on Mac OS* X:

1. use the following command line to generate object files and create the library file:
   
   ```
   icpc -fPIC -o my_lib.so -dynamiclib my_source1.cpp my_source2.cpp my_source3.cpp
   ```

2. compile and link your project with your new library:
   
   ```
   icpc main.cpp my_lib.dylib
   ```

See Also

Using Intel Shared Libraries
Compiling for Non-shared Libraries
Using Intel Shared Libraries

By default, the Intel® C++ Compiler links Intel-provided C++ libraries dynamically. The GNU*, Linux* OS, and Mac OS* X system libraries are also linked dynamically.

Options for Shared Libraries (Linux* OS)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-shared-intel</td>
<td>Use the -shared-intel option to link Intel-provided C++ libraries dynamically (default). This has the advantage of reducing the size of the application binary, but it also requires the libraries to be on the systems where the application runs.</td>
</tr>
<tr>
<td>-shared</td>
<td>The -shared option instructs the compiler to build a Dynamic Shared Object (DSO) instead of an executable. For more details, refer to the ld man page documentation.</td>
</tr>
<tr>
<td>-fpic</td>
<td>Use the -fpic option when building shared libraries. It is required for the compilation of each object file included in the shared library.</td>
</tr>
</tbody>
</table>

Options for Shared Libraries (Mac OS* X)

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dynamiclib</td>
<td>Use the -dynamiclib option to invoke the libtool command to generate dynamic libraries.</td>
</tr>
<tr>
<td>-fpic</td>
<td>Use the -fpic option when building shared libraries. It is required for the compilation of each object file included in the shared library.</td>
</tr>
</tbody>
</table>

Compiling for Non-shared Libraries
Overview: Compiling for Non-shared Libraries

This section includes information on:

- **Global Symbols and Visibility Attributes**
- **Symbol Preemption**
- **Specifying Symbol Visibility Explicitly**
- **Other Visibility-related Command-line Options**

Global Symbols and Visibility Attributes

A global symbol is one that is visible outside the compilation unit (single source file and its include files) in which it is declared. In C/C++, this means anything declared at file level without the `static` keyword. For example:

```c
int x = 5; // global data definition
extern int y; // global data reference
int five() { return 5; } // global function definition
extern int four(); // global function reference
```

A complete program consists of a main program file and possibly one or more shareable object (.so) files that contain the definitions for data or functions referenced by the main program. Similarly, shareable objects might reference data or functions defined in other shareable objects. Shareable objects are so called because if more than one simultaneously executing process has the shareable object mapped into its virtual memory, there is only one copy of the read-only portion of the object resident in physical memory. The main program file and any shareable objects that it references are collectively called the components of the program.

Each global symbol definition or reference in a compilation unit has a visibility attribute that controls how (or if) it may be referenced from outside the component in which it is defined. There are five possible values for visibility:

- **EXTERNAL** – The compiler must treat the symbol as though it is defined in another component. For a definition, this means that the compiler must assume that the symbol will be overridden (preempted) by a definition of the same name in another component. See Symbol Preemption. If a function
symbol has external visibility, the compiler knows that it must be called indirectly and can inline the indirect call stub.

- **DEFAULT** – Other components can reference the symbol. Furthermore, the symbol definition may be overridden (preempted) by a definition of the same name in another component.
- **PROTECTED** – Other components can reference the symbol, but it cannot be preempted by a definition of the same name in another component.
- **HIDDEN** – Other components cannot directly reference the symbol. However, its address might be passed to other components indirectly (for example, as an argument to a call to a function in another component, or by having its address stored in a data item reference by a function in another component).
- **INTERNAL** – The symbol cannot be referenced outside its defining component, either directly or indirectly.

Static local symbols (in C/C++, declared at file scope or elsewhere with the keyword static) usually have HIDDEN visibility--they cannot be referenced directly by other components (or, for that matter, other compilation units within the same component), but they might be referenced indirectly.

**Note**

Visibility applies to references as well as definitions. A symbol reference's visibility attribute is an assertion that the corresponding definition will have that visibility.

**Symbol Preemption**

Sometimes you may need to use some of the functions or data items from a shareable object, but may wish to replace others with your own definitions. For example, you may want to use the standard C runtime library shareable object, *libc.so*, but to use your own definitions of the heap management routines *malloc()* and *free()* within *libc.so* call your definition of the routines and not the definitions present in *libc.so*. Your definition should override, or preempt, the definition within the shareable object.
This feature of shareable objects is called symbol preemption. When the runtime loader loads a component, all symbols within the component that have default visibility are subject to preemption by symbols of the same name in components that are already loaded. Since the main program image is always loaded first, none of the symbols it defines will be preempted.

The possibility of symbol preemption inhibits many valuable compiler optimizations because symbols with default visibility are not bound to a memory address until runtime. For example, calls to a routine with default visibility cannot be inlined because the routine might be preempted if the compilation unit is linked into a shareable object. A preemptable data symbol cannot be accessed using GP-relative addressing because the name may be bound to a symbol in a different component; the GP-relative address is not known at compile time.

Symbol preemption is a very rarely used feature that has drastic negative consequences for compiler optimization. For this reason, by default the compiler treats all global symbol definitions as non-preemptable (i.e., protected visibility). Global references to symbols defined in other compilation units are assumed by default to be preemptable (i.e., default visibility). In those rare cases when you need all global definitions, as well as references, to be preemptable, specify the -fpic option to override this default.

**Specifying Symbol Visibility Explicitly**

You can explicitly set the visibility of an individual symbol using the `visibility` attribute on a data or function declaration. For example:

```c++
int i __attribute____((visibility("default")));
void __attribute__((visibility("hidden"))) x () {...}
extern void y() __attribute__((visibility("protected"));
```

The `visibility` declaration attribute accepts one of the five keywords:

- `external`
- `default`
- `protected`
- `hidden`
- `internal`
The value of the visibility declaration attribute overrides the default set by the -fvisibility, -fpic, or -fno-common attributes.

If you have a number of symbols for which you wish to specify the same visibility attribute, you can set the visibility using one of the five command line options:

- `-fvisibility-external=file`
- `-fvisibility-default=file`
- `-fvisibility-protected=file`
- `-fvisibility-hidden=file`
- `-fvisibility-internal=file`

where file is the pathname of a file containing a list of the symbol names whose visibility you wish to set. The symbol names in the file are separated by white space (blanks, TAB characters, or newlines). For example, the command line option:

- `-fvisibility-protected=prot.txt`

where file prot.txt contains:

```
a b c d
e
```

sets protected visibility for symbols a, b, c, d, and e. This has the same effect as __attribute__((visibility="protected")) on the declaration for each of the symbols. Note that these two ways to explicitly set visibility are mutually exclusive – you may use __attribute__((visibility())) on the declaration, or specify the symbol name in a file, but not both.

You can set the default visibility for symbols using one of the command line options:

- `-fvisibility=external`
- `-fvisibility=default`
- `-fvisibility=protected`
- `-fvisibility=hidden`
- `-fvisibility=internal`
This option sets the visibility for symbols not specified in a visibility list file and that do not have `__attribute__((visibility()))` in their declaration. For example, the command line options:

```
-fvisibility=protected -fvisibility-default=prot.txt
```

where file `prot.txt` is as previously described, will cause all global symbols except a, b, c, d, and e to have protected visibility. Those five symbols, however, will have default visibility and thus be preemptable.

**Other Visibility-related Command-line Options**

- **-fminshared**

The `-fminshared` option specifies that the compilation unit will be part of a main program component and will not be linked as part of a shareable object. Since symbols defined in the main program cannot be preempted, this allows the compiler to treat symbols declared with default visibility as though they have protected visibility (i.e., `-fminshared` implies `-fvisibility=protected`). Also, the compiler need not generate position-independent code for the main program. It can use absolute addressing, which may reduce the size of the global offset table (GOT) and may reduce memory traffic.

- **-fpic**

The `-fpic` option specifies full symbol preemption. Global symbol definitions as well as global symbol references get default (i.e., preemptable) visibility unless explicitly specified otherwise.

- **-fno-common**

Normally a C/C++ file-scope declaration with no initializer and without the `extern` or `static` keyword

```c
int i;
```

is represented as a common symbol. Such a symbol is treated as an external reference, except that if no other compilation unit has a global definition for the name, the linker allocates memory for it. The `-fno-common` option causes the
compiler to treat what otherwise would be common symbols as global definitions and to allocate memory for the symbol at compile time. This may permit the compiler to use the more efficient GP-relative addressing mode when accessing the symbol.

See Also

- `fminshared` compiler option
- `fpic` compiler option
- `fno-common` compiler option

**gcc** Compatibility

C language object files created with the Intel® C++ Compiler are binary compatible with the GNU gcc* compiler and glibc*, the GNU C language library. You can use the Intel compiler or the gcc compiler to pass object files to the linker. However, to correctly pass the Intel libraries to the linker, use the Intel compiler.

The Intel C++ Compiler supports many of the language extensions provided by the GNU compilers.

**gcc Extensions to the C Language**

GNU C includes several, non-standard features not found in ISO standard C. This version of the Intel C++ Compiler supports most of these extensions listed in the following table. See http://www.gnu.org for more information.

<table>
<thead>
<tr>
<th>gcc Language Extension</th>
<th>Intel Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Statements and Declarations in Expressions</td>
<td>Yes</td>
</tr>
<tr>
<td>Locally Declared Labels</td>
<td>Yes</td>
</tr>
<tr>
<td>Labels as Values</td>
<td>Yes</td>
</tr>
<tr>
<td>Nested Functions</td>
<td>No</td>
</tr>
<tr>
<td>gcc Language Extension</td>
<td>Intel Support</td>
</tr>
<tr>
<td>------------------------------------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Constructing Function Calls</td>
<td>No</td>
</tr>
<tr>
<td>Naming an Expression’s Type</td>
<td>Yes</td>
</tr>
<tr>
<td>Referring to a Type with typeof</td>
<td>Yes</td>
</tr>
<tr>
<td>Generalized Lvalues</td>
<td>Yes</td>
</tr>
<tr>
<td>Conditionals with Omitted Operands</td>
<td>Yes</td>
</tr>
<tr>
<td>Double-Word Integers</td>
<td>Yes</td>
</tr>
<tr>
<td>Complex Numbers</td>
<td>Yes</td>
</tr>
<tr>
<td>Hex Floats</td>
<td>Yes</td>
</tr>
<tr>
<td>Arrays of Length Zero</td>
<td>Yes</td>
</tr>
<tr>
<td>Arrays of Variable Length</td>
<td>Yes</td>
</tr>
<tr>
<td>Macros with a Variable Number of Arguments.</td>
<td>Yes</td>
</tr>
<tr>
<td>Slightly Looser Rules for Escaped Newlines</td>
<td>No</td>
</tr>
<tr>
<td>String Literals with Embedded Newlines</td>
<td>Yes</td>
</tr>
<tr>
<td>Non-Lvalue Arrays May Have Subscripts</td>
<td>Yes</td>
</tr>
<tr>
<td>Arithmetic on void-Pointers</td>
<td>Yes</td>
</tr>
<tr>
<td>Arithmetic on Function-Pointers</td>
<td>Yes</td>
</tr>
<tr>
<td>Non-Constant Initializers</td>
<td>Yes</td>
</tr>
<tr>
<td>Compound Literals</td>
<td>Yes</td>
</tr>
<tr>
<td>Designated Initializers</td>
<td>Yes</td>
</tr>
<tr>
<td>Cast to a Union Type</td>
<td>Yes</td>
</tr>
<tr>
<td>Case Ranges</td>
<td>Yes</td>
</tr>
<tr>
<td>Mixed Declarations and Code</td>
<td>Yes</td>
</tr>
<tr>
<td>gcc Language Extension</td>
<td>Intel Support</td>
</tr>
<tr>
<td>----------------------------------------------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Declaring Attributes of Functions</td>
<td>Yes</td>
</tr>
<tr>
<td>Attribute Syntax</td>
<td>Yes</td>
</tr>
<tr>
<td>Prototypes and Old-Style Function Definitions</td>
<td>No</td>
</tr>
<tr>
<td>C++ Style Comments</td>
<td>Yes</td>
</tr>
<tr>
<td>Dollar Signs in Identifier Names</td>
<td>Yes</td>
</tr>
<tr>
<td>ESC Character in Constants</td>
<td>Yes</td>
</tr>
<tr>
<td>Specifying Attributes of Variables</td>
<td>Yes</td>
</tr>
<tr>
<td>Specifying Attributes of Types</td>
<td>Yes</td>
</tr>
<tr>
<td>Inquiring on Alignment of Types or Variables</td>
<td>Yes</td>
</tr>
<tr>
<td>Inline Function is As Fast As a Macro</td>
<td>Yes</td>
</tr>
<tr>
<td>Assembler Instructions with C Expression Operands</td>
<td>Yes</td>
</tr>
<tr>
<td>Controlling Names Used in Assembler Code</td>
<td>Yes</td>
</tr>
<tr>
<td>Variables in Specified Registers</td>
<td>Yes</td>
</tr>
<tr>
<td>Alternate Keywords</td>
<td>Yes</td>
</tr>
<tr>
<td>Incomplete enum Types</td>
<td>Yes</td>
</tr>
<tr>
<td>Function Names as Strings</td>
<td>Yes</td>
</tr>
<tr>
<td>Getting the Return or Frame Address of a Function</td>
<td>Yes</td>
</tr>
<tr>
<td>Using Vector Instructions Through Built-in Functions</td>
<td>No</td>
</tr>
<tr>
<td>Other built-in functions provided by GCC</td>
<td>Yes</td>
</tr>
<tr>
<td>Built-in Functions Specific to Particular Target Machines</td>
<td>No</td>
</tr>
<tr>
<td>Pragmas Accepted by GCC</td>
<td>Yes</td>
</tr>
<tr>
<td>Unnamed struct/union fields within structs/union</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### g++ Language Extension

<table>
<thead>
<tr>
<th>Feature</th>
<th>Intel Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal floating types</td>
<td>Yes</td>
</tr>
<tr>
<td>Minimum and Maximum operators in C++</td>
<td>Yes</td>
</tr>
<tr>
<td>When is a Volatile Object Accessed?</td>
<td>No</td>
</tr>
<tr>
<td>Restricting Pointer Aliasing</td>
<td>Yes</td>
</tr>
<tr>
<td>Vague Linkage</td>
<td>Yes</td>
</tr>
<tr>
<td>Declarations and Definitions in One Header</td>
<td>No</td>
</tr>
<tr>
<td>Where's the Template?</td>
<td>extern supported</td>
</tr>
<tr>
<td>Extracting the function pointer from a bound pointer to member function</td>
<td>Yes</td>
</tr>
<tr>
<td>C++-Specific Variable, Function, and Type Attributes</td>
<td>Yes</td>
</tr>
<tr>
<td>Java Exceptions</td>
<td>No</td>
</tr>
<tr>
<td>Deprecated Features</td>
<td>No</td>
</tr>
<tr>
<td>Backwards Compatibility</td>
<td>No</td>
</tr>
</tbody>
</table>
Statement expressions are supported, except the following are prohibited inside them:

- dynamically-initialized local static variables
- local non-POD class definitions
- try/catch

Also, branching out of a statement expression is not allowed, and statement expressions may not be used in default argument expressions. Variable-length arrays are no longer allowed in statement expressions.

**Note**

The Intel C++ Compiler supports gcc-style inline ASM if the assembler code uses AT&T* System V/386 syntax.

**gcc* Interoperability**

**gcc Interoperability**

C++ compilers are interoperable if they can link object files and libraries generated by one compiler with object files and libraries generated by the second compiler, and the resulting executable runs successfully. The Intel® C++ Compiler is highly compatible with the gcc and g++ compilers. This section describes features of the Intel C++ Compiler that provide interoperability with gcc and g++.

**See Also**

* gcc Compatibility
* Compiler Options for Interoperability
* Predefined Macros for Interoperability

**Compiler Options for Interoperability**

The Intel® C++ Compiler options that affect gcc* interoperability include:

- `-gcc-name=dir`
- `-gcc-version=nnn`
- `-gxx-name=dir`
- `-cxxlib`
-fabi-version=n
- no-gcc

-gcc-name option
The -gcc-name=dir option, used with -cxxlib, lets you specify the full-path location of gcc if the compiler cannot locate the gcc C++ libraries. Use this option when referencing a non-standard gcc installation.

-gcc-version option
The -gcc-version=nnn option provides compatible behavior with gcc, where nnn indicates the gcc version. The -gcc-version option is ON by default, and the value of nnn depends on the version of gcc installed on your system. This option selects the version of gcc with which you achieve ABI interoperability.

<table>
<thead>
<tr>
<th>Installed Version of gcc</th>
<th>Default Value of -gcc-version</th>
</tr>
</thead>
<tbody>
<tr>
<td>older than version 3.2</td>
<td>not set</td>
</tr>
<tr>
<td>3.2</td>
<td>320</td>
</tr>
<tr>
<td>3.3</td>
<td>330</td>
</tr>
<tr>
<td>3.4</td>
<td>340</td>
</tr>
<tr>
<td>4.0</td>
<td>400</td>
</tr>
<tr>
<td>4.1</td>
<td>410</td>
</tr>
<tr>
<td>4.2</td>
<td>420</td>
</tr>
<tr>
<td>4.3</td>
<td>430</td>
</tr>
</tbody>
</table>

-gxx-name option
The -gxx-name=dir option specifies that the g++ compiler should be used to set up the environment for C++ compilations.
The `-cxxlib[=dir]` option (ON by default) builds your applications using the C++ libraries and header files included with the gcc compiler. They include:

- `libstdc++` standard C++ header files
- `libstdc++` standard C++ library
- `libgcc` C++ language support

Use the optional argument, `=dir`, to specify the top-level location for the gcc binaries and libraries.

**Note**

The Intel C++ Compiler is compatible with gcc 3.2, 3.3, 3.4, 4.0, 4.1 and 4.2.

When you compile and link your application, the resulting C++ object files and libraries can interoperate with C++ object files and libraries generated by gcc 3.2 or higher. This means that third-party C++ libraries built with gcc 3.2 will work with C++ code generated by the Intel Compiler.

**Note**

gcc 3.2, 3.3, and 3.4 are not interoperable. gcc 4.0, 4.1, and 4.2 are interoperable. By default, the Intel compiler will generate code that is interoperable with the version of gcc it finds on your system.

By default, the Intel C++ Compiler uses headers and libraries included with the product, when the system includes a version of gcc less than 3.2.

If you build one shared library against the Intel C++ libraries, build a second shared library against the gnu C++ libraries, and use both libraries in a single application, you will have two C++ run-time libraries in use. Since the application might use symbols from both libraries, the following problems may occur:

- partially initialized libraries
- lost I/O operations from data put in unaccessed buffers
- other unpredictable results, such as jumbled output

The Intel C++ Compiler does not support more than one run-time library in one application.

**Caution**
If you successfully compile your application using more than one run-time library, the resulting program will likely be very unstable, especially when new code is linked against the shared libraries.

-fabi-version

The \texttt{-fabi-version=n} option directs the compiler to select a specific ABI implementation. By default, the Intel compiler uses the ABI implementation that corresponds to the installed version of gcc. Both gcc 3.2 and 3.3 are not fully ABI-compliant.

<table>
<thead>
<tr>
<th>Value of n</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{n=0}</td>
<td>Select most recent ABI implementation</td>
</tr>
<tr>
<td>\texttt{n=1}</td>
<td>Select g++ 3.2 compatible ABI implementation</td>
</tr>
<tr>
<td>\texttt{n=2}</td>
<td>Select most conformant ABI implementation</td>
</tr>
</tbody>
</table>

See Also

- \texttt{Specifying Alternate Tools and Paths}
- \texttt{-gcc-name} compiler option
- \texttt{-gcc-version} compiler option
- \texttt{-cxxlib} compiler option
- \texttt{-fabi-version} compiler option
- \texttt{-no-gcc} compiler option

For more information on ABI conformance, see \url{http://www.codesourcery.com/}

Predefined Macros for Interoperability

The Intel® C++ Compiler and gcc*/g++* support the following predefined macros:

- \texttt{__GNUC__}
- \texttt{__GNUG__}
- \texttt{__GNUC_MINOR__}
- \texttt{__GNUC_PATCHLEVEL__}
You can specify the `-no-gcc` option to undefine these macros. If you need gcc interoperability (`-cxxlib`), do not use the `-no-gcc` compiler option.

⚠️ Caution

Not defining these macros results in different paths through system header files. These alternate paths may be poorly tested or otherwise incompatible.

See Also

- **Predefined Macros**
- **GNU Environment Variables**

**gcc Built-in Functions**

This version of the Intel® C++ compiler supports the following gcc* built-in functions:

```cpp
__builtin_abs
__builtin_labs
__builtin_cos
__builtin_cosf
__builtin_expect
__builtin_fabs
__builtin_fabsf
__builtin_memcmp
__builtin_memcpy
__builtin_sin
__builtin_sinf
__builtin_sqrt
__builtin_sqrtf
__builtin_strcmp
__builtin_strlen
__builtin_strncmp
__builtin_abort
__builtin_prefetch
```
__builtin_constant_p
__builtin_printf
__builtin_fprintf
__builtin_fscanf
__builtin_scanf
__builtin_fputs
__builtin_memset
__builtin_strcat
__builtin_strcpy
__builtin_strncpy
__builtin_exit
__builtin_strchr
__builtin_strspn
__builtin_strcspn
__builtin_strlen
__builtin_strchr
__builtin_strrchr
__builtin_strrchr
__builtin_strncat
__builtin_alloca
__builtin_ffs
__builtin_index
__builtin_rindex
__builtin_bcmp
__builtin_bzero
__builtin_sinl
__builtin_cosl
__builtin_sqrtl
__builtin_fabsl
__builtin_frame_address (IA-32 architecture only)
__builtin_return_address (IA-32 architecture only)

Thread-local Storage
The Intel® C++ Compiler supports the storage class keyword __thread, which can be used in variable definitions and declarations. Variables defined and declared this way are automatically allocated locally to each thread:

```c
__thread int i;
__thread struct state s;
extern __thread char *p;
```

**Note**
The __thread keyword is only recognized when the GNU compatibility version is 3.3 or higher. You may need to specify the -gcc-version=330 compiler option to enable thread-local storage.

## Language Conformance

### Conformance to the C Standard

The Intel® C++ Compiler provides conformance to the ANSI/ISO standard for C language compilation (ISO/IEC 9899:1990). This standard requires that conforming C compilers accept minimum translation limits. This compiler exceeds all of the ANSI/ISO requirements for minimum translation limits.

### C99 Support

The following C99 features are supported in this version of the Intel C++ Compiler:

- restricted pointers (restrict keyword).
- variable-length Arrays
- flexible array members
- complex number support (__Complex keyword)
- hexadecimal floating-point constants
- compound literals
- designated initializers
- mixed declarations and code
- macros with a variable number of arguments
- inline functions (inline keyword)
- boolean type (_Bool keyword)

Conformance to the C++ Standard


Exported Templates

The Intel® C++ Compiler supports exported templates using the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-export</td>
<td>Enable recognition of exported templates. Supported in C++ mode only.</td>
</tr>
<tr>
<td>-export-dir</td>
<td>Specifies a directory name to be placed on the exported template search path.</td>
</tr>
</tbody>
</table>

| dir         | Specifies a directory name to be placed on the exported template search path. |

Exported templates are templates declared with the export keyword. Exporting a class template is equivalent to exporting each of its static data members and each of its non-inline member functions. An exported template is unique because its definition does not need to be present in a translation unit that uses that template. For example, the following C++ program consists of two separate translation units:

```cpp
// file1.cpp
#include <stdio.h>
static void trace() { printf("File 1\n"); } export template<class T> T const& min(T const&, T const&);
```
int main() {
    trace();
    return min(2, 3);
}

// file2.cpp
#include <stdio.h>
static void trace() { printf("File 2\n"); }
export template<class T> T const& min(T const &a, T const &b) {
    trace();
    return a<b? a: b;
}

Note that these two files are separate translation units: one is not included in the other. That allows the two functions trace() to coexist (with internal linkage).

Usage

icpc -export -export-dir /usr2/export/ -c file1.cpp icpc -export
    -export-dir /usr2/export/ -c file2.cpp icpc -export -export-dir
    /usr2/export/ file1.o file2.o

See Also

-export compiler option
-export-dir compiler option

Template Instantiation

The Intel® C++ Compiler supports extern template, which lets you specify that a template in a specific translation unit will not be instantiated because it will be instantiated in a different translation unit or different library. The compiler now includes additional support for:

- inline template – instantiates the compiler support data for the class (i.e. the vtable) for a class without instantiating its members.
- static template – instantiates the static data members of the template, but not the virtual tables or member functions.

You can now use the following options to gain more control over the point of template instantiation:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fno-</td>
<td>Never emit code for non-inline templates which are instantiated</td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>implicit-templates</code></td>
<td>implicitly (i.e. by use). only emit code for explicit instantiations.</td>
</tr>
<tr>
<td><code>-fno-implicit-templates</code></td>
<td>Do not emit code for implicit instantiations of inline templates either.</td>
</tr>
<tr>
<td><code>-fno-implicit-inline-templates</code></td>
<td>The default is to handle inlines differently so that compilations, with and without optimization, will need the same set of explicit instantiations.</td>
</tr>
</tbody>
</table>

**See Also**

- `-fno-implicit-templates` compiler option
- `-fno-implicit-inline-templates` compiler option

## Porting Applications

### Overview: Porting Applications

This section describes a basic approach to porting applications from GCC's C/C++ compilers to the Intel® C/C++ compilers. These compilers correspond to each other in the following ways:

<table>
<thead>
<tr>
<th>Language</th>
<th>Intel Compiler</th>
<th>GCC* Compiler</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>icc</td>
<td>gcc</td>
</tr>
<tr>
<td>C++</td>
<td>icpc</td>
<td>g++</td>
</tr>
</tbody>
</table>

It also contains information on how to use the `-diag enable port-win` option to issue warnings about general syntactical problems when porting from GNU gcc* to Microsoft C++.

**Note**

To simplify this discussion on porting applications, the term "gcc", unless otherwise indicated, refers to both gcc and g++ compilers from the GNU Compiler Collection*.
Advantages to Using the Intel Compiler

In many cases, porting applications from gcc to the Intel compiler can be as easy as modifying your makefile to invoke the Intel compiler (icc) instead of gcc. Using the Intel compiler typically improves the performance of your application, especially for those that run on Intel processors. In many cases, your application's performance may also show improvement when running on non-Intel processors. When you compile your application with the Intel compiler, you have access to:

- compiler options that optimize your code for the latest Intel processors on IA-32 architecture, Intel® 64 architecture, and IA-64 architecture.
- advanced profiling tools (PGO) similar to gprof.
- high-level optimizations (HLO).
- interprocedural optimization (IPO).
- Intel intrinsic functions that the compiler uses to inline instructions, including SSE, SSE2, SSE3, SSSE3, and SSE4.
- the highly-optimized Intel Math Library for improved accuracy.

Since the Intel compiler is compatible and interoperable with gcc, porting your gcc application to the Intel compiler includes the benefits of binary compatibility. As a result, you should not have to re-build libraries from your gcc applications. The Intel compiler also supports many of the same compiler options, macros, and environment variables you already use in your gcc work.

Porting Strategy

For many gcc applications, porting to the Intel compiler requires little more than modifying your makefile to account for differences that may exist between compiling with gcc and compiling with icc.

One challenge in porting applications from one compiler to another is making sure there is support for the compiler options you use to build your application. The Compiler Options reference lists compiler options that are supported by both the Intel® C++ Compiler and gcc.

Next Steps
Modifying Your Makefile

If you use makefiles to build your gcc application, you need to change the value for the `CC` compiler variable to use the Intel compiler. You may also want to review the options specified by `CFLAGS`. A simple example follows:

```bash
# Use gcc compiler
CC = gcc
# Compile-time flags
CFLAGS = -O2 -std=c99
all: area_app
area_app: area_main.o area_functions.o
  $(CC) area_main.o area_functions.o -o area
area_main.o: area_main.c
  $(CC) -c $(CFLAGS) area_main.c
area_functions.o: area_functions.c
  $(CC) -c -fno-asm $(CFLAGS) area_functions.c
clean:
  rm -rf *o area
```

**Modified makefile for Intel Compiler**

```bash
# Use Intel C compiler
CC = icc
# Compile-time flags
CFLAGS = -std=c99
all: area-app
area-app: area_main.o area_functions.o
  $(CC) area_main.o area_functions.o -o area
area_main.o: area_main.c
  $(CC) -c $(CFLAGS) area_main.c
area_functions.o: area_functions.c
  $(CC) -c -fno-asm $(CFLAGS) area_functions.c
clean:
  rm -rf *o area
```

If your gcc code includes features that are not supported with the Intel compiler, such as compiler options, language extensions, macros, pragmas, etc., you can compile those sources separately with gcc if necessary.

In the above makefile, `area_functions.c` is an example of a source file that includes features unique to gcc. Since the Intel compiler uses the `-O2` compiler option by default and gcc's default is `-O0`, we instruct gcc to compile with `-O2`. 
We also include the \texttt{-fno-asim} switch from the original makefile since this switch is not supported with the Intel compiler. With the modified makefile, the output of \texttt{make} is:

\begin{verbatim}
  icc -c -std=c99 area_main.c  
  gcc -c -O2 -fno-asim -std=c99 area_functions.c  
  icc area_main.o area_functions.o -o area
\end{verbatim}

\textbf{See Also}

\begin{itemize}
\item \texttt{Equivalent Macros}
\item \texttt{Equivalent Environment Variables}
\item \texttt{Compiler Options for Interoperability}
\item \texttt{Support for gcc and g++ Language Extensions}
\item \texttt{Predefined Macros}
\item \texttt{gcc Builtin Functions}
\end{itemize}

\textbf{Equivalent Macros}

Macro support is an important aspect in porting applications from \texttt{gcc} to the Intel compiler. The following table lists the most common macros used in both compilers.

\begin{verbatim}
__CHAR_BIT__
__DATE__
__DBL_DENORM_MIN__
__DBL_DIG__
__DBL_EPSILON__
__DBL_HAS_INFINITY__
__DBL_HAS_QUIET_NAN__
__DBL_MANT_DIG__
__DBL_MAX__
__DBL_MAX_10_EXP__
\end{verbatim}
__GNUC_MINOR__
__GNUC_PATCHLEVEL__
__GXX_ABI_VERSION
__i386
__i386__
__INT_MAX__
__LDBL_DENORM_MIN__
__LDBL_DIG__
__LDBL_EPSILON__
__LDBL_HAS_INFINITY__
__LDBL_HAS_QUIET_NAN__
__LDBL_MANT_DIG__
__LDBL_MAX__
__LDBL_MAX_10_EXP__
__LDBL_MAX_EXP__
__LDBL_MIN__
__LDBL_MIN_10_EXP__
__LDBL_MIN_EXP__
__linux
__linux__
__LONG_LONG_MAX__
__LONG_MAX__
__NO_INLINE__
__OPTIMIZE__
See Also

Predefined Macros

Equivalent Environment Variables

The Intel® C++ Compiler supports many of the same environment variables used with gcc as well as other compiler-specific variables. See Setting Environment Variables for a list of supported variables.
Other Considerations

There are some notable differences between the Intel® C++ Compiler and gcc. Consider the following as you begin compiling your source code with the Intel C++ Compiler.

Setting the Environment

The Intel compilers rely on environment variables for the location of compiler binaries, libraries, man pages, and license files. In some cases these are different from the environment variables that gcc uses. Another difference is that these variables are not set by default after installing the Intel compiler. The following environment variables need to be set prior to running the Intel compiler:

- PATH – add the location of the compiler binaries to PATH
- LD_LIBRARY_PATH (Linux OS) – sets the location of the compiler libraries as well as the resulting binary generated by the compiler
- DYLD_LIBRARY_PATH (Mac OS X) - sets the location of the compiler libraries as well as the resulting binary generated by the compiler
- MANPATH – add the location of the compiler man pages (icc and icpc) to MANPATH
- INTEL_LICENSE_FILE – sets the location of the Intel compiler license file

You can 'source' the iccvars.sh shell script (included with the compiler) to set these environment variables for you. See Invoking the Compiler from the Command Line for details on using iccvars.sh.

Note

Setting these environment variables with iccvars.sh does not impose a conflict with gcc. You should be able to use both compilers in the same shell.

Using Optimization

The Intel C++ Compiler is an optimizing compiler that begins with the assumption that you want improved performance from your application when it is executed on
Intel architecture. Consequently, certain optimizations, such as `-O2`, are part of the default invocation of the Intel compiler. By default, gcc turns off optimization - the equivalent of compiling with `-O` or `-O0`. Other forms of the `-O<n>` option compare as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Intel</th>
<th>gcc</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-O0</code></td>
<td>Turns off optimization.</td>
<td>Default. Turns off optimization. Same as <code>-O</code>.</td>
</tr>
<tr>
<td><code>-O1</code></td>
<td>Decreases code size with some increase in speed.</td>
<td>Decreases code size with some increase in speed.</td>
</tr>
<tr>
<td><code>-O2</code></td>
<td>Default. Favors speed optimization with some increase in code size. Same as <code>-O</code>. Intrinsics, loop unrolling, and inlining are performed.</td>
<td>Optimizes for speed as long as there is not an increase in code size. Loop unrolling and function inlining, for example, are not performed.</td>
</tr>
<tr>
<td><code>-O3</code></td>
<td>Enables <code>-O2</code> optimizations plus more aggressive optimizations, such as prefetching, scalar replacement, and loop and memory access transformations.</td>
<td>Optimizes for speed while generating larger code size. Includes <code>-O2</code> optimizations plus loop unrolling and inlining. Similar to <code>-O2 -ip</code> on Intel compiler.</td>
</tr>
</tbody>
</table>

**Targeting Intel Processors**

While many of the same options that target specific processors are supported with both compilers, Intel includes options that utilize processor-specific instruction scheduling to target the latest Intel processors. If you compile your gcc application with the `-march` or `-mtune` option, consider using Intel’s `-x` or `-ax` options for applications that run on IA-32 or Intel® 64 architecture.

**Modifying Your Configuration**
The Intel compiler lets you maintain configuration and response files that are part of compilation. Options stored in the configuration file apply to every compilation, while options stored in response files apply only where they are added on the command line. If you have several options in your makefile that apply to every build, you may find it easier to move these options to the configuration file (.../bin/icc.cfg and .../bin/icpc.cfg).

In a multi-user, networked environment, options listed in the icc.cfg and icpc.cfg files are generally intended for everyone who uses the compiler. If you need a separate configuration, you can use the ICCCFG or ICPCCFG environment variable to specify the name and location of your own .cfg file, such as /my_code/my_config.cfg. Anytime you instruct the compiler to use a different configuration file, the system configuration files (icc.cfg and icpc.cfg) are ignored.

Using the Intel Math Library

With the Intel C++ Compiler, the Intel Math Library, libimf, is linked by default when calling math functions that require the library. Some functions, such as sin, do not require a call to the library, since the compiler already knows how to compute the sin function. The Intel Math Library includes many functions not found in the standard libm. A simple example using the sind function follows:

```c
define sind(double) 

do oble x = 90.0;

int main(void)
{
    printf("The sine of %f degrees is %f\n", x, sind(x));
    return(0);
}
```

Note

You cannot make calls the Intel Math Library with gcc.

See Also

-0 compiler option
Using Configuration Files
Using Response Files

Porting from GNU gcc* to Microsoft Visual C++*

You can use the -diag-enable port-win option to alert you to general syntactical problems when porting an application from GNU gcc* to Microsoft Visual C++*. The following examples illustrate use of this option.

Example 1:

```bash
$ cat -n attribute.c
 1 int main()
 2 {  
  3   int i __attribute__((unused));
  4   return 0;
  5 }
$ icc -c -diag-enable port-win attribute.c
attribute.c(3): warning #2133: attributes are a GNU extension
    int i __attribute__((unused));
```

Example 2:

```bash
$ cat -n state_expr.c
 1 int main()
 2 {  
  3   int i = ({ int j; j = 1; j--; j;});
  4   return i;
  5 }
$ icc -c -diag-enable port-win state_expr.c
state_expr.c(3): warning #2132: statement expressions are a GNU extension
    int i = ({ int j; j = 1; j--; j;});
```

Error Handling

Remarks, Warnings, and Errors

This topic describes compiler remarks, warnings, and errors. The Intel® C++ Compiler displays these messages, along with the erroneous source line, on the standard output.

Remarks

Remark messages report common but sometimes unconventional use of C or C++. The compiler does not print or display remarks unless you specify the -w2 option. Remarks do not stop translation or linking. Remarks do not interfere with any output files. The following are some representative remark messages:
function declared implicitly
• type qualifiers are meaningless in this declaration
• controlling expression is constant

Warnings

Warning messages report legal but questionable use of the C or C++. The compiler displays warnings by default. You can suppress all warning messages with the \(-w\) compiler option. Warnings do not stop translation or linking. Warnings do not interfere with any output files. The following are some representative warning messages:
• declaration does not declare anything
• pointless comparison of unsigned integer with zero
• possible use of = where == was intended

Additional Warnings

This version of Intel C++ Compiler includes support for the following options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-W[no-]missing-prototypes)</td>
<td>Warn for missing prototypes</td>
</tr>
<tr>
<td>(-W[no-]pointer-arith)</td>
<td>Warn for questionable pointer arithmetic</td>
</tr>
<tr>
<td>(-W[no-]uninitialized)</td>
<td>Warn if a variable is used before being initialized</td>
</tr>
<tr>
<td>(-W[no-]deprecated)</td>
<td>Display warnings related to deprecated features</td>
</tr>
<tr>
<td>(-W[no-]abi)</td>
<td>Warn if generated code is not C++ ABI compliant</td>
</tr>
<tr>
<td>(-W[no-]unused-function)</td>
<td>Warn if declared function is not used</td>
</tr>
<tr>
<td>(-W[no-]unknown-pragmas)</td>
<td>Warn if an unknown #pragma directive is used</td>
</tr>
</tbody>
</table>
Errors

These messages report syntactic or semantic misuse of C or C++. The compiler always displays error messages. Errors suppress object code for the module containing the error and prevent linking, but they allow parsing to continue to detect other possible errors. Some representative error messages are:

- missing closing quote
- expression must have arithmetic type
- expected a ";"

Option Summary

Use the following compiler options to control remarks, warnings, and errors:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-w</td>
<td>Suppress all warnings</td>
</tr>
<tr>
<td>-w0</td>
<td>Display only errors</td>
</tr>
<tr>
<td>-w1</td>
<td>Display only errors and warnings</td>
</tr>
<tr>
<td>-w2</td>
<td>Display errors, warnings, and remarks</td>
</tr>
<tr>
<td>-Wbrief</td>
<td>Display brief one-line diagnostics</td>
</tr>
<tr>
<td>-Wcheck</td>
<td>Enable more strict diagnostics</td>
</tr>
</tbody>
</table>
Option | Result
---|---
-Werror-all | Change all warnings and remarks to errors
-Werror | Change all warnings to errors

You can also control the display of diagnostic information with variations of the `--diag` compiler option. This compiler option accepts numerous arguments and values, allowing you wide control over displayed diagnostic messages and reports.

Some of the most common variations include the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-diag-enable list</td>
<td>Enables a diagnostic message or a group of messages</td>
</tr>
<tr>
<td>-diag-disable list</td>
<td>Disables a diagnostic message or a group of messages</td>
</tr>
<tr>
<td>-diag-warning list</td>
<td>Tells the compiler to change diagnostics to warnings</td>
</tr>
<tr>
<td>-diag-error list</td>
<td>Tells the compiler to change diagnostics to errors</td>
</tr>
<tr>
<td>-diag-remark list</td>
<td>Tells the compiler to change diagnostics to remarks (comments)</td>
</tr>
</tbody>
</table>

The `list` items can be specific diagnostic IDs, one of the keywords `warn`, `remark`, or `error`, or a keyword specifying a certain group (`par`, `vec`, `driver`, `thread`, `par`, `port-win`, `sv`). For more information, see `--diag`.

Other diagnostic-related options include the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-diag-dump</td>
<td>Tells the compiler to print all enabled diagnostic messages and stop compilation</td>
</tr>
<tr>
<td>Option</td>
<td>Result</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-diag-file[=file]</td>
<td>Causes the results of diagnostic analysis to be output to a file</td>
</tr>
<tr>
<td>-diag-file-append[=file]</td>
<td>Causes the results of diagnostic analysis to be appended to a file</td>
</tr>
<tr>
<td>-diag-error-limit n</td>
<td>Specifies the maximum number of errors allowed before compilation stops</td>
</tr>
</tbody>
</table>

Using Static Verification Diagnostics

Static Verification Overview

Static verification is an additional diagnostic capability to help you debug your programs. You can use the static verification options to detect potential errors in your compiled code including:

- incorrect usage of OpenMP* directives
- inconsistent object declarations in different program units
- boundary violations
- uninitialized memory
- memory corruptions
- memory leaks
- incorrect usage of pointers and allocatable arrays
- dead code and redundant executions
- typographical errors or uninitialized variables

Static verification options can be used to analyze and find issues with source files; these source files need not form a whole program. (For instance, you can pass sources for libraries on the static verification command line.) In such cases, because only partial information is available about usage and modification of global objects, calls to routines, and so forth, analysis will be less exact. Your code must successfully compile, with no errors, for static verification options to take effect.
The current usage model is that static verification is added as an alternate build option to produce a diagnostic report. When you enable static verification, the compiler will not create an executable file. Object files that are produced when using static verification are not valid and should not be used for generating real executable or static/dynamic link libraries.

Static verification cannot be used in conjunction with cross-file interprocedural optimization (/Qipo or -ipo).

Due to generalization of input data (static verification is performed for all possible input data), static verification cannot detect all possible errors that may appear during program execution.

Static verification uses a set of heuristics associated with the modification and usage of program objects. For example, when an undefined element of an array is set to some value, it is assumed that any arbitrary element of the array has been set to the same value. Similarly, after an undefined procedure call, it is assumed that all arguments substituted by reference are used and possibly modified.

Run-time checking executes a program with a fixed set of values for its input variables so it is difficult to check all edge effects. However, static verification performs a general overview check of a program for all possible values simultaneously.

Using the Static Verification Options

**Use the** `-diag-enable sv{[1|2|3]}` (Linux* OS and Mac OS* X) or `/Qdiag-enable:sv{[1|2|3]}` (Windows* OS) **compiler option to enable static verification.** The number specifies the severity level of the diagnostics (1=all critical errors, 2=all errors, and 3=all errors and warnings).

Other related options include the following:

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>-diag-disable sv</td>
<td>Disables static verification</td>
</tr>
<tr>
<td>/Qdiag-disable:sv</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Result</td>
</tr>
<tr>
<td>---------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-diag-disable warn</td>
<td>Suppresses all warnings, cautions and comments (issues errors only), including those specific to static verification</td>
</tr>
<tr>
<td>/Qdiag-disable:warn</td>
<td>Suppresses messages by number list, where num-list is either a single message or a list of message numbers separated by commas and enclosed in parentheses.</td>
</tr>
<tr>
<td>-diag-file [file]</td>
<td>Directs diagnostic results to file with .diag as the default extension. You need to enable static verification diagnostics before you can send them to a file. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.</td>
</tr>
<tr>
<td>/Qdiag-file[:file]</td>
<td></td>
</tr>
<tr>
<td>-diag-file-append[=file]</td>
<td>Appsends diagnostic results to file with .diag as the default extension. You need to enable static verification diagnostics before you can send the results to a file. If you do not specify a path, the current working directory will be searched. If the named file is not found, a new file with that name will be created. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.</td>
</tr>
<tr>
<td>/Qdiag-file-append[:file]</td>
<td></td>
</tr>
<tr>
<td>-diag-enable sv-include</td>
<td>Diagnoses include files as well as source(s)</td>
</tr>
<tr>
<td>/Qdiag-enable:sv-include</td>
<td></td>
</tr>
</tbody>
</table>
Using -diag-enable sv with other Compiler Options

If the -c (Linux OS and Mac OS X) or /c (Windows OS) compiler option is used on the command line along with a command line option to enable static verification, an object file is created; static verification diagnostics are not produced. This object file may be used in a further invocation of static verification. This feature is useful when a program consists of files written in different languages (C/C++ and Fortran).

To analyze OpenMP directives, add the -openmp (Linux OS and Mac OS X) or /Qopenmp (Windows OS) option to the command line.

Using Static Verification within the IDE

When static verification support is enabled within the IDE, the customary final build target (e.g. an executable image) is not created. Therefore, create a separate "Static Verification" configuration.

In the Microsoft Visual Studio Environment, modify the existing Debug (development) configuration, as follows:

1. Select the Project Configuration Manager either from the Build menu or from the Project Properties window.
2. In the Active Solution Configuration dropdown, select <New..>
3. Provide a name for this configuration.
4. Specify to "Copy Settings from" the Debug configuration.
5. Exit the Configuration Manager.

With the new configuration active, navigate to the C/C++ > Diagnostics property page. Use the Level of Static Analysis and Analyze Include Files properties to control static verification.

In the Eclipse* IDE, do the following:

1. Open the property pages for the project and select C/C++ Build.
2. Click the Manage... button.
3. In the Manage dialog box, click the New... button to open the Create configuration dialog box.
4. Supply a name for the new configuration in the Name box.
5. Supply a Description for the configuration if you want (optional).
6. You can choose to Copy settings from a Default configuration or an Existing configuration by clicking the appropriate radio button and then selecting a configuration from the corresponding drop down menu.
7. Click OK to close the Create configuration dialog box.
8. Click OK to close the Manage dialog box (with your new configuration name selected).

The property pages will now display the settings for your new configuration; this becomes the active build configuration. Navigate to the Intel compiler’s Compilation Diagnostics properties. Use the Level of Static Analysis and Analyze Include Files properties to control Static verification.

In the Xcode IDE, modify the existing debug (development) configuration, as follows:

1. Double click the Target to Get Info
2. Navigate to the Build properties page.
3. Expand the Configuration drop down and select Edit Configurations...
4. Select the Debug configuration.
5. Click the Duplicate button at the bottom of the page.
6. Optionally change the configuration name; the default name is Debug copy.
7. Exit the page.

With the Configuration set to the new configuration, navigate to the Intel compiler’s DIAGNOSTIC collection of properties. Use the Level of Static Analysis and Analyze Include Files properties to control static verification.

Set the Active Target and Active Build Configuration appropriately to BUILD the static verification enabled configuration.
Static Verification Capabilities

Static verification capabilities include the following:

- **Interprocedural Analysis** for detecting inconsistent objects declarations in different program units.
- **Local Program Analysis** for analyzing each program unit separately and checking for various kinds of problems that will errors or warnings.
- **C/C++ specific Analysis** for analyzing C/C++ source code and checking for C/C++ specific error and warning conditions. Static verification also detects improper code style and flaws in object-oriented design solutions.
- **Fortran-specific Analysis** for analyzing Fortran source code and checking for Fortran-specific error and warning conditions.
- **OpenMP* Analysis** for checking for OpenMP API restrictions.

### Interprocedural Analysis

Static verification detects inconsistent object declarations in different program units, for example:

- Different external objects with the same name.
- Inconsistent declarations of a COMMON block.
- Mismatched number of arguments.
- Mismatched type, rank, shape, or/and size of an argument.
- Inconsistent declaration of a procedure and predeclarations (or interfaces) of this procedure.

The following example illustrates interprocedural analysis.

**Example: Wrong number of arguments**

File `controlf.c` contains function declaration in the following line:

```c
controlf()
```

File `uloop2.c` contains the following line:

```c
65  fds = controlf( 1 ) ;
```

Static verification issues the following message:

`uloop2.c(65): error #12020: [SV] number of actual arguments (1) in call of 'controlf' doesn't match the number of formal arguments (0); 'controlf' is defined at (file:controlf.c line:4)`
With static verification, the compiler analyzes each program unit separately and checks for various kinds of errors, warnings, and/or debatable points in user program. Examples of these errors are:

- Incorrect use or modification of an object
- Problems with memory (for example, leaks, corruptions, uninitialized memory)
- Incorrect use with pointers
- Boundaries violations
- Wrong value of an argument in an intrinsic call
- Dead code and redundant executions

The following examples illustrate local program analysis.

**Example 1: Object is smaller than required size**

File `chess.h` contains the following:

```c
278  typedef struct{
279     int path[MAXPLY];
280     unsigned char path_hashed;
281     unsigned char path_length;
282     unsigned char path_iteration_depth;
283  } CHESS_PATH;
```

File `initdata.h` contains the following:

```c
237  CHESS_PATH pv[MAXPLY];
```

File `quiesce.c` contains the following:

```c
153  memcpy(&pv[ply-1].path_hashed,&pv[ply].path_hashed,3);
```

Static verification issues the following message:

`quiesce.c(153): error #12224: [SV] Buffer overflow: size of object 'path_hashed' (1 bytes) is less than required size (3 bytes)`

**Example 2: Wrong type of intrinsic argument**

File `makefile` contains the following:

```c
31  CFLAGS2 = $(CFLAGS) -DVERSION=9 -DCOMPDATE=1994
```

File `version.c` contains the following:

```c
20  fprintf (stderr, '%s: version: %d, compiled: %s, cflags: %s\n',
21     ego, VERSION, COMPDATE, 'CFLAGS');
```

Static verification issues the following message:

`version.c(21): error #12037: [SV] actual argument 5 in call of 'fprintf' should be a pointer to 'char'`
C/C++ Specific Analysis

Static verification analyzes C/C++ source code and checks for various kinds of errors, warnings, and/or debatable points in your program. It also points out places of improper code style and flaws in object-oriented design solutions.

Static verification detects issues with the following:

- Memory management (leaks, mixing C and C++ memory management routines, smart pointer usage)
- C++ exception handling (uncaught exception, exception specification, exception from destructor/operator delete)
- operator new/operator delete

The following example illustrates C/C++ specific analysis.

```cpp
#include "stdio.h"
class A {
public:
    A() { destroy(); }
    void destroy() { clear0(); }
    virtual void clear() = 0;
    void clear0() { clear(); }
};
class B : public A {
public:
    B() { }
    virtual void clear() { printf("overloaded clear"); }
    virtual ~B() { }
};
int main() {
    B b;
    return 0;
}
```

Static verification issues the following message:

```
test2.cpp(10): warning #12327: [SV] pure virtual function 'clear' is called from constructor (file: test2.cpp line:4)
```

Fortran-Specific Analysis
Static verification analyzes Fortran source code and checks for various kinds of errors, warnings, and/or debatable points in user program. Static verification detects issues with the following:

- Function result
- COMMON blocks

The following example illustrates Fortran-specific analysis.

**Example 1: Undefined function result**

File dlarnd.f contains the following lines:

```fortran
  2 REAL*8 FUNCTION DLARND( IDIST, ISEED)
     . . .
  82  T1 = DLARAN( ISEED )
  84  IF( IDIST.EQ.1 ) THEN
  88  DLARND = T1
  89  ELSE IF( IDIST.EQ.2 ) THEN
  93  DLARND = TWO*T1 - ONE
  94  ELSE IF( IDIST.EQ.3 ) THEN
  98  T2 = DLARAN( ISEED )
  99  DLARND = SQRT( -TWO*LOG( T1 ) )*COS(TWOPI*T2 )
 100  END IF
 101  RETURN
 105  END
```

Static verification issues the following message:

dlarnd.f(105): error #12077: [SV] function result is possibly not set

**OpenMP* Analysis**

The compiler detects some restrictions noted in the OpenMP* API Versions 2.0 and 2.5. When static verification is enabled, the compiler performs some additional checks against restrictions in the OpenMP API, including checks for the correct use of the following:

- nested parallel regions including dynamic extent of parallel regions
- private variables in parallel regions
• thread-private variables
• expressions which are used in OpenMP clauses

**Example: Incorrect usage of OpenMP directives**

File gafort.f90 contains the following lines:

```fortran
310  !$OMP PARALLEL DO ORDERED
311  ! create an array of locks
312  !$ DO i = 1,indmax
313  !$ CALL omp_init_lock(lck(i))
314  !$ ENDDO
315  !$OMP END PARALLEL DO
```

The parallel region has the clause ORDERED but has no corresponding ORDERED OpenMP directive. Static verification issues the following message:

gafort.f90(310): error #12204: [SV] ORDERED clause is used in the dynamic extent of non-ORDERED DO construct

**Reference**

**ANSI Standard Predefined Macros**

The ANSI/ISO standard for the C language requires that certain predefined macros be supplied with conforming compilers. The following table lists the macros that the Intel® C++ Compiler supplies in accordance with this standard:

The compiler includes predefined macros in addition to those required by the standard. The default predefined macros differ among Windows*, Linux*, and Mac OS* X operating systems due to the default /Za compiler option on Windows. Differences also exist on Linux OS and Mac OS X as a result of the –std compiler option.

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DATE</strong></td>
<td>The date of compilation as a string literal in the form Mmm dd yyyy.</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>A string literal representing the name of the file being</td>
</tr>
</tbody>
</table>
__LINE__ The current line number as a decimal constant.

__STDC__ The name __STDC__ is defined when compiling a C translation unit.

__STDC_HOSTED__ 1

__TIME__ The time of compilation as a string literal in the form hh:mm:ss.

See Also

Additional Predefined Macros

The Intel® C++ Compiler supports the predefined macros listed in the table below. The compiler also includes predefined macros specified by the ISO/ANSI standard.

The following designations apply:

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td>Included on systems based on IA-32 architecture.</td>
</tr>
<tr>
<td>i64em</td>
<td>Included on systems based on Intel® 64 architecture.</td>
</tr>
<tr>
<td>i64</td>
<td>Included on systems based on IA-64 architecture.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
<th>i32</th>
<th>i64em</th>
<th>i64</th>
</tr>
</thead>
<tbody>
<tr>
<td>__ARRAY_OPERATORS</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>BASE_FILE</strong></td>
<td>Name of source file</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_BOOL</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__cplusplus</td>
<td>1 (with C++ compiler)</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Value</td>
<td>i32</td>
<td>i64em</td>
<td>i64</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>_CLUSTER_OPENMP</td>
<td>Defined as 1 when the <code>-cluster-openmp</code> or <code>-cluster-openmp-profile</code> compiler option is used.</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__DEPRECATED</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__ECC</td>
<td>Intel Compiler Version</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EDG</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>EDG_VERSION</strong></td>
<td>EDG version</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>ELF</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>extension</strong></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__EXCEPTIONS</td>
<td>Defined as 1 when <code>-fno-exceptions</code> is not used.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>GNUC</strong></td>
<td>The major version number of gcc installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>GNUG</strong></td>
<td>The major version number of g++ installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>gnu_linux</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>GNUC_MINOR</strong></td>
<td>The minor version number of gcc or g++ installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>GNUC_PATCHLEVEL</strong></td>
<td>The patch level version number of gcc or g++ installed on the system.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Value</td>
<td>i32</td>
<td>i64em</td>
<td>i64</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>__GXX_ABI_VERSION</td>
<td>102</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__HONOR_STD</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>__i386</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>i386</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>i386</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__ia64</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>ia64</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__ICC</td>
<td>Intel compiler version</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>_INTEGRAL_MAX_BITS</td>
<td>64</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__INTEL_COMPILER</td>
<td>Intel compiler version</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__INTEL_COMPILER_BUILD_DATE</td>
<td>YYYYMMDD</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>INTEL_RTTI</strong></td>
<td>Defined as 1 when -fno-rtti is not specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>INTEL STRICT_ANSI</strong></td>
<td>Defined as 1 when -strict-ansi is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>itanium</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>__linux</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>linux</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>linux</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>LONG_DOUBLE_SIZE</strong></td>
<td>80</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>LONG_MAX</strong></td>
<td>9223372036854775807L</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>__lp64</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>LP64</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>Macro Name</td>
<td>Value</td>
<td>i32</td>
<td>i64em</td>
<td>i64</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td>_LP64</td>
<td>1</td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_MT</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>MMX</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>NO_INLINE</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>__NO_MATH_INLINES</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>__NO_STRING_INLINES</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>_OPENMP</td>
<td>Defined as 200805 when -openmp is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>OPTIMIZE</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>__pentium4</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>pentium4</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>PIC</strong></td>
<td>Defined as 1 when -fPIC is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>pic</strong></td>
<td>Defined as 1 when -fPIC is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_PGO_INSTRUMENT</td>
<td>Defined as 1 when -prof-gen[x] is specified.</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>_PLACEMENT_DELETE</td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>PTRDIFF_TYPE</strong></td>
<td>int on IA-32 architecture</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>long on Intel® 64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>long on IA-64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REGISTER_PREFIX</strong></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td><strong>SIGNED_CHARS</strong></td>
<td>1</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Macro Name</td>
<td>Value</td>
<td>i32</td>
<td>i64em</td>
<td>i64</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------</td>
<td>-----</td>
<td>-------</td>
<td>-----</td>
</tr>
<tr>
<td><strong>SIZE_TYPE</strong></td>
<td>unsigned on IA-32</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned long on Intel® 64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>unsigned long on IA-64 architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SSE</strong></td>
<td>Defined as 1 for processors</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>that support SSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>instructions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SSE2</strong></td>
<td>Defined as 1 for processors</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>that support SSE2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>instructions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SSE3</strong></td>
<td>Defined as 1 for processors</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>that support SSE3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>instructions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SSSE3</strong></td>
<td>Defined as 1 for processors</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td></td>
<td>that support SSSE3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>instructions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>__unix</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>unix</strong></td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>unix</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>USER_LABEL_PREFIX</strong></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>VERSION</strong></td>
<td>Intel version string</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__WCHAR_T</td>
<td>1</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td><strong>WCHAR_TYPE</strong></td>
<td>long int on IA-32</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>architecture</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Macro Names

<table>
<thead>
<tr>
<th>Macro Name</th>
<th>Value</th>
<th>i32</th>
<th>i64em</th>
<th>i64</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>WINT_TYPE</strong></td>
<td>unsigned int</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>__x86_64</td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td><strong>x86_64</strong></td>
<td>1</td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>

### See Also

- `-D` compiler option
- `-U` compiler option

### ANSI Standard Predefined Macros

### Compiler Options

### Overview: Compiler Options

This document provides details on all current Linux*, Mac OS* X, and Windows* compiler options. It provides the following information:

- **New options**
  
  This topic lists new compiler options in this release.

- **Deprecated**

  This topic lists deprecated and removed compiler options for this release. Some deprecated options show suggested replacement options.

- **Alphabetical Compiler Options**

  This topic is the main source in the documentation set for general information on all compiler options. Options are described in alphabetical order. The **Overview** describes what information appears in each compiler option description.

- **Quick Reference Guide and Cross Reference**

  This topic contains a table summarizing compiler options. The table shows
the option name, a short description of the option, the default setting for the option, and the equivalent option on the alternate operating system.

- **Related Options**
  
  This topic lists related options that can be used under certain conditions.

In this guide, compiler options are available on all supported operating systems and architectures unless otherwise identified.

For information on compiler options that are equivalent to gcc options, see **Equivalent Options**.

For further information on compiler options, see Building Applications and Optimizing Applications.

**Functional Groupings of Compiler Options**

To see functional groupings of compiler options, specify a functional category for option `help` on the command line. For example, to see a list of options that affect diagnostic messages displayed by the compiler, enter one of the following commands:

- `-help diagnostics`  ! Linux and Mac OS X systems
- `/help diagnostics`  ! Windows systems

For details on the categories you can specify, see `help`.

**New Options**

This topic lists the options that provide new functionality in this release.

Some compiler options are only available on certain systems, as indicated by these labels:

<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td>The option is available on systems using IA-32 architecture.</td>
</tr>
<tr>
<td>i64em</td>
<td>The option is available on systems using Intel® 64 architecture.</td>
</tr>
<tr>
<td>i64</td>
<td>The option is available on systems using IA-64 architecture.</td>
</tr>
</tbody>
</table>

If no label appears, the option is available on all supported systems.

If "only" appears in the label, the option is only available on the identified system.
For more details on the options, refer to the Alphabetical Compiler Options section.

For information on conventions used in this table, see Conventions.

New compiler options are listed in tables below:

- **The first table lists new options that are available on Windows* systems.**
- **The second table lists new options that are available on Linux* and Mac OS* X systems. If an option is only available on one of these operating systems, it is labeled.**

<table>
<thead>
<tr>
<th>Windows* Options</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>/arch:IA32 (i32 only)</td>
<td>Generates code that will run on any Pentium or later processor.</td>
<td>OFF</td>
</tr>
<tr>
<td>/arch:SSE3 (i32, i64em)</td>
<td>Optimizes for Intel® Streaming SIMD Extensions 3 (Intel® SSE3).</td>
<td>OFF</td>
</tr>
<tr>
<td>/arch:SSSE3 (i32, i64em)</td>
<td>Optimizes for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3).</td>
<td>OFF</td>
</tr>
<tr>
<td>/arch:SSE4.1 (i32, i64em)</td>
<td>Optimizes for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.</td>
<td>OFF</td>
</tr>
<tr>
<td>/bigobj</td>
<td>Increases the</td>
<td>OFF</td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>/homeparams</td>
<td>Tells the compiler to store parameters passed in registers to the stack.</td>
<td>OFF</td>
</tr>
<tr>
<td>/MP</td>
<td>Creates multiple processes that can be used to compile large numbers of source files at the same time.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QaxSSE2</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2.</td>
<td>OFF</td>
</tr>
<tr>
<td>/QaxSSE3</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for</td>
<td>OFF</td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture.</td>
<td>/QaxSSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. OFF</td>
</tr>
<tr>
<td>/QaxSSE4.1 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation OFF</td>
<td></td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>/QaxSSE4.2 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qdiag-error-limit:n</td>
<td>Specifies the maximum number of errors allowed before compilation stops.</td>
<td>n=30</td>
</tr>
<tr>
<td>/Qdiag-once:id[,id,...]</td>
<td>Tells the compiler to issue one or more diagnostic messages only once.</td>
<td>OFF</td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>/Qfast-transcendentals</td>
<td>Enables the compiler to replace calls to transcendental functions with faster but less precise implementations.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qfma</td>
<td>Enables the combining of floating-point multiplies and add/subtract operations.</td>
<td>ON</td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qfp-relaxed</td>
<td>Enables use of faster but slightly less accurate code sequences for math functions.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qfreestanding</td>
<td>Ensures that compilation takes place in a freestanding environment.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qhelp-pragma</td>
<td>Displays all supported pragmas.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qinstruction:[no]movbe</td>
<td>Determines whether MOVBE instructions are generated for Intel processors.</td>
<td>ON</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>/Qopenmp-link:library</td>
<td>Controls whether the compiler links to static or dynamic OpenMP run-time libraries.</td>
<td>/Qopenmp-link:dynamic</td>
</tr>
<tr>
<td>/Qopenmp-task:model</td>
<td>Lets you choose an OpenMP* tasking model.</td>
<td>/Qopenmp-task:omp</td>
</tr>
<tr>
<td>/Qopenmp-threadprivate:type</td>
<td>Lets you specify an OpenMP* threadprivate implementation.</td>
<td>/Qopenmp-threadprivate:legacy</td>
</tr>
<tr>
<td>/Qopt-block-factor:n</td>
<td>Lets you specify a loop blocking factor.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qopt-jump-tables:keyword</td>
<td>Enables or disables generation of jump tables for switch statements.</td>
<td>/Qopt-jump-tables:default</td>
</tr>
<tr>
<td>/Qopt-loadpair</td>
<td>Enables loadpair optimization.</td>
<td>/Qopt-loadpair</td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qopt-mod-versioning</td>
<td>Enables versioning of modulo operations for certain types of operands.</td>
<td>/Qopt-mod-versioning</td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qopt-prefetch-initial-values</td>
<td>Enables or disables prefetches that are issued before a loop</td>
<td>/Qopt-prefetch-initial-values</td>
</tr>
<tr>
<td>(i64 only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>/Qopt-prefetch-issue-excl-hint (i64 only)</td>
<td>Determines whether the compiler issues prefetches for stores with exclusive hint.</td>
<td></td>
</tr>
<tr>
<td>/Qopt-prefetch-next-iteration (i64 only)</td>
<td>Enables or disables prefetches for a memory access in the next iteration of a loop.</td>
<td></td>
</tr>
<tr>
<td>/Qopt-subscript-in-range (i32, i64em)</td>
<td>Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.</td>
<td></td>
</tr>
<tr>
<td>/Qprof-data-order</td>
<td>Enables or disables data ordering if profiling information is enabled.</td>
<td></td>
</tr>
<tr>
<td>/Qprof-func-order</td>
<td>Enables or disables function ordering if profiling information is enabled.</td>
<td></td>
</tr>
<tr>
<td>/Qprof-hotness</td>
<td>Lets you set the</td>
<td>OFF</td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>threshold</td>
<td>hotness threshold for function grouping and function ordering.</td>
<td></td>
</tr>
<tr>
<td>/Qprof-src-dir</td>
<td>Determines whether directory information of the source file under compilation is considered when looking up profile data records.</td>
<td>/Qprof-src-dir</td>
</tr>
<tr>
<td>/Qprof-src-root</td>
<td>Lets you use relative directory paths when looking up profile data and specifies a directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qprof-src-root-cwd</td>
<td>Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td>/Qtcollect-filter</td>
<td>Lets you enable or disable the instrumentation of specified functions.</td>
<td>OFF</td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>/Quse-msasm-symbols</td>
<td>Tells the compiler to use a dollar sign (&quot;$&quot;) when producing symbol names.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qvc9</td>
<td>Specifies compatibility with Microsoft* Visual Studio 2008.</td>
<td>varies</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/Qvec</td>
<td>Enables or disables vectorization and transformations enabled for vectorization.</td>
<td>/Qvec</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxHost</td>
<td>Can generate specialized code paths for the highest instruction set and processor available on the compilation host.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxSSE2</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M</td>
<td>ON</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>/QxSSE3</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors, and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxSSE3_ATOM</td>
<td>Can generate MOVBE instructions for Intel processors and it can optimize for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxSSSE3</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>----------</td>
</tr>
<tr>
<td>/QxSSE4.1</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>/QxSSE4.2</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Windows* Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
<td></td>
</tr>
<tr>
<td>/Werror-all</td>
<td>Changes all warnings and remarks to errors.</td>
<td>OFF</td>
</tr>
</tbody>
</table>

## Linux* and Mac OS* X Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>-axSSE2</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-axSSE3</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors, and</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>it can optimize for processors based on Intel® Core microarchitecture and Intel NetBurst® microarchitecture.</td>
<td></td>
</tr>
<tr>
<td>-axSSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family.</td>
<td>OFF</td>
</tr>
<tr>
<td>-axSSE4.1 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>-axSSE4.2 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
<td>OFF</td>
</tr>
<tr>
<td>-diag-error-limit n</td>
<td>Specifies the maximum number of errors allowed before compilation stops.</td>
<td>n=30</td>
</tr>
<tr>
<td>-diag-once id[,id,...]</td>
<td>Tells the compiler to issue one or more diagnostic messages only once.</td>
<td>OFF</td>
</tr>
<tr>
<td>-falign-stack (i32 only)</td>
<td>Tells the compiler the stack alignment to use on entry to routines.</td>
<td>-falign-stack=default</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>-fasm-blocks</td>
<td>Enables the use of blocks and entire functions of assembly code within a C or C++ file.</td>
<td>OFF</td>
</tr>
<tr>
<td>(i32, i64em; Mac OS X only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-fast-transcendentals</td>
<td>Enables the compiler to replace calls to transcendental functions with faster but less precise implementation.</td>
<td>OFF</td>
</tr>
<tr>
<td>-ffreestanding</td>
<td>Ensures that compilation takes place in a freestanding environment.</td>
<td>OFF</td>
</tr>
<tr>
<td>-fma</td>
<td>Enables the combining of floating-point multiplies and add/subtract operations.</td>
<td>ON</td>
</tr>
<tr>
<td>(i64 only; Linux only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fnon-call-exceptions</td>
<td>Allows trapping instructions to throw C++ exceptions.</td>
<td>-fno-non-call-exceptions</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-fp-relaxed</td>
<td>Enables use of faster but slightly less accurate code</td>
<td>OFF</td>
</tr>
<tr>
<td>(i64 only; Linux only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>sequences for math functions.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-fpie (Linux only)</td>
<td>Tells the compiler to generate position-independent code to link into executables.</td>
<td>OFF</td>
</tr>
<tr>
<td>-fstack-protector (i32, i64em)</td>
<td>Determines whether the compiler generates code that detects some buffer overruns. Same as option -fstack-security-check.</td>
<td>-fno-stack-protector</td>
</tr>
<tr>
<td>-help-pragma (i32, i64em)</td>
<td>Displays all supported pragmas.</td>
<td>OFF</td>
</tr>
<tr>
<td>-m32, -m64 (i32, i64em)</td>
<td>Tells the compiler to generate code for a specific architecture.</td>
<td>OFF</td>
</tr>
<tr>
<td>-mia32 (i32 only)</td>
<td>Generates code that will run on any Pentium or later processor.</td>
<td>OFF</td>
</tr>
<tr>
<td>minstruction=[no]movbe (i32, i64em)</td>
<td>Determines whether MOVBE instructions are generated for Intel processors.</td>
<td>ON</td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>-mssse3 ( (i32, i64em) )</td>
<td>Generates code for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3).</td>
<td>Linux systems: OFF Mac OS X systems using Intel® 64 architecture: ON</td>
</tr>
<tr>
<td>-msse4.1 ( (i32, i64em) )</td>
<td>Generates code for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.</td>
<td>OFF</td>
</tr>
<tr>
<td>-multiple-processes</td>
<td>Creates multiple processes that can be used to compile large numbers of source files at the same time.</td>
<td>OFF</td>
</tr>
<tr>
<td>-openmp-link library</td>
<td>Controls whether the compiler links to static or dynamic OpenMP run-time libraries.</td>
<td>-openmp-link dynamic</td>
</tr>
<tr>
<td>-openmp-task model</td>
<td>Lets you choose an OpenMP* tasking model.</td>
<td>-openmp-task omp</td>
</tr>
<tr>
<td>-openmp-threadprivate=type ( \text{Linux only} )</td>
<td>Lets you specify an OpenMP* threadprivate implementation.</td>
<td>-openmp-threadprivate=legacy</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>-opt-block-factor=n</td>
<td>Lets you specify a loop blocking factor.</td>
<td>OFF</td>
</tr>
<tr>
<td>-opt-calloc</td>
<td>Tells the compiler to substitute a call to _intel_fast_calloc() for a call to calloc().</td>
<td>OFF</td>
</tr>
<tr>
<td>-opt-jump-tables=keyword</td>
<td>Enables or disables generation of jump tables for switch statements.</td>
<td>-opt-jump-tables=default</td>
</tr>
<tr>
<td>-opt-loadpair</td>
<td>Enables loadpair optimization.</td>
<td>-no-opt-loadpair</td>
</tr>
<tr>
<td>-opt-mod-versioning</td>
<td>Enables versioning of modulo operations for certain types of operands.</td>
<td>-no-opt-mod-versioning</td>
</tr>
<tr>
<td>-opt-prefetch-initial-values</td>
<td>Enables or disables prefetches that are issued before a loop is entered.</td>
<td>-opt-prefetch-initial-values</td>
</tr>
<tr>
<td>-opt-prefetch-issue-excl-hint</td>
<td>Determines whether the compiler issues prefetches for stores with exclusive hint.</td>
<td>-no-opt-prefetch-issue-excl-hint</td>
</tr>
<tr>
<td>-opt-prefetch-next-iteration</td>
<td>Enables or disables prefetches for a</td>
<td>-opt-prefetch-next-iteration</td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>(i64 only; Linux only)</td>
<td>memory access in the next iteration of a loop.</td>
<td></td>
</tr>
<tr>
<td>-opt-subscript-in-range</td>
<td>Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.</td>
<td>-no-opt-subscript-in-range</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td>no overflows in the intermediate computation of subscript expressions in loops.</td>
<td></td>
</tr>
<tr>
<td>-pie</td>
<td>Produces a position-independent executable on processors that support it.</td>
<td>OFF</td>
</tr>
<tr>
<td>(Linux only)</td>
<td>(Linux only)</td>
<td></td>
</tr>
<tr>
<td>-prof-data-order</td>
<td>Enables or disables data ordering if profiling information is enabled.</td>
<td>-no-prof-data-order</td>
</tr>
<tr>
<td>(Linux only)</td>
<td>(Linux only)</td>
<td></td>
</tr>
<tr>
<td>-prof-func-groups</td>
<td>Enables or disables function grouping if profiling information is enabled.</td>
<td>-no-prof-func-groups</td>
</tr>
<tr>
<td>(i32, i64em; Linux only)</td>
<td>(i32, i64em; Linux only)</td>
<td></td>
</tr>
<tr>
<td>-prof-func-order</td>
<td>Enables or disables function ordering if profiling information is enabled.</td>
<td>-no-prof-func-order</td>
</tr>
<tr>
<td>(Linux only)</td>
<td>(Linux only)</td>
<td></td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><code>-prof-hotness-threshold</code></td>
<td>Lets you set the hotness threshold for function grouping and function ordering.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-prof-src-root</code></td>
<td>Lets you use relative directory paths when looking up profile data and specifies a directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-prof-src-root-cwd</code></td>
<td>Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-staticlib</code></td>
<td>Invokes the libtool command to generate static libraries.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-tcollect-filter</code></td>
<td>Lets you enable or disable the instrumentation of specified functions.</td>
<td>OFF</td>
</tr>
<tr>
<td><code>-vec</code></td>
<td>Enables or disables vectorization and transformations enabled for vectorization.</td>
<td>-vec</td>
</tr>
<tr>
<td>Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>-Werror-all</td>
<td>Changes all warnings and remarks to errors.</td>
<td>OFF</td>
</tr>
<tr>
<td>-Wformat-security</td>
<td>Issues a warning when the use of format functions may cause security problems.</td>
<td>-Wno-format-security</td>
</tr>
<tr>
<td>-xHost (i32, i64em)</td>
<td>Can generate instructions for the highest instruction set and processor available on the compilation host.</td>
<td>OFF</td>
</tr>
<tr>
<td>-xSSE2 (i32, i64em)</td>
<td>Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2.</td>
<td>Linux systems:ON Mac OS X systems: OFF</td>
</tr>
<tr>
<td>-xSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSE3, SSE2, and SSE instructions for Intel® processors with Intel® SSE2.</td>
<td>Linux systems:OFF Mac OS X systems using IA-32 architecture: ON</td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td>Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture.</td>
<td></td>
</tr>
<tr>
<td>-xSSE3_ATOM (i32, i64em)</td>
<td>Can generate MOVBE OFF instructions for Intel processors and it can optimize for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology.</td>
<td>OFF</td>
</tr>
<tr>
<td>-xSSE3 (i32, i64em)</td>
<td>Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family.</td>
<td>Linux systems: OFF Mac OS X systems using Intel® 64 architecture: ON</td>
</tr>
<tr>
<td>-xSSE4.1 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel</td>
<td>OFF</td>
</tr>
<tr>
<td><em><em>Linux</em> and Mac OS</em> X Options**</td>
<td><strong>Description</strong></td>
<td><strong>Default</strong></td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-xSSE4.2 (i32, i64em)</td>
<td>Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.</td>
<td>OFF</td>
</tr>
</tbody>
</table>

**Deprecated and Removed Compiler Options**

This topic lists deprecated and removed compiler options and suggests replacement options, if any are available.
**Deprecated Options**

Occasionally, compiler options are marked as "deprecated." Deprecated options are still supported in the current release, but are planned to be unsupported in future releases.

The following options are deprecated in this release of the compiler:

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X Options Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-A-</td>
</tr>
<tr>
<td>-alias-args</td>
</tr>
<tr>
<td>-axK</td>
</tr>
<tr>
<td>-axN</td>
</tr>
<tr>
<td>-axP</td>
</tr>
<tr>
<td>-axS</td>
</tr>
<tr>
<td>-axT</td>
</tr>
<tr>
<td>-axW</td>
</tr>
<tr>
<td>-c99</td>
</tr>
<tr>
<td>-create-pch</td>
</tr>
<tr>
<td>-func-groups</td>
</tr>
<tr>
<td>-fwritable-strings</td>
</tr>
<tr>
<td>-i-dynamic</td>
</tr>
<tr>
<td>-i-static</td>
</tr>
<tr>
<td>-inline-debug-info</td>
</tr>
<tr>
<td>-IPF-flt-eval-method0</td>
</tr>
<tr>
<td>-IPF-fltacc</td>
</tr>
<tr>
<td>-no-IPF-fltacc</td>
</tr>
<tr>
<td>-IPF-fma</td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options Suggested Replacement</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>-IPF-fp-relaxed</td>
</tr>
<tr>
<td>-Kc++</td>
</tr>
<tr>
<td>-march=pentiumii</td>
</tr>
<tr>
<td>-march=pentiumiii</td>
</tr>
<tr>
<td>-mcpu</td>
</tr>
<tr>
<td>-mp</td>
</tr>
<tr>
<td>-Ob</td>
</tr>
<tr>
<td>-openmp-lib legacy</td>
</tr>
<tr>
<td>-openmpP</td>
</tr>
<tr>
<td>-openmpS</td>
</tr>
<tr>
<td>-prefetch</td>
</tr>
<tr>
<td>-prof-genx</td>
</tr>
<tr>
<td>-use-asm</td>
</tr>
<tr>
<td>-use-pch</td>
</tr>
<tr>
<td>-wd</td>
</tr>
<tr>
<td>-we</td>
</tr>
<tr>
<td>-wn</td>
</tr>
<tr>
<td>-wo</td>
</tr>
<tr>
<td>-wr</td>
</tr>
<tr>
<td>-ww</td>
</tr>
<tr>
<td>-xK</td>
</tr>
<tr>
<td>-xN</td>
</tr>
<tr>
<td>-xO</td>
</tr>
</tbody>
</table>
## Linux* and Mac OS* X Options Suggested Replacement

<table>
<thead>
<tr>
<th>Option</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-xP</code></td>
<td><code>-xSSE3</code></td>
</tr>
<tr>
<td><code>-xS</code></td>
<td><code>-xSSE4.1</code></td>
</tr>
<tr>
<td><code>-xT</code></td>
<td><code>-xSSE3</code></td>
</tr>
<tr>
<td><code>-xW</code></td>
<td><code>-msse2</code></td>
</tr>
</tbody>
</table>

## Windows* Options Suggested Replacement

<table>
<thead>
<tr>
<th>Option</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/Fm</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/Fr</code></td>
<td><code>/FR</code></td>
</tr>
<tr>
<td><code>/G5</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/G6</code> (or <code>/GB</code>)</td>
<td>None</td>
</tr>
<tr>
<td><code>/G7</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/Ge</code></td>
<td><code>/Gs0</code></td>
</tr>
<tr>
<td><code>/Gf</code></td>
<td><code>/GF</code></td>
</tr>
<tr>
<td><code>/GX</code></td>
<td><code>/EHs</code></td>
</tr>
<tr>
<td><code>/Gy</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/GZ</code></td>
<td><code>/RTC$s</code></td>
</tr>
<tr>
<td><code>/H</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/ML</code> and <code>/MLd</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/Op</code></td>
<td><code>/fp</code></td>
</tr>
<tr>
<td><code>/QA-</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/QaxK</code></td>
<td>None</td>
</tr>
<tr>
<td><code>/QaxN</code></td>
<td><code>/QaxSSE2</code></td>
</tr>
<tr>
<td><code>/QaxP</code></td>
<td><code>/QaxSSE3</code></td>
</tr>
<tr>
<td>Windows* Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>/QaxS</td>
<td>/QaxSSE4.1</td>
</tr>
<tr>
<td>/QaxT</td>
<td>/QaxSSSE3</td>
</tr>
<tr>
<td>/QaxW</td>
<td>/arch:SSE2</td>
</tr>
<tr>
<td>/Qc99</td>
<td>/Qstd=c99</td>
</tr>
<tr>
<td>/Qinline-debug-info</td>
<td>None</td>
</tr>
<tr>
<td>/QIPF-flt-eval-method0</td>
<td>/fp:source</td>
</tr>
<tr>
<td>/QIPF-fltacc</td>
<td>/fp:precise</td>
</tr>
<tr>
<td>/QIPF-fltacc-</td>
<td>/fp:fast</td>
</tr>
<tr>
<td>/QIPF-fma</td>
<td>/Qfma</td>
</tr>
<tr>
<td>/QIPF-fp-relaxed</td>
<td>/Qfp-relaxed</td>
</tr>
<tr>
<td>/Qmspp</td>
<td>None</td>
</tr>
<tr>
<td>/Qopenmp-lib:legacy</td>
<td>None</td>
</tr>
<tr>
<td>/Qprefetch</td>
<td>/Qopt-prefetch</td>
</tr>
<tr>
<td>/Qprof-genx</td>
<td>/Qprof-gen=srcpos</td>
</tr>
<tr>
<td>/Quse-asm</td>
<td>None</td>
</tr>
<tr>
<td>/Quse-vcdebug</td>
<td>None</td>
</tr>
<tr>
<td>/Qwd</td>
<td>/Qdiag-disable</td>
</tr>
<tr>
<td>/Qwe</td>
<td>/Qdiag-error</td>
</tr>
<tr>
<td>/Qwn</td>
<td>/Qdiag-error-limit</td>
</tr>
<tr>
<td>/Qwo</td>
<td>/Qdiag-once id[,id,...]</td>
</tr>
<tr>
<td>/Qwr</td>
<td>/Qdiag-remark</td>
</tr>
<tr>
<td>/Qww</td>
<td>/Qdiag-warning</td>
</tr>
<tr>
<td>/QxK</td>
<td>None</td>
</tr>
</tbody>
</table>
Windows* Options  |  Suggested Replacement
---|---
/QxN | /QxSSE2
/QxO | /arch:SSE3
/QxP | /QxSSE3
/QxS | /QxSSE4.1
/QxT | /QxSSSE3
/QxW | /arch:SSE2
/Zd | /debug:minimal
/Ze | None

Deprecated options are not limited to this list.

Removed Options

Some compiler options are no longer supported and have been removed. If you use one of these options, the compiler issues a warning, ignores the option, and then proceeds with compilation.

This version of the compiler no longer supports the following compiler options:

Linux* and Mac OS* X Options  |  Suggested Replacement
---|---
-0f_check | None
-axB | -axSSE2
-axi | None
-axM | None
-cxxlib-gcc[=dir] | -cxxlib[=dir]
-cxxlib-icc | None
-F | -P
-fdiv_check | None
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fp</td>
<td>-fno-omit-frame-pointer</td>
</tr>
<tr>
<td>-fpstkchk</td>
<td>-fp-stack-check</td>
</tr>
<tr>
<td>-IPF-fp-speculation</td>
<td>-fp-speculation</td>
</tr>
<tr>
<td>-ipo-obj (and -ipo_obj)</td>
<td>None</td>
</tr>
<tr>
<td>-Knopic, -KNOPIC</td>
<td>-fpic</td>
</tr>
<tr>
<td>-Kpic, -KPIC</td>
<td>-fpic</td>
</tr>
<tr>
<td>-mtune=itanium</td>
<td>None</td>
</tr>
<tr>
<td>-no-c99</td>
<td>-std=c89</td>
</tr>
<tr>
<td>-no-cpprt</td>
<td>-no-cxxlib</td>
</tr>
<tr>
<td>-nobss-init</td>
<td>-no-bss-init</td>
</tr>
<tr>
<td>-norestrict</td>
<td>-no-restrict</td>
</tr>
<tr>
<td>-opt-report-level</td>
<td>-opt-report</td>
</tr>
<tr>
<td>-prof-format-32</td>
<td>None</td>
</tr>
<tr>
<td>-prof-gen-sampling</td>
<td>None</td>
</tr>
<tr>
<td>-qp</td>
<td>-p</td>
</tr>
<tr>
<td>-shared-libcxx</td>
<td>-shared-libgcc</td>
</tr>
<tr>
<td>-ssp</td>
<td>None</td>
</tr>
<tr>
<td>-static-libcxx</td>
<td>-static-libgcc</td>
</tr>
<tr>
<td>-syntax</td>
<td>-fsyntax-only</td>
</tr>
<tr>
<td>-tpp1</td>
<td>None</td>
</tr>
<tr>
<td>-tpp2</td>
<td>-mtune=itanium2</td>
</tr>
<tr>
<td>-tpp5</td>
<td>None</td>
</tr>
<tr>
<td>-tpp6</td>
<td>None</td>
</tr>
<tr>
<td>Linux* and Mac OS* X Options</td>
<td>Suggested Replacement</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><code>-tpp7</code></td>
<td><code>-mtune=pentium4</code></td>
</tr>
<tr>
<td><code>-use-pch</code></td>
<td><code>-pch-use</code></td>
</tr>
<tr>
<td><code>-xB</code></td>
<td><code>-xSSE2</code></td>
</tr>
<tr>
<td><code>-xi</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>-xM</code></td>
<td><code>None</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Windows* Options</th>
<th>Suggested Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/QaxB</code></td>
<td><code>/QaxSSE2</code></td>
</tr>
<tr>
<td><code>/Qaxi</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/QaxM</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/Qfpstkchk</code></td>
<td><code>/Qfp-stack-check</code></td>
</tr>
<tr>
<td><code>/QIfdiv</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/QIPF-fp-speculation</code></td>
<td><code>/Qfp-speculation</code></td>
</tr>
<tr>
<td><code>/Qipo-obj (and /Qipo_obj)</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/QI0f</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/Qopt-report-level</code></td>
<td><code>/Qopt-report</code></td>
</tr>
<tr>
<td><code>/Qprof-format-32</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/Qprof-gen-sampling</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/Qssp</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/Qvc6</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/Qvc7</code></td>
<td><code>None</code></td>
</tr>
<tr>
<td><code>/QxB</code></td>
<td><code>/QxSSE2</code></td>
</tr>
<tr>
<td><code>/Qxi</code></td>
<td><code>None</code></td>
</tr>
</tbody>
</table>
Windows* Options | Suggested Replacement
---|---
/Qxm | None

Removed options are not limited to these lists.

Alphabetical Compiler Options

Compiler Option Descriptions and General Rules

This section describes all the current Linux*, Mac OS* X, and Windows* compiler options in alphabetical order.

Option Descriptions

Each option description contains the following information:

- A short description of the option.
- IDE Equivalent
  This shows information related to the integrated development environment (IDE) Property Pages on Windows, Linux, and Mac OS X systems. It shows on which Property Page the option appears, and under what category it's listed. The Windows IDE is Microsoft* Visual Studio* .NET; the Linux IDE is Eclipse*; the Mac OS X IDE is Xcode*. If the option has no IDE equivalent, it will specify "None".
- Architectures
  This shows the architectures where the option is valid. Possible architectures are:
  - IA-32 architecture
  - Intel® 64 architecture
  - IA-64 architecture
- Syntax
  This shows the syntax on Linux and Mac OS X systems and the syntax on Windows systems. If the option has no syntax on one of these systems, that is, the option is not valid on a particular system, it will specify "None".
• Arguments
  This shows any arguments (parameters) that are related to the option. If the
  option has no arguments, it will specify "None".
• Default
  This shows the default setting for the option.
• Description
  This shows the full description of the option. It may also include further
  information on any applicable arguments.
• Alternate Options
  These are options that are synonyms of the described option. If there are no
  alternate options, it will specify "None".
  Many options have an older spelling where underscores ("_") instead of
  hyphens ("-") connect the main option names. The older spelling is a valid
  alternate option name.

Some option descriptions may also have the following:
• Example
  This shows a short example that includes the option
• See Also
  This shows where you can get further information on the option or related
  options.

General Rules for Compiler Options

You cannot combine options with a single dash (Linux and Mac OS X) or slash
(Windows). For example:
• On Linux and Mac OS X systems: This is incorrect: \texttt{-wc}; this is correct: \texttt{-w -c}
• On Windows systems: This is incorrect: \texttt{/wc}; this is correct: \texttt{/w /c}

All compiler options are case sensitive. Some options have different meanings
depending on their case; for example, option "c" prevents linking, but option "C"
places comments in preprocessed source output.
Options specified on the command line apply to all files named on the command line.
Options can take arguments in the form of file names, strings, letters, or numbers. If a string includes spaces, the string must be enclosed in quotation marks. For example:

- On Linux and Mac OS X systems, `-dynamic-linker mylink (file name)` or `-Umacro3 (string)`
- On Windows systems, `/Famyfile.s (file name)` or `/V"version 5.0" (string)`

Compiler options can appear in any order.
On Windows systems, all compiler options must precede `/link` options, if any, on the command line.
Unless you specify certain options, the command line will both compile and link the files you specify.
You can abbreviate some option names, entering as many characters as are needed to uniquely identify the option.
Certain options accept one or more keyword arguments following the option name. For example, the `arch` option accepts several keywords.
To specify multiple keywords, you typically specify the option multiple times. However, there are exceptions; for example, the following are valid: `-axNB (Linux)` or `/QaxNB (Windows)`.

Compiler options remain in effect for the whole compilation unless overridden by a compiler `#pragma`.
To disable an option, specify the negative form of the option.
On Windows systems, you can also disable one or more options by specifying option `/Od last on the command line.

⚠️ Note

On Windows systems, the `/Od` option is part of a mutually-exclusive group of options that includes `/Od, /O1, /O2, /O3, and /Ox. The last of any of these options specified on the command line will override the previous options from this group.
If there are enabling and disabling versions of an option on the command line, the last one on the command line takes precedence.

**Lists and Functional Groupings of Compiler Options**

To see a list of all the compiler options, specify option `help` on the command line.

To see functional groupings of compiler options, specify a functional category for option `help`. For example, to see a list of options that affect diagnostic messages displayed by the compiler, enter one of the following commands:

- `help diagnostics` ! Linux and Mac OS X systems
- `/help diagnostics` ! Windows systems

For details on the categories you can specify, see `help`.

**A, QA**

Specifies an identifier for an assertion.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `Aname[(value)]`

Windows: `/QAnam{(value)}`

**Arguments**

name

Is the identifier for the assertion.
value

Is an optional value for the assertion. If a value is specified, it must be within quotes, including the parentheses delimiting it.

Default

OFF

Assertions have no identifiers or symbol names.

Description

This option specifies an identifier (symbol name) for an assertion. It is equivalent to an #assert preprocessing directive. Note that this option is not the positive form of the C++ /QA- option.

Alternate Options

None

Example
To make an assertion for the identifier fruit with the associated values orange and banana use the following command.

On Windows* systems:

```
icl /QA"fruit(orange,banana)" prog1.cpp
```

On Linux* and Mac OS* X systems:

```
icpc -A"fruit(orange,banana)" prog1.cpp
```

**A-, QA-**

Disables all predefined macros. This is a deprecated option.

**IDE Equivalent**

Windows: None

Linux: **Preprocessor > Undefine All Preprocessor Definitions**

Mac OS X: **Preprocessor > Undefine All Preprocessor Definitions**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-A-`

Windows: `/QA-`

**Arguments**

None

**Default**

**OFF**

Predefined macros remain enabled.

**Description**

152
This option disables all predefined macros. It causes all predefined macros and assertions to become inactive.

Note that this option is not the negative form of the C++ /QA option.

Alternate Options

None

fargument-alias, Qalias-args

Determines whether function arguments can alias each other.

IDE Equivalent

Windows: None
Linux: Data > Enable Argument Aliasing
Mac OS X: Data > Enable Argument Aliasing

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fargument-alias
- fargument-noalias

Windows: /Qalias-args
/Qalias-args-

Arguments

None

Default

-fargument-alias or /Qalias-args

Function
arguments
can alias each other and can alias global storage.

Description

This option determines whether function arguments can alias each other. If you specify `-fargument-noalias` or `/Qalias-args-`, function arguments cannot alias each other, but they can alias global storage. On Linux and Mac OS X systems, you can also disable aliasing for global storage, by specifying option `-fargument-noalias-global`.

Alternate Options

Linux and Mac OS X: `-no-alias-args` (this is a deprecated option)
Windows: None

See Also

`fargument-noalias-global` compiler option

`alias-const`, `Qalias-const`

Determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const.

IDE Equivalent

Windows: None
Linux: **Data > Assume Restrict Semantics for Const**
Mac OS X: **Data > Assume Restrict Semantics for Const**
IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**  `-alias-const`  
  `-no-alias-const`

**Windows:**  `/Qalias-const`
  `/Qalias-const-`

Arguments

None

Default

`-no-alias-const`  
*or*  `/Qalias-const-`

The compiler uses standard C/C++ rules for the interpretation of const.

Description

This option determines whether the compiler assumes a parameter of type `pointer-to-const` does not alias with a parameter of type `pointer-to-non-const`. It implies an additional attribute for const.

This functionality complies with the input/output buffer rule, which assumes that input and output buffer arguments do not overlap. This option allows the compiler to do some additional optimizations with those parameters.

In C99, you can also get the same result if you additionally declare your pointer parameters with the restrict keyword.

Alternate Options
None

**align**

Determines whether variables and arrays are naturally aligned.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X: `-align
- noalign

Windows: None

**Arguments**

None

**Default**

OFF

Variables and arrays are aligned according to the gcc model, which means they are aligned to 4-byte boundaries.
Description

This option determines whether variables and arrays are naturally aligned. Option `-align` forces the following natural alignment:

<table>
<thead>
<tr>
<th>Type</th>
<th>Alignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>double</td>
<td>8 bytes</td>
</tr>
<tr>
<td>long long</td>
<td>8 bytes</td>
</tr>
<tr>
<td>long double</td>
<td>16 bytes</td>
</tr>
</tbody>
</table>

If you are not interacting with system libraries or other libraries that are compiled without `-align`, this option can improve performance by reducing misaligned accesses.

⚠️ Caution

If you are interacting with system libraries or other libraries that are compiled without `-align`, your application may not perform as expected.

Alternate Options

None

ansi

Enables language compatibility with the gcc option `-ansi`.

IDE Equivalent

Windows: None
Linux: Language > ANSI Conformance
Mac OS X: Language > C ANSI Conformance

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -ansi
Windows: None

Arguments

None

Default

OFF

GNU C++ is more strongly supported than ANSI C.

Description

This option enables language compatibility with the gcc option -ansi and provides the same level of ANSI standard conformance as that option. This option sets option fmath-errno.

If you want strict ANSI conformance, use the -strict-ansi option.

Alternate Options

None

ansi-alias, Qansi-alias

Enable use of ANSI aliasing rules in optimizations.

IDE Equivalent

Windows: None
Linux: Language > Enable Use of ANSI Aliasing Rules in Optimizations
Mac OS X: **Language > Enable ANSI Aliasing**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `.ansi-alias`

`.no-ansi-alias`

Windows: `/ansi-alias`

`/ansi-alias-`

**Arguments**

None

**Default**

`.no-ansi-alias` or `/ansi-alias-`

**Description**

This option tells the compiler to assume that the program adheres to ISO C Standard aliasability rules.

If your program adheres to these rules, then this option allows the compiler to optimize more aggressively. If it doesn't adhere to these rules, then it can cause the compiler to generate incorrect code.

**Alternate Options**

None
Ap64

Enables 64-bit pointers.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux and Mac OS X: None
Windows: /Ap64

Arguments

None

Default

ON 64-bit pointers are enabled.

Description

This option enables 64-bit pointers.

Alternate Options

None

arch

Tells the compiler to generate optimized code specialized for the processor that executes your program.
IDE Equivalent

Windows: **Code Generation > Enable Enhanced Instruction Set**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: None
Windows: `/arch:processor`

Arguments

`processor` Is the processor type.
Possible values are:

**IA32** Generates code that will run on any Pentium or later processor. Disables any default extended instruction settings, and any previously set extended instruction...
settings. This value is only available on IA-32 architecture.

**SSE**
This is the same as specifying IA32.

**SSE2**
Generates code for Intel® Streaming SIMD Extensions 2 (Intel® SSE2).

**SSE3**
Generates code for Intel® Streaming SIMD Extensions 3 (Intel® SSE3).

**SSSE3**
Generates code for Intel® Supplemental
Streaming SIMD Extensions 3 (Intel® SSSE3).

SSE4.1 Generates code for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.

Default

SSE2

The compiler generates code for Intel® Streaming SIMD Extensions 2 (Intel® SSE2).

Description
This option tells the compiler to generate optimized code specialized for the processor that executes your program. Code generated with the values IA32, SSE, SSE2 or SSE3 should execute on any compatible non-Intel processor with support for the corresponding instruction set.

Alternate Options

Linux and Mac OS X: –m
Windows: None

See Also

x, Qx compiler option
ax, Qax compiler option
m compiler option

As

Determines the size of virtual address space.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux and Mac OS X: None
Windows: /As

Arguments

n
Is the virtual
The virtual address space is 16 exabytes.

**Default**

/As64

**Description**

This option determines the size of virtual address space.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/As32</td>
<td>Sets the virtual address space to 4 gigabytes (GB).</td>
</tr>
<tr>
<td>/As64</td>
<td>Sets the virtual address space to 16 exabytes (EB).</td>
</tr>
</tbody>
</table>

**Alternate Options**

None

**auto-ilp32, Qauto-ilp32**

Instructs the compiler to analyze the program to determine if there are 64-bit pointers which can be safely shrunk into 32-bit pointers.

**IDE Equivalent**
None

Architectures

Intel® 64 architecture, IA-64 architecture

Syntax

Linux and Mac OS X:  -auto-ilp32
Windows:           /Qauto-ilp32

Arguments

None

Default

OFF

The optimization is not attempted.

Description

This option instructs the compiler to analyze and transform the program so that 64-bit pointers are shrunk to 32-bit pointers, and 64-bit longs (on Linux) are shrunk into 32-bit longs wherever it is legal and safe to do so. In order for this option to be effective the compiler must be able to optimize using the -ipo/-Qipo option and must be able to analyze all library/external calls the program makes.

This option requires that the size of the program executable never exceeds $2^{32}$ bytes and all data values can be represented within 32 bits. If the program can run correctly in a 32-bit system, these requirements are implicitly satisfied. If the program violates these size restrictions, unpredictable behavior might occur.

Alternate Options
None

ax, Qax

Tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit.

IDE Equivalent

Windows: **Code Generation > Add Processor-Optimized Code Path**
Linux: **Code Generation > Add Processor-Optimized Code Path**
Mac OS X: **Code Generation > Add Processor-Optimized Code Path**

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: `-axprocessor`  
Windows: `/Qaxprocessor`

Arguments

`processor`  

Indicates the processor for which code is generated.  
The following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE).  
Possible values are:

- **SSE4.2**  
  Can generate Intel® SSE4
Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.

SSE4.1 Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3,
SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.

SSSE3 Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. This replaces value T, which is deprecated.

SSE3 Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors
and it can
optimize for
processors based
on Intel® Core™
microarchitecture
and Intel
NetBurst®
microarchitecture.
This replaces
value P, which is
deprecated.

SSE2
Can generate
Intel® SSE2 and
SSE instructions
for Intel
processors, and it
can optimize for
Intel® Pentium®
4 processors,
Intel® Pentium®
M processors,
and Intel® Xeon®
processors with
Intel® SSE2. This
value is not
available on Mac
OS X systems.
This replaces
value N, which is
deprecated.
Default

OFF

No auto-dispatch code is generated. Processor-specific code is generated and is controlled by the setting of compiler option –m (Linux), compiler option /arch (Windows), or compiler option –x (Mac OS* X).

Description

This option tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit. It also generates a baseline code path. The baseline code is usually slower than the specialized code.
The baseline code path is determined by the architecture specified by the `-x` (Linux and Mac OS X) or `/Qx` (Windows) option. While there are defaults for the `-x` or `/Qx` option that depend on the operating system being used, you can specify an architecture for the baseline code that is higher or lower than the default. The specified architecture becomes the effective minimum architecture for the baseline code path.

If you specify both the `-ax` and `-x` options (Linux and Mac OS X) or the `/Qax` and `/Qx` options (Windows), the baseline code will only execute on processors compatible with the processor type specified by the `-x` or `/Qx` option.

This option tells the compiler to find opportunities to generate separate versions of functions that take advantage of features of the specified Intel® processor. If the compiler finds such an opportunity, it first checks whether generating a processor-specific version of a function is likely to result in a performance gain. If this is the case, the compiler generates both a processor-specific version of a function and a baseline version of the function. At run time, one of the versions is chosen to execute, depending on the Intel processor in use. In this way, the program can benefit from performance gains on more advanced Intel processors, while still working properly on older processors.

You can use more than one of the processor values by combining them. For example, you can specify `-axSSE4.1,SSSE3` (Linux and Mac OS X) or `/QaxSSE4.1,SSSE3` (Windows). You cannot combine the old style, deprecated options and the new options. For example, you cannot specify `-axSSE4.1,T` (Linux and Mac OS X) or `/QaxSSE4.1,T` (Windows).

Previous values W and K are deprecated. The details on replacements are as follows:

- **Mac OS X systems**: On these systems, there is no exact replacement for W or K. You can upgrade to the default option `-msse3` (IA-32 architecture) or option `-mssse3` (Intel® 64 architecture).
- **Windows and Linux systems**: The replacement for W is `-msse2` (Linux) or `/arch:SSE2` (Windows). There is no exact replacement for K.
However, on Windows systems, `/QaxK` is interpreted as `/arch:IA32`; on Linux systems, `-axK` is interpreted as `-mia32`. You can also do one of the following:

- Upgrade to option `-msse2` (Linux) or option `/arch:SSE2` (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors.
- Specify the two option combination `-mia32` `-axSSE2` (Linux) or `/arch:IA32` `/QaxSSE2` (Windows). This combination will produce an executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path.

The `-ax` and `/Qax` options enable additional optimizations not enabled with option `-m` or option `/arch`.

Alternate Options

None

See Also

- `x, Qx` compiler option
- `m` compiler option
- `arch` compiler option

B

Specifies a directory that can be used to find include files, libraries, and executables.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:  \(-B\textit{dir}\)

Windows:  \(\text{None}\)

**Arguments**

\(\textit{dir}\)

Is the directory to be used. If necessary, the compiler adds a directory separator character at the end of \(\textit{dir}\).

**Default**

OFF

The compiler looks for files in the directories specified in your PATH environment variable.

**Description**
This option specifies a directory that can be used to find include files, libraries, and executables.

The compiler uses \texttt{dir} as a prefix.

For include files, the \texttt{dir} is converted to \texttt{-I/dir/include}. This command is added to the front of the includes passed to the preprocessor.

For libraries, the \texttt{dir} is converted to \texttt{-L/dir}. This command is added to the front of the standard \texttt{-L} inclusions before system libraries are added.

For executables, if \texttt{dir} contains the name of a tool, such as \texttt{ld} or \texttt{as}, the compiler will use it instead of those found in the default directories.

The compiler looks for include files in \texttt{dir/include} while library files are looked for in \texttt{dir}.

Another way to get the behavior of this option is to use the environment variable \texttt{GCC_EXEC_PREFIX}.

\begin{description}
\item[Alternate Options]
None
\item[Bdynamic]
Enables dynamic linking of libraries at run time.
\item[IDE Equivalent]
None
\item[Architectures]
IA-32, Intel® 64, IA-64 architectures
\item[Syntax]
\begin{description}
\item[Linux]: \texttt{-Bdynamic}
\item[Mac OS X]: None
\item[Windows]: None
\end{description}
\item[Arguments]
None

Default

OFF

Limited
dynamic
linking
occurs.

Description

This option enables dynamic linking of libraries at run time. Smaller executables are created than with static linking.

This option is placed in the linker command line corresponding to its location on the user command line. It controls the linking behavior of any library that is passed using the command line.

All libraries on the command line following option `-Bdynamic` are linked dynamically until the end of the command line or until a `-Bstatic` option is encountered. The `-Bstatic` option enables static linking of libraries.

Alternate Options

None

See Also

- static compiler option
- bigobj

bigobj

Increases the number of sections that an object file can contain.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: None
Windows: /bigobj

Arguments

None

Default

OFF

An object file can hold up to 65,536 (2**16) addressable sections.

Description

This option increases the number of sections that an object file can contain. It increases the address capacity to 4,294,967,296 (2**32).
An .obj file produced with this option can only be effectively passed to a linker that shipped in Microsoft Visual C++ 2005 or later. Linkers shipped with earlier versions of the product cannot read .obj files of this size.
This option may be helpful for .obj files that can hold more sections, such as machine generated code or code that makes heavy use of template libraries.

Alternate Options

None

Bstatic

Enables static linking of a user's library.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux: \(-B\text{static}\)
Mac OS X: None
Windows: None

Arguments
None

Default
OFF

Description
This option enables static linking of a user’s library.
This option is placed in the linker command line corresponding to its location on the user command line. It controls the linking behavior of any library that is passed using the command line.
All libraries on the command line following option \(-B\text{static}\) are linked statically until the end of the command line or until a \(-B\text{dynamic}\) option is encountered. The \(-B\text{dynamic}\) option enables dynamic linking of libraries.

Alternate Options
None
See Also

Bdynamic compiler option

c

Prevents linking.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -c

Windows: /c

Arguments

None

Default

OFF

Linking is performed.

Description

This option prevents linking. Compilation stops after the object file is generated. The compiler generates an object file for each C or C++ source file or preprocessed source file. It also takes an assembler file and invokes the assembler to generate an object file.

Alternate Options

None
C

Placed comments in preprocessed source output.

IDE Equivalent

Windows: **Preprocessor > Keep Comments**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-C`
Windows: `//C`

Arguments

None

Default

OFF

No comments are placed in preprocessed source output.

Description

This option places (or preserves) comments in preprocessed source output. Comments following preprocessing directives, however, are not preserved.

Alternate Options
None

See Also
Building Applications: About Preprocessor Options

c99, Qc99

Determines whether C99 support is enabled for C programs. This is a deprecated option.

IDE Equivalent
Windows: Language > Enable C99 Support
Linux: Language > Disable C99 Support
Mac OS X: Language > Disable C99 Support

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: \( -c99 \)
\( -\text{no-c99} \)
Windows: \( /Qc99 \)
\( /Qc99- \)

Arguments
None

Default
\( -\text{no-c99} \) \( \text{C99} \) support is
Description

This option determines whether C99 support is enabled for C programs. One of the features enabled by `–c99` (Linux and Mac OS X) or `/Qc99` (Windows), restricted pointers, is available by using option `restrict`. For more information, see `restrict`.

Alternate Options

- `-std`, `/Qstd`

See Also

`restrict`, `/Qrestrict` compiler option

check-uninit

Determines whether checking occurs for uninitialized variables.

IDE Equivalent

Windows: None

Linux: **Runtime > Check Uninitialized**

Mac OS X: **Runtime > Check Uninitialized**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `–check-uninit`

`–no-check-uninit`
Windows: None

Arguments
None

Default
-no-check-uninit

Description
Enables run-time checking for uninitialized variables. If a variable is read before it is written, a run-time error routine will be called. Run-time checking of undefined variables is only implemented on local, scalar variables. It is not implemented on dynamically allocated variables, extern variables or static variables. It is not implemented on structs, classes, unions or arrays.

Alternate Options
None

complex-limited-range, Qcomplex-limited-range
Determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled.

IDE Equivalent
Windows: Floating Point > Limit COMPLEX Range
Linux: None
Mac OS X: Floating Point > Limit COMPLEX Range

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:  
- complex-limited-range  
  - no-complex-limited-range  

Windows:  
/ Qcomplex-limited-range  
/ Qcomplex-limited-range-

Arguments
None

Default
- no-complex-limited-range  
or / Qcomplex-limited-range-

Basic algebraic expansions of some arithmetic operations involving data of type COMPLEX are disabled.

Description
This option determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled. When the option is enabled, this can cause performance improvements in programs that use a lot of COMPLEX arithmetic. However, values at the extremes of the exponent range may not compute correctly.

Alternate Options
None


cxxlib

Determines whether the compile links using the C++ run-time libraries and header files provided by gcc.

IDE Equivalent

Windows: None
Linux: Preprocessor > gcc Compatibility Options
Mac OS X: Preprocessor > gcc Compatibility Options

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -cxxlib[=dir]
-cxxlib-nostd
-no-cxxlib

Windows: None

Arguments

\textit{dir} \hspace{1cm} \text{Is an optional top-level location for the gcc binaries and libraries.}

Default
C++: -cxxlib
C: -no-cxxlib

For C++, the compiler uses the run-time libraries and headers provided by gcc.
For C, the compiler uses the default run-time libraries and headers and does not link to any additional C++ run-time libraries and headers.
However, if you specify compiler option
-std=gnu++98,
the default is -cxxlib.

Description
This option determines whether the compile links using the C++ run-time libraries and header files provided by gcc.
Option -cxxlib-nostd prevents the compiler from linking with the standard C++ library.

Alternate Options
None

See Also
Building Applications: Options for Interoperability

D
Defines a macro name that can be associated with an optional value.

IDE Equivalent
Windows: Preprocessor > Preprocessor Definitions
Linux: Preprocessor > Preprocessor Definitions
Mac OS X: Preprocessor > Preprocessor Definitions

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: \texttt{-Dname[=value]}
Windows: \texttt{/Dname[=value]}

Arguments
\begin{itemize}
\item \texttt{name} \hspace{1cm} Is the name of the macro.
\item \texttt{value} \hspace{1cm} Is an optional integer or an optional character string delimited by
\end{itemize}
double quotes; for example, 
\texttt{Dname=string}.

**Default**

**OFF**

**Only**

default symbols 
or macros are defined.

**Description**

Defines a macro name that can be associated with an optional value. This option is equivalent to a \texttt{#define} preprocessor directive. If a \texttt{value} is not specified, \texttt{name} is defined as "1". If you specify \texttt{noD}, all preprocessor definitions apply only to \texttt{fpp} and not to Intel\textregistered Fortran conditional compilation directives. To use this option, you must also specify option \texttt{fpp}.

**Caution**

On Linux and Mac OS X systems, if you are not specifying a \texttt{value}, do not use \texttt{D} for \texttt{name}, because it will conflict with the \texttt{-DD} option.

**Alternate Options**

None

**See Also**

188
Building Applications: Predefined Preprocessor Symbols

\textbf{dD, QdD}

Same as \texttt{-dM}, but outputs \texttt{#define} directives in preprocessed source.

\textbf{IDE Equivalent}

None

\textbf{Architectures}

IA-32, Intel® 64, IA-64 architectures

\textbf{Syntax}

Linux and Mac OS X: \texttt{-dD}

Windows: \texttt{/QdD}

\textbf{Arguments}

None

\textbf{Default}

OFF

The compiler does not output \texttt{#define} directives.

\textbf{Description}

Same as \texttt{-dM}, but outputs \texttt{#define} directives in preprocessed source. To use this option, you must also specify the \texttt{E} option.

\textbf{Alternate Options}
None

debug (Linux* OS and Mac OS* X)

Enables or disables generation of debugging information.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -debug[keyword]
Windows:  None

Arguments

*keyword*

Is the type of debugging information to be generated.

Possible values are:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Disables generation of debugging information.</td>
</tr>
<tr>
<td>full or all</td>
<td>Generates complete debugging information.</td>
</tr>
<tr>
<td>minimal</td>
<td>Generates line number</td>
</tr>
</tbody>
</table>
information for debugging.

[no]emit_column Determines whether the compiler generates column number information for debugging.

[no]expr-source-pos Determines whether the compiler generates source position information at the expression level of granularity.

[no]inline-debug-info Determines whether the compiler generates enhanced debug information for inlined code.

[no]semantic-
stepping  whether the compiler generates enhanced debug information useful for breakpoints and stepping.

[no]variable-locations  Determines whether the compiler generates enhanced debug information useful in finding scalar local variables.

extended  Sets keyword values semantic-stepping and variable-locations.
For information on the non-default settings for these keywords, see the Description section.

**Default**

- `--debug none`  
  No debugging information is generated.

**Description**

This option enables or disables generation of debugging information. Note that if you turn debugging on, optimization is turned off.

**Keywords** `semantic-stepping`, `inline-debug-info`, `variable-locations`, and `extended` can be used in combination with each other. If conflicting keywords are used in combination, the last one specified on the command line has precedence.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-debug none</code></td>
<td>Disables generation of debugging information.</td>
</tr>
<tr>
<td><code>-debug full</code> or <code>-debug all</code></td>
<td>Generates complete debugging information. It is the same as specifying <code>-debug</code> with no keyword.</td>
</tr>
<tr>
<td><code>-debug minimal</code></td>
<td>Generates line number information for debugging.</td>
</tr>
<tr>
<td><code>-debug emit_column</code></td>
<td>Generates column number information for debugging.</td>
</tr>
<tr>
<td><code>-debug expr-source-pos</code></td>
<td>Generates source position information at the statement level of granularity.</td>
</tr>
<tr>
<td><code>-debug inline-debug-info</code></td>
<td>Generates enhanced debug information for inlined code. It provides more information to debuggers for</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-debug semantic-stepping</td>
<td>Generates enhanced debug information useful for breakpoints and stepping. It tells the debugger to stop only at machine instructions that achieve the final effect of a source statement. For example, in the case of an assignment statement, this might be a store instruction that assigns a value to a program variable; for a function call, it might be the machine instruction that executes the call. Other instructions generated for those source statements are not displayed during stepping. This option has no impact unless optimizations have also been enabled.</td>
</tr>
<tr>
<td>-debug variable-locations</td>
<td>Generates enhanced debug information useful in finding scalar local variables. It uses a feature of the Dwarf object module known as &quot;location lists&quot;. This feature allows the run-time locations of local scalar variables to be specified more accurately; that is, whether, at a given position in the code, a variable value is found in memory or a machine register.</td>
</tr>
<tr>
<td>-debug extended</td>
<td>Sets keyword values semantic-stepping and variable-locations. It also tells the compiler to include column numbers in the line information.</td>
</tr>
</tbody>
</table>

On Linux® systems, debuggers read debug information from executable images. As a result, information is written to object files and then added to the executable by the linker. On Mac OS® X systems, debuggers read debug information from object files. As a result, the executables don't contain any debug information.
Therefore, if you want to be able to debug on these systems, you must retain the object files.

**Alternate Options**

<table>
<thead>
<tr>
<th>Option</th>
<th>Linux and Mac</th>
<th>OS X</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>For debug full, -debug all, or -debug</td>
<td>-g</td>
<td>/debug:full,</td>
<td>/debug:full,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>/debug:all,</td>
<td>/debug:all,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>or /debug</td>
<td>or /debug</td>
</tr>
</tbody>
</table>

For -debug inline-debug-info

<table>
<thead>
<tr>
<th>Option</th>
<th>Linux and Mac</th>
<th>OS X</th>
<th>Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>For -debug inline-debug-info</td>
<td></td>
<td>-inline-debug-info</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(this is a</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>deprecated option)</td>
<td></td>
</tr>
</tbody>
</table>

**debug (Windows* OS)**

Enables or disables generation of debugging information.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X: None
Windows: /debug[:keyword]

Arguments

*keyword*

Is the type of debugging information to be generated.
Possible values are:

- **none** Generates no symbol table information.
- **full or all** Generates complete debugging information.
- **minimal** Generates line numbers and minimal debugging information.
- **partial** Deprecated. Generates global symbol table information needed for
linking.

For information on the non-default settings for these keywords, see the Description section.

**Default**

`/debug: none`  
This is the default on the command line and for a release configuration in the IDE.

`/debug: full`  
This is the default for a debug configuration in the IDE.

**Description**

This option enables or disables generation of debugging information. It is passed to the linker.

Note that if you turn debugging on, optimization is turned off.

If conflicting keywords are used in combination, the last one specified on the command line has precedence.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/debug: none</code></td>
<td>Disables generation of debugging information.</td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/debug:full or /debug:all</td>
<td>Generates complete debugging information. It produces symbol table information needed for full symbolic debugging of unoptimized code and global symbol information needed for linking. It is the same as specifying /debug with no keyword.</td>
</tr>
<tr>
<td>/debug:minimal</td>
<td>Generates line number information for debugging. It produces global symbol information needed for linking, but not local symbol table information needed for debugging.</td>
</tr>
<tr>
<td>/debug:partial</td>
<td>Generates global symbol table information needed for linking, but not local symbol table information needed for debugging. This option is deprecated and is not available in the IDE.</td>
</tr>
</tbody>
</table>

### Alternate Options

**For /debug:minimal**

- Linux and Mac OS X: None
- Windows: /Zd (this is a deprecated option)

**For /debug:full or /debug**

- Linux and Mac OS X: None
- Windows: /zi, /z7
**diag, Qdiag**

Controls the display of diagnostic information.

**IDE Equivalent**

Windows: **Diagnostics > Disable Specific Diagnostics** (/Qdiag-disable id)

**Diagnostics > Level of Static Analysis** (/Qdiag-enable[:sv1,sv2, sv3])

Linux: **Compilation Diagnostics > Disable Specific Diagnostics** (-diag-disable id)

**Compilation Diagnostics > Level of Static Analysis** (-diag-enable [sv1,sv2, sv3] or -diag-disable sv)

Mac OS X: **Diagnostics > Disable Specific Diagnostics** (-diag-disable id)

**Diagnostics > Level of Static Analysis** (-diag-enable [sv1,sv2, sv3])

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  -diag-type diag-list

Windows: /Qdiag-type:diag-list

**Arguments**

*type*  

Is an action to perform on diagnostics. Possible values are:

| enable | Enables a diagnostic |
message or a group of messages.

disable Disables a diagnostic message or a group of messages.

error Tells the compiler to change diagnostics to errors.

warning Tells the compiler to change diagnostics to warnings.

remark Tells the compiler to change diagnostics to remarks (comments).

**diag-list** Is a diagnostic group or ID value. Possible values are:

driver Specifies diagnostic
messages issued by the compiler driver.

**port-linux** Specifies diagnostic messages for language features that may cause errors when porting to Linux. This diagnostic group is only available on Windows systems.

**port-win** Specifies diagnostic messages for GNU extensions that may cause errors when
porting to Windows. This diagnostic group is only available on Linux and Mac OS X systems.

**thread**

Specifies diagnostic messages that help in thread-enabling a program.

**vec**

Specifies diagnostic messages issued by the vectorizer.

**par**

Specifies diagnostic messages issued by the auto-parallelizer.
(parallel optimizer).

sv[n]  Specifies diagnostic messages issued by the Static Verifier. \( n \) can be any of the following: 1, 2, 3. For more details on these values, see below.

warn  Specifies diagnostic messages that have a "warning" severity level.

error  Specifies diagnostic messages that have an "error" severity level.
### remark

Specifies diagnostic messages that are remarks or comments.

### cpu-dispatch

Specifies the CPU dispatch remarks for diagnostic messages. These remarks are enabled by default. This diagnostic group is only available on IA-32 architecture and Intel® 64 architecture.

### id[,id,...]

Specifies the ID number of
one or more messages. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each id.

tag[,tag,...] Specifies the mnemonic name of one or more messages. If you specify more than one mnemonic name, they
must be separated by commas. There can be no intervening white space between each tag.

Default

OFF

The compiler issues certain diagnostic messages by default.

Description

This option controls the display of diagnostic information. Diagnostic messages are output to stderr unless compiler option -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows) is specified.

When diag-list value "warn" is used with the Static Verifier (sv) diagnostics, the following behavior occurs:

• **Option -diag-enable warn (Linux and Mac OS X) and /Qdiag-enable:warn (Windows)** enable all Static Verifier diagnostics except those that have an "error" severity level. They enable all Static Verifier warnings, cautions, and remarks.
- Option `-diag-disable warn` (Linux and Mac OS X) and `/Qdiag-disable:warn` (Windows) disable all Static Verifier diagnostics except those that have an "error" severity level. They suppress all Static Verifier warnings, cautions, and remarks.

The following table shows more information on values you can specify for diag-list item `sv`.

<table>
<thead>
<tr>
<th>diag-list Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sv[n]</code></td>
<td>The value of <code>n</code> for Static Verifier messages can be any of the following:</td>
</tr>
<tr>
<td>1</td>
<td>Produces the diagnostics with severity level set to all critical errors.</td>
</tr>
<tr>
<td>2</td>
<td>Produces the diagnostics with severity level set to all errors. This is the default if <code>n</code> is not specified.</td>
</tr>
<tr>
<td>3</td>
<td>Produces the diagnostics with severity level set to all errors and warnings.</td>
</tr>
</tbody>
</table>

To control the diagnostic information reported by the vectorizer, use the `-vec-report` (Linux and Mac OS X) or `/Qvec-report` (Windows) option.

To control the diagnostic information reported by the auto-parallelizer, use the `-par-report` (Linux and Mac OS X) or `/Qpar-report` (Windows) option.

**Alternate Options**

**enable vec**

Linux and Mac OS X: `-vec-report`  
Windows: `/Qvec-report`

**disable vec**

Linux and Mac OS X: `-vec-report0`  
Windows: `/Qvec-report0`
enable par

Linux and Mac OS X: \texttt{-par-report}

Windows: \texttt{/Qpar-report}

disable par

Linux and Mac OS X: \texttt{-par-report0}

Windows: \texttt{/Qpar-report0}

Example

The following example shows how to enable diagnostic IDs 117, 230 and 450:

\texttt{-diag-enable 117,230,450} ! Linux and Mac OS X systems
\texttt{/Qdiag-enable:117,230,450} ! Windows systems

The following example shows how to change vectorizer diagnostic messages to warnings:

\texttt{-diag-enable vec -diag-warning vec} ! Linux and Mac OS X systems
\texttt{/Qdiag-enable:vec /Qdiag-warning:vec} ! Windows systems

Note that you need to enable the vectorizer diagnostics before you can change them to warnings.

The following example shows how to disable all auto-parallelizer diagnostic messages:

\texttt{-diag-disable par} ! Linux and Mac OS X systems
\texttt{/Qdiag-disable:par} ! Windows systems

The following example shows how to produce Static Verifier diagnostic messages for all critical errors:

\texttt{-diag-enable sv1} ! Linux and Mac OS X systems
\texttt{/Qdiag-enable:sv1} ! Windows system

The following example shows how to cause Static Verifier diagnostics (and default diagnostics) to be sent to a file:

\texttt{-diag-enable sv -diag-file=stat_ver_msg} ! Linux and Mac OS X systems
\texttt{/Qdiag-enable:sv /Qdiag-file=stat_ver_msg} ! Windows systems
Note that you need to enable the Static Verifier diagnostics before you can send them to a file. In this case, the diagnostics are sent to file stat_ver_msg.diag. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.

The following example shows how to change all diagnostic warnings and remarks to errors:

```
-diag-error warn,remark    ! Linux and Mac OS X systems
/Qdiag-error:warn,remark   ! Windows systems
```

See Also

diag-dump, Qdiag-dump compiler option
diag-id-numbers, Qdiag-id-numbers compiler option
diag-file, Qdiag-file compiler option
par-report, Qpar-report compiler option
vec-report, Qvec-report compiler option

diag-dump, Qdiag-dump

Tells the compiler to print all enabled diagnostic messages and stop compilation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -diag-dump
Windows: /Qdiag-dump

Arguments

None

Default
The compiler issues certain diagnostic messages by default.

**Description**

This option tells the compiler to print all enabled diagnostic messages and stop compilation. The diagnostic messages are output to `stdout`.

This option prints the enabled diagnostics from all possible diagnostics that the compiler can issue, including any default diagnostics.

If `-diag-enable diag-list` (Linux and Mac OS X) or `/Qdiag-enable diag-list` (Windows) is specified, the print out will include the `diag-list` diagnostics.

**Alternate Options**

None

**Example**

The following example adds vectorizer diagnostic messages to the printout of default diagnostics:

```
-diag-enable vec -diag-dump ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-dump ! Windows systems
```

**See Also**

`diag, Qdiag` compiler option

`diag-enable sv-include, Qdiag-enable sv-include`

Tells the Static Verifier to analyze include files and source files when issuing diagnostic messages.
IDE Equivalent

Windows: Diagnostics > Analyze Include Files
Linux: Compilation Diagnostics > Analyze Include Files
Mac OS X: Diagnostics > Analyze Include Files

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -diag-enable sv-include
Windows: /Qdiag-enable sv-include

Arguments

None

Default

OFF

The compiler issues certain diagnostic messages by default. If the Static Verifier is enabled, include files are not
analyzed by default.

Description

This option tells the Static Verifier to analyze include files and source files when issuing diagnostic messages. Normally, when Static Verifier diagnostics are enabled, only source files are analyzed.

To use this option, you must also specify `-diag-enable sv` (Linux and Mac OS X) or `/Qdiag-enable:sv` (Windows) to enable the Static Verifier diagnostics.

Alternate Options

None

Example

The following example shows how to cause include files to be analyzed as well as source files:

```
-diag-enable sv -diag-enable sv-include ! Linux and Mac OS systems
/Qdiag-enable:sv /Qdiag-enable:sv-include ! Windows systems
```

In the above example, the first compiler option enables Static Verifier messages. The second compiler option causes include files referred to by the source file to be analyzed also.

See Also

`diag`, `Qdiag` compiler option

`diag-error-limit`, `Qdiag-error-limit`

Specifies the maximum number of errors allowed before compilation stops.

IDE Equivalent

Windows: Compilation Diagnostics > Error Limit
Linux: **Compilation Diagnostics > Set Error Limit**
Mac OS X: **Compilation Diagnostics > Error Limit**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-diag-error-limit n`
- `-no-diag-error-limit`

Windows: `/Qdiag-error-limit:n`
- `/Qdiag-error-limit-`

**Arguments**

$n$

Is the maximum number of error-level or fatal-level compiler errors allowed.

**Default**

30

A maximum of 30 error-level and fatal-level
Description

This option specifies the maximum number of errors allowed before compilation stops. It indicates the maximum number of error-level or fatal-level compiler errors allowed for a file specified on the command line.

If you specify -no-diag-error-limit (Linux and Mac OS X) or /Qdiag-error-limit- (Windows) on the command line, there is no limit on the number of errors that are allowed.

If the maximum number of errors is reached, a warning message is issued and the next file (if any) on the command line is compiled.

Alternate Options

Linux and Mac OS X: -wn (this is a deprecated option)
Windows: /Qwn (this is a deprecated option)

diag-file, Qdiag-file

Causes the results of diagnostic analysis to be output to a file.

IDE Equivalent

Windows: Diagnostics > Diagnostics File
Linux: Compilation Diagnostics > Diagnostics File
Mac OS X: Diagnostics > Diagnostics File

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -diag-file[=file]
Windows: /Qdiag-file[::file]

Arguments

file
Is the name of the file for output.

Default

OFF
Diagnostic messages are output to stderr.

Description

This option causes the results of diagnostic analysis to be output to a file. The file is placed in the current working directory.

If file is specified, the name of the file is file.diag. The file can include a file extension; for example, if file.ext is specified, the name of the file is file.ext.

If file is not specified, the name of the file is name-of-the-first-source-file.diag. This is also the name of the file if the name specified for file conflicts with a source file name provided in the command line.

Note

If you specify -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows) and you also specify -diag-file-append (Linux and Mac OS X) or /Qdiag-file-append (Windows), the last option specified on the command line takes precedence.
Alternate Options

None

Example

The following example shows how to cause diagnostic analysis to be output to a file named `my_diagnostics.diag`:

```
-diag-file=my_diagnostics     ! Linux and Mac OS X systems
/Qdiag-file:my_diagnostics    ! Windows systems
```

See Also

diag, Qdiag compiler option
diag-file-append, Qdiag-file-append compiler option

diag-file-append, Qdiag-file-append

Causes the results of diagnostic analysis to be appended to a file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-diag-file-append[=file]`
Windows: `/Qdiag-file-append[:file]`

Arguments

`file`  
Is the name of the file to be appended
Default

OFF

Diagnostic messages are output to stderr.

Description

This option causes the results of diagnostic analysis to be appended to a file. If you do not specify a path, the driver will look for file in the current working directory. If file is not found, then a new file with that name is created in the current working directory. If the name specified for file conflicts with a source file name provided in the command line, the name of the file is name-of-the-first-source-file.diag.

Note

If you specify -diag-file-append (Linux and Mac OS X) or /Qdiag-file-append (Windows) and you also specify -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows), the last option specified on the command line takes precedence.

Alternate Options

None

Example
The following example shows how to cause diagnostic analysis to be appended to a file named my_diagnostics.txt:

- diag-file-append=my_diagnostics.txt ! Linux and Mac OS X systems  
/Qdiag-file-append=my_diagnostics.txt ! Windows systems

**See Also**

diag, Qdiag compiler option
diag-file, Qdiag-file compiler option

diag-id-numbers, Qdiag-id-numbers

Determines whether the compiler displays diagnostic messages by using their ID number values.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  -diag-id-numbers  
- no-diag-id-numbers  

Windows:  /Qdiag-id-numbers  
/Qdiag-id-numbers-

**Arguments**

None

**Default**

- diag-id-numbers  
or /Qdiag-id-numbers  
The compiler displays
This option determines whether the compiler displays diagnostic messages by using their ID number values. If you specify `-no-diag-id-numbers` (Linux and Mac OS X) or `/Qdiag-id-numbers-` (Windows), mnemonic names are output for driver diagnostics only.

Alternate Options
None

See Also
diag, Qdiag compiler option
diag-once, Qdiag-once

Tells the compiler to issue one or more diagnostic messages only once.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: `-diag-onceid[,id, ...]`
Windows: `/Qdiag-once:id[,id, ...]`
Arguments

$id$

Is the ID number of the diagnostic message. If you specify more than one message number, they must be separated by commas. There can be no intervening white space between each $id$.

Default

OFF

The compiler issues certain
This option tells the compiler to issue one or more diagnostic messages only once.

**Alternate Options**

Linux: `-wo` (this is a deprecated option)
Windows: `/Qwo` (this is a deprecated option)

**dM, QdM**

Tells the compiler to output macro definitions in effect after preprocessing.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-dM`
Windows: `/QdM`

**Arguments**

None
OFF

The compiler does not output macro definitions after preprocessing.

Description

This option tells the compiler to output macro definitions in effect after preprocessing. To use this option, you must also specify the E option.

Alternate Options

None

See Also

E compiler option
dN, QdN

Same as -D, but output #define directives contain only macro names.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -dN
Windows: /QdN

Arguments
Intel(R) C++ Compiler User and Reference Guides

None

Default

OFF

The compiler does not output #define directives.

Description

Same as -dD, but output #define directives contain only macro names. To use this option, you must also specify the E option.

Alternate Options

None

dryrun

Specifies that driver tool commands should be shown but not executed.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -dryrun
Windows: None

Arguments
None

Default

OFF

No tool commands are shown, but they are executed.

Description

This option specifies that driver tool commands should be shown but not executed.

Alternate Options

None

See Also

v compiler option

dumpmachine

Displays the target machine and operating system configuration.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -dumpmachine
Windows: None

Arguments
None

Default
OFF

The compiler does not display target machine or operating system information.

Description
This option displays the target machine and operating system configuration. No compilation is performed.

Alternate Options
None

See Also
dumpversion compiler option
dumpversion
Displays the version number of the compiler.

IDE Equivalent
None
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -dumpversion
Windows: None

Arguments
None

Default
OFF
The compiler does not display the compiler version number.

Description
This option displays the version number of the compiler. It does not compile your source files.

Alternate Options
None

Example
Consider the following command:

```
icc -dumpversion
```
If it is specified when using the Intel C++ Compiler 10.1, the compiler displays "10.1".

See Also

dumpmachine compiler option

dynamiclib

Invokes the libtool command to generate dynamic libraries.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux: None
Mac OS X: -dynamiclib
Windows: None

Arguments

None

Default

OFF

The compiler produces an executable.

Description
This option invokes the `libtool` command to generate dynamic libraries. When passed this option, the compiler uses the `libtool` command to produce a dynamic library instead of an executable when linking. To build static libraries, you should specify option `-staticlib` or `libtool -static <objects>`.

**Alternate Options**

None

**See Also**

`staticlib` compiler option

**dynamic-linker**

Specifies a dynamic linker other than the default.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux: `--dynamic-linker file`

Mac OS X: None

Windows: None

**Arguments**

`file` Is the name of the dynamic
Default

OFF

The default dynamic linker is used.

Description

This option lets you specify a dynamic linker other than the default.

Alternate Options

None

E

Causes the preprocessor to send output to `stdout`.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-E`

Windows: `/E`

Arguments
None

Default

OFF

Preprocessed source files are output to the compiler.

Description

This option causes the preprocessor to send output to `stdout`. Compilation stops when the files have been preprocessed.

When you specify this option, the compiler's preprocessor expands your source module and writes the result to `stdout`. The preprocessed source contains `#line` directives, which the compiler uses to determine the source file and line number.

Alternate Options

None

`early-template-check`

Lets you semantically check template function template prototypes before instantiation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-early-template-check`

`-no-early-template-check`
Windows: None

Arguments
None

Default
-no-early-template-check

The prototype instantiation of function templates and function members of class templates is deferred.

Description

Lets you semantically check template function template prototypes before instantiation. On Linux* OS platforms, gcc 3.4 (or newer) compatibility modes (i.e. -gcc-version=340 and later) must be in effect. For all Mac OS* X platforms, gcc 4.0 (or newer) is required.

Alternate Options
None

EH

Enables different models of exception handing.

IDE Equivalent
Windows: **Code Generation > Enable C++ Exceptions**

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None

Windows: `/EHtype`

Arguments

\[type\]

Possible values are: \(a\), \(s\), or \(c\)

Default

OFF

Description

This option enables different models of exception handler:

- `/EHa -- enable asynchronous C++ exception handling model`
- `/EHs -- enable synchronous C++ exception handling model`
- `/EHc -- assume extern "C" functions do not throw exceptions`

Alternate Options

None

See Also

232
Qsafeeh compiler option

EP

Causes the preprocessor to send output to stdout, omitting #line directives.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -EP
Windows: /EP

Arguments

None

Default

OFF

Preprocessed source files are output to the compiler.

Description

This option causes the preprocessor to send output to stdout, omitting #line directives.
If you also specify option P or Linux option F, the preprocessor will write the results (without #line directives) to a file instead of stdout.

Alternate Options
export

Enables support for the C++ export template feature.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -export
Windows: None

Arguments

None

Default

OFF

The export template feature is not enabled.

Description

This option enables support for the C++ export template feature. This option is supported only in C++ mode.

Alternate Options
See Also

export-dir compiler option

declaration

export-dir

Specifies a directory name for the exported template search path.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -export-dir dir

Windows: None

Arguments

dir

Is the directory name to add to the search path.

Default

OFF

The compiler does not
recognize exported templates.

Description

This option specifies a directory name for the exported template search path. To use this option, you must also specify the \texttt{-export} option. Directories in the search path are used to find the definitions of exported templates and are searched in the order in which they are specified on the command-line. The current directory is always the first entry in the search path.

Alternate Options

None

See Also

\texttt{export} compiler option

F (Mac OS\* X)

Add framework directory to head of include file search path.

IDE Equivalent

None

Architectures

IA-32 architecture

Syntax

Linux: None
Mac OS X: \texttt{-Fdir}
Windows: None

Arguments

236
dir

Is the name for the framework directory.

Default

OFF

The compiler does add a framework directory to head of include file search path.

Description

Add framework directory to head of include file search path.

Alternate Options

None

F (Windows*)

Specifies the stack reserve amount for the program.

IDE Equivalent

None

Architectures
Syntax

Linux and Mac OS X: None
Windows: \(/Fn\)

Arguments

\(n\)

Is the stack reserve amount. It can be specified as a decimal integer or by using a C-style convention for constants (for example, \(/F0x1000\)).

Default

OFF

The stack size default is
Description

This option specifies the stack reserve amount for the program. The amount ($n$) is passed to the linker.

Note that the linker property pages have their own option to do this.

Alternate Options

None

Fa

Specifies the contents of an assembly listing file.

IDE Equivalent

Windows: **Output Files > ASM List Location**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Fa[file|dir]

Arguments

file Is the
name of
the
assembly
listing
file.

dir

Is the
directory
where
the file
should
be
placed. It
can
include
file.

Default

OFF

No
assembly
listing file
is
produced.

Description

This option specifies that an assembly listing file should be generated (optionally
named file).

Alternate Options

Linux and Mac OS X: -s
Windows: None
FA

Specifies the contents of an assembly listing file.

IDE Equivalent

Windows: **Output Files > Assembler Output**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: `/FAspecifier`

Arguments

*specifier*  
Denotes the contents of the assembly listing file. Possible values are `c, s, orcs`.

Default
OFF

No additional information appears in the assembly listing file, if one is produced.

Description

These options specify what information, in addition to the assembly code, should be generated in the assembly listing file:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/FAc</td>
<td>Produces an assembly listing with machine code.</td>
</tr>
<tr>
<td>/FAs</td>
<td>Produces an assembly listing with source code.</td>
</tr>
<tr>
<td>/FAcS</td>
<td>Produces an assembly listing with machine code and source code.</td>
</tr>
</tbody>
</table>

Alternate Options

/fAc

Linux and Mac OS X: –fcode-asm
Windows: None

fabi-version

Instructs the compiler to select a specific ABI implementation.
IDE Equivalent

Windows: None
Linux: **Preprocessor > gcc Compatibility Options**
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-fabi-version=n`
Windows: None

Arguments

$n$

Is the ABI implementation.
Possible values are:

0 Requests the latest ABI implementation.

1 Requests the ABI implementation used in gcc 3.2 and gcc 3.3.

2 Requests the ABI implementation used in gcc 3.4
Varies

The compiler uses the ABI implementation that corresponds to the installed version of gcc.

Description

This option tells the compiler to select a specific ABI implementation. This option is compatible with gcc option `-fabi-version`. If you have multiple versions of gcc installed, the compiler may change the value of n depending on which gcc is detected in your path.

**Note**

gcc 3.2 and 3.3 are not fully ABI-compliant, but gcc 3.4 is highly ABI-compliant.

**Caution**

Do not mix different values for `-fabi-version` in one link.

Alternate Options

None

**falias**

Determines whether aliasing should be assumed in the program.
IDE Equivalent

Windows: None
Linux: Data > Assume No Aliasing in Program
Mac OS X: Data > Assume No Aliasing in Program

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -falias
   -fno-alias

Windows: None

Arguments

None

Default

-falias

Aliasing is assumed in the program.

Description

This option determines whether aliasing should be assumed in the program. If you do not want aliasing to be assumed in the program, specify -fno-alias.

Alternate Options

Linux and Mac OS X: None
Windows: /Oa[-]
See Also

**ffnalias** compiler option

**falign-functions, Qfnalign**

Tells the compiler to align functions on an optimal byte boundary.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: 

```
-falign-functions[n]
-fno-align-functions
```

Windows: 

```
/Qfnalign[n]
/Qfnalign-
```

Arguments

\( n \)

Is the byte boundary for function alignment. Possible values are 2 or 16.

Default

```
-fno-align-functions or /Qfnalign-
```

The compiler
aligns functions on 2-byte boundaries. This is the same as specifying `falign-functions=2` (Linux and Mac OS X) or `/Qfnalign:2` (Windows).

**Description**

This option tells the compiler to align functions on an optimal byte boundary. If you do not specify \( n \), the compiler aligns the start of functions on 16-byte boundaries.

**Alternate Options**

None

**falign-stack**

Tells the compiler the stack alignment to use on entry to routines.

**IDE Equivalent**

None

**Architectures**

IA-32 architecture

**Syntax**
Linux and Mac OS X: \texttt{-falign-stack=mode}

Windows: \texttt{None}

**Arguments**

\texttt{mode} 

Is the method to use for stack alignment. Possible values are:

- \texttt{default} Tells the compiler to use default heuristics for stack alignment. If alignment is required, the compiler dynamically aligns the stack.

- \texttt{maintain-16-byte} Tells the compiler to not assume any specific stack alignment, but attempt to maintain alignment.
in case the stack is already aligned. If alignment is required, the compiler dynamically aligns the stack. This setting is compatible with GCC.

assume-16-byte

tells the compiler to assume the stack is aligned on 16-byte boundaries and continue to maintain 16-byte alignment. This setting is compatible with GCC.
Default

-falign-stack=default

The compiler uses default heuristics for stack alignment.

Description

This option tells the compiler the stack alignment to use on entry to routines.

Alternate Options

None

fargument-alias, Qalias-args

Determines whether function arguments can alias each other.

IDE Equivalent

Windows: None
Linux: **Data > Enable Argument Aliasing**
Mac OS X: **Data > Enable Argument Aliasing**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
-fargument-alias
-fargument-noalias

Windows:  
/Qalias-args
Arguments

None

Default

-fargument-alias or /Qalias-args

Function arguments can alias each other and can alias global storage.

Description

This option determines whether function arguments can alias each other. If you specify -fargument-noalias or /Qalias-args-, function arguments cannot alias each other, but they can alias global storage. On Linux and Mac OS X systems, you can also disable aliasing for global storage, by specifying option -fargument-noalias-global.

Alternate Options

Linux and Mac OS X: -[no-]alias-args (this is a deprecated option)
Windows: None

See Also

-fargument-noalias-global compiler option

fargument-noalias-global
Tells the compiler that function arguments cannot alias each other and cannot alias global storage.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -fargument-noalias-global

Windows: None

Arguments

None

Default

OFF

Function arguments can alias each other and can alias global storage.

Description

This option tells the compiler that function arguments cannot alias each other and they cannot alias global storage.

If you only want to prevent function arguments from being able to alias each other, specify option -fargument-noalias.
fasm-blocks

Enables the use of blocks and entire functions of assembly code within a C or C++ file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux: None

Mac OS X: -fasm-blocks

Windows: None

Arguments

None

Default

OFF

You cannot use these features within a
This option enables the use of blocks and entire functions of assembly code within a C or C++ file.
Note that this option enables a Microsoft*-style assembly block not a GNU*-style assembly block.
This option is provided for compatibility with the Apple* GNU compiler.

Alternate Options
None

fast
Maximizes speed across the entire program.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -fast
Windows: /fast

Arguments
None

Default
OFF

The optimizations that maximize speed are not enabled.

Description

This option maximizes speed across the entire program. It sets the following options:

- **On systems using IA-64 architecture:**
  - Windows: /O3 and /Qipo
  - Linux: -ipo, -O3, and -static

- **On systems using IA-32 architecture and Intel® 64 architecture:**
  - Mac OS X: -ipo, -mdynamic-no-pic, -O3, -no-prec-div, -static, and -xHost
  - Windows: /O3, /Qipo, /Qprec-div-, and /QxHost
  - Linux: -ipo, -O3, -no-prec-div, -static, and -xHost

When option fast is specified on systems using IA-32 architecture or Intel® 64 architecture, you can override the -xHost or /QxHost setting by specifying a different processor-specific -x or /Qx option on the command line. However, the last option specified on the command line takes precedence. For example, if you specify -fast -xSSE3 (Linux) or /fast /QxSSE3 (Windows), option -xSSE3 or /QxSSE3 takes effect. However, if you specify -xSSE3 -fast (Linux) or /QxSSE3 /fast (Windows), option -xHost or /QxHost takes effect.

**Note**

The options set by option fast may change from release to release.
Alternate Options

None

**fast-transcendentals, Qfast-transcendentals**

Enables the compiler to replace calls to transcendental functions with faster but less precise implementations.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:**

- `-fast-transcendentals`
- `-no-fast-transcendentals`

**Windows:**

- `/Qfast-transcendentals`
- `/Qfast-transcendentals-`

Default

- `-fast-transcendentals` or `/Qfast-transcendentals`

The default depends on the setting of `-fp-model` (Linux and Mac OS X) or `/fp` (Windows). The default is ON if default
setting -fp-model fast or /fp:fast is in effect. However, if a value-safe option such as -fp-model precise or /fp:precise is specified, the default is OFF.

Description

This option enables the compiler to replace calls to transcendental functions with implementations that may be faster but less precise. It tells the compiler to perform certain optimizations on transcendental functions, such as replacing individual calls to sine in a loop with a single call to a less precise vectorized sine library routine.

This option has an effect only when specified with one of the following options:

- **Windows* OS**: /fp:except or /fp:precise
- **Linux* OS and Mac OS* X**: -fp-model except or -fp-model precise

You cannot use this option with option -fp-model strict (Linux and Mac OS X) or /fp:strict (Windows).

Alternate Options

None

See Also

(fp-model, fp compiler option)


**fbuiltin, Oi**

Determines whether inline expansion of intrinsic functions is enabled.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-fbuiltin[-func]`
   `-fno-builtin[-func]`

**Windows:** `/Oi[-]`

**Arguments**

`func`  
A comma-separated list of intrinsic functions.

**Default**

OFF

**Description**

This option determines whether inline expansion of one or more intrinsic functions is enabled. If `func` is not specified, `-fno-builtin` disables inline expansion for all intrinsic functions.
For a list of built-in functions affected by `-fbuiltin`, search for "built-in functions" in the appropriate gcc* documentation.

For a list of built-in functions affected by `/Oi`, search for "/Oi" in the appropriate Microsoft* Visual C/C++* documentation.

**Alternate Options**

None

**FC**

Displays the full path of source files passed to the compiler in diagnostics.

**IDE Equivalent**

Windows: **Advanced > Use Full Paths**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: **None**

Windows: **/FC**

**Arguments**

None

**Default**

OFF  

The compiler does not display the full path of
source files passed to the compiler in diagnostics.

Description
Displays the full path of source files passed to the compiler in diagnostics. This option is supported with Microsoft Visual Studio .NET 2003* or newer.

Alternate Options
None

fcode-asm
Produces an assembly listing with machine code annotations.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -fcode-asm
Windows: None

Arguments
None

Default
OFF

No machine code annotations appear in the assembly listing file, if one is produced.

Description

This option produces an assembly listing file with machine code annotations. The assembly listing file shows the hex machine instructions at the beginning of each line of assembly code. The file cannot be assembled; the filename is the name of the source file with an extension of .cod.

To use this option, you must also specify option -S, which causes an assembly listing to be generated.

Alternate Options

Linux and Mac OS X: None
Windows: /FAc

See Also

S compiler option

fcommon

Determines whether the compiler treats common symbols as global definitions.

IDE Equivalent

Windows: None
Linux: Data > Allow gprel Addressing of Common Data Variables
Mac OS X: Data > Allow gprel Addressing of Common Data Variables

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -fcommon
               -fno-common
Windows: None

Arguments
None

Default
-fcommon

The compiler does not treat common symbols as global definitions.

Description
This option determines whether the compiler treats common symbols as global definitions and to allocate memory for each symbol at compile time.
Option -fno-common tells the compiler to treat common symbols as global definitions. When using this option, you can only have a common variable declared in one module; otherwise, a link time error will occur for multiple defined symbols.
On IA-64 architecture, this option allows the use of gp-relative (gprel) addressing of common data variables. Normally, a file-scope declaration with no initializer and without the `extern` or `static` keyword "int i;" is represented as a common symbol. Such a symbol is treated as an external reference. However, if no other compilation unit has a global definition for the name, the linker allocates memory for it.

**Alternate Options**

None

**FD**

Generates file dependencies related to the Microsoft* C/C++ compiler.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None

Windows: `/FD`

**Arguments**

None

**Default**

OFF

The compiler does not generate Microsoft
Description

This option generates file dependencies related to the Microsoft C/C++ compiler. It invokes the Microsoft C/C++ compiler and passes the option to it.

Alternate Options

None

Fe

Specifies the name for a built program or dynamic-link library.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None

Windows: /Fe[file|dir]

Arguments

file Is the name for the built program or dynamic-
link library.

*dir*

Is the directory where the built program or dynamic-link library should be placed. It can include include *file*.

Default

OFF

The name of the file is the name of the first source file on the command line with file extension
.exe, so
file.f
becomes
file.exe.

Description

This option specifies the name for a built program (.EXE) or a dynamic-link library (.DLL).

You can use this option to specify an alternate name for an executable file. This is especially useful when compiling and linking a set of input files. You can use the option to give the resulting file a name other than that of the first input file (source or object) on the command line.

Alternate Options

Linux and Mac OS X: -o
Windows: None

Example

In the following example, the command produces an executable file named outfile.exe as a result of compiling and linking three files: one object file and two C++ source files.

```
prompt> icl /Feoutfile.exe file1.obj file2.cpp file3.cpp
```

By default, this command produces an executable file named file1.exe.

See Also

- compiler option
- fexceptions

Enables exception handling table generation.

IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -fexceptions
                        -fno-exceptions

Windows:  None

Arguments

None

Default

-fexceptions  Exception handling table generation is enabled. Default for C++.

-fno-exceptions  Exception handling table generation is disabled. Default for C.

Description
This option enables exception handling table generation. The `-fno-exceptions` option disables exception handling table generation, resulting in smaller code. When this option is used, any use of exception handling constructs (such as try blocks and throw statements) will produce an error. Exception specifications are parsed but ignored. It also undefines the preprocessor symbol `__EXCEPTIONS`.

Alternate Options
None

`ffnalias`

Specifies that aliasing should be assumed within functions.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-ffnalias`

Windows: None

Arguments
None

Default

-`ffnalias` Aliasing is assumed
Description

This option specifies that aliasing should be assumed within functions. The `-fnofnalias` option specifies that aliasing should not be assumed within functions, but should be assumed across calls.

Alternate Options

Linux and Mac OS X: None
Windows: `/Ow[-]`

See Also

`falias` compiler option

`ffreestanding, Qfreestanding`

Ensures that compilation takes place in a freestanding environment.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `ffreestanding`
Windows: `Qfreestanding`

Arguments

None

Default
OFF

Standard libraries are used during compilation.

Description

This option ensures that compilation takes place in a freestanding environment. The compiler assumes that the standard library may not exist and program startup may not necessarily be at main. This environment meets the definition of a freestanding environment as described in the C and C++ standard. An example of an application requiring such an environment is an OS kernel.

Note

When you specify this option, the compiler will not assume the presence of compiler-specific libraries. It will only generate calls that appear in the source code.

Alternate Options

None

ffunction-sections

Places each function in its own COMDAT section.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -ffunction-sections
Windows: None

Arguments

None

Default

OFF

Description

Places each function in its own COMDAT section.

Alternate Options

-fdata-sections

FI

Tells the preprocessor to include a specified filename as the header file.

IDE Equivalent

Windows: Advanced > Force Includes
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /FI file

Arguments
file

Is the file name to be included as the header file.

Default
OFF

The compiler uses default header files.

Description
This option tells the preprocessor to include a specified file name as the header file.
The file specified with /FI is included in the compilation before the first line of the primary source file.

Alternate Options
None

finline

Tells the compiler to inline functions declared with __inline and perform C++ inlining.

IDE Equivalent

272
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:  
-`finline`
-`fno-inline`

Windows:  
None

Arguments
None

Default
-`fno-inline`

The compiler does not inline functions declared with `__inline`.

Description
This option tells the compiler to inline functions declared with `__inline` and perform C++ inlining.

Alternate Options
None

finline-functions
Enables function inlining for single file compilation.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: 
- finline-functions
- fno-inline-functions

Windows: None

Arguments
None

Default
-finline-functions

Description
This option enables function inlining for single file compilation. It enables the compiler to perform inline function expansion for calls to functions defined within the current source file. The compiler applies a heuristic to perform the function expansion. To specify the size of the function to be expanded, use the -finline-limit option.
Alternate Options

Linux and Mac OS X: -inline-level=2
Windows: /Ob2

See Also

ip, Qip compiler option
finline-limit compiler option

finline-limit

Lets you specify the maximum size of a function to be inlined.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -finline-limit=n
Windows: None

Arguments

$n$

Must be an integer greater than or equal to zero. It is the maximum number of
This option lets you specify the maximum size of a function to be inlined. The compiler inlines smaller functions, but this option lets you inline large functions. For example, to indicate a large function, you could specify 100 or 1000 for \( n \). Note that parts of functions cannot be inlined, only whole functions.

This option is a modification of the \(-f\text{inline-functions}\) option, whose behavior occurs by default.

**Alternate Options**

None

**See Also**

*finline-functions* compiler option
finstrument-functions, Qinstrument-functions

Determines whether function entry and exit points are instrumented.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -finstrument-functions
                     -fno-instrument-functions
Windows:            /Qinstrument-functions
                     /Qinstrument-functions-

Arguments

None

Default

-fno-instrument-functions

or /Qinstrument-functions-

Function

entry and exit

points are not
instrumented.

Description

This option determines whether function entry and exit points are instrumented. It may increase execution time.

The following profiling functions are called with the address of the current function and the address of where the function was called (its "call site"):

- **This function is called upon function entry:**
  - On IA-32 architecture and Intel® 64 architecture:
void __cyg_profile_func_enter (void *this_fn,  
void *call_site);

- On IA-64 architecture:
  void __cyg_profile_func_enter (void **this_fn,  
  void *call_site);

- This function is called upon function exit:
  
- On IA-32 architecture and Intel® 64 architecture:
    void __cyg_profile_func_exit (void *this_fn,  
    void *call_site);

- On IA-64 architecture:
  void __cyg_profile_func_exit (void **this_fn,  
  void *call_site);

On IA-64 architecture, the additional de-reference of the function pointer argument is required to obtain the function entry point contained in the first word of the function descriptor for indirect function calls. The descriptor is documented in the Intel® Itanium® Software Conventions and Runtime Architecture Guide, section 8.4.2. You can find this design guide at web site [http://www.intel.com](http://www.intel.com).

These functions can be used to gather more information, such as profiling information or timing information. Note that it is the user's responsibility to provide these profiling functions.

If you specify `-finstrument-functions` (Linux and Mac OS X) or `/Qinstrument-functions` (Windows), function inlining is disabled. If you specify `-fno-instrument-functions` or `/Qinstrument-functions-`, inlining is not disabled.

On Linux and Mac OS X systems, you can use the following attribute to stop an individual function from being instrumented:

```
__attribute__((__no_instrument_function__))
```

It also stops inlining from being disabled for that individual function.

This option is provided for compatibility with gcc.

**Alternate Options**

None

fixed
Causes the linker to create a program that can be loaded only at its preferred base address.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None

Windows: /fixed

**Arguments**

None

**Default**

OFF

The compiler uses default methods to load programs.

**Description**

This option is passed to the linker, causing it to create a program that can be loaded only at its preferred base address.

**Alternate Options**

None
fjump-tables

Determines whether jump tables are generated for switch statements.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -fjump-tables
                 -fno-jump-tables

Windows:          None

Arguments

None

Default

-fjump-tables                              The
                                                   compiler
                                                   uses jump
tables for
                                                   switch
                                                   statements.

Description

This option determines whether jump tables are generated for switch statements.
Option -fno-jump-tables prevents the compiler from generating jump tables
for switch statements. This action is performed unconditionally and independent
of any generated code performance consideration.
Option `-fno-jump-tables` also prevents the compiler from creating switch statements internally as a result of optimizations.

Use `-fno-jump-tables` with `-fpic` when compiling objects that will be loaded in a way where the jump table relocation cannot be resolved.

Alternate Options

None

See Also

`fpic` compiler option

`fkeep-staticconsts`, `Qkeep-static-consts`

Tells the compiler to preserve allocation of variables that are not referenced in the source.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
`-fkeep-static-consts`
`-fno-keep-static-consts`

Windows:  
`/Qkeep-static-consts`
`/Qkeep-static-consts-`

Arguments

None

Default


-fno-keep-static-consts or /Qkeep-static-consts-

If a variable is never referenced in a routine, the variable is discarded unless optimizations are disabled by option –O0 (Linux and Mac OS X) or /Od (Windows).

Description

This option tells the compiler to preserve allocation of variables that are not referenced in the source.
The negated form can be useful when optimizations are enabled to reduce the memory usage of static data.

Alternate Options

None

Fm

Tells the linker to generate a link map file.
This option has been deprecated.

IDE Equivalent

None

Architectures

282
Syntax

Linux and Mac OS X: None
Windows: /Fm[ file | dir ]

Arguments

file
Is the name for the link map file.

dir
Is the directory where the link map file should be placed. It can include file.

Default

OFF
No link map is generated.

Description
This option tells the linker to generate

Alternate Options
None

**fma, Qfma**

Enables the combining of floating-point multiplies and add/subtract operations.

IDE Equivalent
Windows: None
Linux: **Floating Point > Floating-point Operation Contraction**
Mac OS X: None

Architectures
IA-64 architecture

Syntax

Linux: 
  -fma
  -no-fma

Mac OS X: None

Windows: 
  /Qfma
  /Qfma-

Arguments
None

Default

-fma
or /Qfma

Floating-point multiplies
Description

This option enables the combining of floating-point multiplies and add/subtract operations. It also enables the contraction of floating-point multiply and add/subtract operations into a single operation. The compiler contracts these operations whenever possible.
Alternate Options

Linux: -IPF-fma (this is a deprecated option)
Windows: /QIPF-fma (this is a deprecated option)

See Also

fp-model, fp compiler option
Floating-point Operations: Floating-point Options Quick Reference

fmath-errno

Tells the compiler that errno can be reliably tested after calls to standard math library functions.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fmath-errno
-fno-math-errno

Windows: None

Arguments

None

Default

-fno-math-errno

The compiler assumes that the
program does not test \texttt{errno} after calls to standard math library functions.

\textbf{Description}

This option tells the compiler to assume that the program tests \texttt{errno} after calls to math library functions. This restricts optimization because it causes the compiler to treat most math functions as having side effects. Option \texttt{-fno-math-errno} tells the compiler to assume that the program does not test \texttt{errno} after calls to math library functions. This frequently allows the compiler to generate faster code. Floating-point code that relies on IEEE exceptions instead of \texttt{errno} to detect errors can safely use this option to improve performance.

\textbf{Alternate Options}

None

\textbf{fminshared}

Specifies that a compilation unit is a component of a main program and should not be linked as part of a shareable object.

\textbf{IDE Equivalent}

None
This option specifies that a compilation unit is a component of a main program and should not be linked as part of a shareable object.

This option allows the compiler to optimize references to defined symbols without special visibility settings. To ensure that external and common symbol references are optimized, you need to specify visibility hidden or protected by using the -fvisibility, -fvisibility-hidden, or -fvisibility-protected option.
Also, the compiler does not need to generate position-independent code for the main program. It can use absolute addressing, which may reduce the size of the global offset table (GOT) and may reduce memory traffic.

Alternate Options
None

See Also

fvisibility compiler option

fmudflap

The compiler instruments risky pointer operations to prevent buffer overflows and invalid heap use.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -fmudflap
Mac OS X: None
Windows: None

Arguments
None

Default
OFF

The compiler
The compiler instruments risky pointer operations to prevent buffer overflows and invalid heap use. Requires gcc 4.0 or newer. When using this compiler option, you must specify linker option `-lmudflap` in the link command line to resolve references to the `libmudflap` library.

**Alternate Options**

None

**fno-gnu-keywords**

Do not recognize `typeof` as keyword.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `–fno-gnu-keywords`

Windows: None

**Arguments**

None
Default
OFF

Description
Do not recognize `typeof` as keyword.

Alternate Options
None

`fno-implicit-inline-templates`
Tells the compiler to not emit code for implicit instantiations of inline templates.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: `-fno-implicit-inline-templates`
Windows: None

Arguments
None

Default
OFF
The compiler handles inlines so that compilations,
with and without optimization, will need the same set of explicit instantiations.

Description

This option tells the compiler to not emit code for implicit instantiations of inline templates.

Alternate Options

None

\texttt{fno-implicit-templates}

Tells the compiler to not emit code for non-inline templates that are instantiated implicitly.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \texttt{-fno-implicit-templates}

Windows: None

Arguments

None
Default

OFF

The compiler handles inlines so that compilations, with and without optimization, will need the same set of explicit instantiations.

Description

This option tells the compiler to not emit code for non-inline templates that are instantiated implicitly, but to only emit code for explicit instantiations.

Alternate Options

None

fno-operator-names

Disables support for the operator names specified in the standard.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fno-operator-names
Arguments

None

Default

OFF

Description

Disables support for the operator names specified in the standard.

Alternate Options

None

fno-rtti

Disables support for run-time type information (RTTI).

IDE Equivalent

None

Architectures

IA-32, IA-64 architectures

Syntax

Linux and Mac OS X:  -fno-rtti
Windows:  None
OFF

Description
This option disables support for run-time type information (RTTI).

Alternate Options
None

\texttt{fnon-call-exceptions}
Allows trapping instructions to throw C++ exceptions.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
\textbf{Linux and Mac OS X:} \texttt{-fnon-call-exceptions}
\hfill \texttt{-fno-non-call-exceptions}
\textbf{Windows:} None

Arguments
None

Default
\texttt{-fno-non-call-exceptions}

C++ exceptions are not thrown from
trapping instructions.

**Description**

This option allows trapping instructions to throw C++ exceptions. It allows hardware signals generated by trapping instructions to be converted into C++ exceptions and caught using the standard C++ exception handling mechanism. Examples of such signals are SIGFPE (floating-point exception) and SIGSEGV (segmentation violation).

You must write a signal handler that catches the signal and throws a C++ exception. After that, any occurrence of that signal within a C++ try block can be caught by a C++ catch handler of the same type as the C++ exception thrown within the signal handler.

Only signals generated by trapping instructions (that is, memory access instructions and floating-point instructions) can be caught. Signals that can occur at any time, such as SIGALRM, cannot be caught in this manner.

**Alternate Options**

None

**fnon-lvalue-assign**

Determines whether casts and conditional expressions can be used as lvalues.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-fnon-lvalue-assign`
-fno-non-lvalue-assign

Windows: None

Arguments
None

Default
-fnon-lvalue-assign

The compiler allows casts and conditional expressions to be used as lvalues.

Description
This option determines whether casts and conditional expressions can be used as lvalues.

Alternate Options
None

fnsplit, Qfnsplit

Enables function splitting.

IDE Equivalent
Windows: Code Generation > Disable Function Splitting
Linux: None
Mac OS X: None
Architectures

/Qfnsplit[-]: IA-32 architecture, Intel® 64 architecture
-[no-]fnsplit: IA-64 architecture

Syntax

Linux:       -fnsplit
             -no-fnsplit
Mac OS X:    None
Windows:     /Qfnsplit
             /Qfnsplit-

Arguments

None

Default

-no-fnsplit
or/Qfnsplit-

Function splitting is not enabled unless –prof-use (Linux) or /Qprof-use (Windows) is also specified.

Description
This option enables function splitting if `-prof-use` (Linux) or `/Qprof-use` (Windows) is also specified. Otherwise, this option has no effect. It is enabled automatically if you specify `-prof-use` or `/Qprof-use`. If you do not specify one of those options, the default is `-no-fnsplit` (Linux) or `/Qfnsplit-` (Windows), which disables function splitting but leaves function grouping enabled.

To disable function splitting when you use `-prof-use` or `/Qprof-use`, specify `-no-fnsplit` or `/Qfnsplit-`.

Alternate Options

None

**Fo**

Specifies the name for an object file.

**IDE Equivalent**

Windows: **Output Files > Object File Name**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: **None**

Windows: `/Fo[file|dir]`

**Arguments**

`file`          Is the name for the object
dir

Is the directory where the object file should be placed. It can include file.

Default

OFF

An object file has the same name as the name of the first source file and a file extension of .obj.

Description

This option specifies the name for an object file.
Alternate Options

None

**fomit-frame-pointer, Oy**

Determines whether EBP is used as a general-purpose register in optimizations.

**IDE Equivalent**

Windows: Optimization > Omit Frame Pointers
Linux: Optimization > Provide Frame Pointer
Mac OS X: Optimization > Provide Frame Pointer

**Architectures**

-f[no-]omit-frame-pointer: IA-32 architecture, Intel® 64 architecture
/Oy[-]: IA-32 architecture

**Syntax**

Linux and Mac OS X: -fomit-frame-pointer
- fno-omit-frame-pointer

Windows: /Oy
       /Oy-

**Arguments**

None

**Default**

-fomit-frame-pointer
or /Oy

EBP is used as a general-purpose register in optimizations.
However, on Linux* and Mac OS X systems, the default is -fno-omit-frame-pointer if option -O0 or -g is specified. On Windows* systems, the default is /Oy- if option /Od is specified.

Description

These options determine whether EBP is used as a general-purpose register in optimizations. Options -fomit-frame-pointer and /Oy allow this use. Options -fno-omit-frame-pointer and /Oy- disallow it.

Some debuggers expect EBP to be used as a stack frame pointer, and cannot produce a stack backtrace unless this is so. The -fno-omit-frame-pointer and /Oy- options direct the compiler to generate code that maintains and uses EBP as a stack frame pointer for all functions so that a debugger can still produce a stack backtrace without doing the following:

- **For -fno-omit-frame-pointer:** turning off optimizations with -O0
- **For /Oy-:** turning off /O1, /O2, or /O3 optimizations
The `-fno-omit-frame-pointer` option is set when you specify option `-O0` or the `-g` option. The `-fomit-frame-pointer` option is set when you specify option `-O1`, `-O2`, or `-O3`. The `/Oy` option is set when you specify the `/O1`, `/O2`, or `/O3` option. Option `/Oy-` is set when you specify the `/Od` option.

Using the `-fno-omit-frame-pointer` or `/Oy-` option reduces the number of available general-purpose registers by 1, and can result in slightly less efficient code.

**Note**

There is currently an issue with GCC 3.2 exception handling. Therefore, the Intel compiler ignores this option when GCC 3.2 is installed for C++ and exception handling is turned on (the default).

**Alternate Options**

**Linux and Mac OS X:** `-fp` (this is a deprecated option)

**Windows:** None

**fp-model, fp**

Controls the semantics of floating-point calculations.

**IDE Equivalent**

**Windows:** None

**Linux:** Floating Point > Floating Point Model

**Mac OS X:** Floating Point > Floating Point Model

**Floating Point > Reliable Floating Point Exceptions Model** *(fp-model except)*

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-fp-model keyword`
Windows: `/fp:keyword`

Arguments

`keyword` Specifies the semantics to be used. Possible values are:

- `precise` Enables value-safe optimizations on floating-point data.
- `fast[=1|2]` Enables more aggressive optimizations on floating-point data.
- `strict` Enables precise and except, disables contractions, and enables `pragma stdc_fenv_access`.
- `source` Rounds intermediate results to
source-defined precision and enables value-safe optimizations.

**double**

Rounds intermediate results to 53-bit (double) precision.

**extended**

Rounds intermediate results to 64-bit (extended) precision.

**[no-]**

Determined whether floating-point exception semantics are used.

(except (Linux and Mac OS X)

or except[-] (Windows)

**Default**

- `-fp-model fast=1`

or `/fp:fast=1`

The compiler uses more aggressive optimizations on floating-
point calculations, except when
-00 (Linux and Mac OS X) or /Od (Windows) is specified.

Description

This option controls the semantics of floating-point calculations.

The *keywords* can be considered in groups:

- **Group A**: precise, fast, strict
- **Group B**: source, double, extended
- **Group C**: except (or the negative form)

You can use more than one *keyword*. However, the following rules apply:

- **You cannot specify fast and except together in the same compilation.**
  
  You can specify any other combination of group A, group B, and group C.
  
  Since fast is the default, you must not specify except without a group A or group B *keyword*.

- **You should specify only one *keyword* from group A and only one *keyword* from group B. If you try to specify more than one *keyword* from either group A or group B, the last (rightmost) one takes effect.

- **If you specify except more than once, the last (rightmost) one takes effect.**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fp-model precise or /fp:precise</td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when</td>
</tr>
</tbody>
</table>
implementing floating-point calculations. It disables optimizations that can change the result of floating-point calculations, which is required for strict ANSI conformance. These semantics ensure the accuracy of floating-point computations, but they may slow performance.

The compiler assumes the default floating-point environment; you are not allowed to modify it.

Intermediate results are computed with the precision shown in the following table, unless it is overridden by a keyword from Group B:

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Windows</th>
<th>Linux</th>
<th>Mac OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>IA-32</td>
<td>Double</td>
<td>Extended</td>
<td>Extended</td>
</tr>
<tr>
<td>Intel® 64</td>
<td>Source</td>
<td>Source</td>
<td>N/A</td>
</tr>
<tr>
<td>IA-64</td>
<td>Extended</td>
<td>Extended</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Floating-point exception semantics are disabled by default. To enable these semantics, you must also specify `-fp-model except` or `/fp:except`.

For information on the semantics used to
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fp-model fast [=1</td>
<td>2] or /fp:fast [=1</td>
</tr>
<tr>
<td>-fp-model strict or /fp:strict</td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations</td>
</tr>
</tbody>
</table>
and enables floating-point exception semantics. This is the strictest floating-point model.

The compiler does not assume the default floating-point environment; you are allowed to modify it. Floating-point exception semantics can be disabled by explicitly specifying `-fp-model no-except` or `/fp:except-`.

For information on the semantics used to interpret floating-point calculations in the source code, see strict in Floating-point Operations: Using the `-fp-model (/fp)` Option.

**-fp-model source or `/fp:source`**

This option causes intermediate results to be rounded to the precision defined in the source code. It also implies keyword `precise` unless it is overridden by a keyword from Group A. Intermediate expressions use the precision of the operand with higher precision, if any.

<table>
<thead>
<tr>
<th>Type</th>
<th>Precision</th>
<th>Data Type</th>
<th>Exponent</th>
</tr>
</thead>
<tbody>
<tr>
<td>long</td>
<td>64-bit</td>
<td>80-bit</td>
<td>15-bit</td>
</tr>
<tr>
<td>double</td>
<td>precision</td>
<td>data type</td>
<td>exponent</td>
</tr>
<tr>
<td>double</td>
<td>53-bit</td>
<td>64-bit</td>
<td>11-bit</td>
</tr>
</tbody>
</table>

...
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>using IA-32 architecture, the exponent may be 15-bit if an x87 register is used to hold the value.</td>
</tr>
<tr>
<td>float</td>
<td>24-bit 32-bit 8-bit precision data type exponent</td>
</tr>
<tr>
<td></td>
<td>The compiler assumes the default floating-point environment; you are not allowed to modify it.</td>
</tr>
<tr>
<td></td>
<td>For information on the semantics used to interpret floating-point calculations in the source code, see source in Floating-point Operations: Using the <code>-fp-model (/fp)</code> Option.</td>
</tr>
<tr>
<td>-fp-model double or /fp:double</td>
<td>This option causes intermediate results to be rounded as follows:</td>
</tr>
<tr>
<td></td>
<td>53-bit (double) precision</td>
</tr>
<tr>
<td></td>
<td>64-bit data type</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>11-bit exponent; on Windows systems using IA-32 architecture, the exponent may be 15-bit if an x87 register is used to hold the value. This option also implies keyword <code>precise</code> unless it is overridden by a keyword from Group A. The compiler assumes the default <code>floating-point environment</code>; you are not allowed to modify it. For information on the semantics used to interpret floating-point calculations in the source code, see <code>double</code> in <em>Floating-point Operations: Using the <code>-fp-model</code> /fp Option.</em></td>
</tr>
<tr>
<td><code>-fp-model extended</code></td>
<td>This option causes intermediate results to be rounded as follows:</td>
</tr>
<tr>
<td><code>/fp:extended</code></td>
<td>64-bit (extended) precision</td>
</tr>
<tr>
<td></td>
<td>80-bit data type</td>
</tr>
<tr>
<td></td>
<td>15-bit exponent</td>
</tr>
<tr>
<td></td>
<td>This option also implies keyword <code>precise</code> unless it is overridden by a keyword from Group A. The compiler assumes the default <code>floating-point environment</code>; you are not allowed to modify it. For information on the semantics used to interpret floating-point calculations in the source code, see <code>double</code> in <em>Floating-point Operations: Using the <code>-fp-model</code> /fp Option.</em></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-fp-model except or</td>
<td>interpret floating-point calculations in the source code, see double in Floating-point Operations: Using the -fp-model (/fp) Option.</td>
</tr>
<tr>
<td>/fp:except</td>
<td>Tells the compiler to use floating-point exception semantics.</td>
</tr>
</tbody>
</table>

**Note**

On Windows and Linux operating systems on IA-32 architecture, the compiler, by default, implements floating-point (FP) arithmetic using SSE2 and SSE instructions. This can cause differences in floating-point results when compared to previous x87 implementations.

**Alternate Options**

None

**Example**

For examples of how to use this option, see *Floating-point Operations: Using the -fp-model (/fp) Option*.

**See Also**

- **_** compiler option (specifically O0)
- **Od** compiler option
- **mpl, Qprec** compiler option

The MSDN article Microsoft Visual C++ Floating-Point Optimization, which discusses concepts that apply to this option.

**Floating-point Operations: Floating-Point Environment**

**Fp**

312
Lets you specify an alternate path or file name for precompiled header files.

**IDE Equivalent**

Windows: **Precompiled Headers > Precompiled Header File**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None

Windows: 

```
/Fp{file|dir}
```

**Arguments**

- `file`
  - Is the name for the precompiled header file.

- `dir`
  - Is the directory where the precompiled header file should be placed. It can include `file`.

**Default**

OFF

The compiler
does not create or use precompiled headers unless you tell it to do so.

Description

This option lets you specify an alternate path or file name for precompiled header files.

Alternate Options

None

**fp-model, fp**

Controls the semantics of floating-point calculations.

IDE Equivalent

Windows: None
Linux: **Floating Point > Floating Point Model**
Mac OS X: **Floating Point > Floating Point Model**
**Floating Point > Reliable Floating Point Exceptions Model** *(fp-model except)*

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-fp-model keyword`
Windows: `/fp:keyword`

Arguments

`keyword` Specifies the semantics to be used. Possible values are:

`precise` Enables value-safe optimizations on floating-point data.

`fast[=1|2]` Enables more aggressive optimizations on floating-point data.

`strict` Enables precise and except, disables contractions, and enables `pragma stdc_fenv_access`.

`source` Rounds intermediate results to source-defined precision and enables value-
**double**

Rounds intermediate results to 53-bit (double) precision.

**extended**

Rounds intermediate results to 64-bit (extended) precision.

**[no-]except**

Determines whether floating-point exception semantics are used.

*Default*

- `-fp-model fast=1`
- `/fp:fast=1`

The compiler uses more aggressive optimizations on floating-point calculations, except when...
Description

This option controls the semantics of floating-point calculations.

The *keywords* can be considered in groups:

- **Group A:** `precise`, `fast`, `strict`
- **Group B:** `source`, `double`, `extended`
- **Group C:** `except` (or the negative form)

You can use more than one *keyword*. However, the following rules apply:

- **You cannot specify `fast` and `except` together in the same compilation.**
  You can specify any other combination of group A, group B, and group C.
  Since `fast` is the default, you must not specify `except` without a group A or group B *keyword*.

- **You should specify only one *keyword* from group A and only one *keyword* from group B.**
  If you try to specify more than one *keyword* from either group A or group B, the last (rightmost) one takes effect.

- **If you specify `except` more than once, the last (rightmost) one takes effect.**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fp-model precise</code> or</td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when</td>
</tr>
<tr>
<td><code>/fp:precise</code></td>
<td>implementing floating-point calculations. It disables optimizations that can change the result of floating-point calculations, which</td>
</tr>
</tbody>
</table>
is required for strict ANSI conformance. These semantics ensure the accuracy of floating-point computations, but they may slow performance.

The compiler assumes the default floating-point environment; you are not allowed to modify it.

Intermediate results are computed with the precision shown in the following table, unless it is overridden by a keyword from Group B:

<table>
<thead>
<tr>
<th>Windows</th>
<th>Linux</th>
<th>Mac OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IA-32 Double Extended Extended architecture

Intel® 64 Source Source N/A architecture

IA-64 Extended Extended N/A architecture

Floating-point exception semantics are disabled by default. To enable these semantics, you must also specify `-fp-model except` or `/fp:except`.

For information on the semantics used to interpret floating-point calculations in the source code, see `precise` in *Floating-point Operations: Using the `-fp-model`*
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>`-fp-model fast[=1</td>
<td>2]` or</td>
</tr>
<tr>
<td>`/fp:fast[=1</td>
<td>2]`</td>
</tr>
<tr>
<td><code>-fp-model strict</code> or</td>
<td>Tells the compiler to strictly adhere to value-safe optimizations when implementing floating-point calculations and enables floating-point exception semantics. This is the strictest floating-point model.</td>
</tr>
<tr>
<td><code>/fp:strict</code></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>The compiler does not assume the default floating-point environment; you are allowed to modify it. Floating-point exception semantics can be disabled by explicitly specifying <code>-fp-model no-except</code> or <code>/fp:except-</code>. For information on the semantics used to interpret floating-point calculations in the source code, see strict in <em>Floating-point Operations: Using the <code>-fp-model</code> (/fp) Option</em>.</td>
</tr>
<tr>
<td><code>-fp-model source</code> or</td>
<td>This option causes intermediate results to be rounded to the precision defined in the source code. It also implies keyword <code>precise</code> unless it is overridden by a keyword from Group A. Intermediate expressions use the precision of the operand with higher precision, if any.</td>
</tr>
<tr>
<td><code>/fp:source</code></td>
<td></td>
</tr>
</tbody>
</table>

<p>| long                   | 64-bit 80-bit 15-bit                                                                                                                         |
| double                 | precision data type exponent                                                             |
| double                 | 53-bit 64-bit 11-bit                                                                  |
|                        | precision data type exponent; on Windows systems using IA-32 architecture, the          |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>exponent</td>
<td>may be 15-bit if an x87 register is used to hold the value.</td>
</tr>
<tr>
<td>float</td>
<td>24-bit 32-bit 8-bit precision data type exponent</td>
</tr>
</tbody>
</table>

The compiler assumes the default floating-point environment; you are not allowed to modify it.

For information on the semantics used to interpret floating-point calculations in the source code, see source in Floating-point Operations: Using the `-fp-model (/fp)` Option.

- `fp-model double` or `fp:double`

This option causes intermediate results to be rounded as follows:

53-bit (double) precision

64-bit data type

11-bit exponent; on Windows systems using IA-32 architecture, the exponent may be 15-bit if an x87 register is used to hold the value.

This option also implies keyword precise unless it is overridden by a keyword from
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Group A.</td>
</tr>
<tr>
<td></td>
<td>The compiler assumes the default floating-point environment; you are not allowed to modify it.</td>
</tr>
<tr>
<td></td>
<td>For information on the semantics used to interpret floating-point calculations in the source code, see double in <em>Floating-point Operations: Using the <code>-fp-model (/fp)</code> Option</em>.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-fp-model extended</code></td>
<td>This option causes intermediate results to be rounded as follows:</td>
</tr>
<tr>
<td></td>
<td>64-bit (extended) precision</td>
</tr>
<tr>
<td></td>
<td>80-bit data type</td>
</tr>
<tr>
<td></td>
<td>15-bit exponent</td>
</tr>
<tr>
<td></td>
<td>This option also implies keyword <code>precise</code> unless it is overridden by a keyword from Group A.</td>
</tr>
<tr>
<td></td>
<td>The compiler assumes the default floating-point environment; you are not allowed to modify it.</td>
</tr>
<tr>
<td></td>
<td>For information on the semantics used to interpret floating-point calculations in the source code, see double in <em>Floating-point Operations: Using the <code>-fp-model (/fp)</code> Option</em>.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-fp-model except</code></td>
<td>Tells the compiler to use floating-point exception semantics.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Note**

On Windows and Linux operating systems on IA-32 architecture, the compiler, by default, implements floating-point (FP) arithmetic using SSE2 and SSE instructions. This can cause differences in floating-point results when compared to previous x87 implementations.

Alternate Options

None

Example

For examples of how to use this option, see *Floating-point Operations: Using the -fp-model (/fp) Option*.

See Also

- compiler option (specifically O0)
- compiler option
- compiler option

The MSDN article Microsoft Visual C++ Floating-Point Optimization, which discusses concepts that apply to this option.

Floating-point Operations: Floating-Point Environment

**fp-port, Qfp-port**

Rounds floating-point results after floating-point operations.

IDE Equivalent

Windows: **Optimization > Floating-point Precision Improvements**

Linux: **Floating Point > Round Floating-Point Results**

Mac OS X: **Floating Point > Round Floating-Point Results**

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

**Linux and Mac OS X:** `-fp-port`  
  `-no-fp-port`

**Windows:**  
  `/Qfp-port`  
  `/Qfp-port-`

Arguments

None

Default

`-no-fp-port`  
**The default rounding behavior depends on the compiler's code generation decisions and the precision parameters of the operating system.**

Description

This option rounds floating-point results after floating-point operations. Rounding to user-specified precision occurs at assignments and type conversions. This has some impact on speed.
The default is to keep results of floating-point operations in higher precision. This provides better performance but less consistent floating-point results.

Alternate Options

None

**fp-relaxed, Qfp-relaxed**

Enables use of faster but slightly less accurate code sequences for math functions.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux: -fp-relaxed
       -no-fp-relaxed

Mac OS X: None

Windows: /Qfp-relaxed
         /Qfp-relaxed-

Arguments

None

Default

-no-fp-relaxed

or/Qfp-relaxed-

Default code sequences are used
for math functions.

Description

This option enables use of faster but slightly less accurate code sequences for math functions, such as divide and sqrt. When compared to strict IEEE* precision, this option slightly reduces the accuracy of floating-point calculations performed by these functions, usually limited to the least significant digit. This option also enables the performance of more aggressive floating-point transformations, which may affect accuracy.

Alternate Options

Linux: -IPF-fp-relaxed (this is a deprecated option)
Windows: /QIPF-fp-relaxed (this is a deprecated option)

See Also

fp-model, fp compiler option
fp-speculation, Qfp-speculation

Tells the compiler the mode in which to speculate on floating-point operations.

IDE Equivalent

Windows: Optimization > Floating-Point Speculation
Linux: Floating Point > Floating-Point Speculation
Mac OS X: Floating Point > Floating-Point Speculation

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fp-speculation=mode
Windows: `/Qfp-speculation:mode`

Arguments

`mode` Is the mode for floating-point operations. Possible values are:

`fast` Tells the compiler to speculate on floating-point operations.

`safe` Tells the compiler to disable speculation if there is a possibility that the speculation may cause a floating-point exception.

`strict` Tells the compiler to disable speculation
on floating-point operations.

off This is the same as specifying strict.

Default

-fp-speculation=fast or /Qfp-speculation:fast

The compiler speculates on floating-point operations. This is also the behavior when optimizations are enabled. However, if you specify no optimizations (-00 on Linux; /Od on Windows), the default is -fp-speculation=safe (Linux) or /Qfp-speculation:safe (Windows).

Description

This option tells the compiler the mode in which to speculate on floating-point operations.
Alternate Options
None

\textbf{fp-stack-check, Qfp-stack-check}

Tells the compiler to generate extra code after every function call to ensure that the floating-point stack is in the expected state.

\textbf{IDE Equivalent}

Windows: None
Linux: \textbf{Floating Point > Check Floating-point Stack}
Mac OS X: \textbf{Floating Point > Check Floating-point Stack}

\textbf{Architectures}

IA-32, Intel® 64 architectures

\textbf{Syntax}

Linux and Mac OS X: \textit{-fp-stack-check}
Windows: \textit{/Qfp-stack-check}

\textbf{Arguments}

None

\textbf{Default}

OFF

There is no checking to ensure that the floating-
Description

This option tells the compiler to generate extra code after every function call to ensure that the floating-point (FP) stack is in the expected state. By default, there is no checking. So when the FP stack overflows, a NaN value is put into FP calculations and the program's results differ. Unfortunately, the overflow point can be far away from the point of the actual bug. This option places code that causes an access violation exception immediately after an incorrect call occurs, thus making it easier to locate these issues.

Alternate Options

None

See Also

Floating-point Operations:
Checking the Floating-point Stack State

fpack-struct

Specifies that structure members should be packed together.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -fpack-struct
Windows: None

Arguments

None

Default

OFF

Description

Specifies that structure members should be packed together. Note: Using this option may result in code that is not usable with standard (system) c and C++ libraries.

Alternate Options

-Zp1

fpascal-strings

Allow for Pascal-style string literals.

IDE Equivalent

Windows: None
Linux: Data > Recognize Pascal Strings
Mac OS X: None

Architectures

IA-32 architecture

Syntax
Linux: None
Mac OS X: -fpascal-strings
Windows: None

Arguments

None

Default

OFF

The compiler does not allow for Pascal-style string literals.

Description

Allow for Pascal-style string literals.

Alternate Options

None

fpermissive

Allow for non-conformant code.

IDE Equivalent

None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fpermissive
Windows: None

Arguments

None

Default

OFF

Description

Allow for non-conformant code.

Alternate Options

None

fpic

Determines whether the compiler generates position-independent code.

IDE Equivalent

Windows: None
Linux: Code Generation > Generate Position Independent Code
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fpic
Intel® C++ Compiler User and Reference Guides

-fno-pic

Windows: None

Arguments

None

Default

-fno-pic or -fpic

On systems using IA-32 or Intel® 64 architecture, the compiler does not generate position-independent code. On systems using IA-64 architecture, the compiler generates position-independent code.

Description

This option determines whether the compiler generates position-independent code.
Option `-fpic` specifies full symbol preemption. Global symbol definitions as well as global symbol references get default (that is, preemptable) visibility unless explicitly specified otherwise.

Option `-fno-pic` is only valid on systems using IA-32 or Intel® 64 architecture.

On systems using IA-32 or Intel® 64 architecture, `-fpic` must be used when building shared objects.

This option can also be specified as `-fPIC`.

**Alternate Options**

None

**fpie**

Tells the compiler to generate position-independent code. The generated code can only be linked into executables.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux: `-fpie`

Mac OS X: None

Windows: None

**Arguments**

None

**Default**

OFF

The
compiler does not generate position-independent code for an executable-only object.

**Description**

This option tells the compiler to generate position-independent code. It is similar to `-fpic`, but code generated by `-fpie` can only be linked into an executable. Because the object is linked into an executable, this option causes better optimization of some symbol references.

To ensure that run-time libraries are set up properly for the executable, you should also specify option `-pie` to the compiler driver on the link command line.

Option `-fpie` can also be specified as `-fPIE`.

**Alternate Options**

None

**See Also**

`fpic` compiler option
`pie` compiler option

**Fr**

Invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file without information on local variables. This is a deprecated option.

**IDE Equivalent**

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Fr[\textit{file}|\textit{dir}]

Arguments

\textit{file} \hspace{1cm} \text{Is the name for the BSCMAKE .sbr file.}

\textit{dir} \hspace{1cm} \text{Is the directory where the file should be placed. It can include } \textit{file}.

Default

OFF \hspace{1cm} \text{The compiler does not invoke the Microsoft}
Description

This option invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file without information on local variables. You can provide a name for the file. If you do not specify a file name, the .sbr file gets the same base name as the source file.

Alternate Options

None

See Also

FR compiler option

FR

Invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file with complete symbolic information.

IDE Equivalent

Windows: Browse Information > Enable Browse Information
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None

Windows: FR[\textit{file}|\textit{dir}]

Arguments

\textit{file} \quad Is the name for the BSCMAKE .sbr file.

\textit{dir} \quad Is the directory where the file should be placed. It can include \textit{file}.

Default

OFF \quad The compiler does not invoke the Microsoft C/C++ compiler to produce
a .sbr file.

Description

This option invokes the Microsoft C/C++ compiler and tells it to produce a BSCMAKE .sbr file with complete symbolic information.
You can provide a name for the file. If you do not specify a file name, the .sbr file gets the same base name as the source file.

Alternate Options

None

See Also

 Fernandez compiler option

fr32

Disables the use of the high floating-point registers.

IDE Equivalent

Windows: None
Linux: Floating Point > Disable Use of High Floating-point Registers
Mac OS X: None

Architectures

IA-64 architecture

Syntax

Linux: -fr32
Mac OS X: None
Windows: None
Arguments

None

Default

OFF

The use of the high floating-point registers is enabled.

Description

This option disables the use of the high floating-point registers. Only the lower 32 floating-point registers are used.

Alternate Options

None

freg-struct-return

Return struct and union values in registers when possible.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -freg-struct-return
Windows: None

Arguments
None

Default
OFF

Description
Return struct and union values in registers when possible.

Alternate Options
None

fshort-enums
Tells the compiler to allocate as many bytes as needed for enumerated types.

IDE Equivalent
Windows: None
Linux: Data > Associate as Many Bytes as Needed for Enumerated Types
Mac OS X: Data > Allocate enumerated types

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -fshort-enums
Windows: None

Arguments
None
Default

OFF

The compiler allocates a default number of bytes for enumerated types.

Description

This option tells the compiler to allocate as many bytes as needed for enumerated types.

Alternate Options

None

fsource-asm

Produces an assembly listing with source code annotations.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fsource-asm
Windows: None

Arguments
None

**Default**

OFF  

No source code annotations appear in the assembly listing file, if one is produced.

**Description**

This option produces an assembly listing file with source code annotations. The assembly listing file shows the source code as interspersed comments. To use this option, you must also specify option `-S`, which causes an assembly listing to be generated.

**Alternate Options**

None

**See Also**

- `$` compiler option
- `fstack-security-check, GS`

Determines whether the compiler generates code that detects some buffer overruns.

**IDE Equivalent**

Windows: Code Generation > Buffer Security Check
Linux: None  
Mac OS X: None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:  
-fstack-security-check  
-fno-stack-security-check

Windows:  
/GS  
/GS-

Arguments

None

Default

-fno-stack-security-check  
or /GS-

The compiler does not detect buffer overruns.

Description

This option determines whether the compiler generates code that detects some buffer overruns that overwrite the return address. This is a common technique for exploiting code that does not enforce buffer size restrictions.

The /GS option is supported with Microsoft Visual Studio .NET 2003* and Microsoft Visual Studio 2005*.

Alternate Options
Linux and Mac OS X: \(-f[no-]stack-protector\)
Windows: None

**fstack-security-check, GS**

Determines whether the compiler generates code that detects some buffer overruns.

**IDE Equivalent**

Windows: **Code Generation > Buffer Security Check**
Linux: None
Mac OS X: None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X: \(-fstack-security-check\)
\(-fno-stack-security-check\)

Windows: \(/GS\)
\(/GS-\)

**Arguments**

None

**Default**

\(-fno-stack-security-check\)
\(or /GS-\)

The compiler does not detect buffer overruns.
Description

This option determines whether the compiler generates code that detects some buffer overruns that overwrite the return address. This is a common technique for exploiting code that does not enforce buffer size restrictions. The /GS option is supported with Microsoft Visual Studio .NET 2003* and Microsoft Visual Studio 2005*.

Alternate Options

Linux and Mac OS X: -f[no-]stack-protector
Windows: None

fsyntax-only

Tells the compiler to check only for correct syntax.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -fsyntax-only
Windows: None

Arguments

None

Default
OFF Normal compilation is performed.

**Description**

For details, see option *syntax*.

**Alternate Options**

/zs

*ftemplate-depth, Qtemplate-depth*

Control the depth in which recursive templates are expanded.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  `-ftemplate-depth-n`

Windows:  `/Qtemplate-depth-n`

**Arguments**

*n*  The number of recursive templates that are
Default

OFF

Description

Control the depth in which recursive templates are expanded. On Linux*, this option is supported only by invoking the compiler with \texttt{icpc}.

Alternate Options

None

\texttt{ftls-model}

Change thread local storage model.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

\textbf{Linux and Mac OS X:} \texttt{-ftls-model=model}

Windows: \texttt{None}

Arguments

\textit{model}

Possible values are:

\texttt{global-}

\texttt{dynamic}
local-dynamic
initial-exec
local-exec

Default
OFF

Description
Change thread local storage model.

Alternate Options
None

ftrapuv, Qtrapuv
Initializes stack local variables to an unusual value to aid error detection.

IDE Equivalent
Windows: Code Generation > Initialize Local Variables to NaN
Linux: Code Generation > Initialize Local Variables to NaN
Mac OS X: Code Generation > Initialize Local Variables to NaN

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -ftrapuv
Windows: /Qtrapuv

Arguments
None

Default
OFF

Default
The compiler does not initialize local variables.

Description
This option initializes stack local variables to an unusual value to aid error detection. Normally, these local variables should be initialized in the application. The option sets any uninitialized local variables that are allocated on the stack to a value that is typically interpreted as a very large integer or an invalid address. References to these variables are then likely to cause run-time errors that can help you detect coding errors.

This option sets option -g (Linux and Mac OS X) and /zi or /z7 (Windows).

Alternate Options
None

See Also
g, zi, z7 compiler options

ftz, Qftz

Flushes denormal results to zero.
IDE Equivalent

Windows: **Optimization > Flush Denormal Results to Zero**
Linux: **Floating-Point > Flush Denormal Results to Zero**
Mac OS X: **Floating-Point > Flush Denormal Results to Zero**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-ftz`
   `-no-ftz`

Windows: `/Qftz`
   `/Qftz-`

Arguments

None

Default

Systems using IA-64 architecture: `-no-ftz` or `/Qftz-`

Systems using IA-32 architecture and Intel® 64 architecture: `-ftz` or `/Qftz`

On systems using IA-64 architecture, the compiler lets results gradually underflow.
On systems using IA-32 architecture and Intel® 64
Description

This option flushes denormal results to zero when the application is in the gradual underflow mode. It may improve performance if the denormal values are not critical to your application's behavior.

This option sets or resets the FTZ and the DAZ hardware flags. If FTZ is ON, denormal results from floating-point calculations will be set to the value zero. If FTZ is OFF, denormal results remain as is. If DAZ is ON, denormal values used as input to floating-point instructions will be treated as zero. If DAZ is OFF, denormal instruction inputs remain as is. Systems using IA-64 architecture have FTZ but not DAZ. Systems using Intel® 64 architecture have both FTZ and DAZ. FTZ and DAZ are not supported on all IA-32 architectures.

When `-ftz` (Linux and Mac OS X) or `/Qftz` (Windows) is used in combination with an SSE-enabling option on systems using IA-32 architecture (for example, xN or QxN), the compiler will insert code in the main routine to set FTZ and DAZ.

When `-ftz` or `/Qftz` is used without such an option, the compiler will insert code to conditionally set FTZ/DAZ based on a run-time processor check. `-no-ftz` (Linux and Mac OS X) or `/Qftz-` (Windows) will prevent the compiler from inserting any code that might set FTZ or DAZ.

This option only has an effect when the main program is being compiled. It sets the FTZ/DAZ mode for the process. The initial thread and any threads subsequently created by that process will operate in FTZ/DAZ mode.

On systems using IA-64 architecture, optimization option `O3` sets `-ftz` and `/Qftz`; optimization option `O2` sets `-no-ftz` (Linux) and `/Qftz-` (Windows).
On systems using IA-32 architecture and Intel® 64 architecture, every optimization option except -O0 sets -ftz and /Qftz. If this option produces undesirable results of the numerical behavior of your program, you can turn the FTZ/DAZ mode off by using -no-ftz or /Qftz- in the command line while still benefiting from the O3 optimizations.

**Note**

Options -ftz and /Qftz are performance options. Setting these options does not guarantee that all denormals in a program are flushed to zero. They only cause denormals generated at run time to be flushed to zero.

**Alternate Options**

None

**Example**

To see sample code showing the state of the FTZ and DAZ flags, see Reading the FTZ and DAZ Flags.

**See Also**

-x, -Qx compiler option

**func-groups**

This is a deprecated option. See prof-func-groups.

**funroll-loops**

See unroll, Qunroll.

**funroll-all-loops**

Unroll all loops even if the number of iterations is uncertain when the loop is entered.

**IDE Equivalent**
None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -funroll-all-loops
Windows: None

Arguments

None

Default

OFF Do not unroll all loops.

Description

Unroll all loops, even if the number of iterations is uncertain when the loop is entered. There may be a performance impact with this option.

Alternate Options

None

funsigned-bitfields

Determines whether the default bitfield type is changed to unsigned.

IDE Equivalent

Windows: None
Linux: Data > Change Default Bitfield Type to unsigned
Mac OS X: **Data > Unsigned bitfield Type**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-funsigned-bitfields`

`-fno-unsigned-bitfields`

Windows: None

**Arguments**

None

**Default**

`-fno-unsigned-bitfields` The default bitfield type is signed.

**Description**

This option determines whether the default bitfield type is changed to unsigned.

**Alternate Options**

None

**funsigned-char**

Change default char type to unsigned.

**IDE Equivalent**

Windows: None
Linux: **Data > Change default char type to unsigned**

Mac OS X: **Data > Unsigned char Type**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-funsigned-char`

Windows: None

**Arguments**

None

**Default**

OFF

Do not change default char type to unsigned.

**Description**

Change default char type to unsigned.

**Alternate Options**

None

**fverbose-asm**

Produces an assembly listing with compiler comments, including options and version information.
Intel® C++ Compiler User and Reference Guides

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-fverbose-asm`  
`-fno-verbose-asm`

Windows: None

Arguments
None

Default

`-fno-verbose-asm`  
No source code annotations appear in the assembly listing file, if one is produced.

Description

This option produces an assembly listing file with compiler comments, including options and version information.

To use this option, you must also specify `-S`, which sets `-fverbose-asm`. If you do not want this default when you specify `-S`, specify `-fno-verbose-asm`. 

358
Alternate Options

None

See Also

S compiler option

fvisibility

Specifies the default visibility for global symbols or the visibility for symbols in a file.

IDE Equivalent

Windows: None
Linux: Data > Default Symbol Visibility
Mac OS X: Data > Default Symbol Visibility

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-fvisibility=keyword`

    `-fvisibility-keyword=file`

Windows: None

Arguments

`keyword` Specifies the visibility setting. Possible values are:

default Sets visibility to default.
extern Sets visibility to extern.

hidden Sets visibility to hidden.

internal Sets visibility to internal.

protected Sets visibility to protected.

file Is the pathname of a file containing the list of symbols whose visibility you want to set. The symbols must be separated by whitespace.
Default

-fvisibility=default

The compiler sets visibility of symbols to default.

Description

This option specifies the default visibility for global symbols (syntax -fvisibility=keyword) or the visibility for symbols in a file (syntax -fvisibility=keyword=file).

Visibility specified by -fvisibility=keyword=file overrides visibility specified by -fvisibility=keyword for symbols specified in a file.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-fvisibility=default</td>
<td>Sets visibility of symbols to default. This means other components can reference the symbol, and the symbol definition can be overridden (preempted) by a definition of the same name in another component.</td>
</tr>
<tr>
<td>-fvisibility-default=file</td>
<td>Sets visibility of symbols to extern. This</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-fvisibility-extern=file</code></td>
<td>means the symbol is treated as though it is defined in another component. It also means that the symbol can be overridden by a definition of the same name in another component.</td>
</tr>
<tr>
<td><code>-fvisibility=hidden</code></td>
<td>Sets visibility of symbols to hidden. This means that other components cannot directly reference the symbol. However, its address may be passed to other components indirectly.</td>
</tr>
<tr>
<td><code>-fvisibility-hidden=file</code></td>
<td></td>
</tr>
<tr>
<td><code>-fvisibility=internal</code></td>
<td>Sets visibility of symbols to internal.</td>
</tr>
<tr>
<td><code>-fvisibility-internal=file</code></td>
<td>This means the symbol cannot be referenced outside its defining component, either directly or indirectly.</td>
</tr>
<tr>
<td><code>-fvisibility=protected</code></td>
<td></td>
</tr>
<tr>
<td><code>-fvisibility-protected=file</code></td>
<td></td>
</tr>
</tbody>
</table>

If an `-fvisibility` option is specified more than once on the command line, the last specification takes precedence over any others.

If a symbol appears in more than one visibility file, the setting with the least visibility takes precedence.

The following shows the precedence of the visibility settings (from greatest to least visibility):

- extern
- default
- protected
- hidden
- internal
Note that `extern` visibility only applies to functions. If a variable symbol is specified as `extern`, it is assumed to be `default`.

Alternate Options

None

Example

A file named `prot.txt` contains symbols a, b, c, d, and e. Consider the following:

```
-fvisibility-protected=prot.txt
```

This option sets `protected` visibility for all the symbols in the file. It has the same effect as specifying `fvisibility=protected` in the declaration for each of the symbols.

See Also

Optimizing Applications: Symbol Visibility Attribute Options (Linux* and Mac OS*X)

`fvisibility-inlines-hidden`

Causes inline member functions (those defined in the class declaration) to be marked hidden.

IDE Equivalent

None

Architectures

IA-32 architecture

Syntax

Linux and Mac OS X: `-fvisibility-inlines-hidden`

Windows: None
Arguments

None

Default

OFF

The compiler does not cause inline member functions to be marked hidden.

Description

Causes inline member functions (those defined in the class declaration) to be marked hidden. This option is particularly useful for templates.

Alternate Options

None

\textit{g, Zi, Z7}

Tells the compiler to generate full debugging information in the object file.

IDE Equivalent

Windows: \textbf{General > Debug Information Format}

Linux: \textbf{General > Include Debug Information}

Mac OS X: \textbf{General > Generate Debug Information}

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -g
Windows:           /Zi
                   /Z7

Arguments

None

Default

OFF  No  debugging
     information
     is
     produced
     in the
     object file.

Description

This option tells the compiler to generate symbolic debugging information in the object file for use by debuggers.

The compiler does not support the generation of debugging information in assemblable files. If you specify this option, the resulting object file will contain debugging information, but the assemblable file will not.

This option turns off 02 and makes 00 (Linux and Mac OS X) or 0d (Windows) the default unless 02 (or another 0 option) is explicitly specified in the same command line.
On Linux systems using Intel® 64 architecture and Linux and Mac OS X systems using IA-32 architecture, specifying the `-g` or `-O0` option sets the `-fno-omit-frame-pointer` option.

**Alternate Options**

Linux: None  
Windows: `/Zi`, `/debug`

**g0**

Disables generation of symbolic debug information.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-g0`  
Windows: None

**Arguments**

None

**Default**

OFF  
The compiler generates symbolic debug information.
Description

This option disables generation of symbolic debug information.

Alternate Options

None

G2, G2-p9000

Optimizes application performance for systems using IA-64 architecture.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux and Mac OS X: None
Windows: /G2
               /G2-p9000

Arguments

None

Default

/G2-p9000

Performance is optimized for Dual-Core Intel® Itanium® 2 processor 9000 series.
Description

These options optimize application performance for a particular Intel® processor or family of processors. The compiler generates code that takes advantage of features of IA-64 architecture.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G2</td>
<td>Optimizes for Intel® Itanium® 2 processors.</td>
</tr>
<tr>
<td>G2-p9000</td>
<td>Optimizes for Dual-Core Intel® Itanium® 2 processor 9000 series. This option affects the order of the generated instructions, but the generated instructions are limited to Intel® Itanium® 2 processor instructions unless the program specifies and executes intrinsics specific to the Dual-Core Intel® Itanium® 2 processor 9000 series.</td>
</tr>
</tbody>
</table>

The resulting executable is backwards compatible and generated code is optimized for specific processors. For example, code generated with /G2-p9000 will run correctly on single-core Itanium® 2 processors, but it might not run as fast as if it had been generated using /G2.

Alternate Options

/G2

Linux: –
mtune=itanium2
Mac OS X: None
Windows: None

/G2-p9000

Linux: –
mtune=itanium2-p9000,
mcpu=itanium2-
Example

In the following example, the compiled binary of the source program prog.c is optimized for the Dual-Core Intel® Itanium® 2 processor 9000 series by default. The same binary will also run on single-core Itanium® 2 processors (unless the program specifies and executes intrinsics specific to the Dual-Core Intel® Itanium® 2 processor 9000 series). All lines in the code example are equivalent.

```
icl prog.c
icl /G2-p9000 prog.c
```

In the following example, the compiled binary is optimized for single-core Itanium® 2 processors:

```
icl /G2 prog.c
```

See Also

mtune compiler option

G5, G6, G7

Optimize application performance for systems using IA-32 architecture and Intel® 64 architecture. These are deprecated options.

IDE Equivalent

Windows: **Optimization > Optimize for Processor**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: None
Windows:

\texttt{/G5}
\texttt{/G6}
\texttt{/G7}

Arguments

None

Default

\texttt{/G7}

On systems using IA-32 architecture and Intel® 64 architecture, performance is optimized for Intel® Pentium® 4 processors, Intel® Xeon® processors, Intel® Pentium® M processors, and Intel® Pentium® 4 processors with Streaming
SIMD Extensions 3 (SSE3) instruction support.

Description

These options optimize application performance for a particular Intel® processor or family of processors. The compiler generates code that takes advantage of features of the specified processor.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G5</td>
<td>Optimizes for Intel® Pentium® and Pentium® with MMX™ technology processors.</td>
</tr>
<tr>
<td>G6</td>
<td>Optimizes for Intel® Pentium® Pro, Pentium® II and Pentium® III processors.</td>
</tr>
<tr>
<td>G7</td>
<td>Optimizes for Intel® Core™ Duo processors, Intel® Core™ Solo processors, Intel® Pentium® 4 processors, Intel® Xeon® processors based on the Intel® Core microarchitecture, Intel® Pentium® M processors, and Intel® Pentium® 4 processors with Streaming SIMD Extensions 3 (SSE3) instruction support.</td>
</tr>
</tbody>
</table>

On systems using Intel® 64 architecture, only option G7 is valid. These options always generate code that is backwards compatible with Intel processors of the same architecture. For example, code generated with the G7 option runs correctly on Pentium III processors, although performance may be faster on Pentium III processors when compiled using or G6.

Alternate Options

Windows: /GB (an alternate for /G6; this option is also deprecated)
Linux: None
Example

In the following example, the compiled binary of the source program prog.c is optimized, by default, for Intel® Pentium® 4 processors, Intel® Xeon® processors, Intel® Pentium® M processors, and Intel® Pentium® 4 processors with Streaming SIMD Extensions 3 (SSE3). The same binary will also run on Pentium, Pentium Pro, Pentium II, and Pentium III processors. All lines in the code example are equivalent.

icl prog.c
icl /G7 prog.c

In the following example, the compiled binary is optimized for Pentium processors and Pentium processors with MMX technology:

ifort /G5 prog.f
icl /G5 prog.c

See Also

mtune compiler option

GA

Enables faster access to certain thread-local storage (TLS) variables.

IDE Equivalent

Windows: Optimization > Optimize for Windows Applications
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: None
Windows: /GA

Arguments
None

Default

OFF Default access to TLS variables is in effect.

Description

This option enables faster access to certain thread-local storage (TLS) variables. When you compile your main executable (.EXE) program with this option, it allows faster access to TLS variables declared with the __declspec(thread) specification.

Note that if you use this option to compile .DLLs, you may get program errors.

Alternate Options

None

gcc

Determines whether certain GNU macros are defined or undefined.

IDE Equivalent

Windows: None

Linux: Preprocessor > gcc Predefined Macro Enablement

Mac OS X: Preprocessor > Predefine gcc Macros

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:  
- gcc
  - no-gcc
  - gcc-sys

Windows:  None

Arguments

None

Default

- gcc

The compiler defines the GNU macros __GNUC__,
  __GNUC_MINOR__, and
  __GNUC_PATCHLEVEL__

Description

This option determines whether the GNU macros __GNUC__, __GNUC_MINOR__,
and __GNUC_PATCHLEVEL__ are defined or defined.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-gcc</td>
<td>Defines GNU macros</td>
</tr>
<tr>
<td>-no-gcc</td>
<td>Undefines GNU macros</td>
</tr>
<tr>
<td>-gcc-sys</td>
<td>Defines GNU macros only during compilation of system headers</td>
</tr>
</tbody>
</table>

Alternate Options

None

gcc

Determines whether certain GNU macros are defined or undefined.
IDE Equivalent

Windows: None
Linux: **Preprocessor > gcc Predefined Macro Enablement**
Mac OS X: **Preprocessor > Predefine gcc Macros**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-gcc`
  `-no-gcc`
  `-gcc-sys`
Windows: None

Arguments

None

Default

- `gcc`  
  The compiler defines the GNU macros `__GNUC__`, `__GNUC_MINOR__`, and `__GNUC_PATCHLEVEL__`

Description

This option determines whether the GNU macros `__GNUC__`, `__GNUC_MINOR__`, and `__GNUC_PATCHLEVEL__` are defined or defined.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-gcc</td>
<td>Defines GNU macros</td>
</tr>
<tr>
<td>-no-gcc</td>
<td>Undefines GNU macros</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>(-\text{gcc-sys})</td>
<td>Defines GNU macros only during compilation of system headers</td>
</tr>
</tbody>
</table>

Alternate Options

None

**gcc-name**

Specifies the location of the gcc compiler when the compiler cannot locate the gcc C++ libraries.

IDE Equivalent

Windows: None

Linux: Preprocessor > Nonstandard gcc Installation

Mac OS X: Preprocessor > gcc Installed to Non-standard Location

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-\text{gcc-name}=dir\)

Windows: None

Arguments

\(dir\)  

Is the full path location of the gcc compiler.
Default

OFF

The compiler locates the gcc libraries in the gcc install directory.

Description

This option specifies the location of the gcc compiler when the compiler cannot locate the gcc C++ libraries. To use this option, you must also specify the -cxxlib option.

This option is helpful when you are referencing a non-standard gcc installation. The C++ equivalent to option -gcc-name is -gxx-name.

Alternate Options

None

See Also

-gxx-name compiler option
cxxlib compiler option

gcc-version

Provides compatible behavior with gcc.

IDE Equivalent

Windows: None
Linux: Preprocessor > gcc Compatibility Options
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: \(-\text{gcc-version}=n\)
Windows: None

Arguments

\(n\)

Is the gcc compatibility.
Possible values are:

320 Specifies gcc 3.2 compatibility.
330 Specifies gcc 3.3 compatibility.
340 Specifies gcc 3.4 compatibility.
400 Specifies gcc 4.0 compatibility.
410 Specifies gcc 4.1 compatibility.
Specifies gcc 4.11 compatibility.

Specifies gcc 4.2 compatibility.

Specifies gcc 4.3 compatibility.

**Default**

**OFF**

This option defaults to the installed version of gcc.

**Description**

This option provides compatible behavior with gcc. It selects the version of gcc with which you achieve ABI interoperability.

**Alternate Options**

None

**See Also**

Compiler Options for Interoperability

**Gd**
Makes __cdecl the default calling convention.

IDE Equivalent

Windows: Advanced > Calling Convention
Linux: None
Mac OS X: None

Architectures

IA-32 architecture

Syntax

Linux and Mac OS X: None
Windows: /Gd

Arguments

None

Default

ON

The default calling convention is __cdecl.

Description

This option makes __cdecl the default calling convention.

Alternate Options

None
**gdwarf-2**

Enables generation of debug information using the DWARF2 format.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-gdwarf-2`

Windows: None

**Arguments**

None

**Default**

OFF

No debug information is generated. However, if compiler option `-g` is specified, debug information is generated.
in the latest DWARF format, which is currently DWARF2.

Description

This option enables generation of debug information using the DWARF2 format. This is currently the default when compiler option -g is specified.

Alternate Options

None

See Also

g compiler option

Ge

Enables stack-checking for all functions. This option has been deprecated.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Ge
Arguments

None

Default

OFF

Description

This option enables stack-checking for all functions.

Alternate Options

None

Gf

Enables read/write string-pooling optimization. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Gf

Arguments
Default

ON          Read/write
            string-pooling optimization is enabled.

Description

This option enables read/write string-pooling optimization. You should not use /Gf if you write to your strings because it can result in unexpected behavior. If you are not writing to your strings, you should use /GF.

Alternate Options

None

See Also

/GF compiler option

GF

Enables read-only string-pooling optimization.

IDE Equivalent

Windows: **Code Generation > Enable String Pooling**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

384
Linux and Mac OS X: None
Windows: /GF

Arguments

None

Default

OFF

Description

This option enables read only string-pooling optimization.

Alternate Options

None

Gh

Calls a function to aid custom user profiling.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Gh
Arguments

None

Default

OFF The compiler uses the default libraries.

Description

This option calls the __penter function to aid custom user profiling. The prototype for __penter is not included in any of the standard libraries or Intel-provided libraries. You do not need to provide a prototype unless you plan to explicitly call __penter.

Alternate Options

None

See Also

GH compiler option

GH

Calls a function to aid custom user profiling.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures
Syntax

Linux and Mac OS X: None
Windows: /GH

Arguments

None

Default

OFF

The compiler uses the default libraries.

Description

This option calls the __pexit function to aid custom user profiling. The prototype for __pexit is not included in any of the standard libraries or Intel-provided libraries. You do not need to provide a prototype unless you plan to explicitly call __pexit.

Alternate Options

None

See Also

Gh compiler option
Gm

Enables a minimal rebuild.

IDE Equivalent
Windows: **Code Generation > Enable Minimal Rebuild**
Linux: None
Mac OS X: None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X: None
Windows: \(/Gm\)

**Arguments**
None

**Default**

**Description**
This option enables a minimal rebuild.

**Alternate Options**
None

**global-hoist, Qglobal-hoist**
Enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source.

**IDE Equivalent**
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X:  
- -global-hoist
  -no-global-hoist
Windows:  
  /Qglobal-hoist
  /Qglobal-hoist-

Arguments
None

Default
- -global-hoist
  or /Qglobal-hoist

Description
This option enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source. In most cases, these optimizations are safe and can improve performance. The -no-global-hoist (Linux and Mac OS X) or /Qglobal-hoist- (Windows) option is useful for some applications, such as those that use shared or dynamically mapped memory, which can fail if a load is moved too early in the execution stream (for example, before the memory is mapped).
Alternate Options

None

Gr

Makes __fastcall the default calling convention.

IDE Equivalent

Windows: Advanced > Calling Convention
Linux: None
Mac OS X: None

Architectures

IA-32 architecture

Syntax

Linux and Mac OS X: None
Windows: /Gr

Arguments

None

Default

OFF

The default calling convention is __cdecl

Description
This option makes __fastcall the default calling convention.

Alternate Options

None

GR

Enables C++ Run Time Type Information (RTTI).

IDE Equivalent

Windows: **Language > Enable Run-Time Type Info**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: 

  /GR
  /GR-

Arguments

None

Default

/GR

When using Microsoft Visual Studio*
2005
When using Microsoft Visual Studio .NET 2003* (or earlier),

**Description**

This option enables C++ Run Time Type Information (RTTI). `/Qvc8` implies `/GR`, while `/Qvc7.1` (or lower) implies `/GR-`.

**Alternate Options**

None

**Gs**

Disables stack-checking for routines with more than a specified number of bytes of local variables and compiler temporaries.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None

Windows: `/Gs[n]`
Arguments

\( n \)

Is the number of bytes of local variables and compiler temporaries.

Default

4096

Stack checking is disabled for routines with more than 4KB of stack space allocated.

Description

This option disables stack-checking for routines with \( n \) or more bytes of local variables and compiler temporaries. If you do not specify \( n \), you get the default of 4096.

Alternate Options
None

**fstack-security-check, GS**

Determines whether the compiler generates code that detects some buffer overruns.

**IDE Equivalent**

Windows: **Code Generation > Buffer Security Check**
Linux: None
Mac OS X: None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X: `-fstack-security-check`  
`-fno-stack-security-check`

Windows: `/GS`  
`/GS-`

**Arguments**

None

**Default**

`-fno-stack-security-check`  
`/GS-`

The compiler does not detect buffer overruns.
Description

This option determines whether the compiler generates code that detects some buffer overruns that overwrite the return address. This is a common technique for exploiting code that does not enforce buffer size restrictions.

The /GS option is supported with Microsoft Visual Studio .NET 2003* and Microsoft Visual Studio 2005*.

Alternate Options

Linux and Mac OS X: -f[no-]stack-protector
Windows: None

GT

Enables fiber-safe thread-local storage of data.

IDE Equivalent

Windows: Optimization > Enable Fiber-safe Optimizations
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /GT

Arguments

None

Default
OFF

There is no fiber-safe thread-local storage.

Description

This option enables fiber-safe thread-local storage (TLS) of data.

Alternate Options

None

GX

Enables C++ exception handling.
This option is deprecated.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /GX
          /GX-

Arguments

None
Default

/GX

When using Microsoft Visual Studio* 2005

/GX-

When using Microsoft Visual Studio .NET 2003* (or earlier)

Description

This option enables C++ exception handling. /Qvc8 implies /GX, while /Qvc7.1 (or lower) implies /GX-.

Alternate Options

None

gxx-name

Specifies that the g++ compiler should be used to set up the environment for C++ compilations.

IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-gxx\text{-name}=dir\)
Windows: None

Arguments

dir

Is the full path location of the g++ compiler.

Default

OFF

The compiler uses the PATH setting to find the g++ compiler and resolve environment settings.

Description

This option specifies that the g++ compiler should be used to set up the environment for C++ compilations.
The C equivalent to option `-gxx-name` is `-gcc-name`.

**Note**

When compiling a C++ file with `icc`, `g++` is used to get the environment.

**Alternate Options**

None

**See Also**

`gcc-name` compiler option

**Gy**

Separates functions into COMDATs for the linker.

This option is deprecated.

**IDE Equivalent**

Windows: **Code Generation > Enable Function-Level Linking**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None

Windows: `/Gy`  

`/Gy-`

**Arguments**

None

**Default**
ON

The compiler separates functions into COMDATs.

Description

This option tells the compiler to separate functions into COMDATs for the linker.

Alternate Options

None

Gz

Makes __stdcall the default calling convention.

IDE Equivalent

Windows: Advanced > Calling Convention
Linux: None
Mac OS X: None

Architectures

IA-32 architecture

Syntax

Linux and Mac OS X: None
Windows: /Gz

Arguments

None
Default

OFF

The default calling convention is __cdecl.

Description

This option makes __stdcall the default calling convention.

Alternate Options

None

GZ

Initializes all local variables.
This option is deprecated.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /GZ

Arguments

None
Default

OFF

The
compiler
does not
initialize
local
variables.

Description

This option initializes all local variables to a non-zero value. To use this option, you must also specify option /Od.

Alternate Options

None

H, QH

Tells the compiler to display the include file order and continue compilation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -H
Windows: /QH

Arguments

None
Default

OFF

Compilation occurs as usual.

Description

This option tells the compiler to display the include file order and continue compilation.

Alternate Options

None

H (Windows*)

Causes the compiler to limit the length of external symbol names. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Hn

Arguments

\( n \)

Is the maximum number of
characters for external symbol names.

Default

OFF

The compiler follows default rules for the length of external symbol names.

Description

This option causes the compiler to limit the length of external symbol names to a maximum of $n$ characters.

Alternate Options

None

help

Displays all available compiler options or a category of compiler options.

IDE Equivalent

404
None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -help[category]
Windows: /help[category]

Arguments

category

Is a category or class of options to display. Possible values are:

advanced Displays advanced optimization options that allow fine tuning of compilation or allow control over advanced features of the compiler.

codegen Displays Code Generation options.

compatibility Displays options affecting language
compatibility.

**component** Displays options for component control.

**data** Displays options related to interpretation of data in programs or the storage of data.

**deprecated** Displays options that have been deprecated.

**diagnostics** Displays options that affect diagnostic messages displayed by the compiler.

**float** Displays options that affect floating-point operations.
help       Displays all the available help categories.
inline    Displays options that affect inlining.
ipo       Displays Interprocedural Optimization (IPO) options
language  Displays options affecting the behavior of the compiler language features.
link      Displays linking or linker options.
misc      Displays miscellaneous options that do not fit within other categories.
onopenmp  Displays OpenMP and
parallel processing options.

opt
Displays options that help you optimize code.

output
Displays options that provide control over compiler output.

pgo
Displays Profile Guided Optimization (PGO) options.

preproc
Displays options that affect preprocessing operations.

reports
Displays options for optimization reports.

Default
OFF

No list is
displayed unless this compiler option is specified.

Description

This option displays all available compiler options or a category of compiler options. If category is not specified, all available compiler options are displayed. This option can also be specified as --help.

Alternate Options

Linux and Mac OS X: None
Windows: /?

help-pragma, Qhelp-pragma

Displays all supported pragmas.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -help-pragma
Windows: /Qhelp-pragma

Arguments

None
Default

OFF  No list is displayed unless this compiler option is specified.

Description

This option displays all supported pragmas and shows their syntaxes.

Alternate Options

None

**homeparams**

Tells the compiler to store parameters passed in registers to the stack.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /homeparams

Arguments

None
Default

OFF

Register parameters are not written to the stack.

Description

This option tells the compiler to store parameters passed in registers to the stack.

Alternate Options

None

I

 Specifies an additional directory to search for include files.

IDE Equivalent

Windows: General > Additional Include Directories
Linux: Preprocessor > Additional Include Directories
Mac OS X: Preprocessor > Additional Include Directories

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \texttt{-Idir}
Windows: \texttt{/Idir}

Arguments

\texttt{dir} \hspace{1cm} Is the
additional
directory
for the
search.

Default

OFF

The
default
directory
is
searched
for
include
files.

Description

This option specifies an additional directory to search for include files. To specify multiple directories on the command line, repeat the include option for each directory.

Alternate Options

None

i-dynamic

This is a deprecated option. See shared-intel.

i-static

This is a deprecated option. See static-intel.

icc
Determines whether certain Intel compiler macros are defined or undefined.

IDE Equivalent

Windows: None
Linux: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-icc`
    `-no-icc`

Windows: None

Arguments

None

Default

`-icc` The `__ICC` and `__INTEL_COMPILER` macros are set to represent the current version of the compiler.

Description

This option determines whether certain Intel compiler macros are defined or undefined.

If you specify `-no-icc`, the compiler undefines the `__ICC` and `__INTEL_COMPILER` macros. These macros are defined by default or by specifying `-icc`. 
Alternate Options

None

**idirafter**

Adds a directory to the second include file search path.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

*Linux and Mac OS X:* `-idirafter dir`
*Windows:* None

**Arguments**

`dir` Is the name of the directory to add.

**Default**

OFF Include file search paths include certain default
directories.

Description

This option adds a directory to the second include file search path (after \-I).

Alternate Options

None

imacros

Allows a header to be specified that is included in front of the other headers in
the translation unit.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \-imacros \texttt{file}

Windows: None

Arguments

\texttt{file} \quad Name of header file.

Default

OFF
Description

Allows a header to be specified that is included in front of the other headers in the translation unit.

Alternate Options

None

inline-calloc, Qinline-calloc

Tells the compiler to inline calls to calloc() as calls to malloc() and memset().

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -inline-calloc

-no-inline-calloc

Windows: /Qinline-calloc

/Qinline-calloc-

Arguments

None

Default

-no-inline-calloc

or/Qinline-calloc-

The compiler
inlines
calls to
calloc()
as calls to calloc().

Description

This option tells the compiler to inline calls to calloc() as calls to malloc() and memset(). This enables additional memset() optimizations. For example, it can enable inlining as a sequence of store operations when the size is a compile time constant.

Alternate Options

None

inline-debug-info, Qinline-debug-info

Produces enhanced source position information for inlined code. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -inline-debug-info
Windows: /Qinline-debug-info

Arguments

None

Default
OFF

No enhanced source position information is produced for inlined code.

Description

This option produces enhanced source position information for inlined code. This leads to greater accuracy when reporting the source location of any instruction. It also provides enhanced debug information useful for function call traceback. To use this option for debugging, you must also specify a debug enabling option, such as `-g` (Linux) or `/debug` (Windows).

Alternate Options

Linux and Mac OS X: `-debug inline-debug-info`
Windows: None

inline-factor, Qinline-factor

Specifies the percentage multiplier that should be applied to all inlining options that define upper limits.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

418
Linux and Mac OS X:  
- **-inline-factor=n**
- **-no-inline-factor**

Windows:  
- **/Qinline-factor=n**
- **/Qinline-factor-**

**Arguments**

*n*  
Is a positive integer specifying the percentage value. The default value is 100 (a factor of 1).

**Default**

- **-no-inline-factor**  
- **/Qinline-factor-**  
The compiler uses default heuristics for inline routine expansion.

**Description**
This option specifies the percentage multiplier that should be applied to all inlining options that define upper limits:

- `-inline-max-size` and `/Qinline-max-size`
- `-inline-max-total-size` and `/Qinline-max-total-size`
- `-inline-max-per-routine` and `/Qinline-max-per-routine`
- `-inline-max-per-compile` and `/Qinline-max-per-compile`

This option takes the default value for each of the above options and multiplies it by \( \frac{n}{100} \). For example, if 200 is specified, all inlining options that define upper limits are multiplied by a factor of 2. This option is useful if you do not want to individually increase each option limit.

If you specify `-no-inline-factor` (Linux and Mac OS X) or `/Qinline-factor-` (Windows), the following occurs:

- Every function is considered to be a small or medium function; there are no large functions.
- There is no limit to the size a routine may grow when inline expansion is performed.
- There is no limit to the number of times some routine may be inlined into a particular routine.
- There is no limit to the number of times inlining can be applied to a compilation unit.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**Caution**

When you use this option to increase default limits, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None
See Also

`inline-max-size, Qinline-max-size` compiler option
`inline-max-total-size, Qinline-max-total-size` compiler option
`inline-max-per-routine, Qinline-max-per-routine` compiler option
`inline-max-per-compile, Qinline-max-per-compile` compiler option
`opt-report, Qopt-report` compiler option

Optimizing Applications:
Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

`inline-forceinline, Qinline-forceinline`

Specifies that an inline routine should be inlined whenever the compiler can do so.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-inline-forceinline`
Windows: `Qinline-forceinline`

Default

OFF The compiler uses default heuristics for inline
routine expansion.

**Description**

This option specifies that a inline routine should be inlined whenever the compiler can do so. This causes the routines marked with an inline keyword or attribute to be treated as if they were "forceinline".

**Note**

Because C++ member functions whose definitions are included in the class declaration are considered inline functions by default, using this option will also make these member functions "forceinline" functions. The "forceinline" condition can also be specified by using the keyword `__forceinline`.

To see compiler values for important inlining limits, specify compiler option –opt-report (Linux and Mac OS) or /Qopt-report (Windows).

**Caution**

When you use this option to change the meaning of inline to "forceinline", the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None

**See Also**

opt-report, Qopt-report compiler option

Optimizing Applications:
Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions
**inline-level, Ob**

Specifies the level of inline function expansion.

**IDE Equivalent**

Windows: **Optimization > Inline Function Expansion**
Linux: **Optimization > Inline Function Expansion**
Mac OS X: **Optimization > Inline Function Expansion**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-inline-level=n`
Windows: `/Obn`

**Arguments**

$n$

Is the inline function expansion level. Possible values are 0, 1, and 2.

**Default**

`-inline-level=2` or `/Ob2`

This is the default if option `O2`
is specified or is in effect by default. On Windows systems, this is also the default if option -O3 is specified.

-\texttt{-inline-level=0} or /\texttt{O}b0

This is the default if option -O0 (Linux and Mac OS) or /\texttt{O}d (Windows) is specified.

### Description

This option specifies the level of inline function expansion. Inlining procedures can greatly improve the run-time performance of certain programs.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-\texttt{-inline-level=0}</td>
<td>Disables inlining of user-defined functions. Note that</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>or Ob0</td>
<td>statement functions are always inlined.</td>
</tr>
<tr>
<td>-inline-level=1</td>
<td>Enables inlining when an inline keyword or an inline attribute is specified. Also enables inlining according to the C++ language.</td>
</tr>
<tr>
<td>or Ob1</td>
<td></td>
</tr>
<tr>
<td>-inline-level=2</td>
<td>Enables inlining of any function at the compiler's discretion.</td>
</tr>
<tr>
<td>or Ob2</td>
<td></td>
</tr>
</tbody>
</table>

**Alternate Options**

Linux: -Ob (this is a deprecated option)

Mac OS X: None

Windows: None

**inline-max-per-compile, Qinline-max-per-compile**

Specifies the maximum number of times inlining may be applied to an entire compilation unit.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-inline-max-per-compile=n`

- `no-inline-max-per-compile`

Windows: `/Qinline-max-per-compile=n`

- `/Qinline-max-per-compile-`
$n$ is a positive integer that specifies the number of times inlining may be applied.

Default

`-no-inline-max-per-compile` or `/Qinline-max-per-compile-`

The compiler uses default heuristics for inline routine expansion.

Description

This option the maximum number of times inlining may be applied to an entire compilation unit. It limits the number of times that inlining can be applied. For compilations using Interprocedural Optimizations (IPO), the entire compilation is a compilation unit. For other compilations, a compilation unit is a file.
If you specify `-no-inline-max-per-compile` (Linux and Mac OS X) or
/Qinline-max-per-compile-` (Windows), there is no limit to the number of
times inlining may be applied to a compilation unit.
To see compiler values for important inlining limits, specify compiler option –
`opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

⚠️ Caution

When you use this option to increase the default limit, the compiler may
do so much additional inlining that it runs out of memory and terminates
with an "out of memory" message.

**Alternate Options**

None

**See Also**

`inline-factor, Qinline-factor` compiler option
`opt-report, Qopt-report` compiler option

Optimizing Applications:

Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

**inline-max-per-routine, Qinline-max-per-routine**

Specifies the maximum number of times the inliner may inline into a particular
routine.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:  `-inline-max-per-routine=n`
`-no-inline-max-per-routine`

Windows:  `/Qinline-max-per-routine=n`
`/Qinline-max-per-routine-`

Arguments

\( n \)

| Is a positive integer that specifies the maximum number of times the inliner may inline into a particular routine.

Default

`-no-inline-max-per-routine`
`/Qinline-max-per-routine-`

| The compiler uses default heuristics for inline routine
expansion.

Description

This option specifies the maximum number of times the inliner may inline into a particular routine. It limits the number of times that inlining can be applied to any routine.

If you specify `-no-inline-max-per-routine` (Linux and Mac OS X) or `/Qinline-max-per-routine-` (Windows), there is no limit to the number of times some routine may be inlined into a particular routine.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

Caution

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

`inline-factor`, `Qinline-factor` compiler option
`opt-report`, `Qopt-report` compiler option
Optimizing Applications:
Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

`inline-max-size`, `Qinline-max-size`
Specifies the lower limit for the size of what the inliner considers to be a large routine.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -inline-max-size=n
-no-inline-max-size

Windows: /Qinline-max-size=n
/Qinline-max-size-

Arguments

\( n \)

Is a positive integer that specifies the minimum size of what the inliner considers to be a large routine.
Default

-no-inline-max-size
or/Qinline-max-size-

The compiler uses default heuristics for inline routine expansion.

Description

This option specifies the lower limit for the size of what the inliner considers to be a large routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be medium and large-size routines.

The inliner prefers to inline small routines. It has a preference against inlining large routines. So, any large routine is highly unlikely to be inlined.

If you specify -no-inline-max-size (Linux and Mac OS X) or/Qinline-max-size- (Windows), there are no large routines. Every routine is either a small or medium routine.

To see compiler values for important inlining limits, specify compiler option –opt-report (Linux and Mac OS X) or/Qopt-report (Windows).

Alternate Options

Caution

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.
inline-max-total-size, Qinline-max-total-size

Specifies how much larger a routine can normally grow when inline expansion is performed.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -inline-max-total-size=n
-neo-inline-max-total-size

Windows: /Qinline-max-total-size=n
/Qinline-max-total-size-

Arguments

n

Is a positive integer that
specifies the permitted increase in the routine's size when inline expansion is performed.

**Default**

-no-inline-max-total-size

or/Qinline-max-total-size-

The compiler uses default heuristics for inline routine expansion.

**Description**

This option specifies how much larger a routine can normally grow when inline expansion is performed. It limits the potential size of the routine. For example, if 2000 is specified for \( n \), the size of any routine will normally not increase by more than 2000.

If you specify -no-inline-max-total-size (Linux and Mac OS X) or /Qinline-max-total-size- (Windows), there is no limit to the size a routine may grow when inline expansion is performed.
To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**Caution**

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None

**See Also**

`inline-factor, Qinline-factor` compiler option
`opt-report, Qopt-report` compiler option

**Optimizing Applications:**

Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

`inline-min-size, Qinline-min-size`

Specifies the upper limit for the size of what the inliner considers to be a small routine.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X:  
- `inline-min-size=n`
- `no-inline-min-size`

Windows:  
- `/Qinline-min-size=n`
- `/Qinline-min-size-`

Arguments

$n$

Is a positive integer that specifies the maximum size of what the inliner considers to be a small routine.

Default

- `no-inline-min-size`
- `/Qinline-min-size-`

The compiler uses default heuristics for inline routine expansion.
Description

This option specifies the upper limit for the size of what the inliner considers to be a small routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be small and medium-size routines.

The inliner has a preference to inline small routines. So, when a routine is smaller than or equal to the specified size, it is very likely to be inlined.

If you specify `-no-inline-min-size` (Linux and Mac OS X) or `/Qinline-min-size-` (Windows), there is no limit to the size of small routines. Every routine is a small routine; there are no medium or large routines.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

Caution

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an “out of memory” message.

Alternate Options

None

See Also

`inline-min-size, Qinline-min-size` compiler option
`opt-report, Qopt-report` compiler option

Optimizing Applications:

Developer Directed Inline Expansion of User Functions

Compiler Directed Inline Expansion of User Functions

`ip, Qip`
Determines whether additional interprocedural optimizations for single-file compilation are enabled.

**IDE Equivalent**

**Windows:** Optimization > Interprocedural Optimization  
**Linux:** Optimization > Enable Interprocedural Optimizations for Single File Compilation  
**Mac OS X:** Optimization > Enable Interprocedural Optimization for Single File Compilation

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-ip`  
`-no-ip`  

**Windows:** `/Qip`  
`/Qip-`

**Arguments**

None

**Default**

OFF  
Some limited interprocedural optimizations occur, including inline function expansion for
calls to
functions
defined within
the current
source file.
These
optimizations
are a subset of
full intra-file
interprocedural
optimizations.
Note that this
setting is not
the same as -
no-ip (Linux
and Mac OS
X) or /Qip-
(Windows).

Description

This option determines whether additional interprocedural optimizations for
single-file compilation are enabled.
Options -ip (Linux and Mac OS X) and /Qip (Windows) enable additional
interprocedural optimizations for single-file compilation.
Options -no-ip (Linux and Mac OS X) and /Qip- (Windows) may not disable
inlining. To ensure that inlining of user-defined functions is disabled, specify -
inline-level=0 or -fno-inline (Linux and Mac OS X), or specify /Ob0
(Windows). To ensure that inlining of compiler intrinsic functions is disabled,
specify -fno-built-in (Linux and Mac OS X) or /Oi- (Windows).

Alternate Options
None

See Also
finline-functions compiler option

**ip-no-inlining, Qip-no-inlining**

Disables full and partial inlining enabled by interprocedural optimization options.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  
Windows:  

**Arguments**

None

**Default**

OFF

Inlining enabled by interprocedural optimization options is performed.

**Description**

This option disables full and partial inlining enabled by the following interprocedural optimization options:
• **On Linux and Mac OS X systems:** `-ip` or `-ipo
• **On Windows systems:** `/Qip`, `/Qipo`, or `/Ob2
It has no effect on other interprocedural optimizations.
On Windows systems, this option also has no effect on user-directed inlining specified by option `/Ob1`.

**Alternate Options**
None

**ip-no-pinlining, Qip-no-pinlining**
Disables partial inlining enabled by interprocedural optimization options.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64 architectures

**Syntax**
Linux and Mac OS X: `-ip-no-pinlining`
Windows: `/Qip-no-pinlining`

**Arguments**
None

**Default**
OFF

Inlining enabled by interprocedural optimization options is
Description

This option disables partial inlining enabled by the following interprocedural optimization options:

- **On Linux and Mac OS X systems:** `-ip` or `-ipo`
- **On Windows systems:** `/Qip` or `/Qipo`

It has no effect on other interprocedural optimizations.

Alternate Options

None

**IPF-flt-eval-method0, QIPF-flt-eval-method0**

Tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

- **Linux:** `-IPF-flt-eval-method0`
- **Mac OS X:** None
- **Windows:** `/QIPF-flt-eval-method0`

Arguments

None
Default

OFF

Expressions involving floating-point operands are evaluated by default rules.

Description

This option tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program.

By default, intermediate floating-point expressions are maintained in higher precision.

The recommended method to control the semantics of floating-point calculations is to use option `-fp-model` (Linux) or `/fp` (Windows).

Instead of using `-IPF-flt-eval-method0` (Linux) or `/QIPF-flt-eval-method0` (Windows), you can use `-fp-model source` (Linux) or `/fp:source` (Windows).

Alternate Options

None

See Also

`fp-model`, `fp` compiler option

`IPF-fltacc`, `QIPF-fltacc`
Disables optimizations that affect floating-point accuracy. This is a deprecated option.

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

Linux: 
-IPF-fltacc
-no-IPF-fltacc

Mac OS X: None

Windows: 
/QIPF-fltacc
/QIPF-fltacc-

**Arguments**

None

**Default**

-no-IPF-fltacc
or/QIPF-fltacc-

Optimizations are enabled that affect floating-point accuracy.

**Description**

This option disables optimizations that affect floating-point accuracy. If the default setting is used, the compiler may apply optimizations that reduce floating-point accuracy.
You can use this option to improve floating-point accuracy, but at the cost of disabling some optimizations.

The recommended method to control the semantics of floating-point calculations is to use option `-fp-model` (Linux) or `/fp` (Windows).

Instead of using `-IPF-fltacc` (Linux) or `/QIPF-fltacc` (Windows), you can use `-fp-model precise` (Linux) or `/fp:precise` (Windows).

Instead of using `-no-IPF-fltacc` (Linux) or `/QIPF-fltacc-` (Windows), you can use `-fp-model fast` (Linux) or `/fp:fast` (Windows).

Alternate Options

None

See Also

`fp-model`, `fp` compiler option

`IPF-fma`, `QIPF-fma`

See `fma`, `Qfma`.

`IPF-fp-relaxed`, `QIPF-fp-relaxed`

See `fp-relaxed`, `Qfp-relaxed`.

`ipo`, `Qipo`

Enables interprocedural optimization between files.

IDE Equivalent

Windows: Optimization > Interprocedural Optimization

Linux: Optimization > Enable Whole Program Optimization

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-ipo[n]`
Windows: `/Qipo[n]`

Arguments

$n$

Is an optional integer that specifies the number of object files the compiler should create. The integer must be greater than or equal to 0.

Default

OFF

Multifile interprocedural optimization is
Description

This option enables interprocedural optimization between files. This is also called multifile interprocedural optimization (multifile IPO) or Whole Program Optimization (WPO).

When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

You cannot specify the names for the files that are created.

If \( n \) is 0, the compiler decides whether to create one or more object files based on an estimate of the size of the application. It generates one object file for small applications, and two or more object files for large applications.

If \( n \) is greater than 0, the compiler generates \( n \) object files, unless \( n \) exceeds the number of source files \( (m) \), in which case the compiler generates only \( m \) object files.

If you do not specify \( n \), the default is 0.

Alternate Options

None

\textit{ipo-c, Qipo-c}

Tells the compiler to optimize across multiple files and generate a single object file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -ipo-c
Windows:       /Qipo-c

Arguments
None

Default
OFF

The compiler does not generate a multifile object file.

Description
This option tells the compiler to optimize across multiple files and generate a single object file (named ipo_out.o on Linux and Mac OS X systems; ipo_out.obj on Windows systems).
It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized object file that can be used in further link steps.

Alternate Options
None

See Also
ipo, Qipo compiler option
ipo-jobs, Qipo-jobs
Specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO).

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-ipo-jobs n`

Windows: `/Qipo-jobs:n`

**Arguments**

$n$

Is the number of commands (jobs) to run simultaneously. The number must be greater than or equal to 1.

**Default**

- `-ipo-jobs1`
- `/Qipo-jobs:1`

One command (job) is executed in an interprocedural optimization parallel build.
Description

This option specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO). It should only be used if the link-time compilation is generating more than one object. In this case, each object is generated by a separate compilation, which can be done in parallel.

This option can be affected by the following compiler options:

- `-ipo (Linux and Mac OS X) or `/Qipo (Windows) when applications are large enough that the compiler decides to generate multiple object files.
- `-ipon (Linux and Mac OS X) or `/Qipon (Windows) when \( n \) is greater than 1.
- `-ipo-separate (Linux) or `/Qipo-separate (Windows)

⚠️ Caution

Be careful when using this option. On a multi-processor system with lots of memory, it can speed application build time. However, if \( n \) is greater than the number of processors, or if there is not enough memory to avoid thrashing, this option can increase application build time.

Alternate Options

None

See Also

- `ipo, Qipo` compiler option
- `ipo-separate, Qipo-separate` compiler option

`ipo-S, Qipo-S`

Tells the compiler to optimize across multiple files and generate a single assembly file.

IDE Equivalent
None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -ipo-S
Windows: /Qipo-S

Arguments

None

Default

OFF

The compiler does not generate a multifile assembly file.

Description

This option tells the compiler to optimize across multiple files and generate a single assembly file (named ipo_out.s on Linux and Mac OS X systems; ipo_out.asm on Windows systems). It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized assembly file that can be used in further link steps.

Alternate Options

None
See Also

ipo, Qipo compiler option

ipo-separate, Qipo-separate

Tells the compiler to generate one object file for every source file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -ipo-separate
Mac OS X: None
Windows: /Qipo-separate

Arguments

None

Default

OFF

The compiler decides whether to create one or more object files.
Description

This option tells the compiler to generate one object file for every source file. It overrides any \texttt{ipo} (Linux) or \texttt{Qipo} (Windows) specification.

Alternate Options

None

See Also

\texttt{ipo, Qipo} compiler option

\texttt{iprefix}

Option for indicating the prefix for referencing directories containing header files.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \texttt{-iprefix \langle prefix\rangle}
Windows: \texttt{None}

Arguments

None

Default

OFF

Description
Options for indicating the prefix for referencing directories containing header files. Use `<prefix>` with `–iwithprefix` as a prefix.

Alternate Options

None

`iquote`

Add directory to the front of the include file search path for files included with quotes but not brackets.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `–iquote dir`
Windows: None

Arguments

`dir`

Is the name of the directory to add.

Default

OFF

The compiler
does not add a directory to the front of the include file search path.

**Description**

Add directory to the front of the include file search path for files included with quotes but not brackets.

**Alternate Options**

None

**-isystem**

Specifies a directory to add to the start of the system include path.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  `-isystemdir`

Windows:  None
Arguments

\textit{dir} \hspace{1cm} \text{Is the directory to add to the system include path.}

Default

OFF \hspace{1cm} \text{The default system include path is used.}

Description

This option specifies a directory to add to the system include path. The compiler searches the specified directory for include files after it searches all directories specified by the \texttt{-I} compiler option but before it searches the standard system directories. This option is provided for compatibility with gcc.

Alternate Options

None

\texttt{ivdep-parallel, Qivdep-parallel}

Tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP pragma.
IDE Equivalent

Windows: None
Linux: Optimization > IVDEP Directive Memory Dependency
Mac OS X: None

Architectures

IA-64 architecture

Syntax

Linux: -ivdep-parallel
Mac OS X: None
Windows: /Qivdep-parallel

Arguments

None

Default

OFF

There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.

Description
This option tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP. There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.

**Alternate Options**

None

**iwithprefix**

Append `<dir>` to the prefix passed in by `-iprefix` and put it on the include search path at the end of the include directories.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-iwithprefix<dir>`

Windows: `None`

**Arguments**

None

**Default**

OFF

**Description**

Append `<dir>` to the prefix passed in by `-iprefix` and put it on the include search path at the end of the include directories.

**Alternate Options**
None

*iwithprefixbefore*

Similar to `-iwithprefix` except the include directory is placed in the same place as `-I` command line include directories.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-iwithprefixbefore <dir>`

Windows: None

**Arguments**

None

**Default**

OFF

**Description**

Similar to `-iwithprefix` except the include directory is placed in the same place as `-I` command line include directories.

**Alternate Options**

None

**J**

Sets the default character type to `unsigned`. 
IDE Equivalent

Windows: **Language > Default Char Unsigned**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: `/J`

Arguments

None

Default

OFF

The default character type is signed

Description

This option sets the default character type to unsigned. This option has no effect on character values that are explicitly declared signed. This option sets `_CHAR_UNSIGNED = 1._`

Alternate Options

None

Kc++, TP
Tells the compiler to process all source or unrecognized file types as C++ source files.
This option is deprecated.

IDE Equivalent

Windows: Advanced > Compile As
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Kc++
Windows: /TP

Arguments

None

Default

OFF

The compiler uses default rules for determining whether a file is a C++ source file.
Description

This option tells the compiler to process all source or unrecognized file types as C++ source files.

Alternate Options

None

kernel

Generates code for inclusion in the kernel.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux: -kernel
Mac OS X: None
Windows: None

Arguments

None

Default

OFF

The restrictions on kernel code are not enforced.
Description

This option generates code for inclusion in the kernel. It prevents generation of speculation because support may not be available when the code runs. This option also suppresses software pipelining.

Alternate Options

None

I

Tells the linker to search for a specified library when linking.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -lstring
Windows: None

Arguments

string

Specifies the library (libstring) that the linker should search.
**Default**

**OFF**

The linker searches for standard libraries in standard directories.

**Description**

This option tells the linker to search for a specified library when linking. When resolving references, the linker normally searches for libraries in several standard directories, in directories specified by the `L` option, then in the library specified by the `l` option.

The linker searches and processes libraries and object files in the order they are specified. So, you should specify this option following the last object file it applies to.

**Alternate Options**

None

**See Also**

`L` compiler option

`L`

Tells the linker to search for libraries in a specified directory before searching the standard directories.

**IDE Equivalent**

None

**Architectures**
Syntax

Linux and Mac OS X: `-Ldir`

Windows: None

Arguments

`dir`

Is the name of the directory to search for libraries.

Default

OFF

The linker searches the standard directories for libraries.

Description

This option tells the linker to search for libraries in a specified directory before searching for them in the standard directories.

Alternate Options
None

See Also

1 compiler option

LD

Specifies that a program should be linked as a dynamic-link (DLL) library.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows:       /LD
              /LDd

Arguments

None

Default

OFF

The program is not linked as a dynamic-link (DLL)
library.

Description

This option specifies that a program should be linked as a dynamic-link (DLL) library instead of an executable (.exe) file. You can also specify /LDd, where d indicates a debug version.

Alternate Options

None

link

Passes user-specified options directly to the linker at compile time.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /link

Arguments

None

Default

OFF No user-specified options
are passed directly to the linker.

**Description**

This option passes user-specified options directly to the linker at compile time. All options that appear following `/link` are passed directly to the linker.

**Alternate Options**

None

**See Also**

[Xlinker](https://www.intel.com) compiler option

**m**

Tells the compiler to generate optimized code specialized for the processor that executes your program.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:** `-m[processor]`

**Windows:** None

**Arguments**
processor

Indicates the processor for which code is generated. Possible values are:

**ia32**

Generates code that will run on any Pentium or later processor. Disables any default extended instruction settings, and any previously set extended instruction settings. This value is only available on Linux systems using IA-32 architecture.

**sse**

This is the same as specifying ia32.
**sse2** Generates code for Intel® Streaming SIMD Extensions 2 (Intel® SSE2). This value is only available on Linux systems.

**sse3** Generates code for Intel® Streaming SIMD Extensions 3 (Intel® SSE3).

**ssse3** Generates code for Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3).
sse4.1 Generates code for Intel® Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators.

Default

Linux systems: -msse2
Mac OS X systems using IA-32 architecture: -msse3
Mac OS X systems using Intel® 64 architecture: -mssse3

For more information on the default values, see Arguments above.

Description

This option tells the compiler to generate optimized code specialized for the processor that executes your program.

Code generated with the values ia32, sse, sse2 or sse3 should execute on any compatible non-Intel processor with support for the corresponding instruction set.

Alternate Options

Linux and Mac OS X: None
Windows: /arch
See Also

\texttt{x}, \texttt{Qx} compiler option
\texttt{ax, Qax} compiler option
\texttt{arch} compiler option

\textbf{M, QM}

Tells the compiler to generate makefile dependency lines for each source file.

\textbf{IDE Equivalent}

None

\textbf{Architectures}

IA-32, Intel® 64, IA-64 architectures

\textbf{Syntax}

Linux and Mac OS X: \texttt{-M}

Windows: \texttt{/QM}

\textbf{Arguments}

None

\textbf{Default}

OFF

The compiler does not generate makefile dependency lines for each source file.
Description

This option tells the compiler to generate makefile dependency lines for each source file, based on the #include lines found in the source file.

Alternate Options

None

m32, m64

Tells the compiler to generate code for a specific architecture.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: `-m32`  `-m64`

Windows: None

Arguments

None

Default

OFF

The compiler's behavior depends on the host
Description

These options tell the compiler to generate code for a specific architecture.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-m32</td>
<td>Tells the compiler to generate code for IA-32 architecture.</td>
</tr>
<tr>
<td>-m64</td>
<td>Tells the compiler to generate code for Intel® 64 architecture.</td>
</tr>
</tbody>
</table>

The `-m32` and `-m64` options are the same as Mac OS* X options `-arch i386` and `-arch x86_64`, respectively. Note that these options are provided for compatibility with gcc. They are not related to the Intel® C++ compiler option `arch`.

Alternate Options

None

m32, m64

Tells the compiler to generate code for a specific architecture.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:  
- `-m32`  
- `-m64`
Windows: None

Arguments

None

Default

OFF

The compiler's behavior depends on the host system.

Description

These options tell the compiler to generate code for a specific architecture.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-m32</td>
<td>Tells the compiler to generate code for IA-32 architecture.</td>
</tr>
<tr>
<td>-m64</td>
<td>Tells the compiler to generate code for Intel® 64 architecture.</td>
</tr>
</tbody>
</table>

The -m32 and -m64 options are the same as Mac OS* X options -arch i386 and -arch x86_64, respectively. Note that these options are provided for compatibility with gcc. They are not related to the Intel® C++ compiler option arch.

Alternate Options

None
**malign-double**

Aligns `double`, `long double`, and `long long` types for better performance for systems based on IA-32 architecture.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X: `-malign-double`

Windows: None

**Arguments**

None

**Default**

OFF

**Description**

Aligns `double`, `long double`, and `long long` types for better performance for systems based on IA-32 architecture.

**Alternate Options**

- `-align`

**malign-mac68k**

Aligns structure fields on 2-byte boundaries (m68k compatible).

**IDE Equivalent**
None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux: None
Mac OS X: -malign-mac68k
Windows: None

Arguments

None

Default

OFF The compiler does not align structure fields on 2-byte boundaries.

Description

This option aligns structure fields on 2-byte boundaries (m68k compatible).

Alternate Options

None

malign-natural

Aligns larger types on natural size-based boundaries (overrides ABI).
IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux: None
Mac OS X: -malign-natural
Windows: None

Arguments

None

Default

OFF

The compiler does not align larger types on natural size-based boundaries.

Description

This option aligns larger types on natural size-based boundaries (overrides ABI).

Alternate Options

None

malign-power
Aligns based on ABI-specified alignment rules.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux: None

Mac OS X: -malign-power

Windows: None

**Arguments**

None

**Default**

ON

The compiler aligns based on ABI-specified alignment rules.

**Description**

Aligns based on ABI-specified alignment rules.

**Alternate Options**

None
map-opts, Qmap-opts

Maps one or more compiler options to their equivalent on a different operating system.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux:      -map-opts
Mac OS X:   None
Windows:    /Qmap-opts

Arguments

None

Default

OFF  No platform mappings are performed.

Description

This option maps one or more compiler options to their equivalent on a different operating system. The result is output to stdout. On Windows systems, the options you provide are presumed to be Windows options, so the options that are output to stdout will be Linux equivalents.
On Linux systems, the options you provide are presumed to be Linux options, so the options that are output to `stdout` will be Windows equivalents. The tool can be invoked from the compiler command line or it can be used directly. No compilation is performed when the option mapping tool is used. This option is useful if you have both compilers and want to convert scripts or makefiles.

**Note**

Compiler options are mapped to their equivalent on the architecture you are using. For example, if you are using a processor with IA-32 architecture, you will only see equivalent options that are available on processors with IA-32 architecture.

**Alternate Options**

None

**Example**

The following command line invokes the option mapping tool, which maps the Linux options to Windows-based options, and then outputs the results to `stdout`:

```
icc -map-opts -xP -O2
```

The following command line invokes the option mapping tool, which maps the Windows options to Linux-based options, and then outputs the results to `stdout`:

```
icl /Qmap-opts /QxP /O2
```

**See Also**

Building Applications: Using the Option Mapping Tool

*march*
Tells the compiler to generate code for a specified processor.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux: `-march=processor`

Mac OS X: None

Windows: None

**Arguments**

`processor` Is the processor for which the compiler should generate code. Possible values are:

- **pentium3** Generates code for Intel® Pentium® III processors.
- **pentium4** Generates code for Intel® Pentium® 4 processors.
Generates code for the Intel® Core 2™ processor family.

Default

OFF or -march=pentium4

On IA-32 architecture, the compiler does not generate processor-specific code unless it is told to do so. On systems using Intel® 64 architecture, the compiler generates code for Intel Pentium 4 processors.
This option tells the compiler to generate code for a specified processor. Specifying `-march=pentium4` sets `-mtune=pentium4`. For compatibility, a number of historical processor values are also supported, but the generated code will not differ from the default.

Alternate Options

None

**mcmode**

Tells the compiler to use a specific memory model to generate code and store data.

IDE Equivalent

None

Architectures

Intel® 64 architecture

Syntax

**Linux:**

- `-mcmode=mem_model`

**Mac OS X:**

None

**Windows:**

None

Arguments

`mem_model` Is the memory model to use.

Possible values are:

`small` Tells the compiler to restrict
code and data to the first 2GB of address space. All accesses of code and data can be done with Instruction Pointer (IP)-relative addressing.

medium Tells the compiler to restrict code to the first 2GB; it places no memory restriction on data. Accesses of code can be done with IP-relative addressing, but
accesses of data must be done with absolute addressing.

large

Places no memory restriction on code or data. All accesses of code and data must be done with absolute addressing.

Default

.mcmodel=small

On systems using Intel® 64 architecture, the compiler restricts code and data to the first 2GB of

address space. Instruction Pointer (IP)-relative addressing can be used to access code and data.

Description
This option tells the compiler to use a specific memory model to generate code and store data. It can affect code size and performance. If your program has global and static data with a total size smaller than 2GB, \texttt{-mcmodel=small} is sufficient. Global and static data larger than 2GB requires \texttt{-mcmodel=medium} or \texttt{-mcmodel=large}. Allocation of memory larger than 2GB can be done with any setting of \texttt{-mcmodel}.

IP-relative addressing requires only 32 bits, whereas absolute addressing requires 64-bits. IP-relative addressing is somewhat faster. So, the small memory model has the least impact on performance.

\begin{quote}
\textbf{Note}
When you specify \texttt{-mcmodel=medium} or \texttt{-mcmodel=large}, you must also specify compiler option \texttt{-shared-intel} to ensure that the correct dynamic versions of the Intel run-time libraries are used.
\end{quote}

Alternate Options

None

Example

486
The following example shows how to compile using `-mcmodel`:

```
icl -shared-intel -mcmodel=medium -o prog prog.c
```

See Also

- `shared-intel` compiler option
- `fpic` compiler option
- `mcpu`

This is a deprecated option. See `mtune`.

**MD, QMD**

Preprocess and compile, generating output file containing dependency information ending with extension `.d`.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-MD`

Windows: `/QMD`

**Arguments**

None

**Default**

OFF

The compiler does not generate
dependency information.

Description
Preprocess and compile, generating output file containing dependency information ending with extension .d.

Alternate Options
None

MD
Tells the linker to search for unresolved references in a multithreaded, debug, dynamic-link run-time library.

IDE Equivalent
Windows: Code Generation > Runtime Library
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Windows:
/MD
/MDd

Arguments
None

Default
OFF

The linker searches for unresolved references in a single-threaded, static run-time library.

Description

This option tells the linker to search for unresolved references in a multithreaded, debug, dynamic-link (DLL) run-time library. You can also specify `/MDd`, where `d` indicates a debug version.

Alternate Options

None

`mdynamic-no-pic`

Generates code that is not position-independent but has position-independent external references.

IDE Equivalent

None

Architectures

IA-32 architecture

Syntax

Linux: None
Mac OS X: -mdynamic-no-pic
Windows: None

Arguments
None

Default
OFF

All references are generated as position independent.

Description
This option generates code that is not position-independent but has position-independent external references. The generated code is suitable for building executables, but it is not suitable for building shared libraries. This option may reduce code size and produce more efficient code. It overrides the -fpic compiler option.

Alternate Options
None

See Also

fpic compiler option

MF, QMF

Tells the compiler to generate makefile dependency information in a file.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -MFfile
Windows: /QMFfile

Arguments

file

Is the name of the file where the makefile dependency information should be placed.

Default
OFF

The compiler does not generate makefile dependency information in files.

Description
This option tells the compiler to generate makefile dependency information in a file. To use this option, you must also specify /QM or /QMM.

Alternate Options

None

See Also

QM compiler option
QMM compiler option

mfixed-range

Reserves certain registers (f12-f15, f32-f127) for use by the Linux* kernel.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux: -mfixed-range=f12-f15,f32-f127
Mac OS X: None
Windows: None

Arguments

None

Default

OFF

Description

Reserves certain registers (f12-f15,f32-f127) for use by the Linux* kernel.
Alternate Options

None

MG, QMG

Tells the compiler to generate makefile dependency lines for each source file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -MG
Windows: /QMG

Arguments

None

Default

OFF

The compiler does not generate makefile dependency information in files.

Description
This option tells the compiler to generate makefile dependency lines for each source file. It is similar to `/QM`, but it treats missing header files as generated files.

**Alternate Options**

None

**See Also**

/QM compiler option

**minstruction, Qinstruction**

Determines whether MOVBE instructions are generated for Intel processors.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X: `-minstruction=[no]movbe`

Windows: `/Qinstruction:[no]movbe`

**Arguments**

None

**Default**

- `-minstruction=movbe` or `/Qinstruction:movbe`

  The compiler generates MOVBE
instructions
for Intel®
Atom™
processors.

Description

This option determines whether MOVBE instructions are generated for Intel processors. To use this option, you must also specify -xSSE3_ATOM (Linux and Mac OS X) or /QxSSE3_ATOM (Windows).

If -minstruction=movbe or /Qinstruction:movbe is specified, the following occurs:

- MOVBE instructions are generated that are specific to the Intel® Atom™ processor.
- The options are ON by default when -xSSE3_ATOM or /QxSSE3_ATOM is specified.
- Generated executables can only be run on Intel® Atom™ processors or processors that support Intel® Streaming SIMD Extensions 3 (Intel® SSE3) and MOVBE.

If -minstruction=nomovbe or /Qinstruction:nomovbe is specified, the following occurs:

- The compiler optimizes code for the Intel® Atom™ processor, but it does not generate MOVBE instructions.
- Generated executables can be run on non-Intel® Atom™ processors that support Intel® SSE3.

Alternate Options

None

See Also

x, Qx compiler option
ML

Tells the linker to search for unresolved references in a single-threaded, static run-time library.
This option has been deprecated.

IDE Equivalent

Windows: Code Generation > Runtime Library
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /ML
            /MLd

Arguments

None

Default

Systems using Intel® 64 architecture: OFF
Systems using IA-32 architecture and IA-64 architecture: /ML.
On systems using Intel® 64 architecture, the linker searches for unresolved references in
a multithreaded, static run-time library. On systems using IA-32 architecture and IA-64 architectures, the linker searches for unresolved references in a single-threaded, static run-time library.

**Description**

This option tells the linker to search for unresolved references in a single-threaded, static run-time library. You can also specify `/MLd`, where `d` indicates a debug version.

**Alternate Options**

None

**MM, QMM**

Tells the compiler to generate makefile dependency lines for each source file.

**IDE Equivalent**

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -MM
Windows: /QMM

Arguments

None

Default

OFF

The compiler does not generate makefile dependency information in files.

Description

This option tells the compiler to generate makefile dependency lines for each source file. It is similar to /QM, but it does not include system header files.

Alternate Options

None

See Also

/QM compiler option

MMD, QMMD
Tells the compiler to generate an output file containing dependency information.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-MMD`

Windows: `/QMMD`

**Arguments**

None

**Default**

OFF

The compiler does not generate an output file containing dependency information.

**Description**

This option tells the compiler to preprocess and compile a file, then generate an output file (with extension `.d`) containing dependency information. It is similar to `/QMD`, but it does not include system header files.

**Alternate Options**
None

**mp**

Maintains floating-point precision while disabling some optimizations. This is a deprecated option.

**IDE Equivalent**

Windows: None

Linux: [Floating Point > Improve Floating-point Consistency](#)

Mac OS X: [Floating Point > Improve Floating-point Consistency](#)

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-mp`

Windows: None

**Arguments**

None

**Default**

OFF

The compiler provides good accuracy and run-time performance at the
Description

This option maintains floating-point precision while disabling some optimizations. It restricts optimization to maintain declared precision and to ensure that floating-point arithmetic conforms more closely to the ANSI* language and IEEE* arithmetic standards.

For most programs, specifying this option adversely affects performance. If you are not sure whether your application needs this option, try compiling and running your program both with and without it to evaluate the effects on both performance and precision.

The recommended method to control the semantics of floating-point calculations is to use option \texttt{-fp-model}.

Alternate Options

Linux and Mac OS X: \texttt{-mieee-fp}

Windows: None

See Also

\texttt{mpl,Qprec} compiler option
\texttt{fp-model,fp} compiler option

MP

Tells the compiler to add a phony target for each dependency.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  \texttt{--MP}
Windows: \texttt{None (see below)}

Arguments
None

Default
OFF
The compiler does not generate dependency information unless it is told to do so.

Description
This option tells the compiler to add a phony target for each dependency. Note that this option is not related to Windows* option \texttt{/MP}.

Alternate Options
None

\texttt{multiple-processes, MP}
Creates multiple processes that can be used to compile large numbers of source files at the same time.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -multiple-processes[=n]
Windows: /MP[=n]

Arguments

$n$

Is the maximum number of processes that the compiler should create.

Default

OFF

A single process is used to compile
source files.

Description

This option creates multiple processes that can be used to compile large numbers of source files at the same time. It can improve performance by reducing the time it takes to compile source files on the command line. This option causes the compiler to create one or more copies of itself, each in a separate process. These copies simultaneously compile the source files.

If $n$ is not specified for this option, the default value is as follows:

- **On Windows OS, the value is based on the setting of the NUMBER_OF_PROCESSORS environment variable.**
- **On Linux OS and Mac OS X, the value is 2.**

This option applies to compilations, but not to linking or link-time code generation.

Alternate Options

None

**mp1, Qprec**

Improves floating-point precision and consistency.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
Windows:  

504
Arguments

None

Default

OFF

The compiler provides good accuracy and run-time performance at the expense of less consistent floating-point results.

Description

This option improves floating-point consistency. It ensures the out-of-range check of operands of transcendental functions and improves the accuracy of floating-point compares.

This option prevents the compiler from performing optimizations that change NaN comparison semantics and causes all values to be truncated to declared precision before they are used in comparisons. It also causes the compiler to use library routines that give better precision results compared to the X87 transcendental instructions.

This option disables fewer optimizations and has less impact on performance than option mp or Op.
Alternate Options

None

See Also

mp compiler option

MQ

Changes the default target rule for dependency generation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -MQ target
Windows: None

Arguments

target

Is the target rule to use.

Default

OFF

The default target rule applies to dependency
Description

This option changes the default target rule for dependency generation. It is similar to `-MT`, but quotes special Make characters.

Alternate Options

None

mregparm

Control the number registers used to pass integer arguments.

IDE Equivalent

Windows: None
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `--mregparm=value`
Windows: `None`

Arguments

None

Default

OFF

The compiler
does not use registers to pass arguments.

**Description**

Control the number registers used to pass integer arguments.

**Alternate Options**

None

**mrelax**

Tells the compiler to pass linker option `-relax` to the linker.

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

Linux and Mac OS X: `-mrelax`

Windows: None

**Arguments**

None
-mno-relax

The compiler does not pass -relax to the linker.

Description

This option tells the compiler to pass linker option -relax to the linker.

Alternate Options

None

mserialize-volatile, Qserialize-volatile

Determines whether strict memory access ordering is imposed for volatile data object references.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:  
-mserialize-volatile
-mno-serialize-volatile

Mac OS X:  
None

Windows:  
/Qserialize-volatile
/Qserialize-volatile-
Arguments

None

Default

-mno-serialize-volatile or /Qserialize-volatile-
The compiler uses default memory access ordering.

Description

This option determines whether strict memory access ordering is imposed for volatile data object references. If you specify -mno-serialize-volatile, the compiler may suppress both run-time and compile-time memory access ordering for volatile data object references. Specifically, the .rel/.acq completers will not be issued on referencing loads and stores.

Alternate Options

None

MT, QMT

Changes the default target rule for dependency generation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:  `-MT target`

Windows:  `/QMT target`

Arguments

`target`  
Is the target rule to use.

Default

OFF  
The default target rule applies to dependency generation.

Description

This option changes the default target rule for dependency generation.

Alternate Options

None

MT

Tells the linker to search for unresolved references in a multithreaded, static run-time library.

IDE Equivalent

Windows: **Code Generation > Runtime Library**

Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Windows: /MT
/MTd

Arguments
None

Default
Systems using Intel® 64 architecture: /MT
IA-32 architecture and IA-64 architecture: OFF

On systems using Intel® 64 architecture, the linker searches for unresolved references in a multithreaded, static run-time library. On systems using IA-32 architecture and IA-64...
architecture, the linker searches for unresolved references in a single-threaded, static run-time library. However, on systems using IA-32 architecture, if option Qvc8 is in effect, the linker searches for unresolved references in threaded libraries.

Description

This option tells the linker to search for unresolved references in a multithreaded, static run-time library. You can also specify /MTd, where d indicates a debug version.

Alternate Options

None

See Also
Qvc compiler option

mtune

Performs optimizations for specific processors.

IDE Equivalent

Windows: None
Linux: **Optimization > Optimize for Intel processor**
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: ```-mtune=processor```
Windows: None

Arguments

**processor**

Is the processor for which the compiler should perform optimizations.

Possible values are:

- **generic**: Generates code for the compiler's default behavior.
- **core2**: Optimizes for the Intel® Core
2 processor family, including support for MMX™, Intel® SSE, SSE2, SSE3 and SSSE3 instruction sets.

**pentium** Optimizes for Intel® Pentium® processors.

**pentium-mmx** Optimizes for Intel® Pentium® with MMX technology.

**pentiumpro** Optimizes for Intel® Pentium® Pro, Intel Pentium II, and Intel Pentium III processors.

**pentium4** Optimizes
for Intel® Pentium® 4 processors.

`pentium4m` Optimizes for Intel® Pentium® 4 processors with MMX technology.

`itanium2` Optimizes for Intel® Itanium® 2 processors.

`itanium2-p9000` Optimizes for the Dual-Core Intel® Itanium® 2 processor 9000 series. This option affects the order of the generated instructions, but the generated instructions
are limited to Intel® Itanium® 2 processor instructions unless the program uses (executes) intrinsics specific to the Dual-Core Intel® Itanium® 2 processor 9000 series.

Default
generic

On systems using IA-32 and Intel® 64 architectures, code is generated for the compiler's default behavior.
itanium2-p9000

On systems using IA-64 architecture, the compiler optimizes for the Dual-Core Intel® Itanium® 2 processor 9000 series.

Description

This option performs optimizations for specific processors. The resulting executable is backwards compatible and generated code is optimized for specific processors. For example, code generated with `-mtune=itanium2-p9000` will run correctly on single-core Itanium® 2 processors, but it might not run as fast as if it had been generated using `-mtune=itanium2`.

The following table shows on which architecture you can use each value.

<table>
<thead>
<tr>
<th>processor Value</th>
<th>Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IA-32 architecture</td>
</tr>
<tr>
<td>generic</td>
<td>X</td>
</tr>
<tr>
<td>core2</td>
<td>X</td>
</tr>
<tr>
<td>pentium</td>
<td>X</td>
</tr>
<tr>
<td>pentium-mmx</td>
<td>X</td>
</tr>
<tr>
<td>pentiumpro</td>
<td>X</td>
</tr>
<tr>
<td>processor Value</td>
<td>Architecture</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>pentium4</td>
<td>X</td>
</tr>
<tr>
<td>pentium4m</td>
<td>X</td>
</tr>
<tr>
<td>itanium2</td>
<td></td>
</tr>
<tr>
<td>itanium2-p9000</td>
<td></td>
</tr>
</tbody>
</table>

**Alternate Options**

- **-mtune**
  - Linux: `--mcpu` (this is a deprecated option)
  - Mac OS X: None
  - Windows: None

- **-mtune=itanium2**
  - Linux: `-mcpu=itanium2` (-`mcpu` is a deprecated option)
  - Mac OS X: None
  - Windows: `/G2`

- **-mtune=itanium2-p9000**
  - Linux: `-mcpu=itanium2-p9000` (-`mcpu` is a deprecated option)
  - Mac OS X: None
  - Windows: `/G2-p9000`

**multibyte-chars, Qmultibyte-chars**

Determines whether multi-byte characters are supported.

**IDE Equivalent**

Windows: None
Linux: **Language > Support Multibyte Characters in Source**
Mac OS X: **Language > Support Multibyte Characters in Source**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: 
```
-multibyte-chars
-no-multibyte-chars
```

Windows: 
```
/Qmultibyte-chars
/Qmultibyte-chars-
```

**Arguments**
None

**Default**
```
-multibyte-chars or /Qmultibyte-chars
```

Multi-byte characters are supported.

**Description**
This option determines whether multi-byte characters are supported.

**Alternate Options**
None

**multiple-processes, MP**
Creates multiple processes that can be used to compile large numbers of source files at the same time.
IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-multiple-processes[=n]`
Windows: `/MP[=n]`

Arguments

$n$

Is the maximum number of processes that the compiler should create.

Default

OFF

A single process is used to compile source files.
This option creates multiple processes that can be used to compile large numbers of source files at the same time. It can improve performance by reducing the time it takes to compile source files on the command line.

This option causes the compiler to create one or more copies of itself, each in a separate process. These copies simultaneously compile the source files.

If \( n \) is not specified for this option, the default value is as follows:

- **On Windows OS, the value is based on the setting of the \texttt{NUMBER_OF_PROCESSORS} environment variable.**
- **On Linux OS and Mac OS X, the value is 2.**

This option applies to compilations, but not to linking or link-time code generation.

**Alternate Options**

None

**noBool**

Disables the \texttt{bool} keyword.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None
Windows: /noBool

**Arguments**

None

**Default**

522
OFF

The `bool` keyword is enabled.

Description

This option disables the `bool` keyword.

Alternate Options

None

`no-bss-init, Qnobss-init`

Tells the compiler to place in the DATA section any variables explicitly initialized with zeros.

IDE Equivalent

Windows: None

Linux: Data > Disable Placement of Zero-initialized Variables in .bss - use .data

Mac OS X: Data > Allocate Zero-initialized Variables to .data

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-no-bss-init`

Windows: `/Qnobss-init`

Arguments

None
Default

OFF

Variables explicitly initialized with zeros are placed in the BSS section.

Description

This option tells the compiler to place in the DATA section any variables explicitly initialized with zeros.

Alternate Options

Linux and Mac OS X: -nobss-init (this is a deprecated option)
Windows: None

nodefaultlibs

Prevents the compiler from using standard libraries when linking.

IDE Equivalent

Windows: None
Linux: Libraries > Use no system libraries
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -nodefaultlibs
Windows: None

Arguments
None

Default
OFF

The standard libraries are linked.

Description
This option prevents the compiler from using standard libraries when linking. It is provided for GNU compatibility.

Alternate Options
None

See Also
nolistlib compiler option

nolib-inline

Disables inline expansion of standard library or intrinsic functions.

IDE Equivalent
Windows: None
Linux: **Optimization > Disable Intrinsic Inline Expansion**
Mac OS X: **Optimization > Disable Intrinsic Inline Expansion**

Architectures
Syntax

Linux and Mac OS X: -nolib-inline
Windows: None

Arguments

None

Default

OFF

The compiler inlines many standard library and intrinsic functions.

Description

This option disables inline expansion of standard library or intrinsic functions. It prevents the unexpected results that can arise from inline expansion of these functions.

Alternate Options

None

logo

Do not display compiler version information.
IDE Equivalent

Windows: **General > Suppress Startup Banner**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /nologo

Arguments

None

Default

OFF

Description

Do not display compiler version information.

Alternate Options

None

**nostartfiles**

Prevents the compiler from using standard startup files when linking.

IDE Equivalent

None

Architectures
Syntax

Linux and Mac OS X: -nostartfiles
Windows: None

Arguments
None

Default

OFF

The compiler uses standard startup files when linking.

Description

This option prevents the compiler from using standard startup files when linking.

Alternate Options
None

See Also

noldlib compiler option

nostdinc++

Do not search for header files in the standard directories for C++, but search the other standard directories.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -nostdinc++
Windows: None

Arguments
None

Default
OFF

Description
Do not search for header files in the standard directories for C++, but search the other standard directories.

Alternate Options
None

nostdlib
Prevents the compiler from using standard libraries and startup files when linking.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -nostdlib
Windows: None

Arguments

None

Default

OFF The compiler uses standard startup files and standard libraries when linking.

Description

This option prevents the compiler from using standard libraries and startup files when linking. It is provided for GNU compatibility.

Alternate Options

None

See Also

nodefaultlibs compiler option
nostartfiles compiler option

530
Specifies the name for an output file.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `–o file`

Windows: None

**Arguments**

`file`

Is the name for the output file. The space before `file` is optional.

**Default**

OFF

The compiler uses the default file
name for an output file.

Description

This option specifies the name for an output file as follows:

- If `-c` is specified, it specifies the name of the generated object file.
- If `-S` is specified, it specifies the name of the generated assembly listing file.
- If `-P` is specified, it specifies the name of the generated preprocessor file.

Otherwise, it specifies the name of the executable file.

Note

If you misspell a compiler option beginning with "-o", such as `-openmp`, `-opt-report`, etc., the compiler interprets the misspelled option as an `-o file` option. For example, say you misspell "-opt-report" as "-opt-reprt"; in this case, the compiler interprets the misspelled option as "-o pt-reprt", where pt-reprt is the output file name.

Alternate Options

Linux and Mac OS X: None

Windows: /Fe

See Also

-Fe compiler option

O

Specifies the code optimization for applications.
IDE Equivalent

Windows: Optimization > Optimization
Linux: General > Optimization Level
Mac OS X: General > Optimization Level

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -O[n]
Windows: /O[n]

Arguments

\( n \)

Is the optimization level. Possible values are 1, 2, or 3. On Linux and Mac OS X systems, you can also specify 0.

Default

02

Optimizes
for code speed. This default may change depending on which other compiler options are specified. For details, see below.

Description

This option specifies the code optimization for applications.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>O (Linux and Mac OS X)</td>
<td>This is the same as specifying O2.</td>
</tr>
<tr>
<td>O0 (Linux and Mac OS X)</td>
<td>Disables all optimizations. On systems using IA-32 architecture and Intel® 64 architecture, this option sets option -fno-omit-frame-pointer and option -fmath-errno. Option -O0 also implies option -mp. So, intermediate floating-point results are evaluated at extended precision.</td>
</tr>
</tbody>
</table>
### Option Description

precision. On IA-32 or Intel® 64 architecture, this may cause the compiler to use x87 instructions instead of SSE instructions. You can use option `-fp-model` to independently control the evaluation precision for intermediate results.

Option `-mp` is deprecated. Therefore, the default behavior of `-O0` may change in a future compiler release.

**-O1** Enables optimizations for speed and disables some optimizations that increase code size and affect speed.

To limit code size, this option:

- Enables global optimization; this includes data-flow analysis, code motion, strength reduction and test replacement, split-lifetime analysis, and instruction scheduling.
- Disables intrinsic recognition and intrinsics inlining.
- On systems using IA-64 architecture, it disables software pipelining, loop unrolling, and global code scheduling.

On systems using IA-64 architecture, this option also enables optimizations for server applications (straight-line and branch-like code with a flat profile). The **-O1** option sets the following options:

- On Linux and Mac OS X systems using IA-32 architecture and Intel® 64 architecture:
  
  ```bash
  -funroll-loops0, -fno-builtins, -mno-ieee-fp, -fomit-frame-pointer, -ffunction-sections, -ftz
  ```
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• On Linux systems using IA-64 architecture:</td>
</tr>
<tr>
<td></td>
<td>-funroll-loops0, -fbuiltin, -mno-ieee-fp,</td>
</tr>
<tr>
<td></td>
<td>-fomit-frame-pointer, -ffunction-sections, -fno-</td>
</tr>
<tr>
<td></td>
<td>• On Windows systems using IA-32 architecture:</td>
</tr>
<tr>
<td></td>
<td>/Qunroll0, /Oi-, /Op-, /Oy, /Gy, /Os, /GF (/Qvc7 and above), /Gf (/Qvc6 and below), /Ob2, /Og, /Qftz</td>
</tr>
<tr>
<td></td>
<td>• On Windows systems using Intel® 64 architecture and IA-64 architecture:</td>
</tr>
<tr>
<td></td>
<td>/Qunroll0, /Oi-, /Op-, /Gy, /Os, /GF (/Qvc7 and above), /Gf (/Qvc6 and below), /Ob2, /Og, /Qftz</td>
</tr>
</tbody>
</table>

The O1 option may improve performance for applications with very large code size, many branches, and execution time not dominated by code within loops.

**O2**

Enables optimizations for speed. This is the generally recommended optimization level.
Vectorization is enabled at O2 and higher levels.
On systems using IA-64 architecture, this option enables optimizations for speed, including global code scheduling, software pipelining, predication, and speculation.
This option also enables:

• Inlining of intrinsics
• Intra-file interprocedural optimization, which includes:
  • inlining
Option | Description
---|---
| constant propagation
| forward substitution
| routine attribute propagation
| variable address-taken analysis
| dead static function elimination
| removal of unreferenced variables

- The following capabilities for performance gain:
  | constant propagation
  | copy propagation
  | dead-code elimination
  | global register allocation
  | global instruction scheduling and control speculation
  | loop unrolling
  | optimized code selection
  | partial redundancy elimination
  | strength reduction/induction variable simplification
  | variable renaming
  | exception handling optimizations
  | tail recursions
  | peephole optimizations
  | structure assignment lowering and optimizations
  | dead store elimination

The **O2** option sets the following options:

- **On Windows systems using IA-32 architecture:**
  
  `/Og, /Oi-, /Os, /Oy, /Ob2, /GF (/Qvc7 and above), /Gf (/Qvc6 and below), /Gs, /Gy, and /Qftz`

- **On Windows systems using Intel® 64 architecture:**
  
  `/Og, /Oi-, /Os, /Ob2, /Gf (/Qvc7 and above),`
**Option** | **Description**
--- | ---
/Gf (/Qvc6 and below), /Gs, /Gy, and /Qftz

This option sets other options that optimize for code speed. The options set are determined by the compiler depending on which architecture and operating system you are using.

**O3**

Enables O2 optimizations plus more aggressive optimizations, such as prefetching, scalar replacement, and loop and memory access transformations. Enables optimizations for maximum speed, such as:

- Loop unrolling, including instruction scheduling
- Code replication to eliminate branches
- Padding the size of certain power-of-two arrays to allow more efficient cache use.

On Windows systems, the O3 option sets the /GF (/Qvc7 and above), /Gf (/Qvc6 and below), and /Ob2 option.

On Linux and Mac OS X systems, the O3 option sets option -fomit-frame-pointer.

On systems using IA-32 architecture or Intel® 64 architecture, when O3 is used with options -ax or -x (Linux) or with options /Qax or /Qx (Windows), the compiler performs more aggressive data dependency analysis than for O2, which may result in longer compilation times.

On systems using IA-64 architecture, the O3 option enables optimizations for technical computing applications (loop-intensive code): loop optimizations and data prefetch.
The **O3** optimizations may not cause higher performance unless loop and memory access transformations take place. The optimizations may slow down code in some cases compared to **O2** optimizations.

The **O3** option is recommended for applications that have loops that heavily use floating-point calculations and process large data sets.

The last **O** option specified on the command line takes precedence over any others.

**Note**

The options set by the **O** option may change from release to release.

Alternate Options

**O1**

Linux and Mac OS

X: None

Windows:

/0d

**See Also**

/od compiler option

Op compiler option

fp-model, fp compiler option

Oa compiler option

**See Also**

**Oa**

Tells the compiler to assume there is no aliasing.
IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Oa
/Oa-

Arguments

None

Default

OFF

The compiler assumes there is aliasing.

Description

This option tells the compiler to assume there is no aliasing.

Alternate Options

None

inline-level, Ob

Specifies the level of inline function expansion.

IDE Equivalent

540
Windows: **Optimization > Inline Function Expansion**
Linux: **Optimization > Inline Function Expansion**
Mac OS X: **Optimization > Inline Function Expansion**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X: `-inline-level=n`
Windows: `/Obn`

**Arguments**

\[n\]

Is the inline function expansion level. Possible values are 0, 1, and 2.

**Default**

`-inline-level=2` or `/Ob2`

This is the default if option `O2` is specified or is in effect by
default. On Windows systems, this is also the default if option `-O3` is specified.

-`-inline-level=0` or `/Ob0`

This is the default if option `-O0` (Linux and Mac OS) or `/Od` (Windows) is specified.

**Description**

This option specifies the level of inline function expansion. Inlining procedures can greatly improve the run-time performance of certain programs.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-inline-level=0</code> or <code>/Ob0</code></td>
<td>Disables inlining of user-defined functions. Note that statement functions are always inlined.</td>
</tr>
<tr>
<td><code>-inline-level=1</code> or <code>/Ob1</code></td>
<td>Enables inlining when an inline keyword or an inline attribute is specified. Also enables inlining according to the C++ language.</td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-inline-level=2</td>
<td>Enables inlining of any function at the compiler's discretion.</td>
</tr>
<tr>
<td>-inline-level=2</td>
<td>or Ob2</td>
</tr>
</tbody>
</table>

**Alternate Options**

Linux: `-Ob` (this is a deprecated option)

Mac OS X: None

Windows: None

**Od**

Disables all optimizations.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None

Windows: `/Od`

**Arguments**

None

**Default**

OFF  The compiler performs default optimizations.
Description

This option disables all optimizations. It can be used for selective optimizations, such as a combination of /Od and /Og (disables all optimizations except global optimizations), or /Od and /Ob1 (disables all optimizations, but enables inlining). On IA-32 architecture, this option sets the /Oy- option.

Option /Od also implies option /Op. So, intermediate floating-point results are evaluated at extended precision. On IA-32 and Intel® 64 architecture, this may cause the compiler to use x87 instructions instead of SSE instructions. You can use option /fp to independently control the evaluation precision for intermediate results.

Option /Op is deprecated. Therefore, the default behavior of /Od may change in a future compiler release.

Alternate Options

Linux and Mac OS X: -O0
Windows: None

See Also

O compiler option (see O0)
fp-model, fp compiler option

Og

Enables global optimizations.

IDE Equivalent

Windows: Optimization > Global Optimization
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: None
Windows: /Og
         /Og-

Arguments

None

Default

/Og

Global optimizations are enabled unless /Od is specified.

Description

This option enables global optimizations.

Alternate Options

None

Oi

Enables inline expansion of intrinsic functions.

IDE Equivalent

Windows: Optimization > Enable Intrinsic Functions
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Oi

Arguments

None

Default

ON

Description

This option enables inline expansion of intrinsic functions. If you specify /Oi-, it disables inlining of all intrinsic functions.

Alternate Options

None

Op

Enables conformance to the ANSI C and IEEE 754 standards for floating-point arithmetic. This is a deprecated option.

IDE Equivalent

Windows: Optimization > Floating-point Precision Improvement
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None

Windows: 

/Op

/Op-

Arguments

None

Default

OFF

Description

This option enables conformance to the ANSI C and IEEE 754 standards for floating-point arithmetic.

It restricts some optimizations to maintain declared precision and to ensure that floating-point arithmetic conforms more closely to the ANSI and IEEE standards. Floating point intermediate results are kept in full 10-byte internal precision. All spills and reloads of the x87 floating-point registers utilize this internal format to prevent accidental loss of precision.

For most programs, specifying this option adversely affects performance. If you are not sure whether your application needs this option, try compiling and running your program both with and without it to evaluate the effects on performance versus precision. Alternatives to /Op include /QxN (for the Intel® Pentium™ 4 processor or newer) and /Qprec.

Specifying the /Op option has the following effects on program compilation:
User variables declared as floating-point types are not assigned to registers.

Whenever an expression is spilled (moved from a register to memory), it is spilled as 80 bits (extended precision), not 64 bits (double precision).

Floating-point arithmetic comparisons conform to the IEEE 754 specification except for NaN behavior.

The exact operations specified in the code are performed. For example, division is never changed to multiplication by the reciprocal.

The compiler performs floating-point operations in the order specified without re-association.

The compiler does not perform the constant-folding optimization on floating-point values. Constant folding also eliminates any multiplication by 1, division by 1, and addition or subtraction of 0. For example, code that adds 0.0 to a number is executed exactly as written. Compile-time floating-point arithmetic is not performed to ensure that floating-point exceptions are also maintained.

Floating-point operations conform to ANSI C. When assignments to type float and double are made, the precision is rounded from 80 bits (extended) down to 32 bits (float) or 64 bits (double). When you do not specify /Op, the extra bits of precision are not always rounded before the variable is reused.

It sets the /Oi- option, which disables inline functions expansion.

The recommended method to control the semantics of floating-point calculations is to use option /fp.

Alternate Options

None

See Also

fp-model, fp compiler options

openmp, Qopenmp
Enables the parallelizer to generate multi-threaded code based on the OpenMP* directives.

**IDE Equivalent**

Windows: **Language > OpenMP* Support**  
Linux: **Language > Process OpenMP Directives**  
Mac OS X: **Language > Process OpenMP Directives**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-openmp`  
Windows: `/Qopenmp`

**Arguments**

None

**Default**

OFF

**Description**

No OpenMP multi-threaded code is generated by the compiler.
This option enables the parallelizer to generate multi-threaded code based on the OpenMP* directives. The code can be executed in parallel on both uniprocessor and multiprocessor systems.
This option works with any optimization level. Specifying no optimization (\texttt{-O0} on Linux or \texttt{/Od} on Windows) helps to debug OpenMP applications.

\begin{quote}
\textbf{Note}
On Mac OS X systems, when you enable OpenMP*, you must also set the \texttt{DYLD_LIBRARY_PATH} environment variable within Xcode or an error will be displayed.
\end{quote}

Alternate Options
None

See Also
openmp-stubs, \texttt{Qopenmp-stubs} compiler option

openmp-lib, \texttt{Qopenmp-lib}
Lets you specify an OpenMP* run-time library to use for linking.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
\begin{itemize}
\item Linux: \texttt{-openmp-lib \textit{type}}
\item Mac OS X: None
\item Windows: \texttt{/Qopenmp-lib:\textit{type}}
\end{itemize}

Arguments
type

Specifies the type of library to use; it implies compatibility levels. Possible values are:

- **legacy** Tells the compiler to use the legacy OpenMP* run-time library (libguide). This setting does not provide compatibility with object files created using other compilers. This is a deprecated option.

- **compat** Tells the compiler to use the compatibility OpenMP* run-time
This setting provides compatibility with object files created using Microsoft* and GNU* compilers.

Default

-openmp-lib compat
or/Qopenmp-lib:compat

The compiler uses the compatibility OpenMP* run-time library (libiomp).

Description

This option lets you specify an OpenMP* run-time library to use for linking. The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers. The compatibility OpenMP run-time library is compatible with object files created using the Microsoft* OpenMP run-time library (vcomp) and GNU OpenMP run-time library (libgomp).
To use the compatibility OpenMP run-time library, compile and link your application using the `-openmp-lib compat` (Linux) or `/Qopenmp-lib:compat` (Windows) option. To use this option, you must also specify one of the following compiler options:

- **Linux OS:** `-openmp`, `-openmp-profile`, or `-openmp-stubs`
- **Windows OS:** `/Qopenmp`, `/Qopenmp-profile`, or `/Qopenmp-stubs`

On Windows* systems, the compatibility OpenMP* run-time library lets you combine OpenMP* object files compiled with the Microsoft* C/C++ compiler with OpenMP* object files compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

On Linux* systems, the compatibility Intel OpenMP* run-time library lets you combine OpenMP* object files compiled with the GNU* gcc or gfortran compilers with similar OpenMP* object files compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

**Note**

The compatibility OpenMP run-time library is not compatible with object files created using versions of the Intel compiler earlier than 10.0.

**Alternate Options**

None

**See Also**

- [openmp, Qopenmp](#) compiler option
- [openmp-stubs, Qopenmp-stubs](#) compiler option
- [openmp-profile, Qopenmp-profile](#) compiler option

**openmp-link, Qopenmp-link**

Controls whether the compiler links to static or dynamic OpenMP run-time libraries.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: 

Windows: 

Arguments

Specifies the
OpenMP library to
use. Possible
values are:

static Tells the
compiler
to link to
static
OpenMP
run-time
libraries.

dynamic Tells the
compiler
to link to
dynamic
OpenMP
run-time
libraries.

Default

554
-openmp-link dynamic or /Qopenmp-link:dynamc

The compiler links to dynamic OpenMP run-time libraries. However, if option static is specified, the compiler links to static OpenMP run-time libraries.

Description

This option controls whether the compiler links to static or dynamic OpenMP run-time libraries.

To link to the static OpenMP run-time library (RTL) and create a purely static executable, you must specify -openmp-link static (Linux and Mac OS X) or /Qopenmp-link:static (Windows). However, we strongly recommend you use the default setting, -openmp-link dynamic (Linux and Mac OS X) or /Qopenmp-link:dynamc (Windows).

Note
Compiler options `--static-intel` and `--shared-intel` (Linux and Mac OS X) have no effect on which OpenMP run-time library is linked.

Alternate Options

None

**openmp-profile, Qopenmp-profile**

Enables analysis of OpenMP* applications if Intel® Thread Profiler is installed.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: `--openmp-profile`

Mac OS X: None

Windows: `/Qopenmp-profile`

Arguments

None

Default

OFF

OpenMP applications are not analyzed.

Description
This option enables analysis of OpenMP* applications. To use this option, you must have previously installed Intel® Thread Profiler, which is one of the Intel® Threading Analysis Tools.
This option can adversely affect performance because of the additional profiling and error checking invoked to enable compatibility with the threading tools. Do not use this option unless you plan to use the Intel® Thread Profiler.
For more information about Intel® Thread Profiler (including an evaluation copy) open the page associated with threading tools at Intel® Software Development Products.

Alternate Options

None

openmp-report, Qopenmp-report

Controls the OpenMP* parallelizer's level of diagnostic messages.

IDE Equivalent

Windows: None
Linux: Compilation Diagnostics > OpenMP Report
Mac OS X: Diagnostics > OpenMP Report

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -openmp-report[n]
Windows: /Qopenmp-report[n]

Arguments

$n$ Is the level of diagnostic
messages to display. Possible values are:

0  No diagnostic messages are displayed.

1  Diagnostic messages are displayed indicating loops, regions, and sections successfully parallelized.

2  The same diagnostic messages are displayed as specified by openmp_report1 plus diagnostic messages indicating successful handling of MASTER constructs, SINGLE constructs,
CRITICAL constructs, ORDERED constructs, ATOMIC directives, and so forth.

**Default**

```
-openmp-report1
or/Qopenmp-report1
```

This is the default if you do not specify \( n \). The compiler displays diagnostic messages indicating loops, regions, and sections successfully parallelized. If you do not specify the option on the command
Description

This option controls the OpenMP* parallelizer's level of diagnostic messages. To use this option, you must also specify -openmp (Linux and Mac OS X) or /Qopenmp (Windows).

If this option is specified on the command line, the report is sent to stdout.

Alternate Options

None

See Also

openmp, Qopenmp compiler option
Optimizing Applications:
Using Parallelism
OpenMP* Report

openmp-stubs, Qopenmp-stubs
Enables compilation of OpenMP programs in sequential mode.

IDE Equivalent

Windows: Language > Process OpenMP Directives
Linux: Language > Process OpenMP Directives
Mac OS X: Language > Process OpenMP Directives

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -openmp-stubs
Windows: /Qopenmp-stubs

Arguments

None

Default

OFF

The library of OpenMP function stubs is not linked.

Description

This option enables compilation of OpenMP programs in sequential mode. The OpenMP directives are ignored and a stub OpenMP library is linked.

Alternate Options

None

See Also

openmp, /Qopenmp compiler option
openmp-task, /Qopenmp-task

Lets you choose an OpenMP* tasking model.

IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -openmp-task model
Windows: /Qopenmp-task:model

Arguments

model  
Is an OpenMP tasking model. Possible values are:

intel  
Tells the compiler to accept Intel® taskqueuing pragmas (#pragma intel_omp_taskq and #pragma intel_omp_task). When this value is specified, OpenMP 3.0 tasking pragmas are ignored; if they are specified, warnings are issued.
omp  Tells the compiler to accept OpenMP 3.0 tasking pragmas (#pragma omp_task).
When this value is specified, Intel taskqueuing pragmas are ignored; if they are specified, warnings are issued.

Default

-oopenmp-task omp or /Qopenmp-task:omp

The compiler accepts OpenMP 3.0 tasking pragmas.

Description

The option lets you choose an OpenMP tasking model.
To use this option, you must also specify option -openmp (Linux and Mac OS X) or /Qopenmp (Windows).
Alternate Options

None

openmp-threadprivate, Qopenmp-threadprivate

Lets you specify an OpenMP* threadprivate implementation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -openmp-threadprivate type
Mac OS X: None
Windows: /Qopenmp-threadprivate:type

Arguments

---

**type**

Specifies the type of threadprivate implementation. Possible values are:

- **legacy** Tells the compiler to use the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel®
compat

Tells the compiler to use the compatibility OpenMP\* threadprivate implementation based on applying the __declspec(thread) attribute to each threadprivate variable. The limitations of the attribute on a given platform also apply to the threadprivate implementation. This setting provides compatibility with the implementation provided by the compiler. This setting does not provide compatibility with the implementation used by other compilers.
Default

 setOpenmp-threadprivate legacy
or/Qopenmp-threadprivate:legacy

The compiler uses the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler.

Description

This option lets you specify an OpenMP* threadprivate implementation. The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers.

To use this option, you must also specify one of the following compiler options:

- **Linux OS**: -openmp, -openmp-profile, or -openmp-stubs
- **Windows OS**: /Qopenmp, /Qopenmp-profile, or /Qopenmp-stubs

The value specified for this option is independent of the value used for option -openmp-lib (Linux) or /Qopenmp-lib (Windows).

Alternate Options

None

open-block-factor, Qopt-block-factor
Lets you specify a loop blocking factor.

IDE Equivalent

Windows: Diagnostics > Optimization Diagnostic File
Diagnostics > Emit Optimization Diagnostics to File
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -opt-block-factor=n
Windows: /Qopt-block-factor:n

Arguments

\(n\)

Is the blocking factor. It must be an integer. The compiler may ignore the blocking factor if the
value is 0 or 1.

Default

OFF

The compiler uses default heuristics for loop blocking.

Description

This option lets you specify a loop blocking factor.

Alternate Options

None

**opt-calloc**

Tells the compiler to substitute a call to _intel_fast_calloc() for a call to calloc().

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux:

- opt-calloc
- no-opt-calloc

Mac OS X:

None
Windows: None

Arguments
None

Default
-no-opt-calloc

The compiler does not substitute a call to _intel_fast_calloc() for a call to calloc().

Description
This option tells the compiler to substitute a call to _intel_fast_calloc() for a call to calloc().
This option may increase the performance of long-running programs that use calloc() frequently. It is recommended for these programs over combinations of options -inline-calloc and -opt-malloc-options=3 because this option causes less memory fragmentation.

Alternate Options
None

opt-class-analysis, Qopt-class-analysis

Determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time.

IDE Equivalent
None

Architectures
Syntax

Linux and Mac OS X: -opt-class-analysis
   -opt-class-analysis
   -no-opt-class-analysis

Windows: /Qopt-class-analysis
   /Qopt-class-analysis-

Arguments

None

Default

-no-opt-class-analysis  C++ class hierarchy information is not used to analyze and resolve C++ virtual function calls at compile time.
or /Qopt-class-analysis-

Description

This option determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time. The option is turned on by default with the -ipo compiler option, enabling improved C++ optimization. If a C++ application contains non-standard C++ constructs, such as pointer down-casting, it may result in different behaviors.
Alternate Options

None

**opt-jump-tables, Qopt-jump-tables**

Enables or disables generation of jump tables for switch statements.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-opt-jump-tables=keyword
- opt-jump-tables`

Windows: `/Qopt-jump-tables:keyword`
`/Qopt-jump-tables-`

Arguments

`keyword` Is the instruction for generating jump tables.

Possible values are:

`never` Tells the compiler to never generate jump tables.
All switch statements are
implemented as chains of if-then-elses. This is the same as specifying `-no-opt-jump-tables` (Linux and Mac OS) or `/Qopt-jump-tables` (Windows).

**default** The compiler uses default heuristics to determine when to generate jump tables.

**large** Tells the compiler to generate jump tables up to a certain pre-


**Defined size**

(n entries).

Must be an integer. Tells the compiler to generate jump tables up to \( n \) entries in size.

**Default**

- `opt-jump-tables=default`
  
  or `/Qopt-jump-tables:default`
  
  The compiler uses default heuristics to determine when to generate jump tables for switch statements.

**Description**

This option enables or disables generation of jump tables for switch statements. When the option is enabled, it may improve performance for programs with large switch statements.
Alternate Options

None

**opt-loadpair, Qopt-loadpair**

Enables or disables loadpair optimization.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:
- `-opt-loadpair`
- `-no-opt-loadpair`

Mac OS X:
None

Windows:
- `/Qopt-loadpair`
- `/Qopt-loadpair-`

Arguments

None

Default

- `-no-opt-loadpair`
- `/Qopt-loadpair-`

Loadpair optimization is disabled unless option `O3` is specified.

Description

574
This option enables or disables loadpair optimization. When `-O3` is specified on IA-64 architecture, loadpair optimization is enabled by default. To disable loadpair generation, specify `-no-opt-loadpair` (Linux) or `/Qopt-loadpair-` (Windows).

**Alternate Options**

None

**opt-malloc-options**

Lets you specify an alternate algorithm for malloc().

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-opt-malloc-options=n`

Windows: None

**Arguments**

\( n \)

Specifies the algorithm to use for malloc(). Possible values are:

0 Tells the compiler to use the default algorithm for malloc(). This is the default.

1 Causes the following adjustments to the malloc() algorithm:

M_MMAP_MAX=2 and
M_TRIM_THRESHOLD=0x10000000.

2 Causes the following adjustments to the malloc() algorithm:
   M_MMAP_MAX=2 and
   M_TRIM_THRESHOLD=0x40000000.

3 Causes the following adjustments to the malloc() algorithm:
   M_MMAP_MAX=0 and
   M_TRIM_THRESHOLD=-1.

4 Causes the following adjustments to the malloc() algorithm:
   M_MMAP_MAX=0,
   M_TRIM_THRESHOLD=-1,
   M_TOP_PAD=4096.

Default

-opt-malloc-options=0

   The compiler uses the default algorithm when malloc() is called. No call is made to mallopt().

Description

This option lets you specify an alternate algorithm for malloc(). If you specify a non-zero value for \( n \), it causes alternate configuration parameters to be set for how malloc() allocates and frees memory. It tells the compiler to
insert calls to mallopt() to adjust these parameters to malloc() for dynamic memory allocation. This may improve speed.

Alternate Options
None

See Also
malloc(3) man page
mallopt function (defined in malloc.h)

**opt-mem-bandwidth, Qopt-mem-bandwidth**

Enables performance tuning and heuristics that control memory bandwidth use among processors.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax

Linux: \texttt{-opt-mem-bandwidth}n
Mac OS X: None
Windows: \texttt{/Qopt-mem-bandwidth}n

Arguments

\( n \) Is the level of optimizing for memory bandwidth usage. Possible
values are:
0 Enables a set of performance tuning and heuristics in compiler optimizations that is optimal for serial code.
1 Enables a set of performance tuning and heuristics in compiler optimizations for multithreaded code generated by the compiler.
2 Enables a set of performance tuning and heuristics in compiler optimizations...
optimizations for parallel code such as Windows Threads, pthreads, and MPI code, besides multithreaded code generated by the compiler.

**Default**

- `opt-mem-bandwidth0`
- `opt-mem-bandwidth0`
- `opt-mem-bandwidth0`

For serial (non-parallel) compilation, a set of performance tuning and heuristics in compiler optimizations is enabled that is optimal for serial code.

If you specify
or/\texttt{Qopt-mem-bandwidth1} compiler option –
parallel (Linux) or
/\texttt{Qparallel}
(Windows), –
openmp (Linux) or
/\texttt{Qopenmp}
(Windows), or Cluster
OpenMP option –
cluster-
openmp (Linux), a set of
performance tuning and
heuristics in compiler
optimizations for
multithreaded code
generated by the compiler
is enabled.
This option enables performance tuning and heuristics that control memory bandwidth use among processors. It allows the compiler to be less aggressive with optimizations that might consume more bandwidth, so that the bandwidth can be well-shared among multiple processors for a parallel program. For values of $n$ greater than 0, the option tells the compiler to enable a set of performance tuning and heuristics in compiler optimizations such as prefetching, privatization, aggressive code motion, and so forth, for reducing memory bandwidth pressure and balancing memory bandwidth traffic among threads. This option can improve performance for threaded or parallel applications on multiprocessors or multicore processors, especially when the applications are bounded by memory bandwidth.

Alternate Options

None

See Also

parallel, Qparallel compiler option
openmp, Qopenmp compiler option

\textbf{opt-mod-versioning, Qopt-mod-versioning}

Enables or disables versioning of modulo operations for certain types of operands.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux: 
- `--opt-mod-versioning`
- `--no-opt-mod-versioning`
Mac OS X: None
Windows: /Qopt-mod-versioning
/Qopt-mod-versioning-

Arguments

None

Default

-no-opt-mod-versioning
or/Qopt-mod-versioning-

Description

This option enables or disables versioning of modulo operations for certain types of operands. It is used for optimization tuning. Versioning of modulo operations may improve performance for x mod y when modulus y is a power of 2.

Alternate Options

None

opt-multi-version-aggressive, Qopt-multi-version-aggressive

Tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement.

IDE Equivalent

None

Architectures

582
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:  
- opt-multi-version-aggressive
- no-opt-multi-version-aggressive

Windows:  
/Qopt-multi-version-aggressive
/Qopt-multi-version-aggressive-

Arguments

None

Default

-no-opt-multi-version-aggressive  
The compiler uses default heuristics when checking for pointer aliasing and scalar replacement.

or/Qopt-multi-version-aggressive-

Description

This option tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement. This option may improve performance.

Alternate Options

None

opt-prefetch, Qopt-prefetch

Enables or disables prefetch insertion optimization.
IDE Equivalent

Windows: None
Linux: **Optimization > Enable Prefetch Insertion**
Mac OS X: **Optimization > Enable Prefetch Insertion**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-opt-prefetch[=n]`
-no-opt-prefetch

Windows: `/Qopt-prefetch[:n]`

Arguments

$n$

Is the level of detail in the report.
Possible values are:

- Disables software prefetching.
  This is the same as specifying -no-opt-prefetch
  (Linux and Mac OS X)
or /Qopt-
prefetch-
(Windows).

Enables
to different
levels of
software
prefetching.
If you do not
specify a
value for \( n \),
the default
is 2 on IA-
32 and
Intel® 64
architecture;
the default
is 3 on IA-
64
architecture.
Use lower
values to
reduce the
amount of
prefetching.

Default

IA-64 architecture: -opt-prefetch
or/Qopt-prefetch

On IA-64
architecture,
prefetch insertion optimization is enabled.

IA-32 architecture and Intel® 64 architecture:
-no-opt-prefetch
or/Qopt-prefetch-

On IA-32 architecture and Intel® 64 architecture, prefetch insertion optimization is disabled.

Description

This option enables or disables prefetch insertion optimization. The goal of prefetching is to reduce cache misses by providing hints to the processor about when data should be loaded into the cache.

On IA-64 architecture, this option is enabled by default if you specify option O1 or higher. To disable prefetching at these optimization levels, specify -no-opt-prefetch (Linux and Mac OS X) or /Qopt-prefetch- (Windows).

On IA-32 architecture and Intel® 64 architecture, this option enables prefetching when higher optimization levels are specified.

Alternate Options

Linux and Mac OS X: -prefetch (this is a deprecated option)
Windows: /Qprefetch (this is a deprecated option)

opt-prefetch-initial-values, Qopt-prefetch-initial-values

Enables or disables prefetches that are issued before a loop is entered.
IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:
- -opt-prefetch-initial-values
  - no-opt-prefetch-initial-values

Mac OS X:
  None

Windows:
  /Qopt-prefetch-initial-values
  /Qopt-prefetch-initial-values-

Arguments

None

Default

- -opt-prefetch-initial-values
  or /Qopt-prefetch-initial-values

Prefetches are issued before a loop is entered.

Description

This option enables or disables prefetches that are issued before a loop is entered. These prefetches target the initial iterations of the loop.

When -01 or higher is specified on IA-64 architecture, prefetches are issued before a loop is entered. To disable these prefetches, specify -no-opt-prefetch-initial-values (Linux) or /Qopt-prefetch-initial-values- (Windows).
Alternate Options
None

**opt-prefetch-issue-excl-hint, Qopt-prefetch-issue-excl-hint**

Determines whether the compiler issues prefetches for stores with exclusive hint.

**IDE Equivalent**
None

**Architectures**
IA-64 architecture

**Syntax**

Linux:  
- `--opt-prefetch-issue-excl-hint`
- `--no-opt-prefetch-issue-excl-hint`

Mac OS X:  None

Windows:  
/`Qopt-prefetch-issue-excl-hint`
/`Qopt-prefetch-issue-excl-hint-`

**Arguments**
None

**Default**

`--no-opt-prefetch-issue-excl-hint`  

or `/Qopt-prefetch-issue-excl-hint-`

The compiler does not issue prefetches for stores with
exclusive hint.

Description

This option determines whether the compiler issues prefetches for stores with exclusive hint. If option `-opt-prefetch-issue-excl-hint` (Linux) or `/Qopt-prefetch-issue-excl-hint` (Windows) is specified, the prefetches will be issued if the compiler determines it is beneficial to do so. When prefetches are issued for stores with exclusive-hint, the cache-line is in "exclusive-mode". This saves on cache-coherence traffic when other processors try to access the same cache-line. This feature can improve performance tuning.

Alternate Options

None

`opt-prefetch-next-iteration`, `Qopt-prefetch-next-iteration`

Enables or disables prefetches for a memory access in the next iteration of a loop.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:  
- `opt-prefetch-next-iteration`
- `no-opt-prefetch-next-iteration`

Mac OS X:  
None

Windows:  
/`Qopt-prefetch-next-iteration`
/`Qopt-prefetch-next-iteration-`
Arguments

None

Default

- opt-prefetch-next-iteration
  or /Qopt-prefetch-next-iteration

Prefetches are issued for a memory access in the next iteration of a loop.

Description

This option enables or disables prefetches for a memory access in the next iteration of a loop. It is typically used in a pointer-chasing loop. When -O1 or higher is specified on IA-64 architecture, prefetches are issued for a memory access in the next iteration of a loop. To disable these prefetches, specify -no-opt-prefetch-next-iteration (Linux) or /Qopt-prefetch-next-iteration (Windows).

Alternate Options

None

opt-ra-region-strategy, Qopt-ra-region-strategy

Selects the method that the register allocator uses to partition each routine into regions.

IDE Equivalent

None
Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X: `-opt-ra-region-strategy[=keyword]`
Windows: `/Qopt-ra-region-strategy[=keyword]`

Arguments

keyword Is the method used for partitioning.
Possible values are:

- **routine** Creates a single region for each routine.
- **block** Partitions each routine into one region per basic block.
- **trace** Partitions each routine into one region per trace.
- **region** Partitions each
routine into one region per loop.

**default**

The compiler determines which method is used for partitioning. This is also the default if the keyword is not specified.

**Default**

- `opt-ra-region-strategy=default`
- `Qopt-ra-region-strategy:default`

**Description**

This option selects the method that the register allocator uses to partition each routine into regions.

When setting **default** is in effect, the compiler attempts to optimize the tradeoff between compile-time performance and generated code performance.
This option is only relevant when optimizations are enabled (O1 or higher).

Alternate Options
None

See Also
O compiler option

opt-report, Qopt-report
Tells the compiler to generate an optimization report to stderr.

IDE Equivalent
Windows: Diagnostics > Optimization Diagnostic Level
Linux: Compilation Diagnostics > Optimization Diagnostic Level
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -opt-report[n]
Windows: /Qopt-report[:n]

Arguments

n Is the level of detail in the report.
Possible values are:
0 Tells the
compiler to generate no optimization report.

1 Tells the compiler to generate a report with the minimum level of detail.

2 Tells the compiler to generate a report with the medium level of detail.

3 Tells the compiler to generate a report with the maximum level of detail.

Default
-opt-report 2 or /Qopt-report:2

If you do not specify n, the compiler generates a report with medium detail. If you do not specify the option on the command line, the compiler does not generate an optimization report.

Description

This option tells the compiler to generate an optimization report to stderr.

Alternate Options

None

See Also

opt-report-file, Qopt-report-file compiler option
Optimizing Applications: Optimizer Report Generation
opt-report-file, Qopt-report-file

Specifies the name for an optimization report.

IDE Equivalent

Windows: Diagnostics > Optimization Diagnostic File
Diagnostics > Emit Optimization Diagnostics to File
Linux: Compilation Diagnostics > Emit Optimization Diagnostics to File
Compilation Diagnostics > Optimization Diagnostics File
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -opt-report-file=file
Windows: /Qopt-report-file:file

Arguments

file

Is the name for the optimization report.

Default

OFF

No optimization report is generated.

Description

596
This option specifies the name for an optimization report. If you use this option, you do not have to specify `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

Alternate Options

None

See Also

`opt-report`, `Qopt-report` compiler option
Optimizing Applications: Optimizer Report Generation

`opt-report-help`, `Qopt-report-help`

Displays the optimizer phases available for report generation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-opt-report-help`
Windows: `/Qopt-report-help`

Arguments

None

Default

OFF

No optimization reports are
This option displays the optimizer phases available for report generation using `opt-report-phase` (Linux and Mac OS X) or `/Qopt-report-phase` (Windows). No compilation is performed.

**Alternate Options**

None

**See Also**

`opt-report`, `Qopt-report` compiler option

`opt-report-phase`, `Qopt-report-phase` compiler option

**opt-report, Qopt-report**

Tells the compiler to generate an optimization report to stderr.

**IDE Equivalent**

Windows: Diagnostics > Optimization Diagnostic Level

Linux: Compilation Diagnostics > Optimization Diagnostic Level

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-opt-report[n]`

Windows: `/Qopt-report[:n]`

**Arguments**

`n` Is the level of
detail in the report.
Possible values are:

0 Tells the compiler to generate no optimization report.

1 Tells the compiler to generate a report with the minimum level of detail.

2 Tells the compiler to generate a report with the medium level of detail.

3 Tells the compiler to generate a report with
the maximum level of detail.

Default

- `opt-report 2` or `/Qopt-report:2`

If you do not specify \( n \), the compiler generates a report with medium detail. If you do not specify the option on the command line, the compiler does not generate an optimization report.

Description

This option tells the compiler to generate an optimization report to `stderr`. 
Alternate Options

None

See Also

opt-report-file, Qopt-report-file compiler option
Optimizing Applications: Optimizer Report Generation

opt-report-phase, Qopt-report-phase

Specifies an optimizer phase to use when optimization reports are generated.

IDE Equivalent

Windows: Diagnostics > Optimization Diagnostic Phase
Linux: Compilation Diagnostics > Optimization Diagnostic Phase
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -opt-report-phase=phase
Windows: /Qopt-report-phase:phase

Arguments

phase

Is the phase to generate reports for. Some of the possible values are:

ipo The Interprocedural Optimizer phase
hlo  The High Level Optimizer phase

hpo  The High Performance Optimizer phase

ilo  The Intermediate Language Scalar Optimizer phase

ecg  The Code Generator phase
     (Windows and Linux systems using IA-64 architecture only)

ecg_swp The software pipelining component of the Code Generator phase
     (Windows and Linux systems)
The Profile Guided Optimization phase only)

**pgo**
The Profile Guided Optimization phase

**all**
All optimizer phases

**Default**

**OFF**
No optimization reports are generated.

**Description**

This option specifies an optimizer phase to use when optimization reports are generated. To use this option, you must also specify `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

This option can be used multiple times on the same command line to generate reports for multiple optimizer phases.

When one of the logical names for optimizer phases is specified for phase, all reports from that optimizer phase are generated.

To find all phase possibilities, use option `-opt-report-help` (Linux and Mac OS X) or `/Qopt-report-help` (Windows).

**Alternate Options**

None
See Also

`opt-report`, `Qopt-report` compiler option

**opt-report-routine, Qopt-report-routine**

Tells the compiler to generate reports on the routines containing specified text.

**IDE Equivalent**

Windows: **Diagnostics > Optimization Diagnostic Routine**

Linux: **Compilation Diagnostics > Optimization Diagnostic Routine**

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `–opt-report-routine=string`

Windows: `/Qopt-report-routine:string`

**Arguments**

`string`  
Is the text (string) to look for.

**Default**

OFF  
No optimization reports are generated.
Description

This option tells the compiler to generate reports on the routines containing specified text as part of their name.

Alternate Options

None

See Also

opt-report, Qopt-report compiler option

opt-streaming-stores, Qopt-streaming-stores

Enables generation of streaming stores for optimization.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -opt-streaming-stores keyword
Windows: /Qopt-streaming-stores:keyword

Arguments

keyword

Specifies whether streaming stores are generated. Possible values are:

always Enables generation of streaming
The compiler optimizes under the assumption that the application is memory bound.

**never** Disables generation of streaming stores for optimization. Normal stores are performed.

**auto** Lets the compiler decide which instructions to use.

Default

-`-opt-streaming-stores auto`

or `/Qopt-streaming-stores:auto`

The compiler
decides whether to use streaming stores or normal stores.

Description

This option enables generation of streaming stores for optimization. This method stores data with instructions that use a non-temporal buffer, which minimizes memory hierarchy pollution.

For this option to be effective, the compiler must be able to generate SSE2 (or higher) instructions. For more information, see compiler option x or ax.

This option may be useful for applications that can benefit from streaming stores.

Alternate Options

None

See Also

ax, Qax compiler option
x, Qx compiler option
opt-mem-bandwidth, Qopt-mem-bandwidth, Qx compiler option

opt-subscript-in-range, Qopt-subscript-in-range

Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.

IDE Equivalent

None

Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -opt-subscript-in-range
                      -no-opt-subscript-in-range

Windows:           /Qopt-subscript-in-range
                      /Qopt-subscript-in-range-

Arguments

None

Default

-no-opt-subscript-in-range

or /Qopt-subscript-in-range-

The compiler assumes no overflows in the intermediate computation of subscript expressions in loops.

Description

This option determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.

If you specify -opt-subscript-in-range (Linux and Mac OS X) or /Qopt-subscript-in-range (Windows), the compiler ignores any data type conversions used and it assumes no overflows in the intermediate computation of subscript expressions. This feature can enable more loop transformations.
Alternate Options

None

Example

The following shows an example where these options can be useful. m is declared as type long (64-bits) and all other variables inside the subscript are declared as type int (32-bits):

\[ A[ i + j + ( n + k) * m ] \]

Os

Enables optimizations that do not increase code size and produces smaller code size than O2.

IDE Equivalent

Windows: **Optimization > Favor Size or Speed**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-os\)
Windows: \(/os\)

Arguments

None

Default

OFF

Optimizations are made for
code speed. However, if 01 is specified, Os is the default.

Description

This option enables optimizations that do not increase code size and produces smaller code size than 02. It disables some optimizations that increase code size for a small speed benefit. This option tells the compiler to favor transformations that reduce code size over transformations that produce maximum performance.

Alternate Options

None

See Also

0 compiler option
Ot compiler option

Ot

Enables all speed optimizations.

IDE Equivalent

Windows: Optimization > Favor Size or Speed
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

610
**Linux and Mac OS X:** None

**Windows:** /Ot

**Arguments**

None

**Default**

/Ot

Optimizations are made for code speed.

If **Od** is specified, all optimizations are disabled.

If **O1** is specified, **Os** is the default.

**Description**

This option enables all speed optimizations.

**Alternate Options**

None

**See Also**

/o compiler option

/os compiler option

**Ow**

Tells the compiler to assume there is no cross-function aliasing.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Windows: /Ow
/Ow-

Arguments
None

Default
OFF

The compiler assumes cross-function aliasing occurs.

Description
This option tells the compiler to assume there is no cross-function aliasing.

Alternate Options
None

Ox
Enables maximum optimizations.
IDE Equivalent

Windows: **Optimization > Optimization**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: `/Ox`

Arguments

None

Default

OFF

The compiler does not enable optimizations.

Description

The compiler enables maximum optimizations by combining the following options:

- `/Ob2`
- `/Og`
- `/Oy`
- `/Ot`
- `/Oi`

Alternate Options
None

**fomit-frame-pointer, Oy**

Determines whether EBP is used as a general-purpose register in optimizations.

**IDE Equivalent**

Windows: **Optimization > Omit Frame Pointers**

Linux: **Optimization > Provide Frame Pointer**

Mac OS X: **Optimization > Provide Frame Pointer**

**Architectures**

- `f[no-]omit-frame-pointer`: IA-32 architecture, Intel® 64 architecture
- `/Oy[-]`: IA-32 architecture

**Syntax**

**Linux and Mac OS X:** `-fomit-frame-pointer`

- `-fno-omit-frame-pointer`

Windows: `/Oy`

`/Oy-`

**Arguments**

None

**Default**

- `fomit-frame-pointer`
  
  EBP is used as a general-purpose register in optimizations.

  However, on
Linux* and Mac OS X systems, the default is `-fno-omit-frame-pointer` if option `-O0` or `-g` is specified. On Windows* systems, the default is `/Oy-` if option `/Od` is specified.

**Description**

These options determine whether EBP is used as a general-purpose register in optimizations. Options `-fomit-frame-pointer` and `/Oy` allow this use. Options `-fno-omit-frame-pointer` and `/Oy-` disallow it.

Some debuggers expect EBP to be used as a stack frame pointer, and cannot produce a stack backtrace unless this is so. The `-fno-omit-frame-pointer` and `/Oy-` options direct the compiler to generate code that maintains and uses EBP as a stack frame pointer for all functions so that a debugger can still produce a stack backtrace without doing the following:

- **For `-fno-omit-frame-pointer`: turning off optimizations with `-O0`**
- **For `/Oy-`: turning off `/O1`, `/O2`, or `/O3` optimizations**
The `-fno-omit-frame-pointer` option is set when you specify option `-00` or the `–g` option. The `-fomit-frame-pointer` option is set when you specify option `-01, -02, or -03.

The `/Oy` option is set when you specify the `/01, /02, or /03` option. Option `/Oy-` is set when you specify the `/Od` option.

Using the `-fno-omit-frame-pointer` or `/Oy-` option reduces the number of available general-purpose registers by 1, and can result in slightly less efficient code.

**Note**

There is currently an issue with GCC 3.2 exception handling. Therefore, the Intel compiler ignores this option when GCC 3.2 is installed for C++ and exception handling is turned on (the default).

**Alternate Options**

Linux and Mac OS X: `-fp` (this is a deprecated option)

Windows: None

**p**

Compiles and links for function profiling with gprof(1).

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-p`

Windows: None
Arguments

None

Default

OFF

Description

This option compiles and links for function profiling with gprof(1).

Alternate Options

Linux and Mac OS X: -qp (this is a deprecated option)
Windows: None

P

Tells the compiler to stop the compilation process and write the results to a file.

IDE Equivalent

Windows: Preprocessor > Generate Preprocessed File
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: `-P`
Windows: `/P`

**Arguments**
None

**Default**
OFF
Normal compilation is performed.

**Description**
This option tells the compiler to stop the compilation process after C or C++ source files have been preprocessed and write the results to files named according to the compiler's default file-naming conventions.

On Linux systems, this option causes the preprocessor to expand your source module and direct the output to a `.i` file instead of `stdout`. Unlike the `-E` option, the output from `-P` on Linux does not include `#line` number directives. By default, the preprocessor creates the name of the output file using the prefix of the source file name with a `.i` extension. You can change this by using the `-o` option.

**Alternate Options**
`-F`

**See Also**
Building Applications: About Preprocessor Options

**par-report, Qpar-report**
Controls the diagnostic information reported by the auto-parallelizer.
IDE Equivalent

Windows: None
Linux: **Compilation Diagnostics > Auto-Parallelizer Report**
Mac OS X: **Diagnostics > Auto-Parallelizer Report**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-par-report[n]`
Windows: `/Qpar-report[n]`

Arguments

`n` is a value denoting which diagnostic messages to report. Possible values are:

0  Tells the auto-parallelizer to report no diagnostic information.
1  Tells the auto-parallelizer to report diagnostic messages for loops successfully
auto-parallelized. The compiler also issues a "LOOP AUTO-PARALLELIZED" message for parallel loops.

2 Tells the auto-parallelizer to report diagnostic messages for loops successfully and unsuccessfully auto-parallelized.

3 Tells the auto-parallelizer to report the same diagnostic messages specified by 2 plus additional information about any proven or assumed dependencies inhibiting auto-parallelization
Default

-par-report1
or/Qpar-report1

If you do not specify $n$, the compiler displays diagnostic messages for loops successfully auto-parallelized. If you do not specify the option on the command line, the default is to display no parallel diagnostic messages.
This option controls the diagnostic information reported by the auto-parallelizer (parallel optimizer). To use this option, you must also specify `-parallel` (Linux and Mac OS X) or `/Qparallel` (Windows).

If this option is specified on the command line, the report is sent to `stdout`.

**Alternate Options**

None

**par-runtime-control, Qpar-runtime-control**

Generates code to perform run-time checks for loops that have symbolic loop bounds.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-par-runtime-control`  
- `-no-par-runtime-control`

**Windows:** `/Qpar-runtime-control`  
- `/Qpar-runtime-control-`

**Arguments**

None

**Default**

- `-no-par-runtime-control`  
- `/Qpar-runtime-control-`

The compiler uses
**Description**

This option generates code to perform run-time checks for loops that have symbolic loop bounds.

If the granularity of a loop is greater than the parallelization threshold, the loop will be executed in parallel.

If you do not specify this option, the compiler may not parallelize loops with symbolic loop bounds if the compile-time granularity estimation of a loop can not ensure it is beneficial to parallelize the loop.

**Alternate Options**

None

**par-schedule, Qpar-schedule**

Lets you specify a scheduling algorithm or a tuning method for loop iterations.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-par-schedule-keyword[-n]`

**Windows:** `/Qpar-schedule-keyword[:n]`
Arguments

_keyword_ Specifies the scheduling algorithm or tuning method. Possible values are:

- **auto**: Lets the compiler or run-time system determine the scheduling algorithm.
- **static**: Divides iterations into contiguous pieces.
- **static-balanced**: Divides iterations into even-sized chunks.
- **static-steal**: Divides iterations into even-sized chunks, but allows
threads to steal parts of chunks from neighboring threads.

dynamic

- Gets a set of iterations dynamically.

guided

- Specifies a minimum number of iterations.

guided-analytical

- Divides iterations by using exponential distribution or dynamic distribution.

runtime

- Defers the scheduling decision until run time.

\[ n \]

- Is the size of the chunk or the number
of iterations for each chunk. This setting can only be specified for static, dynamic, and guided. For more information, see the descriptions of each keyword below.

Default

static-balanced

Iterations are divided into even-sized chunks and the chunks are assigned to the
threads in the team in a round-robin fashion in the order of the thread number.

### Description

This option lets you specify a scheduling algorithm or a tuning method for loop iterations. It specifies how iterations are to be divided among the threads of the team.

This option affects performance tuning and can provide better performance during auto-parallelization.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-par-schedule-auto</code> or <code>/Qpar-schedule-auto</code></td>
<td>Lets the compiler or run-time system determine the scheduling algorithm. Any possible mapping may occur for iterations to threads in the team.</td>
</tr>
<tr>
<td><code>-par-schedule-static</code> or <code>/Qpar-schedule-static</code></td>
<td>Divides iterations into contiguous pieces (chunks) of size ( n ). The chunks are assigned to threads in the team in a round-robin fashion in the order of the thread number. Note that the last chunk to be assigned may have a smaller</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>number of iterations.</td>
<td>If no ( n ) is specified, the iteration space is divided into chunks that are approximately equal in size, and each thread is assigned at most one chunk.</td>
</tr>
<tr>
<td>-par-schedule-static-balanced or /Qpar-schedule-static-balanced</td>
<td>Divides iterations into even-sized chunks. The chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.</td>
</tr>
<tr>
<td>-par-schedule-static-steal or /Qpar-schedule-static-steal</td>
<td>Divides iterations into even-sized chunks, but when a thread completes its chunk, it can steal parts of chunks assigned to neighboring threads. Each thread keeps track of ( L ) and ( U ), which represent the lower and upper bounds of its chunks respectively. Iterations are executed starting from the lower bound, and simultaneously, ( L ) is updated to represent the new lower bound.</td>
</tr>
<tr>
<td>-par-schedule-dynamic or /Qpar-schedule-dynamic</td>
<td>Can be used to get a set of iterations dynamically. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>As each thread finishes a piece of the iteration space, it dynamically gets the next set of iterations. Each chunk contains ( n ) iterations, except for the last chunk to be assigned, which may have fewer iterations. If no ( n ) is specified, the default is 1.</td>
</tr>
<tr>
<td><code>-par-schedule-guided</code> or <code>/Qpar-schedule-guided</code></td>
<td>Can be used to specify a minimum number of iterations. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned. For a chunk of size 1, the size of each chunk is proportional to the number of unassigned iterations divided by the number of threads, decreasing to 1. For an ( n ) with value ( k ) (greater than 1), the size of each chunk is determined in the same way with the restriction that the chunks do not contain fewer than ( k ) iterations (except for the last chunk to be assigned, which may have fewer than ( k ) iterations). If no ( n ) is specified, the default is 1.</td>
</tr>
<tr>
<td><code>-par-schedule-guided-analytical</code> or <code>/Qpar-schedule-analytical</code></td>
<td>Divides iterations by using exponential distribution or dynamic distribution. The</td>
</tr>
</tbody>
</table>
Option | Description
--- | ---
guided-analytical | method depends on run-time implementation. Loop bounds are calculated with faster synchronization and chunks are dynamically dispatched at run time by threads in the team.
-par-schedule-runtime or /Qpar-schedule-runtime | Defers the scheduling decision until run time. The scheduling algorithm and chunk size are then taken from the setting of environment variable OMP_SCHEDULE.

Alternate Options

None

par-threshold, Qpar-threshold

Sets a threshold for the auto-parallelization of loops.

IDE Equivalent

Windows: None
Linux: **Optimization > Auto-Parallelization Threshold**
Mac OS X: **Optimization > Auto-Parallelization Threshold**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-par-threshold[n]`
Windows: `/Qpar-threshold[[[:]]n]`
Arguments

\( n \)

Is an integer whose value is the threshold for the auto-parallelization of loops. Possible values are 0 through 100.

If \( n \) is 0, loops get auto-parallelized always, regardless of computation work volume.

If \( n \) is 100, loops get auto-parallelized when performance gains are predicted based on the compiler analysis data. Loops get
auto-parallelized only if profitable parallel execution is almost certain. The intermediate 1 to 99 values represent the percentage probability for profitable speed-up. For example, $n=50$ directs the compiler to parallelize only if there is a 50% probability of the code speeding up if executed in parallel.
Loops get auto-parallelized only if profitable parallel execution is almost certain. This is also the default if you do not specify $n$.

**Description**

This option sets a threshold for the auto-parallelization of loops based on the probability of profitable execution of the loop in parallel. To use this option, you must also specify `-parallel` (Linux and Mac OS X) or `/Qparallel` (Windows).

This option is useful for loops whose computation work volume cannot be determined at compile-time. The threshold is usually relevant when the loop trip count is unknown at compile-time.

The compiler applies a heuristic that tries to balance the overhead of creating multiple threads versus the amount of work available to be shared amongst the threads.

**Alternate Options**

None

parallel, Qparallel
Tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.

IDE Equivalent

Windows: Optimization > Parallelization
Linux: Optimization > Parallelization
Mac OS X: Optimization > Parallelization

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -parallel
Windows: /Qparallel

Arguments

None

Default

OFF

Multithreaded code is not generated for loops that can be safely executed in parallel.

Description

This option tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.
To use this option, you must also specify option 02 or 03.
Note

On Mac OS X systems, when you enable automatic parallelization, you must also set the `DYLD_LIBRARY_PATH` environment variable within Xcode or an error will be displayed.

Alternate Options

None

See Also

pc, Qpc

Enables control of floating-point significand precision.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: `-pcn`

Windows: `/Qpcn`

Arguments

$n$

Is the floating-point significand precision.

Possible values are:

32 Rounds
the significand to 24 bits (single precision).

64 Rounds the significand to 53 bits (double precision).

80 Rounds the significand to 64 bits (extended precision).

**Default**

- `pc80`
- `Qpc64`

On Linux* and Mac OS* X systems, the floating-point significand is rounded to 64 bits.
On Windows* systems, the floating-point significand is rounded to 53 bits.

Description

This option enables control of floating-point significand precision. Some floating-point algorithms are sensitive to the accuracy of the significand, or fractional part of the floating-point value. For example, iterative operations like division and finding the square root can run faster if you lower the precision with this option.

Note that a change of the default precision control or rounding mode, for example, by using the -pc32 (Linux and Mac OS X) or /Qpc32 (Windows) option or by user intervention, may affect the results returned by some of the mathematical functions.

Alternate Options

None

pch

Tells the compiler to use appropriate precompiled header files.

IDE Equivalent

Windows: None

Linux: Precompiled Headers > Automatic Processing for Precompiled Headers
Mac OS X: **Precompiled Headers > Precompile Prefix Header**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-pch`

Windows: None

**Arguments**

None

**Default**

OFF

The compiler does not create or use precompiled headers unless you tell it to do so.

**Description**

This option tells the compiler to use appropriate precompiled header (PCH) files. If none are available, they are created as `sourcefile.pch`. This option is supported for multiple source files.

The `-pch` option will use PCH files created from other sources if the headers files are the same. For example, if you compile `source1.cpp` using `-pch`, then
source1.pchi is created. If you then compile source2.cpp using -pch, the compiler will use source1.pchi if it detects the same headers.

⚠️ Caution

Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

Alternate Options

None

Example

Consider the following command line:

```plaintext
icpc -pch source1.cpp source2.cpp
```

It produces the following output when .pchi files do not exist:

"source1.cpp": creating precompiled header file "source1.pchi"
"source2.cpp": creating precompiled header file "source2.pchi"

It produces the following output when .pchi files do exist:

"source1.cpp": using precompiled header file "source1.pchi"
"source2.cpp": using precompiled header file "source2.pchi"

See Also

- `pch-create` compiler option
- `pch-dir` compiler option
- `pch-use` compiler option

pch-create, Yc

Lets you create and specify a name for a precompiled header file.

IDE Equivalent

Windows: Precompiled Headers > Create-Use Precompiled Header / Create-Use PCH Through File

Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -pch-createfile
Windows: /Ycfile

Arguments

file

Is the name for the precompiled header file.

Default

OFF

The compiler does not create or use precompiled headers unless you tell it to do so.

Description

This option lets you specify a name for a precompiled header (PCH) file. It is supported only for single source file compilations.
The `.pchi` extension is not automatically appended to the file name. This option cannot be used in the same compilation as the `-pch-use` option. Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

Alternate Options

None

Example

Consider the following command line:

```
icpc -pch-create /pch/source32.pchi source.cpp
```

It produces the following output:

"source.cpp": creating precompiled header file "/pch/source32.pchi"

See Also

Precompiled Headers

`pch-dir`

Tells the compiler where to find or create a file for precompiled headers.

IDE Equivalent

Windows: None

Linux: Precompiled Headers > Precompiled Headers' File Directory

Mac OS X: Precompiled Headers > Prefix Header

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-pch-dir dir`

Windows: None
Arguments

dir

Is the path where the file is located or should be created. The path must exist.

Default

OFF

The compiler does not create or use precompiled headers unless you tell it to do so.

Description
This option tells the compiler where to find or create a file (PCH) for precompiled headers.

This option can be used with the `-pch`, `-pch-create`, and `-pch-use` options.

⚠️ **Caution**

Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

**Alternate Options**

None

**Example**

Consider the following command line:

```bash
icpc -pch -pch-dir /pch source32.cpp
```

It produces the following output:

"source32.cpp": creating precompiled header file /pch/source32.pch

See Also

- `-pch` compiler option
- `-pch-create` compiler option
- `-pch-use` compiler option

**pch-use**

Lets you use a specific precompiled header file.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X: -pch-use \{file | dir\}

Windows: None

Arguments

\textit{file}

Is the name of the precompiled header file to use.

\textit{dir}

Is the path where the file is located, including the path. The path must exist.

Default

OFF

The compiler does not create or use precompiled headers unless you tell it to do so.

Description

644
This option lets you use a specific precompiled header (PCH) file. It is supported for multiple source files when all source files use the same .pchi file. This option cannot be used in the same compilation as the -pch-create option.

Caution

Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

Alternate Options

None

Example

Consider the following command line:

```
icpc -pch-use /pch/source32.pchi source.cpp
```

It produces the following output:

"source.cpp"': using precompiled header file /pch/source32.pchi

See Also

-pch-create compiler option

pie

Produces a position-independent executable on processors that support it.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -pie
Mac OS X: None
Windows: None

Arguments
None

Default
OFF

The driver does not set up special run-time libraries and the linker does not perform the optimizations on executables.

Description
This option produces a position-independent executable on processors that support it. It is both a compiler option and a linker option. When used as a compiler option, this option ensures the linker sets up run-time libraries correctly. Normally the object linked has been compiled with option \(-fpie\). When you specify \(-pie\), it is recommended that you specify the same options that were used during compilation of the object.

Alternate Options
None

See Also
646
fpie compiler option

pragma-optimization-level

Specifies which interpretation of the optimization_level pragma should be used if no prefix is specified.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: 

-pragma-optimization-level=

Windows: None

Arguments

interpretation  Compiler-specific interpretation of optimization_level pragma. Possible values are:

Intel Specify the Intel interpretation.

GCC Specify the GCC interpretation.

Default
-pragma-optimization-level=Intel

Use the Intel interpretation of the optimization_level pragma.

Description

Specifies which interpretation of the optimization_level pragma should be used if no prefix is specified.

Alternate Options

None

See Also

Compiler Reference: pragma optimization_level

prec-div, Qprec-div

Improves precision of floating-point divides.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -prec-div
-no-prec-div

Windows: /Qprec-div
/Qprec-div-

Arguments

None
Default

-prec-div
-or/Qprec-div

The compiler uses this method for floating-point divides.

Description

This option improves precision of floating-point divides. It has a slight impact on speed.

With some optimizations, such as -xSSE2 (Linux) or /QxSSE2 (Windows), the compiler may change floating-point division computations into multiplication by the reciprocal of the denominator. For example, A/B is computed as A * (1/B) to improve the speed of the computation.

However, sometimes the value produced by this transformation is not as accurate as full IEEE division. When it is important to have fully precise IEEE division, use this option to disable the floating-point division-to-multiplication optimization. The result is more accurate, with some loss of performance.

If you specify -no-prec-div (Linux and Mac OS X) or /Qprec-div- (Windows), it enables optimizations that give slightly less precise results than full IEEE division.

Alternate Options

None

prec-sqrt, Qprec-sqrt
Implements precision of square root implementations.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X: `-prec-sqrt`
   `-no-prec-sqrt`
Windows: `/Qprec-sqrt`
   `/Qprec-sqrt-`

Arguments
None

Default
`-no-prec-sqrt` or `/Qprec-sqrt-` The compiler uses a faster but less precise implementation of square root.

Note that the default is `-prec-sqrt` or `/Qprec-sqrt` if any of the following options are
specified: /Od, /Op, or /Qprec on Windows systems; -O0, -mp, or -mp1 on Linux and Mac OS X systems.

Description

This option improves precision of square root implementations. It has a slight impact on speed.

This option inhibits any optimizations that can adversely affect the precision of a square root computation. The result is fully precise square root implementations, with some loss of performance.

Alternate Options

None

print-multi-lib

Prints information about where system libraries should be found.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -print-multi-lib
Windows: None

Arguments
None

Default
OFF

Description
This option prints information about where system libraries should be found, but no compilation occurs. It is provided for compatibility with gcc.

Alternate Options
None

prof-data-order, Qprof-data-order
Enables or disables data ordering if profiling information is enabled.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux: -prof-data-order
-no-prof-data-order

Mac OS X: None
Windows: /Qprof-data-order
        /Qprof-data-order-

Arguments
None

Default
-no-prof-data-order
or /Qprof-data-order-

Data
ordering
is
disabled.

Description
This option enables or disables data ordering if profiling information is enabled. It controls the use of profiling information to order static program data items. For this option to be effective, you must do the following:

- For instrumentation compilation, you must specify -prof-gen=globdata (Linux) or /Qprof-gen:globdata (Windows).
- For feedback compilation, you must specify -prof-use (Linux) or /Qprof-use (Windows). You must not use multi-file optimization by specifying options such as option -ipo (Linux) or /Qipo (Windows), or option -ipo-c (Linux) or /Qipo-c (Windows).

Alternate Options
None

See Also
prof-gen, Qprof-gen compiler option
prof-use, Qprof-use compiler option
prof-func-order, Qprof-func-order compiler option

prof-dir, Qprof-dir

Specifies a directory for profiling information output files.

IDE Equivalent

Windows: General > Profile Directory
Linux: Compiler > Profile Directory
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -prof-dir dir
Windows: /Qprof-dir dir

Arguments

\textit{dir} \hspace{1cm} \text{Is the name of the directory.}

Default

OFF \hspace{1cm} \text{Profiling output files are placed in the directory}
where the program is compiled.

**Description**

This option specifies a directory for profiling information output files (*.dyn and *.dpi). The specified directory must already exist.

You should specify this option using the same directory name for both instrumentation and feedback compilations. If you move the .dyn files, you need to specify the new path.

**Alternate Options**

None

**prof-file, Qprof-file**

Specifies an alternate file name for the profiling summary files.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-prof-file file`

Windows: `/Qprof-file file`

**Arguments**
file

Is the name of the profiling summary file.

Default

OFF

The profiling summary files have the file name pgopti.*

Description

This option specifies an alternate file name for the profiling summary files. The file is used as the base name for files created by different profiling passes. If you add this option to profmerge, the .dpi file will be named file.dpi instead of pgopti.dpi. If you specify -prof-genx (Linux and Mac OS X) or /Qprof-genx (Windows) with this option, the .spi and .spl files will be named file.spi and file.spl instead of pgopti.spi and pgopti.spl. If you specify -prof-use (Linux and Mac OS X) or /Qprof-use (Windows) with this option, the .dpi file will be named file.dpi instead of pgopti.dpi.

Alternate Options

None

See Also

656
**prof-gen, Qprof-gen** compiler option
**prof-use, Qprof-use** compiler option

**prof-func-groups**

Enables or disables function grouping if profiling information is enabled.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux:                      -prof-func-groups
                           -no-prof-func-groups

Mac OS X:                   None

Windows:                    None

**Arguments**

None

**Default**

-no-prof-func-groups                     Function
                                         grouping
                                         is
                                         disabled.

**Description**

This option enables or disables function grouping if profiling information is enabled.

A "function grouping" is a profiling optimization in which entire routines are placed either in the cold code section or the hot code section.
If profiling information is enabled by option -prof-use, option -prof-func-groups is set and function grouping is enabled. However, if you explicitly enable -prof-func-order (Linux) or /Qprof-func-order (Windows), function ordering is performed instead of function grouping.
If you want to disable function grouping when profiling information is enabled, specify -no-prof-func-groups.
To set the hotness threshold for function grouping, use option -prof-hotness-threshold (Linux) or /Qprof-hotness-threshold (Windows).

Alternate Options
-func-groups (this is a deprecated option)

See Also
 prof-use, Qprof-use compiler option
 prof-func-order, Qprof-func-order compiler option
 prof-hotness-threshold, Qprof-hotness-threshold compiler option

prof-func-order, Qprof-func-order
Enables or disables function ordering if profiling information is enabled.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:                -prof-func-order
                    -no-prof-func-order
Mac OS X:            None
Windows:             /Qprof-func-order
/Qprof-func-order-

Arguments

None

Default

-no-prof-func-order
or /Qprof-func-order-

Function ordering is disabled.

Description

This option enables or disables function ordering if profiling information is enabled.

For this option to be effective, you must do the following:

- **For instrumentation compilation, you must specify** `-prof-gen=srcpos` (Linux) or `/Qprof-gen:srcpos` (Windows).
- **For feedback compilation, you must specify** `-prof-use` (Linux) or `/Qprof-use` (Windows). You must not use multi-file optimization by specifying options such as option `-ipo` (Linux) or `/Qipo` (Windows), or option `-ipo-c` (Linux) or `/Qipo-c` (Windows).

If you enable profiling information by specifying option `-prof-use` (Linux) or `/Qprof-use` (Windows), `-prof-func-groups` (Linux) and `/Qprof-func-groups` (Windows) are set and function grouping is enabled. However, if you explicitly enable `/Qprof-func-order` (Windows), function ordering is performed instead of function grouping.

On Linux* systems, this option is only available for Linux linker 2.15.94.0.1, or later.
To set the hotness threshold for function grouping and function ordering, use option `-prof-hotness-threshold` (Linux) or `/Qprof-hotness-threshold` (Windows).

Alternate Options

None

The following example shows how to use this option on a Windows system:

```bash
icl /Qprof-gen:globdata file1.c file2.c /Fe instrumented.exe
icl /Qprof-use /Qprof-func-order file1.c file2.c /Fe feedback.exe
```

The following example shows how to use this option on a Linux system:

```bash
icl -prof-gen:globdata file1.c file2.c -o instrumented
icl -prof-use -prof-func-order file1.c file2.c -o feedback
```

See Also

prof-hotness-threshold, Qprof-hotness-threshold compiler option
prof-gen, Qprof-gen compiler option
prof-use, Qprof-use compiler option
prof-data-order, Qprof-data-order compiler option
prof-func-groups compiler option

**prof-gen, Qprof-gen**

Produces an instrumented object file that can be used in profile-guided optimization.

IDE Equivalent

Windows: General > Profile Guided Optimization
Optimization > Profile Guided Optimization

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-prof-gen[=keyword]`
  -no-prof-gen
Windows: `/Qprof-gen[:keyword]`
  `/Qprof-gen-

Arguments

`keyword` Specifies details for the instrumented file.
Possible values are:

- **default** Produces an instrumented object file.
  This is the same as specifying `-prof-gen` (Linux* and Mac OS* X) or `/Qprof-gen` (Windows*) with no keyword.

- **srcpos** Produces an instrumented object file that includes extra source
position information. This option is the same as option –prof-genx (Linux* and Mac OS* X) or /Qprof-genx (Windows*), which are deprecated.

globdata Produces an instrumented object file that includes information for global data layout.

Default

-no-prof-gen or /Qprof-gen-

Profile generation is disabled.

Description

This option produces an instrumented object file that can be used in profile-guided optimization. It gets the execution count of each basic block.
If you specify keyword `srcpos` or `globdata`, a static profile information file (.spi) is created. These settings may increase the time needed to do a parallel build using `-prof-gen`, because of contention writing the .spi file. These options are used in phase 1 of the Profile Guided Optimizer (PGO) to instruct the compiler to produce instrumented code in your object files in preparation for instrumented execution.

Alternate Options

None

`prof-hotness-threshold, Qprof-hotness-threshold`

Lets you set the hotness threshold for function grouping and function ordering.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: `-prof-hotness-threshold=n`
Mac OS X: None
Windows: `/Qprof-hotness-threshold:n`

Arguments

$n$

Is the hotness threshold. $n$ is a percentage having a
value between 0 and 100 inclusive. If you specify 0, there will be no hotness threshold setting in effect for function grouping and function ordering.

**Default**

**OFF**

The compiler's default hotness threshold setting of 10 percent is in effect for function grouping.
Description

This option lets you set the hotness threshold for function grouping and function ordering.
The "hotness threshold" is the percentage of functions in the application that should be placed in the application's hot region. The hot region is the most frequently executed part of the application. By grouping these functions together into one hot region, they have a greater probability of remaining resident in the instruction cache. This can enhance the application's performance.

For this option to take effect, you must specify option -prof-use (Linux) or /Qprof-use (Windows) and one of the following:

- On Linux systems: -prof-func-groups or -prof-func-order
- On Windows systems: /Qprof-func-order

Alternate Options

None

See Also

prof-use, Qprof-use compiler option
prof-func-groups compiler option
prof-func-order, Qprof-func-order compiler option

prof-src-dir, Qprof-src-dir

Determines whether directory information of the source file under compilation is considered when looking up profile data records.

IDE Equivalent

None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: 
- -prof-src-dir
  -no-prof-src-dir
Windows: 
/Qprof-src-dir
/Qprof-src-dir-

Arguments

None

Default

-prof-src-dir
or/Qprof-src-dir

Directory information is used when looking up profile data records in the .dpi file.

Description

This option determines whether directory information of the source file under compilation is considered when looking up profile data records in the .dpi file. To use this option, you must also specify option -prof-use (Linux and Mac OS X) or /Qprof-use (Windows).

If the option is enabled, directory information is considered when looking up the profile data records within the .dpi file. You can specify directory information by using one of the following options:
• **Linux and Mac OS X:** `-prof-src-root` or `-prof-src-root-cwd`  
• **Windows:** `/Qprof-src-root` or `/Qprof-src-root-cwd`

If the option is disabled, directory information is ignored and only the name of the file is used to find the profile data record.

Note that options `-prof-src-dir` (Linux and Mac OS X) and `/Qprof-src-dir` (Windows) control how the names of the user's source files get represented within the .dyn or .dpir files. Options `-prof-dir` (Linux and Mac OS X) and `/Qprof-dir` (Windows) specify the location of the .dyn or the .dpir files.

**Alternate Options**

None

**See Also**

`prof-use, Qprof-use` compiler option  
`prof-src-root, Qprof-src-root` compiler option  
`prof-src-root-cwd, Qprof-src-root-cwd` compiler option

**prof-src-root, Qprof-src-root**

Lets you use relative directory paths when looking up profile data and specifies a directory as the base.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-prof-src-root=dir`  
Windows: `/Qprof-src-root:dir`

**Arguments**
dir                  Is the base for the relative paths.

Default
OFF                  The setting of relevant options determines the path used when looking up profile data records.

Description
This option lets you use relative directory paths when looking up profile data in .dpi files. It lets you specify a directory as the base. The paths are relative to a base directory specified during the -prof-gen (Linux and Mac OS X) or /Qprof-gen (Windows) compilation phase.

This option is available during the following phases of compilation:
- Linux and Mac OS X: -prof-gen and -prof-use phases
- Windows: /Qprof-gen and /Qprof-use phases

When this option is specified during the -prof-gen or /Qprof-gen phase, it stores information into the .dyn or .dpi file. Then, when .dyn files are merged together or the .dpi file is loaded, only the directory information below the root directory is used for forming the lookup key.
When this option is specified during the -prof-use or /Qprof-use phase, it specifies a root directory that replaces the root directory specified at the -prof-gen or /Qprof-gen phase for forming the lookup keys.

To be effective, this option or option -prof-src-root-cwd (Linux and Mac OS X) or /Qprof-src-root-cwd (Windows) must be specified during the -prof-gen or /Qprof-gen phase. In addition, if one of these options is not specified, absolute paths are used in the .dpi file.

Alternate Options

None

Consider the initial -prof-gen compilation of the source file

c:\user1\feature_foo\myproject\common\glob.c:

```
icc -prof-gen -prof-src-root=c:\user1\feature_foo\myproject -c common\glob.c
```

For the -prof-use phase, the file glob.c could be moved into the directory

c:\user2\feature_bar\myproject\common\glob.c and profile information would be found from the .dpi when using the following:

```
icc -prof-use -prof-src-root=c:\user2\feature_bar\myproject -c common\glob.c
```

If you do not use option -prof-src-root during the -prof-gen phase, by default, the -prof-use compilation can only find the profile data if the file is compiled in the c:\user1\feature_foo\my_project\common directory.

See Also

- prof-gen, Qprof-gen compiler option
- prof-use, Qprof-use compiler option
- prof-src-dir, Qprof-src-dir compiler option
- prof-src-root-cwd, Qprof-src-root-cwd compiler option

prof-src-root-cwd, Qprof-src-root-cwd

Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -prof-src-root-cwd
Windows: /Qprof-src-root-cwd

Arguments
None

Default
OFF

The setting of relevant options determines the path used when looking up profile data records.

Description
This option lets you use relative directory paths when looking up profile data in .dpi files. It specifies the current working directory as the base. To use this option, you must also specify option -prof-use (Linux and Mac OS) or /Qprof-use (Windows).

This option is available during the following phases of compilation:
- Linux and Mac OS X: -prof-gen and -prof-use phases
• **Windows: */Qprof-gen* and */Qprof-use phases**

When this option is specified during the `-prof-gen` or `/Qprof-gen` phase, it stores information into the .dyn or .dpi file. Then, when .dyn files are merged together or the .dpi file is loaded, only the directory information below the root directory is used for forming the lookup key.

When this option is specified during the `-prof-use` or `/Qprof-use` phase, it specifies a root directory that replaces the root directory specified at the `-prof-gen` or `/Qprof-gen` phase for forming the lookup keys.

To be effective, this option or option `-prof-src-root` (Linux and Mac OS X) or `/Qprof-src-root` (Windows) must be specified during the `-prof-gen` or `/Qprof-gen` phase. In addition, if one of these options is not specified, absolute paths are used in the .dpi file.

**Alternate Options**

None

**See Also**

`prof-gen, Qprof-gen` compiler option

`prof-use, Qprof-use` compiler option

`prof-src-dir, Qprof-src-dir` compiler option

`prof-src-root, Qprof-src-root` compiler option

**prof-use, Qprof-use**

Enables the use of profiling information during optimization.

**IDE Equivalent**

Windows: **General > Profile Guided Optimization**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-prof-use[=arg]`
  `-no-prof-use`

Windows: `/Qprof-use[:arg]`
  `/Qprof-use-`

Arguments

`arg` Specifies additional instructions. Possible values are:

`weighted` Tells the profmerge utility to apply a weighting to the .dyn file values when creating the .dpi file to normalize the data counts when the training runs have different execution durations. This argument only has an effect when the compiler invokes the profmerge utility to create the .dpi file. This
argument does not have an effect if the .dpi file was previously created without weighting.

[no]merge Enables or disables automatic invocation of the profmerge utility. The default is merge. Note that you cannot specify both weighted and nomerge. If you try to specify both values, a warning will be displayed and nomerge takes precedence.

default Enables the use of profiling information during optimization. The
profmerge utility is invoked by default. This value is the same as specifying –prof-use (Linux and Mac OS X) or /Qprof-use (Windows) with no argument.

**Default**

-no-prof-use or /Qprof-use-

Profiling information is not used during optimization.

**Description**

This option enables the use of profiling information (including function splitting and function grouping) during optimization. It enables option -fnsplit (Linux) or /Qfnsplit (Windows).

This option instructs the compiler to produce a profile-optimized executable and it merges available profiling output files into a pgopti.dpi file. Note that there is no way to turn off function grouping if you enable it using this option.

To set the hotness threshold for function grouping and function ordering, use option -prof-hotness-threshold (Linux) or /Qprof-hotness-threshold (Windows).
Alternate Options

None

See Also

prof-hotness-threshold, __prof-hotness-threshold__ compiler option

pthread

Tells the compiler to use pthreads library for multithreading support.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -pthread

Windows: None

Arguments

None

Default

OFF

The compiler does not use pthreads library for multithreading support.

Description
Tells the compiler to use pthreads library for multithreading support.

**Alternate Options**

None

**A, QA**

Specifies an identifier for an assertion.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-Aname[value]`

**Windows:** `/QAname[value]`

**Arguments**

- **name**
  
  Is the identifier for the assertion.

- **value**
  
  Is an optional value for the assertion. If a value is specified, it must be
within quotes, including the parentheses delimiting it.

Default

OFF

Assertions have no identifiers or symbol names.

Description

This option specifies an identifier (symbol name) for an assertion. It is equivalent to an #assert preprocessing directive.

Note that this option is not the positive form of the C++ /QA- option.

Alternate Options

None

Example

To make an assertion for the identifier fruit with the associated values orange and banana use the following command.

On Windows* systems:

icl /QA"fruit(orange,banana)" progl.cpp

On Linux* and Mac OS* X systems:

icpc -A"fruit(orange,banana)" progl.cpp

A-, QA-
Disables all predefined macros. This is a deprecated option.

**IDE Equivalent**

Windows: None

**Linux:** Preprocessor > Undefine All Preprocessor Definitions

**Mac OS X:** Preprocessor > Undefine All Preprocessor Definitions

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-A-

Windows: `/QA-`

**Arguments**

None

**Default**

OFF

Predefined macros remain enabled.

**Description**

This option disables all predefined macros. It causes all predefined macros and assertions to become inactive.

Note that this option is not the negative form of the C++ `/QA` option.

**Alternate Options**

None
fargument-alias, Qalias-args

Determines whether function arguments can alias each other.

IDE Equivalent

Windows: None
Linux: **Data > Enable Argument Aliasing**
Mac OS X: **Data > Enable Argument Aliasing**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-fargument-alias`
  `-fargument-noalias`

Windows: `/Qalias-args`
  `/Qalias-args-`

Arguments

None

Default

`-fargument-alias` or `/Qalias-args`

Function arguments can alias each other and can alias global storage.

Description
This option determines whether function arguments can alias each other. If you specify `-fargument-noalias` or `/Qalias-args-`, function arguments cannot alias each other, but they can alias global storage.

On Linux and Mac OS X systems, you can also disable aliasing for global storage, by specifying option `-fargument-noalias-global`.

**Alternate Options**

Linux and Mac OS X: `-no-alias-args` (this is a deprecated option)

Windows: None

**See Also**

- `fargument-noalias-global` compiler option
- `alias-const`, `Qalias-const`

Determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const.

**IDE Equivalent**

Windows: None

Linux: **Data > Assume Restrict Semantics for Const**

Mac OS X: **Data > Assume Restrict Semantics for Const**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-alias-const`

`-no-alias-const`

Windows: `/Qalias-const`

`/Qalias-const-`

**Arguments**

680
None

Default

-no-alias-const
or/Qalias-const-

The compiler uses standard C/C++ rules for the interpretation of const.

Description

This option determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const. It implies an additional attribute for const.

This functionality complies with the input/output buffer rule, which assumes that input and output buffer arguments do not overlap. This option allows the compiler to do some additional optimizations with those parameters.

In C99, you can also get the same result if you additionally declare your pointer parameters with the restrict keyword.

Alternate Options

None

ansi-alias, Qansi-alias

Enable use of ANSI aliasing rules in optimizations.

IDE Equivalent

Windows: None
Linux: Language > Enable Use of ANSI Aliasing Rules in Optimizations
Mac OS X: Language > Enable ANSI Aliasing
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -ansi-alias
                      -no-ansi-alias
Windows:              /Qansi-alias
                      /Qansi-alias-

Arguments
None

Default
-no-ansi-alias or /Qansi-alias-

Description
This option tells the compiler to assume that the program adheres to ISO C Standard aliasability rules.
If your program adheres to these rules, then this option allows the compiler to optimize more aggressively. If it doesn't adhere to these rules, then it can cause the compiler to generate incorrect code.

Alternate Options
None

auto-ilp32, Qauto-ilp32
Instructs the compiler to analyze the program to determine if there are 64-bit pointers which can be safely shrunk into 32-bit pointers.

**IDE Equivalent**

None

**Architectures**

Intel® 64 architecture, IA-64 architecture

**Syntax**

Linux and Mac OS X:  -auto-ilp32
Windows:  /Qauto-ilp32

**Arguments**

None

**Default**

OFF

The optimization is not attempted.

**Description**

This option instructs the compiler to analyze and transform the program so that 64-bit pointers are shrunk to 32-bit pointers, and 64-bit longs (on Linux) are shrunk into 32-bit longs wherever it is legal and safe to do so. In order for this option to be effective the compiler must be able to optimize using the -ipo/ -Qipo option and must be able to analyze all library/external calls the program makes.

This option requires that the size of the program executable never exceeds \(2^{32}\) bytes and all data values can be represented within 32 bits. If the program can
run correctly in a 32-bit system, these requirements are implicitly satisfied. If the program violates these size restrictions, unpredictable behavior might occur.

Alternate Options

None

ax, Qax

Tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit.

IDE Equivalent

Windows: Code Generation > Add Processor-Optimized Code Path
Linux: Code Generation > Add Processor-Optimized Code Path
Mac OS X: Code Generation > Add Processor-Optimized Code Path

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -axprocessor
Windows: /Qaxprocessor

Arguments

processor

Indicates the processor for which code is generated. The following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE).
Possible values are:

**SSE4.2** Can generate
Intel® SSE4
Efficient
Accelerated
String and Text
Processing
instructions
supported by
Intel® Core™ i7
processors. Can
generate Intel®
SSE4 Vectorizing
Compiler and
Media
Accelerator,
Intel® SSSE3,
SSE3, SSE2, and
SSE instructions
and it can
optimize for the
Intel® Core™
processor family.

**SSE4.1** Can generate
Intel® SSE4
Vectorizing
Compiler and
Media
Accelerator
instructions for
Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.

**SSSE3** Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. This replaces value T, which is deprecated.

**SSE3** Can generate Intel® SSE3,
SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. This replaces value P, which is deprecated.

**SSE2** Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2. This value is not available on Mac OS X systems.
This replaces value N, which is deprecated.

Default

OFF

No auto-dispatch code is generated. Processor-specific code is generated and is controlled by the setting of compiler option `-m` (Linux), compiler option `/arch` (Windows), or compiler option `-x` (Mac OS X).

Description
This option tells the compiler to generate multiple, processor-specific auto-
dispatch code paths for Intel processors if there is a performance benefit. It also
generates a baseline code path. The baseline code is usually slower than the
specialized code.

The baseline code path is determined by the architecture specified by the \(-x\)
(Linux and Mac OS X) or \(/Qx\) (Windows) option. While there are defaults for the
\(-x\) or \(/Qx\) option that depend on the operating system being used, you can
specify an architecture for the baseline code that is higher or lower than the
default. The specified architecture becomes the effective minimum architecture
for the baseline code path.

If you specify both the \(-ax\) and \(-x\) options (Linux and Mac OS X) or the \(/Qax\)
and \(/Qx\) options (Windows), the baseline code will only execute on processors
compatible with the processor type specified by the \(-x\) or \(/Qx\) option.

This option tells the compiler to find opportunities to generate separate versions
of functions that take advantage of features of the specified Intel® processor.

If the compiler finds such an opportunity, it first checks whether generating a
processor-specific version of a function is likely to result in a performance gain. If
this is the case, the compiler generates both a processor-specific version of a
function and a baseline version of the function. At run time, one of the versions is
chosen to execute, depending on the Intel processor in use. In this way, the
program can benefit from performance gains on more advanced Intel processors,
while still working properly on older processors.

You can use more than one of the processor values by combining them. For
example, you can specify \(-axSSE4.1,SSSE3\) (Linux and Mac OS X) or
\(/QaxSSE4.1,SSSE3\) (Windows). You cannot combine the old style, deprecated
options and the new options. For example, you cannot specify \(-axSSE4.1,T\)
(Linux and Mac OS X) or \(/QaxSSE4.1,T\) (Windows).

Previous values W and K are deprecated. The details on replacements are as
follows:
Mac OS X systems: On these systems, there is no exact replacement for W or K. You can upgrade to the default option -msse3 (IA-32 architecture) or option -mssse3 (Intel® 64 architecture).

Windows and Linux systems: The replacement for W is -msse2 (Linux) or /arch:SSE2 (Windows). There is no exact replacement for K. However, on Windows systems, /QaxK is interpreted as /arch:IA32; on Linux systems, -axK is interpreted as -mia32. You can also do one of the following:

- Upgrade to option -msse2 (Linux) or option /arch:SSE2 (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors
- Specify the two option combination -mia32 -axSSE2 (Linux) or /arch:IA32 /QaxSSE2 (Windows). This combination will produce an executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path.

The -ax and /Qax options enable additional optimizations not enabled with option -m or option /arch.

Alternate Options
None

See Also

x, Qx compiler option
m compiler option
arch compiler option

c99, Qc99

Determines whether C99 support is enabled for C programs. This is a deprecated option.

IDE Equivalent
Windows: **Language > Enable C99 Support**
Linux: **Language > Disable C99 Support**
Mac OS X: **Language > Disable C99 Support**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-c99
  -no-c99

Windows: `/Qc99
  `/Qc99`

**Arguments**
None

**Default**

- `-no-c99` or `/Qc99`

C99 support is disabled for C programs on Linux.

**Description**

This option determines whether C99 support is enabled for C programs. One of the features enabled by `-c99` (Linux and Mac OS X) or `/Qc99` (Windows), restricted pointers, is available by using option `restrict`. For more information, see `restrict`. 
Alternate Options
-std, /Qstd

See Also
restrict, /Qrestrict compiler option

Qchkstk
Enables stack probing when the stack is dynamically expanded at run-time.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux and Mac OS X: None
Windows: /Qchkstk
          /Qchkstk-

Arguments
None

Default
/Qchkstk

Stack probing is enabled when the stack is dynamically expanded
at run-time.

Description

This option enables stack probing when the stack is dynamically expanded at run-time.
It instructs the compiler to generate a call to _chkstk. The call will probe the requested memory and detect possible stack overflow.
To cancel the call to _chkstk, specify /Qchkstk-.

Alternate Options

None

complex-limited-range, Qcomplex-limited-range

Determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled.

IDE Equivalent

Windows: Floating Point > Limit COMPLEX Range
Linux: None
Mac OS X: Floating Point > Limit COMPLEX Range

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -complex-limited-range
- no-complex-limited-range

Windows: /Qcomplex-limited-range
/ Qcomplex-limited-range-
None

Default

-no-complex-limited-range

or/Qcomplex-limited-range-

Basic

algebraic

expansions

of some

arithmetic

operations

involving

data of

type

COMPLEX

are

disabled.

Description

This option determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled. When the option is enabled, this can cause performance improvements in programs that use a lot of COMPLEX arithmetic. However, values at the extremes of the exponent range may not compute correctly.

Alternate Options

None

Qcxx-features

Enables standard C++ features without disabling Microsoft features.

IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Qcxx-features

Arguments

None

Default

OFF

The compiler enables standard C++ features.

Description

This option enables standard C++ features without disabling Microsoft features within the bounds of what is provided in the Microsoft headers and libraries. This option has the same effect as specifying /GX /GR.

Alternate Options

None

diag, Qdiag

Controls the display of diagnostic information.

IDE Equivalent
Windows: **Diagnostics > Disable Specific Diagnostics** (/Qdiag-disable id)

**Diagnostics > Level of Static Analysis** (/Qdiag-enable[:sv1,sv2, sv3])

Linux: **Compilation Diagnostics > Disable Specific Diagnostics** (-diag-disable id)

**Compilation Diagnostics > Level of Static Analysis** (-diag-enable [sv1,sv2, sv3] or -diag-disable sv)

Mac OS X: **Diagnostics > Disable Specific Diagnostics** (-diag-disable id)

**Diagnostics > Level of Static Analysis** (-diag-enable [sv1,sv2, sv3])

Architectures

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-diag-type diag-list`

Windows: `/Qdiag-type:diag-list`

**Arguments**

`type`  
Is an action to perform on diagnostics. Possible values are:

- `enable`  
  Enables a diagnostic message or a group of messages.

- `disable`  
  Disables a diagnostic
message or a group of messages.

**error**
Tells the compiler to change diagnostics to errors.

**warning**
Tells the compiler to change diagnostics to warnings.

**remark**
Tells the compiler to change diagnostics to remarks (comments).

diag-list
Is a diagnostic group or ID value. Possible values are:

**driver**
Specifies diagnostic messages issued by the compiler driver.

**port-linux**
Specifies
diagnostic messages for language features that may cause errors when porting to Linux. This diagnostic group is only available on Windows systems.

`port-win` Specifies diagnostic messages for GNU extensions that may cause errors when porting to Windows. This diagnostic group is only
available on Linux and Mac OS X systems.

**thread** Specifies diagnostic messages that help in thread-enabling a program.

**vec** Specifies diagnostic messages issued by the vectorizer.

**par** Specifies diagnostic messages issued by the auto-parallelizer (parallel optimizer).

**sv[n]** Specifies diagnostic messages issued by
the Static Verifier. It can be any of the following: 1, 2, 3. For more details on these values, see below.

**warn** Specifies diagnostic messages that have a "warning" severity level.

**error** Specifies diagnostic messages that have an "error" severity level.

**remark** Specifies diagnostic messages that are remarks or
cpu-dispatch

Specifies the CPU dispatch remarks for diagnostic messages. These remarks are enabled by default. This diagnostic group is only available on IA-32 architecture and Intel® 64 architecture.

id[,id,...]

Specifies the ID number of one or more messages. If you specify more than one
message number, they must be separated by commas. There can be no intervening white space between each id.

tag[,tag,...] Specifies the mnemonic name of one or more messages. If you specify more than one mnemonic name, they must be separated by commas. There can be no intervening
white space between each tag.

Default

OFF

The compiler issues certain diagnostic messages by default.

Description

This option controls the display of diagnostic information. Diagnostic messages are output to stderr unless compiler option -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows) is specified. When diag-list value "warn" is used with the Static Verifier (sv) diagnostics, the following behavior occurs:

- **Option -diag-enable warn (Linux and Mac OS X) and /Qdiag-enable:warn (Windows)** enable all Static Verifier diagnostics except those that have an "error" severity level. They enable all Static Verifier warnings, cautions, and remarks.

- **Option -diag-disable warn (Linux and Mac OS X) and /Qdiag-disable:warn (Windows)** disable all Static Verifier diagnostics except those that have an "error" severity level. They suppress all Static Verifier warnings, cautions, and remarks.

The following table shows more information on values you can specify for diag-list item sv.
<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sv[n]</td>
<td>The value of n for Static Verifier messages can be any of the following:</td>
</tr>
<tr>
<td>1</td>
<td>Produces the diagnostics with severity level set to all critical errors.</td>
</tr>
<tr>
<td>2</td>
<td>Produces the diagnostics with severity level set to all errors. This is the default if n is not specified.</td>
</tr>
<tr>
<td>3</td>
<td>Produces the diagnostics with severity level set to all errors and warnings.</td>
</tr>
</tbody>
</table>

To control the diagnostic information reported by the vectorizer, use the `-vec-report` (Linux and Mac OS X) or `/Qvec-report` (Windows) option.

To control the diagnostic information reported by the auto-parallelizer, use the `-par-report` (Linux and Mac OS X) or `/Qpar-report` (Windows) option.

**Alternate Options**

| enable vec | Linux and Mac OS X: `-vec-report`  
| Windows: `/Qvec-report` |
| disable vec | Linux and Mac OS X: `-vec-report0`  
| Windows: `/Qvec-report0` |
| enable par | Linux and Mac OS X: `-par-report`  
| Windows: `/Qpar-report` |
Example

The following example shows how to enable diagnostic IDs 117, 230 and 450:

```bash
-diag-enable 117,230,450  ! Linux and Mac OS X systems
/Qdiag-enable:117,230,450  ! Windows systems
```

The following example shows how to change vectorizer diagnostic messages to warnings:

```bash
-diag-enable vec -diag-warning vec  ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-warning:vec  ! Windows systems
```

Note that you need to enable the vectorizer diagnostics before you can change them to warnings.

The following example shows how to disable all auto-parallelizer diagnostic messages:

```bash
-diag-disable par  ! Linux and Mac OS X systems
/Qdiag-disable:par  ! Windows systems
```

The following example shows how to produce Static Verifier diagnostic messages for all critical errors:

```bash
-diag-enable sv1  ! Linux and Mac OS X systems
/Qdiag-enable:sv1  ! Windows system
```

The following example shows how to cause Static Verifier diagnostics (and default diagnostics) to be sent to a file:

```bash
-diag-enable sv -diag-file=stat_ver_msg  ! Linux and Mac OS X systems
/Qdiag-enable:sv /Qdiag-file:stat_ver_msg ! Windows systems
```

Note that you need to enable the Static Verifier diagnostics before you can send them to a file. In this case, the diagnostics are sent to file stat_ver_msg.diag. If a file name is not specified, the diagnostics are sent to name-of-the-first-source-file.diag.

The following example shows how to change all diagnostic warnings and remarks to errors:

```bash
-diag-error warn,remark  ! Linux and Mac OS X systems
/Qdiag-error:warn,remark  ! Windows systems
```
See Also

diag-dump, Qdiag-dump compiler option
diag-id-numbers, Qdiag-id-numbers compiler option
diag-file, Qdiag-file compiler option
par-report, Qpar-report compiler option
vec-report, Qvec-report compiler option

diag-dump, Qdiag-dump

Tells the compiler to print all enabled diagnostic messages and stop compilation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -diag-dump
Windows:       /Qdiag-dump

Arguments

None

Default

OFF

The compiler issues certain diagnostic messages by
default.

Description

This option tells the compiler to print all enabled diagnostic messages and stop compilation. The diagnostic messages are output to stdout. This option prints the enabled diagnostics from all possible diagnostics that the compiler can issue, including any default diagnostics. If -diag-enable diag-list (Linux and Mac OS X) or /Qdiag-enable diag-list (Windows) is specified, the print out will include the diag-list diagnostics.

Alternate Options

None

Example

The following example adds vectorizer diagnostic messages to the printout of default diagnostics:

- diag-enable vec -diag-dump               ! Linux and Mac OS X systems
/Qdiag-enable:vec /Qdiag-dump               ! Windows systems

See Also

diag, Qdiag compiler option

diag-enable sv-incl ude, Qdiag-enable sv-incl ude

Tells the Static Verifier to analyze include files and source files when issuing diagnostic messages.

IDE Equivalent

Windows: Diagnostics > Analyze Include Files
Linux: Compilation Diagnostics > Analyze Include Files
Mac OS X: Diagnostics > Analyze Include Files

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `diag-enable sv-include`
Windows: `/Qdiag-enable sv-include`

Arguments

None

Default

OFF

The compiler issues certain diagnostic messages by default. If the Static Verifier is enabled, include files are not analyzed by default.

Description
This option tells the Static Verifier to analyze include files and source files when issuing diagnostic messages. Normally, when Static Verifier diagnostics are enabled, only source files are analyzed.

To use this option, you must also specify `-diag-enable sv` (Linux and Mac OS X) or `/Qdiag-enable:sv` (Windows) to enable the Static Verifier diagnostics.

Alternate Options

None

Example

The following example shows how to cause include files to be analyzed as well as source files:

```
-diag-enable sv -diag-enable sv-include ! Linux and Mac OS systems
/Qdiag-enable:sv /Qdiag-enable:sv-include ! Windows systems
```

In the above example, the first compiler option enables Static Verifier messages. The second compiler option causes include files referred to by the source file to be analyzed also.

See Also

diag, Qdiag compiler option

diag-error-limit, Qdiag-error-limit

Specifies the maximum number of errors allowed before compilation stops.

IDE Equivalent

Windows: **Compilation Diagnostics > Error Limit**

Linux: **Compilation Diagnostics > Set Error Limit**

Mac OS X: **Compilation Diagnostics > Error Limit**

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:  
-diag-error-limit\texttt{n}
-no-diag-error-limit

Windows:  
/Qdiag-error-limit:\texttt{n}
/Qdiag-error-limit-

Arguments

\textit{n}

\textit{n} is the maximum number of error-level or fatal-level compiler errors allowed.

Default

30

A maximum of 30 error-level and fatal-level messages are allowed.

Description
This option specifies the maximum number of errors allowed before compilation stops. It indicates the maximum number of error-level or fatal-level compiler errors allowed for a file specified on the command line.

If you specify `-no-diag-error-limit` (Linux and Mac OS X) or `/Qdiag-error-limit-` (Windows) on the command line, there is no limit on the number of errors that are allowed.

If the maximum number of errors is reached, a warning message is issued and the next file (if any) on the command line is compiled.

**Alternate Options**

Linux and Mac OS X: `-wn` (this is a deprecated option)

Windows: `/Qwn` (this is a deprecated option)

**diag-file, Qdiag-file**

Causes the results of diagnostic analysis to be output to a file.

**IDE Equivalent**

Windows: Diagnostics > Diagnostics File
Linux: Compilation Diagnostics > Diagnostics File
Mac OS X: Diagnostics > Diagnostics File

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-diag-file[=file]`
Windows: `/Qdiag-file[:file]`

**Arguments**

`file` Is the
name of the file for output.

Default

OFF   Diagnostic messages are output to stderr.

Description

This option causes the results of diagnostic analysis to be output to a file. The file is placed in the current working directory.

If `file` is specified, the name of the file is `file.diag`. The file can include a file extension; for example, if `file.ext` is specified, the name of the file is `file.ext`. If `file` is not specified, the name of the file is `name-of-the-first-source-file.diag`. This is also the name of the file if the name specified for file conflicts with a source file name provided in the command line.

Note

If you specify `-diag-file` (Linux and Mac OS X) or `/Qdiag-file` (Windows) and you also specify `-diag-file-append` (Linux and Mac OS X) or `/Qdiag-file-append` (Windows), the last option specified on the command line takes precedence.

Alternate Options

None

Example
The following example shows how to cause diagnostic analysis to be output to a file named `my_diagnostic.diag`:

```
-dia=diag-file=my_diagnostic       ! Linux and Mac OS X systems
/Qdiag-file:my_diagnostic         ! Windows systems
```

See Also

- `diag`, `Qdiag` compiler option
- `diag-file-append`, `Qdiag-file-append` compiler option

**diag-file-append, Qdiag-file-append**

Causes the results of diagnostic analysis to be appended to a file.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-diag-file-append[=file]`

Windows: `/Qdiag-file-append[:file]`

**Arguments**

`file`

Is the name of the file to be appended to. It can include a path.

**Default**
OFF  

Diagnostic messages are output to stderr.

Description

This option causes the results of diagnostic analysis to be appended to a file. If you do not specify a path, the driver will look for file in the current working directory.

If file is not found, then a new file with that name is created in the current working directory. If the name specified for file conflicts with a source file name provided in the command line, the name of the file is name-of-the-first-source-file.diag.

Note

If you specify -diag-file-append (Linux and Mac OS X) or /Qdiag-file-append (Windows) and you also specify -diag-file (Linux and Mac OS X) or /Qdiag-file (Windows), the last option specified on the command line takes precedence.

Alternate Options

None

Example

The following example shows how to cause diagnostic analysis to be appended to a file named my_diagnostics.txt:

-diag-file-append=my_diagnostics.txt       ! Linux and Mac OS X systems
/Qdiag-file-append:my_diagnostics.txt      ! Windows systems

See Also

diag, Qdiag compiler option
diag-file, Qdiag-file compiler option
diag-id-numbers, Qdiag-id-numbers

Determines whether the compiler displays diagnostic messages by using their ID number values.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
-diag-id-numbers
-no-diag-id-numbers

Windows:  
/Qdiag-id-numbers
/Qdiag-id-numbers-

Arguments

None

Default

-diag-id-numbers  
or/Qdiag-id-numbers

The compiler displays diagnostic messages by using their ID number values.

Description
This option determines whether the compiler displays diagnostic messages by using their ID number values. If you specify `-no-diag-id-numbers` (Linux and Mac OS X) or `/Qdiag-id-numbers-` (Windows), mnemonic names are output for driver diagnostics only.

**Alternate Options**

None

**See Also**

diag, Qdiag compiler option

diag-once, Qdiag-once

Tells the compiler to issue one or more diagnostic messages only once.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-diag-once id[,id,...]

Windows: `/Qdiag-once: id[,id,...]

**Arguments**

`id`

Is the ID number of the diagnostic message. If you
In default OFF, the compiler issues certain diagnostic messages by default.

Description
This option tells the compiler to issue one or more diagnostic messages only once.

Alternate Options

Linux: `-wo` (this is a deprecated option)
Windows: `/Qwo` (this is a deprecated option)

dD, QdD

Same as `-dM`, but outputs `#define` directives in preprocessed source.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-dD`
Windows: `/QdD`

Arguments

None

Default

OFF

The compiler does not output `#define` directives.
Description

Same as -dM, but outputs #define directives in preprocessed source. To use this option, you must also specify the E option.

Alternate Options

None

dM, QdM

Tells the compiler to output macro definitions in effect after preprocessing.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -dM
Windows: /QdM

Arguments

None

Default

OFF

The compiler does not output macro definitions after preprocessing.
**Description**

This option tells the compiler to output macro definitions in effect after preprocessing. To use this option, you must also specify the E option.

**Alternate Options**

None

**See Also**

E compiler option

dN, QdN

Same as -dD, but output #define directives contain only macro names.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: -dN

Windows: /QdN

**Arguments**

None

**Default**

OFF

The compiler does not output
#define directives.

Description

Same as -dD, but output #define directives contain only macro names. To use this option, you must also specify the E option.

Alternate Options

None

Weffc++, Qeffc++

This option enables warnings based on certain C++ programming guidelines.

IDE Equivalent

Windows: None
Linux: Compilation Diagnostics > Enable Warnings for Style Guideline Violations
Mac OS X: Diagnostics > Report Effective C++ Violations

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Weffc++
Windows: /Qeffc++

Arguments

None

Default
Description

This option enables warnings based on certain programming guidelines developed by Scott Meyers in his books on effective C++ programming. With this option, the compiler emits warnings for these guidelines:

- Use `const` and `inline` rather than `#define`. Note that you will only get this in user code, not system header code.
- Use `<iostream>` rather than `<stdio.h>`.
- Use `new` and `delete` rather than `malloc` and `free`.
- Use C++ style comments in preference to C style comments. C comments in system headers are not diagnosed.
- Use `delete` on pointer members in destructors. The compiler diagnoses any pointer that does not have a `delete`.
- Make sure you have a user copy constructor and assignment operator in classes containing pointers.
- Use initialization rather than assignment to members in constructors.
- Make sure the initialization list ordering matches the declaration list ordering in constructors.
- Make sure base classes have virtual destructors.
- Make sure `operator=` returns `*this`.
- Make sure prefix forms of increment and decrement return a `const` object.
- Never overload operators `&&`, `||`, and `,`.

Note

The warnings generated by this compiler option are based on the following books from Scott Meyers:
• Effective C++ Second Edition - 50 Specific Ways to Improve Your Programs and Designs
• More Effective C++ - 35 New Ways to Improve Your Programs and Designs

Alternate Options

None

fast-transcendentals, Qfast-transcendentals

Enables the compiler to replace calls to transcendental functions with faster but less precise implementations.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -fast-transcendentals
                      -no-fast-transcendentals
Windows:            /Qfast-transcendentals
                      /Qfast-transcendentals-

Default

-fast-transcendentals or /Qfast-transcendentals

The default depends on the setting of –fp-model (Linux and Mac OS X) or

This option enables the compiler to replace calls to transcendental functions with implementations that may be faster but less precise. It tells the compiler to perform certain optimizations on transcendental functions, such as replacing individual calls to sine in a loop with a single call to a less precise vectorized sine library routine.

This option has an effect only when specified with one of the following options:

- **Windows OS**: `/fp:except` or `/fp:precise`
- **Linux OS and Mac OS X**: `-fp-model except` or `-fp-model precise`

You cannot use this option with option `-fp-model strict` (Linux and Mac OS X) or `/fp:strict` (Windows).
None

See Also

`fp-model, fp` compiler option

**fma, Qfma**

Enables the combining of floating-point multiplies and add/subtract operations.

**IDE Equivalent**

Windows: None

Linux: **Floating Point > Floating-point Operation Contraction**

Mac OS X: None

**Architectures**

IA-64 architecture

**Syntax**

Linux: 
- `-fma`
- `-no-fma`

Mac OS X: None

Windows: 
- `/Qfma`
- `/Qfma-`

**Arguments**

None

**Default**

- `-fma`
  Floating-point
  multiplies
  and

- `/Qfma`
add/subtract operations are combined. However, if you specify -mp (Linux), /Op (Windows), /fp:strict (Windows), or -fp-model strict (Linux) but do not explicitly specify -fma or /Qfma, the default is -no-fma or /Qfma-

**Description**

This option enables the combining of floating-point multiplies and add/subtract operations. It also enables the contraction of floating-point multiply and add/subtract operations into a single operation. The compiler contracts these operations whenever possible.

**Alternate Options**
Linux: -IPF-fma (this is a deprecated option)
Windows: /QIPF-fma (this is a deprecated option)

See Also

fp-model, fp compiler option
Floating-point Operations: Floating-point Options Quick Reference

falign-functions, Qfnalign

Tells the compiler to align functions on an optimal byte boundary.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -falign-functions[=n]
-fno-align-functions

Windows: /Qfnalign[=n]
/Qfnalign-

Arguments

\( n \)

| Is the byte boundary for function alignment. |
| Possible values are 2 or 16. |
Default

-fno-align-functions or /Qfналign-

The compiler aligns functions on 2-byte boundaries. This is the same as specifying -fnalign-functions=2 (Linux and Mac OS X) or /Qfnalign:2 (Windows).

Description

This option tells the compiler to align functions on an optimal byte boundary. If you do not specify $n$, the compiler aligns the start of functions on 16-byte boundaries.

Alternate Options

None

fnsplit, Qfnsplit

Enables function splitting.

IDE Equivalent

Windows: **Code Generation > Disable Function Splitting**
Linux: None
Mac OS X: None
Architectures

/Qfnsplit[-]: IA-32 architecture, Intel® 64 architecture
-[no-]fnsplit: IA-64 architecture

Syntax

Linux: 

- fnsplit
- no-fnsplit

Mac OS X: None

Windows: 

/Qfnsplit
/Qfnsplit-

Arguments

None

Default

-no-fnsplit or /Qfnsplit- Function splitting is not enabled unless –prof-use (Linux) or /Qprof-use (Windows) is also specified.

Description
This option enables function splitting if `-prof-use` (Linux) or `/Qprof-use` (Windows) is also specified. Otherwise, this option has no effect.

It is enabled automatically if you specify `-prof-use` or `/Qprof-use`. If you do not specify one of those options, the default is `-no-fnsplit` (Linux) or `/Qfnsplit-` (Windows), which disables function splitting but leaves function grouping enabled.

To disable function splitting when you use `-prof-use` or `/Qprof-use`, specify `-no-fnsplit` or `/Qfnsplit-`.

Alternate Options

None

**fp-port, Qfp-port**

Rounds floating-point results after floating-point operations.

IDE Equivalent

Windows: **Optimization > Floating-point Precision Improvements**
Linux: **Floating Point > Round Floating-Point Results**
Mac OS X: **Floating Point > Round Floating-Point Results**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-fp-port`

`-no-fp-port`

Windows: `/Qfp-port`

`/Qfp-port-`

Arguments

None
Default

-no-fp-port
or/Qfp-port-

The default rounding behavior depends on the compiler's code generation decisions and the precision parameters of the operating system.

Description

This option rounds floating-point results after floating-point operations. Rounding to user-specified precision occurs at assignments and type conversions. This has some impact on speed. The default is to keep results of floating-point operations in higher precision. This provides better performance but less consistent floating-point results.

Alternate Options

None

fp-relaxed, Qfp-relaxed

Enables use of faster but slightly less accurate code sequences for math functions.
IDE Equivalent
None

Architectures
IA-64 architecture

Syntax
Linux:  
-fp-relaxed
-no-fp-relaxed

Mac OS X:  None

Windows:  /Qfp-relaxed
/Qfp-relaxed-

Arguments
None

Default

-no-fp-relaxed  
/Qfp-relaxed-

Default

code
sequences
are used
for math
functions.

Description
This option enables use of faster but slightly less accurate code sequences for math functions, such as divide and sqrt. When compared to strict IEEE* precision, this option slightly reduces the accuracy of floating-point calculations performed by these functions, usually limited to the least significant digit. This option also enables the performance of more aggressive floating-point transformations, which may affect accuracy.
Alternate Options

Linux: `-IPF-fp-relaxed` (this is a deprecated option)

Windows: `/QIPF-fp-relaxed` (this is a deprecated option)

See Also

`fp-model`, `fp` compiler option

`fp-speculation`, `Qfp-speculation`

Tells the compiler the mode in which to speculate on floating-point operations.

IDE Equivalent

Windows: **Optimization > Floating-Point Speculation**

Linux: **Floating Point > Floating-Point Speculation**

Mac OS X: **Floating Point > Floating-Point Speculation**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-fp-speculation=mode`

Windows: `/Qfp-speculation:mode`

Arguments

`mode`  
Is the mode for floating-point operations. Possible values are:

`fast`  
Tells the compiler to speculate
on floating-point operations.

- **safe**
  
  Tells the compiler to disable speculation if there is a possibility that the speculation may cause a floating-point exception.

- **strict**
  
  Tells the compiler to disable speculation on floating-point operations.

- **off**
  
  This is the same as specifying strict.

**Default**

```
-fp-speculation=fast
```

The compiler
or/Qfp-speculation:fast

speculates on floating-point operations. This is also the behavior when optimizations are enabled. However, if you specify no optimizations (-O0 on Linux; /Od on Windows), the default is -fp-speculation=safe (Linux) or /Qfp-speculation:safe (Windows).

Description

This option tells the compiler the mode in which to speculate on floating-point operations.

Alternate Options

None

ffreestanding, Qfreestanding

Ensures that compilation takes place in a freestanding environment.

IDE Equivalent

None

Architectures
Syntax

Linux and Mac OS X: -ffreestanding
Windows: /Qfreestanding

Arguments
None

Default

OFF

Standard libraries are used during compilation.

Description

This option ensures that compilation takes place in a freestanding environment. The compiler assumes that the standard library may not exist and program startup may not necessarily be at main. This environment meets the definition of a freestanding environment as described in the C and C++ standard. An example of an application requiring such an environment is an OS kernel.

Note

When you specify this option, the compiler will not assume the presence of compiler-specific libraries. It will only generate calls that appear in the source code.

Alternate Options
None

ftz, Qftz

736
Flushes denormal results to zero.

IDE Equivalent

Windows: **Optimization > Flush Denormal Results to Zero**
Linux: **Floating-Point > Flush Denormal Results to Zero**
Mac OS X: **Floating-Point > Flush Denormal Results to Zero**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-ftz`
    `-no-ftz`

Windows: `/Qftz`
    `/Qftz-`

Arguments

None

Default

Systems using IA-64 architecture: `-no-ftz` or `/Qftz-`

On systems using IA-64 architecture, the compiler lets results gradually underflow.

Systems using IA-32 architecture and Intel® 64 architecture: `-ftz` or `/Qftz`

On systems using IA-32 architecture
Description

This option flushes denormal results to zero when the application is in the gradual underflow mode. It may improve performance if the denormal values are not critical to your application's behavior.

This option sets or resets the FTZ and the DAZ hardware flags. If FTZ is ON, denormal results from floating-point calculations will be set to the value zero. If FTZ is OFF, denormal results remain as is. If DAZ is ON, denormal values used as input to floating-point instructions will be treated as zero. If DAZ is OFF, denormal instruction inputs remain as is. Systems using IA-64 architecture have FTZ but not DAZ. Systems using Intel® 64 architecture have both FTZ and DAZ. FTZ and DAZ are not supported on all IA-32 architectures.

When `-ftz` (Linux and Mac OS X) or `/Qftz` (Windows) is used in combination with an SSE-enabling option on systems using IA-32 architecture (for example, `xN` or `/QxN`), the compiler will insert code in the main routine to set FTZ and DAZ. When `-ftz` or `/Qftz` is used without such an option, the compiler will insert code to conditionally set FTZ/DAZ based on a run-time processor check. `-no-` `ftz` (Linux and Mac OS X) or `/Qnoftz` (Windows) will prevent the compiler from inserting any code that might set FTZ or DAZ.

This option only has an effect when the main program is being compiled. It sets the FTZ/DAZ mode for the process. The initial thread and any threads subsequently created by that process will operate in FTZ/DAZ mode.
On systems using IA-64 architecture, optimization option O3 sets -ftz and /Qftz; optimization option O2 sets -no-ftz (Linux) and /Qftz- (Windows). On systems using IA-32 architecture and Intel® 64 architecture, every optimization option level, except O0, sets -ftz and /Qftz.

If this option produces undesirable results of the numerical behavior of your program, you can turn the FTZ/DAZ mode off by using -no-ftz or /Qftz- in the command line while still benefiting from the O3 optimizations.

**Note**

Options -ftz and /Qftz are performance options. Setting these options does not guarantee that all denormals in a program are flushed to zero. They only cause denormals generated at run time to be flushed to zero.

**Alternate Options**

None

**Example**

To see sample code showing the state of the FTZ and DAZ flags, see Reading the FTZ and DAZ Flags.

**See Also**

x, Qx compiler option

global-hoist, Qglobal-hoist

Enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -global-hoist
- no-global-hoist

Windows: /Qglobal-hoist
/ Qglobal-hoist-

Arguments
None

Default
- global-hoist
or / Qglobal-hoist

Certain optimizations are enabled that can move memory loads.

Description

This option enables certain optimizations that can move memory loads to a point earlier in the program execution than where they appear in the source. In most cases, these optimizations are safe and can improve performance. The -no-global-hoist (Linux and Mac OS X) or /Qglobal-hoist- (Windows) option is useful for some applications, such as those that use shared or dynamically mapped memory, which can fail if a load is moved too early in the execution stream (for example, before the memory is mapped).

Alternate Options
None

H, QH
Tells the compiler to display the include file order and continue compilation.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-H`

Windows: `/QH`

**Arguments**

None

**Default**

OFF

Compilation occurs as usual.

**Description**

This option tells the compiler to display the include file order and continue compilation.

**Alternate Options**

None

**help pragma, Qhelp pragma**

Displays all supported pragmas.

**IDE Equivalent**
None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: `-help-pragma`
Windows: `/Qhelp-pragma`

Arguments

None

Default

OFF

No list is displayed unless this compiler option is specified.

Description

This option displays all supported pragmas and shows their syntaxes.

Alternate Options

None

QIA64-fr32

Disables use of high floating-point registers.

IDE Equivalent
None

Architectures

IA-64 architecture

Syntax

Linux and Mac OS X: None
Windows: /QIA64-fr32

Arguments

None

Default

OFF

Use of high floating-point registers is enabled.

Description

This option disables use of high floating-point registers.

Alternate Options

None

Qlfist

See rcd, Qrcd.

inline-calloc, Qinline-calloc
Tells the compiler to inline calls to calloc() as calls to malloc() and memset().

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X:  
- `–inline-calloc`
  
- `–no-inline-calloc`

Windows:  
- `/Qinline-calloc`
  
- `/Qinline-calloc-`

**Arguments**

None

**Default**

- `–no-inline-calloc`

or `/Qinline-calloc-`

The compiler inlines calls to calloc().

This option tells the compiler to inline calls to calloc() as calls to malloc() and memset(). This enables additional memset() optimizations. For example, it can enable inlining as a sequence of store operations when the size is a compile time constant.
Alternate Options

None

inline-debug-info, Qinline-debug-info

Produces enhanced source position information for inlined code. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -inline-debug-info
Windows: /Qinline-debug-info

Arguments

None

Default

OFF

No enhanced source position information is produced for inlined code.
Description

This option produces enhanced source position information for inlined code. This leads to greater accuracy when reporting the source location of any instruction. It also provides enhanced debug information useful for function call traceback. To use this option for debugging, you must also specify a debug enabling option, such as `-g` (Linux) or `/debug` (Windows).

Alternate Options

Linux and Mac OS X: `-debug inline-debug-info`
Windows: None

**Qinline-dllimport**

Determines whether dllimport functions are inlined.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: `/Qinline-dllimport`

Arguments

None

Default

`/Qinline-dllimport`  

The dllimport

746
functions are inlined.

Description
This option determines whether dllimport functions are inlined. To disable dllimport functions from being inlined, specify /Qinline-dllimport-.

Alternate Options
None

inline-factor, Qinline-factor
Specifies the percentage multiplier that should be applied to all inlining options that define upper limits.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -inline-factor=n
  -no-inline-factor
Windows: /Qinline-factor=n
  /Qinline-factor-

Arguments

$n$ Is a positive
integer specifying the percentage value. The default value is 100 (a factor of 1).

Default

-no-inline-factor or /Qinline-factor-

The compiler uses default heuristics for inline routine expansion.

Description

This option specifies the percentage multiplier that should be applied to all inlining options that define upper limits:

- -inline-max-size and /Qinline-max-size
- -inline-max-total-size and /Qinline-max-total-size
- -inline-max-per-routine and /Qinline-max-per-routine
- -inline-max-per-compile and /Qinline-max-per-compile

This option takes the default value for each of the above options and multiplies it by \( n \) divided by 100. For example, if 200 is specified, all inlining options that
define upper limits are multiplied by a factor of 2. This option is useful if you do not want to individually increase each option limit.

If you specify `-no-inline-factor` (Linux and Mac OS X) or `/Qinline-factor-` (Windows), the following occurs:

- Every function is considered to be a small or medium function; there are no large functions.
- There is no limit to the size a routine may grow when inline expansion is performed.
- There is no limit to the number of times some routine may be inlined into a particular routine.
- There is no limit to the number of times inlining can be applied to a compilation unit.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

⚠️ Caution

When you use this option to increase default limits, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

- `inline-max-size, Qinline-max-size` compiler option
- `inline-max-total-size, Qinline-max-total-size` compiler option
- `inline-max-per-routine, Qinline-max-per-routine` compiler option
- `inline-max-per-compile, Qinline-max-per-compile` compiler option
- `opt-report, Qopt-report` compiler option

Optimizing Applications:

Developer Directed Inline Expansion of User Functions

Compiler Directed Inline Expansion of User Functions
inline-forceinline, Qinline-forceinline

Specifies that an inline routine should be inlined whenever the compiler can do so.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -inline-forceinline
Windows: /Qinline-forceinline

Default
OFF

The compiler uses default heuristics for inline routine expansion.

Description

This option specifies that an inline routine should be inlined whenever the compiler can do so. This causes the routines marked with an inline keyword or attribute to be treated as if they were "forceinline".
Because C++ member functions whose definitions are included in the class declaration are considered inline functions by default, using this option will also make these member functions "forceinline" functions. The "forceinline" condition can also be specified by using the keyword __forceinline.

To see compiler values for important inlining limits, specify compiler option –opt-report (Linux and Mac OS) or /Qopt-report (Windows).

Caution

When you use this option to change the meaning of inline to "forceinline", the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

opt-report, Qopt-report compiler option

Optimizing Applications:
Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

inline-max-per-compile, Qinline-max-per-compile

Specifies the maximum number of times inlining may be applied to an entire compilation unit.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:  
-inline-max-per-compile=n
-no-inline-max-per-compile

Windows:  
/Qinline-max-per-compile=n
/Qinline-max-per-compile- 

Arguments

\( n \)

Is a positive integer that specifies the number of times inlining may be applied.

Default

-no-inline-max-per-compile
or/Qinline-max-per-compile-

The compiler uses default heuristics for inline routine expansion.

Description
This option the maximum number of times inlining may be applied to an entire compilation unit. It limits the number of times that inlining can be applied. For compilations using Interprocedural Optimizations (IPO), the entire compilation is a compilation unit. For other compilations, a compilation unit is a file.

If you specify `-no-inline-max-per-compile` (Linux and Mac OS X) or `/Qinline-max-per-compile-` (Windows), there is no limit to the number of times inlining may be applied to a compilation unit.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

⚠️ **Caution**

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None

**See Also**

`inline-factor, Qinline-factor` compiler option

`opt-report, Qopt-report` compiler option

Optimizing Applications:

Developer Directed Inline Expansion of User Functions

Compiler Directed Inline Expansion of User Functions

`inline-max-per-routine, Qinline-max-per-routine` compiler option

Specifies the maximum number of times the inliner may inline into a particular routine.

**IDE Equivalent**
None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-inline-max-per-routine=n`
   `-no-inline-max-per-routine`

Windows:  `/Qinline-max-per-routine=n`
   `/Qinline-max-per-routine-`

Arguments

\( n \)

Is a positive integer that specifies the maximum number of times the inliner may inline into a particular routine.

Default

`-no-inline-max-per-routine`

The
or/Qinline-max-per-routine-

compiler uses default heuristics for inline routine expansion.

Description

This option specifies the maximum number of times the inliner may inline into a particular routine. It limits the number of times that inlining can be applied to any routine.

If you specify -no-inline-max-per-routine (Linux and Mac OS X) or /Qinline-max-per-routine- (Windows), there is no limit to the number of times some routine may be inlined into a particular routine.

To see compiler values for important inlining limits, specify compiler option –opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

Caution

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

inline-factor, Qinline-factor compiler option
opt-report, Qopt-report compiler option
Optimizer Applications:
Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

.inline-max-size, .Qinline-max-size

Specifies the lower limit for the size of what the inliner considers to be a large routine.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -inline-max-size=n
-no-inline-max-size

Windows: /Qinline-max-size=n
/Qinline-max-size-

Arguments

n

Is a positive integer that specifies the minimum size of what the inliner...
Default

-no-inline-max-size
or/Qinline-max-size-

The compiler uses default heuristics for inline routine expansion.

Description

This option specifies the lower limit for the size of what the inliner considers to be a large routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be medium and large-size routines.

The inliner prefers to inline small routines. It has a preference against inlining large routines. So, any large routine is highly unlikely to be inlined.

If you specify -no-inline-max-size (Linux and Mac OS X) or /Qinline-max-size- (Windows), there are no large routines. Every routine is either a small or medium routine.

To see compiler values for important inlining limits, specify compiler option –opt-report (Linux and Mac OS X) or /Qopt-report (Windows).

To see compiler values for important inlining limits, specify compiler option –opt-report (Linux and Mac OS X) or /Qopt-report (Windows).
Caution

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

`inline-min-size, Qinline-min-size` compiler option
`inline-factor, Qinline-factor` compiler option
`opt-report, Qopt-report` compiler option

Optimizing Applications:

Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

`inline-max-total-size, Qinline-max-total-size`

Specifies how much larger a routine can normally grow when inline expansion is performed.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  `-inline-max-total-size=n`
  `-no-inline-max-total-size`

Windows:  `/Qinline-max-total-size=n`
  `/Qinline-max-total-size-`
Arguments

\( n \)

Is a positive integer that specifies the permitted increase in the routine's size when inline expansion is performed.

Default

-no-inline-max-total-size

or/Qinline-max-total-size-

The compiler uses default heuristics for inline routine expansion.

Description

This option specifies how much larger a routine can normally grow when inline expansion is performed. It limits the potential size of the routine. For example, if
2000 is specified for $n$, the size of any routine will normally not increase by more than 2000.

If you specify `-no-inline-max-total-size` (Linux and Mac OS X) or `/Qinline-max-total-size-` (Windows), there is no limit to the size a routine may grow when inline expansion is performed.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

⚠️ **Caution**

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

**Alternate Options**

None

**See Also**

- `inline-factor, Qinline-factor` compiler option
- `opt-report, Qopt-report` compiler option

**Optimizing Applications:**

Developer Directed Inline Expansion of User Functions

Compiler Directed Inline Expansion of User Functions

- `inline-min-size, Qinline-min-size`

Specifies the upper limit for the size of what the inliner considers to be a small routine.

**IDE Equivalent**

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
- \texttt{-inline-min-size=}n
  \texttt{-no-inline-min-size}

Windows:  
\texttt{/Qinline-min-size=}n
\texttt{/Qinline-min-size-}

Arguments

\textit{n}

Is a positive integer that specifies the maximum size of what the inliner considers to be a small routine.

Default

\texttt{-no-inline-min-size}

or \texttt{/Qinline-min-size-}

The compiler uses
Description

This option specifies the upper limit for the size of what the inliner considers to be a small routine (a function). The inliner classifies routines as small, medium, or large. This option specifies the boundary between what the inliner considers to be small and medium-size routines.

The inliner has a preference to inline small routines. So, when a routine is smaller than or equal to the specified size, it is very likely to be inlined.

If you specify `-no-inline-min-size` (Linux and Mac OS X) or `/Qinline-min-size-` (Windows), there is no limit to the size of small routines. Every routine is a small routine; there are no medium or large routines.

To see compiler values for important inlining limits, specify compiler option `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

Caution

When you use this option to increase the default limit, the compiler may do so much additional inlining that it runs out of memory and terminates with an "out of memory" message.

Alternate Options

None

See Also

`inline-min-size, Qinline-min-size` compiler option

762
**opt-report, Qopt-report** compiler option

Optimizing Applications:
Developer Directed Inline Expansion of User Functions
Compiler Directed Inline Expansion of User Functions

**Qinstall**

Specifies the root directory where the compiler installation was performed.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-Qinstall dir`
Windows: None

**Arguments**

`dir`  
Is the root directory where the installation was performed.

**Default**

OFF  
The default root
directory
for
compiler
installation
is
searched
for the
compiler.

Description

This option specifies the root directory where the compiler installation was performed. It is useful if you want to use a different compiler or if you did not use the iccvars shell script to set your environment variables.

Alternate Options

None

**minstruction, Qinstruction**

Determines whether MOVBE instructions are generated for Intel processors.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

**Linux and Mac OS X:** `-minstruction=[no]movbe`

**Windows:** `/Qinstruction:[no]movbe`

Arguments
None

Default

-minstruction=movbe  The
or/Qinstruction:movbe  compiler
generates
MOVBE
instructions
for Intel®
Atom™
processors.

Description

This option determines whether MOVBE instructions are generated for Intel processors. To use this option, you must also specify -xSSE3_ATOM (Linux and Mac OS X) or /QxSSE3_ATOM (Windows).

If -minstruction=movbe or /Qinstruction:movbe is specified, the following occurs:

- MOVBE instructions are generated that are specific to the Intel® Atom™ processor.
- The options are ON by default when -xSSE3_ATOM or /QxSSE3_ATOM is specified.
- Generated executables can only be run on Intel® Atom™ processors or processors that support Intel® Streaming SIMD Extensions 3 (Intel® SSE3) and MOVBE.

If -minstruction=nomovbe or /Qinstruction:nomovbe is specified, the following occurs:

- The compiler optimizes code for the Intel® Atom™ processor, but it does not generate MOVBE instructions.
- Generated executables can be run on non-Intel® Atom™ processors that support Intel® SSE3.
Alternate Options

None

See Also

x, Qx compiler option

finstrument-functions, Qinstrument-functions

Determines whether function entry and exit points are instrumented.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -finstrument-functions
                 -fno-instrument-functions

Windows:        /Qinstrument-functions
                 /Qinstrument-functions-

Arguments

None

Default

-fno-instrument-functions
or/Qinstrument-functions-

Function entry and exit points are not instrumented.

Description
This option determines whether function entry and exit points are instrumented. It may increase execution time.

The following profiling functions are called with the address of the current function and the address of where the function was called (its "call site"):

- **This function is called upon function entry:**
  - On IA-32 architecture and Intel® 64 architecture:
    ```c
    void __cyg_profile_func_enter (void *this_fn, void *call_site);
    ```
  - On IA-64 architecture:
    ```c
    void __cyg_profile_func_enter (void **this_fn, void *call_site);
    ```

- **This function is called upon function exit:**
  - On IA-32 architecture and Intel® 64 architecture:
    ```c
    void __cyg_profile_func_exit (void *this_fn, void *call_site);
    ```
  - On IA-64 architecture:
    ```c
    void __cyg_profile_func_exit (void **this_fn, void *call_site);
    ```

On IA-64 architecture, the additional de-reference of the function pointer argument is required to obtain the function entry point contained in the first word of the function descriptor for indirect function calls. The descriptor is documented in the Intel® Itanium® Software Conventions and Runtime Architecture Guide, section 8.4.2. You can find this design guide at web site [http://www.intel.com](http://www.intel.com).

These functions can be used to gather more information, such as profiling information or timing information. Note that it is the user's responsibility to provide these profiling functions.

If you specify `-finstrument-functions` (Linux and Mac OS X) or `/Qinstrument-functions` (Windows), function inlining is disabled. If you specify `-fno-instrument-functions` or `/Qinstrument-functions-`, inlining is not disabled.

On Linux and Mac OS X systems, you can use the following attribute to stop an individual function from being instrumented:

```
__attribute__((__no_instrument_function__))
```

It also stops inlining from being disabled for that individual function.

This option is provided for compatibility with gcc.
Alternate Options

None

ip, Qip

Determines whether additional interprocedural optimizations for single-file compilation are enabled.

IDE Equivalent

Windows: Optimization > Interprocedural Optimization
Linux: Optimization > Enable Interprocedural Optimizations for Single File Compilation
Mac OS X: Optimization > Enable Interprocedural Optimization for Single File Compilation

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -ip
- no-ip

Windows: /Qip
/ Qip-

Arguments

None

Default

OFF  Some limited interprocedural optimizations
occur, including inline function expansion for calls to functions defined within the current source file. These optimizations are a subset of full intra-file interprocedural optimizations.

Note that this setting is not the same as `-no-ip` (Linux and Mac OS X) or `/Qip-` (Windows).

**Description**

This option determines whether additional interprocedural optimizations for single-file compilation are enabled. Options `-ip` (Linux and Mac OS X) and `/Qip` (Windows) enable additional interprocedural optimizations for single-file compilation. Options `-no-ip` (Linux and Mac OS X) and `/Qip-` (Windows) may not disable inlining. To ensure that inlining of user-defined functions is disabled, specify -
inline-level=0 or -fno-inline (Linux and Mac OS X), or specify /Ob0 (Windows). To ensure that inlining of compiler intrinsic functions is disabled, specify -fno-builtin (Linux and MacOS X) or /Oi- (Windows).

Alternate Options

None

See Also

finline-functions compiler option

ip-no-inlining, Qip-no-inlining

Disables full and partial inlining enabled by interprocedural optimization options.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -ip-no-inlining
Windows: /Qip-no-inlining

Arguments

None

Default

OFF

Inlining enabled by interprocedural optimization
options is performed.

Description

This option disables full and partial inlining enabled by the following interprocedural optimization options:

- **On Linux and Mac OS X systems:** `-ip` or `-ipo`
- **On Windows systems:** `/Qip`, `/Qipo`, or `/Ob2`

It has no effect on other interprocedural optimizations.

On Windows systems, this option also has no effect on user-directed inlining specified by option `/Ob1`.

Alternate Options

None

**ip-no-pinlining, Qip-no-pinlining**

Disables partial inlining enabled by interprocedural optimization options.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: `-ip-no-pinlining`

Windows: `/Qip-no-pinlining`

Arguments

None
Default

OFF

Description

This option disables partial inlining enabled by the following interprocedural optimization options:

- **On Linux and Mac OS X systems:** `-ip` or `-ipo`
- **On Windows systems:** `/Qip` or `/Qipo`

It has no effect on other interprocedural optimizations.

Alternate Options

None

**IPF-flt-eval-method0, QIPF-flt-eval-method0**

Tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

**Linux:** `-IPF-flt-eval-method0`
Mac OS X: None
Windows: /QIPF-flt-eval-method0

Arguments
None

Default
OFF

Expressions involving floating-point operands are evaluated by default rules.

Description
This option tells the compiler to evaluate the expressions involving floating-point operands in the precision indicated by the variable types declared in the program.
By default, intermediate floating-point expressions are maintained in higher precision.
The recommended method to control the semantics of floating-point calculations is to use option -fp-model (Linux) or /fp (Windows).
Instead of using -IPF-flt-eval-method0 (Linux) or /QIPF-flt-eval-method0 (Windows), you can use -fp-model source (Linux) or /fp:source (Windows).

Alternate Options
None

See Also

def-p-model, fp compiler option

IPF-fltacc, QIPF-fltacc

Disables optimizations that affect floating-point accuracy. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:    -IPF-fltacc
          -no-IPF-fltacc

Mac OS X: None

Windows:  /QIPF-fltacc
          /QIPF-fltacc-

Arguments

None

Default

-no-IPF-fltacc       or /QIPF-fltacc-

Optimizations are enabled that affect floating-point accuracy.
Description

This option disables optimizations that affect floating-point accuracy.
If the default setting is used, the compiler may apply optimizations that reduce floating-point accuracy.
You can use this option to improve floating-point accuracy, but at the cost of disabling some optimizations.
The recommended method to control the semantics of floating-point calculations is to use option -fp-model (Linux) or /fp (Windows).
Instead of using -IPF-fltacc (Linux) or /QIPF-fltacc (Windows), you can use -fp-model precise (Linux) or /fp:precise (Windows).
Instead of using -no-IPF-fltacc (Linux) or /QIPF-fltacc- (Windows), you can use -fp-model fast (Linux) or /fp:fast (Windows).

Alternate Options

None

See Also

fp-model, fp compiler option
IPF-fma, QIPF-fma
See fma, Qfma.
IPF-fp-relaxed, QIPF-fp-relaxed
See fp-relaxed, Qfp-relaxed.
ipo, Qipo
Enables interprocedural optimization between files.

IDE Equivalent

Windows: Optimization > Interprocedural Optimization
Linux: Optimization > Enable Whole Program Optimization
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: 
Windows: 

Arguments

$n$

Is an optional integer that specifies the number of object files the compiler should create. The integer must be greater than or equal to 0.
Multifile interprocedural optimization is not enabled.

Description

This option enables interprocedural optimization between files. This is also called multifile interprocedural optimization (multifile IPO) or Whole Program Optimization (WPO).

When you specify this option, the compiler performs inline function expansion for calls to functions defined in separate files.

You cannot specify the names for the files that are created.

If $n$ is 0, the compiler decides whether to create one or more object files based on an estimate of the size of the application. It generates one object file for small applications, and two or more object files for large applications.

If $n$ is greater than 0, the compiler generates $n$ object files, unless $n$ exceeds the number of source files ($m$), in which case the compiler generates only $m$ object files.

If you do not specify $n$, the default is 0.

Alternate Options

None

ipo-c, Qipo-c

Tells the compiler to optimize across multiple files and generate a single object file.

IDE Equivalent

None

Architectures
Syntax

Linux and Mac OS X: -ipo-c
Windows:   /Qipo-c

Arguments

None

Default

OFF

The compiler does not generate a multifile object file.

Description

This option tells the compiler to optimize across multiple files and generate a single object file (named ipo_out.o on Linux and Mac OS X systems; ipo_out.obj on Windows systems).

It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized object file that can be used in further link steps.

Alternate Options

None

See Also
ipo, Qipo compiler option

ipo-jobs, Qipo-jobs

Specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO).

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -ipo-jobs \( n \)

Windows: /Qipo-jobs: \( n \)

Arguments

\( n \)

Is the number of commands (jobs) to run simultaneously. The number must be greater than or equal to 1.

Default

-ipo-jobs1
or/Qipo-jobs:1

One command (job) is executed in an interprocedural
optimization
parallel build.

Description

This option specifies the number of commands (jobs) to be executed simultaneously during the link phase of Interprocedural Optimization (IPO). It should only be used if the link-time compilation is generating more than one object. In this case, each object is generated by a separate compilation, which can be done in parallel.

This option can be affected by the following compiler options:

- `-ipo` (Linux and Mac OS X) or `/Qipo` (Windows) when applications are large enough that the compiler decides to generate multiple object files.
- `-ipo` (Linux and Mac OS X) or `/Qipo-n` (Windows) when \( n \) is greater than 1.
- `-ipo-separate` (Linux) or `/Qipo-separate` (Windows)

⚠️ Caution

Be careful when using this option. On a multi-processor system with lots of memory, it can speed application build time. However, if \( n \) is greater than the number of processors, or if there is not enough memory to avoid thrashing, this option can increase application build time.

Alternate Options

None

See Also

- ipo, Qipo compiler option
- ipo-separate, Qipo-separate compiler option
- ipo-S, Qipo-S
Tells the compiler to optimize across multiple files and generate a single assembly file.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-ipo-S`

Windows: `/Qipo-S`

**Arguments**

None

**Default**

OFF

The compiler does not generate a multifile assembly file.

**Description**

This option tells the compiler to optimize across multiple files and generate a single assembly file (named `ipo_out.s` on Linux and Mac OS X systems; `ipo_out.asm` on Windows systems).
It performs the same optimizations as -ipo (Linux and Mac OS X) or /Qipo (Windows), but compilation stops before the final link stage, leaving an optimized assembly file that can be used in further link steps.

Alternate Options
None

See Also
ipo, Qipo compiler option

ipo-separate, Qipo-separate
Tells the compiler to generate one object file for every source file.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux: -ipo-separate
Mac OS X: None
Windows: /Qipo-separate

Arguments
None

Default
OFF

The compiler decides
whether to create one or more object files.

Description

This option tells the compiler to generate one object file for every source file. It overrides any -ipo (Linux) or /Qipo (Windows) specification.

Alternate Options

None

See Also

ipo, Qipo compiler option

ivdep-parallel, Qivdep-parallel

Tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP pragma.

IDE Equivalent

Windows: None
Linux: Optimization > IVDEP Directive Memory Dependency
Mac OS X: None

Architectures

IA-64 architecture

Syntax
Linux: -ivdep-parallel
Mac OS X: None
Windows: /Qivdep-parallel

Arguments
None

Default
OFF

There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.

Description
This option tells the compiler that there is no loop-carried memory dependency in the loop following an IVDEP. There may be loop-carried memory dependency in a loop that follows an IVDEP pragma.

Alternate Options
None

fkeep-static-consts, Qkeep-static-consts
Tells the compiler to preserve allocation of variables that are not referenced in the source.

IDE Equivalent

784
None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
- fkeep-static-consts  
  -fno-keep-static-consts

Windows:  
/Qkeep-static-consts  
/Qkeep-static-consts-

Arguments

None

Default

-fno-keep-static-consts or /Qkeep-static-consts  
If a variable is never referenced in a routine, the variable is discarded unless optimizations are disabled by option –00 (Linux and Mac OS X) or /Od (Windows).

Description
This option tells the compiler to preserve allocation of variables that are not referenced in the source. The negated form can be useful when optimizations are enabled to reduce the memory usage of static data.

Alternate Options

None

Qlocation

Specifies the directory for supporting tools.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Qlocation,string,dir
Windows: /Qlocation,string,dir

Arguments

string

Is the name of the tool.

dir

Is the directory (path) where the tool
Default

OFF

The compiler looks for tools in a default area.

Description

This option specifies the directory for supporting tools.

_string_ can be any of the following:

- **c** - Indicates the Intel C++ compiler.
- **cpp** (or **fpp**) - Indicates the Intel C++ preprocessor.
- **cxxinc** - Indicates C++ header files.
- **cinc** - Indicates C header files.
- **asm** - Indicates the assembler.
- **link** - Indicates the linker.
- **prof** - Indicates the profiler.
- **On Windows systems, the following is also available:**
  - **masm** - Indicates the Microsoft assembler.
- **On Linux and Mac OS X systems, the following are also available:**
  - **as** - Indicates the assembler.
  - **gas** - Indicates the GNU assembler.
  - **ld** - Indicates the loader.
  - **gld** - Indicates the GNU loader.
  - **lib** - Indicates an additional library.
o crt - Indicates the crt%.o files linked into executables to contain the place to start execution.

Alternate Options

None

See Also

Qoption compiler option

Qlong-double

Changes the default size of the long double data type.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Qlong-double

Arguments

None

Default

OFF

The default size of the long
**Description**

This option changes the default size of the long double data type to 80 bits. However, the alignment requirement of the data type is 16 bytes, and its size must be a multiple of its alignment, so the size of a long double on Windows is also 16 bytes. Only the lower 10 bytes (80 bits) of the 16 byte space will have valid data stored in it.

Note that the Microsoft compiler and Microsoft-provided library routines (such as `printf`) do not provide support for 80-bit floating-point values. As a result, this option should only be used when referencing symbols within parts of your application built with this option or symbols in libraries that were built with this option.

**Alternate Options**

None

**M, QM**

Tells the compiler to generate makefile dependency lines for each source file.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X: –M
Windows: /QM

Arguments
None

Default
OFF  The compiler does not generate makefile dependency lines for each source file.

Description
This option tells the compiler to generate makefile dependency lines for each source file, based on the #include lines found in the source file.

Alternate Options
None

map-opts, Qmap-opts
Maps one or more compiler options to their equivalent on a different operating system.

IDE Equivalent
None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -map-opts
Mac OS X: None
Windows: /Qmap-opts

Arguments

None

Default

OFF No platform mappings are performed.

Description

This option maps one or more compiler options to their equivalent on a different operating system. The result is output to stdout.

On Windows systems, the options you provide are presumed to be Windows options, so the options that are output to stdout will be Linux equivalents.

On Linux systems, the options you provide are presumed to be Linux options, so the options that are output to stdout will be Windows equivalents.

The tool can be invoked from the compiler command line or it can be used directly.

No compilation is performed when the option mapping tool is used.

This option is useful if you have both compilers and want to convert scripts or makefiles.
Note

Compiler options are mapped to their equivalent on the architecture you are using. For example, if you are using a processor with IA-32 architecture, you will only see equivalent options that are available on processors with IA-32 architecture.

Alternate Options

None

Example

The following command line invokes the option mapping tool, which maps the Linux options to Windows-based options, and then outputs the results to stdout:

```
icc -map-opts -xP -O2
```

The following command line invokes the option mapping tool, which maps the Windows options to Linux-based options, and then outputs the results to stdout:

```
icl /Qmap-opts /QxP /O2
```

See Also

Building Applications: Using the Option Mapping Tool

MD, QMD

Preprocess and compile, generating output file containing dependency information ending with extension .d.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -MD
Windows: /QMD

Arguments

None

Default

OFF

The compiler does not generate dependency information.

Description

Preprocess and compile, generating output file containing dependency information ending with extension .d.

Alternate Options

None

MF, QMF

Tells the compiler to generate makefile dependency information in a file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:  \(-MFfile\)
Windows: \(/QMFfile\)

Arguments

`file`  
Is the name of the file where the makefile dependency information should be placed.

Default

OFF  
The compiler does not generate makefile dependency information in files.

Description

This option tells the compiler to generate makefile dependency information in a file. To use this option, you must also specify `/QM` or `/QMM`.

Alternate Options

None
See Also

QH compiler option
QHM compiler option

MG, QMG

Tells the compiler to generate makefile dependency lines for each source file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-MG`
Windows: `/QMG`

Arguments

None

Default

OFF

The compiler does not generate makefile dependency information in files.
This option tells the compiler to generate makefile dependency lines for each source file. It is similar to /QM, but it treats missing header files as generated files.

Alternate Options

None

See Also

/QM compiler option

MM, QMM

Tells the compiler to generate makefile dependency lines for each source file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -MM
Windows: /QMM

Arguments

None

Default

OFF

The compiler does not generate
makefile
dependency
information
in files.

Description

This option tells the compiler to generate makefile dependency lines for each source file. It is similar to `/QM`, but it does not include system header files.

Alternate Options

None

See Also

QM compiler option

MMD, QMMD

Tells the compiler to generate an output file containing dependency information.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-MMD`
Windows: `/QMMD`

Arguments

None

Default
The compiler does not generate an output file containing dependency information.

**Description**

This option tells the compiler to preprocess and compile a file, then generate an output file (with extension .d) containing dependency information. It is similar to /QMD, but it does not include system header files.

**Alternate Options**

None

**Qms**

Tells the compiler to emulate Microsoft compatibility bugs.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** None

**Windows:** /Qmsn

**Arguments**
Possible values are:

0  Instructs the compiler to disable some Microsoft compatibility bugs. It tells the compiler to emulate the fewest number of Microsoft compatibility bugs.

1  Instructs the compiler to enable most Microsoft compatibility bugs. It tells the compiler to emulate more Microsoft compatibility bugs than /Qms0.
Instructs the compiler to generate code that is Microsoft compatible. The compiler emulates the largest number of Microsoft compatibility bugs.

Default

/Qms1

The compiler emulates most Microsoft compatibility bugs.

Description

This option tells the compiler to emulate Microsoft compatibility bugs.

Caution
When using `/Qms0`, your program may not compile if it depends on Microsoft headers with compatibility bugs that are disabled with this option. Use `/Qms1` if your compilation fails.

Alternate Options

None

Qmspp

Enables Microsoft Visual C++* 6.0 Processor Pack binary compatibility. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32 architecture

Syntax

Linux and Mac OS X: None
Windows: `/Qmspp`

Arguments

None

Default

ON

The compiler is compatible with the Microsoft
Description

This option enables Microsoft Visual C++ 6.0 Processor Pack binary compatibility among modules using the SIMD data types.

The /Qmspp- option is useful when you need to maintain compatibility with binaries that were built with earlier versions of the Intel® C++ Compiler.

Alternate Options

None

MT, QMT

Changes the default target rule for dependency generation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -MTtarget
Windows: /QMTtarget

Arguments

(target) Is the
Default

OFF

The default target rule applies to dependency generation.

Description

This option changes the default target rule for dependency generation.

Alternate Options

None

multibyte-chars, Qmultibyte-chars

Determines whether multi-byte characters are supported.

IDE Equivalent

Windows: None
Linux: Language > Support Multibyte Characters in Source
Mac OS X: Language > Support Multibyte Characters in Source

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -multibyte-chars
-no-multibyte-chars

Windows:
/Qmultibyte-chars
/Qmultibyte-chars-

Arguments
None

Default
-multibyte-chars or /Qmultibyte-chars

Multi-byte characters are supported.

Description
This option determines whether multi-byte characters are supported.

Alternate Options
None

no-bss-init, Qnobss-init

Tells the compiler to place in the DATA section any variables explicitly initialized with zeros.

IDE Equivalent
Windows: None
Linux: Data > Disable Placement of Zero-initialized Variables in .bss - use .data
Mac OS X: Data > Allocate Zero-initialized Variables to .data

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-no-bss-init`
Windows: `/Qnобss-init`

Arguments

None

Default

OFF

Variables explicitly initialized with zeros are placed in the BSS section.

Description

This option tells the compiler to place in the DATA section any variables explicitly initialized with zeros.

Alternate Options

Linux and Mac OS X: `-nobss-init` (this is a deprecated option)
Windows: None

Qnopic

Disables generation of position-independent code.

IDE Equivalent

None
Architectures

IA-64 architecture

Syntax

Linux and Mac OS X: None
Windows: /Qnopic

Arguments

None

Default

OFF

The compiler can generate position-independent code.

Description

This option disables generation of position-independent code.

Alternate Options

None

openmp, Qopenmp

Enables the parallelizer to generate multi-threaded code based on the OpenMP* directives.

IDE Equivalent

Windows: Language > OpenMP* Support
Linux: **Language > Process OpenMP Directives**
Mac OS X: **Language > Process OpenMP Directives**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-openmp`
Windows: `/Qopenmp`

**Arguments**

None

**Default**

OFF

No

OpenMP multi-threaded code is generated by the compiler.

**Description**

This option enables the parallelizer to generate multi-threaded code based on the OpenMP* directives. The code can be executed in parallel on both uniprocessor and multiprocessor systems.

This option works with any optimization level. Specifying no optimization (`-O0` on Linux or `/Od` on Windows) helps to debug OpenMP applications.

**Note**
On Mac OS X systems, when you enable OpenMP*, you must also set the DYLD_LIBRARY_PATH environment variable within Xcode or an error will be displayed.

Alternate Options
None

See Also
openmp-stubs, Qopenmp-stubs compiler option
openmp-lib, Qopenmp-lib

Lets you specify an OpenMP* run-time library to use for linking.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux: -openmp-lib type
Mac OS X: None
Windows: /Qopenmp-lib:type

Arguments

type

Specifies the type of library to use; it implies compatibility levels. Possible values are:

legacy

Tells the
compat

Tells the compiler to use the compatibility OpenMP* run-time library (libiomp).
This setting provides compatibility with object files created using other compilers.
This is a deprecated option.

compat

Tells the compiler to use the legacy OpenMP* run-time library (libguide).
This setting does not provide compatibility with object files created using other compilers.
This is a deprecated option.
files created using Microsoft* and GNU* compilers.

Default

-openmp-lib compat or /Qopenmp-lib:compat

The compiler uses the compatibility OpenMP* run-time library (libiomp).

Description

This option lets you specify an OpenMP* run-time library to use for linking. The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers. The compatibility OpenMP run-time library is compatible with object files created using the Microsoft* OpenMP run-time library (vcomp) and GNU OpenMP run-time library (libgomp).

To use the compatibility OpenMP run-time library, compile and link your application using the -openmp-lib compat (Linux) or /Qopenmp-lib:compat (Windows) option. To use this option, you must also specify one of the following compiler options:

- **Linux OS**: -openmp, -openmp-profile, or -openmp-stubs
- **Windows OS**: /Qopenmp, /Qopenmp-profile, or /Qopenmp-stubs
On Windows* systems, the compatibility OpenMP* run-time library lets you combine OpenMP* object files compiled with the Microsoft* C/C++ compiler with OpenMP* object files compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

On Linux* systems, the compatibility Intel OpenMP* run-time library lets you combine OpenMP* object files compiled with the GNU* gcc or gfortran compilers with similar OpenMP* object files compiled with the Intel C/C++ or Fortran compilers. The linking phase results in a single, coherent copy of the run-time library.

**Note**

The compatibility OpenMP run-time library is not compatible with object files created using versions of the Intel compiler earlier than 10.0.

**Alternate Options**

None

**See Also**

openmp, Qopenmp compiler option
openmp-stubs, Qopenmp-stubs compiler option
openmp-profile, Qopenmp-profile compiler option

**openmp-link, Qopenmp-link**

Controls whether the compiler links to static or dynamic OpenMP run-time libraries.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X: `-openmp-link library`
Windows: `/Qopenmp-link:library`

Arguments

*library*  
Specifies the OpenMP library to use. Possible values are:

- `static` Tells the compiler to link to static OpenMP run-time libraries.
- `dynamic` Tells the compiler to link to dynamic OpenMP run-time libraries.

Default

`-openmp-link dynamic` or `/Qopenmp-link:dynmac`  
The compiler links to dynamic OpenMP
run-time libraries. However, if option **static** is specified, the compiler links to static OpenMP run-time libraries.

**Description**

This option controls whether the compiler links to static or dynamic OpenMP run-time libraries.

To link to the static OpenMP run-time library (RTL) and create a purely static executable, you must specify `-openmp-link static` (Linux and Mac OS X) or `/Qopenmp-link:static` (Windows). However, we strongly recommend you use the default setting, `-openmp-link dynamic` (Linux and Mac OS X) or `/Qopenmp-link:dynamiс` (Windows).

**Note**

Compiler options `-static-intel` and `-shared-intel` (Linux and Mac OS X) have no effect on which OpenMP run-time library is linked.

**Alternate Options**

None
openmp-profile, Qopenmp-profile

Enables analysis of OpenMP* applications if Intel® Thread Profiler is installed.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -openmp-profile
Mac OS X: None
Windows: /Qopenmp-profile

Arguments

None

Default

OFF OpenMP applications are not analyzed.

Description

This option enables analysis of OpenMP* applications. To use this option, you must have previously installed Intel® Thread Profiler, which is one of the Intel® Threading Analysis Tools.

This option can adversely affect performance because of the additional profiling and error checking invoked to enable compatibility with the threading tools. Do not use this option unless you plan to use the Intel® Thread Profiler.
For more information about Intel® Thread Profiler (including an evaluation copy) open the page associated with threading tools at Intel® Software Development Products.

Alternate Options
None

**openmp-report, Qopenmp-report**
Controls the OpenMP® parallelizer's level of diagnostic messages.

**IDE Equivalent**
Windows: None
Linux: **Compilation Diagnostics > OpenMP Report**
Mac OS X: **Diagnostics > OpenMP Report**

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**
Linux and Mac OS X: `-openmp-report [n]`
Windows: `/Qopenmp-report [n]`

**Arguments**

\(n\) Is the level of diagnostic messages to display. Possible values are:

\(\emptyset\) No diagnostic messages are
Diagnostic messages are displayed indicating loops, regions, and sections successfully parallelized.

The same diagnostic messages are displayed as specified by openmp_report1 plus diagnostic messages indicating successful handling of MASTER constructs, SINGLE constructs, CRITICAL constructs, ORDERED constructs, ATOMIC
Default

-oopenmp-report1
or/Qopenmp-report1

This is the default if you do not specify n. The compiler displays diagnostic messages indicating loops, regions, and sections successfully parallelized. If you do not specify the option on the command line, the default is to display no messages.
Description

This option controls the OpenMP* parallelizer's level of diagnostic messages. To use this option, you must also specify `-openmp` (Linux and Mac OS X) or `/Qopenmp` (Windows).

If this option is specified on the command line, the report is sent to `stdout`.

Alternate Options

None

See Also

openmp, Qopenmp compiler option

Optimizing Applications:
Using Parallelism
OpenMP* Report

openmp-stubs, Qopenmp-stubs

Enables compilation of OpenMP programs in sequential mode.

IDE Equivalent

Windows: Language > Process OpenMP Directives
Linux: Language > Process OpenMP Directives
Mac OS X: Language > Process OpenMP Directives

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -openmp-stubs
Windows: /Qopenmp-stubs

Arguments
None

Default

OFF

The library of OpenMP function stubs is not linked.

Description

This option enables compilation of OpenMP programs in sequential mode. The OpenMP directives are ignored and a stub OpenMP library is linked.

Alternate Options

None

See Also

openmp, Qopenmp compiler option

openmp-task, Qopenmp-task

Lets you choose an OpenMP* tasking model.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -openmp-task model
Windows: 

/Qopenmp-task: model

Arguments

model

Is an OpenMP tasking model. Possible values are:

intel Tells the compiler to accept Intel® taskqueuing pragmas (#pragma intel_omp_taskq and #pragma intel_omp_task). When this value is specified, OpenMP 3.0 tasking pragmas are ignored; if they are specified, warnings are issued.

omp Tells the compiler to accept OpenMP 3.0 tasking pragmas.
When this value is specified, Intel taskqueuing pragmas are ignored; if they are specified, warnings are issued.

**Default**

- `-openmp-task omp` or `/Qopenmp-task:omp`

The compiler accepts OpenMP 3.0 tasking pragmas.

**Description**

The option lets you choose an OpenMP tasking model. To use this option, you must also specify option `-openmp` (Linux and Mac OS X) or `/Qopenmp` (Windows).

**Alternate Options**

None

- `openmp-threadprivate`, `Qopenmp-threadprivate`
Lets you specify an OpenMP* threadprivate implementation.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux:**  `-openmp-threadprivate type`

**Mac OS X:** None

**Windows:**  `/Qopenmp-threadprivate:type`

**Arguments**

`type`  
Specifies the type of threadprivate implementation. Possible values are:

- **legacy** Tells the compiler to use the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler. This setting does not provide compatibility with
the implementation
used by other
compilers.

**compat** Tells the compiler
to use the
compatibility
OpenMP*
threadprivate
implementation
based on applying
the
__declspec(thread)
attribute to each
threadprivate
variable. The
limitations of the
attribute on a given
platform also apply
to the
threadprivate
implementation.
This setting
provides
compatibility with
the implementation
provided by the
Microsoft* and
GNU* compilers.

**Default**
-openmp-threadprivate legacy
or/Qopenmp-threadprivate:legacy

The compiler uses the legacy OpenMP* threadprivate implementation used in the previous releases of the Intel® compiler.

Description

This option lets you specify an OpenMP* threadprivate implementation. The legacy OpenMP run-time library is not compatible with object files created using OpenMP run-time libraries supported in other compilers. To use this option, you must also specify one of the following compiler options:

- **Linux OS**: -openmp, -openmp-profile, or -openmp-stubs
- **Windows OS**: /Qopenmp, /Qopenmp-profile, or /Qopenmp-stubs

The value specified for this option is independent of the value used for option -openmp-lib (Linux) or /Qopenmp-lib (Windows).

Alternate Options

None

**opt-block-factor, Qopt-block-factor**

Lets you specify a loop blocking factor.

IDE Equivalent

Windows: **Diagnostics > Optimization Diagnostic File**

Diagnostics > Emit Optimization Diagnostics to File
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: \(-\text{opt-block-factor}=n\)
Windows: \(/\text{Qopt-block-factor}:n\)

Arguments

\(n\)

Is the blocking factor. It must be an integer. The compiler may ignore the blocking factor if the value is 0 or 1.

Default
OFF

The
Intel® C++ Compiler User and Reference Guides

This option lets you specify a loop blocking factor.

**Alternate Options**

None

**opt-class-analysis, Qopt-class-analysis**

Determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-opt-class-analysis`
`-no-opt-class-analysis`

Windows: `/Qopt-class-analysis`
`/Qopt-class-analysis-`

**Arguments**

None
Default

-no-opt-class-analysis          C++ class hierarchy information is not used to analyze and resolve C++ virtual function calls at compile time.
or/Qopt-class-analysis-

Description

This option determines whether C++ class hierarchy information is used to analyze and resolve C++ virtual function calls at compile time. The option is turned on by default with the -ipo compiler option, enabling improved C++ optimization. If a C++ application contains non-standard C++ constructs, such as pointer down-casting, it may result in different behaviors.

Alternate Options

None

opt-jump-tables, Qopt-jump-tables

Enables or disables generation of jump tables for switch statements.

IDE Equivalent

None

Architectures
Syntax

Linux and Mac OS X: `-opt-jump-tables=keyword`
   `-no-opt-jump-tables`

Windows: `/Qopt-jump-tables:keyword`
   `/Qopt-jump-tables-

Arguments

`keyword` Is the instruction for generating jump tables. Possible values are:

- `never` Tells the compiler to never generate jump tables. All switch statements are implemented as chains of if-then-elems. This is the same as specifying `-no-opt-jump-tables` (Linux and...
Mac OS) or
/Qopt-
jump-
tables-
(Windows).

default The
compiler
uses default
heuristics to
determine
when to
generate
jump tables.

large Tells the
compiler to
generate
jump tables
up to a
certain pre-
defined size
(64K
entries).

n Must be an
integer. Tells
the compiler
to generate
jump tables
up to \(n\)
entries in
Default

-`opt-jump-tables=default`
  or `-Qopt-jump-tables:default`

The compiler uses default heuristics to determine when to generate jump tables for switch statements.

Description

This option enables or disables generation of jump tables for switch statements. When the option is enabled, it may improve performance for programs with large switch statements.

Alternate Options

None

**opt-loadpair, -Qopt-loadpair**

Enables or disables loadpair optimization.

IDE Equivalent

None

Architectures
IA-64 architecture

Syntax

Linux:  
- opt-loadpair  
  -no-opt-loadpair

Mac OS X:  None

Windows:  /Qopt-loadpair  
  /Qopt-loadpair-

Arguments

None

Default

-no-opt-loadpair  
or /Qopt-loadpair-

Loadpair optimization is disabled unless option O3 is specified.

Description

This option enables or disables loadpair optimization.

When -O3 is specified on IA-64 architecture, loadpair optimization is enabled by default. To disable loadpair generation, specify -no-opt-loadpair (Linux) or /Qopt-loadpair- (Windows).

Alternate Options

None

opt-mem-bandwidth, Qopt-mem-bandwidth
Enables performance tuning and heuristics that control memory bandwidth use among processors.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux: \texttt{-opt-mem-bandwidth\,n}

Mac OS X: None

Windows: \texttt{/Qopt-mem-bandwidth\,n}

Arguments

\begin{itemize}
  \item \texttt{n} \hspace{1cm} Is the level of optimizing for memory bandwidth usage. Possible values are:
  \item \texttt{0} Enables a set of performance tuning and heuristics in compiler optimizations that is optimal for
serial code.

1 Enables a set of performance tuning and heuristics in compiler optimizations for multithreaded code generated by the compiler.

2 Enables a set of performance tuning and heuristics in compiler optimizations for parallel code such as Windows Threads, pthreads, and MPI code, besides multithreaded
Default

- `opt-mem-bandwidth0`
  or `/Qopt-mem-bandwidth0`

For serial (non-parallel) compilation, a set of performance tuning and heuristics in compiler optimizations is enabled that is optimal for serial code.

If you specify compiler option –
`parallel` (Linux) or `/Qparallel` (Windows), –
`openmp` (Linux) or `/Qopenmp` (Windows),
or Cluster
OpenMP
option –
cluster-
openmp
(Linux), a set of
performance
tuning and
heuristics in
compiler
optimizations
for
multithreaded
code
generated by
the compiler
is enabled.

Description
This option enables performance tuning and heuristics that control memory
bandwidth use among processors. It allows the compiler to be less aggressive
with optimizations that might consume more bandwidth, so that the bandwidth
can be well-shared among multiple processors for a parallel program.
For values of $n$ greater than 0, the option tells the compiler to enable a set of
performance tuning and heuristics in compiler optimizations such as prefetching,
privatization, aggressive code motion, and so forth, for reducing memory
bandwidth pressure and balancing memory bandwidth traffic among threads.
This option can improve performance for threaded or parallel applications on
multiprocessors or multicore processors, especially when the applications are
bounded by memory bandwidth.
Alternate Options

None

See Also

parallel, Qparallel compiler option
openmp, Qopenmp compiler option

**opt-mod-versioning, Qopt-mod-versioning**

Enables or disables versioning of modulo operations for certain types of operands.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:

- -opt-mod-versioning
- -no-opt-mod-versioning

Mac OS X:

None

Windows:

/Qopt-mod-versioning
/Qopt-mod-versioning-

Arguments

None

Default

- -no-opt-mod-versioning
  
  or/Qopt-mod-versioning-

  Versioning
  of modulo
  operations
is disabled.

Description

This option enables or disables versioning of modulo operations for certain types of operands. It is used for optimization tuning.
Versioning of modulo operations may improve performance for $x \mod y$ when modulus $y$ is a power of 2.

Alternate Options

None

opt-multi-version-aggressive, Qopt-multi-version-aggressive

Tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:  
- opt-multi-version-aggressive
- no-opt-multi-version-aggressive

Windows:  
/Qopt-multi-version-aggressive
/Qopt-multi-version-aggressive-

Arguments

None

Default
The compiler uses default heuristics when checking for pointer aliasing and scalar replacement.

Description

This option tells the compiler to use aggressive multi-versioning to check for pointer aliasing and scalar replacement. This option may improve performance.

Alternate Options

None

opt-prefetch, Qopt-prefetch

Enables or disables prefetch insertion optimization.

IDE Equivalent

Windows: None
Linux: Optimization > Enable Prefetch Insertion
Mac OS X: Optimization > Enable Prefetch Insertion

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -opt-prefetch [=n]
-no-opt-prefetch
Arguments

$n$

Is the level of detail in the report. Possible values are:

0  Disables software prefetching. This is the same as specifying –no-opt-prefetch (Linux and Mac OS X) or /Qopt-prefetch- (Windows).

1  Enables to different

4  levels of software prefetching.

If you do not specify a
value for \( n \), the default is 2 on IA-32 and Intel® 64 architecture; the default is 3 on IA-64 architecture. Use lower values to reduce the amount of prefetching.

**Default**

**IA-64 architecture:** `-opt-prefetch`

or `-Qopt-prefetch`

**IA-32 architecture and Intel® 64 architecture:**

`-no-opt-prefetch`

or `-Qopt-prefetch-`

On IA-64 architecture, prefetch insertion optimization is enabled.

On IA-32 architecture and Intel® 64 architecture, ```prefetch```
insertion optimization is disabled.

**Description**

This option enables or disables prefetch insertion optimization. The goal of prefetching is to reduce cache misses by providing hints to the processor about when data should be loaded into the cache.

On IA-64 architecture, this option is enabled by default if you specify option `O1` or higher. To disable prefetching at these optimization levels, specify `-no-opt-prefetch` (Linux and Mac OS X) or `/Qopt-prefetch-` (Windows).

On IA-32 architecture and Intel® 64 architecture, this option enables prefetching when higher optimization levels are specified.

**Alternate Options**

Linux and Mac OS X: `-prefetch` (this is a [deprecated](https://www.example.com) option)

Windows: `/Qprefetch` (this is a [deprecated](https://www.example.com) option)

**opt-prefetch-initial-values, Qopt-prefetch-initial-values**

Enables or disables prefetches that are issued before a loop is entered.

**IDE Equivalent**

None

**Architectures**

IA-64 architecture

**Syntax**

Linux:  
- `-opt-prefetch-initial-values`
- `-no-opt-prefetch-initial-values`

Mac OS X: None
Windows: /Qopt-prefetch-initial-values
/Qopt-prefetch-initial-values-

Arguments
None

Default
-opt-prefetch-initial-values
or /Qopt-prefetch-initial-values

Description
This option enables or disables prefetches that are issued before a loop is entered. These prefetches target the initial iterations of the loop.
When -O1 or higher is specified on IA-64 architecture, prefetches are issued before a loop is entered. To disable these prefetches, specify -no-opt-prefetch-initial-values (Linux) or /Qopt-prefetch-initial-values- (Windows).

Alternate Options
None

opt-prefetch-issue-excl-hint, Qopt-prefetch-issue-excl-hint
Determines whether the compiler issues prefetches for stores with exclusive hint.

IDE Equivalent
None

Architectures
IA-64 architecture

Syntax

Linux:          -opt-prefetch-issue-excl-hint
                -no-opt-prefetch-issue-excl-hint

Mac OS X:       None

Windows:        /Qopt-prefetch-issue-excl-hint
                /Qopt-prefetch-issue-excl-hint-

Arguments

None

Default

-no-opt-prefetch-issue-excl-hint
or /Qopt-prefetch-issue-excl-hint-

The compiler does not issue prefetches for stores with exclusive hint.

Description

This option determines whether the compiler issues prefetches for stores with exclusive hint. If option -opt-prefetch-issue-excl-hint (Linux) or /Qopt-prefetch-issue-excl-hint (Windows) is specified, the prefetches will be issued if the compiler determines it is beneficial to do so.
When prefetches are issued for stores with exclusive-hint, the cache-line is in "exclusive-mode". This saves on cache-coherence traffic when other processors try to access the same cache-line. This feature can improve performance tuning.

Alternate Options

None

**opt-prefetch-next-iteration, /Qopt-prefetch-next-iteration**

Enables or disables prefetches for a memory access in the next iteration of a loop.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:  
- **-opt-prefetch-next-iteration**
- **-no-opt-prefetch-next-iteration**

Mac OS X:  
None

Windows:  
/\Qopt-prefetch-next-iteration

Arguments

None

Default

- **-opt-prefetch-next-iteration**  
  or /\Qopt-prefetch-next-iteration  
  Prefetches are issued for a
Description

This option enables or disables prefetches for a memory access in the next iteration of a loop. It is typically used in a pointer-chasing loop. When -O1 or higher is specified on IA-64 architecture, prefetches are issued for a memory access in the next iteration of a loop. To disable these prefetches, specify -no-opt-prefetch-next-iteration (Linux) or /Qopt-prefetch-next-iteration- (Windows).

Alternate Options

None

opt-ra-region-strategy, Qopt-ra-region-strategy

Selects the method that the register allocator uses to partition each routine into regions.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -opt-ra-region-strategy[=keyword]

Windows: /Qopt-ra-region-strategy[:keyword]
Arguments

*keyword*

Is the method used for partitioning. Possible values are:

- **routine** Creates a single region for each routine.
- **block** Partitions each routine into one region per basic block.
- **trace** Partitions each routine into one region per trace.
- **region** Partitions each routine into one region per loop.
- **default** The compiler determines which
The method is used for partitioning.

**Default**

- `-opt-ra-region-strategy=default`
  - `or/Qopt-ra-region-strategy:default`

The compiler determines which method is used for partitioning. This is also the default if keyword is not specified.

**Description**

This option selects the method that the register allocator uses to partition each routine into regions.

When setting default is in effect, the compiler attempts to optimize the tradeoff between compile-time performance and generated code performance.

This option is only relevant when optimizations are enabled (O1 or higher).

**Alternate Options**

None

**See Also**

o compiler option
**opt-report, Qopt-report**

Tells the compiler to generate an optimization report to stderr.

**IDE Equivalent**

Windows: **Diagnostics > Optimization Diagnostic Level**

Linux: **Compilation Diagnostics > Optimization Diagnostic Level**

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-opt-report[n]`

Windows: `/Qopt-report[:n]`

**Arguments**

- **n**

  Is the level of detail in the report.

  Possible values are:

  0 Tells the compiler to generate no optimization report.

  1 Tells the compiler to
generate a report with the minimum level of detail.

2 Tells the compiler to generate a report with the medium level of detail.

3 Tells the compiler to generate a report with the maximum level of detail.

Default

-`opt-report 2` or `/Qopt-report:2`

If you do not specify `n`, the compiler generates a report with
medium detail. If you do not specify the option on the command line, the compiler does not generate an optimization report.

Description

This option tells the compiler to generate an optimization report to stderr.

Alternate Options

None

See Also

opt-report-file, Qopt-report-file compiler option

Optimizing Applications: Optimizer Report Generation

opt-report-file, Qopt-report-file

Specifies the name for an optimization report.

IDE Equivalent

Windows: Diagnostics > Optimization Diagnostic File
Diagnostics > Emit Optimization Diagnostics to File
Linux: **Compilation Diagnostics > Emit Optimization Diagnostics to File**  
**Compilation Diagnostics > Optimization Diagnostics File**  
Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-opt-report-file=file`

Windows: `/Qopt-report-file:file`

**Arguments**

- `file` is the name for the optimization report.

**Default**

OFF: No optimization report is generated.

**Description**

This option specifies the name for an optimization report. If you use this option, you do not have to specify `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

**Alternate Options**

None
opt-report-help, Qopt-report-help

Displays the optimizer phases available for report generation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  -opt-report-help
Windows:  /Qopt-report-help

Arguments

None

Default

OFF  No optimization reports are generated.

Description

This option displays the optimizer phases available for report generation using -opt-report-phase (Linux and Mac OS X) or /Qopt-report-phase (Windows). No compilation is performed.
Alternate Options

None

See Also

opt-report, Qopt-report compiler option
opt-report-phase, Qopt-report-phase compiler option

**opt-report-phase, Qopt-report-phase**

Specifies an optimizer phase to use when optimization reports are generated.

**IDE Equivalent**

Windows: **Diagnostics > Optimization Diagnostic Phase**
Linux: **Compilation Diagnostics > Optimization Diagnostic Phase**
Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-opt-report-phase=phase`
Windows: `Qopt-report-phase:phase`

**Arguments**

`phase` Is the phase to generate reports for. Some of the possible values are:

ipo The Interprocedural Optimizer phase
hlo  The High Level Optimizer phase

hpo  The High Performance Optimizer phase

ilo  The Intermediate Language Scalar Optimizer phase

ecg  The Code Generator phase
     (Windows and Linux systems using IA-64 architecture only)

ecg_swp The software pipelining component of the Code Generator phase
         (Windows and Linux systems)
pgo The Profile Guided Optimization phase

all All optimizer phases

**Default**

OFF No optimization reports are generated.

**Description**

This option specifies an optimizer phase to use when optimization reports are generated. To use this option, you must also specify `-opt-report` (Linux and Mac OS X) or `/Qopt-report` (Windows).

This option can be used multiple times on the same command line to generate reports for multiple optimizer phases.

When one of the logical names for optimizer phases is specified for phase, all reports from that optimizer phase are generated.

To find all phase possibilities, use option `-opt-report-help` (Linux and Mac OS X) or `/Qopt-report-help` (Windows).

**Alternate Options**

None
opt-report-routine, Qopt-report-routine

Tells the compiler to generate reports on the routines containing specified text.

IDE Equivalent

Windows: Diagnostics > Optimization Diagnostic Routine
Linux: Compilation Diagnostics > Optimization Diagnostic Routine
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -opt-report-routine=string
Windows: /Qopt-report-routine:string

Arguments

string

Is the text (string) to look for.

Default

OFF

No optimization reports are generated.
Description

This option tells the compiler to generate reports on the routines containing specified text as part of their name.

Alternate Options

None

See Also

opt-report, Qopt-report compiler option

opt-streaming-stores, Qopt-streaming-stores

Enables generation of streaming stores for optimization.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -opt-streaming-stores keyword
Windows: /Qopt-streaming-stores:keyword

Arguments

keyword Specifies whether streaming stores are generated. Possible values are:
always Enables generation of streaming
stores for optimization. The
compiler optimizes under the assumption that the application is memory bound.

never Disables generation of streaming stores for optimization. Normal stores are performed.

auto Lets the compiler decide which instructions to use.

Default

-opt-streaming-stores auto
or/Qopt-streaming-stores:auto

The compiler
decides whether to use streaming stores or normal stores.

**Description**

This option enables generation of streaming stores for optimization. This method stores data with instructions that use a non-temporal buffer, which minimizes memory hierarchy pollution. For this option to be effective, the compiler must be able to generate SSE2 (or higher) instructions. For more information, see compiler option \texttt{x} or \texttt{ax}. This option may be useful for applications that can benefit from streaming stores.

**Alternate Options**

None

**See Also**

\texttt{ax, Qax} compiler option
\texttt{x, Qx} compiler option
\texttt{opt-mem-bandwidth, Qopt-mem-bandwidth, Qx} compiler option

**opt-subscript-in-range, Qopt-subscript-in-range**

Determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.

**IDE Equivalent**

None

**Architectures**
Syntax

Linux and Mac OS X: `-opt-subscript-in-range`  
`-no-opt-subscript-in-range`

Windows:  `/Qopt-subscript-in-range`  
`/Qopt-subscript-in-range-`

Arguments  
None

Default

`-no-opt-subscript-in-range`  
`/Qopt-subscript-in-range-`  

The compiler assumes overflows in the intermediate computation of subscript expressions in loops.

Description

This option determines whether the compiler assumes no overflows in the intermediate computation of subscript expressions in loops.  
If you specify `-opt-subscript-in-range` (Linux and Mac OS X) or `/Qopt-subscript-in-range` (Windows), the compiler ignores any data type conversions used and it assumes no overflows in the intermediate computation of subscript expressions. This feature can enable more loop transformations.
Alternate Options

None

Example

The following shows an example where these options can be useful. m is declared as type long (64-bits) and all other variables inside the subscript are declared as type int (32-bits):

\[ A[ i + j + ( n + k) \times m ] \]

Qoption

Passes options to a specified tool.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-Qoption, string, options\)
Windows: \(/Qoption, string, options\)

Arguments

string Is the name of the tool.

options Are one or more comma-separated, valid
options for the designated tool.

Default
OFF
No options are passed to tools.

Description
This option passes options to a specified tool. If an argument contains a space or tab character, you must enclose the entire argument in quotation marks (" "). You must separate multiple arguments with commas.

\textit{string} can be any of the following:

- \texttt{asm} - Indicates the assembler.
- \texttt{link} - Indicates the linker.
- \texttt{prof} - Indicates the profiler.
- \textbf{On Windows systems, the following is also available:}
  - \texttt{masm} - Indicates the Microsoft assembler.
- \textbf{On Linux and Mac OS X systems, the following are also available:}
  - \texttt{as} - Indicates the assembler.
  - \texttt{gas} - Indicates the GNU assembler.
  - \texttt{ld} - Indicates the loader.
  - \texttt{gld} - Indicates the GNU loader.
  - \texttt{lib} - Indicates an additional library.
o cyn - Indicates the crts.o files linked into executables to contain the place to
start execution.

Alternate Options

None

See Also

Qlocation compiler option

qp

See p.

Qpar-adjust-stack

Tells the compiler to generate code to adjust the stack size for a fiber-based
main thread.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Qpar-adjust-stack:n

Arguments

n Is the
stack
size (in
bytes)
for the fiber-based main thread. It must be a number equal to or greater than zero.

**Default**

/Qpar-adjust-stack:0  

No adjustment is made to the main thread stack size.

**Description**

This option tells the compiler to generate code to adjust the stack size for a fiber-based main thread. This can reduce the stack size of threads. For this option to be effective, you must also specify option /Qparallel.

**Alternate Options**

None

**See Also**

864
par-report, Qpar-report

Controls the diagnostic information reported by the auto-parallelizer.

IDE Equivalent

Windows: None
Linux: **Compilation Diagnostics > Auto-Parallelizer Report**
Mac OS X: **Diagnostics > Auto-Parallelizer Report**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-par-report[n]`
Windows: `/Qpar-report[n]`

Arguments

\( n \)

Is a value denoting which diagnostic messages to report. Possible values are:

0  Tells the auto-parallelizer to report no diagnostic information.
1  Tells the auto-parallelizer to
report diagnostic messages for loops successfully auto-parallelized. The compiler also issues a "LOOP AUTO-PARALLELIZED" message for parallel loops.

2 Tells the auto-parallelizer to report diagnostic messages for loops successfully and unsuccessfully auto-parallelized.

3 Tells the auto-parallelizer to report the same diagnostic messages specified by 2 plus additional information about any proven or
assumed dependencies inhibiting auto-parallelization (reasons for not parallelizing).

**Default**

- `-par-report1`
- `/Qpar-report1`

If you do not specify the option on the command line, the default is to display no parallel diagnostic messages.
Description

This option controls the diagnostic information reported by the auto-parallelizer (parallel optimizer). To use this option, you must also specify \texttt{-parallel} (Linux and Mac OS X) or \texttt{/Qparallel} (Windows).

If this option is specified on the command line, the report is sent to \texttt{stdout}.

Alternate Options

None

\texttt{par-runtime-control, Qpar-runtime-control}

Generates code to perform run-time checks for loops that have symbolic loop bounds.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  \texttt{-par-runtime-control}
                     \texttt{-no-par-runtime-control}
Windows:            \texttt{/Qpar-runtime-control}
                     \texttt{/Qpar-runtime-control-}

Arguments

None

Default

\texttt{-no-par-runtime-control}  \texttt{The}
or/Qpar-runtime-control-

compiler uses default heuristics when checking loops.

**Description**

This option generates code to perform run-time checks for loops that have symbolic loop bounds.

If the granularity of a loop is greater than the parallelization threshold, the loop will be executed in parallel.

If you do not specify this option, the compiler may not parallelize loops with symbolic loop bounds if the compile-time granularity estimation of a loop can not ensure it is beneficial to parallelize the loop.

**Alternate Options**

None

**par-schedule, Qpar-schedule**

Lets you specify a scheduling algorithm or a tuning method for loop iterations.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-par-schedule-keyword [=n]`
Windows: 

/par-schedule=keyword[[::]n]

Arguments

keyword

Specifies the scheduling algorithm or tuning method. Possible values are:

**auto**

Lets the compiler or run-time system determine the scheduling algorithm.

**static**

Divides iterations into contiguous pieces.

**static-balanced**

Divides iterations into even-sized chunks.

**static-steal**

Divides iterations into even-sized chunks.
chunks, but allows threads to steal parts of chunks from neighboring threads.

- **dynamic**: Gets a set of iterations dynamically.

- **guided**: Specifies a minimum number of iterations.

- **guided-analytical**: Divides iterations by using exponential distribution or dynamic distribution.

- **runtime**: Defers the scheduling decision until runtime.

\[ n \]

\( n \) is the size of the
chunk or
the number
of iterations
for each
chunk. This
setting can
only be
specified
for static,
dynamic,
and guided.
For more
information,
see the
descriptions
of each
keyword
below.

Default

static-balanced

Iterations
are
divided
into
even-sized
chunks
and the
chunks
are
assigned to the threads in the team in a round-robin fashion in the order of the thread number.

**Description**

This option lets you specify a scheduling algorithm or a tuning method for loop iterations. It specifies how iterations are to be divided among the threads of the team.

This option affects performance tuning and can provide better performance during auto-parallelization.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-par-schedule-auto or /Qpar-schedule-auto</td>
<td>Lets the compiler or run-time system determine the scheduling algorithm. Any possible mapping may occur for iterations to threads in the team.</td>
</tr>
<tr>
<td>-par-schedule-static or /Qpar-schedule-static</td>
<td>Divides iterations into contiguous pieces (chunks) of size $n$. The chunks are assigned to threads in the team in a round-robin fashion in the order of the thread number.</td>
</tr>
</tbody>
</table>
### Option Description

<table>
<thead>
<tr>
<th><strong>Option</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>-par-schedule-static-balanced or /Qpar-schedule-static-balanced</td>
<td>Divides iterations into even-sized chunks. The chunks are assigned to the threads in the team in a round-robin fashion in the order of the thread number.</td>
</tr>
<tr>
<td>-par-schedule-static-steal or /Qpar-schedule-static-steal</td>
<td>Divides iterations into even-sized chunks, but when a thread completes its chunk, it can steal parts of chunks assigned to neighboring threads. Each thread keeps track of $L$ and $U$, which represent the lower and upper bounds of its chunks respectively. Iterations are executed starting from the lower bound, and simultaneously, $L$ is updated to represent the new lower bound.</td>
</tr>
<tr>
<td>-par-schedule-dynamic or /Qpar-schedule-dynamic</td>
<td>Can be used to get a set of iterations dynamically. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-par-schedule-guided or /Qpar-schedule-guided</td>
<td>Can be used to specify a minimum number of iterations. Assigns iterations to threads in chunks as the threads request them. The thread executes the chunk of iterations, then requests another chunk, until no chunks remain to be assigned. For a chunk of size 1, the size of each chunk is proportional to the number of unassigned iterations divided by the number of threads, decreasing to 1. For an $n$ with value $k$ (greater than 1), the size of each chunk is determined in the same way with the restriction that the chunks do not contain fewer than $k$ iterations (except for the last chunk to be assigned, which may have fewer than $k$ iterations). If no $n$ is specified, the default is 1.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-par-schedule-guided-analytical or /Qpar-schedule-guided-analytical</td>
<td>Divides iterations by using exponential distribution or dynamic distribution. The method depends on run-time implementation. Loop bounds are calculated with faster synchronization and chunks are dynamically dispatched at run time by threads in the team.</td>
</tr>
<tr>
<td>-par-schedule-runtime or /Qpar-schedule-runtime</td>
<td>Defers the scheduling decision until run time. The scheduling algorithm and chunk size are then taken from the setting of environment variable OMP_SCHEDULE.</td>
</tr>
</tbody>
</table>

**Alternate Options**

None

**par-threshold, Qpar-threshold**

Sets a threshold for the auto-parallelization of loops.

**IDE Equivalent**

Windows: None

Linux: **Optimization > Auto-Parallelization Threshold**

Mac OS X: **Optimization > Auto-Parallelization Threshold**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-par-threshold[n]`
Windows: 
/Qpar-threshold[:n]

Arguments

$n$

Is an integer whose value is the threshold for the auto-parallelization of loops. Possible values are 0 through 100. If $n$ is 0, loops get auto-parallelized always, regardless of computation work volume. If $n$ is 100, loops get auto-parallelized when performance gains are predicted based on the compiler
analysis data. Loops get auto-parallelized only if profitable parallel execution is almost certain. The intermediate 1 to 99 values represent the percentage probability for profitable speed-up. For example, \( n=50 \) directs the compiler to parallelize only if there is a 50% probability of the code speeding up if executed in parallel.
**Default**

-par-threshold100
or/Qpar-threshold100

Loops get auto-parallelized only if profitable parallel execution is almost certain. This is also the default if you do not specify \( n \).

**Description**

This option sets a threshold for the auto-parallelization of loops based on the probability of profitable execution of the loop in parallel. To use this option, you must also specify `-parallel` (Linux and Mac OS X) or `/Qparallel` (Windows).

This option is useful for loops whose computation work volume cannot be determined at compile-time. The threshold is usually relevant when the loop trip count is unknown at compile-time.

The compiler applies a heuristic that tries to balance the overhead of creating multiple threads versus the amount of work available to be shared amongst the threads.

**Alternate Options**

None
**parallel, Qparallel**

Tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.

**IDE Equivalent**

Windows: Optimization > Parallelization
Linux: Optimization > Parallelization
Mac OS X: Optimization > Parallelization

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-parallel`
Windows: `/Qparallel`

**Arguments**

None

**Default**

OFF  
Multithreaded code is not generated for loops that can be safely executed in parallel.

**Description**
This option tells the auto-parallelizer to generate multithreaded code for loops that can be safely executed in parallel.
To use this option, you must also specify option `O2` or `O3`.

![Note]

On Mac OS X systems, when you enable automatic parallelization, you must also set the `DYLD_LIBRARY_PATH` environment variable within Xcode or an error will be displayed.

**Alternate Options**

None

**See Also**

`pc`, `Qpc`

Enables control of floating-point significand precision.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X: `-pcn`

Windows: `/Qpcn`

**Arguments**

$n$

Is the floating-point significand
precision. Possible values are:

32 Rounds the significand to 24 bits (single precision).

64 Rounds the significand to 53 bits (double precision).

80 Rounds the significand to 64 bits (extended precision).

Default

- pc80
or/Qpc64

On Linux* and Mac OS* X systems, the floating-
Description

This option enables control of floating-point significand precision. Some floating-point algorithms are sensitive to the accuracy of the significand, or fractional part of the floating-point value. For example, iterative operations like division and finding the square root can run faster if you lower the precision with the this option.

Note that a change of the default precision control or rounding mode, for example, by using the -pc32 (Linux and Mac OS X) or /Qpc32 (Windows) option or by user intervention, may affect the results returned by some of the mathematical functions.

Alternate Options

None

Qpchi

Enable precompiled header coexistence to reduce build time.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Qpchi
          /Qpchi-

Arguments
None

Default
ON

The compiler enables precompiled header coexistence.

Description
This option enables precompiled header (PCH) files generated by the Intel® C++ compiler and those generated by the Microsoft Visual C++* compiler to coexist, which reduces build time.
If build time is not an issue and you do not want an additional set of PCH files on your system, specify /Qpchi-.

Alternate Options
None

884
**mp1, Qprec**

Improves floating-point precision and consistency.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: **-mp1**

Windows: **/Qprec**

**Arguments**

None

**Default**

OFF

The compiler provides good accuracy and run-time performance at the expense of less consistent floating-point
Description

This option improves floating-point consistency. It ensures the out-of-range check of operands of transcendental functions and improves the accuracy of floating-point compares.

This option prevents the compiler from performing optimizations that change NaN comparison semantics and causes all values to be truncated to declared precision before they are used in comparisons. It also causes the compiler to use library routines that give better precision results compared to the X87 transcendental instructions.

This option disables fewer optimizations and has less impact on performance than option \texttt{mp} or \texttt{Op}.

Alternate Options

None

See Also

\texttt{mp} compiler option

\textbf{prec-div, Qprec-div}

Improves precision of floating-point divides.

IDE Equivalent

None

Architectures

IA-32, Intel\textregistered\ 64, IA-64 architectures

Syntax

Linux and Mac OS X: \texttt{-prec-div}
The compiler uses this method for floating-point divides.

Description

This option improves precision of floating-point divides. It has a slight impact on speed.

With some optimizations, such as \(-xSSE2\) (Linux) or /QxSSE2 (Windows), the compiler may change floating-point division computations into multiplication by the reciprocal of the denominator. For example, \(A/B\) is computed as \(A \times (1/B)\) to improve the speed of the computation.

However, sometimes the value produced by this transformation is not as accurate as full IEEE division. When it is important to have fully precise IEEE division, use this option to disable the floating-point division-to-multiplication optimization. The result is more accurate, with some loss of performance.
If you specify `-no-prec-div` (Linux and Mac OS X) or `/Qprec-div-` (Windows), it enables optimizations that give slightly less precise results than full IEEE division.

**Alternate Options**

None

**prec-sqrt, Qprec-sqrt**

Improves precision of square root implementations.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

**Linux and Mac OS X:** `-prec-sqrt`  
- `-no-prec-sqrt`

**Windows:** `/Qprec-sqrt`  
/`Qprec-sqrt-`

**Arguments**

None

**Default**

- `-no-prec-sqrt`  
  The compiler uses a faster but less precise implementation

or `/Qprec-sqrt-`
Note that the default is `-prec-sqrt` or `/Qprec-sqrt` if any of the following options are specified: `/Od`, `/Op`, or `/Qprec` on Windows systems; `-O0`, `-mp`, or `-mp1` on Linux and Mac OS X systems.

**Description**

This option improves precision of square root implementations. It has a slight impact on speed.

This option inhibits any optimizations that can adversely affect the precision of a square root computation. The result is fully precise square root implementations, with some loss of performance.

**Alternate Options**

None

**prof-data-order, Qprof-data-order**

Enables or disables data ordering if profiling information is enabled.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux:       -prof-data-order
             -no-prof-data-order
Mac OS X:   None
Windows:    /Qprof-data-order
             /Qprof-data-order-

Arguments
None

Default
-no-prof-data-order  Data
or /Qprof-data-order- ordering
is
  disabled.

Description
This option enables or disables data ordering if profiling information is enabled. It controls the use of profiling information to order static program data items. For this option to be effective, you must do the following:

- **For instrumentation compilation, you must specify** -prof-gen=globdata (Linux) or /Qprof-gen:globdata (Windows).
- **For feedback compilation, you must specify** -prof-use (Linux) or /Qprof-use (Windows). You must not use multi-file optimization by
specifying options such as option -iipo (Linux) or /Qiipo (Windows), or option -iipo-c (Linux) or /Qiipo-c (Windows).

Alternate Options

None

See Also

prof-gen, Qprof-gen compiler option
prof-use, Qprof-use compiler option
prof-func-order, Qprof-func-order compiler option

prof-dir, Qprof-dir

Specifies a directory for profiling information output files.

IDE Equivalent

Windows: General > Profile Directory
Linux: Compiler > Profile Directory
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -prof-dir dir
Windows: /Qprof-dir dir

Arguments

dir

Is the name of the directory.
Default

OFF

Profiling output files are placed in the directory where the program is compiled.

Description

This option specifies a directory for profiling information output files (*.dyn and *.dpi). The specified directory must already exist.

You should specify this option using the same directory name for both instrumentation and feedback compilations. If you move the .dyn files, you need to specify the new path.

Alternate Options

None

prof-file, Qprof-file

Specifies an alternate file name for the profiling summary files.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -prof-file file
Windows: /Qprof-file file

Arguments

file

Is the name of the profiling summary file.

Default

OFF

The profiling summary files have the file name pgopti.*

Description

This option specifies an alternate file name for the profiling summary files. The file is used as the base name for files created by different profiling passes. If you add this option to profmerge, the .dpi file will be named file.dpi instead of pgopti.dpi. If you specify -prof-genx (Linux and Mac OS X) or /Qprof-genx (Windows) with this option, the .spi and .spl files will be named file.spi and file.spl instead of pgopti.spi and pgopti.spl.
If you specify `-prof-use` (Linux and Mac OS X) or `/Qprof-use` (Windows) with this option, the .dpi file will be named `file.dpi` instead of `pgopti.dpi`.

Alternate Options

None

See Also

`prof-gen, Qprof-gen` compiler option
`prof-use, Qprof-use` compiler option

`prof-func-order, Qprof-func-order`

Enables or disables function ordering if profiling information is enabled.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: 
- `prof-func-order`
- `no-prof-func-order`

Mac OS X: None

Windows: 
- `/Qprof-func-order`
- `/Qprof-func-order-`

Arguments

None

Default

`-no-prof-func-order` Function
or /Qprof-func-order-
ordering
is
disabled.

Description

This option enables or disables function ordering if profiling information is enabled.

For this option to be effective, you must do the following:

• **For instrumentation compilation, you must specify** `-prof-gen=srcpos` (Linux) or `/Qprof-gen:srcpos` (Windows).

• **For feedback compilation, you must specify** `-prof-use` (Linux) or `/Qprof-use` (Windows). You must not use multi-file optimization by specifying options such as option `-ipo` (Linux) or `/Qipo` (Windows), or option `-ipo-c` (Linux) or `/Qipo-c` (Windows).

If you enable profiling information by specifying option `-prof-use` (Linux) or `/Qprof-use` (Windows), `-prof-func-groups` (Linux) and `/Qprof-func-groups` (Windows) are set and function grouping is enabled. However, if you explicitly enable `-prof-func-order` (Linux) or `/Qprof-func-order` (Windows), function ordering is performed instead of function grouping.

On Linux* systems, this option is only available for Linux linker 2.15.94.0.1, or later.

To set the hotness threshold for function grouping and function ordering, use option `-prof-hotness-threshold` (Linux) or `/Qprof-hotness-threshold` (Windows).

Alternate Options

None

The following example shows how to use this option on a Windows system:

```
icl /Qprof-gen:globdata file1.c file2.c /Fe instrumented.exe
./instrumented.exe
icl /Qprof-use /Qprof-func-order file1.c file2.c /Fe feedback.exe
```
The following example shows how to use this option on a Linux system:

```
icl -prof-gen:globdata file1.c file2.c -o instrumented
./instrumented.exe
icl -prof-use -prof-func-order file1.c file2.c -o feedback
```

See Also

prof-hotness-threshold, Qprof-hotness-threshold compiler option
prof-gen, Qprof-gen compiler option
prof-use, Qprof-use compiler option
prof-data-order, Qprof-data-order compiler option
prof-func-groups compiler option

**prof-gen, Qprof-gen**

Produces an instrumented object file that can be used in profile-guided optimization.

**IDE Equivalent**

Windows: **General > Profile Guided Optimization**
**Optimization > Profile Guided Optimization**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-prof-gen[=keyword]`
- `-no-prof-gen`

**Windows:** `/Qprof-gen[=keyword]`
/`Qprof-gen-`

**Arguments**

*keyword* Specifies details for the
instrumented file.
Possible values are:

**default**  Produces an instrumented object file. This is the same as specifying `-prof-gen`
(Windows*)
or `/Qprof-gen`
(Windows*)
with no keyword.

**srcpos**  Produces an instrumented object file that includes extra source position information. This option is the same as option `-prof-genx`
(Windows*)
or `/Qprof-genx`
genx
(Windows*), which are deprecated.

globdata Produces an instrumented object file that includes information for global data layout.

Default

-no-prof-gen or /Qprof-gen-

Profile generation is disabled.

Description

This option produces an instrumented object file that can be used in profile-guided optimization. It gets the execution count of each basic block. If you specify keyword srcpos or globdata, a static profile information file (.spi) is created. These settings may increase the time needed to do a parallel build using -prof-gen, because of contention writing the .spi file. These options are used in phase 1 of the Profile Guided Optimizer (PGO) to instruct the compiler to produce instrumented code in your object files in preparation for instrumented execution.

Alternate Options

None

898
prof-hotness-threshold, Qprof-hotness-threshold

Lets you set the hotness threshold for function grouping and function ordering.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -prof-hotness-threshold=n
Mac OS X: None
Windows: /Qprof-hotness-threshold:n

Arguments

n

Is the hotness threshold.

n is a percentage having a value between 0 and 100 inclusive. If you specify 0, there will be no hotness threshold
setting in effect for function grouping and function ordering.

Default

OFF

The compiler's default hotness threshold setting of 10 percent is in effect for function grouping and function ordering.

Description

This option lets you set the hotness threshold for function grouping and function ordering.

The "hotness threshold" is the percentage of functions in the application that should be placed in the application's hot region. The hot region is the most
frequently executed part of the application. By grouping these functions together into one hot region, they have a greater probability of remaining resident in the instruction cache. This can enhance the application's performance.

For this option to take effect, you must specify option `-prof-use` (Linux) or `/Qprof-use` (Windows) and one of the following:

- **On Linux systems**: `-prof-func-groups` or `-prof-func-order`
- **On Windows systems**: `/Qprof-func-order`

**Alternate Options**

None

**See Also**

`prof-use`, `Qprof-use` compiler option
`prof-func-groups` compiler option
`prof-func-order`, `Qprof-func-order` compiler option

**prof-src-dir**, `Qprof-src-dir`

Determines whether directory information of the source file under compilation is considered when looking up profile data records.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X**: `-prof-src-dir`  
 `-no-prof-src-dir`

**Windows**: `/Qprof-src-dir`  
 `/Qprof-src-dir-`

**Arguments**
None

**Default**

- `-prof-src-dir` or `/Qprof-src-dir`

Directory information is used when looking up profile data records in the `.dpi` file.

**Description**

This option determines whether directory information of the source file under compilation is considered when looking up profile data records in the `.dpi` file. To use this option, you must also specify option `-prof-use` (Linux and Mac OS X) or `/Qprof-use` (Windows).

If the option is enabled, directory information is considered when looking up the profile data records within the `.dpi` file. You can specify directory information by using one of the following options:

- **Linux and Mac OS X**: `-prof-src-root` or `-prof-src-root-cwd`
- **Windows**: `/Qprof-src-root` or `/Qprof-src-root-cwd`

If the option is disabled, directory information is ignored and only the name of the file is used to find the profile data record.

Note that options `-prof-src-dir` (Linux and Mac OS X) and `/Qprof-src-dir` (Windows) control how the names of the user's source files get represented within the `.dyn` or `.dpi` files. Options `-prof-dir` (Linux and Mac OS X) and `/Qprof-dir` (Windows) specify the location of the `.dyn` or the `.dpi` files.

**Alternate Options**

902
None

See Also

prof-use, Qprof-use compiler option
prof-src-root, Qprof-src-root compiler option
prof-src-root-cwd, Qprof-src-root-cwd compiler option

prof-src-root, Qprof-src-root

Lets you use relative directory paths when looking up profile data and specifies a directory as the base.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -prof-src-root=dir
Windows: /Qprof-src-root:dir

Arguments

dir Is the base for the relative paths.

Default

OFF The setting of relevant
options determines the path used when looking up profile data records.

Description

This option lets you use relative directory paths when looking up profile data in .dpi files. It lets you specify a directory as the base. The paths are relative to a base directory specified during the -prof-gen (Linux and Mac OS X) or /Qprof-gen (Windows) compilation phase.

This option is available during the following phases of compilation:

- **Linux and Mac OS X**: -prof-gen and -prof-use phases
- **Windows**: /Qprof-gen and /Qprof-use phases

When this option is specified during the -prof-gen or /Qprof-gen phase, it stores information into the .dyn or .dpi file. Then, when .dyn files are merged together or the .dpi file is loaded, only the directory information below the root directory is used for forming the lookup key.

When this option is specified during the -prof-use or /Qprof-use phase, it specifies a root directory that replaces the root directory specified at the -prof-gen or /Qprof-gen phase for forming the lookup keys.

To be effective, this option or option -prof-src-root-cwd (Linux and Mac OS X) or /Qprof-src-root-cwd (Windows) must be specified during the -prof-gen or /Qprof-gen phase. In addition, if one of these options is not specified, absolute paths are used in the .dpi file.

Alternate Options

None
Consider the initial `-prof-gen` compilation of the source file `c:\user1\feature_foo\myproject\common\glob.c`:

```
icc -prof-gen -prof-src-root=c:\user1\feature_foo\myproject -c common\glob.c
```

For the `-prof-use` phase, the file `glob.c` could be moved into the directory `c:\user2\feature_bar\myproject\common\glob.c` and profile information would be found from the `.dpi` when using the following:

```
icc -prof-use -prof-src-root=c:\user2\feature_bar\myproject -c common\glob.c
```

If you do not use option `-prof-src-root` during the `-prof-gen` phase, by default, the `-prof-use` compilation can only find the profile data if the file is compiled in the `c:\user1\feature_foo\my_project\common` directory.

**See Also**

`prof-gen`, `Qprof-gen` compiler option
`prof-use`, `Qprof-use` compiler option
`prof-src-dir`, `Qprof-src-dir` compiler option
`prof-src-root-cwd`, `Qprof-src-root-cwd` compiler option

**prof-src-root-cwd, Qprof-src-root-cwd**

Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-prof-src-root-cwd`

Windows: `/Qprof-src-root-cwd`

**Arguments**
None

Default

OFF

The setting of relevant options determines the path used when looking up profile data records.

Description

This option lets you use relative directory paths when looking up profile data in .dpi files. It specifies the current working directory as the base. To use this option, you must also specify option -prof-use (Linux and Mac OS) or /Qprof-use (Windows).

This option is available during the following phases of compilation:

- **Linux and Mac OS X**: -prof-gen and -prof-use phases
- **Windows**: /Qprof-gen and /Qprof-use phases

When this option is specified during the -prof-gen or /Qprof-gen phase, it stores information into the .dyn or .dpi file. Then, when .dyn files are merged together or the .dpi file is loaded, only the directory information below the root directory is used for forming the lookup key.

When this option is specified during the -prof-use or /Qprof-use phase, it specifies a root directory that replaces the root directory specified at the -prof-gen or /Qprof-gen phase for forming the lookup keys.

To be effective, this option or option -prof-src-root (Linux and Mac OS X) or /Qprof-src-root (Windows) must be specified during the -prof-gen or
/Qprof-gen phase. In addition, if one of these options is not specified, absolute paths are used in the .dpi file.

Alternate Options

None

See Also

prof-gen, Qprof-gen compiler option
prof-use, Qprof-use compiler option
prof-src-dir, Qprof-src-dir compiler option
prof-src-root, Qprof-src-root compiler option

prof-use, Qprof-use

Enables the use of profiling information during optimization.

IDE Equivalent

Windows: General > Profile Guided Optimization
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -prof-use[=arg]
-no-prof-use

Windows: /Qprof-use[::arg]
/Qprof-use-

Arguments

arg Specifies additional
instructions. Possible values are:

**weighted**  Tells the profmerge utility to apply a weighting to the .dyn file values when creating the .dpi file to normalize the data counts when the training runs have different execution durations. This argument only has an effect when the compiler invokes the profmerge utility to create the .dpi file. This argument does not have an effect if the .dpi file was previously created without weighting.

**[no]merge** Enables or
disables automatic invocation of the profmerge utility. The default is merge. Note that you cannot specify both weighted and nomerge. If you try to specify both values, a warning will be displayed and nomerge takes precedence.

**default** Enables the use of profiling information during optimization. The profmerge utility is invoked by default. This value is the same as specifying –prof-use (Linux and Mac OS X) or /Qprof-use
(Windows) with no argument.

Default

-no-prof-use or /Qprof-use-

Profiling information is not used during optimization.

Description

This option enables the use of profiling information (including function splitting and function grouping) during optimization. It enables option -fnsplit (Linux) or /Qfnsplit (Windows).

This option instructs the compiler to produce a profile-optimized executable and it merges available profiling output files into a pgopti.dpi file.

Note that there is no way to turn off function grouping if you enable it using this option.

To set the hotness threshold for function grouping and function ordering, use option -prof-hotness-threshold (Linux) or /Qprof-hotness-threshold (Windows).

Alternate Options

None

See Also

prof-hotness-threshold, Qprof-hotness-threshold compiler option

rcd, Qrcd

Enables fast float-to-integer conversions.
IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -rcd
Windows: /Qrcd

Arguments

None

Default

OFF

Floating-point values are truncated when a conversion to an integer is involved. On Windows, this is the same as specifying /QIfist-.
Description

This option enables fast float-to-integer conversions. It can improve the performance of code that requires floating-point-to-integer conversions. The system default floating-point rounding mode is round-to-nearest. However, the C language requires floating-point values to be truncated when a conversion to an integer is involved. To do this, the compiler must change the rounding mode to truncation before each floating-point-to-integer conversion and change it back afterwards.

This option disables the change to truncation of the rounding mode for all floating-point calculations, including floating-point-to-integer conversions. This option can improve performance, but floating-point conversions to integer will not conform to C semantics.

Alternate Options

Linux and Mac OS X: None
Windows: /QIfist

rct, Qrct

Sets the internal FPU rounding control to Truncate.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -rct
Windows: /Qrct

Arguments
None

Default

OFF

The compiler uses the default setting for the FPU rounding control.

Description

This option sets the internal FPU rounding control to Truncate.

Alternate Options

None

restrict, Qrestrict

Determines whether pointer disambiguation is enabled with the restrict qualifier.

IDE Equivalent

Windows: Language > Recognize Restrict Keyword
Linux: Language > Recognize Restrict Keyword
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: `-restrict`  
 `-no-restrict`

Windows: `/Qrestrict`  
 `/Qrestrict-`

**Arguments**

None

**Default**

`-no-restrict` or `/Qrestrict-`  

**Pointers** are not qualified with the restrict keyword.

**Description**

This option determines whether pointer disambiguation is enabled with the restrict qualifier. Options `-restrict` (Linux and Mac OS X) and `/Qrestrict` (Windows) enable the recognition of the restrict keyword as defined by the ANSI standard.

By qualifying a pointer with the restrict keyword, you assert that an object accessed by the pointer is only accessed by that pointer in the given scope. You should use the restrict keyword only when this is true. When the assertion is true, the restrict option will have no effect on program correctness, but may allow better optimization.

**Alternate Options**

None

**See Also**
Qc99 compiler option

Qsafeseh

Registers exception handlers for safe exception handling.

IDE Equivalent

None

Architectures

IA-32 architecture

Syntax

Linux and Mac OS X: None
Windows: /Qsafeseh[-]

Arguments

None

Default

ON (if /Qvc7.1 or higher is specified)

Description

Registers exception handlers for safe exception handling. It also marks objects as "compatible with the Registered Exception Handling feature" whether they contain handlers or not. This is important because the Windows linker will only generate the "special registered EH table" if ALL objects that it is building into an
image are marked as compatible. If any objects are not marked as compatible, the EH table is not generated.

Digital signatures certify security and are required for components that are shipped with Windows, such as device drivers.

Alternate Options
None

See Also

/\texttt{E}H compiler option

\texttt{save-temps}, \texttt{Qsave-temps}

Tells the compiler to save intermediate files created during compilation.

IDE Equivalent
None

Architectures
IA-32, Intel\textregistered 64, IA-64 architectures

Syntax

Linux and Mac OS X: \texttt{-save-temps}
\texttt{ -no-save-temps}

Windows: \texttt{/Qsave-temps}
\texttt{/Qsave-temps-}

Arguments
None

Default

Linux and Mac OS X: \texttt{-no-save-temps} On Linux
Windows: .obj files are saved and Mac OS X systems, the compiler deletes intermediate files after compilation is completed. On Windows systems, the compiler saves only intermediate object files after compilation is completed.

Description

This option tells the compiler to save intermediate files created during compilation. The names of the files saved are based on the name of the source file; the files are saved in the current working directory.

If `-save-temps` or `/Qsave-temps` is specified, the following occurs:

- The object .o file (Linux and Mac OS X) or .obj file (Windows) is saved.
- The assembler .s file (Linux and Mac OS X) or .asm file (Windows) is saved if you specified `-use-asm` (Linux or Mac OS X) or `/Quse-asm` (Windows).
If `-no-save-temps` is specified on Linux or Mac OS X systems, the following occurs:

- The `.o` file is put into `/tmp` and deleted after calling `ld`.
- The preprocessed file is not saved after it has been used by the compiler.

If `/Qsave-temps` is specified on Windows systems, the following occurs:

- The `.obj` file is not saved after the linker step.
- The preprocessed file is not saved after it has been used by the compiler.

**Note**

This option only saves intermediate files that are normally created during compilation.

**Alternate Options**

None

**Example**

If you compile program `my_foo.c` on a Linux or Mac OS X system and you specify option `-save-temps` and option `-use-asm`, the compilation will produce files `my_foo.o` and `my_foo.s`.

If you compile program `my_foo.c` on a Windows system and you specify option `/Qsave-temps` and option `/Quse-asm`, the compilation will produce files `my_foo.o` and `my_foo.asm`.

**scalar-rep, Qscalar-rep**

Enables scalar replacement performed during loop transformation.

**IDE Equivalent**

None

**Architectures**
IA-32 architecture

Syntax

Linux and Mac OS X: `-scalar-rep`
  `-no-scalar-rep`
Windows: `/Qscalar-rep`
  `/Qscalar-rep-

Arguments
None

Default

```
-no-scalar-rep                 Scalar
or/Qscalar-rep-               replacement is
not performed
 during loop
 transformation.
```

Description

This option enables scalar replacement performed during loop transformation. To
use this option, you must also specify `O3`.

Alternate Options

None

See Also

O compiler option

mserialize-volatile, Qserialize-volatile

Determines whether strict memory access ordering is imposed for volatile data
object references.
IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux:  
-mserialize-volatile
-mno-serialize-volatile

Mac OS X:  None

Windows:  /Qserialize-volatile
/Qserialize-volatile-

Arguments

None

Default

-mno-serialize-volatile or /Qserialize-volatile-  The compiler uses default memory access ordering.

Description

This option determines whether strict memory access ordering is imposed for volatile data object references.

If you specify -mno-serialize-volatile, the compiler may suppress both run-time and compile-time memory access ordering for volatile data object
references. Specifically, the .rel/.acq completers will not be issued on referencing loads and stores.

Alternate Options
None

Qsfalign
Specifies stack alignment for functions.

IDE Equivalent
None

Architectures
IA-32 architecture

Syntax
Linux and Mac OS X: None
Windows: /Qsfalign[n]

Arguments

n
Is the byte size of aligned variables.
Possible values are:
8 Specifies that alignment should occur for
functions with 8-byte aligned variables. At this setting the compiler aligns the stack to 16 bytes if there is any 16-byte or 8-byte data on the stack. For 8-byte data, the compiler only aligns the stack if the alignment will produce a performance advantage.

16 Specifies that alignment should occur for functions
with 16-byte aligned variables. At this setting, the compiler only aligns the stack for 16-byte data. No attempt is made to align for 8-byte data.

Default

/Qsfalign8

Alignment occurs for functions with 8-byte aligned variables.

Description

This option specifies stack alignment for functions. It lets you disable the normal optimization that aligns a stack for 8-byte data. If you do not specify \( n \), stack alignment occurs for all functions. If you specify /Qsfalign-, no stack alignment occurs for any function.

Alternate Options
None

std, Qstd

Tells the compiler to conform to a specific language standard.

IDE Equivalent

Windows: Language > Enable C++0x Support
Linux: Language > ANSI Conformance
Mac OS X: Language > C ANSI Conformance

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -std=val
Windows: /Qstd:val

Arguments

val

Possible values are:


**gnu89** Conforms to ISO C90 plus GNU* extensions.

**gnu++98** Conforms to the 1998 ISO C++ standard plus GNU extensions.

**c++0x** Enable support for the following C++0x features:
- Empty macro arguments
- Variadic macros
- Type `long long`
- Trailing comma in enum definition
- Concatenation of mixed-width string literals
- Extended friend declarations
- Use of ">>" to close two
• Relaxed rules for use of "typename"
• Relaxed rules for disambiguation using the "template" keyword
• Copy constructor does not need to be callable on direct reference
• Binding to class rvalue
• "extern template" to suppress instantiation of an entity
• "auto" type specifier
• decltype operator
• static_assert
• compliant
__func__

- lambda
  expressions

Default

-std=gnu89  (default for C)

Conforms to
ISO C90
plus GNU
extensions.

-std=gnu++98  (default for C++)

Conforms to
the 1998
ISO C++
standard
plus GNU*
extensions.

/Qstd=c89

Conforms to
the ISO/IEC
9899:1990
International
Standard.

Description

Tells the compiler to conform to a specific language standard.

Alternate Options

None

sox, Qsox
Tells the compiler to save the compiler options and version number in the executable.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  
\[ -sox \\  
\-no-sox \]

Windows:  
\[ /Qsox \]
\[ /Qsox- \]

**Arguments**

None

**Default**

\[ -no-sox \]  
\[ or /Qsox- \]

The compiler does not save the compiler options and version number in the executable.

**Description**

928
This option tells the compiler to save the compiler options and version number in the executable. The size of the executable on disk is increased slightly by the inclusion of these information strings.

This option forces the compiler to embed in each object file or assembly output a string that contains information about the compiler version and compilation options for each source file that has been compiled. When you link the object files into an executable file, the linker places each of the information strings into the header of the executable. It is then possible to use a tool, such as a strings utility, to determine what options were used to build the executable file.

If `--no-sox` or `/Qsox-` is specified, this extra information is not put into the object or assembly output generated by the compiler.

Alternate Options

None

**tcheck, Qtcheck**

Enables analysis of threaded applications.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux: `-tcheck`

Mac OS X: None

Windows: `/Qtcheck`

**Arguments**

None
Default

OFF

Threaded applications are not instrumented by the compiler for analysis by Intel® Thread Checker.

Description

This option enables analysis of threaded applications.
To use this option, you must have Intel® Thread Checker installed, which is one of the Intel® Threading Analysis Tools. If you do not have this tool installed, the compilation will fail. Remove the -tcheck (Linux) or /Qtcheck (Windows) option from the command line and recompile.
For more information about Intel® Thread Checker (including an evaluation copy), open the page associated with threading tools at Intel® Software Development Products.

Alternate Options

None

tcollect, Qtcollect

Inserts instrumentation probes calling the Intel® Trace Collector API.

IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: \(-tcollect[lib]\)
Mac OS X: None
Windows: \(/Qtcollect[:lib]\)

Arguments

\(lib\) Is one of the Intel® Trace Collector libraries; for example, VT, VTcs, VTmc, or VTfs. If you do not specify \(lib\), the default library is VT.

Default
OFF Instrumentation probes are not inserted into compiled applications.

Description

This option inserts instrumentation probes calling the Intel® Trace Collector API. To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

This option provides a flexible and convenient way of instrumenting functions of a compiled application. For every function, the entry and exit points are instrumented at compile time to let the Intel® Trace Collector record functions beyond the default MPI calls. For non-MPI applications (for example, threaded or serial), you must ensure that the Intel® Trace Collector is properly initialized (VT_initialize/VT_init).

⚠️ Caution

Be careful with full instrumentation because this feature can produce very large trace files.

For more details, see the Intel® Trace Collector User Guide.

Alternate Options

None

See Also

tcollect-filter, Qtcollect-filter compiler option

tcollect-filter, Qtcollect-filter

Lets you enable or disable the instrumentation of specified functions.
IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: \texttt{-tcollect-filter \textit{file}}

Mac OS X: None

Windows: \texttt{/Qtcollect-filter:file}

Arguments

\textit{file}

Is a configuration file that lists filters, one per line. Each filter consists of a regular expression string and a switch. Strings with leading or trailing white spaces must be quoted. Other strings do not have
to be quoted. The switch value can be ON, on, OFF, or off.

**Default**

| OFF       | Functions are not instrumented. However, if option `-tcollect` (Linux) or `/Qtcollect` (Windows) is specified, the filter setting is ".* ON" and all functions get instrumented. |

**Description**

This option lets you enable or disable the instrumentation of specified functions. During instrumentation, the regular expressions in the file are matched against the function names. The switch specifies whether matching functions are to be instrumented or not. Multiple filters are evaluated from top to bottom with increasing precedence.
The names of the functions to match against are formatted as follows:

- **C++ function names are demangled and the C++ class hierarchy is used.**
  
  Function parameters are stripped to keep the function names shorter.

- **The source file name is followed by a colon-separated function name.**
  
  Source file names should contain the full path, if available. For example:

  
  `/home/joe/src/foo.c:FOO_bar`

- **Classes and function names are separated by double colons. For example:**

  
  `/home/joe/src/foo.cpp::app::foo::bar`

You can use option `-opt-report` (Linux) or `/Qopt-report` (Windows) to get a full list of file and function names that the compiler recognizes from the compilation unit. This list can be used as the basis for filtering in the configuration file.

To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

For more details, see the Intel® Trace Collector User Guide.

**Alternate Options**

None

Consider the following filters in a configuration file:

```
'*' OFF '.*vector.*' ON
```

The above will cause instrumentation of only those functions having the string 'vector' in their names. No other function will be instrumented. Note that reversing the order of the two lines will prevent instrumentation of all functions.

To get a list of the file or routine strings that can be matched by the regular expression filters, generate an optimization report with tcollect information. For example:

```
Windows OS: icl /Qtcollect /Qopt-report /Qopt-report-phase tcollect
Linux OS: icc -tcollect -opt-report -opt-report-phase tcollect
```

**See Also**

`tcollect`, `Qtcollect` compiler option
**ftemplate-depth, Qtemplate-depth**

Control the depth in which recursive templates are expanded.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-ftemplate-depth-\(n\)`

Windows: `/Qtemplate-depth-\(n\)`

**Arguments**

\(n\)

The number of recursive templates that are expanded.

**Default**

OFF

**Description**

Control the depth in which recursive templates are expanded. On Linux*, this option is supported only by invoking the compiler with `icpc`.

**Alternate Options**

None
ftrapuv, Qtrapuv

Initializes stack local variables to an unusual value to aid error detection.

IDE Equivalent

Windows: **Code Generation > Initialize Local Variables to NaN**
Linux: **Code Generation > Initialize Local Variables to NaN**
Mac OS X: **Code Generation > Initialize Local Variables to NaN**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-ftrapuv`
Windows: `/Qtrapuv`

Arguments

None

Default

OFF

The compiler does not initialize local variables.

Description

This option initializes stack local variables to an unusual value to aid error detection. Normally, these local variables should be initialized in the application.
The option sets any uninitialized local variables that are allocated on the stack to a value that is typically interpreted as a very large integer or an invalid address. References to these variables are then likely to cause run-time errors that can help you detect coding errors.

This option sets option `-g` (Linux and Mac OS X) and `/zi` or `/Z7` (Windows).

Alternate Options

None

See Also

g, zi, Z7 compiler options

unroll-aggressive, Qunroll-aggressive

Determines whether the compiler uses more aggressive unrolling for certain loops.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  

-unroll-aggressive
-no-unroll-aggressive

Windows:  

/Qunroll-aggressive
/Qunroll-aggressive-
The compiler uses default heuristics when unrolling loops.

**Description**

This option determines whether the compiler uses more aggressive unrolling for certain loops. The positive form of the option may improve performance. On IA-32 architecture and Intel® 64 architecture, this option enables aggressive, complete unrolling for loops with small constant trip counts. On IA-64 architecture, this option enables additional complete unrolling for loops that have multiple exits or outer loops that have a small constant trip count.

**Alternate Options**

None

**unroll, Qunroll**

Tells the compiler the maximum number of times to unroll loops.

**IDE Equivalent**

Windows: Optimization > Loop Unrolling
Linux: Optimization > Loop Unroll Count
Mac OS X: Optimization > Loop Unrolling

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: 
\texttt{-unroll} [=n]

Windows: 
\texttt{/Qunroll} [=n]

Arguments

\( n \)

Is the maximum number of times a loop can be unrolled. To disable loop unrolling, specify 0. On systems using IA-64 architecture, you can only specify a value of 0.

Default

\texttt{-unroll} \hspace{5cm} \texttt{or/Qunroll}

The compiler uses default heuristics when unrolling
Description

This option tells the compiler the maximum number of times to unroll loops. If you do not specify $n$, the optimizer determines how many times loops can be unrolled.

Alternate Options

Linux and Mac OS X: -funroll-loops
Windows: None

use-asm, Quse-asm

Tells the compiler to produce objects through the assembler.

IDE Equivalent

None

Architectures

-use-asm: IA-32 architecture, Intel® 64 architecture, IA-64 architecture
/Quse-asm: IA-64 architecture

Syntax

Linux and Mac OS X: -use-asm
- no-use-asm
Windows: /Quse-asm
/Quse-asm-

Arguments

None

Default
-no-use-asm

or/Quse-asm-

The compiler produces objects directly.

Description

This option tells the compiler to produce objects through the assembler.

Alternate Options

None

Quse-msasm-symbols

Tells the compiler to use a dollar sign ("$") when producing symbol names.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: None

Windows: /Quse-msasm-symbols

Arguments

None

Default

OFF
This option tells the compiler to use a dollar sign ("$") when producing symbol names.
Use this option if you require symbols in your .asm files to contain characters that are accepted by the MS assembler.

Alternate Options

None

V (Linux* and Mac OS* X)

Displays the compiler version information.

IDE Equivalent

Windows: None
Linux: General > Show Startup Banner
Mac OS X: General > Show Startup Banner

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: −V
Windows: /Qv

Arguments

None

Default

OFF

The compiler version information is not displayed.

Description

This option displays the startup banner, which contains the following compiler version information:

- **ID**: unique identification number for the compiler
- **x.y.z**: version of the compiler
- **years**: years for which the software is copyrighted

This option can be placed anywhere on the command line.

Alternate Options

None

**Qvc**

Specifies compatibility with Microsoft* Visual C++ or Microsoft* Visual Studio.

IDE Equivalent

None

Architectures
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Qvc7.1
/Qvc8
/Qvc9

Arguments
None

Default
varies

When the compiler is installed, it detects which version of Visual Studio is on your system. Qvc defaults to the form of the option that is compatible with that version.
When multiple versions of Visual Studio are installed, the compiler installation lets you select which version you want to use. In this case, Qvc defaults to the version you choose.

**Description**

This option specifies compatibility with Visual C++ or Visual Studio.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/Qvc7.1</td>
<td>Specifies compatibility with Microsoft* Visual Studio .NET 2003.</td>
</tr>
<tr>
<td>/Qvc8</td>
<td>Specifies compatibility with Microsoft*</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
</tbody>
</table>

Alternate Options

None

vec, Qvec

Enables or disables vectorization and transformations enabled for vectorization.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:  
-vec

-no-vec

Windows:  
/Qvec

/Qvec-

Arguments

None

Default

-vec  
Vectorization is enabled.

or/Qvec
Description
This option enables or disables vectorization and transformations enabled for vectorization.
To disable vectorization and transformations enabled for vectorization, specify -no-vec (Linux and Mac OS X) or /Qvec- (Windows).

Alternate Options
None

vec-guard-write, Qvec-guard-write
Tells the compiler to perform a conditional check in a vectorized loop.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X: -vec-guard-write
-no-vec-guard-write
Windows: /Qvec-guard-write
/Qvec-guard-write-

Arguments
None

Default
-no-vec-guard-write
or/Qvec-guard-write-
The compiler uses
default heuristics when checking vectorized loops.

Description
This option tells the compiler to perform a conditional check in a vectorized loop. This checking avoids unnecessary stores and may improve performance.

Alternate Options
None

`vec-report, Qvec-report`
Controls the diagnostic information reported by the vectorizer.

IDE Equivalent
Windows: None
Linux: **Compilation Diagnostics > Vectorizer Report**
Mac OS X: **Diagnostics > Vectorizer Diagnostic Report**

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X: `-vec-report[n]`
Windows: `/Qvec-report[n]`

Arguments
Is a value denoting which diagnostic messages to report. Possible values are:

0 Tells the vectorizer to report no diagnostic information.

1 Tells the vectorizer to report on vectorized loops.

2 Tells the vectorizer to report on vectorized and non-vectorized loops.

3 Tells the vectorizer to report on vectorized and non-vectorized
loops and any proven or assumed data dependences.

4 Tells the vectorizer to report on non-vectorized loops.

5 Tells the vectorizer to report on non-vectorized loops and the reason why they were not vectorized.

**Default**

-vec-report1
or/Qvec-report1

If the vectorizer has been enabled, it reports diagnostics on vectorized loops.
This option controls the diagnostic information reported by the vectorizer. The vectorizer report is sent to stdout.
If you do not specify \( n \), it is the same as specifying \(-\text{vec-report}1\) (Linux and Mac OS X) or \(/Q\text{vec-report}1\) (Windows).
The vectorizer is enabled when certain compiler options are specified, such as option \(-\text{ax}\) or \(-x\) (Linux and Mac OS X), option \(/Q\text{ax}\) or \(/Qx\) (Windows), option \(/\text{arch}:\text{SSE}\) or \(/\text{arch}:\text{SSE2}\) (Windows), and option fast.

Alternate Options
None

**wd, Qwd**
Disables a soft diagnostic. This is a deprecated option.

IDE Equivalent
Windows: **Advanced > Disable Specific Warnings**

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: \(-\text{wd}Ln[,Ln,\ldots]\)
Windows: \(/Q\text{wd}Ln[,Ln,\ldots]\)

Arguments

\( Ln \)
Is the number of the diagnostic to
This option disables the soft diagnostic that corresponds to the specified number. If you specify more than one \textit{Ln}, each \textit{Ln} must be separated by a comma.

**Alternate Options**

None

**\texttt{we, Qwe}**

Changes a soft diagnostic to an error. This is a deprecated option.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: \texttt{--weLn[,Ln,...]}

Windows: \texttt{/QweLn[,Ln,...]}

**Arguments**
$Ln$

Is the number of the diagnostic to be changed.

Default

OFF

The compiler returns soft diagnostics as usual.

Description

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to an error.

If you specify more than one $Ln$, each $Ln$ must be separated by a comma.

Alternate Options

None

$wn$, $Qwn$

Controls the number of errors displayed before compilation stops. This is a deprecated option.

IDE Equivalent

Windows: Diagnostics > Error Limit
Linux: Compilation Diagnostics > Set Error Limit
Mac OS X: Diagnostics > Error Limit
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -wnn
Windows: /Qwnn

Arguments

\( n \)

Is the number of errors to display.

Default

100

The compiler displays a maximum of 100 errors before aborting compilation.

Description

This option controls the number of errors displayed before compilation stops.

Alternate Options
None

**wo, Qwo**

Tells the compiler to issue one or more diagnostic messages only once. This is a deprecated option.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: \(-woLn[,Ln,...]\)

Windows: \(/QwoLn[,Ln,...]\)

**Arguments**

\(Ln\)  

Is the number of the diagnostic.

**Default**

OFF

**Description**

Specifies the ID number of one or more messages. If you specify more than one \(Ln\), each \(Ln\) must be separated by a comma.

**Alternate Options**

None

956
wr, Qwr

Changes a soft diagnostic to an remark. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-wr Ln[,Ln,...]`

Windows: `/Qwr Ln[,Ln,...]`

Arguments

$Ln$

Is the number of the diagnostic to be changed.

Default

OFF

The compiler returns soft diagnostics as usual.
This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to a remark. If you specify more than one $Ln$, each $Ln$ must be separated by a comma.

Alternate Options
None

ww, Qww
Changes a soft diagnostic to a warning. This is a deprecated option.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: `-wwLn [,Ln,...]`
Windows: `/QwwLn [,Ln,...]`

Arguments

$Ln$ Is the number of the diagnostic to be changed.

Default
OFF The
compiler returns soft diagnostics as usual.

Description

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to an warning. If you specify more than one Ln, each Ln must be separated by a comma.

Alternate Options

None

x, Qx

Tells the compiler to generate optimized code specialized for the Intel processor that executes your program.

IDE Equivalent

Windows: Code Generation > Intel Processor-Specific Optimization
Linux: Code Generation > Intel Processor-Specific Optimization
Mac OS X: Code Generation > Intel Processor-Specific Optimization

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -xprocessor
Windows: /Qxprocessor

Arguments
processor

Indicates the processor for which code is generated. Many of the following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE).
Possible values are:

Host Can generate instructions for the highest instruction set and processor available on the compilation host.

SSE4.2 Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator,
Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.

**SSE4.1**

Can generate

Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors.

Can generate

Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k next generation Intel® Core™ microarchitecture.

This replaces value S, which is deprecated.

**SSE3_ATOM** Can generate
MOVBE instructions for Intel processors and it can optimize for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology.

SSSE3 can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. This replaces value T, which is deprecated.

SSE3 can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors.
and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. This replaces value P, which is deprecated.

**SSE2**

Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel Pentium® 4 processors, Intel Pentium® M processors, and Intel® Xeon® processors with Intel® SSE2. This value is not available on Mac OS X systems. This replaces value N, which is deprecated.
Default

Windows systems: `/arch:SSE2`
Linux systems: `-msse2`
Mac OS X systems using IA-32 architecture: `SSE3`
Mac OS X systems using Intel® 64 architecture: `SSSE3`

For more information on the default values, see Arguments above, option `m` (Linux and Mac OS X) and option `arch` (Windows).

Description

This option tells the compiler to generate optimized code specialized for the Intel processor that executes your program. It also enables optimizations in addition to Intel processor-specific optimizations. The specialized code generated by this option may run only on a subset of Intel processors.

This option can enable optimizations depending on the argument specified. For example, it may enable Intel® Streaming SIMD Extensions 4 (Intel® SSE4), Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Intel® Streaming SIMD Extensions 2 (Intel® SSE2), or Intel® Streaming SIMD Extensions (Intel® SSE) instructions.

The binaries produced by these values will run on Intel processors that support all of the features for the targeted processor. For example, binaries produced with SSE3 will run on an Intel® Core™ 2 Duo processor, because that processor completely supports all of the capabilities of the Intel® Pentium® 4 processor, which the SSE3 value targets. Specifying the SSSE3 value has the potential of using more features and optimizations available to the Intel® Core™ 2 Duo processor.

Do not use processor values to create binaries that will execute on a processor that is not compatible with the targeted processor. The resulting program may fail with an illegal instruction exception or display other unexpected behavior. For
example, binaries produced with SSE3 may produce code that will not run on Intel® Pentium® III processors or earlier processors that do not support SSE2 instructions.

Compiling the main program with any of the processor values produces binaries that display a fatal run-time error if they are executed on unsupported processors. For more information, see Optimizing Applications.

If you specify more than one processor value, code is generated for only the highest-performing processor specified. The highest-performing to lowest-performing processor values are: SSE4.2, SSE4.1, SSSE3, SSE3, SSE2. Note that processor value SSE3_ATOM does not fit within this group.

Compiler options m and arch produce binaries that should run on processors not made by Intel that implement the same capabilities as the corresponding Intel processors.

Previous value O is deprecated and has been replaced by option -msse3 (Linux and Mac OS X) and option /arch:SSE3 (Windows).

Previous values W and K are deprecated. The details on replacements are as follows:

- **Mac OS X systems:** On these systems, there is no exact replacement for W or K. You can upgrade to the default option -msse3 (IA-32 architecture) or option -msse3 (Intel® 64 architecture).

- **Windows and Linux systems:** The replacement for W is -msse2 (Linux) or /arch:SSE2 (Windows). There is no exact replacement for K. However, on Windows systems, /QxK is interpreted as /arch:IA32; on Linux systems, -xK is interpreted as -mia32. You can also do one of the following:
  - Upgrade to option -msse2 (Linux) or option /arch:SSE2 (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors
  - Specify the two option combination -mia32 -axSSE2 (Linux) or /arch:IA32 /QaxSSE2 (Windows). This combination will produce an
executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path. The \texttt{-x} and \texttt{/Qx} options enable additional optimizations not enabled with option \texttt{-m} or option \texttt{/arch}.

Alternate Options

None

See Also

\texttt{ax, Qax} compiler option
\texttt{m} compiler option
\texttt{arch} compiler option

\texttt{rcd, Qrcd}

Enables fast float-to-integer conversions.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: \texttt{-rcd}
Windows: \texttt{/Qrcd}

Arguments

None

Default

OFF Floating-
point values are truncated when a conversion to an integer is involved. On Windows, this is the same as specifying `/QIfist-`.

**Description**

This option enables fast float-to-integer conversions. It can improve the performance of code that requires floating-point-to-integer conversions. The system default floating-point rounding mode is round-to-nearest. However, the C language requires floating-point values to be truncated when a conversion to an integer is involved. To do this, the compiler must change the rounding mode to truncation before each floating-point-to-integer conversion and change it back afterwards.

This option disables the change to truncation of the rounding mode for all floating-point calculations, including floating point-to-integer conversions. This option can improve performance, but floating-point conversions to integer will not conform to C semantics.

**Alternate Options**

Linux and Mac OS X: None
Windows: /QIfist

rct, Qrct

Sets the internal FPU rounding control to Truncate.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -rct
Windows: /Qrct

Arguments

None

Default

OFF

The compiler uses the default setting for the FPU rounding control.

Description

This option sets the internal FPU rounding control to Truncate.
Alternate Options

None

reserve-kernel-regs

Reserves registers f12-f15 and f32-f127 for use by the kernel.

IDE Equivalent

None

Architectures

IA-64 architecture

Syntax

Linux: -reserve-kernel-regs
Mac OS X: None
Windows: None

Arguments

None

Default

OFF  The compiler can use registers f12-f15 and f32-f127.

Description

This option reserves registers f12-f15 and f32-f127 for use by the kernel.
Alternate Options

None

restrict, Qrestrict

Determines whether pointer disambiguation is enabled with the restrict qualifier.

IDE Equivalent

Windows: **Language > Recognize Restrict Keyword**
Linux: **Language > Recognize Restrict Keyword**
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-restrict`
    `-no-restrict`

Windows: `/Qrestrict`
    `/Qrestrict-`

Arguments

None

Default

`-no-restrict` or `/Qrestrict-`  
Pointers are not qualified with the restrict keyword.
Description

This option determines whether pointer disambiguation is enabled with the restrict qualifier. Options \texttt{-restrict} (Linux and Mac OS X) and \texttt{/Qrestrict} (Windows) enable the recognition of the restrict keyword as defined by the ANSI standard.

By qualifying a pointer with the restrict keyword, you assert that an object accessed by the pointer is only accessed by that pointer in the given scope. You should use the restrict keyword only when this is true. When the assertion is true, the restrict option will have no effect on program correctness, but may allow better optimization.

Alternate Options

None

See Also

\texttt{Qc99} compiler option

RTC

Enables checking for certain run-time conditions.

IDE Equivalent

Windows: \textbf{Code Generation > Basic Runtime Checks / Smaller Type Check}

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None

Windows: \texttt{/RTC\textit{option}}
Arguments

option

Specifies the condition to check. Possible values are 1, s, u, or c.

Default

OFF

No checking is performed for these run-time conditions.

Description

This option enables checking for certain run-time conditions. Using the /RTC option sets __MSVC_RUNTIME_CHECKS = 1.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/RTC1</td>
<td>This is the same as specifying /RTCsu.</td>
</tr>
<tr>
<td>/RTCs</td>
<td>Enables run-time checks of the stack frame.</td>
</tr>
<tr>
<td>/RTCu</td>
<td>Enables run-time checks for uninitialized variables.</td>
</tr>
<tr>
<td>/RTCc</td>
<td>Enables checks for converting to smaller types.</td>
</tr>
</tbody>
</table>
Alternate Options

None

S

Causes the compiler to compile to an assembly file only and not link.

IDE Equivalent

Windows: None
Linux: **Output Files > Generate Assembler Source File**
Mac OS X: **Output Files > Generate Assembler Source File**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-S\)
Windows: \(/S\)

Arguments

None

Default

OFF Normal compilation and linking occur.

Description

This option causes the compiler to compile to an assembly file only and not link. On Linux and Mac OS X systems, the assembly file name has a .s suffix. On Windows systems, the assembly file name has an .asm suffix.
Alternate Options

Linux and Mac OS X: None
Windows: /Fa

See Also

Fa compiler option

save-temps, Qsave-temps

Tells the compiler to save intermediate files created during compilation.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -save-temps
- no-save-temps
Windows: /Qsave-temps
 /Qsave-temps-

Arguments

None

Default

Linux and Mac OS X: -no-save-temps
Windows: .obj files are saved

On Linux and Mac OS X systems, the compiler
This option tells the compiler to save intermediate files created during compilation. The names of the files saved are based on the name of the source file; the files are saved in the current working directory.

If `-save-temps` or `/Qsave-temps` is specified, the following occurs:

- **The object `.o` file (Linux and Mac OS X) or `.obj` file (Windows) is saved.**
- **The assembler `.s` file (Linux and Mac OS X) or `.asm` file (Windows) is saved if you specified `-use-asm` (Linux or Mac OS X) or `/Quse-asm` (Windows).**

If `-no-save-temps` is specified on Linux or Mac OS X systems, the following occurs:

- **The `.o` file is put into `/tmp` and deleted after calling `ld`.**
- The preprocessed file is not saved after it has been used by the compiler.

If /Qsave-temps- is specified on Windows systems, the following occurs:

- The .obj file is not saved after the linker step.
- The preprocessed file is not saved after it has been used by the compiler.

**Note**

This option only saves intermediate files that are normally created during compilation.

**Alternate Options**

None

**Example**

If you compile program `my_foo.c` on a Linux or Mac OS X system and you specify option `-save-temps` and option `-use-asm`, the compilation will produce files `my_foo.o` and `my_foo.s`.

If you compile program `my_foo.c` on a Windows system and you specify option `/Qsave-temps` and option `/Quse-asm`, the compilation will produce files `my_foo.o` and `my_foo.asm`.

**scalar-rep, Qscalar-rep**

Enables scalar replacement performed during loop transformation.

**IDE Equivalent**

None

**Architectures**

IA-32 architecture

**Syntax**
Linux and Mac OS X:  
- scalar-rep
  - no-scalar-rep
Windows:  
/Qscalar-rep
/Qscalar-rep-

Arguments

None

Default

-no-scalar-rep  
Scalar replacement is not performed during loop transformation.

or/Qscalar-rep-

Description

This option enables scalar replacement performed during loop transformation. To use this option, you must also specify O3.

Alternate Options

None

See Also

O compiler option

shared

Tells the compiler to produce a dynamic shared object instead of an executable.

IDE Equivalent

None

Architectures
Syntax

Linux: \(-\text{shared}\)
Mac OS X: None
Windows: None

Arguments

None

Default

OFF

The compiler produces an executable.

Description

This option tells the compiler to produce a dynamic shared object (DSO) instead of an executable. This includes linking in all libraries dynamically and passing \(-\text{shared}\) to the linker.

On systems using IA-32 architecture and Intel® 64 architecture, you must specify option \texttt{fpic} for the compilation of each object file you want to include in the shared library.

Alternate Options

None

See Also

\texttt{dynamiclib} compiler option
\texttt{fpic} compiler option
Xlinker compiler option

**shared-intel**

Causes Intel-provided libraries to be linked in dynamically.

**IDE Equivalent**

Windows: None
Linux: None
Mac OS X: **Libraries > Intel Runtime Libraries**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-shared-intel`
Windows: None

**Arguments**

None

**Default**

OFF

Intel libraries are linked in statically, with the exception of libguide on Linux* and Mac OS* X systems,
where it is linked in dynamically.

Description

This option causes Intel-provided libraries to be linked in dynamically. It is the opposite of `-static-intel`.

Note

On Mac OS X systems, when you set "Intel Runtime Libraries" to "Dynamic", you must also set the DYLD_LIBRARY_PATH environment variable within Xcode or an error will be displayed.

Alternate Options

Linux and Mac OS X: `-i-dynamic` (this is a deprecated option)
Windows: None

See Also

`static-intel` compiler option

`shared-libgcc`

Links the GNU `libgcc` library dynamically.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: `–shared-libgcc`
Mac OS X: None
Windows: None

Arguments

None

Default

-shared-libgcc

The compiler links the libgcc library dynamically.

Description

This option links the GNU libgcc library dynamically. It is the opposite of option static-libgcc.

This option is useful when you want to override the default behavior of the static option, which causes all libraries to be linked statically.

Alternate Options

None

See Also

static-libgcc compiler option

showIncludes

Tells the compiler to display a list of the include files.

IDE Equivalent

Windows: Advanced > Show Includes
Linux: None  
Mac OS X: None  

Architectures  
IA-32, Intel® 64, IA-64 architectures  

Syntax  
Linux and Mac OS X: None  
Windows: /showIncludes  

Arguments  
None  

Default  
OFF  
The compiler does not display a list of the include files. Nested include files (files that are included from the files that you include) are also displayed.  

Description  
This option tells the compiler to display a list of the include files. Nested include files (files that are included from the files that you include) are also displayed.  

Alternate Options  
None  

982
**sox, Qsox**

Tells the compiler to save the compiler options and version number in the executable.

**IDE Equivalent**
None

**Architectures**
IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-sox`
- `-no-sox`

**Windows:** `/Qsox`
- `/Qsox-`

**Arguments**
None

**Default**
- `-no-sox`

The compiler does not save the compiler options and version number in the

Description

This option tells the compiler to save the compiler options and version number in the executable. The size of the executable on disk is increased slightly by the inclusion of these information strings.

This option forces the compiler to embed in each object file or assembly output a string that contains information about the compiler version and compilation options for each source file that has been compiled. When you link the object files into an executable file, the linker places each of the information strings into the header of the executable. It is then possible to use a tool, such as a strings utility, to determine what options were used to build the executable file.

If `-no-sox` or `/Qsox-` is specified, this extra information is not put into the object or assembly output generated by the compiler.

Alternate Options

None

**static**

Prevents linking with shared libraries.

IDE Equivalent

Windows: None
Linux: **Libraries > Link with static libraries**
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: `-static`
Mac OS X: None
Windows: None

Arguments
None

Default
OFF

Description
This option prevents linking with shared libraries. It causes the executable to link all libraries statically.

Alternate Options
None

staticlib

Invokes the libtool command to generate static libraries.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux: None
Mac OS X: -staticlib
Windows: None

Arguments
None

Default
OFF

The compiler produces an executable.

Description
This option invokes the libtool command to generate static libraries. When passed this option, the compiler uses the libtool command to produce a static library instead of an executable when linking. To build dynamic libraries, you should specify option -dynamiclib or libtool -dynamic <objects>.

Alternate Options
None

See Also
dynamiclib compiler option
static-intel
Causes Intel-provided libraries to be linked in statically.

IDE Equivalent

986
Windows: None
Linux: None
Mac OS X: Libraries > Intel Runtime Libraries

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -static-intel
Windows: None

Arguments
None

Default
OFF  Intel libraries are linked in statically, with the exception of libguide, which is linked in dynamically.

Description
This option causes Intel-provided libraries to be linked in statically. It is the opposite of -shared-intel.

Alternate Options
Linux and Mac OS X: i-static (this is a deprecated option)
Windows: None

See Also

shared-intel compiler option

static-libgcc

Links the GNU libgcc library statically.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux: -static-libgcc
Mac OS X: None
Windows: None

Arguments

None

Default

OFF DEFAULT_DESC

Description

This option links the GNU libgcc library statically. It is the opposite of option libgcc.
This option is useful when you want to override the default behavior of the libgcc option, which causes all libraries to be linked statically.
Alternate Options

None

See Also

shared-libgcc compiler option

std, Qstd

Tells the compiler to conform to a specific language standard.

IDE Equivalent

Windows: Language > Enable C++0x Support
Linux: Language > ANSI Conformance
Mac OS X: Language > C ANSI Conformance

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -std=\text{val}
Windows: /Qstd:\text{val}

Arguments

\textit{val}

Possible values are:

\begin{itemize}
  \item \texttt{c89} Conforms to the ISO/IEC 9899:1990 International Standard.
  \item \texttt{c99} Conforms to The ISO/IEC

gnu89 Conforms to ISO C90 plus GNU* extensions.

gnu++98 Conforms to the 1998 ISO C++ standard plus GNU extensions.

c++0x Enable support for the following C++0x features:

• Empty macro arguments
• Variadic macros
• Type long long
• Trailing comma in enum definition
• Concatenation of mixed-width string literals
• Extended friend
declarations
• Use of ">>" to close two template argument lists
• Relaxed rules for use of "typename"
• Relaxed rules for disambiguation using the "template" keyword
• Copy constructor does not need to be callable on direct reference
• Binding to class rvalue
• "extern template" to suppress instantiation of an entity
• "auto" type specifier
• decltype
operator
• static_assert
• compliant
  __func__
• lambda
  expressions

Default

-std=gnu89 (default for C)  Conforms to ISO C90 plus GNU extensions.

-std=gnu++98 (default for C++)  Conforms to the 1998 ISO C++ standard plus GNU* extensions.


Description
Tells the compiler to conform to a specific language standard.

Alternate Options
None
strict-ansi

Tells the compiler to implement strict ANSI conformance dialect.

IDE Equivalent

Windows: None
Linux: Language > ANSI Conformance
Mac OS X: Language > C ANSI Conformance

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -strict-ansi
Windows: None

Arguments

None

Default

OFF

The compiler conforms to default standards.

Description

This option tells the compiler to implement strict ANSI conformance dialect. If you need to be compatible with gcc, use the -ansi option.

This option sets option fmath-errno.

Alternate Options
None

\textbf{T}

Tells the linker to read link commands from a file.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux: \texttt{-Tfile}

Mac OS X: None

Windows: None

**Arguments**

\texttt{file} \hspace{2cm} \text{Is the name of the file.}

**Default**

OFF \hspace{2cm} The linker does not read link commands from a file.

**Description**

This option tells the linker to read link commands from a file.
Alternate Options

None

**Tc**

Tells the compiler to process a file as a C source file.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None
Windows: /Tc file

**Arguments**

*file*  
Is the file name to be processed as a C source file.

**Default**

OFF  
The compiler uses default
rules for determining whether a file is a C source file.

Description

This option tells the compiler to process a file as a C source file.

Alternate Options

None

See Also

TC compiler option  
Tp compiler option

TC

Tells the compiler to process all source or unrecognized file types as C source files.

IDE Equivalent

Windows: Advanced > Compile As

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None

Windows: /TC
Arguments

None

Default

OFF

The compiler uses default rules for determining whether a file is a C source file.

Description

This option tells the compiler to process all source or unrecognized file types as C source files.

Alternate Options

None

See Also

TP compiler option

Tc compiler option

tcheck, Qtcheck

Enables analysis of threaded applications.

IDE Equivalent

None

Architectures
Syntax

Linux: -tcheck
Mac OS X: None
Windows: /Qtcheck

Arguments

None

Default

OFF

Threaded applications are not instrumented by the compiler for analysis by Intel® Thread Checker.

Description

This option enables analysis of threaded applications.
To use this option, you must have Intel® Thread Checker installed, which is one of the Intel® Threading Analysis Tools. If you do not have this tool installed, the compilation will fail. Remove the -tcheck (Linux) or /Qtcheck (Windows) option from the command line and recompile.
For more information about Intel® Thread Checker (including an evaluation copy), open the page associated with threading tools at Intel® Software Development Products.

Alternate Options

None

**tcollect, Qtcollect**

Inserts instrumentation probes calling the Intel® Trace Collector API.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux: \[-tcollect[lib]\]

Mac OS X: None

Windows: /Qtcollect[:lib]

**Arguments**

*lib*

Is one of the Intel® Trace Collector libraries; for example, VT,
VTcs, VTmc, or VTfs. If you do not specify the library, the default library is VT.

**Default**

**OFF** Instrumentation probes are not inserted into compiled applications.

**Description**

This option inserts instrumentation probes calling the Intel® Trace Collector API. To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

This option provides a flexible and convenient way of instrumenting functions of a compiled application. For every function, the entry and exit points are instrumented at compile time to let the Intel® Trace Collector record functions beyond the default MPI calls. For non-MPI applications (for example, threaded or serial), you must ensure that the Intel® Trace Collector is properly initialized (VT_initialize/VT_init).

**Caution**
Be careful with full instrumentation because this feature can produce very large trace files.
For more details, see the Intel® Trace Collector User Guide.

**Alternate Options**

None

**See Also**

tcollect-filter, Qtcollect-filter compiler option

tcollect-filter, Qtcollect-filter

Lets you enable or disable the instrumentation of specified functions.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux: 

```
-tcollect-filter file
```

Mac OS X: 

None

Windows: 

```
/Qtcollect-filter:file
```

**Arguments**

`file`

Is a configuration file that lists filters, one per line.

Each filter
consists of a regular expression string and a switch. Strings with leading or trailing white spaces must be quoted. Other strings do not have to be quoted. The switch value can be ON, on, OFF, or off.

Default

OFF

Functions are not instrumented. However, if option -tcollect (Linux) or /Qtcollect (Windows) is specified, the
filter setting
is ".* ON"
and all
functions get
instrumented.

Description
This option lets you enable or disable the instrumentation of specified functions. During instrumentation, the regular expressions in the file are matched against the function names. The switch specifies whether matching functions are to be instrumented or not. Multiple filters are evaluated from top to bottom with increasing precedence.

The names of the functions to match against are formatted as follows:

- **C++ function names are demangled and the C++ class hierarchy is used.**
  Function parameters are stripped to keep the function names shorter.
- **The source file name is followed by a colon-separated function name.**
  Source file names should contain the full path, if available. For example:
  ```
  /home/joe/src/foo.c:FOO_bar
  ```
- **Classes and function names are separated by double colons.** For example:
  ```
  /home/joe/src/foo.cpp:app::foo::bar
  ```

You can use option `-opt-report` (Linux) or `/Qopt-report` (Windows) to get a full list of file and function names that the compiler recognizes from the compilation unit. This list can be used as the basis for filtering in the configuration file.

To use this option, you must have the Intel® Trace Collector installed and set up through one of its set-up scripts. This tool is a component of the Intel® Trace Analyzer and Collector.

For more details, see the Intel® Trace Collector User Guide.

Alternate Options

None
Consider the following filters in a configuration file:

```
'.*' OFF '.*vector.*' ON
```

The above will cause instrumentation of only those functions having the string 'vector' in their names. No other function will be instrumented. Note that reversing the order of the two lines will prevent instrumentation of all functions.

To get a list of the file or routine strings that can be matched by the regular expression filters, generate an optimization report with tcollect information. For example:

```
Windows OS: icl /Qtcollect /Qopt-report /Qopt-report-phase tcollect
Linux OS: icc -tcollect -opt-report -opt-report-phase tcollect
```

See Also

tcollect, Qtcollect compiler option

Tp

Tells the compiler to process a file as a C++ source file.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Tpfile

Arguments

file

Is the file name to be processed
as a C++ source file.

**Default**

OFF

The compiler uses default rules for determining whether a file is a C++ source file.

**Description**

This option tells the compiler to process a file as a C++ source file.

**Alternate Options**

None

**See Also**

TP compiler option

**Kc++, TP**

Tells the compiler to process all source or unrecognized file types as C++ source files.

This option is deprecated.
IDE Equivalent

Windows: **Advanced > Compile As**
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-Kc++`
Windows: `/TP`

Arguments

None

Default

**OFF**

The compiler uses default rules for determining whether a file is a C++ source file.

Description

This option tells the compiler to process all source or unrecognized file types as C++ source files.
Alternate Options
None

tprofile, Qtprofile
Generates instrumentation to analyze multi-threading performance.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux: -tprofile
Mac OS X: None
Windows: /Qtprofile

Arguments
None

Default
OFF Instrumentation is not generated by the compiler for analysis by Intel® Thread Profiler.

Description
This option generates instrumentation to analyze multi-threading performance.
To use this option, you must have Intel® Thread Profiler installed, which is one of the Intel® Threading Analysis Tools. If you do not have this tool installed, the compilation will fail. Remove the `-tprofile` (Linux) or `/Qtprofile` (Windows) option from the command line and recompile.

For more information about Intel® Thread Profiler (including an evaluation copy), open the page associated with threading tools at Intel® Software Development Products.

Alternate Options

None

**traceback**

Tells the compiler to generate extra information in the object file to provide source file traceback information when a severe error occurs at run time.

**IDE Equivalent**

Windows: None

Linux: **Runtime > Generate Traceback Information**

Mac OS X: **Runtime > Generate Traceback Information**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-traceback -notraceback`

Windows: `/traceback /notraceback`

**Arguments**

None
Default

notraceback

No extra information is generated in the object file to produce traceback information.

Description

This option tells the compiler to generate extra information in the object file to provide source file traceback information when a severe error occurs at run time. This is intended for use with C code that is to be linked into a Fortran program. When the severe error occurs, source file, routine name, and line number correlation information is displayed along with call stack hexadecimal addresses (program counter trace).

Note that when a severe error occurs, advanced users can also locate the cause of the error using a map file and the hexadecimal addresses of the stack displayed when the error occurs.

This option increases the size of the executable program, but has no impact on run-time execution speeds.

It functions independently of the debug option.

On Windows systems, traceback sets the /Oy- option, which forces the compiler to use EBP as the stack frame pointer.

On Windows systems, the linker places the traceback information in the executable image, in a section named ".trace". To see which sections are in an image, use the command:

```link -dump -summary your_app_name.exe```

To see more detailed information, use the command:
On Linux systems, to display the section headers in the image (including the header for the .trace section, if any), use the command:

```
link -dump -headers your_app_name.exe
```

On Mac OS X systems, to display the section headers in the image, use the command:

```
objdump -h your_app_name.exe
```

**Alternate Options**

None

**u (Linux*)**

Tells the compiler the specified symbol is undefined.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  
```
-u symbol
```

Windows:  
```
```

**Arguments**

None

**Default**

OFF  
Standard rules are in effect for
variables.

Description

This option tells the compiler the specified symbol is undefined.

Alternate Options

None

u (Windows*)

Disables all predefined macros and assertions.

IDE Equivalent

Windows: Advanced > Undefine All Preprocessor Definitions

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /u

Arguments

None

Default

OFF

Defined preprocessor values are in effect until they are
undefined.

Description
This option disables all predefined macros and assertions.

Alternate Options
/QA

U
Undefines any definition currently in effect for the specified macro.

IDE Equivalent
Windows: Advanced > Undefine Preprocessor Definitions
Linux: Preprocessor > Undefine Preprocessor Definitions
Mac OS X: Preprocessor > Undefine Preprocessor Definitions

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: \texttt{-Uname}
Windows: \texttt{/Uname}

Arguments
\texttt{name} \quad \text{Is the name of the macro to be undefined.}

Default
1012
OFF

Macro definitions are in effect until they are undefined.

Description

This option undefines any definition currently in effect for the specified macro. It is equivalent to a #undef preprocessing directive. On Windows systems, use the /u option to undefine all previously defined preprocessor values.

Alternate Options

None

See Also

Building Applications: About Preprocessor Options

unroll, Qunroll

Tells the compiler the maximum number of times to unroll loops.

IDE Equivalent

Windows: Optimization > Loop Unrolling
Linux: Optimization > Loop Unroll Count
Mac OS X: Optimization > Loop Unrolling

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: \texttt{-unroll\[=n\]}
Windows: \texttt{/Qunroll\[:n\]}

**Arguments**

\(n\)  

Is the maximum number of times a loop can be unrolled. To disable loop unrolling, specify 0. On systems using IA-64 architecture, you can only specify a value of 0.

**Default**

\texttt{-unroll}  
\texttt{or/Qunroll}

The compiler uses default heuristics when unrolling loops.
Description

This option tells the compiler the maximum number of times to unroll loops. If you do not specify $n$, the optimizer determines how many times loops can be unrolled.

Alternate Options

Linux and Mac OS X: -funroll-loops
Windows: None

unroll-aggressive, Qunroll-aggressive

Determines whether the compiler uses more aggressive unrolling for certain loops.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -unroll-aggressive
               -no-unroll-aggressive
Windows:       /Qunroll-aggressive
               /Qunroll-aggressive-

Arguments

None

Default

-no-unroll-aggressive        The
or/Qunroll-aggressive-     compiler
uses default heuristics when unrolling loops.

Description

This option determines whether the compiler uses more aggressive unrolling for certain loops. The positive form of the option may improve performance. On IA-32 architecture and Intel® 64 architecture, this option enables aggressive, complete unrolling for loops with small constant trip counts. On IA-64 architecture, this option enables additional complete unrolling for loops that have multiple exits or outer loops that have a small constant trip count.

Alternate Options

None

use-asm, Quse-asm

Tells the compiler to produce objects through the assembler.

IDE Equivalent

None

Architectures

-use-asm: IA-32 architecture, Intel® 64 architecture, IA-64 architecture
/Quse-asm: IA-64 architecture

Syntax

Linux and Mac OS X: -use-asm

-no-use-asm
Windows: /Quse-asm
         /Quse-asm-

Arguments
None

Default
-no-use-asm
or /Quse-asm-
The compiler produces objects directly.

Description
This option tells the compiler to produce objects through the assembler.

Alternate Options
None

use-msasm
Tells the compiler to accept the Microsoft* MASM-style inlined assembly format.

IDE Equivalent
None

Architectures
IA-32, Intel® 64 architectures

Syntax
Linux and Mac OS X: -use-msasm
Windows: None
Arguments

None

Default

OFF

The compiler accepts the GNU-style inlined assembly format.

Description

This option tells the compiler to accept the Microsoft MASM-style inlined assembly format instead of the GNU-style format.

Alternate Options

None

v

Specifies that driver tool commands should be displayed and executed.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -v[\texttt{\textit{file}}]
Windows: None

Arguments

file Is the name of a file.

Default

OFF No tool commands are shown.

Description

This option specifies that driver tool commands should be displayed and executed.
If you use this option without specifying a file name, the compiler displays only the version of the compiler.

Alternate Options

None

See Also

dryrun compiler option

V (Linux* and Mac OS* X)

Displays the compiler version information.

IDE Equivalent

Windows: None
Linux: General > Show Startup Banner
Mac OS X: **General > Show Startup Banner**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-v`

Windows: `/Qv`

**Arguments**

None

**Default**

OFF

The compiler version information is not displayed.

**Description**

This option displays the startup banner, which contains the following compiler version information:

- **ID**: unique identification number for the compiler
- **x.y.z**: version of the compiler
- **years**: years for which the software is copyrighted

This option can be placed anywhere on the command line.

**Alternate Options**

None
**V (Windows*)**

Places the text string specified into the object file being generated by the compiler.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None

Windows: `/Vstring`

**Arguments**

`string`  

Is the text string to go into the object file.

**Default**

OFF  

No text string is placed
Intel® C++ Compiler User and Reference Guides
in the
object
file.

Description

Places the text string specified into the object file (.obj) being generated by the
compiler.
This option places the text string specified into the object file (.obj) being
generated by the compiler. The string also gets propagated into the executable
file.
For example, this option is useful if you want to place the version number or
copyright information into the object and executable.
If the string contains a space or tab, the string must be enclosed by double
quotation marks ("). A backslash (\) must precede any double quotation marks
contained within the string.
Alternate Options

None
vd
Disable or enable hidden vtordisp field in C++ objects.
IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: None
Windows:

1022

/vdval


Arguments

val

Possible values are:

0 disables hidden vtordisp field in C++ objects.

1 enables hidden vtordisp field in C++ objects.

Default

/vd1

The compiler enables hidden vtordisp field in C++ objects.

Description

This option disables or enables hidden vtordisp field in C++ objects.

Alternate Options
vec, Qvec

Enables or disables vectorization and transformations enabled for vectorization.

IDE Equivalent

None

Architectures

IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X:  -vec
  -no-vec
Windows:  /Qvec
  /Qvec-

Arguments

None

Default

-vec or /Qvec  Vectorization is enabled.

Description

This option enables or disables vectorization and transformations enabled for vectorization.

To disable vectorization and transformations enabled for vectorization, specify -no-vec (Linux and Mac OS X) or /Qvec- (Windows).

Alternate Options
None

**vec-guard-write, Qvec-guard-write**

Tells the compiler to perform a conditional check in a vectorized loop.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X:  
- vec-guard-write
  - no-vec-guard-write

Windows:  
/ Qvec-guard-write
  / Qvec-guard-write-

**Arguments**

None

**Default**

-no-vec-guard-write  
or/Qvec-guard-write-

The compiler uses default heuristics when checking vectorized loops.

**Description**


This option tells the compiler to perform a conditional check in a vectorized loop. This checking avoids unnecessary stores and may improve performance.

Alternate Options
None

**vec-report, Qvec-report**

Controls the diagnostic information reported by the vectorizer.

**IDE Equivalent**

Windows: None
Linux: **Compilation Diagnostics > Vectorizer Report**
Mac OS X: **Diagnostics > Vectorizer Diagnostic Report**

**Architectures**

IA-32, Intel® 64 architectures

**Syntax**

Linux and Mac OS X: **-vec-report[n]**
Windows: **/Qvec-report[n]**

**Arguments**

\( n \)

Is a value denoting which diagnostic messages to report. Possible values are:

- 0 Tells the vectorizer to
report no diagnostic information.

1. Tells the vectorizer to report on vectorized loops.

2. Tells the vectorizer to report on vectorized and non-vectorized loops.

3. Tells the vectorizer to report on vectorized and non-vectorized loops and any proven or assumed data dependences.

4. Tells the vectorizer to report on non-vectorized
loops.

5 Tells the vectorizer to report on non-vectorized loops and the reason why they were not vectorized.

Default

-vec-report1
or /Qvec-report1

Description

This option controls the diagnostic information reported by the vectorizer. The vectorizer report is sent to stdout.

If you do not specify \textit{n}, it is the same as specifying -vec-report1 (Linux and Mac OS X) or /Qvec-report1 (Windows).

The vectorizer is enabled when certain compiler options are specified, such as option -ax or -x (Linux and Mac OS X), option /Qax or /Qx (Windows), option /arch:SSE or /arch:SSE2 (Windows), and option fast.
Alternate Options

None

version

Display GCC-style version information.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: --version
Windows: None

Arguments

None

Default

OFF

Description

Display GCC-style version information.

Alternate Options

None

vmb

Selects the smallest representation that the compiler uses for pointers to members.
IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /vmb

Arguments

None

Default

OFF

The compiler uses default rules to represent pointers to members.

Description

This option selects the smallest representation that the compiler uses for pointers to members. Use this option if you define each class before you declare a pointer to a member of the class.

Alternate Options

None
vmg

Selects the general representation that the compiler uses for pointers to members.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /vmg

Arguments

None

Default

OFF

The compiler uses default rules to represent pointers to members.
This option selects the general representation that the compiler uses for pointers to members. Use this option if you declare a pointer to a member before you define the corresponding class.

**Alternate Options**

None

**vmm**

Enables pointers to class members with single or multiple inheritance.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None
Windows: /vmm

**Arguments**

None

**Default**

OFF

The compiler uses default rules to represent pointers
Description

This option enables pointers to class members with single or multiple inheritance. To use this option, you must also specify option /vmg.

Alternate Options

None

/vms

Enables pointers to members of single-inheritance classes.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /vms

Arguments

None

Default

OFF

The compiler uses
default
rules to represent
pointers to
members.

Description

This option enables pointers to members of single-inheritance classes. To use this option, you must also specify option /vmg.

Alternate Options

None

vmv

Enables pointers to members of any inheritance type.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /vmv

Arguments

None

Default

1034
OFF

The compiler uses default rules to represent pointers to members.

Description

This option enables pointers to members of any inheritance type. To use this option, you must also specify option /vmg.

Alternate Options

None

w

Disables all warning messages.

IDE Equivalent

Windows: None
Linux: General > Warning Level
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-w\)
Windows: \(/w\)

Arguments
None

Default
OFF

Description
This option disables all warning messages.

Alternate Options
Linux and Mac OS X: \(-w0\)
Windows: \(/w0\)

\(w, W\)

Determines which diagnostic message level is set.

IDE Equivalent
Windows: General > Warning Level
Linux: General > Warning Level
Mac OS X: General > Warning Level

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

1036
Linux and Mac OS X: \(-wn\)

Windows: \(/wn\)

**Arguments**

\(n\)

Is the diagnostic message level.

**Default**

\(n=1\)

Displays warnings and errors. DEFAULT.

\(n=0\)

Displays errors.

\(n=2\)

Displays warnings and errors. This setting is equivalent to \(n=1\).

\(n=3\)

Displays remarks, warnings, and errors. Recommended for use on production code.

\(n=4\)

Displays remarks, warnings, and errors.

**Description**

This option determines which diagnostic message level is set.

**Alternate Options**

None
w, W

Determines which diagnostic message level is set.

IDE Equivalent

Windows: General > Warning Level
Linux: General > Warning Level
Mac OS X: General > Warning Level

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-wn\)
Windows: \(/wn\)

Arguments

\(n\)

Is the diagnostic message level.

Default

\(n=1\)

Displays warnings and errors.

Description

This option determines which diagnostic message level is set.
<table>
<thead>
<tr>
<th>n</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Displays errors.</td>
</tr>
<tr>
<td>1</td>
<td>Displays warnings and errors. DEFAULT.</td>
</tr>
<tr>
<td>2</td>
<td>Displays warnings and errors. This setting is equivalent to n=1.</td>
</tr>
<tr>
<td>3</td>
<td>Displays remarks, warnings, and errors. Recommended for use on production code.</td>
</tr>
<tr>
<td>4</td>
<td>Displays remarks, warnings, and errors.</td>
</tr>
</tbody>
</table>

Alternate Options

None

**Wa**

Passes options to the assembler for processing.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-Wa,option1[,option2,...]`

**Windows:** None

**Arguments**

`option`

Is an assembler option.

This option
is not processed by the driver and is directly passed to the assembler.

Default

OFF

No options are passed to the assembler.

Description

This option passes one or more options to the assembler for processing. If the assembler is not invoked, these options are ignored.

Alternate Options

None

Wabi

Determines whether a warning is issued if generated code is not C++ ABI compliant.

IDE Equivalent

None

Architectures

1040
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
-Wabi
- Wno-abi

Windows:  None

Arguments

None

Default

-Wno-abi

No warning is issued when generated code is not C++ ABI compliant.

Description

This option determines whether a warning is issued if generated code is not C++ ABI compliant.

Alternate Options

None

Wall

Tells the compiler to display errors, warnings, and remarks.
IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Wall
Windows: /Wall

Arguments

None

Default

OFF

Default warning messages are enabled.

Description

This option tells the compiler to display errors, warnings, and remarks. On Windows, this is the same as specifying the /W4 option.

Alternate Options

None

Wbrief

Tells the compiler to display a shorter form of diagnostic output.
None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-Wbrief`
Windows: `/WL`

Arguments

None

Default

OFF

The compiler displays its normal diagnostic output.

Description

This option tells the compiler to display a shorter form of diagnostic output. In this form, the original source line is not displayed and the error message text is not wrapped when too long to fit on a single line.

Alternate Options

Linux: None
Windows: `/WL`

Wcheck

Tells the compiler to perform compile-time code checking for certain code.
IDE Equivalent

Windows: None
Linux: **Compilation Diagnostics > Allow Usage Messages**
Mac OS X: **Diagnostics > Allow Usage Messages**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-Wcheck`
Windows: `/Wcheck`

Arguments

None

Default

OFF

No compile-time code checking is performed.

Description

This option tells the compiler to perform compile-time code checking for certain code. It specifies to check for code that exhibits non-portable behavior, represents a possible unintended code sequence, or possibly affects operation of the program because of a quiet change in the ANSI C Standard.

Alternate Options

None

1044
**Wcomment**

Determines whether a warning is issued when /* appears in the middle of a /* */ comment.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  
- `Wcomment`
  - `Wno-comment`

Windows:  
None

**Arguments**

None

**Default**

- `Wno-comment`

No warning is issued when /* appears in the middle of a /* */ comment.

**Description**
This option determines whether a warning is issued when /* appears in the middle of a /* */ comment.

Alternate Options

None

_Wcontext-limit, Qcontext-limit_

Set the maximum number of template instantiation contexts shown in diagnostic.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Wcontext-limit=n
Windows: /Qcontext-limit:n

Arguments

\( n \) Number of template instantiation contexts.

Default

OFF

Description

Set maximum number of template instantiation contexts shown in diagnostic.
Alternate Options

None

wd, Qwd

Disables a soft diagnostic. This is a deprecated option.

IDE Equivalent

Windows: Advanced > Disable Specific Warnings

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \texttt{-wdLn[,Ln,...]}

Windows: \texttt{/QwdLn[,Ln,...]}

Arguments

\textit{Ln}

Is the number of the diagnostic to disable.

Default

OFF

The compiler returns soft diagnostics as usual.
Description

This option disables the soft diagnostic that corresponds to the specified number. If you specify more than one $Ln$, each $Ln$ must be separated by a comma.

Alternate Options

None

Wdeprecated

Determines whether warnings are issued for deprecated features.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  
-Wdeprecated
   -Wno-deprecated

Windows:  
None

Arguments

None

Default

-Wno-deprecated  
No warnings are issued for deprecated features.
Description

This option determines whether warnings are issued for deprecated features.

Alternate Options

None

**we, Qwe**

Changes a soft diagnostic to an error. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-weLn[,Ln,...]`
Windows: `/QweLn[,Ln,...]`

Arguments

\( Ln \quad \text{Is the number of the diagnostic to be changed.} \)

Default

OFF \quad \text{The compiler}
Description

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to an error. If you specify more than one \( Ln \), each \( Ln \) must be separated by a comma.

Alternate Options

None

\textbf{Weffc++, Qeffc++}

This option enables warnings based on certain C++ programming guidelines.

IDE Equivalent

Windows: None
Linux: \textbf{Compilation Diagnostics > Enable Warnings for Style Guideline Violations}
Mac OS X: \textbf{Diagnostics > Report Effective C++ Violations}

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-\text{Weffc++}\)
Windows: \(/\text{Qeffc++}\)

Arguments

None

1050
Default

OFF

Diagnostics are not enabled.

Description

This option enables warnings based on certain programming guidelines developed by Scott Meyers in his books on effective C++ programming. With this option, the compiler emits warnings for these guidelines:

- **Use** `const` and `inline` rather than `#define`. Note that you will only get this in user code, not system header code.
- **Use** `<iostream>` rather than `<stdio.h>`.
- **Use** `new` and `delete` rather than `malloc` and `free`.
- **Use** C++ style comments in preference to C style comments. C comments in system headers are not diagnosed.
- **Use** `delete` on pointer members in destructors. The compiler diagnoses any pointer that does not have a `delete`.
- **Make sure** you have a user copy constructor and assignment operator in classes containing pointers.
- **Use** initialization rather than assignment to members in constructors.
- **Make sure** the initialization list ordering matches the declaration list ordering in constructors.
- **Make sure** base classes have virtual destructors.
- **Make sure** `operator=` returns `*this`.
- **Make sure** prefix forms of increment and decrement return a `const` object.
- **Never overload** operators `&`, `|`, and `.`.

**Note**
The warnings generated by this compiler option are based on the following books from Scott Meyers:

- Effective C++ Second Edition - 50 Specific Ways to Improve Your Programs and Designs
- More Effective C++ - 35 New Ways to Improve Your Programs and Designs

Alternate Options
None

**Werror, WX**
Changes all warnings to errors.

IDE Equivalent

Windows: **General > Treat Warnings As Errors**
Linux: **Compilation Diagnostics > Treat Warnings As Errors**
Mac OS X: **Diagnostics > Treat Warnings As Errors**

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-Werror`
Windows: `/WX`

Arguments
None

Default
OFF
compiler returns diagnostics as usual.

Description

This option changes all warnings to errors.

Alternate Options

Linux and Mac OS X: -diag-error warn
Windows: /Qdiag-error:warn

Werror-all

Changes all warnings and remarks to errors.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Werror-all
Windows: /Werror-all

Arguments

None

Default

OFF  The
compiler returns diagnostics as usual.

**Description**

This option changes all warnings and remarks to errors.

**Alternate Options**

Linux and Mac OS X: `-diag-error warn, remark`
Windows: `/Qdiag-error:warn, remark`

**Wextra-tokens**

Determines whether warnings are issued about extra tokens at the end of preprocessor directives.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-Wextra-tokens
- Wno-extra-tokens`
Windows: None

**Arguments**

None

**Default**

1054
-Wno-extra-tokens

The compiler does not warn about extra tokens at the end of preprocessor directives.

Description
This option determines whether warnings are issued about extra tokens at the end of preprocessor directives.

Alternate Options
None

Wformat
Determines whether argument checking is enabled for calls to printf, scanf, and so forth.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -Wformat
- Wno-format

Windows: None

Arguments
None

Default

-Wno-format

Argument checking is not enabled for calls to printf, scanf, and so forth.

Description

This option determines whether argument checking is enabled for calls to printf, scanf, and so forth.

Alternate Options

None

Wformat-security

Determines whether the compiler issues a warning when the use of format functions may cause security problems.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

1056
Linux and Mac OS X:  
-`-Wformat-security`  
-`-Wno-format-security`  

Windows:  
None

**Arguments**

None

**Default**

-`-Wno-format-security`  
No warning is issued when the use of format functions may cause security problems.

**Description**

This option determines whether the compiler issues a warning when the use of format functions may cause security problems.  
*When `-Wformat-security` is specified, it warns about uses of format functions where the format string is not a string literal and there are no format arguments.*

**Alternate Options**

None

**Winline**

Enables diagnostics about what is inlined and what is not inlined.
IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -Winline
Windows: None

Arguments
None

Default
OFF

Description
This option enables diagnostics about what is inlined and what is not inlined. The diagnostics depend on what interprocedural functionality is available.

Alternate Options
None
WI

Passes options to the linker for processing.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Wl,\textit{option1[,option2,...]}

Windows: None

Arguments

\textit{option}

Is a linker option. This option is not processed by the driver and is directly passed to the linker.

Default

OFF

No options are
Description

This option passes one or more options to the linker for processing. If the linker is not invoked, these options are ignored.

This option is equivalent to specifying option \(-\text{option,link, options}\).

Alternate Options

None

See Also

\texttt{Option} compiler option

\texttt{WL}

Tells the compiler to display a shorter form of diagnostic output.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: See \texttt{Wbrief}.

Windows: \texttt{/WL}

Arguments

None

Default

1060
OFF

The compiler displays its normal diagnostic output.

Description

This option tells the compiler to display a shorter form of diagnostic output. In this form, the original source line is not displayed and the error message text is not wrapped when too long to fit on a single line.

Alternate Options

Linux: -Wbrief
Windows: None

Wmain

Determines whether a warning is issued if the return type of `main` is not expected.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Wmain
-Wno-main

Windows: None
Arguments

None

Default

-Wno-main

No warning is issued if the return type of main is not expected.

Description

This option determines whether a warning is issued if the return type of main is not expected.

Alternate Options

None

Wmissing-declarations

Determines whether warnings are issued for global functions and variables without prior declaration.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

1062
Linux and Mac OS X:  
-`-Wmissing-declarations`  
  `-Wno-missing-declarations`  

Windows: None

Arguments

None

Default

`-Wno-missing-declarations`  
No warnings are issued for global functions and variables without prior declaration.

Description

This option determines whether warnings are issued for global functions and variables without prior declaration.

Alternate Options

None

**Wmissing-prototypes**

Determines whether warnings are issued for missing prototypes.

IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-Wmissing-prototypes`

- `-Wno-missing-prototypes`

Windows: None

Arguments

None

Default

- `-Wno-missing-prototypes`

No

warnings

are issued

for missing

prototypes.

Description

Determines whether warnings are issued for missing prototypes.

Alternate Options

None

Wnon-virtual-dtor

Issue a warning when a class appears to be polymorphic, yet it declares a non-virtual one.

IDE Equivalent

Mac OS X: Diagnostics > Report Non-Virtual Destructor
Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -Wnon-virtual-dtor
Windows: None

Arguments
None

Default
OFF

The compiler does not issue a warning.

Description
Issue a warning when a class appears to be polymorphic, yet it declares a non-virtual one. This option is supported in C++ only.

Alternate Options
None

wn, Qwn
Controls the number of errors displayed before compilation stops. This is a deprecated option.

IDE Equivalent
Windows: Diagnostics > Error Limit
Linux: **Compilation Diagnostics > Set Error Limit**  
Mac OS X: **Diagnostics > Error Limit**

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `\-wnn`  
Windows: `/Qwn n`

Arguments

$n$

Is the number of errors to display.

Default

100  
The compiler displays a maximum of 100 errors before aborting compilation.

Description

1066
This option controls the number of errors displayed before compilation stops.

Alternate Options

None

**wo, Qwo**

Tells the compiler to issue one or more diagnostic messages only once. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

**Linux and Mac OS X:** `-woLn[,Ln,...]`

**Windows:** `/QwoLn[,Ln,...]`

Arguments

`Ln` Is the number of the diagnostic.

Default

OFF

Description

Specifies the ID number of one or more messages. If you specify more than one `Ln`, each `Ln` must be separated by a comma.
Alternate Options

None

\textbf{Wp}

Passes options to the preprocessor.

\textbf{IDE Equivalent}

None

\textbf{Architectures}

IA-32, Intel® 64, IA-64 architectures

\textbf{Syntax}

\textbf{Linux and Mac OS X:} \(-\text{Wp}, \text{option1[,option2,...]}\)

\textbf{Windows:} None

\textbf{Arguments}

\textit{option} \hspace{2cm} \text{Is a preprocessor option. This option is not processed by the driver and is directly passed to the preprocessor.}

\textbf{Default}

\textbf{OFF} \hspace{2cm} \text{No options are passed to}
Description

This option passes one or more options to the preprocessor. If the preprocessor is not invoked, these options are ignored. This option is equivalent to specifying option `-Qoption, cpp, options`.

Alternate Options

None

See Also

`Qoption` compiler option

**Wp64**

Tells the compiler to display diagnostics for 64-bit porting.

IDE Equivalent

Windows: **General > Detect 64-bit Portability Issues**

Linux: None

Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-Wp 64`

Windows: `/Wp 64`

Arguments

None
Default

OFF  
The compiler does not display diagnostics for 64-bit porting.

Description

This option tells the compiler to display diagnostics for 64-bit porting.

Alternate Options

None

Wpointer-arith

Determines whether warnings are issued for questionable pointer arithmetic.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  \(-\text{Wpointer-arith}\)

\(-\text{Wno-pointer-arith}\)

Windows:  

None

Arguments

None
Default

-\textit{-Wno-pointer-arith} \quad \text{No warnings are issued for questionable pointer arithmetic.}

\textbf{Description}

Determines whether warnings are issued for questionable pointer arithmetic.

\textbf{Alternate Options}

None

\textbf{Wport}

Tells the compiler to issue portability diagnostics.

\textbf{IDE Equivalent}

None

\textbf{Architectures}

IA-32, Intel® 64, IA-64 architectures

\textbf{Syntax}

\textbf{Linux and Mac OS X:} None

\textbf{Windows:} /Wport

\textbf{Arguments}

None

\textbf{Default}
OFF

Description
This option tells the compiler to issue portability diagnostics.

Alternate Options
None

Wpragma-once
Determines whether a warning is issued about the use of #pragma once.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: -Wpragma-once
-Wno-pragma-once

Windows: None

Arguments
None

Default
-Wno-pragmas-once

No warning is issued about the use of #pragma once.

Description

This option determines whether a warning is issued about the use of #pragma once.

Alternate Options

None

wr, Qwr

Changes a soft diagnostic to an remark. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -wrLn[,] Ln...

Windows: /QwrLn[,] Ln...

Arguments

Ln Is the
number of the diagnostic to be changed.

**Default**

**OFF**

The compiler returns soft diagnostics as usual.

**Description**

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to a remark. If you specify more than one \( Ln \), each \( Ln \) must be separated by a comma.

**Alternate Options**

None

**Wreorder**

Issue a warning when the order of member initializers does not match the order in which they must be executed.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: -Wreorder
Windows: None

Arguments

None

Default

OFF

The compiler does not issue a warning.

Description

Issue a warning when the order of member initializers does not match the order in which they must be executed. This option is supported with C++ only.

Alternate Options

None

Wreturn-type

Determines whether warnings are issued when a function uses the default int return type or when a return statement is used in a void function.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-Wreturn-type
- Wno-return-type

Windows: None

Arguments

None

Default

- Wno-return-type

No warnings are issued when a function uses the default int return type or when a return statement is used in a void function.

Description

This option determines whether warnings are issued when a function uses the default int return type or when a return statement is used in a void function.
Alternate Options

None

Wshadow

Determines whether a warning is issued when a variable declaration hides a previous declaration.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  \(-\text{Wshadow}\)
                      \(-\text{Wno-shadow}\)

Windows:  None

Arguments

None

Default

\(-\text{Wno-shadow}\)

No warning is issued when a variable declaration hides a previous declaration.
Description

This option determines whether a warning is issued when a variable declaration hides a previous declaration. Same as `-ww1599`.

Alternate Options

None

Wstrict-prototypes

Determines whether warnings are issued for functions declared or defined without specified argument types.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-Wstrict-prototypes`

- `-Wno-strict-prototypes`

Windows: None

Arguments

None

Default

- `-Wno-strict-prototypes`

No warnings are issued for
Description

This option determines whether warnings are issued for functions declared or defined without specified argument types.

Alternate Options

None

Wtrigraphs

Determines whether warnings are issued if any trigraphs are encountered that might change the meaning of the program.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X:  `-Wtrigraphs`

`-Wno-trigraphs`

Windows:  None

Arguments
None

Default

-Wno-trigraphs

No warnings are issued if any trigraphs are encountered that might change the meaning of the program.

Description

This option determines whether warnings are issued if any trigraphs are encountered that might change the meaning of the program.

Alternate Options

None

Wuninitialized

Determines whether a warning is issued if a variable is used before being initialized.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X:  
- Wuninitialized
- Wno-uninitialized

Windows:  
None

Arguments

None

Default

-Wno-uninitialized  
No warning is issued if a variable is used before being initialized.

Description

This option determines whether a warning is issued if a variable is used before being initialized. Equivalent to -ww592 and -wd592.

Alternate Options

-ww592 and -wd592

Wunknown-pragmas

Determines whether a warning is issued if an unknown #pragma directive is used.

IDE Equivalent
None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

**Linux and Mac OS X:** `-Wunknown-pragmas`

`-Wno-unknown-pragmas`

**Windows:** None

**Arguments**

None

**Default**

`-Wunknown-pragmas` No warning is issued if an unknown `#pragma` directive is used.

**Description**

This option determines whether a warning is issued if an unknown `#pragma` directive is used.

**Alternate Options**

None

**Wunused-function**
Determines whether a warning is issued if a declared function is not used.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  
- `-Wunused-function`
- `-Wno-unused-function`

Windows:  
None

**Arguments**

None

**Default**

`-Wno-unused-function`  
No warning is issued if a declared function is not used.

**Description**

This option determines whether a warning is issued if a declared function is not used.

**Alternate Options**
None

**Wunused-variable**

Determines whether a warning is issued if a local or non-constant static variable is unused after being declared.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  
-Wunused-variable  
- Wno-unused-variable

Windows:  
None

**Arguments**

None

**Default**

- Wno-unused-variable

No warning is issued if a local or non-constant static variable is unused
Description

This option determines whether a warning is issued if a local or non-constant static variable is unused after being declared.

Alternate Options

None

ww, Qww

Changes a soft diagnostic to a warning. This is a deprecated option.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: `-wwLn[,Ln,...]`
Windows: `/QwwLn[,Ln,...]`

Arguments

Ln

Is the number of the diagnostic to be
Default

OFF

The compiler returns soft diagnostics as usual.

Description

This option overrides the severity of the soft diagnostic that corresponds to the specified number and changes it to a warning. If you specify more than one Ln, each Ln must be separated by a comma.

Alternate Options

None

Wwrite-strings

Issues a diagnostic message if const char * is converted to (non-const) char *.

IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: -Wwrite-strings

Windows: None

Arguments

1086
None

Default

OFF

No diagnostic message is issued if const char * is converted to (non-const) char *.

Description

This option issues a diagnostic message if const char* is converted to (non-const) char *.

Alternate Options

None

Werror, WX

Changes all warnings to errors.

IDE Equivalent

Windows: General > Treat Warnings As Errors
Linux: Compilation Diagnostics > Treat Warnings As Errors
Mac OS X: Diagnostics > Treat Warnings As Errors

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-Werror`
Windows: `/WX`

Arguments
None

Default
OFF

The compiler returns diagnostics as usual.

Description

This option changes all warnings to errors.

Alternate Options

Linux and Mac OS X: `-diag-error warn`
Windows: `/Qdiag-error:warn`

x, Qx

Tells the compiler to generate optimized code specialized for the Intel processor that executes your program.

IDE Equivalent

Windows: Code Generation > Intel Processor-Specific Optimization
Linux: Code Generation > Intel Processor-Specific Optimization
Mac OS X: Code Generation > Intel Processor-Specific Optimization

Architectures

1088
IA-32, Intel® 64 architectures

Syntax

Linux and Mac OS X: -xprocessor
Windows: /Qxprocessor

Arguments

processor

Indicates the processor for which code is generated. Many of the following descriptions refer to Intel® Streaming SIMD Extensions (Intel® SSE) and Supplemental Streaming SIMD Extensions (Intel® SSSE).

Possible values are:

Host Can generate instructions for the highest instruction set and processor available on the compilation host.

SSE4.2 Can generate Intel® SSE4 Efficient Accelerated String and Text Processing instructions
supported by Intel® Core™ i7 processors. Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator, Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for the Intel® Core™ processor family.

SSE4.1 Can generate Intel® SSE4 Vectorizing Compiler and Media Accelerator instructions for Intel processors. Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions and it can optimize for Intel® 45nm Hi-k
next generation Intel® Core™ microarchitecture. This replaces value S, which is deprecated.

**SSE3_ATOM** Can generate MOVBE instructions for Intel processors and it can optimize for the Intel® Atom™ processor and Intel® Centrino® Atom™ Processor Technology.

**SSSE3** Can generate Intel® SSSE3, SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for the Intel® Core™2 Duo processor family. This replaces
value T, which is deprecated.

**SSE3**
Can generate Intel® SSE3, SSE2, and SSE instructions for Intel processors and it can optimize for processors based on Intel® Core™ microarchitecture and Intel NetBurst® microarchitecture. This replaces value P, which is deprecated.

**SSE2**
Can generate Intel® SSE2 and SSE instructions for Intel processors, and it can optimize for Intel® Pentium® 4 processors, Intel® Pentium® M processors, and Intel® Xeon®
processors with Intel® SSE2. This value is not available on Mac OS X systems. This replaces value N, which is deprecated.

Default

Windows systems: /arch:SSE2
Linux systems: -msse2
Mac OS X systems using IA-32 architecture: SSE3
Mac OS X systems using Intel® 64 architecture: SSSE3

Description

This option tells the compiler to generate optimized code specialized for the Intel processor that executes your program. It also enables optimizations in addition to Intel processor-specific optimizations. The specialized code generated by this option may run only on a subset of Intel processors.

This option can enable optimizations depending on the argument specified. For example, it may enable Intel® Streaming SIMD Extensions 4 (Intel® SSE4), Intel® Supplemental Streaming SIMD Extensions 3 (Intel® SSSE3), Intel® Streaming SIMD Extensions 3 (Intel® SSE3), Intel® Streaming SIMD Extensions 2 (Intel® SSE2), or Intel® Streaming SIMD Extensions (Intel® SSE) instructions.
The binaries produced by these values will run on Intel processors that support all of the features for the targeted processor. For example, binaries produced with SSE3 will run on an Intel® Core™ 2 Duo processor, because that processor completely supports all of the capabilities of the Intel® Pentium® 4 processor, which the SSE3 value targets. Specifying the SSSE3 value has the potential of using more features and optimizations available to the Intel® Core™ 2 Duo processor.

Do not use processor values to create binaries that will execute on a processor that is not compatible with the targeted processor. The resulting program may fail with an illegal instruction exception or display other unexpected behavior. For example, binaries produced with SSE3 may produce code that will not run on Intel® Pentium® III processors or earlier processors that do not support SSE2 instructions.

Compiling the main program with any of the processor values produces binaries that display a fatal run-time error if they are executed on unsupported processors. For more information, see Optimizing Applications.

If you specify more than one processor value, code is generated for only the highest-performing processor specified. The highest-performing to lowest-performing processor values are: SSE4.2, SSE4.1, SSSE3, SSE3, SSE2. Note that processor value SSE3_ATOM does not fit within this group.

Compiler options m and arch produce binaries that should run on processors not made by Intel that implement the same capabilities as the corresponding Intel processors.

Previous value O is deprecated and has been replaced by option -msse3 (Linux and Mac OS X) and option /arch:SSE3 (Windows).

Previous values W and K are deprecated. The details on replacements are as follows:

- Mac OS X systems: On these systems, there is no exact replacement for W or K. You can upgrade to the default option -msse3 (IA-32 architecture) or option -mssse3 (Intel® 64 architecture).
• **Windows and Linux systems:** The replacement for W is `-msse2` (Linux) or `/arch:SSE2` (Windows). There is no exact replacement for K. However, on Windows systems, `/QxK` is interpreted as `/arch:IA32`; on Linux systems, `-xK` is interpreted as `-mia32`. You can also do one of the following:
  
  • Upgrade to option `-msse2` (Linux) or option `/arch:SSE2` (Windows). This will produce one code path that is specialized for Intel® SSE2. It will not run on earlier processors.
  
  • Specify the two option combination `-mia32` `-axSSE2` (Linux) or `/arch:IA32` `/QaxSSE2` (Windows). This combination will produce an executable that runs on any processor with IA-32 architecture but with an additional specialized Intel® SSE2 code path.

The `-x` and `/Qx` options enable additional optimizations not enabled with option `-m` or option `/arch`.

**Alternate Options**

None

**See Also**

- `ax, Qax` compiler option
- `m` compiler option
- `arch` compiler option

**x (Linux)**

All source files found subsequent to `-x type` will be recognized as a particular type.

**IDE Equivalent**

Windows: None
Linux: None
Mac OS X: None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: \(-x \) \textit{type}
Windows: \textit{None}

Arguments

\textit{type} is the type of source file.
Possible values are:

- \texttt{c} \hspace{0.5cm} C source file
- \texttt{c++} \hspace{0.5cm} C++ source file
- \texttt{c-header} \hspace{0.5cm} C header file
- \texttt{cpp-output} \hspace{0.5cm} C preprocessed file
- \texttt{c++-cpp-output} \hspace{0.5cm} C++ preprocessed file
- \texttt{assembler} \hspace{0.5cm} Assembly file
- \texttt{assembler-with-cpp} \hspace{0.5cm} Assembly file that needs to be preprocessed
- \texttt{none} \hspace{0.5cm} Disable recognition, and revert to
file extension

Default
	none

Disable recognition and revert to file extension.

Description

All source files found subsequent to –x will be recognized as a particular type.

Alternate Options

None

Example

Suppose you want to compile the following C and C++ source files whose extensions are not recognized by the compiler:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>file1.c99</td>
<td>C</td>
</tr>
<tr>
<td>file2.cplusplus</td>
<td>C++</td>
</tr>
</tbody>
</table>

We will also include these files whose extensions are recognized:

<table>
<thead>
<tr>
<th>File Name</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>file3.c</td>
<td>C</td>
</tr>
<tr>
<td>file4.cpp</td>
<td>C++</td>
</tr>
</tbody>
</table>

The command-line invocation using the –x option follows:

```
icpc -x c file1.c99 -x c++ file2.cplusplus -x none file3.c file4.cpp```
**X**

Removes standard directories from the include file search path.

**IDE Equivalent**

Windows: Preprocessor > Ignore Standard Include Path
Linux: Preprocessor > Ignore Standard Include Path
Mac OS X: Preprocessor > Ignore Standard Include Path

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X:  
Windows:  

**Arguments**

None

**Default**

OFF  

**Description**

This option removes standard directories from the include file search path. It prevents the compiler from searching the default path specified by the INCLUDE environment variable.
On Linux and Mac OS X systems, specifying `-X` (or `-noinclude`) prevents the compiler from searching in `/usr/include` for files specified in an INCLUDE statement.

You can use this option with the `I` option to prevent the compiler from searching the default path for include files and direct it to use an alternate path.

**Alternate Options**

Linux and Mac OS X: `-nostdinc`

Windows: None

**See Also**

`I` compiler option

**Xlinker**

Passes a linker option directly to the linker.

**IDE Equivalent**

None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-Xlinker option`

Windows: None

**Arguments**

`option`  
Is a linker option.

**Default**
OFF

No options are passed directly to the linker.

Description

This option passes a linker option directly to the linker.

If `-Xlinker -shared` is specified, only `-shared` is passed to the linker and no special work is done to ensure proper linkage for generating a shared object. `-Xlinker` just takes whatever arguments are supplied and passes them directly to the linker.

If you want to pass compound options to the linker, for example `"-L $HOME/lib"`, you must use one of the following methods:

- `-Xlinker -L -Xlinker $HOME/lib`
- `-Xlinker "-L $HOME/lib"`
- `-Xlinker -L\ $HOME/lib`

Alternate Options

None

See Also

- `shared` compiler option
- `link` compiler option

Y-

Tells the compiler to ignore all other precompiled header files.

IDE Equivalent

None
Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: /Y-

Arguments

None

Default

OFF

The compiler recognizes precompiled header files when certain compiler options are specified.

Description

This option tells the compiler to ignore all other precompiled header files.

Alternate Options

None

See Also

Ye compiler option
YU compiler option
**YY compiler option**

**pch-create, Yc**

Lets you create and specify a name for a precompiled header file.

**IDE Equivalent**

Windows: **Precompiled Headers > Create-Use Precompiled Header / Create-Use PCH Through File**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-pch-createfile`

Windows: `/Ycfile`

**Arguments**

`file`  
Is the name for the precompiled header file.

**Default**

OFF  
The compiler does not create or use
This option lets you specify a name for a precompiled header (PCH) file. It is supported only for single source file compilations.

The `.pchi` extension is not automatically appended to the file name.

This option cannot be used in the same compilation as the `-pch-use` option.

Depending on how you organize the header files listed in your sources, this option may increase compile times. To learn how to optimize compile times using the PCH options, see "Precompiled Header Files" in the User's Guide.

**Alternate Options**

None

**Example**

Consider the following command line:

```
icpc -pch-create /pch/source32.pchi source.cpp
```

It produces the following output:

"source.cpp": creating precompiled header file "/pch/source32.pchi"

**See Also**

Precompiled Headers

**Yu**

Tells the compiler to use a precompiled header file.

**IDE Equivalent**

Windows: **Language > Create/Use Precompiled Header**
Linux: None
Mac OS X: None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Windows: /Yufile

Arguments

file
Is the name of the precompiled header file to use.

Default
OFF
The compiler does not use precompiled header files unless it is told to do so.

Description
This option tells the compiler to use a precompiled header file.
When this option is specified, the Microsoft Visual C++* compiler ignores all text, including declarations preceding the `#include` statement of the specified file.

**Alternate Options**

None

**See Also**

`Y-` compiler option

`YC` compiler option

`YX` compiler option

**YX**

Tells the compiler to use a precompiled header file or to create one if none exist.

**IDE Equivalent**

Windows: **Language > Create/Use Precompiled Header**

Linux: None

Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: **None**

Windows: `/YX[\textit{file}]`

**Arguments**

\textit{file}

Is the name of the precompiled header file
Default

OFF

The compiler does not use or create precompiled header files unless it is told to do so.

Description

This option tells the compiler to use a precompiled header file or to create one if none exists.

Alternate Options

None

See Also

Y- compiler option
Yc compiler option
Yu compiler option
Fp compiler option

g, Z1, Z7

Tells the compiler to generate full debugging information in the object file.

IDE Equivalent

1106
Windows: **General > Debug Information Format**  
Linux: **General > Include Debug Information**  
Mac OS X: **General > Generate Debug Information**

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: `-g`

Windows: `/Zi`

`/Z7`

**Arguments**

None

**Default**

OFF  
No debugging information is produced in the object file.

**Description**

This option tells the compiler to generate symbolic debugging information in the object file for use by debuggers.  
The compiler does not support the generation of debugging information in assemblable files. If you specify this option, the resulting object file will contain debugging information, but the assemblable file will not.
This option turns off `O2` and makes `O0` (Linux and Mac OS X) or `Od` (Windows) the default unless `O2` (or another `O` option) is explicitly specified in the same command line.

On Linux systems using Intel® 64 architecture and Linux and Mac OS X systems using IA-32 architecture, specifying the `-g` or `-O0` option sets the `-fno-omit-frame-pointer` option.

Alternate Options

Linux: None
Windows: `/ZI, /debug`

Za

Disable Microsoft Visual C++ compiler language extensions.

IDE Equivalent

Windows: Language > Disable Language Extensions
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None
Windows: `/Za`

Arguments

None

Default
OFF

The compiler provides support for extended ANSI C.

Description

Disable Microsoft Visual C++ compiler language extensions.

Alternate Options

None

See Also

ze compiler option
zc compiler option

zc

Lets you specify ANSI C standard conformance for certain language features.

IDE Equivalent

Windows: Language > Treat wchar_t as Built-in Type / Force Conformance In For Loop Scope
Linux: None
Mac OS X: None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Windows: /Zc:\texttt{arg}[,,\texttt{arg}]

Arguments

\texttt{arg}

Is the language feature for which you want standard conformance. Possible values are:

\texttt{forScope} Enforce standard behavior for initializers of for loops.

\texttt{wchar_t} Specify that \texttt{wchar_t} is a native data type.

Default

OFF

/\texttt{Zc:forScope}, \texttt{wchar_t} is disabled if /Qvc8 is not specified.
ON /Zc:forScope,
wchar_t is
enabled when
/Qvc8 is
specified.

Description
This option lets you specify ANSI C standard conformance for certain language
features when you also specify /Ze.

Alternate Options
None

See Also
Ze compiler option

Zd
This option has been deprecated. Use keyword minimal in debug (Windows*).

Ze
Enables Microsoft Visual C++* compiler language extensions.
This option has been deprecated.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures

Syntax
Linux and Mac OS X: None
Intel® C++ Compiler User and Reference Guides

Windows: /Zc

Arguments
None

Default
ON

The compiler provides support for extended ANSI C.

Description
This option enables Microsoft Visual C++* compiler language extensions.

Alternate Options
None

See Also

Zc compiler option
Za compiler option

Zg

Tells the compiler to generate function prototypes.

IDE Equivalent
None

Architectures
IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: None
Windows: /Zg

Arguments

None

Default

OFF

The compiler does not create function prototypes.

Description

This option tells the compiler to generate function prototypes.

Alternate Options

None

g, Zi, Z7

Tells the compiler to generate full debugging information in the object file.

IDE Equivalent

Windows: General > Debug Information Format
Linux: General > Include Debug Information
Mac OS X: General > Generate Debug Information

Architectures

IA-32, Intel® 64, IA-64 architectures
Syntax

Linux and Mac OS X: `-g`
Windows: `/Zi`
    `/Z7`

Arguments

None

Default

OFF  No debugging information is produced in the object file.

Description

This option tells the compiler to generate symbolic debugging information in the object file for use by debuggers. The compiler does not support the generation of debugging information in assemblable files. If you specify this option, the resulting object file will contain debugging information, but the assemblable file will not.

This option turns off `O2` and makes `O0` (Linux and Mac OS X) or `Od` (Windows) the default unless `O2` (or another `O` option) is explicitly specified in the same command line.

On Linux systems using Intel® 64 architecture and Linux and Mac OS X systems using IA-32 architecture, specifying the `-g` or `-O0` option sets the `-fno-omit-frame-pointer` option.

Alternate Options
Linux: None
Windows: /ZI, /debug

**ZI**

Tells the compiler to generate full debugging information in the object file.

**IDE Equivalent**

Windows: **General > Debug Information Format**
Linux: None
Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: See g.
Windows: /ZI

**Arguments**

None

**Default**

OFF

No debugging information is produced in the object file.

**Description**
For details, see /zi.

Alternate Options

- **Linux**: -g
- **Windows**: /zi, /Z7

**ZI**

Causes library names to be omitted from the object file.

**IDE Equivalent**

Windows: **Advanced > Omit Default Library Names**
Linux: None
Mac OS X: None

**Architectures**

IA-32, Intel® 64, IA-64 architectures

**Syntax**

Linux and Mac OS X: None
Windows: /Zl

**Arguments**

None

**Default**

OFF or specified library names
Description

This option causes library names to be omitted from the object file.

Alternate Options

None

\textbf{Zp}

Specifies alignment for structures on byte boundaries.

\textbf{IDE Equivalent}

Windows: \textit{Code Generation > Struct Member Alignment}

Linux: \textit{Data > Structure Member Alignment}

Mac OS X: \textit{Data > Structure Member Alignment}

\textbf{Architectures}

IA-64 architecture

\textbf{Syntax}

Linux and Mac OS X: \texttt{-Zp \[n\]}

Windows: \texttt{/Zp \[n\]}

\textbf{Arguments}

\(n\)

Is the byte size
boundary. Possible values are 1, 2, 4, 8, or 16.

Default

Zp16 Structures are aligned on either size boundary 16 or the boundary that will naturally align them.

Description

This option specifies alignment for structures on byte boundaries. If you do not specify \( n \), you get Zp16.

Alternate Options

None

Zs

Tells the compiler to check only for correct syntax.
IDE Equivalent

None

Architectures

IA-32, Intel® 64, IA-64 architectures

Syntax

Linux and Mac OS X: None

Windows: \( /Zs \)

Arguments

None

Default

OFF

Normal compilation is performed.

Description

This option tells the compiler to check only for correct syntax.

Alternate Options

Linux: \(-\text{syntax}, -\text{fsyntax-only}\)

Windows: None

Quick Reference Guide and Cross Reference

The table in this section summarizes Intel® C++ compiler options used on Linux* OS and Mac OS* X systems. Each summary also shows the equivalent compiler options on Windows* operating systems, if any.

Some compiler options are only available on systems using certain architectures, as indicated by these labels:
<table>
<thead>
<tr>
<th>Label</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>i32</td>
<td>The option is available on systems using IA-32 architecture.</td>
</tr>
<tr>
<td>i64em</td>
<td>The option is available on systems using Intel® 64 architecture.</td>
</tr>
<tr>
<td>i64</td>
<td>The option is available on systems using IA-64 architecture.</td>
</tr>
</tbody>
</table>

If "only" appears in the label, the option is only available on the identified system or architecture.
If no label appears, the option is available on all supported systems and architectures.
For more details on the options, refer to the [Alphabetical Compiler Options](#) section.

The Intel® C++ Compiler includes the Intel® Compiler Option Mapping tool. This tool lets you find equivalent options by specifying compiler option `-map-opts` (Linux and Mac OS X) or `/Qmap-opts` (Windows).

For information on conventions used in this table, see [Conventions](#).

**Quick Reference of Linux OS and Mac OS X Options**

The following table summarizes all supported Linux OS and Mac OS X options. It also shows equivalent Windows* OS options, if any.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>Specifies an OFF</td>
<td>OFF</td>
<td>/QA&lt;name&gt;[&lt;(value)&gt;]</td>
</tr>
<tr>
<td>A&lt;name&gt;[(&lt;value&gt;)]</td>
<td>identifier for an assertion.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-A-</td>
<td>Disables all OFF</td>
<td>OFF</td>
<td>/QA[-]</td>
</tr>
<tr>
<td></td>
<td>predefined macros.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-[no-]alias-args</td>
<td>Determines whether function arguments can alias each other. Deprecated; use -fargument - [no]alias .</td>
<td>-alias-args /Qalias-args[-]</td>
<td></td>
</tr>
<tr>
<td>-[no-]alias-const</td>
<td>Determines whether the compiler assumes a parameter of type pointer-to-const does not alias with a parameter of type pointer-to-non-const.</td>
<td>-no-alias- /Qalias-const[-]</td>
<td></td>
</tr>
<tr>
<td>-[no]align</td>
<td>Determines</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td>whether variables and arrays are naturally aligned.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-ansi</td>
<td>Enables language compatibility with the gcc option -ansi.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]ansi-alias</td>
<td>Enables or disables use of ANSI aliasing rules in optimization.</td>
<td>-no-ansi-alias</td>
<td>/Qansi-alias[-]</td>
</tr>
<tr>
<td>-auto-ilp32</td>
<td>Instructs the compiler to analyze the program to determine if there are 64-bit pointers which can</td>
<td>OFF</td>
<td>/Qauto-ilp32</td>
</tr>
<tr>
<td>(i64em, i64)</td>
<td></td>
<td></td>
<td>(i64em, i64)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>-ax&lt;processor&gt;</code> (i32, i64em)</td>
<td>Tells the compiler to generate multiple, processor-specific auto-dispatch code paths for Intel processors if there is a performance benefit.</td>
<td>OFF</td>
<td><code>/Qax&lt;processor&gt;</code> (i32, i64em)</td>
</tr>
<tr>
<td><code>-B&lt;dir&gt;</code></td>
<td>Specifies a directory that can be used to find include files, libraries, and executables.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>-Bdynamic</code></td>
<td>Enables dynamic linking of libraries at run time.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-Bstatic</code></td>
<td>Enables static linking of a user's library.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-c</code></td>
<td>Prevents linking.</td>
<td>OFF</td>
<td>/c</td>
</tr>
<tr>
<td><code>-C</code></td>
<td>Place comments in preprocessed source output.</td>
<td>OFF</td>
<td>/c</td>
</tr>
<tr>
<td><code>-c99</code></td>
<td>Determines whether C99 support is enabled for C programs.</td>
<td><code>-no-c99</code></td>
<td><code>/Qc99[-]</code></td>
</tr>
<tr>
<td><code>-check-uninit</code></td>
<td>Determines whether checking occurs for</td>
<td><code>-no-check-uninit</code></td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-[no-]complex-limited-range</td>
<td>Determines whether the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX is enabled</td>
<td>-no-complex-</td>
<td>/Qcomplex-limited-range[-]</td>
</tr>
<tr>
<td>-cxxlib[=&lt;dir&gt;]</td>
<td>Determines whether the compile links using the C++ run-time libraries and header files provided by gcc.</td>
<td>C++: -cxxlib</td>
<td>None</td>
</tr>
<tr>
<td>-cxxlib-nostd</td>
<td></td>
<td>C: -no-cxxlib</td>
<td></td>
</tr>
<tr>
<td>-no-cxxlib</td>
<td></td>
<td>OFF</td>
<td>/D&lt;name&gt;[=&lt;value&gt;]</td>
</tr>
<tr>
<td>-D&lt;name&gt;[=&lt;value&gt;]</td>
<td>Defines a macro name that can be</td>
<td>OFF</td>
<td>/D&lt;name&gt;[=&lt;value&gt;]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OSI Option</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-----------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-dD</td>
<td>Same as -dN, but outputs #define directives in preprocessed source.</td>
<td>OFF</td>
<td>/QdD</td>
</tr>
<tr>
<td>-debug [keyword]</td>
<td>Enables or disables generation of debugging information.</td>
<td>-debug none</td>
<td>/debug [keyword]</td>
</tr>
<tr>
<td>-diag-&lt;type&gt;</td>
<td>Controls the display of diagnostic information.</td>
<td>OFF</td>
<td>/Qdiag-&lt;type&gt;::&lt;diag-list&gt;</td>
</tr>
<tr>
<td>-diag-dump</td>
<td>Tells the compiler to print all enabled diagnostic messages.</td>
<td>OFF</td>
<td>/Qdiag-dump</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-diag-enable sv-include</td>
<td>Tells the Static Verifier to analyze include files and source files when issuing diagnostic messages.</td>
<td>OFF</td>
<td>/Qdiag-enable sv-include</td>
</tr>
<tr>
<td>-diag-error-limit &lt;n&gt;</td>
<td>Specifies the maximum number of errors allowed before compilation stops.</td>
<td>&lt;n&gt;=30</td>
<td>/Qdiag-error-limit:&lt;n&gt;</td>
</tr>
<tr>
<td>-diag-file[=file]</td>
<td>Causes the results of diagnostic analysis to be output to a file.</td>
<td>OFF</td>
<td>/Qdiag-file[:file]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-diag-file- append[=file]</td>
<td>Causes the results of diagnostic analysis to be appended to a file.</td>
<td>OFF</td>
<td>/Qdiag-file- append[=file]</td>
</tr>
<tr>
<td>-[no-]diag-id-numbers</td>
<td>Tells the compiler to display diagnostic messages by using their ID number values.</td>
<td>-diag-id-numbers</td>
<td>/Qdiag-id-numbers [-]</td>
</tr>
<tr>
<td>-diag-once &lt;id&gt;[,&lt;id&gt;,...]</td>
<td>Tells the compiler to issue one or more diagnostic messages only once.</td>
<td>OFF</td>
<td>/Qdiag-once:&lt;id&gt;[,&lt;id&gt;,..]</td>
</tr>
<tr>
<td>-dM</td>
<td>Output macro definitions in effect after</td>
<td>OFF</td>
<td>/QdM</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-dN</td>
<td>Same as -dD, but output #define directives contain only macro names.</td>
<td>OFF</td>
<td>/QdN</td>
</tr>
<tr>
<td>-dryrun</td>
<td>Specifies that driver tool commands should be shown but not executed.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-dumpmachine</td>
<td>Displays the target machine and operating system configuration.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-dumpversion</td>
<td>Displays the version number of the compiler.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-dynamiclib</td>
<td>Invokes the libtool command to generate dynamic libraries.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>(Mac OS X only; i32, i64em)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-dynamic-linker &lt;file&gt;</td>
<td>Specifies a dynamic linker other than the default.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-E</td>
<td>Preprocess to stdout.</td>
<td>OFF</td>
<td>/E</td>
</tr>
<tr>
<td>-[no-]early-template-check</td>
<td>Lets you semantically check template function prototypes before instantiation</td>
<td>-no-early-template-check</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-EP</td>
<td>Preprocess to stdout omitting #line directives.</td>
<td>OFF</td>
<td>/EP</td>
</tr>
<tr>
<td>-export</td>
<td>Enables support for the C++ export template feature.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-export-dir &lt;dir&gt;</td>
<td>Specifies a directory name for the exported template search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>F&lt;dir&gt;</td>
<td>(Mac OS X only) Adds framework directory to head of include file search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fabi-version=&lt;n&gt;</td>
<td>(i32 only) Instructs the compiler to select a</td>
<td>Varies</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------</td>
<td>-------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>-f[no-]alias</td>
<td>Tells the compiler to assume aliasing in a program.</td>
<td>-falias</td>
<td>/Oa</td>
</tr>
<tr>
<td>-f[no-]align-functions[=&lt;n&gt;] (i32, i64em)</td>
<td>Aligns functions on optimal byte boundary.</td>
<td>-fno-align-functions</td>
<td>/Qfnalign[:&lt;n&gt;][-] (i32, i64em)</td>
</tr>
<tr>
<td>-falign-stack=&lt;mode&gt; (i32 only)</td>
<td>Tells the compiler the stack alignment to use on entry to routines.</td>
<td>OFF</td>
<td>-falign-stack=default</td>
</tr>
<tr>
<td>-fargument-[no]alias</td>
<td>Determines whether function arguments can alias each other.</td>
<td>-fargument-alias</td>
<td>/Qalias-args[-]</td>
</tr>
<tr>
<td>-fargument-noalias-global</td>
<td>Tells the compiler that</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>arguments cannot alias each other and cannot alias global storage.</td>
<td>-fasm-blocks (Mac OS X only; i32, i64em) Enables the use of blocks and entire functions of assembly code within a C or C++ file.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fast</td>
<td>Maximizes speed across the entire program.</td>
<td>OFF</td>
<td>/fast</td>
</tr>
<tr>
<td>-[no-]fast-transcendentals</td>
<td>Enables the compiler to replace calls to transcendental functions with faster</td>
<td>OFF</td>
<td>/Qfast-transcendentals[-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-f[no-]builtin[-func]</td>
<td>Enables or disables inline expansion of intrinsic functions.</td>
<td>OFF</td>
<td>/Oi[-]</td>
</tr>
<tr>
<td>-fcode-asm</td>
<td>Produces an assembly listing with machine code annotations.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]common</td>
<td>Determines whether the compiler treats common symbols as global definitions.</td>
<td>-fcommon</td>
<td>None</td>
</tr>
<tr>
<td>-fno-exceptions</td>
<td>Enables exception exception</td>
<td>C++: -</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fexceptions</td>
<td>C:</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-f(no-)fnalias</td>
<td>Assume aliasing within functions.</td>
<td>-ffnalias</td>
<td>/Ow[-]</td>
</tr>
<tr>
<td>-ffreestanding</td>
<td>Ensures that compilation takes place in a freestanding environment.</td>
<td>OFF</td>
<td>/Qfreestanding</td>
</tr>
<tr>
<td>-ffunction-sections</td>
<td>Places each function in its own COMDAT section.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-f(no-)inline</td>
<td>Tells the compiler to inline functions declared with __inline and perform</td>
<td>-fno-inline</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>-f[no-]inline-functions</td>
<td>Enables C++ inlining.</td>
<td>-finline-functions</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Enables function inlining for single file compilation.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-finline-limit=&lt;n&gt;</td>
<td>Lets you specify the maximum size of a function to be inlined.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Lets you specify the maximum size of a function to be inlined.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-f[no-]instrument-functions</td>
<td>Determines whether function entry and exit points are instrumented.</td>
<td>-fno-</td>
<td>/Qinstrument-functions[-]</td>
</tr>
<tr>
<td></td>
<td>Determines whether function entry and exit points are instrumented.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-f[no-]jump-tables</td>
<td>Determines whether jump tables are generated for switch statements.</td>
<td>-fjump-</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>Determines whether jump tables are generated for switch statements.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-f[no-]keep-static-consts</td>
<td>Tells the compiler to preserve allocation of variables that are not referenced in the source.</td>
<td>-fno-keep-static-consts</td>
<td>/Qkeep-static-consts[-]</td>
</tr>
<tr>
<td>-[no-]fma</td>
<td>enable/disble the combining of floating point multiplies and add/subtract operations</td>
<td>-fma</td>
<td>/Qfma[-]</td>
</tr>
<tr>
<td>(i64 only, Linux only)</td>
<td></td>
<td></td>
<td>(i64 only)</td>
</tr>
<tr>
<td>-f[no-]math-errno</td>
<td>Tells the compiler that errno can be reliably tested after calls to standard math library</td>
<td>-fno-math-errno</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-fminshared</td>
<td>Specifies that a compilation unit is a component of a main program and should not be linked as part of a shareable object.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fmudflap</td>
<td>The compiler instruments risky pointer operations to prevent buffer overflows and invalid heap use.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-gnu-keywords</td>
<td>Does not recognize typeof as</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-fno-implicit-inline-templates</td>
<td>Tells the compiler to not emit code for implicit instantiation of inline templates.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-implicit-templates</td>
<td>Tells the compiler to not emit code for non-inline templates that are instantiated implicitly.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-operator-names</td>
<td>Disables support for the operator names specified in the standard.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fno-rtti (i32, i64em)</td>
<td>Disables support for</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-f[no-]non-call-exceptions</td>
<td>Allows trapping instructions to throw C++ exceptions.</td>
<td>-fno-non-call-exceptions</td>
<td>None</td>
</tr>
<tr>
<td>-f[no-]non-lvalue-assign</td>
<td>Determines whether casts and conditional expressions can be used as lvalues.</td>
<td>-fnon-lvalue-assign</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]fnsplit (i32 only)</td>
<td>Enables or disables function splitting (enabled with -prof-use).</td>
<td>-no-fnsplit</td>
<td>/Qfnsplit[-] (i32, i64)</td>
</tr>
<tr>
<td>-f[no-]omit-frame-pointer</td>
<td>Enables or disables using EBP as a</td>
<td>-fomit-frame-pointer</td>
<td>/Oy[-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-------------------------------------------------------</td>
<td>------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-fp-model keyword</td>
<td>Controls the semantics of floating-point calculations.</td>
<td>-fp-model fast=1</td>
<td>/fp keyword:</td>
</tr>
<tr>
<td>-[no-]fp-port (i32, i64em)</td>
<td>Round floating-point results at assignments and casts (some speed impact).</td>
<td>-no-fp-port /Qfp-port[-] (i32, i64em)</td>
<td></td>
</tr>
<tr>
<td>-[no-]fp-relaxed (i64 only; Linux only)</td>
<td>Enables or disables use of faster but slightly less accurate code sequences for math</td>
<td>-no-fp-relaxed relaxed /Qfp-relaxed[-] (i64 only)</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-fp-speculation=&lt;mode&gt;=mode</td>
<td>Tells the compiler the mode in which to speculate on floating-point operations.</td>
<td></td>
<td>/Qfp-speculation:&lt;mode&gt;</td>
</tr>
<tr>
<td>-fp-stack-check (i32, i64em)</td>
<td>Enables FP stack checking after every function/procedure call.</td>
<td>OFF</td>
<td>/Qfp-stack-check (i32, i64em)</td>
</tr>
<tr>
<td>-fpack-struct</td>
<td>Specifies that structure members should be packed together.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fpascal-strings</td>
<td>Allow for Pascal-style string literals.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fpermissive</td>
<td>Allow for</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>(i32 only)</td>
<td>non-conformant code.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-f[no-]pic</td>
<td>Determines whether the compiler generates position-independent code.</td>
<td>varies</td>
<td>None</td>
</tr>
<tr>
<td>-fpie</td>
<td>Tells the compiler to generate position-independent code. The generated code can only be linked into executables.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fr32</td>
<td>Disables the use of the high floating-point</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-freg-struct-return</td>
<td>Return struct and union values in registers when possible.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fshort-enums</td>
<td>Tells the compiler to allocate as many bytes as needed for enumerated types.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-fsourc-asm</td>
<td>Produce assembly file with optional source annotations.</td>
<td>OFF</td>
<td>/FAs</td>
</tr>
<tr>
<td>-f[no-]stack-protector</td>
<td>Same as -f[no-]stack-security-check.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>-f[no-]stack-security-check</code></td>
<td>Determines whether the compiler generates code that detects some buffer overruns.</td>
<td><code>-fno-stack-security-check</code></td>
<td><code>/GS[-]</code></td>
</tr>
<tr>
<td><code>-fsyntax-only</code></td>
<td>Performs syntax and semantic checking only (no object file produced).</td>
<td>OFF</td>
<td><code>/Zs</code></td>
</tr>
<tr>
<td><code>-ftemplate-depth-&lt;n&gt;</code></td>
<td>Control the depth in which recursive templates are expanded.</td>
<td>OFF</td>
<td><code>/Qtemplate-depth-&lt;n&gt;</code></td>
</tr>
<tr>
<td><code>-ftls-model=model</code></td>
<td>Change thread local storage mode.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-ftrapuv</code></td>
<td>Trap</td>
<td>OFF</td>
<td><code>/Qtrapuv</code></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
</tbody>
</table>
| -[no-]ftz        | Enables or disables flush denormal results to zero. | i64: -no-ftz  
                     (i32, i64em: -ftz) | /Qftz[-]       |
<p>| -funroll-loops   | Same as -unroll.                                | ON      | /Qunroll                     |
| -funroll-all-loops | Unroll all loops even if the number of iterations is uncertain when the loop is entered. | OFF     | None                         |
| -f[no-]unsigned-bitfields | Changes default bitfield type to unsigned. | -fno-unsigned-bitfields | None                         |
| -funsigned-char  | Changes default char type to unsigned.          | OFF     | None                         |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS</th>
</tr>
</thead>
<tbody>
<tr>
<td>-f[no-]verbose-asm</td>
<td>Produces an assembly listing file with compiler comments.</td>
<td>-fno-verbose-asm</td>
<td>None</td>
</tr>
<tr>
<td>-fvisibility=keyword</td>
<td>Specifies the default visibility for global symbols.</td>
<td>-fvisibility=default</td>
<td>None</td>
</tr>
<tr>
<td>-fvisibility-inlines-hidden</td>
<td>Causes inline member functions to be marked hidden.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-g</td>
<td>Generates full debugging information in the object file.</td>
<td>OFF</td>
<td>/zi, z7</td>
</tr>
<tr>
<td>-g0</td>
<td>Disables generation</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-gcc</td>
<td>Defines or undefines GNU macros.</td>
<td>gccc</td>
<td>None</td>
</tr>
<tr>
<td>-gcc-sys</td>
<td>Defines GNU macros only during compilation of system headers.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-gcc-name=&lt;dir&gt;</td>
<td>Specifies the location of the gcc compiler.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-gcc-version</td>
<td>Provides compatible behavior with gcc.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-gdwarf-2</td>
<td>Enables generation of debug information using the</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-gxx-name=&lt;dir&gt;</td>
<td>Uses the g++ compiler as environment for C++ compilation.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]global-hoist</td>
<td>Enables or disables external globals to be load safe.</td>
<td>-global-hoist</td>
<td>/Qglobal-hoist[-]</td>
</tr>
<tr>
<td>-H</td>
<td>Prints include file order.</td>
<td>OFF</td>
<td>/QH</td>
</tr>
<tr>
<td>-help [category]</td>
<td>Displays all available compiler options or a category of compiler options.</td>
<td>OFF</td>
<td>/help [category]</td>
</tr>
<tr>
<td>-help-pragma (i32, i64em)</td>
<td>Displays all supported pragmas.</td>
<td>OFF</td>
<td>/Qhelp-pragma</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-I&lt;dir&gt;</td>
<td>Adds directory to include file search path.</td>
<td>OFF</td>
<td>/I&lt;dir&gt;</td>
</tr>
<tr>
<td>-[no-]icc</td>
<td>Defines certain Intel compiler macros.</td>
<td>-icc</td>
<td>None</td>
</tr>
<tr>
<td>-idirafter&lt;dir&gt;</td>
<td>Adds a directory to the second include file search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-imacros &lt;file&gt;</td>
<td>Specifies a lead header in a translation unit.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]inline-calloc (i32, i64em)</td>
<td>Tells the compiler to inline calls to calloc() as calls to malloc() and memset().</td>
<td>-no-inline-calloc</td>
<td>/Qinline-calloc[-] (i32, i64em)</td>
</tr>
<tr>
<td>-inline-debug-info (Linux only)</td>
<td>Preserve the source</td>
<td>OFF</td>
<td>/Qinline-debug-info</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td>position of inlined code instead of assigning the call-site source position to inlined code</td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td>Specify percentage multiplier that should be applied to all inlining options that define upper limits.</td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td>Inline routine whenever the compiler can do so.</td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
</tr>
<tr>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td>control inline expansion</td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
<td><img src="https://via.placeholder.com/150" alt="image" /></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>--------------------------------------------------</td>
<td>----------------------------------------------</td>
<td>-------------------------------------------------------</td>
</tr>
<tr>
<td>-[no-]inline-max-per-compile[=n]</td>
<td>Specify maximum number of times inlining may be applied to an entire compilation unit.</td>
<td>-no-inline-max-per-compile[=n]</td>
<td>/Qinline-max-per-compile[=n][-]</td>
</tr>
<tr>
<td>-[no-]inline-max-per-routine[=n]</td>
<td>Specify maximum number of times the inliner may inline into a particular routine.</td>
<td>-no-inline-max-per-routine[=n]</td>
<td>/Qinline-max-per-routine[=n][-]</td>
</tr>
<tr>
<td>-[no-]inline-max-size[=n]</td>
<td>Specify lower limit for the size of what the inliner considers to be a large routine.</td>
<td>-no-inline-max-size[=n]</td>
<td>/Qinline-max-size[=n][-]</td>
</tr>
</tbody>
</table>

level=0 (if -O0 is in effect)
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-[no-]inline-max-total-size[=n]</td>
<td>Specify how much larger a routine can normally grow when inline expansion is performed.</td>
<td>-no-inline-max-total-size[=n]/Qinline-max-total-size[=n][-]</td>
<td></td>
</tr>
<tr>
<td>-[no-]inline-min-size[=n]</td>
<td>Specify upper limit for the size of what the inliner considers to be a small routine.</td>
<td>-no-inline-min-size[=n]/Qinline-min-size[=n][-]</td>
<td></td>
</tr>
<tr>
<td>-ip</td>
<td>Enables single-file IP optimization s (within files).</td>
<td>OFF</td>
<td>/Qip</td>
</tr>
<tr>
<td>-ip-no-inlining</td>
<td>Disables full and partial inlining (requires -ip or -ipo).</td>
<td>OFF</td>
<td>/Qip-no-inlining</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>-ip-no-pinlining</td>
<td>Disables partial inlining (requires -ip or -ipo).</td>
<td>OFF</td>
<td>/Qip-no-pinlining (i32, i64em)</td>
</tr>
<tr>
<td>-IPF-flt-eval-method0</td>
<td>Evaluates floating-point operands evaluated to the precision indicated by program; deprecated.</td>
<td>OFF</td>
<td>/QIPF-flt-eval-method0 (i64 only)</td>
</tr>
<tr>
<td>-[no-]IPF-fltacc</td>
<td>Enables or disables optimizations that affect floating point accuracy; deprecated.</td>
<td>-no-IPF-fltacc</td>
<td>/QIPF-fltacc[-] (i64 only)</td>
</tr>
<tr>
<td>-[no-]IPF-fma</td>
<td>Enables or disable the combining of floating</td>
<td>-IPF-fma</td>
<td>/QIPF-fma[-] (i64 only)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-[no-]IPF-fp-relaxed</td>
<td>Enables or disables use of faster but slightly less accurate code sequences for math functions; deprecated, use -fp-relaxed.</td>
<td></td>
<td>/QIPF-fp-relaxed[[-</td>
</tr>
<tr>
<td></td>
<td>(i64 only, Linux only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-ipo[=n]</td>
<td>Enables multi-file IP optimization s (between files).</td>
<td>OFF</td>
<td>/Qipo[n]</td>
</tr>
<tr>
<td>-ipo-c</td>
<td>Generates a multi-file object file</td>
<td>OFF</td>
<td>/Qipo-c</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-ipo-jobs&lt;n&gt;</td>
<td>Specifies the number of commands to be executed simultaneously during the link phase of Interprocedural Optimization (IPO).</td>
<td>OFF</td>
<td>/Qipo-jobs:&lt;n&gt;</td>
</tr>
<tr>
<td>-ipo-S</td>
<td>Generates a multi-file assembly file (ipo_out.s).</td>
<td>OFF</td>
<td>/Qipo-S</td>
</tr>
<tr>
<td>-ipo-separate</td>
<td>Creates one object file for every source file.</td>
<td>OFF</td>
<td>/Qipo-separate</td>
</tr>
<tr>
<td>-iprefix &lt;prefix&gt;</td>
<td>Indicates the prefix for referencing</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>-iquote &lt;dir&gt;</td>
<td>Adds directory for files included with quotes to front of include files search path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-isystem&lt;dir&gt;</td>
<td>Specifies a directory to add to the system include path.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-ivdep-parallel</td>
<td>Makes ivdep directives mean no loop carried dependencies.</td>
<td>OFF</td>
<td>/Qivdep-parallel (i64 only)</td>
</tr>
<tr>
<td></td>
<td>(i64 only; Linux only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-iwithprefix&lt;dir&gt;</td>
<td>Appends &lt;dir&gt; to prefix</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------</td>
</tr>
<tr>
<td>-iprefix</td>
<td>passed in by -iprefix and puts it at the end of the include search path.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-iwiprefixbefore</td>
<td>Similar to -iprefix except include directory is placed in the same place as -I directories.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Kc++</td>
<td>Compiles all source or unrecognized file types as C++ source files.</td>
<td>OFF</td>
<td>/TP</td>
</tr>
<tr>
<td>-kernel</td>
<td>(i64 only; Linux only) Generates code for inclusion in the kernel.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-1&lt;string&gt;</td>
<td>Tells the</td>
<td>OFF</td>
<td>None</td>
</tr>
</tbody>
</table>

1158
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
<th>Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>-L&lt;dir&gt;</td>
<td>linker to search for a specified library.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-m&lt;processor&gt;</td>
<td>Tells the compiler to generate optimized code specialized for the processor that executes your program.</td>
<td>OFF</td>
<td>/arch:&lt;processor&gt;</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
<td></td>
</tr>
<tr>
<td>-M</td>
<td>Generates makefile dependency information.</td>
<td>OFF</td>
<td>/QM</td>
<td></td>
</tr>
<tr>
<td>-m32 (i32, i64em)</td>
<td>Tells the compiler to generate code for IA-32 architecture.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-m64 (i32, i64em)</td>
<td>Tells the compiler to generate code for Intel® 64 architecture.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-malign-double (i32, i64em)</td>
<td>Aligns double, long double, and long long types for systems based on IA-32 architecture.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-malign-mac68k (i32, i64em; Mac OS X)</td>
<td>Aligns structure</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td>-malign-natural</td>
<td>Aligns larger OFF types on natural size-based boundaries (overrides ABI).</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-malign-power</td>
<td>Aligns based on ABI-specified alignment rules.</td>
<td>ON</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-map-opts</td>
<td>Enables option mapping tool.</td>
<td>OFF</td>
<td>/Qmap-opts</td>
<td></td>
</tr>
<tr>
<td>-march=&lt;processor&gt;</td>
<td>Generates code for a specified processor.</td>
<td>i32: OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>i64em:</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>march=pentium</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>m4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uses a</td>
<td>-</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
<td></td>
</tr>
<tr>
<td>mcmodel=&lt;mem_model&gt;</td>
<td>specific memory model to generate code and store data.</td>
<td>OFF</td>
<td>mcmodel=small</td>
<td></td>
</tr>
<tr>
<td>(i64em; Linux only)</td>
<td></td>
<td>mcmodel=small</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-MD</td>
<td>Preprocesses and compiles, generating output file containing dependency information ending with extension .d.</td>
<td>OFF</td>
<td>/QMD</td>
<td></td>
</tr>
<tr>
<td>-mdynamic-no-pic</td>
<td>Generates code that is not position-independent but has position-independent external references.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>(i32; Mac OS X only)</td>
<td></td>
<td>None</td>
<td>/QMF&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-MF&lt;file&gt;</td>
<td>Generates</td>
<td>OFF</td>
<td>/QMF&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td>makefile</td>
<td>dependency information in file (must specify -M or -MM).</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-mfixed-range (i64; Mac OS X only)</td>
<td>Reserves certain register for use by the Linux* kernel.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-MG</td>
<td>Similar to -M, but treat missing header files as generated files.</td>
<td>OFF</td>
<td>/QMG</td>
<td></td>
</tr>
<tr>
<td>-minstruction=[no]movbe (i32, i64em)</td>
<td>Enables or disables generation of MOVBE instructions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-MM</td>
<td>Similar to -M, but do not include</td>
<td>OFF</td>
<td>/QMM</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td><code>-MMD</code></td>
<td>Similar to <code>-MD</code>, but do not include system header files.</td>
<td>OFF</td>
<td><code>/QMMD</code></td>
<td></td>
</tr>
<tr>
<td><code>-mp</code></td>
<td>Maintains floating-point precision while disabling some optimization.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td><code>-MP</code></td>
<td>Adds a phony target for each dependency.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td><code>-mp1</code></td>
<td>Improves precision (speed impact is less than <code>-mp</code>).</td>
<td>OFF</td>
<td><code>/Qprec</code></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
<td></td>
</tr>
<tr>
<td>-MQ&lt;target&gt;</td>
<td>Changes the default target rule for dependency generation.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-mregparm=&lt;value&gt;</td>
<td>Controls the number of registers used to pass integer argument.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-m[no-]relax</td>
<td>Passes linker option -relax to the linker.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-m[no-]serialize-volatile</td>
<td>Imposes strict memory access ordering for volatile data object references.</td>
<td>OFF</td>
<td>/Qserialize-volatile[-]</td>
<td></td>
</tr>
<tr>
<td>-MT&lt;target&gt;</td>
<td>Changes the default target rule</td>
<td>OFF</td>
<td>/QMT&lt;target&gt;</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>--------------------------------------------------</td>
<td>-----------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>-mtune=&lt;processor&gt;</td>
<td>Performs optimization for specific processors.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>i32, i64em: -mtune=generic</td>
<td>None</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>i64: -mtune=itanium m2-p9000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]multibyte-chars</td>
<td>Provides support for multi-byte characters</td>
<td>ON</td>
<td>/Qmultibyte-chars[-]</td>
<td></td>
</tr>
<tr>
<td>-multiple-processes[=&lt;n&gt;]</td>
<td>Creates multiple processes.</td>
<td>OFF</td>
<td>/MP[&lt;n&gt;]</td>
<td></td>
</tr>
<tr>
<td>-no-bss-init</td>
<td>Disable placement of zero-initialized variables</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>in BSS (use DATA).</td>
<td>OFF</td>
<td>/Qnobss-init</td>
<td></td>
</tr>
<tr>
<td>-nodefaultlibs</td>
<td>Prevents using standard</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
<td></td>
</tr>
<tr>
<td>-nolib-inline</td>
<td>Disables inline expansion of standard library or intrinsic functions.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-nostartfiles</td>
<td>Prevents the compiler from using standard startup files when linking.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-nostdinc++</td>
<td>Do not search for header files in the standard directories for C++.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>-nostdlib</td>
<td>Prevents the compiler from using standard libraries when linking.</td>
<td>OFF</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
<td></td>
</tr>
<tr>
<td>standard libraries and startup files when linking.</td>
<td>-o&lt;file&gt; Specifies name of output file.</td>
<td>OFF</td>
<td>/Fe&lt;file&gt;</td>
<td></td>
</tr>
<tr>
<td>-0</td>
<td>Enables optimization s.</td>
<td>-02</td>
<td>/O</td>
<td></td>
</tr>
<tr>
<td>-00</td>
<td>Disables optimization s.</td>
<td>-02</td>
<td>/Od</td>
<td></td>
</tr>
<tr>
<td>-openmp</td>
<td>Generates multi-threaded code based on the OpenMP* directives</td>
<td>OFF</td>
<td>/Qopenmp</td>
<td></td>
</tr>
<tr>
<td>-openmp-lib &lt;type&gt;</td>
<td>(Linux only) Lets you specify an OpenMP* run-time library to use for</td>
<td>-openmp-lib legacy</td>
<td>/Qopenmp-lib: type</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>--------------</td>
<td>------------------------------</td>
<td></td>
</tr>
<tr>
<td>-openmp-link</td>
<td>Links to static or dynamic OpenMP run-time libraries.</td>
<td>-openmp-link dynamic</td>
<td>/Qopenmp-link:&lt;library&gt;</td>
<td></td>
</tr>
<tr>
<td>-openmp-profile (Linux only)</td>
<td>Links with instrumented OpenMP run-time library to generate OpenMP profiling information.</td>
<td>OFF</td>
<td>/Qopenmp-profile</td>
<td></td>
</tr>
<tr>
<td>-openmp-report&lt;n&gt;</td>
<td>Controls the OpenMP parallelizer diagnostic level.</td>
<td>-openmp-report1</td>
<td>/Qopenmp-report&lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>-openmp-stubs</td>
<td>Enables the user to compile OpenMP programs in sequential linking.</td>
<td>OFF</td>
<td>/Qopenmp-stubs</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS</td>
<td></td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------------------------------------</td>
<td>-------------</td>
<td>------------------------</td>
<td></td>
</tr>
<tr>
<td>-openmp-task &lt;model&gt;</td>
<td>Lets you choose an OpenMP* tasking model.</td>
<td>-openmp-task omp</td>
<td>/Qopenmp-task:&lt;model&gt;</td>
<td></td>
</tr>
<tr>
<td>-openmp-threadprivate &lt;type&gt;</td>
<td>Lets you specify an OpenMP* threadprivate implementation.</td>
<td>-openmp-threadprivate legacy</td>
<td>/Qopenmp-threadprivate:&lt;type&gt;</td>
<td></td>
</tr>
<tr>
<td>-opt-block-factor=&lt;n&gt;</td>
<td>Lets you specify a loop blocking factor.</td>
<td>OFF</td>
<td>/Qopt-block-factor:&lt;n&gt;</td>
<td></td>
</tr>
<tr>
<td>-[no]opt-calloc (i32, i64em; Linux only)</td>
<td>Tells the compiler to substitute a call to _intel_fastcalloc() for a call to calloc().</td>
<td>-no-opt-calloc</td>
<td>/Qopenmp-stubs</td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-class-</td>
<td>Tells the</td>
<td>-no-opt-class-</td>
<td>/Qopt-class-</td>
<td></td>
</tr>
</tbody>
</table>

1170
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>analysis</td>
<td>compiler to use C++ class hierarchy information to analyze and resolve C++ virtual function calls at compile time.</td>
<td>class-</td>
<td>analysis[-]</td>
</tr>
<tr>
<td>-[no-]opt-jump-tables=keyword</td>
<td>Enables or disables generation of jump tables for switch statements.</td>
<td>- opt-jump-tables=default</td>
<td>-opt-jump-tables[-]</td>
</tr>
<tr>
<td>-[no-]opt-loadpair</td>
<td>Enables or disables loadpair optimization.</td>
<td>-no-opt-loadpair</td>
<td>-opt-loadpair[-]</td>
</tr>
<tr>
<td>(i64 only; Linux only)</td>
<td></td>
<td>loadpair</td>
<td>(i64 only)</td>
</tr>
<tr>
<td>-[no-]opt-malloc-options=&lt;n&gt;</td>
<td>Lets you specify an alternate</td>
<td>-opt-malloc-</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td></td>
<td>options=0</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td><code>-opt-mem-bandwidth&lt;n&gt;</code> (i64 only; Linux only)</td>
<td>Enables performance tuning and heuristics to control memory bandwidth use among processors.</td>
<td><code>-opt-mem-bandwidth0</code> /<code>Qopt-mem-bandwidth&lt;n&gt;</code> (i64 only)</td>
<td></td>
</tr>
<tr>
<td><code>-[no-]opt-mod-versioning</code> (i64 only; Linux only)</td>
<td>Enables or disables versioning of modulo operations for certain types of operands.</td>
<td><code>- no-opt-mod-versioning</code> /<code>Qopt-mod-versioning[-]</code> (i64 only)</td>
<td></td>
</tr>
<tr>
<td><code>-[no-]opt-multi-version-aggressive</code> (i32, i64em)</td>
<td>Tells the compiler to use aggressive multi-versioning to check for pointer</td>
<td><code>-no-opt-multi-version-aggressive</code> /<code>Qopt-multi-version-aggressive[-]</code> (i32, i64em)</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-[no-]opt-prefetch[=&lt;n&gt;]</td>
<td>Enables or disables aliasing and scalar replacement</td>
<td>i32, i64em: -no- /Qopt-prefetch[prefetch[:&lt;n&gt;]]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>prefetch insert optimization</td>
<td>i64: -opt-prefetch</td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-prefetch-initial-values (i64 only; Linux only)</td>
<td>Enables or disables prefetches that are issued before a loop is entered.</td>
<td>-no-opt-prefetch-initial-values [-] (i64 only)</td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-prefetch-issue-excl-hint (i64 only; Linux only)</td>
<td>Enables or disables prefetches for stores with exclusive hint</td>
<td>-no-opt-prefetch-issue-excl-hint [-] (i64 only)</td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-prefetch-next-iteration (i64 only)</td>
<td>Enables or disables prefetches</td>
<td>-no-opt-prefetch-next-iteration [-] (i64 only)</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>(i64 only; Linux only) for a memory access in the next iteration of a loop.</td>
<td>iteration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]opt-ra-region-strategy[=keyword] (i32, i64em)</td>
<td>Selects the method that the register allocator uses to partition each routine into regions.</td>
<td>-opt-ra-region-strategy=default</td>
<td>/Qopt-ra-region-strategy[:keyword]</td>
</tr>
<tr>
<td>-opt-report [&lt;n&gt;] (i32, i64em)</td>
<td>Generates an optimization report to stderr.</td>
<td>-opt-report 2</td>
<td>/Qopt-report[:&lt;n&gt;]</td>
</tr>
<tr>
<td>-opt-report-file=&lt;file&gt; (i32, i64em)</td>
<td>Specifies the file name for the generated report.</td>
<td>OFF</td>
<td>/Qopt-report-file:&lt;file&gt;</td>
</tr>
<tr>
<td>-opt-report-help (i32, i64em)</td>
<td>Displays the optimization</td>
<td>OFF</td>
<td>/Qopt-report-help</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-opt-report-phase=&lt;phase&gt;</td>
<td>Specifies the phase that reports are generated against.</td>
<td>OFF</td>
<td>/Qopt-report-phase:&lt;phase&gt;</td>
</tr>
<tr>
<td>-opt-report-routine=&lt;string&gt;</td>
<td>Reports on routines containing the given name.</td>
<td>OFF</td>
<td>/Qopt-report-routine:&lt;string&gt;</td>
</tr>
<tr>
<td>-opt-streaming-stores keyword (i32, i64em)</td>
<td>Enables generation of streaming stores for optimization.</td>
<td>-opt-</td>
<td>/Qopt-streaming-stores:keyword(i32, i64em)</td>
</tr>
<tr>
<td></td>
<td>-[no-]opt-subscript-in-range (i32, i64em)</td>
<td>Enables or disables overflows in the intermediate computation of subscript</td>
<td>-[no-]opt-subscript-in-range[-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-Os</td>
<td>Enables speed optimization in loops, but disable some optimization which increase code size for small speed benefit.</td>
<td>OFF</td>
<td>/Os</td>
</tr>
<tr>
<td>-p</td>
<td>Compiles and links for function profiling with gprof(1).</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-P (or -F)</td>
<td>Preprocesses to file omitting #line directives.</td>
<td>OFF</td>
<td>/P</td>
</tr>
<tr>
<td>-par-report[n]</td>
<td>control the auto-</td>
<td>-par-report1 /Qpar-report[n]</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>parallelizer</td>
<td>diagnostic level</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]par-runtime-control</td>
<td>Generates code to perform runtime checks for loops that have symbolic loop</td>
<td>-no-par-runtime-control</td>
<td>/Qpar-runtime-control[-]</td>
</tr>
<tr>
<td></td>
<td>bounds.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-par-schedule-&lt;keyword&gt;[=n]</td>
<td>Specifies a scheduling algorithm for DO loop iterations.</td>
<td>OFF</td>
<td>/Qpar-schedule-&lt;keyword&gt;[=n]</td>
</tr>
<tr>
<td>-par-threshold[&lt;n&gt;]</td>
<td>Sets threshold for the auto-parallelization of loops.</td>
<td>threshold100</td>
<td>/Qpar-threshold[&lt;n&gt;]</td>
</tr>
<tr>
<td>-parallel</td>
<td>Enables the auto-parallelizer to generate multi-</td>
<td>OFF</td>
<td>/Qparallel</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>threaded</td>
<td>code for loops that can be safely executed in parallel.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-pc&lt;n&gt;</td>
<td>Enables control of floating-point significand precision.</td>
<td>-pc80</td>
<td>/Qpc&lt;n&gt;</td>
</tr>
<tr>
<td>-pch</td>
<td>Enable automatic precompiled header file creation/usage.</td>
<td>OFF</td>
<td>/YX</td>
</tr>
<tr>
<td>-pch-create &lt;file&gt;</td>
<td>Creates precompiled header files.</td>
<td>OFF</td>
<td>/Yc[file]</td>
</tr>
<tr>
<td>-pch-dir &lt;dir&gt;</td>
<td>Tells the compiler where to find or create a file</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-pch-use {&lt;file&gt;</td>
<td>&lt;dir&gt;}</td>
<td>Lets you use a specific precompiled header file.</td>
<td>OFF</td>
</tr>
<tr>
<td>-pie</td>
<td>Produces a position-independent executable on processors that support it.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-pragma-optimization-level=&lt;interpretation&gt;</td>
<td>Specifies which interpretation of the optimization _level pragma should be used if no prefix is specified.</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-[no-]prec-div</td>
<td>Improve precision of floating-point divides (some speed impact).</td>
<td>-prec-div</td>
<td>/Qprec-div[-]</td>
</tr>
<tr>
<td>-[no-]prec-sqrt</td>
<td>Determine if certain square root optimizations are enabled.</td>
<td>-no-prec-sqrt</td>
<td>/Qprec-sqrt[-]</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td>sqrt</td>
<td>(i32, i64em)</td>
</tr>
<tr>
<td>-print-multi-lib</td>
<td>Prints information about where system libraries should be found.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-[no-]prof-data-order (Linux only)</td>
<td>Enables or disables data ordering if profiling information is enabled.</td>
<td>-no-prof-data-order</td>
<td>/Qprof-data-order[-]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>-prof-dir &lt;dir&gt;</code></td>
<td>Specifies directory for profiling output files (*.dyn and *.dpi).</td>
<td>OFF</td>
<td><code>/Qprof-dir &lt;dir&gt;</code></td>
</tr>
<tr>
<td><code>-prof-file &lt;file&gt;</code></td>
<td>Specifies file name for profiling summary file.</td>
<td>OFF</td>
<td><code>/Qprof-file &lt;file&gt;</code></td>
</tr>
<tr>
<td><code>[no-]prof-func-groups</code> (i32, i64em; Linux only)</td>
<td>Enables or disables function grouping if profiling information is enabled.</td>
<td>-no-prof-func-groups</td>
<td>None</td>
</tr>
<tr>
<td><code>[no-]prof-func-order</code></td>
<td>Enables or disables function ordering if profiling information is enabled.</td>
<td>-no-prof-func-order</td>
<td>/Qprof-func-order [-]</td>
</tr>
<tr>
<td><code>-prof-gen[x]</code></td>
<td>Instruments program for</td>
<td>OFF</td>
<td><code>/Qprof-gen[x]</code></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-prof-hotness-threshold=&lt;n&gt; (Linux only)</td>
<td>Lets you set the hotness threshold for function grouping and function ordering.</td>
<td>OFF</td>
<td>/Qprof-hotness-threshold:&lt;n&gt;</td>
</tr>
<tr>
<td>-[no-]prof-src-dir</td>
<td>Determines whether directory information of the source file under compilation is considered when looking up profile data records.</td>
<td>OFF</td>
<td>/Qprof-src-dir [-]</td>
</tr>
<tr>
<td>-prof-src-root=&lt;dir&gt;</td>
<td>Lets you use relative directory paths when looking up profile data records.</td>
<td>OFF</td>
<td>/Qprof-src-root=&lt;dir&gt;</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-prof-src-root-cwd</td>
<td>Lets you use relative directory paths when looking up profile data and specifies the current working directory as the base.</td>
<td>OFF</td>
<td>/Qprof-src-root-cwd</td>
</tr>
<tr>
<td>-prof-use</td>
<td>Enables use of profiling information during optimization.</td>
<td>OFF</td>
<td>/Qprof-use</td>
</tr>
<tr>
<td>-pthread</td>
<td>Uses pthreads library for multithreading.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-Qlocation,&lt;string&gt;,&lt;dir&gt;</td>
<td>Sets &lt;dir&gt; as the location of tool specified by &lt;string&gt;; supported tools depend on the operating system.</td>
<td>OFF</td>
<td>/Qlocation,&lt;string&gt;,&lt;dir&gt;</td>
</tr>
<tr>
<td>-Qoption,&lt;string&gt;,&lt;options&gt;</td>
<td>Passes &lt;options&gt; to tool specified by &lt;string&gt;; supported tools depend on the operating system.</td>
<td>OFF</td>
<td>/Qoption,&lt;string&gt;,&lt;options&gt;</td>
</tr>
<tr>
<td>-rcd (i32, i64em)</td>
<td>Sets rounding</td>
<td>OFF</td>
<td>/Qrcd (i32, i64em)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>mode to enable fast float-to-int conversions.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-rct</code> (i32, i64em)</td>
<td>Sets the internal FPU rounding control to Truncate.</td>
<td>OFF</td>
<td><code>/Qrct</code> (i32, i64em)</td>
</tr>
<tr>
<td><code>-reserve-kernel-regs</code> (i64 only; Linux only)</td>
<td>Reserves f12-f15 and f32-f127 for use by the kernel.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-[no]restrict</code></td>
<td>Enables or disables the 'restrict' keyword for disambiguating pointers.</td>
<td>OFF</td>
<td><code>/Qrestrict[-]</code></td>
</tr>
<tr>
<td><code>-S</code></td>
<td>Compiles to assembly (.s) only, do not link (*I).</td>
<td>OFF</td>
<td><code>/S</code></td>
</tr>
<tr>
<td><code>-[no-]save-temps</code></td>
<td>Tells the compiler to save temporary files.</td>
<td><code>-no-save-</code></td>
<td><code>/Qsave-temps[-]</code></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>compiler to save temps</td>
<td>Intermediate files created during compilation.</td>
<td>temps</td>
<td></td>
</tr>
<tr>
<td>-[no-]scalar-rep</td>
<td>Enables or disables scalar replacement performed during loop transformation.</td>
<td>-no-scalar-rep</td>
<td>/Qscalar-rep[-]</td>
</tr>
<tr>
<td>-shared</td>
<td>Produces a dynamic shared object instead of an executable.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-shared-intel</td>
<td>Causes Intel-provided libraries to be linked in dynamically.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>-shared-libgcc</td>
<td>Links the GNU libgcc library dynamically.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>(Linux only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]sox</td>
<td>Enables or disables saving of compiler options and version in the executable.</td>
<td>-no-sox</td>
<td>/Qsox[-]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-static</td>
<td>Prevents linking with shared libraries.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>(Linux only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-staticlib</td>
<td>Invokes the libtool command to generate static libraries.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td></td>
<td>(Mac OS X only)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-static-intel</td>
<td>Causes Intel-provided libraries to be linked in</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-static-libgcc</td>
<td>Links the GNU libgcc library</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>(Linux only)</td>
<td>statically.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-std=&lt;val&gt;</td>
<td>Conforms to a specific language standard.</td>
<td>OFF</td>
<td>/Qstd:&lt;val&gt;</td>
</tr>
<tr>
<td>-strict-ansi</td>
<td>Strict ANSI conformance dialect.</td>
<td>OFF</td>
<td>/Za</td>
</tr>
<tr>
<td>-T &lt;file&gt;</td>
<td>Tells the linker to read link commands from a file.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-tcheck</td>
<td>Generates instrumentation to detect multithreading bugs.</td>
<td>OFF</td>
<td>/Qtcheck</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-tcollect [&lt;lib&gt;]</td>
<td>Inserts instrumentation probes</td>
<td>OFF</td>
<td>/Qtcollect[:&lt;lib&gt;]</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-tcollect-filter &lt;file&gt;</td>
<td>Enables or disables the instrumentation of specified functions.</td>
<td>OFF</td>
<td>/Qtcollect-filter[:&lt;file&gt;]</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-tprofile</td>
<td>Generates instrumentation to analyze multithreading performance.</td>
<td>OFF</td>
<td>/Qtprofile</td>
</tr>
<tr>
<td>(Linux only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no]traceback</td>
<td>Specifies whether the compiler generates data to allow for source file traceback information</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>-U&lt;name&gt;</td>
<td>remove predefined macro</td>
<td>OFF</td>
<td>/U&lt;name&gt;</td>
</tr>
<tr>
<td>-unroll[n]</td>
<td>Set maximum number of times to unroll loops.</td>
<td>-unroll</td>
<td>/Qunroll[:n]</td>
</tr>
<tr>
<td>-[no-]unroll-aggressive (i32, i64em)</td>
<td>Tells the compiler to use aggressive, complete unrolling for loops with small constant trip counts.</td>
<td>OFF</td>
<td>/Qunroll-aggressive[-] (i32, i64em)</td>
</tr>
<tr>
<td>-[no-]use asm</td>
<td>Produces objects through assembler.</td>
<td>-no-use-asm</td>
<td>/Quse-asm[-] (i32 only)</td>
</tr>
<tr>
<td>-use-msasm</td>
<td>Accepts the Microsoft MASM-style inlined</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------------------</td>
<td>--------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>assembly</td>
<td>assembly format.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-V</td>
<td>Displays compiler version</td>
<td>OFF</td>
<td>/QV</td>
</tr>
<tr>
<td></td>
<td>information.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]vec</td>
<td>Enables or disables vectorization.</td>
<td>-vec</td>
<td>/Qvec[-]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-[no-]vec-guard-write</td>
<td>Tells the compiler to perform a conditional check in a vectorized loop.</td>
<td>-no-vec-guard-write</td>
<td>/Qvec-guard-write (i32, i64em)</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
<td>(i32, i64em)</td>
</tr>
<tr>
<td>-vec-report[&lt;n&gt;]</td>
<td>Controls amount of vectorizer diagnostic information-opt-report generate an optimization report to stderr.</td>
<td>-vec-report1</td>
<td>/Qvec-report[&lt;n&gt;] (i32, i64em)</td>
</tr>
<tr>
<td>(i32, i64em)</td>
<td></td>
<td></td>
<td>(i32, i64em)</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------</td>
<td>------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>--version</td>
<td>Displays GCC-style version information.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-w</td>
<td>Disables all warning messages.</td>
<td>OFF</td>
<td>/w</td>
</tr>
<tr>
<td>-w&lt;n&gt;</td>
<td>Determines which diagnostic message level is set.</td>
<td>OFF</td>
<td>/W&lt;n&gt;</td>
</tr>
<tr>
<td>-Wa, &lt;option1&gt;[,&lt;option2&gt;,...]</td>
<td>Passes options to the assembler for processing.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]abi</td>
<td>Determines whether a warning is issued if generated code is not C++ ABI compliant.</td>
<td>-Wno-abi</td>
<td>None</td>
</tr>
<tr>
<td>-Wall</td>
<td>Enables all warning messages.</td>
<td>OFF</td>
<td>/Wall</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-Wbrief</td>
<td>Prints brief one-line diagnostics.</td>
<td>OFF</td>
<td>/WL</td>
</tr>
<tr>
<td>-Wcheck</td>
<td>Enables compile-time code checking for certain code.</td>
<td>OFF</td>
<td>/Wcheck</td>
</tr>
<tr>
<td>-W[no-]comment</td>
<td>Determines whether a warning is issued when /* appears in the middle of a */ */ comment.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Wcontext-limit=&lt;n&gt;</td>
<td>Sets maximum number of template instantiation contexts shown in diagnostic.</td>
<td>OFF</td>
<td>/Qcontext-limit:&lt;n&gt;</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>--------------</td>
<td>---------------------------------------</td>
<td>---------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>-wd&lt;L1&gt;,&lt;Ln&gt;,...</td>
<td>Disables diagnostics L1 through Ln.</td>
<td>OFF</td>
<td>/Qwd&lt;L1&gt;,&lt;Ln&gt;,...</td>
</tr>
<tr>
<td>-W[no-]deprecated</td>
<td>Determines whether warnings are issued for deprecated features.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-we&lt;L1&gt;,&lt;Ln&gt;,...</td>
<td>Changes severity of soft diagnostics L1 through Ln to error.</td>
<td>OFF</td>
<td>/Qwe&lt;L1&gt;,&lt;Ln&gt;,...</td>
</tr>
<tr>
<td>-Weffc++</td>
<td>Enables warnings based on certain C++ programming guidelines.</td>
<td>OFF</td>
<td>/Qeffc++</td>
</tr>
<tr>
<td>-Werror</td>
<td>Forces warnings to be reported</td>
<td>OFF</td>
<td>/WX</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-Werror-all</td>
<td>Changes all warnings and remarks to errors.</td>
<td>OFF</td>
<td>/Werror-all</td>
</tr>
<tr>
<td>-W[no-]extra-tokens</td>
<td>Determines whether warnings are issued about extra tokens at the end of preprocessors directives.</td>
<td>-Wno-extra-tokens</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]format</td>
<td>Determines whether argument checking is enabled for calls to printf, scanf, and so forth.</td>
<td>-Wno-format</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]format-security</td>
<td>Determines whether the compiler issues a warning</td>
<td>-Wno-format-security</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td></td>
<td>when the use of format functions may cause security problems.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-Winline</td>
<td>Enables diagnostics about what is inlined and what is not inlined.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Wl,&lt;option1&gt;[,&lt;option2&gt;,...]</td>
<td>Passes options to the linker for processing.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]main</td>
<td>Determines whether a warning is issued if the return type of main is not expected.</td>
<td>-Wno-main</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]missing-declaration</td>
<td>Determines whether missing-</td>
<td>-Wno-</td>
<td>None</td>
</tr>
</tbody>
</table>

1196
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
<th>Equivalent Windows OS Option</th>
</tr>
</thead>
<tbody>
<tr>
<td>warnings</td>
<td>declarations are issued for global functions and variables without prior declaration.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-W[no-]missing-prototype</td>
<td>Determines whether warnings are issued for missing prototypes.</td>
<td>-Wno-</td>
<td>None</td>
</tr>
<tr>
<td>-Wnon-virtual-dtor</td>
<td>Issues a warning when a class appears to be polymorphic, yet it declares a non-virtual one. (C++ only.)</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-wn&lt;n&gt;</td>
<td>Prints a</td>
<td>-wn100</td>
<td>/Qwn&lt;n&gt;</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-wo&lt;L1&gt;[,&lt; Ln&gt;,...]</td>
<td>Issues one or more diagnostic messages only once.</td>
<td>OFF</td>
<td>/Qwo&lt;L1&gt;[,&lt; Ln&gt;,...]</td>
</tr>
<tr>
<td>-Wp,&lt;option1&gt;[,&lt;option2&gt;,...]</td>
<td>Passes options to the preprocessor.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Wp64</td>
<td>Prints diagnostics for 64-bit porting.</td>
<td>OFF</td>
<td>/Wp64</td>
</tr>
<tr>
<td>-W[no-]pointer-arith</td>
<td>Determines whether warnings are issued for questionable pointer arithmetic.</td>
<td>-Wno-pointer-arith</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]pragma-once</td>
<td>Determines whether a once</td>
<td>-Wno-prag once</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td><code>warning is issued about the use of #pragma once.</code></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>-wr&lt;L1&gt;[,&lt; Ln&gt;,...]</code></td>
<td>Changes severity of soft diagnostics L1 through Ln to remark.</td>
<td>OFF</td>
<td><code>/Qwr&lt;L1&gt;[,&lt; Ln&gt;,...]</code></td>
</tr>
<tr>
<td><code>-Wreorder</code></td>
<td>Issues a warning when the order of member initializers does not match the order in which they must be executed.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td><code>-W[no-]return-type</code></td>
<td>Determines whether type</td>
<td>-Wno-return-</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td>warnings</td>
<td>are issued when a function uses the default int return type.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-W[no-]shadow</td>
<td>Determines whether a warning is issued when a variable declaration hides a previous declaration.</td>
<td>-Wno-shadow None</td>
<td></td>
</tr>
<tr>
<td>-W[no-]strict-prototypes</td>
<td>Determines whether warnings are issued for functions declared or defined without specified argument types.</td>
<td>-Wno-strict- None</td>
<td></td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-W[no-]trigraphs</td>
<td>Determines whether warnings are issued if any trigraphs are encountered.</td>
<td>-Wno-trigraphs</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]uninitialized</td>
<td>Determines whether a warning is issued if a variable is used before being initialized.</td>
<td>-Wno-uninitialized</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]unknown-pragmas</td>
<td>Determines whether a warning is issued if an unknown #pragma directive is used.</td>
<td>-Wno-unknown-pragmas</td>
<td>None</td>
</tr>
<tr>
<td>-W[no-]unused-function</td>
<td>Determines whether a function is unused.</td>
<td>-Wno-unused-function</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-W[no-]unused-variable</td>
<td>Determines whether a warning is issued if a local or non-constant static variable is unused after being declared.</td>
<td>-Wno-unused-variable None</td>
<td>/Qww&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
</tr>
<tr>
<td>-ww&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
<td>Changes soft diagnostics L1 through Ln to warning.</td>
<td>OFF</td>
<td>/Qww&lt;L1&gt;[,&lt;Ln&gt;,...]</td>
</tr>
<tr>
<td>-Wwrite-strings</td>
<td>Issues a diagnostic message if const char * is converted</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
<td>Equivalent Windows OS Option</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>-x&lt;processor&gt; (i32, i64em)</td>
<td>Generates optimized code specialized for the Intel processor that executes your program.</td>
<td>varies; see option description</td>
<td>/Qx&lt;processor&gt; (i32, i64em)</td>
</tr>
<tr>
<td>-X</td>
<td>Removes standard directories from include file search path.</td>
<td>OFF</td>
<td>/X</td>
</tr>
<tr>
<td>-Xlinker &lt;option&gt;</td>
<td>Passes a linker option directly to the linker.</td>
<td>OFF</td>
<td>None</td>
</tr>
<tr>
<td>-Zp[n]</td>
<td>Specifies alignment for structures.</td>
<td>-Zp16</td>
<td>/Zp[n]</td>
</tr>
</tbody>
</table>
Option | Description | Default | Equivalent Windows OS Option
--- | --- | --- | ---
 | on byte boundaries. | | | 

**See Also**

map-opts, Qmap-opts compiler option

**Related Options**

This topic lists related options that can be used under certain conditions.

**Cluster OpenMP* Options (Linux* OS only)**

The Cluster OpenMP* (CLOMP or Cluster OMP) options are available if you have a separate license for the Cluster OpenMP product. These options can be used on Linux* operating systems running on Intel® 64 and IA-64 architectures.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-[no-]cluster-openmp</td>
<td>Lets you run an OpenMP program on a cluster.</td>
</tr>
<tr>
<td>-[no-]cluster-openmp-profile</td>
<td>Links a Cluster OMP program with profiling information.</td>
</tr>
<tr>
<td>-[no-]clomp-sharable-propagation</td>
<td>Reports variables that need to be made sharable by the user with Cluster OpenMP.</td>
</tr>
<tr>
<td>-[no-]clomp-sharable-info</td>
<td>Reports variables that the compiler automatically makes sharable for Cluster OpenMP.</td>
</tr>
</tbody>
</table>
For more information on these options, see the Cluster OpenMP documentation.

**Linking Tools and Options**

This topic describes how to use the Intel® linking tools, `xild` (Linux* OS and Mac OS* X) or `xilink` (Windows* OS).

The Intel linking tools behave differently on different platforms. The following sections summarizes the primary differences between the linking behaviors.

### Linux and Mac OS Linking Behavior Summary

The linking tool invokes the compiler to perform IPO if objects containing IR (intermediate representation) are found. (These are mock objects.) It invokes GNU `ld` to link the application.

The command-line syntax for `xild` is the same as that of the GNU linker:

```
xild [<options>] <normal command-line>
```

where:

- `<options>`: (optional) one or more options supported only by `xild`.
- `<normal command-line>`: linker command line containing a set of valid arguments for `ld`.

To create `app` using IPO, use the option `-qfile` as shown in the following example:

```
xild -qipo_fas -oapp a.o b.o c.o
```

The linking tool calls the compiler to perform IPO for objects containing IR and creates a new list of object(s) to be linked. The linker then calls `ld` to link the object files that are specified in the new list and produce the application with the name specified by the `-o` option. The linker supports the `-ipo`, `-ipoN`, and `-ipo-separate` options.

### Windows Linking Behavior Summary

The linking tool invokes the Intel compiler to perform multi-file IPO if objects containing IR (intermediate representation) is found. These are mock objects. It
invokes Microsoft® *link.exe* to link the application.

The command-line syntax for the Intel® linker is the same as that of the Microsoft linker:

```
xilink [<options>] <normal command-line>
```

where:

- `[<options>]`: (optional) one or more options supported only by *xilink*.
- `<normal command-line>`: linker command line containing a set of valid arguments for the Microsoft linker.

To place the multifile IPO executable in *ipo_file.exe*, use the linker option `/out:file`; for example:

```
xilink -qipo_fas /out:ipo_file.exe a.obj b.obj c.obj
```

The linker calls the compiler to perform IPO for objects containing IR and creates a new list of object(s) to be linked. The linker calls Microsoft *link.exe* to link the object files that are specified in the new list and produce the application with the name specified by the `/out:file` linker option.

### Using the Linking Tools

You must use the Intel linking tools to link your application if the following conditions apply:

- Your source files were compiled with multifile IPO enabled. Multi-file IPO is enabled by specifying the `-ipo` (Linux and Mac OS X) or `/Qipo` (Windows) command-line option.

- You normally would invoke either the GNU linker (*ld*) or the Microsoft linker (*link.exe*) to link your application.

The following table lists the available, case-insensitive options supported by the Intel linking tools and briefly describes the behavior of each option:

<table>
<thead>
<tr>
<th>Linking Tools Option</th>
<th>Description</th>
</tr>
</thead>
</table>

1206
<table>
<thead>
<tr>
<th>Linking Tools Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qhelp</td>
<td>Lists the available linking tool options. Same as passing no option.</td>
</tr>
<tr>
<td>-qnoipo</td>
<td>Disables multi-file IPO compilation.</td>
</tr>
<tr>
<td>-qipo_fa{{file</td>
<td>dir/}}</td>
</tr>
<tr>
<td></td>
<td>• <strong>Linux and Mac OS X:</strong> <em>ipo_out.s</em></td>
</tr>
<tr>
<td></td>
<td>• <strong>Windows:</strong> <em>ipo_out.asm</em></td>
</tr>
<tr>
<td></td>
<td>If the Intel linking tool invocation results in multi-object compilation, either because the application is big or because the user explicitly instructed the compiler to generate multiple objects, the first .s (Linux and Mac OS X) or .asm (Windows) file takes its name from the <code>-qipo_fa</code> option. The compiler derives the names of subsequent .s (Linux and Mac OS X) or .asm (Windows) files by appending an incrementing number to the name, for example, <em>foo.asm</em> and <em>foo1.asm</em> for <em>ipo_faffoo.asm</em>. The same is true for the <code>-qipo_fo</code> option (listed below).</td>
</tr>
<tr>
<td>-qipo_fo{{file</td>
<td>dir/}}</td>
</tr>
<tr>
<td>Linking Tools Option</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>-qipo_fas</td>
<td>Add source lines to assembly listing.</td>
</tr>
<tr>
<td>-qipo_fac</td>
<td>Adds code bytes to the assembly listing.</td>
</tr>
<tr>
<td>-qipo_facs</td>
<td>Add code bytes and source lines to assembly listing.</td>
</tr>
<tr>
<td>-quseenv</td>
<td>Disables override of existing PATH, LIB, and INCLUDE variables.</td>
</tr>
<tr>
<td>-lib</td>
<td>Invokes librarian instead of linker.</td>
</tr>
<tr>
<td>-libtool</td>
<td>Mac OS X: Invokes <code>libtool</code> to create a library instead of <code>ld</code>.</td>
</tr>
<tr>
<td>-qv</td>
<td>Displays version information.</td>
</tr>
</tbody>
</table>

**See Also**

Optimizing Applications: Using IPO

**Portability Options**

A challenge in porting applications from one compiler to another is making sure there is support for the compiler options you use to build your application. The Intel® C++ Compiler supports many of the options that are valid on other compilers you may be using.
The following sections list compiler options that are supported by the Intel® C++ Compiler and the following:
- Microsoft® C++ compiler
- gcc® Compiler

Options that are unique to either compiler are not listed in these sections.

**Options Equivalent to Microsoft® C++ Options (Windows® OS)**

The following table lists compiler options that are supported by both the Intel® C++ Compiler and the Microsoft® C++ compiler.

```markdown
/arch:<SSE|SSE2>
/C
/c
/D<name>{=|#}<text>
/E
/EHa
/EHc
/EHs
/EP
/F<num>
/Fa[file]
/FA[scu]
/FC
/Fe<file>
/FI<file>
/Fm[file]
/Fo<file>
```
/fp:<model>
/Fp<file>
/FR[file]
/FR[file]
/G1
/G2
/GA
/Gd
/Ge
/GF
/Gh
/GH
/Gr
/GR[-]
/GS[-]
/Gs[num]
/GT
/GX[-]
/Gy[-]
/Gz
/GZ
/H<num>
/help
/I<dir>
/RTC
/RTCu
/showIncludes
/TC
/Tc<source file>
/TP
/Tp<source file>
/u
/U<name>
/V<string>
/vd
/vmb
/vmg
/vmm
/vms
/vmv
/w
/W<n>
/Wall
/wd<n>
/we<n>
/WL
/wp64
/WX
Options Equivalent to gcc* Options (Linux* OS)

The following table lists compiler options that are supported by both the Intel® C++ Compiler and the gcc* compiler.

-A
-ansi
-B
-C
-D
-dD
-dM
-dN
-Ed
-fargument-noalias
-fargument-noalias-global
-fdata-sections
-ffunction-sections
-fmudflap
-f[no-]builtin
-f[no-]common
-f[no-]freestanding
-f[no-]gnu-keywords
-fnoimplicit-inline-templates
-fnoimplicit-templates
-f[no-]inline
-f[no-]inline-functions
-f[no-]math-errno
-f[no-]operator-names
-f[no-]stack-protector
-f[no-]unsigned-bitfields
-fpack-struct
-fpermissive
-f PIC
-fpic
-freg-struct-return
-fshort-enums
-fsyntax-only
-ftime template-depth
-ftls-model=global-dynamic
-ftls-model=initial-exec
-ftls-model=local-dynamic
-ftls-model=local-exec
-funroll-loops
-funsigned-char
-fverbose-asm
-fvisibility=default
-fvisibility=hidden
-fvisibility=internal
-fvisibility=protected
-H
-help
-I
-idirafter
- imacros
-iprefix
-iwithprefix
-iwithprefixbefore
-l
-L
-M
-malign-double
-march
-mcpu
-MD
-MF
-MG
-MM
-MMD
-m[no-]ieee-fp
-MP
-mp
-MQ
-msse
-msse2
-msse3
-MT
-mtune
-nodefaultlibs
-nostartfiles
-nostdinc
-nostdinc++
-nostdlib
-o
-O
-O0
-O1
-O2
-O3
-Os
-P
-P
-S
-shared
-static
-std
-trigraphs
-U
-u
-v
-V
-w
-Wall
-Werror
-Winline
-W[no-]cast-qual
-W[no-]comment
-W[no-]comments
-W[no-]deprecated
-W[no-]fatal-errors
-W[no-]format-security
-W[no-]main
-W[no-]missing-declarations
-W[no-]missing-prototypes
-W[no-]overflow
-W[no-]overloaded-virtual
-W[no-]pointer-arith
-W[no-]return-type
-W[no-]strict-prototypes
-W[no-]trigraphs
-W[no-]uninitialized
-W[no-]unknown-pragmas
-W[no-]unused-function
-W[no-]unused-variable
-X
-x assembler-with-cpp
-x c
-x c++
-Xlinker
Unsupported Microsoft Visual Studio* Compiler Options

The Intel® C++ Compiler supports most of the same options as the Microsoft Visual Studio* compiler. However, a small subset of Microsoft Visual Studio compiler options are not supported by the Intel C++ Compiler. Most of the unsupported options, while useful for development purposes, are not required to build a working application. The following table lists some of these unsupported options:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>/AI&lt;dir&gt;</td>
<td>Add to assembly search path.</td>
</tr>
<tr>
<td>/clr</td>
<td>Compile for the common language runtime (managed C++).</td>
</tr>
<tr>
<td>/favor:[blend</td>
<td>AMD64</td>
</tr>
<tr>
<td>/Fd</td>
<td>Name the PDB file used for debug information for specified source files.</td>
</tr>
<tr>
<td>/FU&lt;file&gt;</td>
<td>Force using assembly/module.</td>
</tr>
<tr>
<td>/Fx</td>
<td>Merge injected code to file.</td>
</tr>
<tr>
<td>/Gi</td>
<td>Enable incremental compilation.</td>
</tr>
<tr>
<td>/openmp</td>
<td>Use /Qopenmp instead.</td>
</tr>
<tr>
<td>/w&lt;l&gt;&lt;n&gt;</td>
<td>Set warning level 1-4 for n</td>
</tr>
<tr>
<td>/Yd</td>
<td>Put debug information in every object (Microsoft PCH-specific option)</td>
</tr>
<tr>
<td>/Zm&lt;n&gt;</td>
<td>Control of maximum memory allocated by the compiler.</td>
</tr>
</tbody>
</table>

Unsupported options are not limited to this list.
The Intel C++ Compiler issues a remark stating lack of support for many of these options, but it silently ignores the options.

**Floating-point Operations**

**Overview: Floating-point Operations**

This section introduces the floating-point support in the Intel® C++ Compiler and provides information about using floating-point operations in your applications. The following table lists some possible starting points:

<table>
<thead>
<tr>
<th>If you are trying to...</th>
<th>Then start with...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Understand the programming objectives of floating-point applications</td>
<td>Programming Objectives of Floating-Point Applications</td>
</tr>
<tr>
<td>Use the <code>-fp-model</code> (Linux* and Mac OS* X) or <code>/fp</code> (Windows*) option</td>
<td>Using the <code>-fp-model</code> or <code>/fp</code> Option</td>
</tr>
<tr>
<td>Set the flush-to-zero (FTZ) or denormals-are-zero (DAZ) flags</td>
<td>Setting the FTZ and DAZ Flags</td>
</tr>
<tr>
<td>Tuning the performance of floating-point applications</td>
<td>Overview: Tuning Performance of Floating-Point Applications</td>
</tr>
</tbody>
</table>

**Floating-point Options Quick Reference**

The Intel® Compiler provides various options for you to optimize floating-point calculations with varying degrees of accuracy and predictability on different Intel architectures. This topic lists these compiler options and provides information about their supported architectures and operating systems.

**IA-32, Intel® 64, and IA-64 architectures**

<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-fp-model</code></td>
<td><code>/fp</code></td>
<td>Specifies semantics used in floating-point calculations. Values are</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>precise, fast [=1/2], strict, source, double, extended, [no-]except and except[-].</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>-fp-model</strong> compiler option</td>
</tr>
<tr>
<td>-fp-speculation</td>
<td>/Qfp-speculation</td>
<td>Specifies the speculation mode for floating-point operations. Values are fast, safe, strict, and off.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- <strong>-fp-speculation</strong> compiler option</td>
</tr>
<tr>
<td>-prec-div</td>
<td>/Qprec-div</td>
<td>Attempts to use slower but more accurate implementation of floating-point divide. Use this option to disable the divide optimizations in cases where it is important to maintain the full range and precision for floating-point division. Using this option results in greater accuracy with some loss of performance.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Specifying <strong>-no-prec-div</strong> (Linux and Mac OS X) or</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td>/Qprec-div- (Windows) enables optimizations that result in slightly less precise results than full IEEE division.</td>
<td></td>
</tr>
<tr>
<td>-complex-limited-range</td>
<td>/Qcomplex-limited-range</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Enables the use of basic algebraic expansions of some arithmetic operations involving data of type COMPLEX. This can cause performance improvements in programs that use a lot of COMPLEX arithmetic. Values at the extremes of the exponent range might not compute correctly.</td>
<td></td>
</tr>
<tr>
<td>-ftz</td>
<td>/Qftz</td>
<td></td>
</tr>
<tr>
<td></td>
<td>The default behavior depends on the architecture. Refer to the following topic for details:</td>
<td></td>
</tr>
</tbody>
</table>

**IA-32 and Intel® 64 architectures**

1222
<table>
<thead>
<tr>
<th>Linux* and Mac OS* X</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-prec-sqrt</td>
<td>/Qprec-sqrt</td>
<td>Improves the accuracy of square root implementations, but using this option may impact speed.</td>
</tr>
<tr>
<td>-pc</td>
<td>/Qpc</td>
<td>Changes the floating point significand precision. Use this option when compiling applications. The application must use <code>main()</code> as the entry point, and you must compile the source file containing <code>main()</code> with this option.</td>
</tr>
<tr>
<td>-rcd</td>
<td>/Qrcd</td>
<td>Disables rounding mode changes for floating-point-to-integer conversions.</td>
</tr>
<tr>
<td>-fp-port</td>
<td>/Qfp-port</td>
<td>Causes floating-point values to be rounded to the source precision at assignments and casts.</td>
</tr>
<tr>
<td>Linux* and Mac OS* X</td>
<td>Windows*</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td>-mp1 /Qprec</td>
<td></td>
<td>This option rounds floating-point values to the precision specified in the source program prior to comparisons. It also implies -prec-div and -prec-sqrt (Linux and Mac OS X) or /Qprec-div and /Qprec-sqrt (Windows). This option has less impact to performance and disables fewer optimizations than the -fp-model precise (Linux* and Mac OS* X) or /fp:precise (Windows) option.</td>
</tr>
</tbody>
</table>

**IA-64 architecture only**

<table>
<thead>
<tr>
<th>Linux*</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-IPF-fma /QIPF-fma</td>
<td></td>
<td>Enables or disables the contraction of floating-point multiply and add/subtract operations into a single operation.</td>
</tr>
</tbody>
</table>

- **-mp1** compiler option

- **-IPF-fma** compiler option
<table>
<thead>
<tr>
<th>Linux*</th>
<th>Windows*</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-IPF-fp-relaxed (deprecated)</td>
<td>/QIPF-fp-relaxed (deprecated)</td>
<td>Enables use of faster but slightly less accurate code sequences for math functions, such as the <code>sqrt()</code> function and the divide operation. As compared to strict IEEE* precision, using this option slightly reduces the accuracy of floating-point calculations performed by these functions, usually limited to the least significant binary digit. This option is deprecated.</td>
</tr>
</tbody>
</table>

**Understanding Floating-point Operations**

**Using the -fp-model (/fp) Option**

The `-fp-model` (Linux* and Mac OS* X) or `/fp` (Windows*) option allows you to control the optimizations on floating-point data. You can use this option to tune the performance, level of accuracy, or result consistency across platforms for floating-point applications.

For applications that do not require support for denormalized numbers, the `-fp-model` or `/fp` option can be combined with the `-ftz` (Linux* and Mac OS* X) or `/Qftz` (Windows*) option to flush denormalized results to zero in order to obtain improved runtime performance on processors based on:

- IA-32 and Intel® 64 architectures
IA-64 architecture

You can use keywords to specify the semantics to be used. Possible values of the keywords are as follows:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>precise</td>
<td>Enables value-safe optimizations on floating-point data.</td>
</tr>
<tr>
<td>fast[=1</td>
<td>2]</td>
</tr>
<tr>
<td>strict</td>
<td>Enables precise and except, disables contractions, and enables pragma stdc fenv_access.</td>
</tr>
<tr>
<td>source</td>
<td>Rounds intermediate results to source-defined precision and enables value-safe optimizations.</td>
</tr>
<tr>
<td>[no-] except</td>
<td>Determines whether floating-point exception semantics are used.</td>
</tr>
<tr>
<td>(Linux* and Mac OS* X) or except[-] (Windows*)</td>
<td></td>
</tr>
</tbody>
</table>

The default value of the option is `-fp-model fast=1` or `/fp:fast=1`, which means that the compiler uses more aggressive optimizations on floating-point calculations.

Note

Using the default option keyword `-fp-model fast` or `/fp:fast`, you may get significant differences in your result depending on whether the compiler uses x87 or SSE2 instructions to implement floating-point operations. Results are more consistent when the other option keywords are used.

Several examples are provided to illustrate the usage of the keywords. These examples show:
A small example of source code
Note that the same source code is considered in all the included examples.

The semantics that are used to interpret floating-point calculations in the source code

One or more possible ways the compiler may interpret the source code
Note that there are several ways the compiler may interpret the code; we show just some of these possibilities.

**-fp-model fast or /fp:fast**

Example source code:
```c
float t0, t1, t2;
...
t0 = 4.0f + 0.1f + t1 + t2;
```
When this option is specified, the compiler applies the following semantics:

- Additions may be performed in any order
- Intermediate expressions may use single, double, or extended double precision
- The constant addition may be pre-computed, assuming the default rounding mode

Using these semantics, the following shows some possible ways the compiler may interpret the original code:
```c
float t0, t1, t2;
...
t0 = (float)((double)t1 + (double)t2) + 4.1f;
```
```c
float t0, t1, t2;
...
t0 = (t1 + t2) + 4.1f;
```
```c
float t0, t1, t2;
...
t0 = (t1 + 4.1f) + t2;
```

**-fp-model extended or /fp:extended**

This setting is equivalent to -fp-model precise on Linux* operating systems based on the IA-32 architecture and -fp-model precise or /fp:precise on systems based on the IA-64 architecture.

Example source code:
float t0, t1, t2;
... 
t0 = 4.0f + 0.1f + t1 + t2;

When this option is specified, the compiler applies the following semantics:

- Additions are performed in program order
- Intermediate expressions use extended double precision
- The constant addition may be pre-computed, assuming the default rounding mode

Using these semantics, the following shows a possible way the compiler may interpret the original code:

float t0, t1, t2;
... 
t0 = (float)(((long double)4.1f + (long double)t1) + (long double)t2);

- **-fp-model source or /fp:source**

This setting is equivalent to **-fp-model precise or /fp:precise** on systems based on the Intel® 64 architecture.

Example source code:

float t0, t1, t2;
... 
t0 = 4.0f + 0.1f + t1 + t2;

When this option is specified, the compiler applies the following semantics:

- Additions are performed in program order, taking into account any parentheses
- Intermediate expressions use the precision specified in the source code
- The constant addition may be pre-computed, assuming the default rounding mode

Using these semantics, the following shows a possible way the compiler may interpret the original code:

float t0, t1, t2;
... 
t0 = ((4.1f + t1) + t2);

- **-fp-model double or /fp:double**

This setting is equivalent to **-fp-model precise or /fp:precise** on Windows systems based on the IA-32 architecture.

Example source code:
float t0, t1, t2;
...  
t0 = 4.0f + 0.1f + t1 + t2;

When this option is specified, the compiler applies the following semantics:

- Additions are performed in program order
- Intermediate expressions use double precision
- The constant addition may be pre-computed, assuming the default rounding mode

Using these semantics, the following shows a possible way the compiler may interpret the original code:

float t0, t1, t2;
...  
t0 = (float)(((double)4.1f + (double)t1) + (double)t2);

-fp-model strict or /fp:strict

Example source code:

float t0, t1, t2;
...  
t0 = 4.0f + 0.1f + t1 + t2;

When this option is specified, the compiler applies the following semantics:

- Additions are performed in program order, taking into account any parentheses
- Expression evaluation matches expression evaluation under keyword precise.
- The constant addition will not be pre-computed, because there is no way to tell what rounding mode will be active when the program runs.

Using these semantics, the following shows a possible way the compiler may interpret the original code:

float t0, t1, t2;
...  
t0 = (float)(((long double)4.0f + (long double)0.1f) + (double)t1) + (long double)t2);

See Also

-fp-model compiler option
/fp compiler option

Floating-point Optimizations
Application performance is an important goal of the Intel® Compilers, even at default optimization levels. A number of optimizations involve transformations, such as evaluation of constant expressions at compiler time, hoisting invariant expressions out of loops, or changes in the order of evaluation of expressions. These optimizations usually help the compiler produce most efficient code possible. However, this may not be true for floating-point applications, because some optimizations may affect accuracy, reproducibility, and performance. Some optimizations are not consistent with strict interpretation of the ANSI or ISO standards for C and C++, which can result in differences in rounding and small variants in floating-point results that may be more or less accurate than the ANSI-conformant result.

Intel Compilers provide the -fp-model (Linux* and Mac OS* X) or /fp (Windows*) option, which allows you to control the optimizations performed when you build an application. The option allows you to specify the compiler rules for:

- **Value safety:** Whether the compiler may perform transformations that could affect the result. For example, in the SAFE mode, the compiler won't transform \( \frac{x}{x} \) to 1.0. The UNSAFE mode is the default.
- **Floating-point expression evaluation:** How the compiler should handle the rounding of intermediate expressions. For example, when double precision is specified, the compiler may transform the statement
  \[
  t0=4.0f+0/1f+t1+t2; \quad \text{to}
  \]
  \[
  t0=(\text{float})(4.1+(\text{double})t1+(\text{double})t2);
  \]
- **Floating-point contractions:** Whether the compiler should generate floating-point multiply-add (FMA) on processors based on the IA-64 architecture. When enabled, the compiler may generate FMA for combined multiply/add; when disabled, the compiler must generate separate multiply/add with intermediate rounding.
- **Floating-point environment access:** Whether the compiler must account for the possibility that the program might access the floating-point environment, either by changing the default floating-point control settings or by reading the floating-point status flags. This is disabled by default. You can use the -fp-
model:strict (Linux and Mac OS X) /fp:strict (Windows) option to enable it.

- Precise floating-point exceptions: Whether the compiler should account for the possibility that floating-point operations might produce an exception. This is disabled by default. You can use -fp-model:strict (Linux and Mac OS X) or /fp:strict (Windows); or -fp-model:except (Linux and Mac OS X) or /fp:except (Windows) to enable it.

Consider the following example:

double a=1.5;
int x=0;
...
__try {
    int t0=a; //raises inexact
    x=1;
    a*=2;
} __except(1) {
    printf("SEH Exception: x=%d\n", x);
}

Without precise floating-point exceptions, the result is SEH Exception: x=1;
with precision floating-point exceptions, the result is SEH Exception: x=0.

The following table describes the impact of different keywords of the option on compiler rules and optimizations:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>Value</th>
<th>Safety</th>
<th>Floating-Point Expression</th>
<th>Floating-Point Contractions</th>
<th>Floating-Point Environment Access</th>
<th>Precise Floating-Point Exceptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>precise</td>
<td>source</td>
<td>Safe</td>
<td>Varies</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>double</td>
<td>double</td>
<td></td>
<td>Source Double</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>extended</td>
<td>double</td>
<td></td>
<td>Extended</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>strict</td>
<td></td>
<td>Safe</td>
<td>Varies</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>fast=1</td>
<td></td>
<td>Unsafe</td>
<td>Unknown</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>(default)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
It is illegal to specify the `except` keyword in an unsafe safety mode.

Based on the objectives of an application, you can choose to use different sets of compiler options and keywords to enable or disable certain optimizations, so that you can get the desired result.

**Programming Objectives of Floating-point Applications**

In general, the programming objectives of the floating-point applications fall into the following categories:

- **Accuracy**: The application produces results that are close to the correct result.
- **Reproducibility and portability**: The application produces results that are consistent across different runs, different set of build options, different compilers, different platforms, and different architectures.
- **Performance**: The application produces the most efficient code possible.

Based on the goal of an application, you will need to balance the tradeoffs among these objectives. For example, if you are developing a 3D graphics engine, then performance can be the most important factor to consider, and reproducibility and accuracy can be your secondary concerns.

Intel® Compiler provides appropriate compiler options, such as the `-fp-model` (Linux* and Mac OS* X) or `/fp` (Windows*) option, which allows you to tune your applications based on specific objectives. The compiler processes the code
differently when you specify different compiler options. Take the following code as an example:

```c
float t0, t1, t2;
...
t0=t1+t2+4.0f+0.1f;
```

If you specify the `-fp-model extended` (Linux and Mac OS X) or `/fp:extended` (Windows) option in favor of accuracy, the compiler generates the following assembly code:

```
fld DWORD PTR _t1
fadd DWORD PTR _t2
fadd DWORD PTR _Cnst4.0
fadd DWORD PTR _Cnst0.1
fstp DWORD PTR _t0
```

If you specify the `-fp-model source` (Linux and Mac OS X) or `/fp:source` (Windows) option in favor of reproducibility and portability, the compiler generates the following assembly code:

```
movss xmm0, DWORD PTR _t1
addss xmm0, DWORD PTR _t2
addss xmm0, DWORD PTR _Cnst4.0
addss xmm0, DWORD PTR _Cnst0.1
movss DWORD PTR _t0, xmm0
```

If you specify the `-fp-model fast` (Linux and Mac OS X) or `/fp:fast` (Windows) option in favor of performance, the compiler generates the following assembly code:

```
movss xmm0, DWORD PTR _Cnst4.1
addss xmm0, DWORD PTR _t1
addss xmm0, DWORD PTR _t2
movss DWORD PTR _t0, xmm0
```

In most cases, an application will be much more complicated. You should select appropriate compiler options by carefully considering your programming objectives and balancing the tradeoffs among these objectives.

**Denormal Numbers**

A normalized number is a number for which both the exponent (including offset) and the most significant bit of the mantissa are non-zero. For such numbers, all the bits of the mantissa contribute to the precision of the representation. The smallest normalized single precision floating-point number greater than zero is about $1.1754943^{-38}$. Smaller numbers are possible, but those numbers must be represented with a zero exponent and a mantissa whose leading bit(s) are zero,
which leads to a loss of precision. These numbers are called denormalized numbers; denormals (newer specifications refer to these as subnormal numbers). Denormal computations use both hardware or operating system resources to handle them, which can cost hundreds of clock cycles.

- Denormal computations take much longer to calculate on processors based on IA-32 and Intel® 64 architectures than normal computations.
- Denormals are computed in software on processors based on the IA-64 architecture, and the computation usually requires hundreds of clock cycles, which results in excessive kernel time.

There are several ways to handle denormals and increase the performance of your application:

- Scale the values into the normalized range.
- Use a higher precision data type with a larger dynamic range.
- Flush denormals to zero.

See Also

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture
Institute of Electrical and Electronics Engineers, Inc*. (IEEE) web site for information about the current floating-point standards and recommendations

**Floating-point Environment**

The floating-point environment is a collection of registers that control the behavior of the floating-point machine instructions and indicate the current floating-point status. The floating-point environment can include rounding mode controls, exception masks, flush-to-zero (FTZ) controls, exception status flags, and other floating-point related features.
For example, on IA-32 and Intel® 64 architectures, bit 15 of the MXCSR register enables the flush-to-zero mode, which controls the masked response to an single-instruction multiple-data (SIMD) floating-point underflow condition. The floating-point environment affects most floating-point operations; therefore, correct configuration to meet your specific needs is important. For example, the exception mask bits define which exceptional conditions will be raised as exceptions by the processor. In general, the default floating-point environment is set by the operating system. You don't need to configure the floating-point environment unless the default floating-point environment does not suit your needs.

There are several methods available if you want to modify the default floating-point environment. For example, you can use inline assembly, compiler built-in functions, and library functions.

Changing the default floating-point environment affects runtime results only. This does not affect any calculations which are pre-computed at compile time. If strict reproducibility and consistency are important do not change the floating point environment without also using either `-fp-model strict` (Linux or Mac OS* X) or `/fp:strict` (Windows*) option or `pragma fenv_access`.

See Also

Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 1: Basic Architecture

Setting the FTZ and DAZ Flags

In Intel® processors, the flush-to-zero (FTZ) and denormals-are-zero (DAZ) flags in the MXCSR register are used to control floating-point calculations. When the FTZ and DAZ flags are enabled, the Single Instructions and Multiple Data (SIMD) floating-point computation can be accelerated, thus improving the performance of the application.
You can use the `-ftz` (Linux* and Mac OS* X) or `/Qftz` (Windows*) option to flush denormal results to zero when the application is in the gradual underflow mode. This option may improve performance if the denormal values are not critical to your application’s behavior.

The `-ftz` or `/Qftz` option sets or resets the FTZ and the DAZ hardware flags. The following table describes how the compiler process denormal values based on the status of the FTZ and DAZ flags:

<table>
<thead>
<tr>
<th>Flag</th>
<th>When set to ON, the compiler...</th>
<th>When set to OFF, the compiler...</th>
<th>Supported on</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FTZ</td>
<td>Sets denormal results from floating-point calculations to zero.</td>
<td>Does not change the denormal results.</td>
<td>IA-64 and Intel® 64 architectures</td>
</tr>
<tr>
<td>DAZ</td>
<td>Treats denormal values used as input to floating-point instructions as zero.</td>
<td>Does not change the denormal instruction inputs.</td>
<td>Intel® 64 architecture</td>
</tr>
</tbody>
</table>

- FTZ and DAZ are not supported on all IA-32 architectures. On systems based on the IA-64 architecture, FTZ always works, while on systems based on the IA-32 and Intel® 64 architectures, it only applies to SSE instructions. Hence if your application happened to generate denormals using x87 instructions, FTZ does not apply.
- DAZ and FTZ flags are not compatible with IEEE Standard 754, so you should only consider enabling them when strict compliance to the IEEE standard is not required and application performance has higher priority than application accuracy.

Options `-ftz` and `/Qftz` are performance options. Setting these options does not guarantee that all denormals in a program are flushed to zero. They only cause denormals generated at run time to be flushed to zero.
When -ftz or /Qftz is used in combination with an SSE-enabling option on systems based on the IA-32 architecture (for example, -xW or /QxW), the compiler will insert code in the main routine to set FTZ and DAZ. When -ftz or /Qftz is used without such an option, the compiler will insert code to conditionally set FTZ/DAZ based on a run-time processor check. -no-ftz (Linux and Mac OS X) or /Qftz- (Windows) will prevent the compiler from inserting any code that might set FTZ or DAZ.

The -ftz or /Qftz option only has an effect when the main program is being compiled. It sets the FTZ/DAZ mode for the process. The initial thread and any threads subsequently created by that process will operate in the FTZ/DAZ mode.

On systems based on the IA-64 architecture, optimization option O3 sets -ftz and /Qftz; optimization option O2 sets -no-ftz (Linux) and /Qftz- (Windows). On systems based on the IA-32 and Intel® 64 architectures, every optimization option O level, except O0, sets -ftz and /Qftz.

If this option produces undesirable results of the numerical behavior of your program, you can turn the FTZ/DAZ mode off by using -no-ftz or /Qftz- in the command line while still benefiting from the O3 optimizations.

For some non-Intel processors, you can set the flags manually with the following macros:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable FTZ</td>
<td>_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_ON)</td>
</tr>
<tr>
<td>Enable DAZ</td>
<td>_MM_SET_DENORMALS_ZERO_MODE(_MM_DENORMALS_ZERO_ON)</td>
</tr>
</tbody>
</table>

The prototypes for these macros are in *xmmintrin.h* (FTZ) and *pmmintrin.h* (DAZ).

See Also
Tuning Performance

Overview: Tuning Performance

This section describes several programming guidelines that can help you improve the performance of a floating-point applications:

- Avoid exceeding representable ranges during computation; handling these cases can have a performance impact.
- Use a single precision type (for example, float) unless the extra precision obtained through double or long double is required. Greater precision types increase memory size and bandwidth requirements.
- **Reduce the impact of denormal exceptions for all supported architectures.**
- **Avoid mixed data type arithmetic expressions.**

Handling Floating-point Array Operations in a Loop Body

Following the guidelines below will help autovectorization of the loop.

- Statements within the loop body may contain float or double operations (typically on arrays). The following arithmetic operations are supported: addition, subtraction, multiplication, division, negation, square root, MAX, MIN, and mathematical functions such as SIN and COS.
- Writing to a float scalar/array and a double scalar/array within the same loop decreases the chance of autovectorization due to the differences in the vector length (that is, the number of elements in the vector register) between float and double types. If autovectorization fails, try to avoid using mixed data types.

**Note**

The special __m64 and __m128 datatypes are not vectorizable. The loop body cannot contain any function calls. Use of the Intel(R) Streaming SIMD Extensions intrinsics (_mm_add_ps) are not allowed.

Reducing the Impact of Denormal Exceptions
Denormalized floating-point values are those that are too small to be represented in the normal manner; for example, the mantissa cannot be left-justified.

Denormal values require hardware or operating system interventions to handle the computation, so floating-point computations that result in denormal values may have an adverse impact on performance.

There are several ways to handle denormals to increase the performance of your application:

- Scale the values into the normalized range
- Use a higher precision data type with a larger dynamic range
- Flush denormals to zero

For example, you can translate them to normalized numbers by multiplying them using a large scalar number, doing the remaining computations in the normal space, then scaling back down to the denormal range. Consider using this method when the small denormal values benefit the program design.

Consider using a higher precision data type with a larger dynamic range. For example, converting variables declared as `float` to be declared as `double`.

Understand that making the change has the potential to cause your program to slow down, storage requirements will increase, which increases the amount of time loading and storing data from memory; it can also decrease the potential throughput of SSE operations.

If you change the declaration of a variable you might also need to change the libraries you call to use the variable; for example, `cosd()` instead of `cos()`.

Another strategy that might result in increased performance is to increase the amount of precision of intermediate values using the `-fp-model [double|extended]` option; however, if you increased precision as the solution to slow performance caused by denormal numbers you must verify the resulting changes actually increase performance.

Finally, in many cases denormal numbers be treated safely as zero without adverse effects on program results. Depending on the target architecture, use flush-to-zero (FTZ) options.
IA-32 and Intel® 64 Architectures

These architectures take advantage of the FTZ and DAZ (denormals-are-zero) capabilities of Streaming SIMD Extensions (SSE), Streaming SIMD Extensions 2 (SSE2), and Streaming SIMD Extensions 3 (SSE3), and Supplemental Streaming SIMD Extensions 3 (SSSE3) instructions.

By default, the compiler for the IA-32 architecture generates code that will run on machines that do not support SSE instructions. The compiler implements floating-point calculations using the x87 floating-point unit, which does not benefit from the FTZ and DAZ settings. You can use the -x (Linux* and Mac OS* X) or /Qx (Windows*) option to enable the compiler to implement floating-point calculations using the SSE and SSE2 instructions. The compiler for the Intel® 64 architecture generates SSE2 instructions by default.

The FTZ and DAZ modes are enabled by default when you compile the source file containing main() using the Intel Compiler. The compiler generates a call to a library routine that performs a runtime processor check. The FTZ and DAZ modes are set provided that the modes are available for the machine on which the program is running.

IA-64 Architecture

Enable the FTZ mode by using the -ftz (Linux and Mac OS X) or /Qftz (Windows) option on the source file containing main(). The -O3 (Linux and Mac OS X) or /O3 (Windows) option automatically enables -ftz or /Qftz.

Note

After using flush-to-zero, ensure that your program still gives correct results when treating denormalized values as zero.

See Also

Setting the FTZ and DAZ Flags

Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1: Basic Architecture
**Avoiding Mixed Data Type Arithmetic Expressions**

Avoid mixing integer and floating-point (float, double, or long double) data in the same computation. Expressing all numbers in a floating-point arithmetic expression (assignment statement) as floating-point values eliminates the need to convert data between fixed and floating-point formats. Expressing all numbers in an integer arithmetic expression as integer values also achieves this. This improves run-time performance.

For example, assuming that I and J are both int variables, expressing a constant number (2.) as an integer value (2) eliminates the need to convert the data. The following examples demonstrate inefficient and efficient code.

**Example: Inefficient Code**
```c++
int I, J;
I = J / 2.;
```

**Example: Inefficient Code**
```c++
int I, J;
I = J / 2;
```

You can use different sizes of the same general data type in an expression with minimal or no effect on run-time performance. For example, using float, double, and long double floating-point numbers in the same floating-point arithmetic expression has minimal or no effect on run-time performance. However, this practice of mixing different sizes of the same general data type in an expression can lead to unexpected results due to operations being performed in a lower precision than desired.

**Using Efficient Data Types**
In cases where more than one data type can be used for a variable, consider selecting the data types based on the following hierarchy, listed from most to least efficient:

- char
- short
- int
- long
- long long
- float
- double
- long double

However, keep in mind that in an arithmetic expression, you should avoid mixing integer and floating-point data.

You can use integer data types (int, int long, etc.) in loops to improve floating point performance. Convert the data type to integer data types, process the data, then convert the data to the old type.

**Checking the Floating-point Stack State**

On systems based on the IA-32 architectures, when an application calls a function that returns a floating-point value, the returned floating-point value is supposed to be on the top of the floating-point stack. If the return value is not used, the compiler must pop the value off of the floating-point stack in order to keep the floating-point stack in the correct state.

On systems based on Intel(R) 64 architectures, floating-point values are usually returned in the xmm0 register. The floating-point stack is used only when the return value is a long double on Linux* and Mac OS* X systems.

If the application calls a function without defining or incorrectly defining the function's prototype, the compiler cannot determine if the function must return a floating-point value. Consequently, the return value is not popped off the floating-point stack if it is not used. This can cause the floating-point stack to overflow.

The overflow of the stack results in two undesirable situations:
• A NaN value gets involved in the floating-point calculations
• The program results become unpredictable; the point where the program starts making errors can be arbitrarily far away from the point of the actual error.

For systems based on the IA-32 and Intel® 64 architectures, the -fp-stack-check (Linux* and Mac OS* X) or /Qfp-stack-check (Windows*) option checks whether a program makes a correct call to a function that should return a floating-point value. If an incorrect call is detected, the option places a code that marks the incorrect call in the program. The -fp-stack-check (Linux* and Mac OS* X) or /Qfp-stack-check (Windows*) option marks the incorrect call and makes it easy to find the error.

Note

The -fp-stack-check (Linux* and Mac OS* X) and the /Qfp-stack-check (Windows*) option causes significant code generation after every function/subroutine call to ensure that the floating-point stack is maintained in the correct state. Therefore, using this option slows down the program being compiled. Use the option only as a debugging aid to find floating point stack underflow/overflow problems, which can be otherwise hard to find.

See Also

-fp-stack-check, /Qfp-stack-check option Tells the compiler to generate extra code after every function call to ensure that the floating-point stack is in the expected state.

Intrinsics Reference

Overview: Intrinsics Reference

Intrinsics are assembly-coded functions that allow you to use C++ function calls and variables in place of assembly instructions.

Intrinsics are expanded inline eliminating function call overhead. Providing the same benefit as using inline assembly, intrinsics improve code readability, assist instruction scheduling, and help reduce debugging.
Intrinsics provide access to instructions that cannot be generated using the standard constructs of the C and C++ languages.

**Intrinsics for Intel® C++ Compilers**

The Intel® C++ Compiler enables easy implementation of assembly instructions through the use of intrinsics. Intrinsics are provided for the following instructions:

- Carry-less multiplication instruction and Advanced Encryption Standard Extensions instructions
- Half-float conversion instructions
- MMX™ Technology instructions
- Intel® Streaming SIMD Extensions (SSE) instructions
- Intel® Streaming SIMD Extensions 2 (SSE2) instructions
- Intel® Streaming SIMD Extensions 3 (SSE3) instructions
- Supplemental Streaming SIMD Extensions 3 (SSSE3) instructions
- Intel® Streaming SIMD Extensions 4 (SSE4) instructions

The Intel C++ Compiler provides IA-64 architecture-specific intrinsics and intrinsics that work across IA-32, Intel® 64, and IA-64 architectures. Most intrinsics map directly to a corresponding assembly instruction, some map to several assembly instructions.


**Availability of Intrinsics on Intel Processors**

Not all Intel processors support all intrinsics. For information on which intrinsics are supported on Intel processors, visit [http://processorfinder.intel.com](http://processorfinder.intel.com).

The Processor Spec Finder tool links directly to all processor documentation and the data sheets list the features, including intrinsics, supported by each processor.

**Details about Intrinsics**

All instructions use the following features:
Registers

Intel processors provide special register sets for different instructions. The MMX instructions use eight 64-bit registers \((\text{mm0 to mm7})\) which are aliased on the floating-point stack registers.

The Streaming SIMD Extensions and the Advanced Encryption Standard instructions use eight 128-bit registers \((\text{xmm0 to xmm7})\).

Because each of these registers can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as single-instruction multiple data processing (SIMD).

For each computational and data manipulation instruction in the new extension sets, there is a corresponding C intrinsic that implements that instruction directly. This frees you from managing registers and assembly programming. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

**Note**

The MM and XMM registers are the SIMD registers used by the IA-32 architecture-based platforms to implement MMX™ technology and SSE or SSE2 intrinsics. On the IA-64 architecture, the MMX™ and SSE intrinsics use the 64-bit general registers and the 64-bit significand of the 80-bit floating-point register.

Data Types

Intrinsic functions use new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions.

The following table details for which instructions each of the new data types are available. A 'Yes' indicates that the data type is available for that group of
intrinsics; an 'NA' indicates that the data type is not available for that group of intrinsics.

<table>
<thead>
<tr>
<th>Data Types --&gt;</th>
<th>__m64</th>
<th>__m128</th>
<th>__m128d</th>
<th>__m128i</th>
<th>__m256</th>
<th>__m256d</th>
<th>__m256i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MMX™ Technology</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Intrinsics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming SIMD</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Extensions Intrinsics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming SIMD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Extensions 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intrinsics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming SIMD</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Extensions 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intrinsics</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Advanced Encryption Standard</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Intrinsics + Carry-less Multiplication Intrinsic</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Half-Float Intrinsics</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

__m64 Data Type

The __m64 data type is used to represent the contents of an MMX register, which is the register that is used by the MMX technology intrinsics. The __m64
data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

__m128 Data Types

The __m128 data type is used to represent the contents of a Streaming SIMD Extension register used by the Streaming SIMD Extension intrinsics. The __m128 data type can hold four 32-bit floating-point values.

The __m128d data type can hold two 64-bit floating-point values.

The __m128i data type can hold sixteen 8-bit, eight 16-bit, four 32-bit, or two 64-bit integer values.

The compiler aligns __m128d and __m128i local and global data to 16-byte boundaries on the stack. To align integer, float, or double arrays, you can use thedeclspec align statement.

Data Types Usage Guidelines

These data types are not basic ANSI C data types. You must observe the following usage restrictions:

- Use data types only on either side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (+, -, etc).
- Use data types as objects in aggregates, such as unions, to access the byte elements and structures.
- Use data types only with the respective intrinsics described in this documentation.

Accessing __m128i Data

To access 8-bit data:

```c
#define _mm_extract_epi8(x, imm) \
    (((imm) & 0x1) == 0) ? \n    _mm_extract_epi16((x), (imm) >> 1) & 0xff : \n    _mm_extract_epi16(_mm_srli_epi16((x), 8), (imm) >> 1))
```

For 16-bit data, use the following intrinsic:

```c
int _mm_extract_epi16(__m128i a, int imm)
```

To access 32-bit data:
To access 64-bit data (Intel® 64 architecture only):

```c
#define _mm_extract_epi64(x, imm) 
    _mm_cvtsi128_si64(_mm_srli_si128((x), 8 * (imm)))
```

**Naming and Usage Syntax**

Most intrinsic names use the following notational convention:

```
_mm_<intrin_op>_ <suffix>
```

The following table explains each item in the syntax.

- `<intrin_op>` Indicates the basic operation of the intrinsic; for example, add for addition and sub for subtraction.
- `<suffix>` Denotes the type of data the instruction operates on. The first one or two letters of each suffix denote whether the data is packed (p), extended packed (ep), or scalar (s). The remaining letters and numbers denote the type, with notation as follows:
  - s single-precision floating point
  - d double-precision floating point
  - i128 signed 128-bit integer
  - i64 signed 64-bit integer
  - u64 unsigned 64-bit integer
  - i32 signed 32-bit integer
  - u32 unsigned 32-bit integer
  - i16 signed 16-bit integer
  - u16 unsigned 16-bit integer
  - i8 signed 8-bit integer
  - u8 unsigned 8-bit integer

A number appended to a variable name indicates the element of a packed object. For example, r0 is the lowest word of r. Some intrinsics are "composites" because they require more than one instruction to implement them.
The packed values are represented in right-to-left order, with the lowest value being used for scalar operations. Consider the following example operation:

```c++
double a[2] = {1.0, 2.0};
__m128d t = _mm_load_pd(a);
```

The result is the same as either of the following:

```c++
__m128d t = _mm_set_pd(2.0, 1.0);
__m128d t = _mm_setr_pd(1.0, 2.0);
```

In other words, the xmm register that holds the value `t` appears as follows:

```
0 1 2 3 4 5 6 7 8 9 10 11
2.0 1.0
```

The "scalar" element is 1.0. Due to the nature of the instruction, some intrinsics require their arguments to be immediates (constant integer literals).

**References**

See the following publications and internet locations for more information about intrinsics and the Intel architectures that support them. You can find all publications on the Intel website.

<table>
<thead>
<tr>
<th>Internet Location or Publication</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>developer.intel.com</td>
<td>Technical resource center for hardware designers and developers; contains links to product pages and documentation.</td>
</tr>
<tr>
<td>Intel® Itanium® Architecture Software Developer's Manuals, Volume 3: Instruction Set Reference</td>
<td>Contains information and details about Itanium® instructions.</td>
</tr>
</tbody>
</table>
Intrinsics for Advanced Encryption Standard Implementation

Overview: Intrinsics for Carry-less Multiplication Instruction and Advanced Encryption Standard Instructions

The Intel® C++ Compiler provides intrinsics to enable carry-less multiplication and encryption based on Advanced Encryption Standard (AES) specifications. The carry-less multiplication intrinsic corresponds to a single new instruction, `PCLMULQDQ`. The AES extension intrinsics correspond to AES extension instructions.

The AES extension instructions and the `PCLMULQDQ` instruction follow the same system software requirements for XMM state support and SIMD floating-point exception support as SSE2, SSE3, SSSE3, SSE4 extensions. Intel®64 processors using 32nm processing technology support the AES extension instructions as well as the `PCLMULQDQ` instruction.

AES Encryption and Cryptographic Processing

AES encryption involves processing 128-bit input data (plaintext) through a finite number of iterative operation, referred to as "AES round", into a 128-bit encrypted block (ciphertext). Decryption follows the reverse direction of iterative operation using the "equivalent inverse cipher" instead of the "inverse cipher".

The cryptographic processing at each round involves two input data, one is the "state", the other is the "round key". Each round uses a different "round key". The
round keys are derived from the cipher key using a "key schedule" algorithm. The "key schedule" algorithm is independent of the data processing of encryption/decryption, and can be carried out independently from the encryption/decryption phase.

The AES standard supports cipher key of sizes 128, 192, and 256 bits. The respective cipher key sizes corresponds to 10, 12, and 14 rounds of iteration.

**Carry-less Multiplication Instruction and AES Extension Instructions**

A single instruction, PCLMULQDQ, performs carry-less multiplication for two binary numbers that are up to 64-bit wide.

The AES extensions provide:

- two instructions to accelerate AES rounds on encryption (AESENC and AESENCLAST)
- two instructions for AES rounds on decryption using the equivalent inverse cipher (AESDEC and AESDECLAST)
- instructions for the generation of key schedules (AESIMC and AESKEYGENASSIST)

**Detecting Support for Using Instructions**

Before any application attempts to use the PCLMULQDQ or the AES extension instructions, it must first detect if the instructions are supported by the processor.

To detect support for the PCLMULQDQ instruction, your application must check the following:

\[
\text{CPUID.01H:ECX.PCLMULQDQ}[\text{bit 1}] = 1.
\]

To detect support for the AES extension instructions, your application must check the following:

\[
\text{CPUID.01H:ECX.AES}[\text{bit 25}] = 1.
\]

Operating systems that support handling of the SSE state also support applications that use AES extension instruction and the PCLMULQDQ instruction.

**Intrinsics for Carry-less Multiplication Instruction and Advanced Encryption Standard Instructions**

The prototypes for the Carry-less multiplication intrinsic and the AES intrinsics are defined in the wmmintrin.h file.

**Carry-less Multiplication Intrinsic**
The single general purpose block encryption intrinsic description is provided below.

```c
__m128i _mm_clmulepi64_si128(__m128i v1, __m128i v2, const int imm8);
```

Performs a carry-less multiplication of one quadword of v1 by one quadword of v2, and returns the result. The imm8 value is used to determine which quadwords of v1 and v2 should be used.

**Corresponding Instruction:** PCLMULQDQ

**Advanced Encryption Standard Intrinsics**

The AES intrinsics are described below.

```c
__m128i _mm_aesdec_si128(__m128i v, __m128i rkey);
```

Performs one round of an AES decryption flow using the Equivalent Inverse Cipher operating on a 128-bit data (state) from v with a 128-bit round key from rkey.

**Corresponding Instruction:** AESDEC

```c
__m128i _mm_aesdecrast_si128(__m128i v, __m128i rkey);
```

Performs the last round of an AES decryption flow using the Equivalent Inverse Cipher operating on a 128-bit data (state) from v with a 128-bit round key from rkey.

**Corresponding Instruction:** AESDECLAST

```c
__m128i _mm_aesencc_si128(__m128i v, __m128i rkey);
```

Performs one round of an AES encryption flow operating on a 128-bit data (state) from v with a 128-bit round key from rkey.

**Corresponding Instruction:** AESENC

```c
__m128i _mm_aesenclast_si128(__m128i v, __m128i rkey);
```

Performs the last round of an AES encryption flow operating on a 128-bit data (state) from v with a 128-bit round key from rkey.

**Corresponding Instruction:** AESENCLAST
__m128i _mm_aesimc_si128(__m128i v);
Performs the InvMixColumn transformation on a 128-bit round key from \( v \) and returns the result.
**Corresponding Instruction:** AESIMC

__m128i _mm_aeskeygenassist_si128(__m128i ckey, const int rcon);
Assists in AES round key generation using an 8-bit Round Constant (RCON) specified in rcon operating on 128 bits of data specified in ckey and returns the result.
**Corresponding Instruction:** AESKEYGENASSIST

### Intrinsics for Converting Half Floats

**Overview: Intrinsics to Convert Half Float Types**

The half-float or 16-bit float is a popular type in some application domains. The half-float type is regarded as a storage type because although data is often stored as a half-float, computation is never done on values in these type. Usually values are converted to regular 32-bit floats before any computation.

Support for half-float type is restricted to just conversions to/from 32-bit floats.

The main benefits of using half float type are:
- reduced storage requirements
- less consumption of memory bandwidth and cache
- accuracy and precision adequate for many applications

**Half Float Intrinsics**

The half-float intrinsics are provided to convert half-float values to 32-bit floats for computation purposes and conversely, 32-bit float values to half-float values for data storage purposes.

The intrinsics are translated into library calls that do the actual conversions.

The half-float intrinsics are supported on IA-32 and Intel(R) 64 architectures running on Windows*/Linux*/Mac OS* X operating systems. The minimum processor requirement is an Intel(R) Pentium 4 processor and an operating system supporting Streaming SIMD2 Extensions (SSE2) instructions.
Role of Immediate Byte in Half Float Intrinsic Operations

For all half-float intrinsics an immediate byte controls rounding mode, flush to zero, and other non-volatile set values. The format of the imm8 byte is as shown in the diagram below.

The imm8 value is used for special MXCSR overrides.

In the diagram,
- MBZ = Most significant Bit is Zero; used for error checking
- MS1 = 1 : use MXCSR RC, else use imm8.RC
- SAE = 1 : all exceptions are suppressed
- MS2 = 1 : use MXCSR FTZ/DAZ control, else use imm8.FTZ/DAZ.

The compiler passes the bits to the library function, with error checking - the most significant bit must be zero.

Intrinsics for Converting Half Floats

There are four intrinsics for converting half-floats to 32-bit floats and 32-bit floats to half-floats. The prototypes for these half-float conversion intrinsics are in the emmintrin.h file.

float _cvtsh_ss(unsigned short x, int imm);

This intrinsic takes a half-float value, x, and converts it to a 32-bit float value, which is returned.

unsigned short _cvtss_sh(float x, int imm);

This intrinsic takes a 32-bit float value, x, and converts it to a half-float value, which is returned.

__m128 __m_cvtph_ps(__m128i x, int imm);
This intrinsic takes four packed half-float values and converts them to four 32-bit float values, which are returned. The upper 64-bits of x are ignored. The lower 64-bits are taken as four 16-bit float values for conversion.

```
__m128i _mm_cvtps_ph(_m128 x, int imm);
```

This intrinsic takes four packed 32-bit float values and converts them to four half-float values, which are returned. The upper 64-bits in the returned result are all zeros. The lower 64-bits contain the four packed 16-bit float values.

**Intrinsics for Cross-processor Implementation**

**Overview: Intrinsics Cross-processor Implementation**

This section provides a series of tables that compare intrinsics performance across architectures. Before implementing intrinsics across architectures, please note the following.

- Intrinsics may generate code that does not run on all Intel Architecture processors. You should therefore use `CPUID` to detect the processor and generate the appropriate code.
- Implement intrinsics by processor family, not by specific processor. The guiding principle for which processor family (IA-32 architecture-based processor or Itanium® processor) the intrinsic is implemented on is performance, not compatibility. Where there is added performance on both families, the intrinsic is identical.

**Intrinsics For Implementation Across All IA**

The following intrinsics provide significant performance gain over a non-intrinsic-based code equivalent.

```
int abs(int)
long labs(long)
unsigned long _lrotl(unsigned long value, int shift)
unsigned long _lrotr(unsigned long value, int shift)
```
unsigned int _rotl(unsigned int value, int shift)
unsigned int _rotr(unsigned int value, int shift)
__int64 __i64_rotl(__int64 value, int shift)
__int64 __i64_rotr(__int64 value, int shift)
double fabs(double)
double log(double)
float logf(float)
double log10(double)
float log10f(float)
double exp(double)
float expf(float)
double pow(double, double)
float powf(float, float)
double sin(double)
float sinf(float)
double cos(double)
float cosf(float)
double tan(double)
float tanf(float)
double acos(double)
float acosf(float)
double acosh(double)
float acoshf(float)
double asin(double)
float asinf(float)
double asinh(double)
float asinhf(float)
double atan(double)
float atanf(float)
double atanh(double)
float atanhf(float)
float cabs(double)*
double ceil(double)
float ceilf(float)
double cosh(double)
float coshf(float)
float fabsf(float)
double floor(double)
float floorf(float)
double fmod(double)
float fmodf(float)
double hypot(double, double)
float hypotf(float)
double rint(double)
float rintf(float)
double sinh(double)
float sinhf(float)
float sqrtf(float)
double tanh(double)
float tanhf(float)
char *__strset(char *, _int32)
void *__memcmp(const void *cs, const void *ct, size_t n)
void *__memcpy(void *s, const void *ct, size_t n)
void *__memset(void * s, int c, size_t n)
char *Strcat(char * s, const char * ct)
int *strcmp(const char *, const char *)
char *strcpy(char * s, const char * ct)
size_t strlen(const char * cs)
int strncmp(char *, char *, int)
int strncpy(char *, char *, int)
void *__alloca(int)
int __setjmp(jmp_buf)
_exception_code(void)
_exception_info(void)
_abnormal_termination(void)
void _enable()
void _disable()
int _bswap(int)
int _in_byte(int)
int _in_dword(int)
int _in_word(int)
int _inp(int)
int _inpd(int)
int _inpw(int)
int _out_byte(int, int)
int _out_dword(int, int)
int _out_word(int, int)
int _outp(int, int)
int _outpd(int, int)
int _outpw(int, int)
unsigned short _rotwl(unsigned short val, int count)
unsigned short _rotwr(unsigned short val, int count)

**MMX™ Technology Intrinsics Implementation**

**Key to the table entries**

- **A** = Expected to give significant performance gain over non-intrinsic-based code equivalent.
- **B** = Non-intrinsic-based source code would be better; the intrinsic's implementation may map directly to native instructions, but they offer no significant performance gain.
- **C** = Requires contorted implementation for particular microarchitecture. Will result in very poor performance if used.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>MMX™ Technology SSE</th>
<th>IA-64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_empty</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cvtsi32_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cvtsi64_si32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX™ Technology SSE</td>
<td>IA-64 Architecture</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>_mm_packs_pil6</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_packs_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_packs_pul6</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpackhi_pil8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpackhi_pil6</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpackhi_pil32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpacklo_pil8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpacklo_pil16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpacklo_pil32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_add_pil8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_add_pil16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_add_pil32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pil8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pil16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pil32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pu8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pul8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_adds_pul16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pil8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pil16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pil32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pu8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pul8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_pul16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_subs_pil8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_subs_pil16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_subs_pu8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX™ Technology SSE</td>
<td>IA-64 Architecture</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>_mm_subs_pu16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_madd_pi16</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_mulhi_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_mullo_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sll_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_slli_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sll_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_slli_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sll_pi64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_slli_pi64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sra_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srai_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sra_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srai_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srl_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srli_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srli_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srli_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srl_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_srli_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_and_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_andnot_si64</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>
**Streaming SIMD Extensions Intrinsics Implementation**

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>MMX™ Technology SSE</th>
<th>IA-64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_or_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_xor_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpeq_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpeq_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpeq_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpgt_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpgt_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpgt_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setzero_si64</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set_pi16</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_set_pi8</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_set1_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set1_pi16</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set1_pi8</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setr_pi32</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setr_pi16</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_setr_pi8</td>
<td>A</td>
<td>C</td>
</tr>
</tbody>
</table>

_mm_empty is implemented in IA-64 instructions as a NOP for source compatibility only.
Regular Streaming SIMD Extensions (SSE) intrinsics work on 4 32-bit single precision values. On IA-64 architecture-based systems, basic operations like add and compare require two SIMD instructions. All can be executed in the same cycle so the throughput is one basic SSE operation per cycle or 4 32-bit single precision operations per cycle.

**Key to the table entries**

- **A** = Expected to give significant performance gain over non-intrinsic-based code equivalent.
- **B** = Non-intrinsic-based source code would be better; the intrinsic's implementation may map directly to native instructions but they offer no significant performance gain.
- **C** = Requires contorted implementation for particular microarchitecture. Will result in very poor performance if used.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>MMX(TM) Technology</th>
<th>SSE SSE2</th>
<th>IA-64 Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_add_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sub_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_sub_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_mul_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_mul_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_div_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_div_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sqrt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_sqrt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_rcp_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>SSE</td>
<td>SSE2</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>_mm_rcp_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_rsqrt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_rsqrt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_min_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_min_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_max_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_max_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_and_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_andnot_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_or_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_xor_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpeq_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpeq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmplt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmplt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmple_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmple_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpgt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpgt_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpge_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpge_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>SSE</td>
<td>SSE2</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>SSE</td>
<td>SSE2</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------------------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>_mm_ucomigt_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_ucomige_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_ucomineq_ss</td>
<td>N/A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cvtss_s32</td>
<td>N/A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cvtps_pi32</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cvtts_s32</td>
<td>N/A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cvtts_pi32</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_cvtssi_s32</td>
<td>N/A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>_mm_cvtspi32</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_cvtsp16</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_cvtspu16</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_cvtspi8</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_cvtpu8</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_cvtpi32x2</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_cvtps_pi16</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_cvtps_8</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_move_ss</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_shuffle_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpackhi_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_unpacklo_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_movehl_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_movelh_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>SSE</td>
<td>SSE2</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>_mm_movemask_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_getcsr</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setcsr</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_loadh_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_loadl_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_load_ss</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_load1_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_load_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_loadu_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_loadr_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_storeh_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_storel_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_store_ss</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_store_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_store1_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_storeu_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_storer_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set_ss</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set1_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_set_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setr_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_setzero_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>MMX(TM) Technology</td>
<td>SSE SSE2</td>
<td>IA-64 Architecture</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------</td>
<td>---------</td>
<td>-------------------</td>
</tr>
<tr>
<td>_mm_prefetch</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_stream_pi</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_stream_ps</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sfence</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_extract_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_insert_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_max_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_max_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_min_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_min_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_movemask_pi8</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_mulhi_pu16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_shuffle_pi16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_maskmove_si64</td>
<td>N/A</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>_mm_avg_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_avg_pu16</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>_mm_sad_pu8</td>
<td>N/A</td>
<td>A</td>
<td>A</td>
</tr>
</tbody>
</table>

**Streaming SIMD Extensions 2 Intrinsics Implementation**

On processors that do not support SSE2 instructions but do support MMX Technology, you can use the `sse2mmx.h` emulation pack to enable support for SSE2 instructions. You can use the `sse2mmx.h` header file for the following processors:
• Intel® Itanium® processor
• Intel® Pentium® III processor
• Intel® Pentium® II processor
• Intel® Pentium® processors with MMX™ Technology

**Intrinsics for Data Alignment, Memory Allocation, and Inline Assembly**

**Overview: Data Alignment, Memory Allocation Intrinsics, and Inline Assembly**

This section describes features that support usage of the intrinsics. The following topics are described:

- Alignment Support
- Allocating and Freeing Aligned Memory Blocks
- Inline Assembly

**Alignment Support**

Aligning data improves the performance of intrinsics. When using the Streaming SIMD Extensions, you should align data to 16 bytes in memory operations. Specifically, you must align `__m128` objects as addresses passed to the `_mm_load` and `_mm_store` intrinsics. If you want to declare arrays of floats and treat them as `__m128` objects by casting, you need to ensure that the float arrays are properly aligned.

Use `__declspec(align)` to direct the compiler to align data more strictly than it otherwise would. For example, a data object of type `int` is allocated at a byte address which is a multiple of 4 by default. However, by using `__declspec(align)`, you can direct the compiler to instead use an address which is a multiple of 8, 16, or 32 with the following restriction on IA-32:

- 16-byte addresses can be locally or statically allocated

You can use this data alignment support as an advantage in optimizing cache line usage. By clustering small objects that are commonly used together into a `struct`, and forcing the `struct` to be allocated at the beginning of a cache line,
you can effectively guarantee that each object is loaded into the cache as soon as any one is accessed, resulting in a significant performance benefit. The syntax of this extended-attribute is as follows:

\texttt{align(n)}

where \( n \) is an integral power of 2, up to 4096. The value specified is the requested alignment.

\begin{itemize}
  \item \textbf{Caution}
  \begin{itemize}
    \item In this release, \texttt{declspec(align(8))} does not function correctly.
    \item Use \texttt{declspec(align(16))} instead.
  \end{itemize}
\end{itemize}

\begin{itemize}
  \item \textbf{Note}
  \begin{itemize}
    \item If a value is specified that is less than the alignment of the affected data type, it has no effect. In other words, data is aligned to the maximum of its own alignment or the alignment specified with \texttt{declspec(align)}.
  \end{itemize}
\end{itemize}

You can request alignments for individual variables, whether of static or automatic storage duration. (Global and static variables have static storage duration; local variables have automatic storage duration by default.) You cannot adjust the alignment of a parameter, nor a field of a \texttt{struct} or \texttt{class}. You can, however, increase the alignment of a \texttt{struct} (or \texttt{union} or \texttt{class}), in which case every object of that type is affected.

As an example, suppose that a function uses local variables \( i \) and \( j \) as subscripts into a 2-dimensional array. They might be declared as follows:

\begin{verbatim}
int i, j;
\end{verbatim}

These variables are commonly used together. But they can fall in different cache lines, which could be detrimental to performance. You can instead declare them as follows:

\begin{verbatim}
declspec(align(16)) struct { int i, j; } sub;
\end{verbatim}

The compiler now ensures that they are allocated in the same cache line. In C++, you can omit the \texttt{struct} variable name (written as sub in the previous
example). In C, however, it is required, and you must write references to \( i \) and \( j \) as \textit{sub.i} and \textit{sub.j}.

If you use many functions with such subscript pairs, it is more convenient to declare and use a \textit{struct} type for them, as in the following example:

```cpp
typedef struct __declspec(align(16)) { int i, j; } Sub;
```

By placing the \textit{__declspec(align)} after the keyword \textit{struct}, you are requesting the appropriate alignment for all objects of that type. Note that allocation of parameters is unaffected by \textit{__declspec(align)}. (If necessary, you can assign the value of a parameter to a local variable with the appropriate alignment.)

You can also force alignment of global variables, such as arrays:

```cpp
__declspec(align(16)) float array[1000];
```

### Allocating and Freeing Aligned Memory Blocks

Use the \textit{_mm_malloc} and \textit{_mm_free} intrinsics to allocate and free aligned blocks of memory. These intrinsics are based on \textit{malloc} and \textit{free}, which are in the \textit{libirc.a} library. You need to include \textit{malloc.h}. The syntax for these intrinsics is as follows:

```cpp
void* _mm_malloc (int size, int align)
void _mm_free (void *p)
```

The \textit{_mm_malloc} routine takes an extra parameter, which is the alignment constraint. This constraint must be a power of two. The pointer that is returned from \textit{_mm_malloc} is guaranteed to be aligned on the specified boundary.

\[\text{Note}\]

Memory that is allocated using \textit{_mm_malloc} must be freed using \textit{_mm_free}. Calling \textit{free} on memory allocated with \textit{_mm_malloc} or calling \textit{_mm_free} on memory allocated with \textit{malloc} will cause unpredictable behavior.

### Inline Assembly
Microsoft Style Inline Assembly

The Intel® C++ Compiler supports Microsoft-style inline assembly with the -use-msasm compiler option. See your Microsoft documentation for the proper syntax.

GNU*-like Style Inline Assembly (IA-32 architecture and Intel(R) 64 architecture only)

The Intel® C++ Compiler supports GNU-like style inline assembly. The syntax is as follows:

```
asm-keyword [ volatile-keyword ] ( asm-template [ asm-interface ] ) ;
```

The Intel C++ Compiler also supports mixing UNIX and Microsoft style asms. Use the __asm__ keyword for GNU-style ASM when using the -use_msasm switch.

**Note**

The Intel C++ Compiler supports gcc-style inline ASM if the assembler code uses AT&T* System V/386 syntax.

When compiling an assembly statement on Linux*, the compiler simply emits the asm-template to the assembly file after making any necessary operand substitutions. The compiler then calls the GNU assembler to generate machine code. In contrast, on Windows* the compiler itself must assemble the text contained in the asm-template string into machine code. In essence, the compiler contains a built-in assembler.

The compiler's built-in assembler supports the GNU .byte directive but does not support other functionality of the GNU assembler, so there are limitations in the contents of the asm-template. The following assembler features are not currently supported.

- Directives other than the .byte directive
- Symbols*

**Note**
Direct symbol references in the asm-template are not supported. To access a C++ object, use the asm-interface with a substitution directive.

Example

Incorrect method for accessing a C++ object:
```c
__asm__('addl $5, _x');
```
Proper method for accessing a C++ object:
```c
__asm__('addl $5, %0 : "+rm" (x));
```

Additionally, there are some restrictions on the usage of labels. The compiler only allows local labels, and only references to labels within the same assembly statement are permitted. A local label has the form "N:", where N is a non-negative integer. N does not have to be unique, even within the same assembly statement. To reference the most recent definition of label N, use “Nb”. To reference the next definition of label N, use “Nf”. In this context, “b” means backward and “f” means forward. For more information, refer to the GNU assembler documentation.

GNU-style inline assembly statements on Windows* use the same assembly instruction format as on Linux*. This means that destination operands are on the right and source operands are on the left. This operand order is the reverse of Intel assembly syntax.

Due to the limitations of the compiler's built-in assembler, many assembly statements that compile and run on Linux* will not compile on Windows*. On the other hand, assembly statements that compile and run on Windows* should also compile and run on Linux*.

This feature provides a high-performance alternative to Microsoft-style inline assembly statements when portability between Windows*, Linux*, and Mac OS* X is important. Its intended use is in small primitives where high-performance integration with the surrounding C++ code is essential.

```c
#ifdef _WIN64
#define INT64_PRINTF_FORMAT "I64"
#else
#define __int64 long long
#define INT64_PRINTF_FORMAT "L"
#endif
```
Intel® C++ Compiler User and Reference Guides
#include <stdio.h>
typedef struct {
__int64 lo64;
__int64 hi64;
} my_i128;
#define ADD128(out, in1, in2)
__asm__("addq %2, %0; adcq %3, %1" :
"=r"(out.lo64), "=r"(out.hi64) :
"emr" (in2.lo64), "emr"(in2.hi64),
"0" (in1.lo64), "1" (in1.hi64));

\
\
\
\

extern int
main()
{
my_i128 val1, val2, result;
val1.lo64 = ~0;
val1.hi64 = 0;
val2.hi64 = 65;
ADD128(result, val1, val2);
printf("0x%016" INT64_PRINTF_FORMAT "x%016"
"x\n",
val1.hi64, val1.lo64);

INT64_PRINTF_FORMAT

printf("+0x%016" INT64_PRINTF_FORMAT "x%016" INT64_PRINTF_FORMAT
"x\n",
val2.hi64, val2.lo64);
printf("------------------------------------\n");
printf("0x%016" INT64_PRINTF_FORMAT "x%016" INT64_PRINTF_FORMAT
"x\n",
result.hi64, result.lo64);
return 0;
}

This example, written for Intel(R) 64 architecture, shows how to use a GNU-style
inline assembly statement to add two 128-bit integers. In this example, a 128-bit
integer is represented as two __int64 objects in the my_i128 structure. The
inline assembly statement used to implement the addition is contained in the
ADD128 macro, which takes 3 my_i128 arguments representing 3 128-bit
integers. The first argument is the output. The next two arguments are the inputs.
The example compiles and runs using the Intel Compiler on Linux* or Windows*,
producing the following output.
0x0000000000000000ffffffffffffffff
+ 0x00000000000000410000000000000001
-----------------------------------+ 0x00000000000000420000000000000000

1274


In the GNU-style inline assembly implementation, the asm interface specifies all the inputs, outputs, and side effects of the asm statement, enabling the compiler to generate very efficient code.

```
mov       r13, 0xffffffffffffffff
mov       r12, 0x000000000
add       r13, 1
adc       r12, 65
```

It is worth noting that when the compiler generates an assembly file on Windows*, it uses Intel syntax even though the assembly statement was written using Linux* assembly syntax.

The compiler moves in1.lo64 into a register to match the constraint of operand 4. Operand 4's constraint of "0" indicates that it must be assigned the same location as output operand 0. And operand 0's constraint is "=r", indicating that it must be assigned an integer register. In this case, the compiler chooses r13. In the same way, the compiler moves in1.hi64 into register r12.

The constraints for input operands 2 and 3 allow the operands to be assigned a register location ("r"), a memory location ("m"), or a constant signed 32-bit integer value ("e"). In this case, the compiler chooses to match operands 2 and 3 with the constant values 1 and 65, enabling the `add` and `adc` instructions to utilize the "register-immediate" forms.

The same operation is much more expensive using a Microsoft-style inline assembly statement, because the interface between the assembly statement and the surrounding C++ code is entirely through memory. Using Microsoft assembly, the ADD128 macro might be written as follows.

```
#define ADD128(out, in1, in2)                      
{                                              
  __asm mov rax, in1.lo64                     
  __asm mov rdx, in1.hi64                     
  __asm add rax, in2.lo64                     
  __asm adc rdx, in2.hi64                     
  __asm mov out.lo64, rax                     
  __asm mov out.hi64, rdx                    
}                                              
```

The compiler must add code before the assembly statement to move the inputs into memory, and it must add code after the assembly statement to retrieve the outputs from memory. This prevents the compiler from exploiting some optimization opportunities. Thus, the following assembly code is produced.
The operation that took only 4 instructions and 0 memory references using GNU-style inline assembly takes 12 instructions with 12 memory references using Microsoft-style inline assembly.

### Intrinsic for IA-64 Architecture

#### Overview: Intrinsics for IA-64 Instructions

This section lists and describes the native intrinsics for IA-64 instructions. These intrinsics cannot be used on the IA-32 architecture. These intrinsics give programmers access to IA-64 instructions that cannot be generated using the standard constructs of the C and C++ languages.

The prototypes for these intrinsics are in the `ia64intrin.h` header file.

The Intel® Itanium® processor does not support SSE2 intrinsics. However, you can use the `sse2mmx.h` emulation pack to enable support for SSE2 instructions on IA-64 architecture.

For information on how to use SSE intrinsics on IA-64 architecture, see [Using Streaming SIMD Extensions on IA-64 Architecture](#).

For information on how to use MMX (TM) technology intrinsics on IA-64 architecture, see [MMX(TM) Technology Intrinsics on IA-64 Architecture](#).

#### Native Intrinsics for IA-64 Instructions

The prototypes for these intrinsics are in the `ia64intrin.h` header file.
**Integer Operations**

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding IA-64 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_m64_dep_mr</td>
<td>Deposit</td>
<td>dep</td>
</tr>
<tr>
<td>_m64_dep_mi</td>
<td>Deposit</td>
<td>dep</td>
</tr>
<tr>
<td>_m64_dep_zr</td>
<td>Deposit</td>
<td>dep.z</td>
</tr>
<tr>
<td>_m64_dep_zi</td>
<td>Deposit</td>
<td>dep.z</td>
</tr>
<tr>
<td>_m64_extr</td>
<td>Extract</td>
<td>extr</td>
</tr>
<tr>
<td>_m64_extru</td>
<td>Extract</td>
<td>extr.u</td>
</tr>
<tr>
<td>_m64_xmal</td>
<td>Multiply and add</td>
<td>xma.l</td>
</tr>
<tr>
<td>_m64_xmalu</td>
<td>Multiply and add</td>
<td>xma.lu</td>
</tr>
<tr>
<td>_m64_xmah</td>
<td>Multiply and add</td>
<td>xma.h</td>
</tr>
<tr>
<td>_m64_xmahu</td>
<td>Multiply and add</td>
<td>xma.hu</td>
</tr>
<tr>
<td>_m64_popcnt</td>
<td>Population Count</td>
<td>popcnt</td>
</tr>
<tr>
<td>_m64_shladd</td>
<td>Shift left and add</td>
<td>shladd</td>
</tr>
<tr>
<td>_m64_shrp</td>
<td>Shift right pair</td>
<td>shrp</td>
</tr>
</tbody>
</table>

**FSR Operations**

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void _fsetc(int amask, int omask)</td>
<td>Sets the control bits of FPSR.sf0. Maps to the fsetc.sf0 r, r instruction. There is no corresponding instruction to read the control bits. Use _mm_getfpsr().</td>
</tr>
<tr>
<td>void _fclrf(void)</td>
<td>Clears the floating point status flags (the 6-bit flags of FPSR.sf0). Maps to the</td>
</tr>
</tbody>
</table>
Intrinsic | Description
---|---
fclrf.sf0 instruction.

__int64 _m64_dep_mr(__int64 r, __int64 s, const int pos, const int len)

The right-justified 64-bit value r is deposited into the value in s at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position pos and extends to the left (toward the most significant bit) the number of bits specified by len.

__int64 _m64_dep_mi(const int v, __int64 s, const int p, const int len)

The sign-extended value v (either all 1s or all 0s) is deposited into the value in s at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position p and extends to the left (toward the most significant bit) the number of bits specified by len.

__int64 _m64_dep_zr(__int64 s, const int pos, const int len)

The right-justified 64-bit value s is deposited into a 64-bit field of all zeros at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position pos and extends to the left (toward the most significant bit) the number of bits specified by len.

__int64 _m64_dep_zi(const int v, const int pos, const int len)

The sign-extended value v (either all 1s or all 0s) is deposited into a 64-bit field of all zeros at an arbitrary bit position and the result is returned. The deposited bit field begins at bit position pos and extends to the left (toward the most significant bit) the number of bits specified by len.

__int64 _m64_extr(__int64 r, const int pos, const int len)
A field is extracted from the 64-bit value $r$ and is returned right-justified and sign extended. The extracted field begins at position $\text{pos}$ and extends $\text{len}$ bits to the left. The sign is taken from the most significant bit of the extracted field.

$$\text{__int64 } _m64_{\text{extru}}(\text{__int64 } r, \text{ const int } \text{pos}, \text{ const int } \text{len})$$

A field is extracted from the 64-bit value $r$ and is returned right-justified and zero extended. The extracted field begins at position $\text{pos}$ and extends $\text{len}$ bits to the left.

$$\text{__int64 } _m64_{\text{xmal}}(\text{__int64 } a, \text{__int64 } b, \text{__int64 } c)$$

The 64-bit values $a$ and $b$ are treated as signed integers and multiplied to produce a full 128-bit signed result. The 64-bit value $c$ is zero-extended and added to the product. The least significant 64 bits of the sum are then returned.

$$\text{__int64 } _m64_{\text{xmalu}}(\text{__int64 } a, \text{__int64 } b, \text{__int64 } c)$$

The 64-bit values $a$ and $b$ are treated as signed integers and multiplied to produce a full 128-bit unsigned result. The 64-bit value $c$ is zero-extended and added to the product. The least significant 64 bits of the sum are then returned.

$$\text{__int64 } _m64_{\text{xmah}}(\text{__int64 } a, \text{__int64 } b, \text{__int64 } c)$$

The 64-bit values $a$ and $b$ are treated as signed integers and multiplied to produce a full 128-bit signed result. The 64-bit value $c$ is zero-extended and added to the product. The most significant 64 bits of the sum are then returned.

$$\text{__int64 } _m64_{\text{xmahu}}(\text{__int64 } a, \text{__int64 } b, \text{__int64 } c)$$

The 64-bit values $a$ and $b$ are treated as unsigned integers and multiplied to produce a full 128-bit unsigned result. The 64-bit value $c$ is zero-extended and added to the product. The most significant 64 bits of the sum are then returned.

$$\text{__int64 } _m64_{\text{popcnt}}(\text{__int64 } a)$$

The number of bits in the 64-bit integer $a$ that have the value 1 are counted, and the resulting sum is returned.

$$\text{__int64 } _m64_{\text{shladd}}(\text{__int64 } a, \text{ const int } \text{count}, \text{__int64 } b)$$

$a$ is shifted to the left by $\text{count}$ bits and then added to $b$. The result is returned.
__int64 _m64_shrp(__int64 a, __int64 b, const int count)

a and b are concatenated to form a 128-bit value and shifted to the right count bits. The least significant 64 bits of the result are returned.

**Lock and Atomic Operation Related Intrinsics**

The prototypes for these intrinsics are in the *ia64intrin.h* header files.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned __int64 _InterlockedExchange8(volatile unsigned char *Target, unsigned __int64 value)</td>
<td>Map to the xchg1 instruction. Atomically write the least significant byte of its 2nd argument to address specified by its 1st argument.</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange8_rel(volatile unsigned char *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Compare and exchange atomically the least significant byte at the address specified by its 1st argument. Maps to the cmpxchg1.rel instruction with appropriate setup.</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange8_acq(volatile unsigned char *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Same as the previous intrinsic, but using acquire semantic.</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedExchange16(volatile unsigned short *Target, unsigned __int64 value)</td>
<td>Map to the xchg2 instruction. Atomically write the least significant word of its 2nd argument to address</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange16_rel(volatile unsigned short *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Compare and exchange atomically the least significant word at the address specified by its 1st argument. Maps to the cmpxchg2.rel instruction with appropriate setup.</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange16_acq(volatile unsigned short *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Same as the previous intrinsic, but using acquire semantic.</td>
</tr>
<tr>
<td>int _InterlockedIncrement(volatile int *addend)</td>
<td>Atomically increment by one the value specified by its argument. Maps to the fetchadd4 instruction.</td>
</tr>
<tr>
<td>int _InterlockedDecrement(volatile int *addend)</td>
<td>Atomically decrement by one the value specified by its argument. Maps to the fetchadd4 instruction.</td>
</tr>
<tr>
<td>int _InterlockedExchange(volatile int *Target, long value)</td>
<td>Do an exchange operation atomically. Maps to the xchg4 instruction.</td>
</tr>
<tr>
<td>int _InterlockedCompareExchange(volatile int *Destination, int Exchange, int</td>
<td>Do a compare and exchange operation</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>`Comparand)</td>
<td>atomically. Maps to the cmpxchg4 instruction with appropriate setup.</td>
</tr>
<tr>
<td>int _InterlockedExchangeAdd(volatile int *addend, int increment)</td>
<td>Use compare and exchange to do an atomic add of the increment value to the addend. Maps to a loop with the cmpxchg4 instruction to guarantee atomicity.</td>
</tr>
<tr>
<td>int _InterlockedAdd(volatile int *addend, int increment)</td>
<td>Same as the previous intrinsic, but returns new value, not the original one.</td>
</tr>
<tr>
<td>void * _InterlockedCompareExchangePointer(void *volatile *Destination, void *Exchange, void *Comparand)</td>
<td>Map the exch8 instruction; Atomically compare and exchange the pointer value specified by its first argument (all arguments are pointers)</td>
</tr>
<tr>
<td>unsigned __int64 _InterlockedExchangeU(volatile unsigned int *Target, unsigned __int64 value)</td>
<td>Atomically exchange the 32-bit quantity specified by the 1st argument. Maps to the xchg4 instruction. Maps to the cmpxchg4.rel instruction with appropriate setup. Atomically compare and exchange the value</td>
</tr>
</tbody>
</table>
### Intrinsic

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>unsigned __int64 _InterlockedCompareExchange_acq(volatile unsigned int *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</code></td>
<td>Same as the previous intrinsic, but map the cmpxchg4.acq instruction.</td>
</tr>
<tr>
<td><code>void _ReleaseSpinLock(volatile int *x)</code></td>
<td>Release spin lock.</td>
</tr>
<tr>
<td><code>__int64 _InterlockedIncrement64(volatile __int64 *addend)</code></td>
<td>Increment by one the value specified by its argument.</td>
</tr>
<tr>
<td></td>
<td>Maps to the fetchadd instruction.</td>
</tr>
<tr>
<td><code>__int64 _InterlockedDecrement64(volatile __int64 *addend)</code></td>
<td>Decrement by one the value specified by its argument.</td>
</tr>
<tr>
<td></td>
<td>Maps to the fetchadd instruction.</td>
</tr>
<tr>
<td><code>__int64 _InterlockedExchange64(volatile __int64 *Target, __int64 value)</code></td>
<td>Do an exchange operation atomically. Maps to the xchg instruction.</td>
</tr>
<tr>
<td><code>unsigned __int64 _InterlockedExchangeU64(volatile unsigned __int64 *Target, unsigned __int64 value)</code></td>
<td>Same as InterlockedExchange64 (for unsigned quantities).</td>
</tr>
<tr>
<td><code>unsigned __int64 _InterlockedCompareExchange64_rel(volatile unsigned __int64 *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</code></td>
<td>Maps to the cmpxchg.rel instruction with appropriate setup. Atomically compare and exchange the value specified by the first argument (a 64-bit pointer).</td>
</tr>
</tbody>
</table>
### Intrinsic

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned __int64 _InterlockedCompareExchange64_acq(volatile unsigned __int64 *Destination, unsigned __int64 Exchange, unsigned __int64 Comparand)</td>
<td>Maps to the cmpxchg.acq instruction with appropriate setup. Atomically compare and exchange the value specified by the first argument (a 64-bit pointer).</td>
</tr>
<tr>
<td>__int64 _InterlockedCompareExchange64(volatile __int64 *Destination, __int64 Exchange, __int64 Comparand)</td>
<td>Same as the previous intrinsic for signed quantities.</td>
</tr>
<tr>
<td>__int64 _InterlockedExchangeAdd64(volatile __int64 *addend, __int64 increment)</td>
<td>Use compare and exchange to do an atomic add of the increment value to the addend. Maps to a loop with the cmpxchg instruction to guarantee atomicity</td>
</tr>
<tr>
<td>__int64 _InterlockedAdd64(volatile __int64 *addend, __int64 increment)</td>
<td>Same as the previous intrinsic, but returns the new value, not the original value. See Note.</td>
</tr>
</tbody>
</table>

**Note**

_InterlockedSub64 is provided as a macro definition based on _InterlockedAdd64.

```c
#define _InterlockedSub64(target, incr) _InterlockedAdd64((target),(-(incr)))
```
Uses `cmpxchg` to do an atomic sub of the `incr` value to the `target`.
Maps to a loop with the `cmpxchg` instruction to guarantee atomicity.

**Load and Store**

You can use the load and store intrinsic to force the strict memory access ordering of specific data objects. This intended use is for the case when the user suppresses the strict memory access ordering by using the `-serialize-volatile` option.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__st1_rel</td>
<td><code>void __st1_rel(void *dst, const char value);</code></td>
<td>Generates an <code>st1.rel</code> instruction.</td>
</tr>
<tr>
<td>__st2_rel</td>
<td><code>void __st2_rel(void *dst, const short value);</code></td>
<td>Generates an <code>st2.rel</code> instruction.</td>
</tr>
<tr>
<td>__st4_rel</td>
<td><code>void __st4_rel(void *dst, const int value);</code></td>
<td>Generates an <code>st4.rel</code> instruction.</td>
</tr>
<tr>
<td>__st8_rel</td>
<td><code>void __st8_rel(void *dst, const __int64 value);</code></td>
<td>Generates an <code>st8.rel</code> instruction.</td>
</tr>
<tr>
<td>__ld1_acq</td>
<td><code>unsigned char __ld1_acq(void *src);</code></td>
<td>Generates an <code>ld1.acq</code> instruction.</td>
</tr>
<tr>
<td>__ld2_acq</td>
<td><code>unsigned short __ld2_acq(void *src);</code></td>
<td>Generates an <code>ld2.acq</code> instruction.</td>
</tr>
<tr>
<td>__ld4_acq</td>
<td><code>unsigned int __ld4_acq(void *src);</code></td>
<td>Generates an <code>ld4.acq</code> instruction.</td>
</tr>
</tbody>
</table>
### Intrinsic

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Prototype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__ld4_acq</td>
<td>unsigned __int64 __ld4_acq(void *src);</td>
<td>Instruction.</td>
</tr>
<tr>
<td>__ld8_acq</td>
<td>unsigned __int64 __ld8_acq(void *src);</td>
<td>Generates an ld8.acq instruction.</td>
</tr>
</tbody>
</table>

### Operating System Related Intrinsics

The prototypes for these intrinsics are in the `ia64intrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsigned __int64 __getReg(const int whichReg)</td>
<td>Gets the value from a hardware register based on the index passed in. Produces a corresponding <code>mov = r</code> instruction. Provides access to the following registers: See Register Names for <code>getReg()</code> and <code>setReg()</code></td>
</tr>
<tr>
<td>void __setReg(const int whichReg, unsigned __int64 value)</td>
<td>Sets the value for a hardware register based on the index passed in. Produces a corresponding <code>mov = r</code> instruction. See Register Names for <code>getReg()</code> and <code>setReg()</code></td>
</tr>
<tr>
<td>unsigned __int64 __getIndReg(const int whichIndReg, __int64 index)</td>
<td>Return the value of an indexed register. The index is the 2nd argument; the register file is the first argument.</td>
</tr>
<tr>
<td>void __setIndReg(const int whichIndReg, __int64 index, unsigned __int64 value)</td>
<td>Copy a value in an indexed register. The index is the 2nd argument; the register file is the first argument.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>void *__ptr64 _rdteb(void)</code></td>
<td>Gets TEB address. The TEB address is kept in r13 and maps to the move r=tp instruction.</td>
</tr>
<tr>
<td><code>void __isrlz(void)</code></td>
<td>Executes the serialize instruction. Maps to the srlz.i instruction.</td>
</tr>
<tr>
<td><code>void __dsrlz(void)</code></td>
<td>Serializes the data. Maps to the srlz.d instruction.</td>
</tr>
<tr>
<td><code>unsigned __int64 __fetchadd4_acq(unsigned int *addend, const int increment)</code></td>
<td>Map the fetchadd4.acq instruction.</td>
</tr>
<tr>
<td><code>unsigned __int64 __fetchadd4_rel(unsigned int *addend, const int increment)</code></td>
<td>Map the fetchadd4.rel instruction.</td>
</tr>
<tr>
<td><code>unsigned __int64 __fetchadd8_acq(unsigned __int64 *addend, const int increment)</code></td>
<td>Map the fetchadd8.acq instruction.</td>
</tr>
<tr>
<td><code>unsigned __int64 __fetchadd8_rel(unsigned __int64 *addend, const int increment)</code></td>
<td>Map the fetchadd8.rel instruction.</td>
</tr>
<tr>
<td><code>void __fwb(void)</code></td>
<td>Flushes the write buffers. Maps to the fwb instruction.</td>
</tr>
<tr>
<td><code>void __ldfs(const int whichFloatReg, void *src)</code></td>
<td>Map the ldfs instruction. Load a single precision value to the specified register.</td>
</tr>
<tr>
<td><code>void __ldfd(const int</code></td>
<td>Map the ldfd instruction. Load a</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>whichFloatReg, void *src)</td>
<td>double precision value to the specified register.</td>
</tr>
<tr>
<td>void __ldfe(const int whichFloatReg, void *src)</td>
<td>Map the ldfe instruction. Load an extended precision value to the specified register.</td>
</tr>
<tr>
<td>void __ldf8(const int whichFloatReg, void *src)</td>
<td>Map the ldf8 instruction.</td>
</tr>
<tr>
<td>void __ldf_fill(const int whichFloatReg, void *src)</td>
<td>Map the ldf.fill instruction.</td>
</tr>
<tr>
<td>void __stfs(void *dst, const int whichFloatReg)</td>
<td>Map the sfts instruction.</td>
</tr>
<tr>
<td>void __stfd(void *dst, const int whichFloatReg)</td>
<td>Map the stfd instruction.</td>
</tr>
<tr>
<td>void __stfe(void *dst, const int whichFloatReg)</td>
<td>Map the stfe instruction.</td>
</tr>
<tr>
<td>void __stf8(void *dst, const int whichFloatReg)</td>
<td>Map the stf8 instruction.</td>
</tr>
<tr>
<td>void __stf_spill(void *dst, const int whichFloatReg)</td>
<td>Map the stf.spill instruction.</td>
</tr>
<tr>
<td>void __mf(void)</td>
<td>Executes a memory fence instruction.</td>
</tr>
<tr>
<td></td>
<td>Maps to the mf instruction.</td>
</tr>
<tr>
<td>void __mfa(void)</td>
<td>Executes a memory fence, acceptance form instruction. Maps to the mf.a instruction.</td>
</tr>
<tr>
<td>void __synci(void)</td>
<td>Enables memory synchronization. Maps to the sync.i instruction.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>unsigned __int64 __thash(__int64)</td>
<td>Generates a translation hash entry address. Maps to the <code>thash r = r</code> instruction.</td>
</tr>
<tr>
<td>unsigned __int64 __ttag(__int64)</td>
<td>Generates a translation hash entry tag. Maps to the <code>ttag r=r</code> instruction.</td>
</tr>
<tr>
<td>void __itcd(__int64 pa)</td>
<td>Insert an entry into the data translation cache (Map <code>itc.d</code> instruction).</td>
</tr>
<tr>
<td>void __itci(__int64 pa)</td>
<td>Insert an entry into the instruction translation cache (Map <code>itc.i</code>).</td>
</tr>
<tr>
<td>void __itr(d(__int64 whichTransReg, __int64 pa)</td>
<td>Map the <code>itr.d</code> instruction.</td>
</tr>
<tr>
<td>void __itr(i(__int64 whichTransReg, __int64 pa)</td>
<td>Map the <code>itr.i</code> instruction.</td>
</tr>
<tr>
<td>void __ptce(__int64 va)</td>
<td>Purges the local translation cache. Maps to the <code>ptc.e r, r</code> instruction.</td>
</tr>
<tr>
<td>void __ptcl(__int64 va, __int64 pagesz)</td>
<td>Purges the global translation cache. Maps to the <code>ptc.g r, r</code> instruction.</td>
</tr>
<tr>
<td>void __ptcg(__int64 va, __int64 pagesz)</td>
<td>Purges the global translation cache and ALAT. Maps to the <code>ptc.ga r, r</code> instruction.</td>
</tr>
<tr>
<td>void __ptri(__int64 va, __int64 pagesz)</td>
<td>Purges the translation register. Maps to the <code>ptr.i r, r</code> instruction.</td>
</tr>
<tr>
<td>void __ptrd(__int64 va, __int64 pagesz)</td>
<td>Purges the translation register. Maps to the <code>ptr.d r, r</code> instruction.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>__int64 __tpa(__int64 va)</td>
<td>Map the tpa instruction.</td>
</tr>
<tr>
<td>void __invalat(void)</td>
<td>Invalidates ALAT. Maps to the invala instruction.</td>
</tr>
<tr>
<td>void __invala (void)</td>
<td>Same as void __invalat(void)</td>
</tr>
<tr>
<td>void __invala_gr(const int whichGeneralReg)</td>
<td>whichGeneralReg = 0-127</td>
</tr>
<tr>
<td>void __invala_fr(const int whichFloatReg)</td>
<td>whichFloatReg = 0-127</td>
</tr>
<tr>
<td>void __break(const int)</td>
<td>Generates a break instruction with an immediate.</td>
</tr>
<tr>
<td>void __nop(const int)</td>
<td>Generate a nop instruction.</td>
</tr>
<tr>
<td>void __debugbreak(void)</td>
<td>Generates a Debug Break Instruction fault.</td>
</tr>
<tr>
<td>void __fc(void*)</td>
<td>Flushes a cache line associated with the address given by the argument. Maps to the fc instruction.</td>
</tr>
<tr>
<td>void __sum(int mask)</td>
<td>Sets the user mask bits of PSR. Maps to the sum imm24 instruction.</td>
</tr>
<tr>
<td>void __rum(int mask)</td>
<td>Resets the user mask.</td>
</tr>
<tr>
<td>__int64 _ReturnAddress(void)</td>
<td>Get the caller's address.</td>
</tr>
<tr>
<td>void __lfetch(int lfhint, void const *y)</td>
<td>Generate the lfetch.lfhint instruction. The value of the first argument specifies the hint type.</td>
</tr>
<tr>
<td>void __lfetch_fault(int lfhint, void const *y)</td>
<td>Generate the lfetch.fault.lfhint instruction. The value of the first argument specifies the hint type.</td>
</tr>
</tbody>
</table>
### Intrinsic Description

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void __lfetch_excl(int lfhint, void const *y)</td>
<td>Generate the <code>lfetch.excl.lfhint</code> instruction. The value {0</td>
</tr>
<tr>
<td>void __lfetch_fault_excl(int lfhint, void const *y)</td>
<td>Generate the <code>lfetch.fault.excl.lfhint</code> instruction. The value of the first argument specifies the hint type.</td>
</tr>
<tr>
<td>unsigned int __cacheSize(unsigned int cacheLevel)</td>
<td>See <code>__cacheSize()</code> intrinsic under Intrinsics for Use Across All IA&gt;Miscellaneous Intrinsics.</td>
</tr>
<tr>
<td>void __memory_barrier(void)</td>
<td>Creates a barrier across which the compiler will not schedule any data access instruction. The compiler may allocate local data in registers across a memory barrier, but not global data.</td>
</tr>
<tr>
<td>void __ssm(int mask)</td>
<td>Sets the system mask. Maps to the <code>ssm imm24</code> instruction.</td>
</tr>
<tr>
<td>void __rsm(int mask)</td>
<td>Resets the system mask bits of PSR. Maps to the <code>rsm imm24</code> instruction.</td>
</tr>
</tbody>
</table>

### Conversion Intrinsics

The prototypes for these intrinsics are in the `ia64intrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__int64 _m_to_int64(__m64 a)</td>
<td>Convert a of type <code>__m64</code> to type</td>
</tr>
</tbody>
</table>
**Intrinsic**

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__int64</td>
<td>Translates to <strong>nop</strong> since both types reside in the same register for systems based on IA-64 architecture.</td>
</tr>
<tr>
<td>__m64 _m_from_int64(__int64 a)</td>
<td>Convert a of type __int64 to type __m64. Translates to <strong>nop</strong> since both types reside in the same register for systems based on IA-64 architecture.</td>
</tr>
<tr>
<td>__int64</td>
<td>Convert its double precision argument to a signed integer.</td>
</tr>
<tr>
<td>__round_double_to_int64(double d)</td>
<td>Map the getf.exp instruction and return the 16-bit exponent and the sign of its operand.</td>
</tr>
<tr>
<td>unsigned __int64 __getf_exp(double d)</td>
<td></td>
</tr>
</tbody>
</table>

**Register Names for getReg() and setReg()**

The prototypes for getReg() and setReg() intrinsics are in the *ia64regs.h* header file.

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_IP</td>
<td>1016</td>
</tr>
<tr>
<td>_IA64_REG_PSR</td>
<td>1019</td>
</tr>
<tr>
<td>_IA64_REG_PSR_L</td>
<td>1019</td>
</tr>
</tbody>
</table>

**General Integer Registers**

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_GP</td>
<td>1025</td>
</tr>
</tbody>
</table>
## General Integer Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_SP</td>
<td>1036</td>
</tr>
<tr>
<td>_IA64_REG_TP</td>
<td>1037</td>
</tr>
</tbody>
</table>

## Application Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_AR_KR0</td>
<td>3072</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR1</td>
<td>3073</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR2</td>
<td>3074</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR3</td>
<td>3075</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR4</td>
<td>3076</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR5</td>
<td>3077</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR6</td>
<td>3078</td>
</tr>
<tr>
<td>_IA64_REG_AR_KR7</td>
<td>3079</td>
</tr>
<tr>
<td>_IA64_REG_AR_RSC</td>
<td>3088</td>
</tr>
<tr>
<td>_IA64_REG_AR_BSP</td>
<td>3089</td>
</tr>
<tr>
<td>_IA64_REG_AR_BSPSTORE</td>
<td>3090</td>
</tr>
<tr>
<td>_IA64_REG_AR_RNAT</td>
<td>3091</td>
</tr>
<tr>
<td>_IA64_REG_AR_FCR</td>
<td>3093</td>
</tr>
<tr>
<td>_IA64_REG_AR_EFLAG</td>
<td>3096</td>
</tr>
<tr>
<td>_IA64_REG_AR_CSD</td>
<td>3097</td>
</tr>
<tr>
<td>_IA64_REG_AR_SSD</td>
<td>3098</td>
</tr>
<tr>
<td>_IA64_REG_AR_CFLAG</td>
<td>3099</td>
</tr>
<tr>
<td>_IA64_REG_AR_FSR</td>
<td>3100</td>
</tr>
</tbody>
</table>
## Application Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_AR_FIR</td>
<td>3101</td>
</tr>
<tr>
<td>_IA64_REG_AR_FDR</td>
<td>3102</td>
</tr>
<tr>
<td>_IA64_REG_AR_CCV</td>
<td>3104</td>
</tr>
<tr>
<td>_IA64_REG_AR_UNAT</td>
<td>3108</td>
</tr>
<tr>
<td>_IA64_REG_AR_FPSR</td>
<td>3112</td>
</tr>
<tr>
<td>_IA64_REG_AR_ITC</td>
<td>3116</td>
</tr>
<tr>
<td>_IA64_REG_AR_PFS</td>
<td>3136</td>
</tr>
<tr>
<td>_IA64_REG_AR_LC</td>
<td>3137</td>
</tr>
<tr>
<td>_IA64_REG_AR_EC</td>
<td>3138</td>
</tr>
</tbody>
</table>

## Control Registers

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_CR_DCR</td>
<td>4096</td>
</tr>
<tr>
<td>_IA64_REG_CR_ITM</td>
<td>4097</td>
</tr>
<tr>
<td>_IA64_REG_CR_IVA</td>
<td>4098</td>
</tr>
<tr>
<td>_IA64_REG_CR_PTA</td>
<td>4104</td>
</tr>
<tr>
<td>_IA64_REG_CR_IPSR</td>
<td>4112</td>
</tr>
<tr>
<td>_IA64_REG_CR_ISR</td>
<td>4113</td>
</tr>
<tr>
<td>_IA64_REG_CR_IIP</td>
<td>4115</td>
</tr>
<tr>
<td>_IA64_REG_CR_IFA</td>
<td>4116</td>
</tr>
<tr>
<td>_IA64_REG_CR_ITIR</td>
<td>4117</td>
</tr>
<tr>
<td>_IA64_REG_CR_IIPA</td>
<td>4118</td>
</tr>
<tr>
<td>_IA64_REG_CR_IFS</td>
<td>4119</td>
</tr>
<tr>
<td>Control Registers</td>
<td></td>
</tr>
<tr>
<td>-------------------</td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>whichReg</td>
</tr>
<tr>
<td>_IA64_REG_CR_IIM</td>
<td>4120</td>
</tr>
<tr>
<td>_IA64_REG_CR_IHA</td>
<td>4121</td>
</tr>
<tr>
<td>_IA64_REG_CR_LID</td>
<td>4160</td>
</tr>
<tr>
<td>_IA64_REG_CR_IVR</td>
<td>4161</td>
</tr>
<tr>
<td>_IA64_REG_CR_TPR</td>
<td>4162</td>
</tr>
<tr>
<td>_IA64_REG_CR_EOI</td>
<td>4163</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR0</td>
<td>4164</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR1</td>
<td>4165</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR2</td>
<td>4166</td>
</tr>
<tr>
<td>_IA64_REG_CR_IRR3</td>
<td>4167</td>
</tr>
<tr>
<td>_IA64_REG_CR_ITV</td>
<td>4168</td>
</tr>
<tr>
<td>_IA64_REG_CR_PMV</td>
<td>4169</td>
</tr>
<tr>
<td>_IA64_REG_CR_CMCV</td>
<td>4170</td>
</tr>
<tr>
<td>_IA64_REG_CR_LRR0</td>
<td>4176</td>
</tr>
<tr>
<td>_IA64_REG_CR_LRR1</td>
<td>4177</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Indirect Registers for getIndReg() and setIndReg()</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>_IA64_REG_INDR_CPUID</td>
</tr>
<tr>
<td>_IA64_REG_INDR,DBR</td>
</tr>
<tr>
<td>_IA64_REG_INDR_IBR</td>
</tr>
<tr>
<td>_IA64_REG_INDR_PKR</td>
</tr>
<tr>
<td>_IA64_REG_INDR_PMC</td>
</tr>
</tbody>
</table>
Indirect Registers for `getIndReg()` and `setIndReg()`

<table>
<thead>
<tr>
<th>Name</th>
<th>whichReg</th>
</tr>
</thead>
<tbody>
<tr>
<td>_IA64_REG_INDR_PMD</td>
<td>9005</td>
</tr>
<tr>
<td>_IA64_REG_INDR_RR</td>
<td>9006</td>
</tr>
<tr>
<td>_IA64_REG_INDR_RESERVED</td>
<td>9007</td>
</tr>
</tbody>
</table>

**Multimedia Addition Intrinsics**

The prototypes for these intrinsics are in the `ia64intrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding IA-64 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_m64_czx1l</td>
<td>Compute Zero Index</td>
<td>czx1.l</td>
</tr>
<tr>
<td>_m64_czx1r</td>
<td>Compute Zero Index</td>
<td>czx1.r</td>
</tr>
<tr>
<td>_m64_czx2l</td>
<td>Compute Zero Index</td>
<td>czx2.l</td>
</tr>
<tr>
<td>_m64_czx2r</td>
<td>Compute Zero Index</td>
<td>czx2.r</td>
</tr>
<tr>
<td>_m64_mix1l</td>
<td>Mix</td>
<td>mix1.l</td>
</tr>
<tr>
<td>_m64_mix1r</td>
<td>Mix</td>
<td>mix1.r</td>
</tr>
<tr>
<td>_m64_mix2l</td>
<td>Mix</td>
<td>mix2.l</td>
</tr>
<tr>
<td>_m64_mix2r</td>
<td>Mix</td>
<td>mix2.r</td>
</tr>
<tr>
<td>_m64_mix4l</td>
<td>Mix</td>
<td>mix4.l</td>
</tr>
<tr>
<td>_m64_mix4r</td>
<td>Mix</td>
<td>mix4.r</td>
</tr>
<tr>
<td>_m64_mux1</td>
<td>Permutation</td>
<td>mux1</td>
</tr>
<tr>
<td>_m64_mux2</td>
<td>Permutation</td>
<td>mux2</td>
</tr>
<tr>
<td>_m64_padd1uus</td>
<td>Parallel add</td>
<td>padd1.uus</td>
</tr>
<tr>
<td>_m64_padd2uus</td>
<td>Parallel add</td>
<td>padd2.uus</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding IA-64 Instruction</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td>_m64_pavg1_nraz</td>
<td>Parallel average</td>
<td>pavg1</td>
</tr>
<tr>
<td>_m64_pavg2_nraz</td>
<td>Parallel average</td>
<td>pavg2</td>
</tr>
<tr>
<td>_m64_pavgsub1</td>
<td>Parallel average subtract</td>
<td>pavgsub1</td>
</tr>
<tr>
<td>_m64_pavgsub2</td>
<td>Parallel average subtract</td>
<td>pavgsub2</td>
</tr>
<tr>
<td>_m64_pmpy2r</td>
<td>Parallel multiply</td>
<td>pmpy2.r</td>
</tr>
<tr>
<td>_m64_pmpy2l</td>
<td>Parallel multiply</td>
<td>pmpy2.l</td>
</tr>
<tr>
<td>_m64_pmpyshr2</td>
<td>Parallel multiply and shift</td>
<td>pmpyshr2</td>
</tr>
<tr>
<td>_m64_pshladd2</td>
<td>Parallel shift left and add</td>
<td>pshladd2</td>
</tr>
<tr>
<td>_m64_pshradd2</td>
<td>Parallel shift right and add</td>
<td>pshradd2</td>
</tr>
<tr>
<td>_m64_psub1uus</td>
<td>Parallel subtract</td>
<td>psub1.uus</td>
</tr>
<tr>
<td>_m64_psub2uus</td>
<td>Parallel subtract</td>
<td>psub2.uus</td>
</tr>
</tbody>
</table>

__int64 _m64_czx1l(__m64 a)

The 64-bit value a is scanned for a zero element from the most significant element to the least significant element, and the index of the first zero element is returned. The element width is 8 bits, so the range of the result is from 0 - 7. If no zero element is found, the default result is 8.

__int64 _m64_czx1r(__m64 a)

The 64-bit value a is scanned for a zero element from the least significant element to the most significant element, and the index of the first zero element is returned. The element width is 8 bits, so the range of the result is from 0 - 7. If no zero element is found, the default result is 8.
__int64 _m64_czx2l(__m64 a)

The 64-bit value \texttt{a} is scanned for a zero element from the most significant element to the least significant element, and the index of the first zero element is returned. The element width is 16 bits, so the range of the result is from 0 - 3. If no zero element is found, the default result is 4.

__int64 _m64_czx2r(__m64 a)

The 64-bit value \texttt{a} is scanned for a zero element from the least significant element to the most significant element, and the index of the first zero element is returned. The element width is 16 bits, so the range of the result is from 0 - 3. If no zero element is found, the default result is 4.

__m64 _m64_mix1l(__m64 a, __m64 b)

Interleave 64-bit quantities \texttt{a} and \texttt{b} in 1-byte groups, starting from the left, as shown in Figure 1, and return the result.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure1.png}
\caption{Interleave 64-bit quantities in 1-byte groups, starting from the left.}
\end{figure}

__m64 _m64_mix1r(__m64 a, __m64 b)

Interleave 64-bit quantities \texttt{a} and \texttt{b} in 1-byte groups, starting from the right, as shown in Figure 2, and return the result.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure2.png}
\caption{Interleave 64-bit quantities in 1-byte groups, starting from the right.}
\end{figure}

__m64 _m64_mix2l(__m64 a, __m64 b)

Interleave 64-bit quantities \texttt{a} and \texttt{b} in 2-byte groups, starting from the left, as shown in Figure 3, and return the result.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure3.png}
\caption{Interleave 64-bit quantities in 2-byte groups, starting from the left.}
\end{figure}
__m64 _m64_mix2r(__m64 a, __m64 b)
Interleave 64-bit quantities \( a \) and \( b \) in 2-byte groups, starting from the right, as shown in Figure 4, and return the result.

__m64 _m64_mix4l(__m64 a, __m64 b)
Interleave 64-bit quantities \( a \) and \( b \) in 4-byte groups, starting from the left, as shown in Figure 5, and return the result.

__m64 _m64_mix4r(__m64 a, __m64 b)
Interleave 64-bit quantities \( a \) and \( b \) in 4-byte groups, starting from the right, as shown in Figure 6, and return the result.

__m64 _m64_mux1(__m64 a, const int n)
Based on the value of \( n \), a permutation is performed on \( a \) as shown in Figure 7, and the result is returned. Table 1 shows the possible values of \( n \).
Table 1. Values of \( n \) for \texttt{m64\_mux1} Operation

<table>
<thead>
<tr>
<th></th>
<th>( n )</th>
</tr>
</thead>
<tbody>
<tr>
<td>@br cst</td>
<td>0</td>
</tr>
<tr>
<td>@mix</td>
<td>8</td>
</tr>
<tr>
<td>@shuf</td>
<td>9</td>
</tr>
<tr>
<td>@alt</td>
<td>0xA</td>
</tr>
<tr>
<td>@rev</td>
<td>0xB</td>
</tr>
</tbody>
</table>

\texttt{__m64\_m64\_mux2(__m64 a, const int n)}

Based on the value of \( n \), a permutation is performed on \( a \) as shown in Figure 8, and the result is returned.
__m64 _m64_pavgsub1(__m64 a, __m64 b)

The unsigned data elements (bytes) of \( b \) are subtracted from the unsigned data elements (bytes) of \( a \) and the results of the subtraction are then each independently shifted to the right by one position. The high-order bits of each element are filled with the borrow bits of the subtraction.

__m64 _m64_pavgsub2(__m64 a, __m64 b)

The unsigned data elements (double bytes) of \( b \) are subtracted from the unsigned data elements (double bytes) of \( a \) and the results of the subtraction are then each independently shifted to the right by one position. The high-order bits of each element are filled with the borrow bits of the subtraction.

__m64 _m64_pmpy2l(__m64 a, __m64 b)

Two signed 16-bit data elements of \( a \), starting with the most significant data element, are multiplied by the corresponding two signed 16-bit data elements of \( b \), and the two 32-bit results are returned as shown in Figure 9.
__m64 _m64_pmpy2r(__m64 a, __m64 b)

Two signed 16-bit data elements of \( a \), starting with the least significant data element, are multiplied by the corresponding two signed 16-bit data elements of \( b \), and the two 32-bit results are returned as shown in Figure 10.

__m64 _m64_pmpyshr2(__m64 a, __m64 b, const int count)

The four signed 16-bit data elements of \( a \) are multiplied by the corresponding signed 16-bit data elements of \( b \), yielding four 32-bit products. Each product is then shifted to the right count bits and the least significant 16 bits of each shifted product form 4 16-bit results, which are returned as one 64-bit word.

__m64 _m64_pmpyshr2u(__m64 a, __m64 b, const int count)

The four unsigned 16-bit data elements of \( a \) are multiplied by the corresponding unsigned 16-bit data elements of \( b \), yielding four 32-bit products. Each product is then shifted to the right count bits and the least significant 16 bits of each shifted product form 4 16-bit results, which are returned as one 64-bit word.
__m64 _m64_pshladd2(__m64 a, const int count, __m64 b)
a is shifted to the left by count bits and then is added to b. The upper 32 bits of
the result are forced to 0, and then bits [31:30] of b are copied to bits [62:61] of
the result. The result is returned.

__m64 _m64_pshradd2(__m64 a, const int count, __m64 b)
The four signed 16-bit data elements of a are each independently shifted to the
right by count bits (the high order bits of each element are filled with the initial
value of the sign bits of the data elements in a); they are then added to the four
signed 16-bit data elements of b. The result is returned.

__m64 _m64_padd1uus(__m64 a, __m64 b)
a is added to b as eight separate byte-wide elements. The elements of a are
treated as unsigned, while the elements of b are treated as signed. The results
are treated as unsigned and are returned as one 64-bit word.

__m64 _m64_padd2uus(__m64 a, __m64 b)
a is added to b as four separate 16-bit wide elements. The elements of a are
treated as unsigned, while the elements of b are treated as signed. The results
are treated as unsigned and are returned as one 64-bit word.

__m64 _m64_psub1uus(__m64 a, __m64 b)
a is subtracted from b as eight separate byte-wide elements. The elements of a are
treated as unsigned, while the elements of b are treated as signed. The results
are treated as unsigned and are returned as one 64-bit word.

__m64 _m64_psub2uus(__m64 a, __m64 b)
a is subtracted from b as four separate 16-bit wide elements. The elements of a are
treated as unsigned, while the elements of b are treated as signed. The results
are treated as unsigned and are returned as one 64-bit word.

__m64 _m64_pavg1_nraz(__m64 a, __m64 b)
The unsigned byte-wide data elements of a are added to the unsigned byte-wide
data elements of b and the results of each add are then independently shifted to
the right by one position. The high-order bits of each element are filled with the
carry bits of the sums.

__m64 _m64_pavg2_nraz(__m64 a, __m64 b)

The unsigned 16-bit wide data elements of a are added to the unsigned 16-bit
wide data elements of b and the results of each add are then independently
shifted to the right by one position. The high-order bits of each element are filled
with the carry bits of the sums.

Synchronization Primitives

The synchronization primitive intrinsics provide a variety of operations. Besides
performing these operations, each intrinsic has two key properties:

- the function performed is guaranteed to be atomic
- associated with each intrinsic are certain memory barrier properties that
  restrict the movement of memory references to visible data across the
  intrinsic operation by either the compiler or the processor

For the following intrinsics, <type> is either a 32-bit or 64-bit integer.

Atomic Fetch-and-op Operations

<type> __sync_fetch_and_add(<type> *ptr,<type> val)
<type> __sync_fetch_and_and(<type> *ptr,<type> val)
<type> __sync_fetch_and_nand(<type> *ptr,<type> val)
<type> __sync_fetch_and_or(<type> *ptr,<type> val)
<type> __sync_fetch_and_sub(<type> *ptr,<type> val)
<type> __sync_fetch_and_xor(<type> *ptr,<type> val)

Atomic Op-and-fetch Operations

<type> __sync_add_and_fetch(<type> *ptr,<type> val)
<type> __sync_sub_and_fetch(<type> *ptr,<type> val)
<type> __sync_or_and_fetch(<type> *ptr,<type> val)
<type> __sync_and_and_fetch(<type> *ptr,<type> val)
<type> __sync_nand_and_fetch(<type> *ptr,<type> val)
<type> __sync_xor_and_fetch(<type> *ptr, <type> val)

**Atomic Compare-and-swap Operations**

<type> __sync_val_compare_and_swap(<type> *ptr, <type> old_val, <type> new_val)
int __sync_bool_compare_and_swap(<type> *ptr, <type> old_val, <type> new_val)

**Atomic Synchronize Operation**

void __sync_synchronize (void);

**Atomic Lock-test-and-set Operation**

<type> __sync_lock_test_and_set(<type> *ptr, <type> val)

**Atomic Lock-release Operation**

void __sync_lock_release(<type> *ptr)

**Miscellaneous Intrinsics**

void* __get_return_address(unsigned int level);

This intrinsic yields the return address of the current function. The `level` argument must be a constant value. A value of 0 yields the return address of the current function. Any other value yields a zero return address. On Linux systems, this intrinsic is synonymous with `__builtin_return_address`. The name and the argument are provided for compatibility with GCC*.

void __set_return_address(void* addr);

This intrinsic overwrites the default return address of the current function with the address indicated by its argument. On return from the current invocation, program execution continues at the address provided.

void* __get_frame_address(unsigned int level);

This intrinsic returns the frame address of the current function. The `level` argument must be a constant value. A value of 0 yields the frame address of the
current function. Any other value yields a zero return value. On Linux systems, this intrinsic is synonymous with \texttt{__builtin_frame_address}. The name and the argument are provided for compatibility with GCC*.

\textbf{Intrinsics for Intel® Itanium® 2 Dual-Core Processor 9000 Sequence}

The Intel® Itanium® 2 dual-core processor 9000 sequence supports the intrinsics listed in the table below.

These intrinsics each generate IA-64 instructions. The first alpha-numerical chain in the intrinsic name represents the return type, and the second alpha-numerical chain in the intrinsic name represents the instruction the intrinsic generates. For example, the intrinsic \texttt{__int64 cmp8xchg} generates the \texttt{__int64} return type and the \texttt{cmp8xchg} IA-64 instruction.

Examples of several of these intrinsics are provided at the end of this topic. For more information about the instructions these intrinsics generate, please see the documentation area of the Itanium® processor website at http://developer.intel.com/products/processor/itanium/index.htm.

\begin{itemize}
\item \textbf{Note}\texttt{Calling these intrinsics on any previous Itanium® processor causes an illegal instruction fault.}
\end{itemize}

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{Intrinsic Name} & \textbf{Operation} \\
\hline
\texttt{__cmp8xchg16} & Compare and exchange \\
\hline
\texttt{__ld16} & Load \\
\hline
\texttt{__fc_i} & Flush cache \\
\hline
\texttt{__hint} & Provide performance hints \\
\hline
\texttt{__st16} & Store \\
\hline
\texttt{__int64 __cmp8xchg16}(const int <sem>, const int <ldhint>, void *<addr>, \texttt{__int64 <xchg_lo>})
\end{tabular}
\end{table}
Generates the 16-byte form of the IA-64 compare and exchange instruction.

Returns the original 64-bit value read from memory at the specified address.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>sem</th>
<th>Idhint</th>
<th>addr</th>
<th>xchg_lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literal value between 0 and 1 that specifies the semaphore completer</td>
<td>Literal value between 0 and 2 that specifies the load hint completer</td>
<td>The address of the value to read.</td>
<td>The least significant 8 bytes of the exchange value.</td>
</tr>
<tr>
<td>(0==.none, (0==.acq, 1==.nt1, 1==.rel)</td>
<td>2==.nta)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following table describes each implicit argument for this intrinsic.

<table>
<thead>
<tr>
<th>xchg_hi</th>
<th>cmpnd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highest 8 bytes of the exchange value.</td>
<td>The 64-bit compare value. Use the _setReg intrinsic to set the &lt;xchg_hi&gt; value in the register AR[CSD].</td>
</tr>
<tr>
<td>Use the _setReg intrinsic to set the &lt;xchg_hi&gt; value in the register AR[CSD].</td>
<td>Use the _setReg intrinsic to set the &lt;cmpnd&gt; value in the register AR[CCV].</td>
</tr>
</tbody>
</table>

Example:

```c
__int64 foo_cmp8xchg16(__int64 xchg_lo, __int64 xchg_hi, __int64 cmpnd, void* addr)
{
__int64 old_value;
/**/
// set the highest bits of the exchange value and the comperand value respectively in CSD and CCV. Then, call the exchange intrinsic //
__setReg(_IA64_REG_AR_CSD, xchg_hi);
__setReg(_IA64_REG_AR_CCV, cmpnd);
```
old_value  = __cmp8xchg16(__semtype_acq, __ldhint_none, addr, xchg_lo);
    /**/
    return old_value;
}

__int64 __ld16(const int <ldtype>, const int <ldhint>, void *<addr>)

Generates the IA-64 instruction that loads 16 bytes from the given address. Returns the lower 8 bytes of the quantity loaded from <addr>. The higher 8 bytes are loaded in register AR[CSD]. Generates implicit return of the higher 8 bytes to the register AR[CSD]. You can use the __getReg intrinsic to copy the value into a user variable. [foo = __getReg(_IA64_REG_AR_CSD);]

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>ldtype</th>
<th>ldhint</th>
<th>addr</th>
</tr>
</thead>
<tbody>
<tr>
<td>A literal value between 0 and 1 that specifies the load type (0==none, 1==.acq).</td>
<td>A literal value between 0 and 2 that specifies the hint completer (0==none, 1==.nt1, 2==.nta).</td>
<td>The address to load from.</td>
</tr>
</tbody>
</table>

Example:

```c
void foo_ld16(__int64* lo, __int64* hi, void* addr)
{
    /**/

    // The following two calls load the 16-byte value at the given address into two (2) 64-bit integers
    // The higher 8 bytes are returned implicitly in the CSD register;
    // The call to __getReg moves that value into a user variable (hi).
    // The instruction generated is a plain ld16
    // ld16 Ra,ar.csd=[Rb]

    *lo = __ld16(__ldtype_none, __ldhint_none, addr);
    *hi = __getReg(_IA64_REG_AR_CSD);
    /**/
}
```
void __fc_i(void *<addr>)
Generates the IA-64 instruction that flushes the cache line associated with the
specified address and ensures coherency between instruction cache and data
cache.

The following table describes the argument for this intrinsic.

<table>
<thead>
<tr>
<th>cache_line</th>
</tr>
</thead>
<tbody>
<tr>
<td>An address associated with the cache line you want to flush</td>
</tr>
</tbody>
</table>

void __hint(const int <hint_value>)
Generates the IA-64 instruction that provides performance hints about the
program being executed.

The following table describes the argument for this intrinsic.

<table>
<thead>
<tr>
<th>hint_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A literal value that specifies the hint. Currently, zero is the only legal value. __hint(0) generates the IA-64 hint@pause instruction.</td>
</tr>
</tbody>
</table>

void __st16(const int <sttype>, const int <sthint>, void *<addr>, __int64 <src_lo>)
Generates the IA-64 instruction to store 16 bytes at the given address.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>sttype</th>
<th>sthint</th>
<th>addr</th>
<th>src_lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>A literal value between 0 and 1 that specifies the store type</td>
<td>A literal value between 0 and 1 that specifies the store hint completer</td>
<td>The address where the 16-byte value is stored.</td>
<td>The lowest 8 bytes of the 16-byte value to store.</td>
</tr>
</tbody>
</table>
The following table describes the implicit argument for this intrinsic.

<table>
<thead>
<tr>
<th>sttype</th>
<th>sthint</th>
<th>addr</th>
<th>src_lo</th>
</tr>
</thead>
<tbody>
<tr>
<td>1==.rel)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The highest 8 bytes of the 16-byte value to store. Use the setReg intrinsic to set the <src_hi> value in the register AR[CSD].

```c
__setReg(_IA64_REG_AR_CSD, <src_hi>); 
```

Example:

```c
void foo_st16(__int64 lo, __int64 hi, void* addr)
{
  /**/
  // first set the highest 64-bits into CSD register. Then call
  // __st16 with the lowest 64-bits as argument
  //
  __setReg(_IA64_REG_AR_CSD, hi);
  __st16(__sttype_none, __sthint_none, addr, lo);
  /**/
}
```

Example of Using Intrinsics Together

The following examples show how to use some of the intrinsics presented above together to generate the corresponding instructions. In all cases, use the __setReg (resp. __getReg) intrinsic to set up implicit arguments (resp. = retrieve implicit return values).

```c
// file foo.c
//
#include <ia64intrin.h>
void foo_ld16(__int64* lo, __int64* hi, void* addr)
{
  /**/
```
The following two calls load the 16-byte value at the given address:

- The higher 8 bytes are returned implicitly in the CSD register;
- The call to __getReg moves that value into a user variable (hi).

The instruction generated is a plain ld16:

```
    ld16 Ra,ar.csd=[Rb]
```

```c
*lo = __ld16(__ldtype_none, __ldhint_none, addr);
*hi = __getReg(_IA64_REG_AR_CSD);
/**/
```

```c
void foo_ld16_acq(__int64* lo, __int64* hi, void* addr)
{
    /**/

    // This is the same as the previous example, except that it uses the
    // __ldtype_acq completer to generate the acquire_from of the ld16:
    // ld16.acq Ra,ar.csd=[Rb]
    //
    *lo = __ld16(__ldtype_acq, __ldhint_none, addr);
    *hi = __getReg(_IA64_REG_AR_CSD);
    /**/
}
```

```c
void foo_st16(__int64 lo, __int64 hi, void* addr)
{
    /**/

    // first set the highest 64-bits into CSD register. Then call
    // __st16 with the lowest 64-bits as argument
    //
    __setReg(_IA64_REG_AR_CSD, hi);
    __st16(__sttype_none, __sthint_none, addr, lo);
    /**/
}
```

```c
__int64 foo_cmp8xchg16(__int64 xchg_lo, __int64 xchg_hi, __int64 cmpnd, void* addr)
{
    __int64 old_value;
    /**/

    // set the highest bits of the exchange value and the comperand value
    // respectively in CSD and CCV. Then, call the exchange intrinsic
//
  __setReg(_IA64_REG_AR_CSD, xchg_hi);
  __setReg(_IA64_REG_AR_CCV, cmpnd);
  old_value  = __cmp8xchg16(__semtype_acq, __ldhint_none, addr, xchg_lo);
  /***/
  return old_value;
}
// end foo.c

Microsoft-Compatible Intrinsics for Intel® Itanium® Dual-Core Processor 9000 Sequence

The Intel® Itanium® dual-core processor 9000 sequence supports the intrinsics listed in the table below. These intrinsics are also compatible with the Microsoft compiler. These intrinsics each generate IA-64 instructions. The second alphanumeric chain in the intrinsic name represents the IA-64 instruction the intrinsic generates. For example, the intrinsic _int64_cmp8xchg generates the cmp8xchg IA-64 instruction.

For more information about the instructions these intrinsics generate, please see the documentation area of the Itanium processor website.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding IA-64 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_InterlockedCompare64Exchange128</td>
<td>Compare and exchange</td>
<td></td>
</tr>
<tr>
<td>_InterlockedCompare64Exchange128_acq</td>
<td>Compare and Exchange</td>
<td></td>
</tr>
<tr>
<td>_InterlockedCompare64Exchange128_rel</td>
<td>Compare and Exchange</td>
<td></td>
</tr>
<tr>
<td>_load128</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>_load128_acq</td>
<td>Read</td>
<td></td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding IA-64 Instruction</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>__store128</td>
<td>Store</td>
<td></td>
</tr>
<tr>
<td>__store128_rel</td>
<td>Store</td>
<td></td>
</tr>
</tbody>
</table>

__int64 __InterlockedCompare64Exchange128(__int64 volatile *<Destination>, __int64 <ExchangeHigh>, __int64 <ExchangeLow>, __int64 <Comperand>)

Generates a compare and exchange IA-64 instruction. Returns the lowest 64-bit value of the destination.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>Destination</th>
<th>ExchangeHigh</th>
<th>ExchangeLow</th>
<th>Comperand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Destination</td>
<td>Highest 64 bits of the Exchange value</td>
<td>Lowest 64 bits of the Exchange value</td>
<td>Value to compare with Destination value</td>
</tr>
</tbody>
</table>

__int64 __InterlockedCompare64Exchange128_acq(__int64 volatile *<Destination>, __int64 <ExchangeHigh>, __int64 <ExchangeLow>, __int64 <Comperand>)

Generates a compare and exchange IA-64 instruction. Same as __InterlockedCompare64Exchange128, but this intrinsic uses acquire semantics. Returns the lowest 64-bit value of the destination.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>Destination</th>
<th>ExchangeHigh</th>
<th>ExchangeLow</th>
<th>Comperand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Destination</td>
<td>Highest 64 bits of the Exchange value</td>
<td>Lowest 64 bits of the Exchange value</td>
<td>Value to compare with Destination value</td>
</tr>
</tbody>
</table>
__int64 __InterlockedCompare64Exchange128_rel( __int64 volatile * <Destination>, __int64 <ExchangeHigh>, __int64 <ExchangeLow>, __int64 <Comperand>

Generates a compare and exchange IA-64 instruction. Same as __InterlockedCompare64Exchange128, but this intrinsic uses release semantics. Returns the lowest 64-bit value of the destination.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>Destination</th>
<th>ExchangeHigh</th>
<th>ExchangeLow</th>
<th>Comperand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Destination</td>
<td>Highest 64 bits of the Exchange value</td>
<td>Lowest 64 bits of the Exchange value</td>
<td>Value to compare with Destination value</td>
</tr>
</tbody>
</table>

__int64 __load128( __int64 volatile * Source, __int64 *<DestinationHigh>)

Generates the IA-64 instruction that atomically reads 128 bits from the memory location. Returns the lowest 64-bit value of the 128-bit loaded value.

The following table describes each argument for this intrinsic.

<table>
<thead>
<tr>
<th>Source</th>
<th>DestinationHigh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Source value</td>
<td>Pointer to the location in memory that stores the highest 64 bits of the 128-bit loaded value</td>
</tr>
</tbody>
</table>

__int64 __load128_acq( __int64 volatile * <Source>, __int64 *<DestinationHigh>)

Generates the IA-64 instruction that atomically reads 128 bits from the memory location. Same as __load128, but the this intrinsic uses acquire semantics. Returns the lowest 64-bit value of the 128-bit loaded value.

The following table describes each argument for this intrinsic.
Intrinsics for MMX(TM) Technology

Overview: MMX™ Technology Intrinsics

MMX™ technology is an extension to the Intel architecture (IA) instruction set. The MMX instruction set adds 57 opcodes and a 64-bit quadword data type, and eight 64-bit registers. Each of the eight registers can be directly addressed using the register names `mm0` to `mm7`. 

<table>
<thead>
<tr>
<th>Source</th>
<th>DestinationHigh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Source value</td>
<td>Pointer to the location in memory that stores the highest 64 bits of the 128-bit loaded value</td>
</tr>
</tbody>
</table>

```
__void __store128( __int64 volatile * <Destination>, __int64 <SourceHigh> __int64 <SourceLow>)
```

Generates the IA-64 instruction that atomically stores 128 bits at the destination memory location. No returns.

<table>
<thead>
<tr>
<th>Destination</th>
<th>SourceHigh</th>
<th>SourceLow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pointer to the 128-bit Destination value</td>
<td>The highest 64 bits of the value to be stored</td>
<td>The lowest 64 bits of the value to be stored</td>
</tr>
</tbody>
</table>

```
__void __store128_rel( __int64 volatile * <Destination>, __int64 <SourceHigh> __int64 <SourceLow>)
```

Generates the IA-64 instruction that atomically stores 128 bits at the destination memory location. Same as `__store128`, but this intrinsic uses release semantics. No returns.
The prototypes for MMX technology intrinsics are in the `mmintrin.h` header file.

### Details about MMX™ Technology Intrinsics

The MMX™ technology instructions use the following features:

- **Registers**—Enable packed data of up to 128 bits in length for optimal SIMD processing
- **Data Types**—Enable packing of up to 16 elements of data in one register

#### Registers

Intel processors provide special register sets. The MMX instructions use eight 64-bit registers (`mm0` to `mm7`) which are aliased on the floating-point stack registers. Because each of these registers can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as single-instruction multiple data processing (SIMD).

For each computational and data manipulation instruction in the new extension sets, there is a corresponding C intrinsic that implements that instruction directly. This frees you from managing registers and assembly programming. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

> **Note**

The MM and XMM registers are the SIMD registers used by the IA-32 architecture-based platforms to implement MMX™ technology and SSE or SSE2 intrinsics. On the IA-64 architecture, the MMX™ and SSE intrinsics use the 64-bit general registers and the 64-bit significand of the 80-bit floating-point register.

#### Data Types

Intrinsic functions use four new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions.
__m64 Data Type

The __m64 data type is used to represent the contents of an MMX register, which is the register that is used by the MMX technology intrinsics. The __m64 data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

Data Types Usage Guidelines

These data types are not basic ANSI C data types. You must observe the following usage restrictions:

- Use data types only on either side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (+, -, etc).
- Use data types as objects in aggregates, such as unions, to access the byte elements and structures.
- Use data types only with the respective intrinsics described in this documentation.

The EMMS Instruction: Why You Need It

Using EMMS is like emptying a container to accommodate new content. The EMMS instruction clears the MMX™ registers and sets the value of the floating-point tag word to empty.

You should clear the MMX registers before issuing a floating-point instruction because floating-point convention specifies that the floating-point stack be cleared after use. Insert the EMMS instruction at the end of all MMX code segments to avoid a floating-point overflow exception.

Why You Need EMMS to Reset After an MMX™ Instruction
Caution

Failure to empty the multimedia state after using an MMX instruction and before using a floating-point instruction can result in unexpected execution or poor performance.

**EMMS Usage Guidelines**

Here are guidelines for when to use the EMMS instruction:

- Use `_mm_empty()` after an MMX™ instruction if the next instruction is a floating-point (FP) instruction. For example, you should use the EMMS instruction before performing calculations on `float`, `double` or `long double`. You must be aware of all situations in which your code generates an MMX instruction:
  - when using an MMX technology intrinsic
  - when using Streaming SIMD Extension integer intrinsics that use the `_m64` data type
  - when referencing an `_m64` data type variable
  - when using an MMX instruction through inline assembly
• Use different functions for operations that use floating point instructions and those that use MMX instructions. This action eliminates the need to empty the multimedia state within the body of a critical loop.

• Use \_mm\_empty() during runtime initialization of \_m64 and FP data types. This ensures resetting the register between data type transitions.

• Do not use \_mm\_empty() before an MMX instruction, since using \_mm\_empty() before an MMX instruction incurs an operation with no benefit (no-op).

• Do not use on systems based on IA-64 architecture. There are no special registers (or overlay) for the MMX™ instructions or Streaming SIMD Extensions on systems based on IA-64 architecture even though the intrinsics are supported.

• See the Correct Usage and Incorrect Usage coding examples in the following table.

<table>
<thead>
<tr>
<th>Incorrect Usage</th>
<th>Correct Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>_m64 x = _m_padd(y, z);</td>
<td>_m64 x = _m_padd(y, z);</td>
</tr>
<tr>
<td>float f = init();</td>
<td>float f = (_mm_empty(), init());</td>
</tr>
</tbody>
</table>

**MMX™ Technology General Support Intrinsics**

The prototypes for MMX™ technology general support intrinsics are in the *mmintrin.h* header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_empty</td>
<td>Empty MM state</td>
<td>EMMS</td>
</tr>
<tr>
<td>_mm_cvtsi32_si64</td>
<td>Convert from int</td>
<td>MOVD</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding MMX Instruction</td>
</tr>
<tr>
<td>---------------</td>
<td>--------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_cvtsi64_si32</td>
<td>Convert to int</td>
<td>MOVD</td>
</tr>
<tr>
<td>_mm_cvtsi64_m64</td>
<td>Convert from __int64</td>
<td>MOVQ</td>
</tr>
<tr>
<td>_mm_cvtm64_si64</td>
<td>Convert to __int64</td>
<td>MOVQ</td>
</tr>
<tr>
<td>_mm_packs_pi16</td>
<td>Pack</td>
<td>PACKSSWB</td>
</tr>
<tr>
<td>_mm_packs_pi32</td>
<td>Pack</td>
<td>PACKSSDW</td>
</tr>
<tr>
<td>_mm_packs_pu16</td>
<td>Pack</td>
<td>PACKUSWB</td>
</tr>
<tr>
<td>_mm_unpackhi_pi8</td>
<td>Interleave</td>
<td>PUNPCKHBW</td>
</tr>
<tr>
<td>_mm_unpackhi_pi16</td>
<td>Interleave</td>
<td>PUNPCKHWD</td>
</tr>
<tr>
<td>_mm_unpackhi_pi32</td>
<td>Interleave</td>
<td>PUNPCKHDQ</td>
</tr>
<tr>
<td>_mm_unpacklo_pi8</td>
<td>Interleave</td>
<td>PUNPCKLBW</td>
</tr>
<tr>
<td>_mm_unpacklo_pi16</td>
<td>Interleave</td>
<td>PUNPCKLWD</td>
</tr>
<tr>
<td>_mm_unpacklo_pi32</td>
<td>Interleave</td>
<td>PUNPCKLDQ</td>
</tr>
<tr>
<td>void _mm_empty(void)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Empties the multimedia state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m64 _mm_cvtsi32_si64(int i)</td>
<td>Convert the integer object i to a 64-bit __m64 object. The integer value is zero-extended to 64 bits.</td>
<td></td>
</tr>
<tr>
<td>int _mm_cvtsi64_si32(__m64 m)</td>
<td>Converts the lower 32 bits of the __m64 object m to an integer.</td>
<td></td>
</tr>
<tr>
<td>__m64 _mm_cvtsi64_m64(__int64 i)</td>
<td>Moves the 64-bit integer object i to a __mm64 object</td>
<td></td>
</tr>
<tr>
<td>__m64 _mm_cvtm64_si64(__m64 m)</td>
<td>Moves the __m64 object m to a 64-bit integer</td>
<td></td>
</tr>
</tbody>
</table>
_m64 _mm_packs_pi16(__m64 m1, __m64 m2)

Packs the four 16-bit values from m1 into the lower four 8-bit values of the result with signed saturation, and pack the four 16-bit values from m2 into the upper four 8-bit values of the result with signed saturation.

__m64 _mm_packs_pi32(__m64 m1, __m64 m2)

Packs the two 32-bit values from m1 into the lower two 16-bit values of the result with signed saturation, and pack the two 32-bit values from m2 into the upper two 16-bit values of the result with signed saturation.

__m64 _mm_packs_pu16(__m64 m1, __m64 m2)

Packs the four 16-bit values from m1 into the lower four 8-bit values of the result with unsigned saturation, and pack the four 16-bit values from m2 into the upper four 8-bit values of the result with unsigned saturation.

__m64 _mm_unpackhi_pi8(__m64 m1, __m64 m2)

Interleaves the four 8-bit values from the high half of m1 with the four values from the high half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpackhi_pi16(__m64 m1, __m64 m2)

Interleaves the two 16-bit values from the high half of m1 with the two values from the high half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpackhi_pi32(__m64 m1, __m64 m2)

Interleaves the 32-bit value from the high half of m1 with the 32-bit value from the high half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpacklo_pi8(__m64 m1, __m64 m2)

Interleaves the four 8-bit values from the low half of m1 with the four values from the low half of m2. The interleaving begins with the data from m1.

__m64 _mm_unpacklo_pi16(__m64 m1, __m64 m2)

Interleaves the two 16-bit values from the low half of m1 with the two values from the low half of m2. The interleaving begins with the data from m1.
```c
__m64 _mm_unpacklo_pi32(__m64 m1, __m64 m2)
```

Interleaves the 32-bit value from the low half of `m1` with the 32-bit value from the low half of `m2`. The interleaving begins with the data from `m1`.

### MMX™ Technology Packed Arithmetic Intrinsics

The prototypes for MMX™ technology packed arithmetic intrinsics are in the `mmintrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_pi8</td>
<td>Addition</td>
<td>PADDB</td>
</tr>
<tr>
<td>_mm_add_pi16</td>
<td>Addition</td>
<td>PADDW</td>
</tr>
<tr>
<td>_mm_add_pi32</td>
<td>Addition</td>
<td>PADDD</td>
</tr>
<tr>
<td>_mm_adds_pi8</td>
<td>Addition</td>
<td>PADDSB</td>
</tr>
<tr>
<td>_mm_adds_pi16</td>
<td>Addition</td>
<td>PADDSW</td>
</tr>
<tr>
<td>_mm_adds_pu8</td>
<td>Addition</td>
<td>PADDUSB</td>
</tr>
<tr>
<td>_mm_adds_pu16</td>
<td>Addition</td>
<td>PADDUSW</td>
</tr>
<tr>
<td>_mm_sub_pi8</td>
<td>Subtraction</td>
<td>PSUBB</td>
</tr>
<tr>
<td>_mm_sub_pi16</td>
<td>Subtraction</td>
<td>PSUBW</td>
</tr>
<tr>
<td>_mm_sub_pi32</td>
<td>Subtraction</td>
<td>PSUBD</td>
</tr>
<tr>
<td>_mm_subs_pi8</td>
<td>Subtraction</td>
<td>SUBSB</td>
</tr>
<tr>
<td>_mm_subs_pi16</td>
<td>Subtraction</td>
<td>SUBSW</td>
</tr>
<tr>
<td>_mm_subs_pu8</td>
<td>Subtraction</td>
<td>SUBUSB</td>
</tr>
<tr>
<td>_mm_subs_pu16</td>
<td>Subtraction</td>
<td>SUBUSW</td>
</tr>
<tr>
<td>_mm_madd_pi16</td>
<td>Multiply and add</td>
<td>PMADDWD</td>
</tr>
<tr>
<td>_mm_mulhi_pi16</td>
<td>Multiplication</td>
<td>PMULHW</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding MMX Instruction</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>_mm_mullo_pi16</td>
<td>Multiplication</td>
<td>PMULLW</td>
</tr>
</tbody>
</table>

__m64  _mm_add_pi8(__m64  m1, __m64  m2)

Add the eight 8-bit values in m1 to the eight 8-bit values in m2.

__m64  _mm_add_pi16(__m64  m1, __m64  m2)

Add the four 16-bit values in m1 to the four 16-bit values in m2.

__m64  _mm_add_pi32(__m64  m1, __m64  m2)

Add the two 32-bit values in m1 to the two 32-bit values in m2.

__m64  _mm_adds_pi8(__m64  m1, __m64  m2)

Add the eight signed 8-bit values in m1 to the eight signed 8-bit values in m2 using saturating arithmetic.

__m64  _mm_adds_pi16(__m64  m1, __m64  m2)

Add the four signed 16-bit values in m1 to the four signed 16-bit values in m2 using saturating arithmetic.

__m64  _mm_adds_pu8(__m64  m1, __m64  m2)

Add the eight unsigned 8-bit values in m1 to the eight unsigned 8-bit values in m2 and using saturating arithmetic.

__m64  _mm_adds_pu16(__m64  m1, __m64  m2)

Add the four unsigned 16-bit values in m1 to the four unsigned 16-bit values in m2 using saturating arithmetic.

__m64  _mm_sub_pi8(__m64  m1, __m64  m2)

Subtract the eight 8-bit values in m2 from the eight 8-bit values in m1.

__m64  _mm_sub_pi16(__m64  m1, __m64  m2)

Subtract the four 16-bit values in m2 from the four 16-bit values in m1.

__m64  _mm_sub_pi32(__m64  m1, __m64  m2)
Subtract the two 32-bit values in $m_2$ from the two 32-bit values in $m_1$.

__m64 _mm_subs_pi8(__m64 m1, __m64 m2)

Subtract the eight signed 8-bit values in $m_2$ from the eight signed 8-bit values in $m_1$ using saturating arithmetic.

__m64 _mm_subs_pi16(__m64 m1, __m64 m2)

Subtract the four signed 16-bit values in $m_2$ from the four signed 16-bit values in $m_1$ using saturating arithmetic.

__m64 _mm_subs_pu8(__m64 m1, __m64 m2)

Subtract the eight unsigned 8-bit values in $m_2$ from the eight unsigned 8-bit values in $m_1$ using saturating arithmetic.

__m64 _mm_subs_pu16(__m64 m1, __m64 m2)

Subtract the four unsigned 16-bit values in $m_2$ from the four unsigned 16-bit values in $m_1$ using saturating arithmetic.

__m64 _mm_madd_pi16(__m64 m1, __m64 m2)

Multiply four 16-bit values in $m_1$ by four 16-bit values in $m_2$ producing four 32-bit intermediate results, which are then summed by pairs to produce two 32-bit results.

__m64 _mm_mulhi_pi16(__m64 m1, __m64 m2)

Multiply four signed 16-bit values in $m_1$ by four signed 16-bit values in $m_2$ and produce the high 16 bits of the four results.

__m64 _mm_mullo_pi16(__m64 m1, __m64 m2)

Multiply four 16-bit values in $m_1$ by four 16-bit values in $m_2$ and produce the low 16 bits of the four results.

**MMX™ Technology Shift Intrinsics**

The prototypes for MMX™ technology shift intrinsics are in the `mmintrin.h` header file.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_sll_pi16</td>
<td>Logical shift left</td>
<td>PSLLW</td>
</tr>
<tr>
<td>_mm_slli_pi16</td>
<td>Logical shift left</td>
<td>PSLLWI</td>
</tr>
<tr>
<td>_mm_sll_pi32</td>
<td>Logical shift left</td>
<td>PSLLD</td>
</tr>
<tr>
<td>_mm_slli_pi32</td>
<td>Logical shift left</td>
<td>PSLLDI</td>
</tr>
<tr>
<td>_mm_sll_pi64</td>
<td>Logical shift left</td>
<td>PSLLQ</td>
</tr>
<tr>
<td>_mm_slli_pi64</td>
<td>Logical shift left</td>
<td>PSLLQI</td>
</tr>
<tr>
<td>_mm_sra_pi16</td>
<td>Arithmetic shift right</td>
<td>PSRAW</td>
</tr>
<tr>
<td>_mm_srai_pi16</td>
<td>Arithmetic shift right</td>
<td>PSRAWI</td>
</tr>
<tr>
<td>_mm_sra_pi32</td>
<td>Arithmetic shift right</td>
<td>PSRAD</td>
</tr>
<tr>
<td>_mm_srai_pi32</td>
<td>Arithmetic shift right</td>
<td>PSRADI</td>
</tr>
<tr>
<td>_mm_srl_pi16</td>
<td>Logical shift right</td>
<td>PSRLW</td>
</tr>
<tr>
<td>_mm_srli_pi16</td>
<td>Logical shift right</td>
<td>PSRLWI</td>
</tr>
<tr>
<td>_mm_srl_pi32</td>
<td>Logical shift right</td>
<td>PSRLD</td>
</tr>
<tr>
<td>_mm_srli_pi32</td>
<td>Logical shift right</td>
<td>PSRLDI</td>
</tr>
<tr>
<td>_mm_srl_pi64</td>
<td>Logical shift right</td>
<td>PSRLQ</td>
</tr>
<tr>
<td>_mm_srli_pi64</td>
<td>Logical shift right</td>
<td>PSRLQI</td>
</tr>
</tbody>
</table>

__m64 _mm_sll_pi16(__m64 m, __m64 count)
Shifts four 16-bit values in m left the amount specified by count while shifting in zeros.

__m64 _mm_slli_pi16(__m64 m, int count)
Shifts four 16-bit values in m left the amount specified by count while shifting in zeros. For the best performance, count should be a constant.

__m64 _mm_sll_pi32(__m64 m, __m64 count)
Shifts two 32-bit values in `m` left the amount specified by `count` while shifting in zeros.

```c
__m64 __m_slli_pi32(__m64 m, int count)
```

Shifts two 32-bit values in `m` left the amount specified by `count` while shifting in zeros. For the best performance, `count` should be a constant.

```c
__m64 __m_sll_pi64(__m64 m, __m64 count)
```

Shifts the 64-bit value in `m` left the amount specified by `count` while shifting in zeros.

```c
__m64 __m_slli_pi64(__m64 m, int count)
```

Shifts the 64-bit value in `m` left the amount specified by `count` while shifting in zeros. For the best performance, `count` should be a constant.

```c
__m64 __m_sra_pi16(__m64 m, __m64 count)
```

Shifts four 16-bit values in `m` right the amount specified by `count` while shifting in the sign bit.

```c
__m64 __m_srai_pi16(__m64 m, int count)
```

Shifts four 16-bit values in `m` right the amount specified by `count` while shifting in the sign bit. For the best performance, `count` should be a constant.

```c
__m64 __m_sra_pi32(__m64 m, __m64 count)
```

Shifts two 32-bit values in `m` right the amount specified by `count` while shifting in the sign bit.

```c
__m64 __m_srai_pi32(__m64 m, int count)
```

Shifts two 32-bit values in `m` right the amount specified by `count` while shifting in the sign bit. For the best performance, `count` should be a constant.

```c
__m64 __m_srl_pi16(__m64 m, __m64 count)
```

Shifts four 16-bit values in `m` right the amount specified by `count` while shifting in zeros.

```c
__m64 __m_srl_i16(__m64 m, int count)
```
Shifts four 16-bit values in m right the amount specified by count while shifting in zeros. For the best performance, count should be a constant.

```c
__m64 __mm_srl_pi32(__m64 m, __m64 count)
```

Shifts two 32-bit values in m right the amount specified by count while shifting in zeros.

```c
__m64 __mm_srl_pi32(__m64 m, int count)
```

Shifts two 32-bit values in m right the amount specified by count while shifting in zeros. For the best performance, count should be a constant.

```c
__m64 __mm_srl_pi64(__m64 m, __m64 count)
```

Shifts the 64-bit value in m right the amount specified by count while shifting in zeros.

```c
__m64 __mm_srl_pi64(__m64 m, int count)
```

Shifts the 64-bit value in m right the amount specified by count while shifting in zeros. For the best performance, count should be a constant.

**MMX™ Technology Logical Intrinsics**

The prototypes for MMX™ technology logical intrinsics are in the `mmintrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_si64</td>
<td>Bitwise AND</td>
<td>PAND</td>
</tr>
<tr>
<td>_mm_andnot_si64</td>
<td>Bitwise ANDNOT</td>
<td>PANDN</td>
</tr>
<tr>
<td>_mm_or_si64</td>
<td>Bitwise OR</td>
<td>POR</td>
</tr>
<tr>
<td>_mm_xor_si64</td>
<td>Bitwise Exclusive OR</td>
<td>PXOR</td>
</tr>
<tr>
<td>__m64 __mm_and_si64(__m64 m1, __m64 m2)</td>
<td>Perform a bitwise AND of the 64-bit value in m1 with the 64-bit value in m2.</td>
<td></td>
</tr>
</tbody>
</table>
**__m64 __mm_andnot_si64(__m64 m1, __m64 m2)**

Perform a bitwise NOT on the 64-bit value in m1 and use the result in a bitwise AND with the 64-bit value in m2.

**__m64 __mm_or_si64(__m64 m1, __m64 m2)**

Perform a bitwise OR of the 64-bit value in m1 with the 64-bit value in m2.

**__m64 __mm_xor_si64(__m64 m1, __m64 m2)**

Perform a bitwise XOR of the 64-bit value in m1 with the 64-bit value in m2.

**MMX™ Technology Compare Intrinsics**

The prototypes for MMX™ technology compare intrinsics are in the *mmintrin.h* header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpeq_pi8</td>
<td>Equal</td>
<td>PCMPEQB</td>
</tr>
<tr>
<td>_mm_cmpeq_pi16</td>
<td>Equal</td>
<td>PCMPEQW</td>
</tr>
<tr>
<td>_mm_cmpeq_pi32</td>
<td>Equal</td>
<td>PCMPEQD</td>
</tr>
<tr>
<td>_mm_cmpgt_pi8</td>
<td>Greater Than</td>
<td>PCMPGTB</td>
</tr>
<tr>
<td>_mm_cmpgt_pi16</td>
<td>Greater Than</td>
<td>PCMPGTW</td>
</tr>
<tr>
<td>_mm_cmpgt_pi32</td>
<td>Greater Than</td>
<td>PCMPGTD</td>
</tr>
<tr>
<td>__m64 __mm_cmpeq_pi8(__m64 m1, __m64 m2)</td>
<td>Sets the corresponding 8-bit resulting values to all ones if the 8-bit values in m1 are equal to the corresponding 8-bit values in m2; otherwise sets them to all zeros.</td>
<td></td>
</tr>
<tr>
<td>__m64 __mm_cmpeq_pi16(__m64 m1, __m64 m2)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sets the corresponding 16-bit resulting values to all ones if the 16-bit values in \texttt{m1} are equal to the corresponding 16-bit values in \texttt{m2}; otherwise set them to all zeros.

\begin{verbatim}
__m64 _mm_cmpeq_pi32(__m64 m1, __m64 m2)
\end{verbatim}

Sets the corresponding 32-bit resulting values to all ones if the 32-bit values in \texttt{m1} are equal to the corresponding 32-bit values in \texttt{m2}; otherwise set them to all zeros.

\begin{verbatim}
__m64 _mm_cmpgt_pi8(__m64 m1, __m64 m2)
\end{verbatim}

Sets the corresponding 8-bit resulting values to all ones if the 8-bit signed values in \texttt{m1} are greater than the corresponding 8-bit signed values in \texttt{m2}; otherwise set them to all zeros.

\begin{verbatim}
__m64 _mm_cmpgt_pi16(__m64 m1, __m64 m2)
\end{verbatim}

Sets the corresponding 16-bit resulting values to all ones if the 16-bit signed values in \texttt{m1} are greater than the corresponding 16-bit signed values in \texttt{m2}; otherwise set them to all zeros.

\begin{verbatim}
__m64 _mm_cmpgt_pi32(__m64 m1, __m64 m2)
\end{verbatim}

Sets the corresponding 32-bit resulting values to all ones, if the 32-bit signed values in \texttt{m1} are greater than the corresponding 32-bit signed values in \texttt{m2}; otherwise set them all to zeros.

\textbf{MMX™ Technology Set Intrinsics}

The prototypes for MMX™ technology intrinsics are in the \texttt{mmintrin.h} header file.

\begin{quote}
\textbf{Note}

In the descriptions regarding the bits of the MMX register, bit 0 is the least significant and bit 63 is the most significant.
\end{quote}
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding MMX Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_setzero_si64</td>
<td>set to zero</td>
<td>PXOR</td>
</tr>
<tr>
<td>_mm_set_pi32</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set_pi16</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set_pi8</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set1_pi32</td>
<td>set integer values</td>
<td></td>
</tr>
<tr>
<td>_mm_set1_pi16</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set1_pi8</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_pi32</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_pi16</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_pi8</td>
<td>set integer values</td>
<td>Composite</td>
</tr>
</tbody>
</table>

___m64 __mm_setzero_si64()

Sets the 64-bit value to zero.

R

0x0

___m64 __mm_set_pi32(int i1, int i0)

Sets the 2 signed 32-bit integer values.

R0 R1
i0 i1

___m64 __mm_set_pi16(short s3, short s2, short s1, short s0)

Sets the 4 signed 16-bit integer values.

R0 R1 R2 R3
__m64 _mm_set_pi8(char b7, char b6, char b5, char b4, char b3, char b2, char b1, char b0)

Sets the 8 signed 8-bit integer values.

__m64 _mm_set1_pi32(int i)

Sets the 2 signed 32-bit integer values to i.

__m64 _mm_set1_pi16(short s)

Sets the 4 signed 16-bit integer values to w.

__m64 _mm_set1_pi8(char b)

Sets the 8 signed 8-bit integer values to b.

__m64 _mm_setr_pi32(int i1, int i0)

Sets the 2 signed 32-bit integer values in reverse order.
__m64 _mm_setr_pi16(short s3, short s2, short s1, short s0)

Sets the 4 signed 16-bit integer values in reverse order.

__m64 _mm_setr_pi8(char b7, char b6, char b5, char b4, char b3, char b2, char b1, char b0)

Sets the 8 signed 8-bit integer values in reverse order.

### MMX™ Technology Intrinsics for IA-64 Architecture

MMX™ technology intrinsics provide access to the MMX technology instruction set on systems based on IA-64 architecture. To provide source compatibility with the IA-32 architecture, these intrinsics are equivalent both in name and functionality to the set of IA-32 architecture-based MMX intrinsics.

The prototypes for MMX technology intrinsics are in the `mmintrin.h` header file.

### Data Types

The C data type `__m64` is used when using MMX technology intrinsics. It can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value. The `__m64` data type is not a basic ANSI C data type. Therefore, observe the following usage restrictions:
• Use the new data type only on the left-hand side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (" + ", " - ", and so on).
• Use the new data type as objects in aggregates, such as unions, to access the byte elements and structures; the address of an __m64 object may be taken.
• Use new data types only with the respective intrinsics described in this documentation.


Intrinsics for Streaming SIMD Extensions

Overview: Streaming SIMD Extensions

This section describes the C++ language-level features supporting the Intel® Streaming SIMD Extensions (SSE) in the Intel® C++ Compiler. These topics explain the following features of the intrinsics:

• Floating Point Intrinsics
• Arithmetic Operation Intrinsics
• Logical Operation Intrinsics
• Comparison Intrinsics
• Conversion Intrinsics
• Load Operations
• Set Operations
• Store Operations
• Cacheability Support
• Integer Intrinsics
• Intrinsics to Read and Write Registers
• Miscellaneous Intrinsics
• Using Streaming SIMD Extensions on IA-64 Architecture

The prototypes for SSE intrinsics are in the xmmintrin.h header file.
Note

You can also use the single `ia32intrin.h` header file for any IA-32 architecture-based intrinsics.

Details about Streaming SIMD Extension (SSE) Intrinsics

The Streaming SIMD Extension (SSE) instructions use the following features:

- Registers--Enable packed data of up to 128 bits in length for optimal SIMD processing
- Data Types--Enable packing of up to 16 elements of data in one register

Registers

Intel processors provide special register sets. The Streaming SIMD Extensions use eight 128-bit registers (`xmm0` to `xmm7`). Because each of these registers can hold more than one data element, the processor can process more than one data element simultaneously. This processing capability is also known as single-instruction multiple data processing (SIMD).

For each computational and data manipulation instruction in the new extension sets, there is a corresponding C intrinsic that implements that instruction directly. This frees you from managing registers and assembly programming. Further, the compiler optimizes the instruction scheduling so that your executable runs faster.

Note

The `MM` and `XMM` registers are the SIMD registers used by the IA-32 architecture-based platforms to implement MMX™ technology and SSE or SSE2 intrinsics. On the IA-64 architecture, the MMX™ and SSE intrinsics use the 64-bit general registers and the 64-bit significand of the 80-bit floating-point register.

Data Types
Intrinsic functions use four new C data types as operands, representing the new registers that are used as the operands to these intrinsic functions.

**New Data Types**

The following table details for which instructions each of the new data types are available.

<table>
<thead>
<tr>
<th>New Data Type</th>
<th>Streaming SIMD Extensions Intrinsic</th>
<th>Streaming SIMD Extensions 2 Intrinsic</th>
<th>Streaming SIMD Extensions 3 Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m64</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>__m128</td>
<td>Available</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>__m128d</td>
<td>Not available</td>
<td>Available</td>
<td>Available</td>
</tr>
<tr>
<td>__m128i</td>
<td>Not available</td>
<td>Available</td>
<td>Available</td>
</tr>
</tbody>
</table>

**__m64 Data Type**

On the IA-64 architecture, the __m64 data type is used to represent the contents of a 64-bit general registers. The __m64 data type can hold eight 8-bit values, four 16-bit values, two 32-bit values, or one 64-bit value.

**__m128 Data Types**

The __m128 data type is used to represent the contents of a Streaming SIMD Extension register used by the Streaming SIMD Extension intrinsics. The __m128 data type can hold four 32-bit floating-point values.

The __m128d data type can hold two 64-bit floating-point values.

The __m128i data type can hold sixteen 8-bit, eight 16-bit, four 32-bit, or two 64-bit integer values.

The compiler aligns __m128d and __m128i local and global data to 16-byte boundaries on the stack. To align integer, float, or double arrays, you can use thedeclspec align statement.
Data Types Usage Guidelines

These data types are not basic ANSI C data types. You must observe the following usage restrictions:

- Use data types only on either side of an assignment, as a return value, or as a parameter. You cannot use it with other arithmetic expressions (+, -, etc).
- Use data types as objects in aggregates, such as unions, to access the byte elements and structures.
- Use data types only with the respective intrinsics described in this documentation.

Accessing __m128i Data

To access 8-bit data:

```
#define _mm_extract_epi8(x, imm)  
  (((imm) & 0x1) == 0) ?  
    _mm_extract_epi16((x), (imm) >> 1) & 0xff :  
    _mm_extract_epi16(_mm_srli_epi16((x), 8), (imm) >> 1))
```

For 16-bit data, use the following intrinsic:

```
int _mm_extract_epi16(__m128i a, int imm)
```

To access 32-bit data:

```
#define _mm_extract_epi32(x, imm)  
  _mm_cvtsi128_si32(_mm_srli_si128((x), 4 * (imm)))
```

To access 64-bit data (Intel® 64 architecture only):

```
#define _mm_extract_epi64(x, imm)  
  _mm_cvtsi128_si64(_mm_srli_si128((x), 8 * (imm)))
```

Writing Programs with Streaming SIMD Extensions Intrinsics

You should be familiar with the hardware features provided by the Streaming SIMD Extensions (SSE) when writing programs with the intrinsics. The following are four important issues to keep in mind:

- Certain intrinsics, such as _mm_loadr_ps and _mm_cmpgt_ss, are not directly supported by the instruction set. While these intrinsics are convenient programming aids, be mindful that they may consist of more than one machine-language instruction.
- Floating-point data loaded or stored as __m128 objects must be generally 16-byte-aligned.
• Some intrinsics require that their argument be immediates, that is, constant integers (literals), due to the nature of the instruction.
• The result of arithmetic operations acting on two NaN (Not a Number) arguments is undefined. Therefore, FP operations using NaN arguments will not match the expected behavior of the corresponding assembly instructions.

Arithmetic Operations for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for arithmetic operations are in the \textit{xmmintrin.h} header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R0-R3. R0, R1, R2 and R3 each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_ss</td>
<td>Addition</td>
<td>ADDSS</td>
</tr>
<tr>
<td>_mm_add_ps</td>
<td>Addition</td>
<td>ADDPS</td>
</tr>
<tr>
<td>_mm_sub_ss</td>
<td>Subtraction</td>
<td>SUBSS</td>
</tr>
<tr>
<td>_mm_sub_ps</td>
<td>Subtraction</td>
<td>SUBPS</td>
</tr>
<tr>
<td>_mm_mul_ss</td>
<td>Multiplication</td>
<td>MULSS</td>
</tr>
<tr>
<td>_mm_mul_ps</td>
<td>Multiplication</td>
<td>MULPS</td>
</tr>
<tr>
<td>_mm_div_ss</td>
<td>Division</td>
<td>DIVSS</td>
</tr>
<tr>
<td>_mm_div_ps</td>
<td>Division</td>
<td>DIVPS</td>
</tr>
<tr>
<td>_mm_sqrt_ss</td>
<td>Squared Root</td>
<td>SQRTSS</td>
</tr>
<tr>
<td>_mm_sqrt_ps</td>
<td>Squared Root</td>
<td>SQRTPS</td>
</tr>
<tr>
<td>_mm_rcp_ss</td>
<td>Reciprocal</td>
<td>RCPSS</td>
</tr>
<tr>
<td>_mm_rcp_ps</td>
<td>Reciprocal</td>
<td>RCPPS</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding SSE Instruction</td>
</tr>
<tr>
<td>--------------------</td>
<td>--------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_rsqrt_ss</td>
<td>Reciprocal Squared Root</td>
<td>RSQRTSS</td>
</tr>
<tr>
<td>_mm_rsqrt_ps</td>
<td>Reciprocal Squared Root</td>
<td>RSQRTPS</td>
</tr>
<tr>
<td>_mm_min_ss</td>
<td>Computes Minimum</td>
<td>MINSS</td>
</tr>
<tr>
<td>_mm_min_ps</td>
<td>Computes Minimum</td>
<td>MINPS</td>
</tr>
<tr>
<td>_mm_max_ss</td>
<td>Computes Maximum</td>
<td>MAXSS</td>
</tr>
<tr>
<td>_mm_max_ps</td>
<td>Computes Maximum</td>
<td>MAXPS</td>
</tr>
</tbody>
</table>

__m128 _mm_add_ss(__m128 a, __m128 b)

Adds the lower single-precision, floating-point (SP FP) values of \(a\) and \(b\); the upper 3 SP FP values are passed through from \(a\).

```
R0  R1  R2  R3
a0 + b0 a1  a2  a3
```

__m128 _mm_add_ps(__m128 a, __m128 b)

Adds the four SP FP values of \(a\) and \(b\).

```
R0  R1  R2  R3
a0 +b0 a1 + b1 a2 + b2 a3 + b3
```

__m128 _mm_sub_ss(__m128 a, __m128 b)

Subtracts the lower SP FP values of \(a\) and \(b\). The upper 3 SP FP values are passed through from \(a\).

```
R0  R1  R2  R3
a0 - b0 a1  a2  a3
```

__m128 _mm_sub_ps(__m128 a, __m128 b)
Subtracts the four SP FP values of \(a\) and \(b\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\hline
a0 - b0 & a1 - b1 & a2 - b2 & a3 - b3 \\
\end{array}
\]

\[
\_\_m128 \_\_m\_mul\_ss(\_m128 a, \_m128 b)
\]

Multiplies the lower SP FP values of \(a\) and \(b\); the upper 3 SP FP values are passed through from \(a\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\hline
a0 \times b0 & a1 & a2 & a3 \\
\end{array}
\]

\[
\_\_m128 \_\_m\_mul\_ps(\_m128 a, \_m128 b)
\]

Multiplies the four SP FP values of \(a\) and \(b\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\hline
a0 \times b0 & a1 \times b1 & a2 \times b2 & a3 \times b3 \\
\end{array}
\]

\[
\_\_m128 \_\_m\_div\_ss(\_m128 a, \_m128 b)
\]

Divides the lower SP FP values of \(a\) and \(b\); the upper 3 SP FP values are passed through from \(a\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\hline
a0 / b0 & a1 & a2 & a3 \\
\end{array}
\]

\[
\_\_m128 \_\_m\_div\_ps(\_m128 a, \_m128 b)
\]

Divides the four SP FP values of \(a\) and \(b\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\hline
a0 / b0 & a1 / b1 & a2 / b2 & a3 / b3 \\
\end{array}
\]

\[
\_\_m128 \_\_m\_sqrt\_ss(\_m128 a)
\]
Computes the square root of the lower SP FP value of \(a\); the upper 3 SP FP values are passed through.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{sqrt}(a0) & a1 & a2 & a3 \\
\end{array}
\]

\_m128 \_mm_sqrt_ps(\_m128 a)

Computes the square roots of the four SP FP values of \(a\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{sqrt}(a0) & \text{sqrt}(a1) & \text{sqrt}(a2) & \text{sqrt}(a3) \\
\end{array}
\]

\_m128 \_mm_rcp_ss(\_m128 a)

Computes the approximation of the reciprocal of the lower SP FP value of \(a\); the upper 3 SP FP values are passed through.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{recip}(a0) & a1 & a2 & a3 \\
\end{array}
\]

\_m128 \_mm_rcp_ps(\_m128 a)

Computes the approximations of reciprocals of the four SP FP values of \(a\).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{recip}(a0) & \text{recip}(a1) & \text{recip}(a2) & \text{recip}(a3) \\
\end{array}
\]

\_m128 \_mm_rsqrt_ss(\_m128 a)

Computes the approximation of the reciprocal of the square root of the lower SP FP value of \(a\); the upper 3 SP FP values are passed through.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{recip}(\text{sqrt}(a0)) & a1 & a2 & a3 \\
\end{array}
\]

\_m128 \_mm_rsqrt_ps(\_m128 a)
Computes the approximations of the reciprocals of the square roots of the four SP FP values of \( a \).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{recip(sqrt(a0))} & \text{recip(sqrt(a1))} & \text{recip(sqrt(a2))} & \text{recip(sqrt(a3))}
\end{array}
\]

\[
\_\_m128 \_\_mm\_min\_ss(\_\_m128 a, \_\_m128 b)
\]

Computes the minimum of the lower SP FP values of \( a \) and \( b \); the upper 3 SP FP values are passed through from \( a \).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{min(a0, b0)} & a1 & a2 & a3
\end{array}
\]

\[
\_\_m128 \_\_mm\_min\_ps(\_\_m128 a, \_\_m128 b)
\]

Computes the minimum of the four SP FP values of \( a \) and \( b \).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{min(a0, b0)} & \text{min(a1, b1)} & \text{min(a2, b2)} & \text{min(a3, b3)}
\end{array}
\]

\[
\_\_m128 \_\_mm\_max\_ss(\_\_m128 a, \_\_m128 b)
\]

Computes the maximum of the lower SP FP values of \( a \) and \( b \); the upper 3 SP FP values are passed through from \( a \).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{max(a0, b0)} & a1 & a2 & a3
\end{array}
\]

\[
\_\_m128 \_\_mm\_max\_ps(\_\_m128 a, \_\_m128 b)
\]

Computes the maximum of the four SP FP values of \( a \) and \( b \).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{max(a0, b0)} & \text{max(a1, b1)} & \text{max(a2, b2)} & \text{max(a3, b3)}
\end{array}
\]
Logical Operations for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for logical operations are in the `xmmintrin.h` header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R0-R3. R0, R1, R2 and R3 each represent one of the four 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_ps</td>
<td>Bitwise AND</td>
<td>ANDPS</td>
</tr>
<tr>
<td>_mm_andnot_ps</td>
<td>Bitwise ANDNOT</td>
<td>ANDNPS</td>
</tr>
<tr>
<td>_mm_or_ps</td>
<td>Bitwise OR</td>
<td>ORPS</td>
</tr>
<tr>
<td>_mm_xor_ps</td>
<td>Bitwise Exclusive OR</td>
<td>XORPS</td>
</tr>
</tbody>
</table>

__m128 _mm_and_ps(__m128 a, __m128 b)

Computes the bitwise AND of the four SP FP values of a and b.

R0  R1  R2  R3
a0 & b0 a1 & b1 a2 & b2 a3 & b3

__m128 _mm_andnot_ps(__m128 a, __m128 b)

Computes the bitwise AND-NOT of the four SP FP values of a and b.

R0  R1  R2  R3
~a0 & b0 ~a1 & b1 ~a2 & b2 ~a3 & b3

__m128 _mm_or_ps(__m128 a, __m128 b)

Computes the bitwise OR of the four SP FP values of a and b.

R0  R1  R2  R3
a0 & b0 a1 & b1 a2 & b2 a3 & b3
__m128 __m_xor_ps(__m128 a, __m128 b)

Computes bitwise XOR (exclusive-or) of the four SP FP values of \(a\) and \(b\).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>b0</td>
<td>a1</td>
<td>b1</td>
</tr>
</tbody>
</table>

Comparisons for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for comparison operations are in the `xmmintrin.h` header file.

Each comparison intrinsic performs a comparison of \(a\) and \(b\). For the packed form, the four SP FP values of \(a\) and \(b\) are compared, and a 128-bit mask is returned. For the scalar form, the lower SP FP values of \(a\) and \(b\) are compared, and a 32-bit mask is returned; the upper three SP FP values are passed through from \(a\). The mask is set to \(0xffffffff\) for each element where the comparison is true and \(0x0\) where the comparison is false.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R or R0-R3. R0, R1, R2 and R3 each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__mm_cmpeq_ss</td>
<td>Equal</td>
<td>CMPEQSS</td>
</tr>
<tr>
<td>__mm_cmpeq_ps</td>
<td>Equal</td>
<td>CMPEQPS</td>
</tr>
<tr>
<td>__mm_cmplt_ss</td>
<td>Less Than</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>__mm_cmplt_ps</td>
<td>Less Than</td>
<td>CMPLTPS</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE Instruction</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>_mm_cmple_ss</td>
<td>Less Than or Equal</td>
<td>CMPLESS</td>
</tr>
<tr>
<td>_mm_cmple_ps</td>
<td>Less Than or Equal</td>
<td>CMPEPS</td>
</tr>
<tr>
<td>_mm_cmpgt_ss</td>
<td>Greater Than</td>
<td>CMPLTSS</td>
</tr>
<tr>
<td>_mm_cmpgt_ps</td>
<td>Greater Than</td>
<td>CMPLTPS</td>
</tr>
<tr>
<td>_mm_cmpge_ss</td>
<td>Greater Than or Equal</td>
<td>CMPLESS</td>
</tr>
<tr>
<td>_mm_cmpge_ps</td>
<td>Greater Than or Equal</td>
<td>CMPEPS</td>
</tr>
<tr>
<td>_mm_cmpneq_ss</td>
<td>Not Equal</td>
<td>CMPNEQSS</td>
</tr>
<tr>
<td>_mm_cmpneq_ps</td>
<td>Not Equal</td>
<td>CMPNEQPS</td>
</tr>
<tr>
<td>_mm_cmpnlt_ss</td>
<td>Not Less Than</td>
<td>CMPNLTSS</td>
</tr>
<tr>
<td>_mm_cmpnlt_ps</td>
<td>Not Less Than</td>
<td>CMPNLTPS</td>
</tr>
<tr>
<td>_mm_cmpnle_ss</td>
<td>Not Less Than or Equal</td>
<td>CMPNLESS</td>
</tr>
<tr>
<td>_mm_cmpnle_ps</td>
<td>Not Less Than or Equal</td>
<td>CMPNLEPS</td>
</tr>
<tr>
<td>_mm_cmpngt_ss</td>
<td>Not Greater Than</td>
<td>CMPNLTSS</td>
</tr>
<tr>
<td>_mm_cmpngt_ps</td>
<td>Not Greater Than</td>
<td>CMPNLTPS</td>
</tr>
<tr>
<td>_mm_cmpnge_ss</td>
<td>Not Greater Than or Equal</td>
<td>CMPNLESS</td>
</tr>
<tr>
<td>_mm_cmpnge_ps</td>
<td>Not Greater Than or Equal</td>
<td>CMPNLEPS</td>
</tr>
<tr>
<td>_mm_cmpord_ss</td>
<td>Ordered</td>
<td>CMPORDSS</td>
</tr>
<tr>
<td>_mm_cmpord_ps</td>
<td>Ordered</td>
<td>CMPORDPS</td>
</tr>
<tr>
<td>_mm_cmpunord_ss</td>
<td>Unordered</td>
<td>CMPUNORDSS</td>
</tr>
<tr>
<td>_mm_cmpunord_ps</td>
<td>Unordered</td>
<td>CMPUNORDPS</td>
</tr>
<tr>
<td>_mm_comieq_ss</td>
<td>Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comilt_ss</td>
<td>Less Than</td>
<td>COMISS</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE Instruction</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_comile_ss</td>
<td>Less Than or Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comiglt_ss</td>
<td>Greater Than</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comige_ss</td>
<td>Greater Than or Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_comineq_ss</td>
<td>Not Equal</td>
<td>COMISS</td>
</tr>
<tr>
<td>_mm_ucomieq_ss</td>
<td>Equal</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomilt_ss</td>
<td>Less Than</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomile_ss</td>
<td>Less Than or Equal</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomiglt_ss</td>
<td>Greater Than</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomige_ss</td>
<td>Greater Than or Equal</td>
<td>UCOMISS</td>
</tr>
<tr>
<td>_mm_ucomineq_ss</td>
<td>Not Equal</td>
<td>UCOMISS</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpeq_ss(__m128 a, __m128 b)

Compares for equality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpeq_ps(__m128 a, __m128 b)

Compares for equality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ?</td>
<td>(a1 == b1) ?</td>
<td>(a2 == b2) ?</td>
<td>(a3 == b3) ?</td>
</tr>
<tr>
<td>0xffffffff</td>
<td>0xffffffff</td>
<td>0xffffffff</td>
<td>0xffffffff</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>
__m128_mm_cmplt_ss(__m128 a, __m128 b)

Compares for less-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128_mm_cmplt_ps(__m128 a, __m128 b)

Compares for less-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0)</td>
<td>(a1 &lt; b1)</td>
<td>(a2 &lt; b2)</td>
<td>(a3 &lt; b3)</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

__m128_mm_cmple_ss(__m128 a, __m128 b)

Compares for less-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt;= b0)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128_mm_cmple_ps(__m128 a, __m128 b)

Compares for less-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt;= b0)</td>
<td>(a1 &lt;= b1)</td>
<td>(a2 &lt;= b2)</td>
<td>(a3 &lt;= b3)</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

__m128_mm_cmpgt_ss(__m128 a, __m128 b)
Compares for greater-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &gt; b0)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
__m128 _mm_cmpgt_ps(__m128 a, __m128 b)
```

Compares for greater-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &gt; b0)</td>
<td>(a1 &gt; b1)</td>
<td>(a2 &gt; b2)</td>
<td>(a3 &gt; b3)</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

```c
__m128 _mm_cmpge_ss(__m128 a, __m128 b)
```

Compares for greater-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &gt;= b0)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

```c
__m128 _mm_cmpge_ps(__m128 a, __m128 b)
```

Compares for greater-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &gt;= b0)</td>
<td>(a1 &gt;= b1)</td>
<td>(a2 &gt;= b2)</td>
<td>(a3 &gt;= b3)</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

```c
__m128 _mm_cmpneq_ss(__m128 a, __m128 b)
```

Compares for inequality.


<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
</tr>
<tr>
<td>(a0 != b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpneq_ps(__m128 a, __m128 b)

Compares for inequality.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
</tr>
<tr>
<td>(a0 != b0) ?</td>
<td>(a1 != b1) ?</td>
<td>(a2 != b2) ?</td>
<td>(a3 != b3) ?</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnlt_ss(__m128 a, __m128 b)

Compares for not-less-than.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
</tr>
<tr>
<td>!(a0 &lt; b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnlt_ps(__m128 a, __m128 b)

Compares for not-less-than.

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>R0</td>
<td>R1</td>
<td>R2</td>
<td>R3</td>
</tr>
<tr>
<td>!(a0 &lt; b0) ?</td>
<td>!(a1 &lt; b1) ?</td>
<td>!(a2 &lt; b2) ?</td>
<td>!(a3 &lt; b3) ?</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnle_ss(__m128 a, __m128 b)

Compares for not-less-than-or-equal.
<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt;= b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnle_ps(__m128 a, __m128 b)

Compares for not-less-than-or-equal.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt;= b0) ?</td>
<td>!(a1 &lt;= b1) ?</td>
<td>!(a2 &lt;= b2) ?</td>
<td>!(a3 &lt;= b3) ?</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpngt_ss(__m128 a, __m128 b)

Compares for not-greater-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &gt; b0) ?</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpngt_ps(__m128 a, __m128 b)

Compares for not-greater-than.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &gt; b0) ?</td>
<td>!(a1 &gt; b1) ?</td>
<td>!(a2 &gt; b2) ?</td>
<td>!(a3 &gt; b3) ?</td>
</tr>
<tr>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
<td>0xffffffff :</td>
</tr>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
</tr>
</tbody>
</table>

__m128 _mm_cmpnge_ss(__m128 a, __m128 b)

Compares for not-greater-than-or-equal.
__(a0 >= b0) ? a1 a2 a3
0xffffffff :
0x0

__m128 _mm_cmpnge_ps(__m128 a, __m128 b)

Compares for not-greater-than-or-equal.

__(a0 >= b0) ? !(a1 >= b1) ? !(a2 >= b2) ? !(a3 >= b3) ?
0xffffffff : 0xffffffff : 0xffffffff : 0xffffffff :
0x0 0x0 0x0 0x0

__m128 _mm_cmpord_ss(__m128 a, __m128 b)

Compares for ordered.

(a0 ord? b0) ? a1 a2 a3
0xffffffff :
0x0

__m128 _mm_cmpord_ps(__m128 a, __m128 b)

Compares for ordered.

(a0 ord? b0) ? (a1 ord? b1) ? (a2 ord? b2) ? (a3 ord? b3) ?
0xffffffff : 0xffffffff : 0xffffffff : 0xffffffff :
0x0 0x0 0x0 0x0

__m128 _mm_cmpunord_ss(__m128 a, __m128 b)

Compares for unordered.
Intel(R) C++ Compiler User and Reference Guides

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 unord? b0)</td>
<td>a1</td>
<td>a2</td>
<td>a3</td>
</tr>
<tr>
<td>? 0xffffffff : 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128 _mm_cmpunord_ps(__m128 a, __m128 b)

Compares for unordered.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 unord? b0)</td>
<td>(a1 unord? b1)</td>
<td>(a2 unord? b2)</td>
<td>(a3 unord? b3)</td>
</tr>
<tr>
<td>? 0xffffffff : ? 0xffffffff : ? 0xffffffff : ? 0xffffffff : 0x0 0x0 0x0 0x0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

int _mm_comieq_ss(__m128 a, __m128 b)

Compares the lower SP FP value of a and b for a equal to b. If a and b are equal, 1 is returned. Otherwise 0 is returned.

<table>
<thead>
<tr>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ? 0x1 : 0x0</td>
</tr>
</tbody>
</table>

int _mm_comilt_ss(__m128 a, __m128 b)

Compares the lower SP FP value of a and b for a less than b. If a is less than b, 1 is returned. Otherwise 0 is returned.

<table>
<thead>
<tr>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0) ? 0x1 : 0x0</td>
</tr>
</tbody>
</table>

int _mm_comile_ss(__m128 a, __m128 b)

Compares the lower SP FP value of a and b for a less than or equal to b. If a is less than or equal to b, 1 is returned. Otherwise 0 is returned.
R
(a0 <= b0) ? 0x1 : 0x0

int _mm_comigt_ss(__m128 a, __m128 b)

Compares the lower SP FP value of a and b for a greater than b. If a is greater than b are equal, 1 is returned. Otherwise 0 is returned.

R
(a0 > b0) ? 0x1 : 0x0

int _mm_comige_ss(__m128 a, __m128 b)

Compares the lower SP FP value of a and b for a greater than or equal to b. If a is greater than or equal to b, 1 is returned. Otherwise 0 is returned.

R
(a0 >= b0) ? 0x1 : 0x0

int _mm_comineq_ss(__m128 a, __m128 b)

Compares the lower SP FP value of a and b for a not equal to b. If a and b are not equal, 1 is returned. Otherwise 0 is returned.

R
(a0 != b0) ? 0x1 : 0x0

int _mm_ucomieq_ss(__m128 a, __m128 b)

Compares the lower SP FP value of a and b for a equal to b. If a and b are equal, 1 is returned. Otherwise 0 is returned.

R
(a0 == b0) ? 0x1 : 0x0

int _mm_ucomilt_ss(__m128 a, __m128 b)

1352
Compares the lower SP FP value of \( a \) and \( b \) for \( a \) less than \( b \). If \( a \) is less than \( b \), 1 is returned. Otherwise 0 is returned.

\[
\text{R}
\]
\[
(a0 < b0) ? 0x1 : 0x0
\]

\[\text{int } \_\text{mm_ucomile_ss}(\_\text{m128 }a, \_\text{m128 }b)\]

Compares the lower SP FP value of \( a \) and \( b \) for \( a \) less than or equal to \( b \). If \( a \) is less than or equal to \( b \), 1 is returned. Otherwise 0 is returned.

\[
\text{R}
\]
\[
(a0 <= b0) ? 0x1 : 0x0
\]

\[\text{int } \_\text{mm_ucomigt_ss}(\_\text{m128 }a, \_\text{m128 }b)\]

Compares the lower SP FP value of \( a \) and \( b \) for \( a \) greater than \( b \). If \( a \) is greater than or equal to \( b \), 1 is returned. Otherwise 0 is returned.

\[
\text{R}
\]
\[
(a0 > b0) ? 0x1 : 0x0
\]

\[\text{int } \_\text{mm_ucomige_ss}(\_\text{m128 }a, \_\text{m128 }b)\]

Compares the lower SP FP value of \( a \) and \( b \) for \( a \) greater than or equal to \( b \). If \( a \) is greater than or equal to \( b \), 1 is returned. Otherwise 0 is returned.

\[
\text{R}
\]
\[
(a0 >= b0) ? 0x1 : 0x0
\]

\[\text{int } \_\text{mm_ucombeq_ss}(\_\text{m128 }a, \_\text{m128 }b)\]

Compares the lower SP FP value of \( a \) and \( b \) for \( a \) not equal to \( b \). If \( a \) and \( b \) are not equal, 1 is returned. Otherwise 0 is returned.
Conversion Operations for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for conversion operations are in the `xmmintrin.h` header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R or R0-R3. R0, R1, R2 and R3 each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cvtsi32</td>
<td>Convert to 32-bit integer</td>
<td>CVTSI2S</td>
</tr>
<tr>
<td>_mm_cvtsi64</td>
<td>Convert from 32-bit integer</td>
<td>CVTSI2S</td>
</tr>
<tr>
<td>_mm_cvtps_pi32</td>
<td>Convert to two 32-bit integers</td>
<td>CVTPS2PI</td>
</tr>
<tr>
<td>_mm_cvttss_si32</td>
<td>Convert to 32-bit integer</td>
<td>CVTSS2S</td>
</tr>
<tr>
<td>_mm_cvttss_si64</td>
<td>Convert to 64-bit integer</td>
<td>CVTSS2S</td>
</tr>
<tr>
<td>_mm_cvttss_pi32</td>
<td>Convert to two 32-bit integers</td>
<td>CVTSS2S</td>
</tr>
<tr>
<td>_mm_cvtss_si32</td>
<td>Convert to 32-bit integer</td>
<td>CVTSS2S</td>
</tr>
<tr>
<td>_mm_cvtss_si64</td>
<td>Convert to 64-bit integer</td>
<td>CVTSS2S</td>
</tr>
<tr>
<td>_mm_cvtts_pi32</td>
<td>Convert to two 32-bit integers</td>
<td>CVTSS2S</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE Instruction</td>
</tr>
<tr>
<td>----------------------</td>
<td>----------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_cvtpu16_ps</td>
<td>Convert from four 16-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtpi8_ps</td>
<td>Convert from four 8-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtpu8_ps</td>
<td>Convert from four 8-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtpi32x2_ps</td>
<td>Convert from four 32-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtps_pi16</td>
<td>Convert to four 16-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvtps_pi8</td>
<td>Convert to four 8-bit integers</td>
<td>composite</td>
</tr>
<tr>
<td>_mm_cvts_f32</td>
<td>Extract</td>
<td>composite</td>
</tr>
</tbody>
</table>

int _mm_cvtss_si32(__m128 a)

Converts the lower SP FP value of a to a 32-bit integer according to the current rounding mode.

R (int)a0

__int64 _mm_cvtss_si64(__m128 a)

Converts the lower SP FP value of a to a 64-bit signed integer according to the current rounding mode.
__m64 _mm_cvtps_pi32(__m128 a)

Converts the two lower SP FP values of \texttt{a} to two 32-bit integers according to the current rounding mode, returning the integers in packed form.

(int)a0 (int)a1

int _mm_cvttss_si32(__m128 a)

Converts the lower SP FP value of \texttt{a} to a 32-bit integer with truncation.

(int)a0

__int64 _mm_cvttss_si64(__m128 a)

Converts the lower SP FP value of \texttt{a} to a 64-bit signed integer with truncation.

(int)a0

__m64 _mm_cvtps_pi32(__m128 a)

Converts the two lower SP FP values of \texttt{a} to two 32-bit integer with truncation, returning the integers in packed form.

(int)a0 (int)a1

__m128 _mm_cvtsi32_ss(__m128 a, int b)

Converts the 32-bit integer value \texttt{b} to an SP FP value; the upper three SP FP values are passed through from \texttt{a}.
__m128 _mm_cvtsi64_ss(__m128 a, __int64 b)

Converts the signed 64-bit integer value \( b \) to an SP FP value; the upper three SP FP values are passed through from \( a \).

__m128 _mm_cvtpi32_ps(__m128 a, __m64 b)

Converts the two 32-bit integer values in packed form in \( b \) to two SP FP values; the upper two SP FP values are passed through from \( a \).

__m128 _mm_cvtpi16_ps(__m64 a)

Converts the four 16-bit signed integer values in \( a \) to four single precision FP values.

__m128 _mm_cvtpu16_ps(__m64 a)

Converts the four 16-bit unsigned integer values in \( a \) to four single precision FP values.

__m128 _mm_cvtpi8_ps(__m64 a)
Converts the lower four 8-bit signed integer values in \( a \) to four single precision FP values.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{float})a0 & (\text{float})a1 & (\text{float})a2 & (\text{float})a3 \\
\end{array}
\]

\[
\_\_m128 \_\_m\_cvtpu8\_ps(\_\_m64 a)
\]

Converts the lower four 8-bit unsigned integer values in \( a \) to four single precision FP values.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{float})a0 & (\text{float})a1 & (\text{float})a2 & (\text{float})a3 \\
\end{array}
\]

\[
\_\_m128 \_\_m\_cvtpi32x2\_ps(\_\_m64 a, \_\_m64 b)
\]

Converts the two 32-bit signed integer values in \( a \) and the two 32-bit signed integer values in \( b \) to four single precision FP values.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{float})a0 & (\text{float})a1 & (\text{float})b0 & (\text{float})b1 \\
\end{array}
\]

\[
\_\_m64 \_\_m\_cvtpsi16(\_\_m128 a)
\]

Converts the four single precision FP values in \( a \) to four signed 16-bit integer values.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
(\text{short})a0 & (\text{short})a1 & (\text{short})a2 & (\text{short})a3 \\
\end{array}
\]

\[
\_\_m64 \_\_m\_cvtpsi8(\_\_m128 a)
\]

Converts the four single precision FP values in \( a \) to the lower four signed 8-bit integer values of the result.
float _mm_cvtss_f32(__m128 a)

Extracts a single precision floating point value from the first vector element of an __m128. It does so in the most efficient manner possible in the context used.

Load Operations for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for load operations are in the xmmintrin.h header file.

The results of each intrinsic operation are placed in a register. This register is illustrated for each intrinsic with R0-R3. R0, R1, R2 and R3 each represent one of the 4 32-bit pieces of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_loadh_pi</td>
<td>Load high</td>
<td>MOVHPS reg, mem</td>
</tr>
<tr>
<td>_mm_loadl_pi</td>
<td>Load low</td>
<td>MOVLPS reg, mem</td>
</tr>
<tr>
<td>_mm_load_ss</td>
<td>Load the low value and clear the</td>
<td>MOVSS</td>
</tr>
<tr>
<td></td>
<td>three high values</td>
<td></td>
</tr>
<tr>
<td>_mm_load1_ps</td>
<td>Load one value into all four</td>
<td>MOVSS + Shuffling</td>
</tr>
<tr>
<td></td>
<td>words</td>
<td></td>
</tr>
<tr>
<td>_mm_load_ps</td>
<td>Load four values, address</td>
<td>MOVAPS</td>
</tr>
<tr>
<td></td>
<td>aligned</td>
<td></td>
</tr>
<tr>
<td>_mm_loadu_ps</td>
<td>Load four values, address</td>
<td>MOVUPS</td>
</tr>
<tr>
<td></td>
<td>unaligned</td>
<td></td>
</tr>
<tr>
<td>_mm_loadr_ps</td>
<td>Load four values in reverse</td>
<td>MOVAPS + Shuffling</td>
</tr>
</tbody>
</table>
__m128 _mm_loadh_pi(__m128 a, __m64 const *p)
Sets the upper two SP FP values with 64 bits of data loaded from the address p.

R0 R1 R2 R3
a0 a1 *p0 *p1

__m128 _mm_loadl_pi(__m128 a, __m64 const *p)
Sets the lower two SP FP values with 64 bits of data loaded from the address p; the upper two values are passed through from a.

R0 R1 R2 R3
*p0 *p1 a2 a3

__m128 _mm_load_ss(float * p )
 Loads an SP FP value into the low word and clears the upper three words.

R0 R1 R2 R3
*p 0.0 0.0 0.0

__m128 _mm_load1_ps(float * p )
 Loads a single SP FP value, copying it into all four words.

R0 R1 R2 R3
*p *p *p *p

__m128 _mm_load_ps(float * p )
 Loads four SP FP values. The address must be 16-byte-aligned.

R0 R1 R2 R3

__m128 _mm_loadu_ps(float * p)
Loads four SP FP values. The address need not be 16-byte-aligned.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\end{array}
\]

__m128 \_mm_loadr\_ps(float \* p)

Loads four SP FP values in reverse order. The address must be 16-byte-aligned.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\end{array}
\]

Set Operations for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for set operations are in the \texttt{xmmintrin.h} header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R0, R1, R2 and R3 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_set_ss</td>
<td>Set the low value and clear the three high values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set1_ps</td>
<td>Set all four words with the same value</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set_ps</td>
<td>Set four values, address aligned</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_ps</td>
<td>Set four values, in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE Instruction</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_setzero_ps</td>
<td>Clear all four values</td>
<td>Composite</td>
</tr>
</tbody>
</table>

```
__m128_mm_set_ss(float w)

Sets the low word of an SP FP value to \( w \) and clears the upper three words.
```

| R0 R1 R2 R3                      | w 0.0 0.0 0.0               |

```
__m128_mm_set1_ps(float w)

Sets the four SP FP values to \( w \).
```

| R0 R1 R2 R3                      | w w w w                     |

```
__m128_mm_set_ps(float z, float y, float x, float w)

Sets the four SP FP values to the four inputs.
```

| R0 R1 R2 R3                      | w x y z                     |

```
__m128_mm_setr_ps(float z, float y, float x, float w)

Sets the four SP FP values to the four inputs in reverse order.
```

| R0 R1 R2 R3                      | z y x w                     |

```
__m128_mm_setzero_ps(void)

Clears the four SP FP values.
```

| R0 R1 R2 R3                      |                           |

1362
Store Operations for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for store operations are in the \texttt{xmmintrin.h} header file.

The description for each intrinsic contains a table detailing the returns. In these tables, \( p[n] \) is an access to the \( n \) element of the result.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_storeh_pi</td>
<td>Store high</td>
<td>MOVHPS mem, reg</td>
</tr>
<tr>
<td>_mm_storel_pi</td>
<td>Store low</td>
<td>MOVLPS mem, reg</td>
</tr>
<tr>
<td>_mm_store_ss</td>
<td>Store the low value</td>
<td>MOVSS</td>
</tr>
<tr>
<td>_mm_storel_ps</td>
<td>Store the low value across all four words, address aligned</td>
<td>Shuffling + MOVSS</td>
</tr>
<tr>
<td>_mm_store_ps</td>
<td>Store four values, address aligned</td>
<td>MOVAPS</td>
</tr>
<tr>
<td>_mm_storeu_ps</td>
<td>Store four values, address unaligned</td>
<td>MOVUPS</td>
</tr>
<tr>
<td>_mm_storer_ps</td>
<td>Store four values, in reverse order</td>
<td>MOVAPS + Shuffling</td>
</tr>
</tbody>
</table>

\texttt{void \_mm_storeh\_pi(__m64 \*p, __m128 a)}

Stores the upper two SP FP values to the address \( p \).
_mm_storel_pi(_m64 *p, _m128 a)

Stores the lower two SP FP values of \( a \) to the address \( p \).

_mm_store_ss(float * p, _m128 a)

Stores the lower SP FP value.

_mm_storel_ps(float * p, _m128 a)

Stores the lower SP FP value across four words.

_mm_store_ps(float *p, _m128 a)

Stores four SP FP values. The address must be 16-byte-aligned.

_mm_storeu_ps(float *p, _m128 a)

Stores four SP FP values. The address need not be 16-byte-aligned.
void _mm_storer_ps(float * p, __m128 a )
Stores four SP FP values in reverse order. The address must be 16-byte-aligned.

Cacheability Support Using Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for cacheability support are in the \texttt{xmmintrin.h} header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_prefetch</td>
<td>Load</td>
<td>PREFETCH</td>
</tr>
<tr>
<td>_mm_stream_pi</td>
<td>Store</td>
<td>MOVNTQ</td>
</tr>
<tr>
<td>_mm_stream_ps</td>
<td>Store</td>
<td>MOVNTPS</td>
</tr>
<tr>
<td>_mm_sfence</td>
<td>Store fence</td>
<td>SFENCE</td>
</tr>
</tbody>
</table>

void _mm_prefetch(char const*a, int sel)
Loads one cache line of data from address \texttt{a} to a location "closer" to the processor. The value \texttt{sel} specifies the type of prefetch operation: the constants _MM_HINT_T0, _MM_HINT_T1, _MM_HINT_T2, \texttt{and} _MM_HINT_NTA should be used for IA-32, corresponding to the type of \texttt{prefetch} instruction. The constants _MM_HINT_T1, _MM_HINT_NT1, _MM_HINT_NT2, \texttt{and} _MM_HINT_NTA should be used for systems based on IA-64 architecture.

void _mm_stream_pi(__m64 *p, __m64 a)
Stores the data in $a$ to the address $p$ without polluting the caches. This intrinsic requires you to empty the multimedia state for the MMX register. See the topic The EMMS Instruction: Why You Need It.

```c
void _mm_stream_ps(float *p, __m128 a)
```

Stores the data in $a$ to the address $p$ without polluting the caches. The address must be 16-byte-aligned.

```c
void _mm_sfence(void)
```

Guarantees that every preceding store is globally visible before any subsequent store.

### Integer Intrinsics Using Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for integer operations are in the `xmmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1...R7 represent the registers in which results are placed.

Before using these intrinsics, you must empty the multimedia state for the MMX™ technology register. See The EMMS Instruction: Why You Need It for more details.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_extract_pi16</td>
<td>Extract one of four words</td>
<td>PEXTRW</td>
</tr>
<tr>
<td>_mm_insert_pi16</td>
<td>Insert word</td>
<td>PINSRW</td>
</tr>
<tr>
<td>_mm_max_pi16</td>
<td>Compute maximum</td>
<td>PMAXSW</td>
</tr>
<tr>
<td>_mm_max_ps8</td>
<td>Compute maximum, unsigned</td>
<td>PMAXUB</td>
</tr>
<tr>
<td>_mm_min_pi16</td>
<td>Compute minimum</td>
<td>PMINSW</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE Instruction</td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_min_pu8</td>
<td>Compute minimum, unsigned</td>
<td>PMINUB</td>
</tr>
<tr>
<td>_mm_movemask_pi8</td>
<td>Create eight-bit mask</td>
<td>PMOVMSKB</td>
</tr>
<tr>
<td>_mm_mulhi_pu16</td>
<td>Multiply, return high bits</td>
<td>PMULHUW</td>
</tr>
<tr>
<td>_mm_shuffle_pi16</td>
<td>Return a combination of four words</td>
<td>PSHUFW</td>
</tr>
<tr>
<td>_mm_maskmove_si64</td>
<td>Conditional Store</td>
<td>MASKMOVQ</td>
</tr>
<tr>
<td>_mm_avg_pu8</td>
<td>Compute rounded average</td>
<td>PAVGB</td>
</tr>
<tr>
<td>_mm_avg_pu16</td>
<td>Compute rounded average</td>
<td>PAVGW</td>
</tr>
<tr>
<td>_mm_sad_pu8</td>
<td>Compute sum of absolute differences</td>
<td>PSADBW</td>
</tr>
</tbody>
</table>

```c
int _mm_extract_pi16(__m64 a, int n)
```

Extracts one of the four words of `a`. The selector `n` must be an immediate.

```c
R
(n==0) ? a0 : ( (n==1) ? a1 : ( (n==2) ? a2 : a3 ) )
```

```c
__m64 _mm_insert_pi16(__m64 a, int d, int n)
```

Inserts word `d` into one of four words of `a`. The selector `n` must be an immediate.

```c
R0
(n==0) ? d : ( (n==1) ? d : ( (n==2) ? d : (n==3) ? d : a0 ; a1 ; a2 ; a3 ;
```
__m64 _mm_max_pi16(__m64 a, __m64 b)

Computes the element-wise maximum of the words in a and b.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\hline
\min(a0, b0) & \min(a1, b1) & \min(a2, b2) & \min(a3, b3) \\
\end{array}
\]

__m64 _mm_max_pu8(__m64 a, __m64 b)

Computes the element-wise maximum of the unsigned bytes in a and b.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\hline
\min(a0, b0) & \min(a1, b1) & \ldots & \min(a7, b7) \\
\end{array}
\]

__m64 _mm_min_pi16(__m64 a, __m64 b)

Computes the element-wise minimum of the words in a and b.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\hline
\min(a0, b0) & \min(a1, b1) & \min(a2, b2) & \min(a3, b3) \\
\end{array}
\]

__m64 _mm_min_pu8(__m64 a, __m64 b)

Computes the element-wise minimum of the unsigned bytes in a and b.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\hline
\min(a0, b0) & \min(a1, b1) & \ldots & \min(a7, b7) \\
\end{array}
\]

__m64 _mm_movemask_pi8(__m64 b)

Creates an 8-bit mask from the most significant bits of the bytes in a.

\[
R
\]

\[
sign(a7)<<7 | sign(a6)<<6 | \ldots | sign(a0)
\]

__m64 _mm_mulhi_pu16(__m64 a, __m64 b)
Multiplies the unsigned words in \(a\) and \(b\), returning the upper 16 bits of the 32-bit intermediate results.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{hiword}(a0 \times b0) & \text{hiword}(a1 \times b1) & \text{hiword}(a2 \times b2) & \text{hiword}(a3 \times b3) \\
\end{array}
\]

\[
\_\text{m64} \_\text{mm\_shuffle\_pi16(\_m64 } a, \text{ int } n)\]

Returns a combination of the four words of \(a\). The selector \(n\) must be an immediate.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\text{word } (n\&0x3) & \text{word } ((n>>2)&0x3) & \text{word } ((n>>4)&0x3) & \text{word } ((n>>6)&0x3) \\
\text{of } a & \text{of } a & \text{of } a & \text{of } a \\
\end{array}
\]

\[
\text{void } \_\text{mm\_maskmove\_si64(\_m64 } d, \_\text{m64 } n, \text{ char } *p)\]

Conditionally stores byte elements of \(d\) to address \(p\). The high bit of each byte in the selector \(n\) determines whether the corresponding byte in \(d\) will be stored.

\[
\text{if } (\text{sign}(n0)) \quad \text{if } (\text{sign}(n1)) \quad \ldots \quad \text{if } (\text{sign}(n7))
\]

\[
p[0] := d0 \quad p[1] := d1 \quad \ldots \quad p[7] := d7
\]

\[
\_\text{m64} \_\text{mm\_avg\_pu8(\_m64 } a, \_\text{m64 } b)\]

Computes the (rounded) averages of the unsigned bytes in \(a\) and \(b\).

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
((t >> 1) \mid (t \& 0x01)), \text{ where } t & ((t >> 1) \mid (t \& 0x01)), \text{ where } t & ((t >> 1) \mid (t \& 0x01)), \text{ where } t & ((t >> 1) \mid (t \& 0x01)), \text{ where } t \\
\text{unsigned } char \text{a0 } + & \text{unsigned } char \text{a1 } + & \text{unsigned } char \text{a7 } + & \text{unsigned } char \text{a7 } +
\end{array}
\]
_mm_avg_pu16(_mm a, _mm b)

Computes the (rounded) averages of the unsigned short in a and b.

__m64 _mm_sad_pu8(__m64 a, __m64 b)

Computes the sum of the absolute differences of the unsigned bytes in a and b, returning the value in the lower word. The upper three words are cleared.

Intrinsics to Read and Write Registers for Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics to read from and write to registers are in the xmmintrin.h header file.
### Intrinsic Name

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_getcsr</td>
<td>Return control register</td>
<td>STMXCSR</td>
</tr>
<tr>
<td>_mm_setcsr</td>
<td>Set control register</td>
<td>LDMXCSR</td>
</tr>
</tbody>
</table>

**unsigned int _mm_getcsr(void)**

Returns the contents of the control register.

**void _mm_setcsr(unsigned int i)**

Sets the control register to the value specified.

### Miscellaneous Intrinsics Using Streaming SIMD Extensions

The prototypes for Streaming SIMD Extensions (SSE) intrinsics for miscellaneous operations are in the `xmmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0, R1, R2 and R3 represent the registers in which results are placed.
__m128 _mm_shuffle_ps(__m128 a, __m128 b, unsigned int imm8)
Selects four specific SP FP values from a and b, based on the mask imm8. The
mask must be an immediate. See Macro Function for Shuffle Using Streaming
SIMD Extensions for a description of the shuffle semantics.

__m128 _mm_unpackhi_ps(__m128 a, __m128 b)
Selects and interleaves the upper two SP FP values from a and b.

```
R0 R1 R2 R3
a2 b2 a3 b3
```

__m128 _mm_unpacklo_ps(__m128 a, __m128 b)
Selects and interleaves the lower two SP FP values from a and b.

```
R0 R1 R2 R3
a0 b0 a1 b1
```

__m128 _mm_move_ss( __m128 a, __m128 b)
Sets the low word to the SP FP value of b. The upper 3 SP FP values are
passed through from a.

```
R0 R1 R2 R3
b0 a1 a2 a3
```

__m128 _mm_movehl_ps(__m128 a, __m128 b)
Moves the upper 2 SP FP values of b to the lower 2 SP FP values of the result.
The upper 2 SP FP values of a are passed through to the result.

```
R0 R1 R2 R3
b2 b3 a2 a3
```

__m128 _mm_movelh_ps(__m128 a, __m128 b)
Moves the lower 2 SP FP values of $b$ to the upper 2 SP FP values of the result.
The lower 2 SP FP values of $a$ are passed through to the result.

```
R0  R1  R2  R3
a0  a1  b0  b1
```

```c
int _mm_movemask_ps(__m128 a)
```

Creates a 4-bit mask from the most significant bits of the four SP FP values.

```
R
sign(a3)<<3 | sign(a2)<<2 | sign(a1)<<1 | sign(a0)
```

**Using Streaming SIMD Extensions on IA-64 Architecture**

The Streaming SIMD Extensions (SSE) intrinsics provide access to IA-64 instructions for Streaming SIMD Extensions. To provide source compatibility with the IA-32 architecture, these intrinsics are equivalent both in name and functionality to the set of IA-32 architecture-based SSE intrinsics.

To write programs with the intrinsics, you should be familiar with the hardware features provided by SSE. Keep the following issues in mind:

- Certain intrinsics are provided only for compatibility with previously-defined IA-32 architecture-based intrinsics. Using them on systems based on IA-64 architecture probably leads to performance degradation.
- Floating-point (FP) data loaded stored as __m128 objects must be 16-byte-aligned.
- Some intrinsics require that their arguments be immediates -- that is, constant integers (literals), due to the nature of the instruction.

**Data Types**

The new data type __m128 is used with the SSE intrinsics. It represents a 128-bit quantity composed of four single-precision FP values. This corresponds to the 128-bit IA-32 architecture-based Streaming SIMD Extensions register.
The compiler aligns \texttt{__m128} local data to 16-byte boundaries on the stack. Global data of these types is also 16 byte-aligned. To align integer, float, or double arrays, you can use the \texttt{declspec} alignment. 

Because IA-64 instructions treat the SSE registers in the same way whether you are using packed or scalar data, there is no \texttt{__m32} data type to represent scalar data. For scalar operations, use the \texttt{__m128} objects and the "scalar" forms of the intrinsics; the compiler and the processor implement these operations with 32-bit memory references. But, for better performance the packed form should be substituting for the scalar form whenever possible.

The address of a \texttt{__m128} object may be taken. For more information, see Intel Architecture Software Developer's Manual, Volume 2: Instruction Set Reference Manual, Intel Corporation, doc. number 243191.

**Implementation on Systems based on IA-64 architecture**

SSE intrinsics are defined for the \texttt{__m128} data type, a 128-bit quantity consisting of four single-precision FP values. SIMD instructions for systems based on IA-64 architecture operate on 64-bit FP register quantities containing two single-precision floating-point values. Thus, each \texttt{__m128} operand is actually a pair of FP registers and therefore each intrinsic corresponds to at least one pair of IA-64 instructions operating on the pair of FP register operands.

**Compatibility versus Performance**

Many of the SSE intrinsics for systems based on IA-64 architecture were created for compatibility with existing IA-32 architecture-based intrinsics and not for performance. In some situations, intrinsic usage that improved performance on IA-32 architecture will not do so on systems based on IA-64 architecture. One reason for this is that some intrinsics map nicely into the IA-32 instruction set but not into the IA-64 instruction set. Thus, it is important to differentiate between intrinsics which were implemented for a performance advantage on systems...
based on IA-64 architecture, and those implemented simply to provide compatibility with existing IA-32 architecture-based code.

The following intrinsics are likely to reduce performance and should only be used to initially port legacy code or in non-critical code sections:

- Any SSE scalar intrinsic (_ss variety) - use packed (_ps) version if possible
- comi and ucomi SSE comparisons - these correspond to IA-32 architecture-based COMISS and UCOMISS instructions only. A sequence of IA-64 instructions are required to implement these.
- Conversions in general are multi-instruction operations. These are particularly expensive: _mm_cvtpi16_ps, _mm_cvtpu16_ps, _mm_cvtpi8_ps, _mm_cvtpu8_ps, _mm_cvtpi32x2_ps, _mm_cvtps_pi16, _mm_cvtps_pi8
- SSE utility intrinsic _mm_movemask_ps

If the inaccuracy is acceptable, the SIMD reciprocal and reciprocal square root approximation intrinsics (rcp and rsqrt) are much faster than the true div and sqrt intrinsics.

Macro Functions

Macro Function for Shuffle Using Streaming SIMD Extensions

The Streaming SIMD Extensions (SSE) provide a macro function to help create constants that describe shuffle operations. The macro takes four small integers (in the range of 0 to 3) and combines them into an 8-bit immediate value used by the SHUFPS instruction.

Shuffle Function Macro

```c
_MM_SHUFFLE2(x, y)
```

expands to the value of

\(<x><y> | y\)
You can view the four integers as selectors for choosing which two words from the first input operand and which two words from the second are to be put into the result word.

View of Original and Result Words with Shuffle Function Macro

```
; n1 = 127
; n2 = 127
n3 = _mm_shuffle_pd(n1, n2, _MM_SHUFFLE2(1, 0))
; n3 = 127
```

Macro Functions to Read and Write the Control Registers

The following macro functions enable you to read and write bits to and from the control register. For details, see Intrinsics to Read and Write Registers. For Itanium®-based systems, these macros do not allow you to access all of the bits of the FPSR. See the descriptions for the `getfpsr()` and `setfpsr()` intrinsics in the Native Intrinsics for IA-64 Instructions topic.

<table>
<thead>
<tr>
<th>Exception State Macros</th>
<th>Macro Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_SET_EXCEPTION_STATE(x)</td>
<td>_MM_EXCEPT_INVALID</td>
</tr>
<tr>
<td>_MM_GET_EXCEPTION_STATE()</td>
<td>_MM_EXCEPT_DIV_ZERO</td>
</tr>
<tr>
<td></td>
<td>_MM_EXCEPT_DENORM</td>
</tr>
<tr>
<td></td>
<td>_MM_EXCEPT_OVERFLOW</td>
</tr>
<tr>
<td></td>
<td>_MM_EXCEPT_UNDERFLOW</td>
</tr>
<tr>
<td></td>
<td>_MM_EXCEPT_INEXACT</td>
</tr>
</tbody>
</table>

Macro Definitions

Write to and read from the six least significant control register bits, respectively.

The following example tests for a divide-by-zero exception.
Exception State Macros with _MM_EXCEPT_DIV_ZERO

```c
if (_MM_GET_EXCEPTION_STATE(x) & _MM_EXCEPT_DIV_ZERO) {
    /* Exception has occurred */
}
```

### Exception Mask Macros

<table>
<thead>
<tr>
<th>Macro Definition</th>
<th>Argument</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_SET_EXCEPTION_MASK(x)</td>
<td>_MM_MASK_INVALID</td>
</tr>
<tr>
<td>_MM_GET_EXCEPTION_MASK()</td>
<td>_MM_MASK_DIV_ZERO, _MM_MASK_DENORM</td>
</tr>
<tr>
<td>_MM_MASK_OVERFLOW, _MM_MASK_UNDERFLOW, _MM_MASK_INEXACT</td>
<td></td>
</tr>
</tbody>
</table>

#### Macro Definitions

Write to and read from the seventh through twelfth control register bits, respectively.

- **Note**
  
  All six exception mask bits are always affected. Bits not set explicitly are cleared.

  - _MM_MASK_UNDERFLOW
  - _MM_MASK_INEXACT

To mask the overflow and underflow exceptions and unmask all other exceptions, use the macros as follows:

```
-MM_SET_EXCEPTION_MASK(_MM_MASK_OVERFLOW | _MM_MASK_UNDERFLOW)
```

The following table lists the macros to set and get rounding modes, and the macro arguments that can be passed with the macros.

<table>
<thead>
<tr>
<th>Rounding Mode</th>
<th>Macro Arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>_MM_SET_ROUNDING_MODE(x)</td>
<td>_MM_ROUND_NEAREST</td>
</tr>
<tr>
<td>_MM_GET_ROUNDING_MODE()</td>
<td>_MM_ROUND_DOWN</td>
</tr>
</tbody>
</table>
Rounding Mode | Macro Arguments
---|---
**Macro Definition** Write to and read from bits thirteen and fourteen of the control register.

- `_MM_ROUND_UP`
- `_MM_ROUND_TOWARD_ZERO`

To test the rounding mode for round toward zero, use the `_MM_ROUND_TOWARD_ZERO` macro as follows.

```c
if (_MM_GET_ROUNDING_MODE() == _MM_ROUND_TOWARD_ZERO) {
    /* Rounding mode is round toward zero */
}
```

The following table lists the macros to set and get the flush-to-zero mode and the macro arguments that can be used.

| Flush-to-Zero Mode | Macro Arguments |
---|---|
- `_MM_SET_FLUSH_ZERO_MODE(x)` | `_MM_FLUSH_ZERO_ON`
- `_MM_GET_FLUSH_ZERO_MODE()` | `_MM_FLUSH_ZERO_OFF`

**Macro Definition** Write to and read from bit fifteen of the control register.

To disable the flush-to-zero mode, use the `_MM_FLUSH_ZERO_OFF` macro.

```c
_MM_SET_FLUSH_ZERO_MODE(_MM_FLUSH_ZERO_OFF)
```

**Macro Function for Matrix Transposition**

The Streaming SIMD Extensions (SSE) provide the following macro function to transpose a 4 by 4 matrix of single precision floating point values.

```c
_MM_TRANSPOSE4_PS(row0, row1, row2, row3)
```

The arguments `row0`, `row1`, `row2`, and `row3` are `__m128` values whose elements form the corresponding rows of a 4 by 4 matrix. The matrix transposition is returned in arguments `row0`, `row1`, `row2`, and `row3` where `row0`
now holds column 0 of the original matrix, row1 now holds column 1 of the
original matrix, and so on.

The transposition function of this macro is illustrated in the "Matrix Transposition
Using the _MM_TRANSPOSE4_PS" figure.

**Intrinsics for Streaming SIMD Extensions 2**

**Overview: Streaming SIMD Extensions 2**

This section describes the C++ language-level features supporting the Intel®
Streaming SIMD Extensions 2 (SSE2) in the Intel® C++ Compiler. The features
are divided into two categories:

- **Floating-Point Intrinsics** -- describes the arithmetic, logical, compare,
  conversion, memory, and initialization intrinsics for the double-precision
  floating-point data type (__m128d).
- **Integer Intrinsics** -- describes the arithmetic, logical, compare, conversion,
  memory, and initialization intrinsics for the extended-precision integer data
  type (__m128i).

**Note**

There are no intrinsics for floating-point move operations. To move data
from one register to another, a simple assignment, \( A = B \), suffices,
where \( A \) and \( B \) are the source and target registers for the move operation.
On processors that do not support SSE2 instructions but do support MMX Technology, you can use the sse2mmx.h emulation pack to enable support for SSE2 instructions. You can use the sse2mmx.h header file for the following processors:

- Itanium® Processor
- Pentium® III Processor
- Pentium® II Processor
- Pentium® with MMX Technology

Some intrinsics are "composites" because they require more than one instruction to implement them. Intrinsics that require one instruction to implement them are referred to as "simple".

You should be familiar with the hardware features provided by the SSE2 when writing programs with the intrinsics. The following are three important issues to keep in mind:

- Certain intrinsics, such as _mm_loadr_pd and _mm_cmpgt_sd, are not directly supported by the instruction set. While these intrinsics are convenient programming aids, be mindful of their implementation cost.
- Data loaded or stored as __m128d objects must be generally 16-byte-aligned.
- Some intrinsics require that their argument be immediates, that is, constant integers (literals), due to the nature of the instruction.

The prototypes for SSE2 intrinsics are in the emmintrin.h header file.

You can also use the single ia32intrin.h header file for any IA-32 architecture-based intrinsics.

**Floating-point Intrinsics**

Floating-point Arithmetic Operations for Streaming SIMD Extensions 2
The Streaming SIMD Extensions 2 (SSE2) intrinsics for floating-point arithmetic operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the *emmintrin.h* header file.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by \texttt{R0} and \texttt{R1}, where \texttt{R0} and \texttt{R1} each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_sd</td>
<td>Addition</td>
<td>ADDSD</td>
</tr>
<tr>
<td>_mm_add_pd</td>
<td>Addition</td>
<td>ADDPD</td>
</tr>
<tr>
<td>_mm_sub_sd</td>
<td>Subtraction</td>
<td>SUBSD</td>
</tr>
<tr>
<td>_mm_sub_pd</td>
<td>Subtraction</td>
<td>SUBPD</td>
</tr>
<tr>
<td>_mm_mul_sd</td>
<td>Multiplication</td>
<td>MULSD</td>
</tr>
<tr>
<td>_mm_mul_pd</td>
<td>Multiplication</td>
<td>MULPD</td>
</tr>
<tr>
<td>_mm_div_sd</td>
<td>Division</td>
<td>DIVSD</td>
</tr>
<tr>
<td>_mm_div_pd</td>
<td>Division</td>
<td>DIVPD</td>
</tr>
<tr>
<td>_mm_sqrt_sd</td>
<td>Computes Square Root</td>
<td>SQRTSD</td>
</tr>
<tr>
<td>_mm_sqrt_pd</td>
<td>Computes Square Root</td>
<td>SQRTPD</td>
</tr>
<tr>
<td>_mm_min_sd</td>
<td>Computes Minimum</td>
<td>MINSD</td>
</tr>
<tr>
<td>_mm_min_pd</td>
<td>Computes Minimum</td>
<td>MINPD</td>
</tr>
<tr>
<td>_mm_max_sd</td>
<td>Computes Maximum</td>
<td>MAXSD</td>
</tr>
<tr>
<td>_mm_max_pd</td>
<td>Computes Maximum</td>
<td>MAXPD</td>
</tr>
</tbody>
</table>

\[
__m128d _mm_add_sd(__m128d a, __m128d b)
\]
Adds the lower DP FP (double-precision, floating-point) values of a and b; the upper DP FP value is passed through from a.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 + b0 & a1 \\
\end{array}
\]

\_m128d \_mm_add_pd(\_m128d a, \_m128d b)

Adds the two DP FP values of a and b.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 + b0 & a1 + b1 \\
\end{array}
\]

\_m128d \_mm_add_sd(\_m128d a, \_m128d b)

Subtracts the lower DP FP value of b from a. The upper DP FP value is passed through from a.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 - b0 & a1 \\
\end{array}
\]

\_m128d \_mm_sub_pd(\_m128d a, \_m128d b)

Subtracts the two DP FP values of b from a.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 - b0 & a1 - b1 \\
\end{array}
\]

\_m128d \_mm_sub_sd(\_m128d a, \_m128d b)

Multiplies the lower DP FP values of a and b. The upper DP FP is passed through from a.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 * b0 & a1 \\
\end{array}
\]

\_m128d \_mm_mul_pd(\_m128d a, \_m128d b)
Multiplies the two DP FP values of \(a\) and \(b\).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 & b0 \\
a1 & b1 \\
\end{array}
\]

\[
\_m128d \_mm\_div\_sd(\_m128d \ a, \_m128d \ b)
\]

Divides the lower DP FP values of \(a\) and \(b\). The upper DP FP value is passed through from \(a\).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 & b0 \\
a1 & \\
\end{array}
\]

\[
\_m128d \_mm\_div\_pd(\_m128d \ a, \_m128d \ b)
\]

Divides the two DP FP values of \(a\) and \(b\).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 & b0 \\
a1 & b1 \\
\end{array}
\]

\[
\_m128d \_mm\_sqrt\_sd(\_m128d \ a, \_m128d \ b)
\]

Computes the square root of the lower DP FP value of \(b\). The upper DP FP value is passed through from \(a\).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
\sqrt(b0) & a1 \\
\end{array}
\]

\[
\_m128d \_mm\_sqrt\_pd(\_m128d \ a)
\]

Computes the square roots of the two DP FP values of \(a\).

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
\sqrt(a0) & \sqrt(a1) \\
\end{array}
\]

\[
\_m128d \_mm\_min\_sd(\_m128d \ a, \_m128d \ b)
\]
Computes the minimum of the lower DP FP values of \( a \) and \( b \). The upper DP FP value is passed through from \( a \).

\[
\begin{array}{|c|c|}
\hline
R0 & R1 \\
\hline
\text{min} (a0, b0) & a1 \\
\hline
\end{array}
\]

\( \text{__m128d } \text{mm_min_pd(__m128d } a, \text{__m128d } b) \)

Computes the minima of the two DP FP values of \( a \) and \( b \).

\[
\begin{array}{|c|c|}
\hline
R0 & R1 \\
\hline
\text{min} (a0, b0) & \text{min}(a1, b1) \\
\hline
\end{array}
\]

\( \text{__m128d } \text{mm_max_sd(__m128d } a, \text{__m128d } b) \)

Computes the maximum of the lower DP FP values of \( a \) and \( b \). The upper DP FP value is passed through from \( a \).

\[
\begin{array}{|c|c|}
\hline
R0 & R1 \\
\hline
\text{max} (a0, b0) & a1 \\
\hline
\end{array}
\]

\( \text{__m128d } \text{mm_max_pd(__m128d } a, \text{__m128d } b) \)

Computes the maxima of the two DP FP values of \( a \) and \( b \).

\[
\begin{array}{|c|c|}
\hline
R0 & R1 \\
\hline
\text{max} (a0, b0) & \text{max}(a1, b1) \\
\hline
\end{array}
\]

Floating-point Logical Operations for Streaming SIMD Extensions 2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for floating-point logical operations are listed in the following table. The prototypes for the SSE2 intrinsics are in the \textit{emmintrin.h} header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed
explanation for each intrinsic. For each intrinsic, the resulting register is represented by R0 and R1, where R0 and R1 each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_pd</td>
<td>Computes AND</td>
<td>ANDPD</td>
</tr>
<tr>
<td>_mm_andnot_pd</td>
<td>Computes AND and NOT</td>
<td>ANDNPD</td>
</tr>
<tr>
<td>_mm_or_pd</td>
<td>Computes OR</td>
<td>ORPD</td>
</tr>
<tr>
<td>_mm_xor_pd</td>
<td>Computes XOR</td>
<td>XORPD</td>
</tr>
</tbody>
</table>

__m128d _mm_and_pd(__m128d a, __m128d b)

Computes the bitwise **AND** of the two DP FP values of a and b.

```
R0     R1
a0 & b0 a1 & b1
```

__m128d _mm_andnot_pd(__m128d a, __m128d b)

Computes the bitwise **AND** of the 128-bit value in b and the bitwise **NOT** of the 128-bit value in a.

```
R0     R1
(~a0) & b0 (~a1) & b1
```

__m128d _mm_or_pd(__m128d a, __m128d b)

Computes the bitwise **OR** of the two DP FP values of a and b.

```
R0     R1
a0 | b0 a1 | b1
```

__m128d _mm_xor_pd(__m128d a, __m128d b)
Computes the bitwise XOR of the two DP FP values of \( a \) and \( b \).

R0       R1
\[
\begin{align*}
a0 & \ ^{\land}\ b0 \\
a1 & \ ^{\land}\ b1
\end{align*}
\]

**Floating-point Comparison Operations for Streaming SIMD Extensions 2**

The Streaming SIMD Extensions 2 (SSE2) intrinsics for floating-point comparison operations are listed in the following table. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

Each comparison intrinsic performs a comparison of \( a \) and \( b \). For the packed form, the two DP FP values of \( a \) and \( b \) are compared, and a 128-bit mask is returned. For the scalar form, the lower DP FP values of \( a \) and \( b \) are compared, and a 64-bit mask is returned; the upper DP FP value is passed through from \( a \).

The mask is set to \( 0xffffffffffffffff \) for each element where the comparison is true, and set to \( 0x0 \) where the comparison is false. The \( r \) following the instruction name indicates that the operands to the instruction are reversed in the actual implementation.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by \( R, R0 \) and \( R1 \), where \( R, R0 \) and \( R1 \) each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpeq_pd</td>
<td>Equality</td>
<td>CMPEQPD</td>
</tr>
<tr>
<td>_mm_cmplt_pd</td>
<td>Less Than</td>
<td>CMPLTPD</td>
</tr>
<tr>
<td>_mm_cmple_pd</td>
<td>Less Than or Equal</td>
<td>CMPLEPD</td>
</tr>
<tr>
<td>_mm_cmpgt_pd</td>
<td>Greater Than</td>
<td>CMPLTPDr</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE2 Instruction</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>_mm_cmpge_pd</td>
<td>Greater Than or Equal</td>
<td>CMPLEPDr</td>
</tr>
<tr>
<td>_mm_cmpord_pd</td>
<td>Ordered</td>
<td>CMPORDPD</td>
</tr>
<tr>
<td>_mm_cmpunord_pd</td>
<td>Unordered</td>
<td>CMPUNORDPD</td>
</tr>
<tr>
<td>_mm_cmpneq_pd</td>
<td>Inequality</td>
<td>CMPNEQPD</td>
</tr>
<tr>
<td>_mm_cmpnltd_pd</td>
<td>Not Less Than</td>
<td>CMPNLTPD</td>
</tr>
<tr>
<td>_mm_cmpnltd_sd</td>
<td>Not Less Than or Equal</td>
<td>CMPNLTPD</td>
</tr>
<tr>
<td>_mm_cmpngt_pd</td>
<td>Not Greater Than</td>
<td>CMPNLTPDr</td>
</tr>
<tr>
<td>_mm_cmpnge_pd</td>
<td>Not Greater Than or Equal</td>
<td>CMPNLTPDr</td>
</tr>
<tr>
<td>_mm_cmpeq_sd</td>
<td>Equality</td>
<td>CMPEQSD</td>
</tr>
<tr>
<td>_mm_cmplt_sd</td>
<td>Less Than</td>
<td>CMPLTSD</td>
</tr>
<tr>
<td>_mm_cmple_sd</td>
<td>Less Than or Equal</td>
<td>CMPELSD</td>
</tr>
<tr>
<td>_mm_cmplt_sd</td>
<td>Greater Than</td>
<td>CMPLTSDr</td>
</tr>
<tr>
<td>_mm_cmpte_sd</td>
<td>Greater Than or Equal</td>
<td>CMPLTSDr</td>
</tr>
<tr>
<td>_mm_cmpeord_sd</td>
<td>Ordered</td>
<td>CMPORDS</td>
</tr>
<tr>
<td>_mm_cmpeunord_sd</td>
<td>Unordered</td>
<td>CMPUNORDSD</td>
</tr>
<tr>
<td>_mm_cmpeq_sd</td>
<td>Inequality</td>
<td>CMPNEQSD</td>
</tr>
<tr>
<td>_mm_cmplt_sd</td>
<td>Not Less Than</td>
<td>CMPNLTS</td>
</tr>
<tr>
<td>_mm_cmplt_sd</td>
<td>Not Less Than or Equal</td>
<td>CMPNLTS</td>
</tr>
<tr>
<td>_mm_cmpltd_sd</td>
<td>Not Greater Than</td>
<td>CMPNLTSDr</td>
</tr>
<tr>
<td>_mm_cmpltd_sd</td>
<td>Not Greater Than or Equal</td>
<td>CMPNLTSDr</td>
</tr>
<tr>
<td>_mm_comieq_sd</td>
<td>Equality</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comilt_sd</td>
<td>Less Than</td>
<td>COMISD</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE2 Instruction</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>_mm_comile_sd</td>
<td>Less Than or Equal</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comigt_sd</td>
<td>Greater Than</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comige_sd</td>
<td>Greater Than or Equal</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_comineq_sd</td>
<td>Not Equal</td>
<td>COMISD</td>
</tr>
<tr>
<td>_mm_ucomieq_sd</td>
<td>Equality</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomilt_sd</td>
<td>Less Than</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomile_sd</td>
<td>Less Than or Equal</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomigt_sd</td>
<td>Greater Than</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomige_sd</td>
<td>Greater Than or Equal</td>
<td>UCOMISD</td>
</tr>
<tr>
<td>_mm_ucomineq_sd</td>
<td>Not Equal</td>
<td>UCOMISD</td>
</tr>
</tbody>
</table>

__m128d  _mm_cmpeq_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for equality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 == b0) ?</td>
<td>(a1 == b1) ?</td>
</tr>
<tr>
<td>0xfffffffffffffff : 0x0</td>
<td>0xfffffffffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128d  _mm_cmplt_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a less than b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 &lt; b0) ?</td>
<td>(a1 &lt; b1) ?</td>
</tr>
<tr>
<td>0xfffffffffffffff : 0x0</td>
<td>0xfffffffffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128d  _mm_cmple_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a less than or equal to b.
__m128d __m128d

Compares the two DP FP values of \text{a} and \text{b} for \text{a} greater than \text{b}.

__m128d __m128d

Compares the two DP FP values of \text{a} and \text{b} for \text{a} greater than or equal to \text{b}.

__m128d __m128d

Compares the two DP FP values of \text{a} and \text{b} for ordered.

__m128d __m128d

Compares the two DP FP values of \text{a} and \text{b} for unordered.
__m128d _mm_cmpneq_pd ( __m128d a, __m128d b)

Compares the two DP FP values of a and b for inequality.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 != b0) ?</td>
<td>(a1 != b1) ?</td>
</tr>
<tr>
<td>0xfffffffffffffffffffffff : 0x0</td>
<td>0xfffffffffffffffffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128d _mm_cmpnlt_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a not less than b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt; b0) ?</td>
<td>!(a1 &lt; b1) ?</td>
</tr>
<tr>
<td>0xfffffffffffffffffffffff : 0x0</td>
<td>0xfffffffffffffffffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128d _mm_cmpnle_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a not less than or equal to b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &lt;= b0) ?</td>
<td>!(a1 &lt;= b1) ?</td>
</tr>
<tr>
<td>0xfffffffffffffffffffffff : 0x0</td>
<td>0xfffffffffffffffffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128d _mm_cmpngt_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a not greater than b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>!(a0 &gt; b0) ?</td>
<td>!(a1 &gt; b1) ?</td>
</tr>
<tr>
<td>0xfffffffffffffffffffffff : 0x0</td>
<td>0xfffffffffffffffffffffff : 0x0</td>
</tr>
</tbody>
</table>

__m128d _mm_cmpnge_pd(__m128d a, __m128d b)

Compares the two DP FP values of a and b for a not greater than or equal to b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
</table>
__m128d _mm_cmpeq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of `a` and `b` for equality. The upper DP FP value is passed through from `a`.

__m128d _mm_cmplt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of `a` and `b` for `a` less than `b`. The upper DP FP value is passed through from `a`.

__m128d _mm_cmple_sd(__m128d a, __m128d b)

Compares the lower DP FP value of `a` and `b` for `a` less than or equal to `b`. The upper DP FP value is passed through from `a`.

__m128d _mm_cmpgt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of `a` and `b` for `a` greater than `b`. The upper DP FP value is passed through from `a`.
(a0 > b0) ? 0xffffffffffffffffffffff : 0x0

__m128d _mm_cmpge_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \(a\) and \(b\) for \(a\) greater than or equal to \(b\). The upper DP FP value is passed through from \(a\).

(a0 >= b0) ? 0xffffffffffffffffffffff : 0x0

__m128d _mm_cmpord_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \(a\) and \(b\) for ordered. The upper DP FP value is passed through from \(a\).

(a0 ord b0) ? 0xffffffffffffffffffffff : 0x0

__m128d _mm_cmpunord_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \(a\) and \(b\) for unordered. The upper DP FP value is passed through from \(a\).

(a0 unord b0) ? 0xffffffffffffffffffffff : 0x0

__m128d _mm_cmpneq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of \(a\) and \(b\) for inequality. The upper DP FP value is passed through from \(a\).
### __m128d _mm_cmpnlt_sd(__m128d a, __m128d b)___

Compares the lower DP FP value of `a` and `b` for `a` not less than `b`. The upper DP FP value is passed through from `a`.

```c
__m128d _mm_cmpnlt_sd(__m128d a, __m128d b)
```

### __m128d _mm_cmpnle_sd(__m128d a, __m128d b)___

Compares the lower DP FP value of `a` and `b` for `a` not less than or equal to `b`. The upper DP FP value is passed through from `a`.

```c
__m128d _mm_cmpnle_sd(__m128d a, __m128d b)
```

### __m128d _mmcmpngt_sd(__m128d a, __m128d b)___

Compares the lower DP FP value of `a` and `b` for `a` not greater than `b`. The upper DP FP value is passed through from `a`.

```c
__m128d _mmcmpngt_sd(__m128d a, __m128d b)
```

### __m128d _mmcmpnge_sd(__m128d a, __m128d b)___

Compares the lower DP FP value of `a` and `b` for `a` not greater than or equal to `b`. The upper DP FP value is passed through from `a`.

```c
__m128d _mmcmpnge_sd(__m128d a, __m128d b)
```
int _mm_comieq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a equal to b. If a and b are equal, 1 is returned. Otherwise 0 is returned.

int _mm_comilt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a less than b. If a is less than b, 1 is returned. Otherwise 0 is returned.

int _mm_comile_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a less than or equal to b. If a is less than or equal to b, 1 is returned. Otherwise 0 is returned.

int _mm_comigt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a greater than b. If a is greater than b are equal, 1 is returned. Otherwise 0 is returned.
int _mm_comige_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a greater than or equal to b. If a is greater than or equal to b, 1 is returned. Otherwise 0 is returned.

R
(a0 >= b0) ? 0x1 : 0x0

int _mm_comineq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a not equal to b. If a and b are not equal, 1 is returned. Otherwise 0 is returned.

R
(a0 != b0) ? 0x1 : 0x0

int _mm_ucomieq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a equal to b. If a and b are equal, 1 is returned. Otherwise 0 is returned.

R
(a0 == b0) ? 0x1 : 0x0

int _mm_ucomilt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a less than b. If a is less than b, 1 is returned. Otherwise 0 is returned.

R
(a0 < b0) ? 0x1 : 0x0

int _mm_ucomile_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a less than or equal to b. If a is less than or equal to b, 1 is returned. Otherwise 0 is returned.
(a0 <= b0) ? 0x1 : 0x0

int _mm_ucomigt_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a greater than b. If a is greater than b are equal, 1 is returned. Otherwise 0 is returned.

(a0 > b0) ? 0x1 : 0x0

int _mm_ucomige_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a greater than or equal to b. If a is greater than or equal to b, 1 is returned. Otherwise 0 is returned.

(a0 >= b0) ? 0x1 : 0x0

int _mm_ucomineq_sd(__m128d a, __m128d b)

Compares the lower DP FP value of a and b for a not equal to b. If a and b are not equal, 1 is returned. Otherwise 0 is returned.

(a0 != b0) ? 0x1 : 0x0

Floating-point Conversion Operations for Streaming SIMD Extensions 2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for floating-point conversion operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the emmintrin.h header file.

Each conversion intrinsic takes one data type and performs a conversion to a different type. Some conversions, such as those performed by the
_mm_cvtpd_ps intrinsic, result in a loss of precision. The rounding mode used in such cases is determined by the value in the MXCSR register. The default rounding mode is round-to-nearest.

**Note**
The rounding mode used by the C and C++ languages when performing a type conversion is to truncate. The _mm_cvtpd_epi32 and _mm_cvttsd_si32 intrinsics use the truncate rounding mode regardless of the mode specified by the MXCSR register. The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by R, R0, R1, R2, and R3, where each represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cvtpd_ps</td>
<td>Convert DP FP to SP FP</td>
<td>CVTPD2PS</td>
</tr>
<tr>
<td>_mm_cvtps_pd</td>
<td>Convert from SP FP to DP FP</td>
<td>CVTPS2PD</td>
</tr>
<tr>
<td>_mm_cvtepi32_pd</td>
<td>Convert lower integer values to DP FP</td>
<td>CVTDQ2PD</td>
</tr>
<tr>
<td>_mm_cvtpd_epi32</td>
<td>Convert DP FP values to integer values</td>
<td>CVTPD2DQ</td>
</tr>
<tr>
<td>_mm_cvtsd_si32</td>
<td>Convert lower DP FP value to integer value</td>
<td>CVTSD2SI</td>
</tr>
<tr>
<td>_mm_cvtsd_ss</td>
<td>Convert lower DP FP value to SP FP</td>
<td>CVTSD2SS</td>
</tr>
<tr>
<td>_mm_cvtsi32_sd</td>
<td>Convert signed integer value to DP FP</td>
<td>CVTSI2SD</td>
</tr>
<tr>
<td>_mm_cvtss_sd</td>
<td>Convert lower SP FP value to DP FP</td>
<td>CVTSS2SD</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE2 Instruction</td>
</tr>
<tr>
<td>------------------------</td>
<td>------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_cvttpd_epi32</td>
<td>Convert DP FP values to signed integers</td>
<td>CVTTPD2DQ</td>
</tr>
<tr>
<td>_mm_cvttsd_si32</td>
<td>Convert lower DP FP to signed integer</td>
<td>CVTTSD2SI</td>
</tr>
<tr>
<td>_mm_cvtpd_pi32</td>
<td>Convert two DP FP values to signed integer values</td>
<td>CVTPD2PI</td>
</tr>
<tr>
<td>_mm_cvtpd_pi32</td>
<td>Convert two DP FP values to signed integer values using truncate</td>
<td>CVTPD2PI</td>
</tr>
<tr>
<td>_mm_cvtpi32_pd</td>
<td>Convert two signed integer values to DP FP</td>
<td>CVTP2PD</td>
</tr>
<tr>
<td>_mm_cvtsd_f64</td>
<td>Extract DP FP value from first vector element</td>
<td>None</td>
</tr>
</tbody>
</table>

__m128 _mm_cvtpd_ps(__m128d a)  
Converting the two DP FP values of a to SP FP values.  

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(float) a0</td>
<td>(float) a1</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

__m128d _mm_cvtps_pd(__m128 a)  
Converting the lower two SP FP values of a to DP FP values.  

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>(double) a0</td>
<td>(double) a1</td>
</tr>
</tbody>
</table>

__m128d _mm_cvtepi32_pd(__m128i a)  
Converting the lower two signed 32-bit integer values of a to DP FP values.
R0          R1
(double) a0 (double) a1

__m128i __mm_cvtpd_epi32(__m128d a)

Converts the two DP FP values of a to 32-bit signed integer values.

R0          R1          R2          R3
(int) a0 (int) a1 0x0 0x0

int __mm_cvtsd_si32(__m128d a)

Converts the lower DP FP value of a to a 32-bit signed integer value.

R
(int) a0

__m128 __mm_cvtsd_ss(__m128 a, __m128d b)

Converts the lower DP FP value of b to an SP FP value. The upper SP FP values in a are passed through.

R0          R1          R2          R3
(float) b0 a1 a2 a3

__m128d __mm_cvtsi32_sd(__m128d a, int b)

Converts the signed integer value in b to a DP FP value. The upper DP FP value in a is passed through.

R0          R1
(double) b a1

__m128d __mm_cvtss_sd(__m128d a, __m128 b)

Converts the lower SP FP value of b to a DP FP value. The upper value DP FP value in a is passed through.
Converts the two DP FP values of a to 32-bit signed integers using truncate.

Converts the lower DP FP value of a to a 32-bit signed integer using truncate.

Converts the two DP FP values of a to 32-bit signed integer values.

Converts the two DP FP values of a to 32-bit signed integer values using truncate.

Converts the two 32-bit signed integer values of a to DP FP values.
R0 R1
(doble)a0 (double)a1

double _mm_cvtsd_f64(__m128d a)

This intrinsic extracts a double precision floating point value from the first vector element of an __m128d. It does so in the most efficient manner possible in the context used. This intrinsic does not map to any specific SSE2 instruction.

Floating-point Load Operations for Streaming SIMD Extensions 2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for floating-point load operations are listed in this topic. The prototypes for SSE2 intrinsics are in the emmintrin.h header file.

The load and set operations are similar in that both initialize __m128d data. However, the set operations take a double argument and are intended for initialization with constants, while the load operations take a double pointer argument and are intended to mimic the instructions for loading data from memory.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by R0 and R1, where R0 and R1 each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_load_pd</td>
<td>Loads two DP FP values</td>
<td>MOVAPD</td>
</tr>
<tr>
<td>_mm_load1_pd</td>
<td>Loads a single DP FP value, copying to both elements</td>
<td>MOVSD + shuffling</td>
</tr>
<tr>
<td>_mm_loadr_pd</td>
<td>Loads two DP FP values</td>
<td>MOVAPD + shuffling</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE2 Instruction</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_loadu_pd</td>
<td>Loads two DP FP values in reverse order</td>
<td>MOVUPD</td>
</tr>
<tr>
<td>_mm_load_sd</td>
<td>Loads a DP FP value, sets upper DP FP to zero</td>
<td>MOVSD</td>
</tr>
<tr>
<td>_mm_loadh_pd</td>
<td>Loads a DP FP value as the upper DP FP value of the result</td>
<td>MOVHPD</td>
</tr>
<tr>
<td>_mm_loadl_pd</td>
<td>Loads a DP FP value as the lower DP FP value of the result</td>
<td>MOVLPD</td>
</tr>
</tbody>
</table>

__m128d _mm_load_pd(double const*dp)

Loads two DP FP values. The address p must be 16-byte aligned.

```
R0  R1
p[0] p[1]
```

__m128d _mm_load1_pd(double const*dp)

 Loads a single DP FP value, copying to both elements. The address p need not be 16-byte aligned.

```
R0  R1
*p  *p
```

__m128d _mm_loadr_pd(double const*dp)

Loads two DP FP values in reverse order. The address p must be 16-byte aligned.
__m128d _mm_loadu_pd(double const*dp)

Loads two DP FP values. The address p need not be 16-byte aligned.

__m128d _mm_load_sd(double const*dp)

Loads a DP FP value. The upper DP FP is set to zero. The address p need not be 16-byte aligned.

__m128d _mm_loadh_pd(__m128d a, double const*dp)

 Loads a DP FP value as the upper DP FP value of the result. The lower DP FP value is passed through from a. The address p need not be 16-byte aligned.

__m128d _mm_loadl_pd(__m128d a, double const*dp)

Loads a DP FP value as the lower DP FP value of the result. The upper DP FP value is passed through from a. The address p need not be 16-byte aligned.
The Streaming SIMD Extensions 2 (SSE2) intrinsics for floating-point set operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The load and set operations are similar in that both initialize `__m128d` data. However, the set operations take a double argument and are intended for initialization with constants, while the load operations take a double pointer argument and are intended to mimic the instructions for loading data from memory.

Some of the these intrinsics are composite intrinsics because they require more than one instruction to implement them.

The results of each intrinsic operation are placed in a register. The information about what is placed in each register appears in the tables below, in the detailed explanation for each intrinsic. For each intrinsic, the resulting register is represented by `R0` and `R1`, where `R0` and `R1` each represent one piece of the result register.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_mm_set_sd</code></td>
<td>Sets lower DP FP value to w and upper to zero</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_set1_pd</code></td>
<td>Sets two DP FP values to w</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_set_pd</code></td>
<td>Sets lower DP FP to x and upper to w</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_setr_pd</code></td>
<td>Sets lower DP FP to w and upper to x</td>
<td>Composite</td>
</tr>
<tr>
<td><code>_mm_setzero_pd</code></td>
<td>Sets two DP FP values to zero</td>
<td>XORPD</td>
</tr>
<tr>
<td><code>_mm_move_sd</code></td>
<td>Sets lower DP FP value to the lower DP FP value of</td>
<td>MOVSD</td>
</tr>
</tbody>
</table>
Intrinsic Name | Operation | Corresponding SSE2 Instruction
--- | --- | ---
__m128d _mm_set_sd(double w) | Sets the lower DP FP value to \( w \) and sets the upper DP FP value to zero. |
R0 R1
\[
\begin{array}{c}
w \\
0.0
\end{array}
\]

__m128d _mm_set1_pd(double w) | Sets the 2 DP FP values to \( w \). |
R0 R1
\[
\begin{array}{c}
w \\
w
\end{array}
\]

__m128d _mm_set_pd(double w, double x) | Sets the lower DP FP value to \( x \) and sets the upper DP FP value to \( w \). |
R0 R1
\[
\begin{array}{c}
x \\
w
\end{array}
\]

__m128d _mm_setr_pd(double w, double x) | Sets the lower DP FP value to \( w \) and sets the upper DP FP value to \( x \). \( r0 := w \) \( r1 := x \) |
R0 R1
\[
\begin{array}{c}
w \\
x
\end{array}
\]

__m128d _mm_setzero_pd(void) | Sets the 2 DP FP values to zero. |
__m128d _mm_move_sd(__m128d a, __m128d b)

Sets the lower DP FP value to the lower DP FP value of b. The upper DP FP value is passed through from a.

Floating-point Store Operations for Streaming SIMD Extensions 2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for floating-point store operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The store operations assign the initialized data to the address.

The detailed description of each intrinsic contains a table detailing the returns. In these tables, dp[n] is an access to the n element of the result.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_stream_pd</td>
<td>Store</td>
<td>MOVNTPD</td>
</tr>
<tr>
<td>_mm_store_sd</td>
<td>Stores lower DP FP value</td>
<td>MOVSD</td>
</tr>
<tr>
<td></td>
<td>of a</td>
<td></td>
</tr>
<tr>
<td>_mm_store1_pd</td>
<td>Stores lower DP FP value</td>
<td>MOVAPD + shuffling</td>
</tr>
<tr>
<td></td>
<td>of a twice</td>
<td></td>
</tr>
<tr>
<td>_mm_store_pd</td>
<td>Stores two DP FP values</td>
<td>MOVAPD</td>
</tr>
<tr>
<td>_mm_storeu_pd</td>
<td>Stores two DP FP values</td>
<td>MOVUPD</td>
</tr>
<tr>
<td>_mm_storer_pd</td>
<td>Stores two DP FP values</td>
<td>MOVAPD + shuffling</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE2 Instruction</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>_mm_storeh_pd</td>
<td>Stores upper DP FP value of <code>a</code> in reverse order</td>
<td>MOVHPD</td>
</tr>
<tr>
<td>_mm_storel_pd</td>
<td>Stores lower DP FP value of <code>a</code></td>
<td>MOVLPD</td>
</tr>
</tbody>
</table>

```c
void _mm_store_sd(double *dp, __m128d a)
```

Stores the lower DP FP value of `a`. The address `dp` need not be 16-byte aligned.

```c
*dp
```

*a0

```c
void _mm_store1_pd(double *dp, __m128d a)
```

Stores the lower DP FP value of `a` twice. The address `dp` must be 16-byte aligned.

```c
dp[0] dp[1]
a0     a0
```

```c
void _mm_store_pd(double *dp, __m128d a)
```

Stores two DP FP values. The address `dp` must be 16-byte aligned.

```c
dp[0] dp[1]
a0     a1
```

```c
void _mm_storeu_pd(double *dp, __m128d a)
```

Stores two DP FP values. The address `dp` need not be 16-byte aligned.

```c
dp[0] dp[1]
```
void _mm_storer_pd(double *dp, __m128d a)
Stores two DP FP values in reverse order. The address dp must be 16-byte aligned.

void _mm_storeh_pd(double *dp, __m128d a)
Stores the upper DP FP value of a.

void _mm_storel_pd(double *dp, __m128d a)
Stores the lower DP FP value of a.

**Integer Intrinsics**

**Integer Arithmetic Operations for Streaming SIMD Extensions 2**

The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer arithmetic operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the _emmintrin.h_ header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed
explanation of each intrinsic. R, R0, R1...R15 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_add_epi8</td>
<td>Addition</td>
<td>PADDB</td>
</tr>
<tr>
<td>_mm_add_epi16</td>
<td>Addition</td>
<td>PADDW</td>
</tr>
<tr>
<td>_mm_add_epi32</td>
<td>Addition</td>
<td>PADDW</td>
</tr>
<tr>
<td>_mm_add_si64</td>
<td>Addition</td>
<td>PADDQ</td>
</tr>
<tr>
<td>_mm_add_epi64</td>
<td>Addition</td>
<td>PADDQ</td>
</tr>
<tr>
<td>_mm_adds_epi8</td>
<td>Addition</td>
<td>PADDQ</td>
</tr>
<tr>
<td>_mm_adds_epi16</td>
<td>Addition</td>
<td>PADDQ</td>
</tr>
<tr>
<td>_mm_adds_epu8</td>
<td>Addition</td>
<td>PADDUSB</td>
</tr>
<tr>
<td>_mm_adds_epu16</td>
<td>Addition</td>
<td>PADDUSW</td>
</tr>
<tr>
<td>_mm_avg_epi8</td>
<td>Computes Average</td>
<td>PAVGB</td>
</tr>
<tr>
<td>_mm_avg_epi16</td>
<td>Computes Average</td>
<td>PAVGW</td>
</tr>
<tr>
<td>_mm_madd_epi16</td>
<td>Multiplication and Addition</td>
<td>PMADDWD</td>
</tr>
<tr>
<td>_mm_max_epi16</td>
<td>Computes Maxima</td>
<td>PMAXSW</td>
</tr>
<tr>
<td>_mm_max_epi8</td>
<td>Computes Maxima</td>
<td>PMAXUB</td>
</tr>
<tr>
<td>_mm_min_epi16</td>
<td>Computes Minima</td>
<td>PMINSW</td>
</tr>
<tr>
<td>_mm_min_epi8</td>
<td>Computes Minima</td>
<td>PMINUB</td>
</tr>
<tr>
<td>_mm_mulhi_epi16</td>
<td>Multiplication</td>
<td>PMULHW</td>
</tr>
<tr>
<td>_mm_mulhi_epi16</td>
<td>Multiplication</td>
<td>PMULHUW</td>
</tr>
<tr>
<td>_mm_mullo_epi16</td>
<td>Multiplication</td>
<td>PMULLW</td>
</tr>
<tr>
<td>_mm_mul_su32</td>
<td>Multiplication</td>
<td>PMULUDQ</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding SSE2 Instruction</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>_mm_mul_epu32</td>
<td>Multiplication</td>
<td>PMULUDQ</td>
</tr>
<tr>
<td>_mm_sad_epu8</td>
<td>Computes</td>
<td>PSADBW</td>
</tr>
<tr>
<td>_mm_sub_epi8</td>
<td>Subtraction</td>
<td>PSUBB</td>
</tr>
<tr>
<td>_mm_sub_epi16</td>
<td>Subtraction</td>
<td>PSUBW</td>
</tr>
<tr>
<td>_mm_sub_epi32</td>
<td>Subtraction</td>
<td>PSUBD</td>
</tr>
<tr>
<td>_mm_sub_si64</td>
<td>Subtraction</td>
<td>PSUBQ</td>
</tr>
<tr>
<td>_mm_sub_epi64</td>
<td>Subtraction</td>
<td>PSUBQ</td>
</tr>
<tr>
<td>_mm_subs_epi8</td>
<td>Subtraction</td>
<td>PSUBSB</td>
</tr>
<tr>
<td>_mm_subs_epi16</td>
<td>Subtraction</td>
<td>PSUBSW</td>
</tr>
<tr>
<td>_mm_subs_epu8</td>
<td>Subtraction</td>
<td>PSUBUSB</td>
</tr>
<tr>
<td>_mm_subs_epu16</td>
<td>Subtraction</td>
<td>PSUBUSW</td>
</tr>
</tbody>
</table>

__m128i _mm_add_epi8(__m128i a, __m128i b)  
Adds the 16 signed or unsigned 8-bit integers in a to the 16 signed or unsigned 8-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0 + b0</td>
<td>a1 + b1;</td>
<td>...</td>
<td>a15 + b15</td>
</tr>
</tbody>
</table>

__m128i _mm_add_epi16(__m128i a, __m128i b)  
Adds the 8 signed or unsigned 16-bit integers in a to the 8 signed or unsigned 16-bit integers in b.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
</table>
__m128i _mm_add_epi32(__m128i a, __m128i b)

Adds the 4 signed or unsigned 32-bit integers in a to the 4 signed or unsigned 32-bit integers in b.

__m64 _mm_add_si64(__m64 a, __m64 b)

Adds the signed or unsigned 64-bit integer a to the signed or unsigned 64-bit integer b.

__m128i _mm_add_epi64(__m128i a, __m128i b)

Adds the 2 signed or unsigned 64-bit integers in a to the 2 signed or unsigned 64-bit integers in b.

__m128i _mm_adds_epi8(__m128i a, __m128i b)

Adds the 16 signed 8-bit integers in a to the 16 signed 8-bit integers in b using saturating arithmetic.
__m128i _mm_adds_epi16(__m128i a, __m128i b)

Adds the 8 signed 16-bit integers in `a` to the 8 signed 16-bit integers in `b` using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>SignedSaturate</td>
<td>SignedSaturate</td>
<td>...</td>
<td>SignedSaturate</td>
</tr>
<tr>
<td>(a0 + b0)</td>
<td>(a1 + b1)</td>
<td></td>
<td>(a7 + b7)</td>
</tr>
</tbody>
</table>

__m128i _mm_adds_epu8(__m128i a, __m128i b)

Adds the 16 unsigned 8-bit integers in `a` to the 16 unsigned 8-bit integers in `b` using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnsignedSaturate</td>
<td>UnsignedSaturate</td>
<td>...</td>
<td>UnsignedSaturate</td>
</tr>
<tr>
<td>(a0 + b0)</td>
<td>(a1 + b1)</td>
<td></td>
<td>(a15 + b15)</td>
</tr>
</tbody>
</table>

__m128i _mm_adds_epu16(__m128i a, __m128i b)

Adds the 8 unsigned 16-bit integers in `a` to the 8 unsigned 16-bit integers in `b` using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnsignedSaturate</td>
<td>UnsignedSaturate</td>
<td>...</td>
<td>UnsignedSaturate</td>
</tr>
<tr>
<td>(a0 + b0)</td>
<td>(a1 + b1)</td>
<td></td>
<td>(a7 + b7)</td>
</tr>
</tbody>
</table>

__m128i _mm_avg_epu8(__m128i a, __m128i b)

Computes the average of the 16 unsigned 8-bit integers in `a` and the 16 unsigned 8-bit integers in `b` and rounds.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a0 + b0) / 2</td>
<td>(a1 + b1) / 2</td>
<td>...</td>
<td>(a15 + b15) / 2</td>
</tr>
</tbody>
</table>
__m128i _mm_avg_epu16(__m128i a, __m128i b)

Computes the average of the 8 unsigned 16-bit integers in \(a\) and the 8 unsigned 16-bit integers in \(b\) and rounds.

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \ldots & \text{R7} \\
(a0 + b0) / 2 & (a1 + b1) / 2 & \ldots & (a7 + b7) / 2
\end{array}
\]

__m128i _mm_madd_epi16(__m128i a, __m128i b)

Multiplies the 8 signed 16-bit integers from \(a\) by the 8 signed 16-bit integers from \(b\). Adds the signed 32-bit integer results pairwise and packs the 4 signed 32-bit integer results.

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \text{R2} & \text{R3} \\
(a0 \times b0) + (a2 \times b2) + (a4 \times b4) + (a6 \times b6) + (a1 \times b1) + (a3 \times b3) + (a5 \times b5) + (a7 \times b7)
\end{array}
\]

__m128i _mm_max_epi16(__m128i a, __m128i b)

Computes the pairwise maxima of the 8 signed 16-bit integers from \(a\) and the 8 signed 16-bit integers from \(b\).

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \ldots & \text{R7} \\
\max(a0, b0) \max(a1, b1) \ldots & \max(a7, b7)
\end{array}
\]

__m128i _mm_max_epu8(__m128i a, __m128i b)

Computes the pairwise maxima of the 16 unsigned 8-bit integers from \(a\) and the 16 unsigned 8-bit integers from \(b\).

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \ldots & \text{R15} \\
\max(a0, b0) \max(a1, b1) \ldots & \max(a15, b15)
\end{array}
\]

__m128i _mm_min_epi16(__m128i a, __m128i b)
Computes the pairwise minima of the 8 signed 16-bit integers from $a$ and the 8 signed 16-bit integers from $b$.

\[
\begin{array}{cccc}
R0 & R1 & \cdots & R7 \\
\min(a_0, b_0) & \min(a_1, b_1) & \ldots & \min(a_7, b_7) \\
\end{array}
\]

__m128i _mm_min_epu8(__m128i a, __m128i b)

Computes the pairwise minima of the 16 unsigned 8-bit integers from $a$ and the 16 unsigned 8-bit integers from $b$.

\[
\begin{array}{cccc}
R0 & R1 & \cdots & R15 \\
\min(a_0, b_0) & \min(a_1, b_1) & \ldots & \min(a_{15}, b_{15}) \\
\end{array}
\]

__m128i _mm_mulhi_epi16(__m128i a, __m128i b)

Multiplies the 8 signed 16-bit integers from $a$ by the 8 signed 16-bit integers from $b$. Packs the upper 16-bits of the 8 signed 32-bit results.

\[
\begin{array}{cccc}
R0 & R1 & \cdots & R7 \\
(a_0 \times b_0)[31:16] & (a_1 \times b_1)[31:16] & \ldots & (a_7 \times b_7)[31:16] \\
\end{array}
\]

__m128i _mm_mulhi_epu16(__m128i a, __m128i b)

Multiplies the 8 unsigned 16-bit integers from $a$ by the 8 unsigned 16-bit integers from $b$. Packs the upper 16-bits of the 8 unsigned 32-bit results.

\[
\begin{array}{cccc}
R0 & R1 & \cdots & R7 \\
(a_0 \times b_0)[31:16] & (a_1 \times b_1)[31:16] & \ldots & (a_7 \times b_7)[31:16] \\
\end{array}
\]

__m128i _mm_mullo_epi16(__m128i a, __m128i b)
Multiplies the 8 signed or unsigned 16-bit integers from \( a \) by the 8 signed or unsigned 16-bit integers from \( b \). Packs the lower 16-bits of the 8 signed or unsigned 32-bit results.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
(a0 \times b0)[15:0] & (a1 \times b1)[15:0] & \ldots & (a7 \times b7)[15:0] \\
\end{array}
\]

\[
__m64\ _mm\_mul\_su32(__m64\ a,\ __m64\ b)
\]

Multiplies the lower 32-bit integer from \( a \) by the lower 32-bit integer from \( b \), and returns the 64-bit integer result.

\[
\begin{array}{c}
R0 \\
a0 \times b0 \\
\end{array}
\]

\[
__m128i\ _mm\_mul\_epu32(__m128i\ a,\ __m128i\ b)
\]

Multiplies 2 unsigned 32-bit integers from \( a \) by 2 unsigned 32-bit integers from \( b \). Packs the 2 unsigned 64-bit integer results.

\[
\begin{array}{cc}
R0 & R1 \\
a0 \times b0 & a2 \times b2 \\
\end{array}
\]

\[
__m128i\ _mm\_sad\_epu8(__m128i\ a,\ __m128i\ b)
\]

Computes the absolute difference of the 16 unsigned 8-bit integers from \( a \) and the 16 unsigned 8-bit integers from \( b \). Sums the upper 8 differences and lower 8 differences, and packs the resulting 2 unsigned 16-bit integers into the upper and lower 64-bit elements.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
abs(a0 - b0) + 0x0 & 0x0 & 0x0 \\
abs(a1 - b1) \\
\end{array}
\]
Subtracts the 16 signed or unsigned 8-bit integers of \( b \) from the 16 signed or unsigned 8-bit integers of \( a \).

Subtracts the 8 signed or unsigned 16-bit integers of \( b \) from the 8 signed or unsigned 16-bit integers of \( a \).

Subtracts the 4 signed or unsigned 32-bit integers of \( b \) from the 4 signed or unsigned 32-bit integers of \( a \).

Subtracts the signed or unsigned 64-bit integer \( b \) from the signed or unsigned 64-bit integer \( a \).
\_\_m128i \_\_mm\_sub\_epi64(\_\_m128i a, \_\_m128i b)

Subtracts the 2 signed or unsigned 64-bit integers in \texttt{b} from the 2 signed or unsigned 64-bit integers in \texttt{a}.

\begin{tabular}{l l l}
R0 & R1 & \\ a0 - b0 & a1 - b1 \\
\end{tabular}

\_\_m128i \_\_mm\_subs\_epi8(\_\_m128i a, \_\_m128i b)

Subtracts the 16 signed 8-bit integers of \texttt{b} from the 16 signed 8-bit integers of \texttt{a} using saturating arithmetic.

\begin{tabular}{l l l l l}
R0 & R1 & ... & R15 \\
SignedSaturate & SignedSaturate & ... & SignedSaturate \\
(a0 - b0) & (a1 - b1) & & (a15 - b15) \\
\end{tabular}

\_\_m128i \_\_mm\_subs\_epi16(\_\_m128i a, \_\_m128i b)

Subtracts the 8 signed 16-bit integers of \texttt{b} from the 8 signed 16-bit integers of \texttt{a} using saturating arithmetic.

\begin{tabular}{l l l l l}
R0 & R1 & ... & R15 \\
SignedSaturate & SignedSaturate & ... & SignedSaturate \\
(a0 - b0) & (a1 - b1) & & (a7 - b7) \\
\end{tabular}

\_\_m128i \_\_mm\_subs\_epu8 (\_\_m128i a, \_\_m128i b)

Subtracts the 16 unsigned 8-bit integers of \texttt{b} from the 16 unsigned 8-bit integers of \texttt{a} using saturating arithmetic.

\begin{tabular}{l l l l l}
R0 & R1 & ... & R15 \\
UnsignedSaturate & UnsignedSaturate & ... & UnsignedSaturate \\
(a0 - b0) & (a1 - b1) & & (a15 - b15) \\
\end{tabular}
Subtracts the 8 unsigned 16-bit integers of b from the 8 unsigned 16-bit integers of a using saturating arithmetic.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>UnsignedSaturate</td>
<td>UnsignedSaturate</td>
<td>...</td>
<td>UnsignedSaturate</td>
</tr>
<tr>
<td>(a0 - b0)</td>
<td>(a1 - b1)</td>
<td></td>
<td>(a7 - b7)</td>
</tr>
</tbody>
</table>

**Integer Logical Operations for Streaming SIMD Extensions 2**

The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer logical operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the *emmintrin.h* header file.

The results of each intrinsic operation are placed in register R. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_and_si128</td>
<td>Computes AND</td>
<td>PAND</td>
</tr>
<tr>
<td>_mm_andnot_si128</td>
<td>Computes AND and NOT</td>
<td>PANDN</td>
</tr>
<tr>
<td>_mm_or_si128</td>
<td>Computes OR</td>
<td>POR</td>
</tr>
<tr>
<td>_mm_xor_si128</td>
<td>Computes XOR</td>
<td>PXOR</td>
</tr>
<tr>
<td>__m128i _mm_and_si128(__m128i a, __m128i b)</td>
<td>Computes the bitwise AND of the 128-bit value in a and the 128-bit value in b.</td>
<td></td>
</tr>
</tbody>
</table>

R0

a & b

__m128i _mm_andnot_si128(__m128i a, __m128i b)
Computes the bitwise AND of the 128-bit value in \( b \) and the bitwise NOT of the 128-bit value in \( a \).

\[
R0 = (~a) \& b
\]

\[\_\_m128i \_mm_or_si128(\_m128i a, \_m128i b)\]

Computes the bitwise OR of the 128-bit value in \( a \) and the 128-bit value in \( b \).

\[
R0 = a \mid b
\]

\[\_\_m128i \_mm_xor_si128(\_m128i a, \_m128i b)\]

Computes the bitwise XOR of the 128-bit value in \( a \) and the 128-bit value in \( b \).

\[
R0 = a ^ b
\]

### Integer Shift Operations for Streaming SIMD Extensions 2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer shift operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. \( R, R0, R1...R7 \) represent the registers in which results are placed.

#### Note

The **count** argument is one shift count that applies to all elements of the operand being shifted. It is not a vector shift count that shifts each element by a different amount.
### Intrinsic Operation

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Shift Type</th>
<th>Corresponding Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_slli_si128</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLDQ</td>
</tr>
<tr>
<td>_mm_slli_epi16</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLW</td>
</tr>
<tr>
<td>_mm_sll_epi16</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLW</td>
</tr>
<tr>
<td>_mm_slli_epi32</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLD</td>
</tr>
<tr>
<td>_mm_sll_epi32</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLD</td>
</tr>
<tr>
<td>_mm_slli_epi64</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLQ</td>
</tr>
<tr>
<td>_mm_sll_epi64</td>
<td>Shift left</td>
<td>Logical</td>
<td>PSLLQ</td>
</tr>
<tr>
<td>_mm_srai_epi16</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAW</td>
</tr>
<tr>
<td>_mm_sra_epi16</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAW</td>
</tr>
<tr>
<td>_mm_srai_epi32</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAD</td>
</tr>
<tr>
<td>_mm_sra_epi32</td>
<td>Shift right</td>
<td>Arithmetic</td>
<td>PSRAD</td>
</tr>
<tr>
<td>_mm_srli_si128</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLDQ</td>
</tr>
<tr>
<td>_mm_srli_epi16</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLW</td>
</tr>
<tr>
<td>_mm_srl_epi16</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLW</td>
</tr>
<tr>
<td>_mm_srli_epi32</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLD</td>
</tr>
<tr>
<td>_mm_srl_epi32</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLD</td>
</tr>
<tr>
<td>_mm_srli_epi64</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLQ</td>
</tr>
<tr>
<td>_mm_srl_epi64</td>
<td>Shift right</td>
<td>Logical</td>
<td>PSRLQ</td>
</tr>
</tbody>
</table>

__m128i _mm_slli_si128(__m128i a, int imm)

Shifts the 128-bit value in a left by imm bytes while shifting in zeros. imm must be an immediate.

a << (imm * 8)

__m128i _mm_slli_epi16(__m128i a, int count)

Shifts the 8 signed or unsigned 16-bit integers in `a` left by `count` bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\end{array}
\]

a0 << count a1 << count ... a7 << count

__m128i _mm_slli_epi16(__m128i a, __m128i count)

Shifts the 8 signed or unsigned 16-bit integers in `a` left by `count` bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\end{array}
\]

a0 << count a1 << count ... a7 << count

__m128i _mm_slli_epi32(__m128i a, int count)

Shifts the 4 signed or unsigned 32-bit integers in `a` left by `count` bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\end{array}
\]

a0 << count a1 << count a2 << count a3 << count

__m128i _mm_slli_epi32(__m128i a, __m128i count)

Shifts the 4 signed or unsigned 32-bit integers in `a` left by `count` bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\end{array}
\]

a0 << count a1 << count a2 << count a3 << count

__m128i _mm_slli_epi64(__m128i a, int count)
Shifts the 2 signed or unsigned 64-bit integers in a left by `count` bits while shifting in zeros.

\[
\begin{align*}
R0 & \quad R1 \\
a0 & \ll \text{count} \\
a1 & \ll \text{count}
\end{align*}
\]

__m128i _mm_sll_epi64(__m128i a, __m128i count)

Shifts the 2 signed or unsigned 64-bit integers in a left by `count` bits while shifting in zeros.

\[
\begin{align*}
R0 & \quad R1 \\
a0 & \ll \text{count} \\
a1 & \ll \text{count}
\end{align*}
\]

__m128i _mm_srai_epi16(__m128i a, int count)

Shifts the 8 signed 16-bit integers in a right by `count` bits while shifting in the sign bit.

\[
\begin{align*}
R0 & \quad R1 & \ldots & \quad R7 \\
a0 & \gg \text{count} \\
a1 & \gg \text{count} \\
\ldots & \gg \text{count} \\
a7 & \gg \text{count}
\end{align*}
\]

__m128i _mm_sra_epi16(__m128i a, __m128i count)

Shifts the 8 signed 16-bit integers in a right by `count` bits while shifting in the sign bit.

\[
\begin{align*}
R0 & \quad R1 & \ldots & \quad R7 \\
a0 & \gg \text{count} \\
a1 & \gg \text{count} \\
\ldots & \gg \text{count} \\
a7 & \gg \text{count}
\end{align*}
\]

__m128i _mm_srai_epi32(__m128i a, int count)

Shifts the 4 signed 32-bit integers in a right by `count` bits while shifting in the sign bit.

\[
\begin{align*}
R0 & \quad R1 & \quad R2 & \quad R3
\end{align*}
\]
**__m128i _mm_sra_epi32(__m128i a, __m128i count)***

Shifts the 4 signed 32-bit integers in a right by count bits while shifting in the sign bit.

**__m128i _mm_srl_epi32(__m128i a, __m128i count)***

Shifts the 8 signed or unsigned 16-bit integers in a right by count bits while shifting in zeros.

**__m128i _mm_srl_epi32(__m128i a, __m128i count)***

Shifts the 128-bit value in a right by imm bytes while shifting in zeros. imm must be an immediate.

**srl(a, imm*8)**

**__m128i _mm_srl_epi32(__m128i a, __m128i count)***

Shifts the 8 signed or unsigned 16-bit integers in a right by count bits while shifting in zeros.
Shifts the 4 signed or unsigned 32-bit integers in a right by count bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
srl(a0, count) & srl(a1, count) & srl(a2, count) & srl(a3, count) \\
\end{array}
\]

__m128i __mm_srl_epi32(__m128i a, __m128i count)

Shifts the 4 signed or unsigned 32-bit integers in a right by count bits while shifting in zeros.

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
srl(a0, count) & srl(a1, count) & srl(a2, count) & srl(a3, count) \\
\end{array}
\]

__m128i __mm_srl_epi64(__m128i a, int count)

Shifts the 2 signed or unsigned 64-bit integers in a right by count bits while shifting in zeros.

\[
\begin{array}{cc}
R0 & R1 \\
srl(a0, count) & srl(a1, count) \\
\end{array}
\]

__m128i __mm_srl_epi64(__m128i a, __m128i count)

Shifts the 2 signed or unsigned 64-bit integers in a right by count bits while shifting in zeros.

\[
\begin{array}{cc}
R0 & R1 \\
srl(a0, count) & srl(a1, count) \\
\end{array}
\]

**Integer Comparison Operations for Streaming SIMD Extensions 2**

The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer comparison operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.
The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. \( R, R_0, R_1...R_{15} \) represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpeq_epi8</td>
<td>Equality</td>
<td>PCMPEQB</td>
</tr>
<tr>
<td>_mm_cmpeq_epi16</td>
<td>Equality</td>
<td>PCMPEQW</td>
</tr>
<tr>
<td>_mm_cmpeq_epi32</td>
<td>Equality</td>
<td>PCMPEQD</td>
</tr>
<tr>
<td>_mm_cmpgt_epi8</td>
<td>Greater Than</td>
<td>PCMPGTB</td>
</tr>
<tr>
<td>_mm_cmpgt_epi16</td>
<td>Greater Than</td>
<td>PCMPGTW</td>
</tr>
<tr>
<td>_mm_cmpgt_epi32</td>
<td>Greater Than</td>
<td>PCMPGTD</td>
</tr>
<tr>
<td>_mm_cmplt_epi8</td>
<td>Less Than</td>
<td>PCMPGTBr</td>
</tr>
<tr>
<td>_mm_cmplt_epi16</td>
<td>Less Than</td>
<td>PCMPGTWr</td>
</tr>
<tr>
<td>_mm_cmplt_epi32</td>
<td>Less Than</td>
<td>PCMPGTDr</td>
</tr>
</tbody>
</table>

\[
\text{__m128i } \_\text{mm_cmpeq_epi8}(\text{__m128i } a, \text{__m128i } b)
\]

Compares the 16 signed or unsigned 8-bit integers in \( a \) and the 16 signed or unsigned 8-bit integers in \( b \) for equality.

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \ldots & \text{R15} \\
(a_0 == b_0) \ ? & (a_1 == b_1) \ ? & \ldots & (a_{15} == b_{15}) \ ? \\
0xff : 0x0 & 0xff : 0x0 & \ldots & 0xff : 0x0 \\
\end{array}
\]

\[
\text{__m128i } \_\text{mm_cmpeq_epi16}(\text{__m128i } a, \text{__m128i } b)
\]

Compares the 8 signed or unsigned 16-bit integers in \( a \) and the 8 signed or unsigned 16-bit integers in \( b \) for equality.

\[
\begin{array}{cccc}
\text{R0} & \text{R1} & \ldots & \text{R7} \\
(a_0 == b_0) \ ? & (a_1 == b_1) \ ? & \ldots & (a_7 == b_7) \ ? \\
0xff : 0x0 & 0xff : 0x0 & \ldots & 0xff : 0x0 \\
\end{array}
\]
__m128i _mm_cmpeq_epi32(__m128i a, __m128i b)

Compares the 4 signed or unsigned 32-bit integers in `a` and the 4 signed or unsigned 32-bit integers in `b` for equality.

__m128i _mm_cmpgt_epi8(__m128i a, __m128i b)

Compares the 16 signed 8-bit integers in `a` and the 16 signed 8-bit integers in `b` for greater than.

__m128i _mm_cmpgt_epi16(__m128i a, __m128i b)

Compares the 8 signed 16-bit integers in `a` and the 8 signed 16-bit integers in `b` for greater than.
Compares the 4 signed 32-bit integers in \texttt{a} and the 4 signed 32-bit integers in \texttt{b} for greater than.

\begin{verbatim}
(a0 > b0) ? (a1 > b1) ? (a2 > b2) ? (a3 > b3) ?
0xffffffff : 0xffffffff : 0xffffffff : 0xffffffff :
0x0 : 0x0 : 0x0 : 0x0
\end{verbatim}

\texttt{__m128i \_mm\_cmplt\_epi8( \_m128i a, \_m128i b)}

Compares the 16 signed 8-bit integers in \texttt{a} and the 16 signed 8-bit integers in \texttt{b} for less than.

\begin{verbatim}
(a0 < b0) ? (a1 < b1) ? \ldots (a15 < b15) ?
0xff : 0x0 : \ldots : 0xff : 0x0
\end{verbatim}

\texttt{__m128i \_mm\_cmplt\_epi16( \_m128i a, \_m128i b)}

Compares the 8 signed 16-bit integers in \texttt{a} and the 8 signed 16-bit integers in \texttt{b} for less than.

\begin{verbatim}
(a0 < b0) ? (a1 < b1) ? \ldots (a7 < b7) ?
0xffffffff : 0x0 : 0xffffffff : 0x0 : \ldots : 0xffffffff : 0x0
\end{verbatim}

\texttt{__m128i \_mm\_cmplt\_epi32( \_m128i a, \_m128i b)}

Compares the 4 signed 32-bit integers in \texttt{a} and the 4 signed 32-bit integers in \texttt{b} for less than.

\begin{verbatim}
(a0 < b0) ? (a1 < b1) ? (a2 < b2) ? (a3 < b3) ?
0xffffffff : 0xffffffff : 0xffffffff : 0xffffffff :
0x0 : 0x0 : 0x0 : 0x0
\end{verbatim}
The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer conversion operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. \( R, R0, R1, R2 \) and \( R3 \) represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_mm_cvtsi64_sd</code></td>
<td>Convert and pass through</td>
<td>CVTSI2SD</td>
</tr>
<tr>
<td><code>_mm_cvtsd_si64</code></td>
<td>Convert according to rounding</td>
<td>CVTSD2SI</td>
</tr>
<tr>
<td><code>_mm_cvttsd_si64</code></td>
<td>Convert using truncation</td>
<td>CVTTSD2SI</td>
</tr>
<tr>
<td><code>_mm_cvtepi32_ps</code></td>
<td>Convert to SP FP</td>
<td>None</td>
</tr>
<tr>
<td><code>_mm_cvtps_epi32</code></td>
<td>Convert from SP FP</td>
<td>None</td>
</tr>
<tr>
<td><code>_mm_cvttemps_epi32</code></td>
<td>Convert from SP FP using truncate</td>
<td>None</td>
</tr>
</tbody>
</table>

\[
\text{__m128d \_mm_cvtsi64\_sd(__m128d a, __int64 b)}
\]

Converts the signed 64-bit integer value in \( b \) to a DP FP value. The upper DP FP value in \( a \) is passed through.

\[
\text{R0} \quad \text{R1}
\]

\[(\text{double})b \ a1\]

\[
\text{__int64 \_mm_cvtsd\_si64(__m128d a)}
\]
Converts the lower DP FP value of \( a \) to a 64-bit signed integer value according to the current rounding mode.

```c
(__int64) a0
__int64 _mm_cvttsd_si64(__m128d a)
```

Converts the lower DP FP value of \( a \) to a 64-bit signed integer value using truncation.

```c
(__int64) a0
__m128 _mm_cvtepi32_ps(__m128i a)
```

Converts the 4 signed 32-bit integer values of \( a \) to SP FP values.

```c
(float) a0 (float) a1 (float) a2 (float) a3
__m128i _mm_cvtps_epi32(__m128 a)
```

Converts the 4 SP FP values of \( a \) to signed 32-bit integer values.

```c
(int) a0 (int) a1 (int) a2 (int) a3
__m128i _mm_cvttps_epi32(__m128 a)
```

Converts the 4 SP FP values of \( a \) to signed 32 bit integer values using truncate.
The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer move operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. \( R, R_0, R_1, R_2 \) and \( R_3 \) represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cvtsi32_si128</td>
<td>Move and zero</td>
<td>MOVD</td>
</tr>
<tr>
<td>_mm_cvtsi64_si128</td>
<td>Move and zero</td>
<td>MOVQ</td>
</tr>
<tr>
<td>_mm_cvtsi128_si32</td>
<td>Move lowest 32 bits</td>
<td>MOVD</td>
</tr>
<tr>
<td>_mm_cvtsi128_si64</td>
<td>Move lowest 64 bits</td>
<td>MOVQ</td>
</tr>
</tbody>
</table>

__m128i _mm_cvtsi32_si128(int a)

Moves 32-bit integer \( a \) to the least significant 32 bits of an __m128i object. Zeroes the upper 96 bits of the __m128i object.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>0x0</td>
<td>0x0</td>
<td></td>
</tr>
</tbody>
</table>

__m128i _mm_cvtsi64_si128(__int64 a)

Moves 64-bit integer \( a \) to the lower 64 bits of an __m128i object, zeroing the upper bits.

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td></td>
</tr>
</tbody>
</table>

int _mm_cvtsi128_si32(__m128i a)

Moves the least significant 32 bits of \( a \) to a 32-bit integer.
__int64 _mm_cvtsi128_si64(__m128i a)

Moves the lower 64 bits of a to a 64-bit integer.

---

Integer Load Operations for Streaming SIMD Extensions 2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer load operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. R, R0 and R1 represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_load_si128</td>
<td>Load</td>
<td>MOVDQA</td>
</tr>
<tr>
<td>_mm_loadu_si128</td>
<td>Load</td>
<td>MOVDQU</td>
</tr>
<tr>
<td>_mm_loadl_epi64</td>
<td>Load and zero</td>
<td>MOVQ</td>
</tr>
</tbody>
</table>

__m128i _mm_load_si128(__m128i const*p)

Loads 128-bit value. Address p must be 16-byte aligned.

---

__m128i _mm_loadu_si128(__m128i const*p)
Loads 128-bit value. Address $p$ not need be 16-byte aligned.

$$*p$$

__m128i _mm_loadl_epi64(__m128i const*p)

Load the lower 64 bits of the value pointed to by $p$ into the lower 64 bits of the result, zeroing the upper 64 bits of the result.

R0 R1

*($p[63:0])$ 0x0

### Integer Set Operations for SSE2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer set operations are listed in this topic. These intrinsics are composite intrinsics because they require more than one instruction to implement them. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The results of each intrinsic operation are placed in registers. The information about what is placed in each register appears in the tables below, in the detailed explanation of each intrinsic. $R, R0, R1...R15$ represent the registers in which results are placed.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__mm_set_epi64</td>
<td>Set two integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__mm_set_epi32</td>
<td>Set four integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__mm_set_epi16</td>
<td>Set eight integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__mm_set_epi8</td>
<td>Set sixteen integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>__mm_setl_epi64</td>
<td>Set two integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE2 Instruction</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>_mm_set1_epi32</td>
<td>Set four integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set1_epi16</td>
<td>Set eight integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_set1_epi8</td>
<td>Set sixteen integer values</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_epi64</td>
<td>Set two integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_epi32</td>
<td>Set four integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_epi16</td>
<td>Set eight integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setr_epi8</td>
<td>Set sixteen integer values in reverse order</td>
<td>Composite</td>
</tr>
<tr>
<td>_mm_setzero_si128</td>
<td>Set to zero</td>
<td>Composite</td>
</tr>
</tbody>
</table>

__m128i _mm_set_epi64(__m64 q1, __m64 q0)  
Sets the 2 64-bit integer values.

R0 R1  
q0 q1

__m128i _mm_set_epi32(int i3, int i2, int i1, int i0)  
Sets the 4 signed 32-bit integer values.

R0 R1 R2 R3  
i0 i1 i2 i3

__m128i _mm_set_epi16(short w7, short w6, short w5, short w4, short w3, short w2, short w1, short w0)
Sets the 8 signed 16-bit integer values.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\end{array}
\]

\[
w0 \ w1 \ \ldots \ w7
\]

\[
_{\text{m128i}} \ \text{mm_set_epi8}(\text{char } b15, \text{char } b14, \text{char } b13, \text{char } b12, \text{char } b11, \text{char } b10, \text{char } b9, \text{char } b8, \text{char } b7, \text{char } b6, \text{char } b5, \text{char } b4, \text{char } b3, \text{char } b2, \text{char } b1, \text{char } b0)
\]

Sets the 16 signed 8-bit integer values.

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R15 \\
\end{array}
\]

\[
b0 \ b1 \ \ldots \ b15
\]

\[
_{\text{m128i}} \ \text{mm_set1_epi64}(\text{__m64 } q)
\]

Sets the 2 64-bit integer values to \( q \).

\[
\begin{array}{cc}
R0 & R1 \\
\end{array}
\]

\[
q \ q
\]

\[
_{\text{m128i}} \ \text{mm_set1_epi32}(\text{int } i)
\]

Sets the 4 signed 32-bit integer values to \( i \).

\[
\begin{array}{cccc}
R0 & R1 & R2 & R3 \\
\end{array}
\]

\[
i \ i \ i \ i
\]

\[
_{\text{m128i}} \ \text{mm_set1_epi16}(\text{short } w)
\]

Sets the 8 signed 16-bit integer values to \( w \).

\[
\begin{array}{cccc}
R0 & R1 & \ldots & R7 \\
\end{array}
\]

\[
w \ w \ w \ w
\]

\[
_{\text{m128i}} \ \text{mm_set1_epi8}(\text{char } b)
\]

Sets the 16 signed 8-bit integer values to \( b \).
__m128i _mm_setr_epi64(__m64 q0, __m64 q1)

Sets the 2 64-bit integer values in reverse order.

__m128i _mm_setr_epi32(int i0, int i1, int i2, int i3)

Sets the 4 signed 32-bit integer values in reverse order.

__m128i _mm_setr_epi16(short w0, short w1, short w2, short w3, short w4, short w5, short w6, short w7)

Sets the 8 signed 16-bit integer values in reverse order.

__m128i _mm_setr_epi8(char b15, char b14, char b13, char b12, char b11, char b10, char b9, char b8, char b7, char b6, char b5, char b4, char b3, char b2, char b1, char b0)

Sets the 16 signed 8-bit integer values in reverse order.

__m128i _mm_setzero_si128()

Sets the 128-bit value to zero.
Integer Store Operations for Streaming SIMD Extensions 2

The Streaming SIMD Extensions 2 (SSE2) intrinsics for integer store operations are listed in this topic. The prototypes for the SSE2 intrinsics are in the `emmintrin.h` header file.

The detailed description of each intrinsic contains a table detailing the returns. In these tables, \( p \) is an access to the result.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_stream_si128</td>
<td>Store</td>
<td>MOVNTDQ</td>
</tr>
<tr>
<td>_mm_stream_si32</td>
<td>Store</td>
<td>MOVNTI</td>
</tr>
<tr>
<td>_mm_store_si128</td>
<td>Store</td>
<td>MOVDQA</td>
</tr>
<tr>
<td>_mm_storeu_si128</td>
<td>Store</td>
<td>MOVDQU</td>
</tr>
<tr>
<td>_mm_maskmoveu_si128</td>
<td>Conditional store</td>
<td>MASKMOVDQU</td>
</tr>
<tr>
<td>_mm_storel_epi64</td>
<td>Store lowest</td>
<td>MOVQ</td>
</tr>
</tbody>
</table>

void _mm_stream_si128(_m128i *p, _m128i a)

Stores the data in \( a \) to the address \( p \) without polluting the caches. If the cache line containing address \( p \) is already in the cache, the cache will be updated. Address \( p \) must be 16 byte aligned.

\[ a \]

void _mm_stream_si32(int *p, int a)
Stores the data in \( a \) to the address \( p \) without polluting the caches. If the cache line containing address \( p \) is already in the cache, the cache will be updated.

\[
*p\]
\[
\text{void } _\text{mm_store_si128}(\_\text{m128i } *p, \_\text{m128i } b)
\]
Stores 128-bit value. Address \( p \) must be 16 byte aligned.

\[
*p\]
\[
\text{void } _\text{mm_storeu_si128}(\_\text{m128i } *p, \_\text{m128i } b)
\]
Stores 128-bit value. Address \( p \) need not be 16-byte aligned.

\[
*p\]
\[
\text{void } _\text{mm_maskmoveu_si128}(\_\text{m128i } d, \_\text{m128i } n, \text{char } *p)
\]
Conditionally store byte elements of \( d \) to address \( p \). The high bit of each byte in the selector \( n \) determines whether the corresponding byte in \( d \) will be stored. Address \( p \) need not be 16-byte aligned.

\[
\begin{array}{llll}
\text{if } (n0[7]) & \text{if } (n1[7]) & \ldots & \text{if } (n15[7]) \\
\end{array}
\]

\[
\text{void } _\text{mm_storel_epi64}(\_\text{m128i } *p, \_\text{m128i } a)
\]
Stores the lower 64 bits of the value pointed to by \( p \).

\[
*p[63:0]
\]
\[
a0
\]
Miscellaneous Functions and Intrinsics

Cacheability Support Operations for Streaming SIMD Extensions 2

The prototypes for Streaming SIMD Extensions 2 (SSE2) intrinsics for cacheability support are in the `emmintrin.h` header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE2 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_stream_pd</td>
<td>Store</td>
<td>MOVNTPD</td>
</tr>
<tr>
<td>_mm_stream_si128</td>
<td>Store</td>
<td>MOVNTDQ</td>
</tr>
<tr>
<td>_mm_stream_si32</td>
<td>Store</td>
<td>MOVNTI</td>
</tr>
<tr>
<td>_mm_stream_si64</td>
<td>Store</td>
<td>MOVNTI</td>
</tr>
<tr>
<td>_mm_clflush</td>
<td>Flush</td>
<td>CLFLUSH</td>
</tr>
<tr>
<td>_mm_lfence</td>
<td>Guarantee visibility</td>
<td>LFENCE</td>
</tr>
<tr>
<td>_mm_mfence</td>
<td>Guarantee visibility</td>
<td>MFENCE</td>
</tr>
</tbody>
</table>

void _mm_stream_pd(double *p, __m128d a)

Stores the data in `a` to the address `p` without polluting caches. The address `p` must be 16-byte aligned. If the cache line containing address `p` is already in the cache, the cache will be updated. `p[0] := a0 p[1] := a1`

<table>
<thead>
<tr>
<th>p[0]</th>
<th>p[1]</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
</tr>
</tbody>
</table>

void _mm_stream_si128(__m128i *p, __m128i a)

Stores the data in `a` to the address `p` without polluting the caches. If the cache line containing address `p` is already in the cache, the cache will be updated. Address `p` must be 16-byte aligned.

*p
void _mm_stream_si32(int *p, int a)
Stores the 32-bit integer data in a to the address p without polluting the caches. If the cache line containing address p is already in the cache, the cache will be updated.

void _mm_stream_si64(__int64 *p, __int64 a)
Stores the 64-bit integer data in a to the address p without polluting the caches. If the cache line containing address p is already in the cache, the cache is updated.

void _mm_clflush(void const*p)
Cache line containing p is flushed and invalidated from all caches in the coherency domain.

void _mm_lfence(void)
Guarantees that every load instruction that precedes, in program order, the load fence instruction is globally visible before any load instruction which follows the fence in program order.

void _mm_mfence(void)
Guarantees that every memory access that precedes, in program order, the memory fence instruction is globally visible before any memory instruction which follows the fence in program order.

**Miscellaneous Operations for Streaming SIMD Extensions 2**

The Streaming SIMD Extensions 2 (SSE2) intrinsics for miscellaneous operations are listed in the following table followed by descriptions. The prototypes for SSE2 intrinsics are in the *emmintrin.h* header file.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_packs_epi16</td>
<td>Packed Saturation</td>
<td>PACKSSWB</td>
</tr>
<tr>
<td>_mm_packs_epi32</td>
<td>Packed Saturation</td>
<td>PACKSSDW</td>
</tr>
<tr>
<td>_mm_packus_epi16</td>
<td>Packed Saturation</td>
<td>PACKUSWB</td>
</tr>
<tr>
<td>_mm_extract_epi16</td>
<td>Extraction</td>
<td>PEXTRW</td>
</tr>
<tr>
<td>_mm_insert_epi16</td>
<td>Insertion</td>
<td>PINSRW</td>
</tr>
<tr>
<td>_mm_movemask_epi8</td>
<td>Mask Creation</td>
<td>PMOVMSKB</td>
</tr>
<tr>
<td>_mm_shuffle_epi32</td>
<td>Shuffle</td>
<td>PSHUFD</td>
</tr>
<tr>
<td>_mm_shufflehi_epi16</td>
<td>Shuffle</td>
<td>PSHUFW</td>
</tr>
<tr>
<td>_mm_shufflelo_epi16</td>
<td>Shuffle</td>
<td>PSHUFLW</td>
</tr>
<tr>
<td>_mm_unpackhi_epi8</td>
<td>Interleave</td>
<td>PUNPCKHBW</td>
</tr>
<tr>
<td>_mm_unpackhi_epi16</td>
<td>Interleave</td>
<td>PUNPCKHWD</td>
</tr>
<tr>
<td>_mm_unpackhi_epi32</td>
<td>Interleave</td>
<td>PUNPCKHDQ</td>
</tr>
<tr>
<td>_mm_unpackhi_epi64</td>
<td>Interleave</td>
<td>PUNPCKHDQ</td>
</tr>
<tr>
<td>_mm_unpacklo_epi8</td>
<td>Interleave</td>
<td>PUNPCKLBW</td>
</tr>
<tr>
<td>_mm_unpacklo_epi16</td>
<td>Interleave</td>
<td>PUNPCKLWD</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Operation</td>
<td>Corresponding Instruction</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-------------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>_mm_unpacklo_epi32</td>
<td>Interleave</td>
<td>PUNPCKLDQ</td>
</tr>
<tr>
<td>_mm_unpacklo_epi64</td>
<td>Interleave</td>
<td>PUNPCKLQDQ</td>
</tr>
<tr>
<td>_mm_movepi64_pi64</td>
<td>Move</td>
<td>MOVDQ2Q</td>
</tr>
<tr>
<td>_mm_movpi64_epi64</td>
<td>Move</td>
<td>MOVDQ2Q</td>
</tr>
<tr>
<td>_mm_move_epi64</td>
<td>Move</td>
<td>MOVQ</td>
</tr>
<tr>
<td>_mm_unpackhi_pd</td>
<td>Interleave</td>
<td>UNPCKHPD</td>
</tr>
<tr>
<td>_mm_unpacklo_pd</td>
<td>Interleave</td>
<td>UNPCKLPD</td>
</tr>
<tr>
<td>_mm_movemask_pd</td>
<td>Create mask</td>
<td>MOVMSKPD</td>
</tr>
<tr>
<td>_mm_shuffle_pd</td>
<td>Select values</td>
<td>SHUFPD</td>
</tr>
</tbody>
</table>

__m128i _mm_packs_epi16(__m128i a, __m128i b)

Packs the 16 signed 16-bit integers from \(a\) and \(b\) into 8-bit integers and saturates.

<table>
<thead>
<tr>
<th>R0</th>
<th>...</th>
<th>R7</th>
<th>R8</th>
<th>...</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
</tr>
<tr>
<td>Saturate(a0)</td>
<td>Saturate(a7)</td>
<td>Saturate(b0)</td>
<td>Saturate(b7)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128i _mm_packs_epi32(__m128i a, __m128i b)

Packs the 8 signed 32-bit integers from \(a\) and \(b\) into signed 16-bit integers and saturates.

<table>
<thead>
<tr>
<th>R0</th>
<th>...</th>
<th>R3</th>
<th>R4</th>
<th>...</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
<td>Signed</td>
<td>...</td>
<td>Signed</td>
</tr>
<tr>
<td>Saturate(a0)</td>
<td>Saturate(a3)</td>
<td>Saturate(b0)</td>
<td>Saturate(b3)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

__m128i _mm_packus_epi16(__m128i a, __m128i b)
Packs the 16 signed 16-bit integers from \( a \) and \( b \) into 8-bit unsigned integers and saturates.

\[
\begin{array}{cccccc}
\text{R0} & \ldots & \text{R7} & \text{R8} & \ldots & \text{R15} \\
\text{Unsigned} & \ldots & \text{Unsigned} & \text{Unsigned} & \ldots & \text{Unsigned} \\
\text{Saturate} (a0) & \text{Saturate} (a7) & \text{Saturate} (b0) & \text{Saturate} (b15)
\end{array}
\]

\[
\text{int } _\text{mm\_extract\_epi16(__m128i } a, \text{ int } \text{imm})
\]

Extracts the selected signed or unsigned 16-bit integer from \( a \) and zero extends. The selector \( \text{imm} \) must be an immediate.

\[
\begin{array}{c}
\text{R0} \\
(\text{imm == 0}) \ ? \ a0: \ (\text{imm == 1}) \ ? \ a1: \ldots \ (\text{imm==7}) \ ? \ a7
\end{array}
\]

\[
\text{__m128i } _\text{mm\_insert\_epi16(__m128i } a, \text{ int } b, \text{ int } \text{imm})
\]

Inserts the least significant 16 bits of \( b \) into the selected 16-bit integer of \( a \). The selector \( \text{imm} \) must be an immediate.

\[
\begin{array}{ccc}
\text{R0} & \text{R1} & \ldots \\
(\text{imm == 0}) \ ? \ b \ (\text{imm == 1}) \ ? \ b \ldots & (\text{imm == 7}) \ ? \ b \\
: \ a0; & : \ a1; & : \ a7;
\end{array}
\]

\[
\text{int } _\text{mm\_movemask\_epi8(__m128i } a)
\]

Creates a 16-bit mask from the most significant bits of the 16 signed or unsigned 8-bit integers in \( a \) and zero extends the upper bits.

\[
\begin{array}{c}
\text{R0} \\
a15[7] \ll 15 \mid a14[7] \ll 14 \mid \ldots \ a1[7] \ll 1 \mid a0[7]
\end{array}
\]

\[
\text{__m128i } _\text{mm\_shuffle\_epi32(__m128i } a, \text{ int } \text{imm})
\]
Shuffles the 4 signed or unsigned 32-bit integers in \( a \) as specified by \( \text{imm} \). The shuffle value, \( \text{imm} \), must be an immediate. See Macro Function for Shuffle for a description of shuffle semantics.

\[
\text{__m128i} \ _\text{mm}_\text{shufflehi}_\text{epi}16(\text{__m128i} \ a, \ \text{int} \ \text{imm})
\]

Shuffles the upper 4 signed or unsigned 16-bit integers in \( a \) as specified by \( \text{imm} \). The shuffle value, \( \text{imm} \), must be an immediate. See Macro Function for Shuffle for a description of shuffle semantics.

\[
\text{__m128i} \ _\text{mm}_\text{shufflelo}_\text{epi}16(\text{__m128i} \ a, \ \text{int} \ \text{imm})
\]

Shuffles the lower 4 signed or unsigned 16-bit integers in \( a \) as specified by \( \text{imm} \). The shuffle value, \( \text{imm} \), must be an immediate. See Macro Function for Shuffle for a description of shuffle semantics.

\[
\text{__m128i} \ _\text{mm}_\text{unpackhi}_\text{epi}8(\text{__m128i} \ a, \ \text{__m128i} \ b)
\]

Interleaves the upper 8 signed or unsigned 8-bit integers in \( a \) with the upper 8 signed or unsigned 8-bit integers in \( b \).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>...</th>
<th>R14</th>
<th>R15</th>
</tr>
</thead>
<tbody>
<tr>
<td>a8</td>
<td>b8</td>
<td>a9</td>
<td>b9</td>
<td>...</td>
<td>a15</td>
<td>b15</td>
</tr>
</tbody>
</table>

\[
\text{__m128i} \ _\text{mm}_\text{unpackhi}_\text{epi}16(\text{__m128i} \ a, \ \text{__m128i} \ b)
\]

Interleaves the upper 4 signed or unsigned 16-bit integers in \( a \) with the upper 4 signed or unsigned 16-bit integers in \( b \).

<table>
<thead>
<tr>
<th>R0</th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
<th>R5</th>
<th>R6</th>
<th>R7</th>
</tr>
</thead>
<tbody>
<tr>
<td>a4</td>
<td>b4</td>
<td>a5</td>
<td>b5</td>
<td>a6</td>
<td>b6</td>
<td>a7</td>
<td>b7</td>
</tr>
</tbody>
</table>

\[
\text{__m128i} \ _\text{mm}_\text{unpackhi}_\text{epi}32(\text{__m128i} \ a, \ \text{__m128i} \ b)
\]

Interleaves the upper 2 signed or unsigned 32-bit integers in \( a \) with the upper 2 signed or unsigned 32-bit integers in \( b \).
R0 R1 R2 R3
a2 b2 a3 b3

__m128i _mm_unpackhi_epi64(__m128i a, __m128i b)
Interleaves the upper signed or unsigned 64-bit integer in a with the upper signed or unsigned 64-bit integer in b.

R0 R1
a1 b1

__m128i _mm_unpacklo_epi8(__m128i a, __m128i b)
Interleaves the lower 8 signed or unsigned 8-bit integers in a with the lower 8 signed or unsigned 8-bit integers in b.

R0 R1 R2 R3 ... R14 R15
a0 b0 a1 b1 ... a7 b7

__m128i _mm_unpacklo_epi16(__m128i a, __m128i b)
Interleaves the lower 4 signed or unsigned 16-bit integers in a with the lower 4 signed or unsigned 16-bit integers in b.

R0 R1 R2 R3 R4 R5 R6 R7
a0 b0 a1 b1 a2 b2 a3 b3

__m128i _mm_unpacklo_epi32(__m128i a, __m128i b)
Interleaves the lower 2 signed or unsigned 32-bit integers in a with the lower 2 signed or unsigned 32-bit integers in b.

R0 R1 R2 R3
a0 b0 a1 b1

__m128i _mm_unpacklo_epi64(__m128i a, __m128i b)
Interleaves the lower signed or unsigned 64-bit integer in \(a\) with the lower signed or unsigned 64-bit integer in \(b\).

\[
\begin{array}{ll}
R0 & R1 \\
a0 & b0 \\
\end{array}
\]

__m64 _mm_movepi64_pi64(__m128i a)

Returns the lower 64 bits of \(a\) as an \__m64\ type.

\[
\begin{array}{ll}
R0 \\
a0 \\
\end{array}
\]

__m128i _mm_movpi64_pi64(__m64 a)

Moves the 64 bits of \(a\) to the lower 64 bits of the result, zeroing the upper bits.

\[
\begin{array}{ll}
R0 & R1 \\
a0 & 0X0 \\
\end{array}
\]

__m128i _mm_move_epi64(__m128i a)

Moves the lower 64 bits of \(a\) to the lower 64 bits of the result, zeroing the upper bits.

\[
\begin{array}{ll}
R0 & R1 \\
a0 & 0X0 \\
\end{array}
\]

__m128d _mm_unpackhi_pd(__m128d a, __m128d b)

Interleaves the upper DP FP values of \(a\) and \(b\).

\[
\begin{array}{ll}
R0 & R1 \\
a1 & b1 \\
\end{array}
\]

__m128d _mm_unpacklo_pd(__m128d a, __m128d b)
Interleaves the lower DP FP values of \(a\) and \(b\).

\[
\begin{array}{c|c}
R0 & R1 \\
a0 & b0 \\
\end{array}
\]

\[
\text{int } _\text{mm_movemask_pd}(\_\text{m128d } a) \\
\]

Creates a two-bit mask from the sign bits of the two DP FP values of \(a\).

\[
\text{sign}(a1) << 1 | \text{sign}(a0) \\
\]

\[
\_\text{m128d } _\text{mm_shuffle_pd}(\_\text{m128d } a, \_\text{m128d } b, \text{ int } i) \\
\]

Selects two specific DP FP values from \(a\) and \(b\), based on the mask \(i\). The mask must be an immediate. See Macro Function for Shuffle for a description of the shuffle semantics.

**Intrinsics for Casting Support**

The Intel® C++ Compiler supports casting between various single-precision, double-precision, and integer vector types. These intrinsics do not convert values; they change one data type to another without changing the value. The intrinsics for casting support do not have any corresponding Streaming SIMD Extensions 2 (SSE2) instructions.

\[
\begin{array}{c}
\_\text{m128d } _\text{mm_castpd_ps}(\_\text{m128d } in) \\
\_\text{m128d } _\text{mm_castpd_si128}(\_\text{m128d } in) \\
\_\text{m128d } _\text{mm_castps_pd}(\_\text{m128 in}) \\
\_\text{m128d } _\text{mm_castps_si128}(\_\text{m128 in}) \\
\_\text{m128 } _\text{mm_castsi128_ps}(\_\text{m128i in}) \\
\_\text{m128d } _\text{mm_castsi128_pd}(\_\text{m128i in}) \\
\end{array}
\]

**Pause Intrinsic for Streaming SIMD Extensions 2**

The prototype for this Streaming SIMD Extensions (SSE) intrinsic is in the `xmmintrin.h` header file.
**PAUSE Intrinsic**

```c
void _mm_pause(void)
```

The `pause` intrinsic is used in spin-wait loops with the processors implementing dynamic execution (especially out-of-order execution). In the spin-wait loop, the `pause` intrinsic improves the speed at which the code detects the release of the lock and provides especially significant performance gain.

The execution of the next instruction is delayed for an implementation-specific amount of time. The `PAUSE` instruction does not modify the architectural state.

For dynamic scheduling, the `PAUSE` instruction reduces the penalty of exiting from the spin-loop.

**Example of loop with the PAUSE instruction:**

In this example, the program spins until memory location `A` matches the value in register `eax`. The code sequence that follows shows a test-and-test-and-set.

```c
spin_loop: pause
cmp eax, A
jne spin_loop
```

In this example, the spin occurs only after the attempt to get a lock has failed.

```c
get_lock: mov eax, 1
xchg eax, A ; Try to get lock
cmp eax, 0 ; Test if successful
jne spin_loop
```

**Critical Section**

```c
// critical_section code
mov A, 0 ; Release lock
jmp continue
spin_loop: pause;
// spin-loop hint
cmp 0, A ;
// check lock availability
jne spin_loop
jmp get_lock
// continue: other code
```

**Note**

The first branch is predicted to fall-through to the critical section in anticipation of successfully gaining access to the lock. It is highly recommended that all spin-wait loops include the `PAUSE` instruction. Since `PAUSE` is backwards compatible to all existing IA-32 architecture-based processor generations, a test for processor type (a CPUID test) is not needed. All legacy processors execute `PAUSE` instruction as a `NOP`, but in processors that use the `PAUSE` instruction as a hint there can be significant performance benefit.
Macro Function for Shuffle

The Streaming SIMD Extensions 2 (SSE2) provide a macro function to help create constants that describe shuffle operations. The macro takes two small integers (in the range of 0 to 1) and combines them into an 2-bit immediate value used by the `SHUFPD` instruction. See the following example.

**Shuffle Function Macro**

```
__m128_shuffle((x, y)
expands to the value of
(x<<1) | y
```

You can view the two integers as selectors for choosing which two words from the first input operand and which two words from the second are to be put into the result word.

**View of Original and Result Words with Shuffle Function Macro**

```
| m1 | 127 | a | b |
| m2 | 127 | c | d |
```

```
m3 = _mm_shuffle_ps(m1, m2, __m128_shuffle(1, 0))
| m3 | 127 | c | b |
```

**Intrinsics for Streaming SIMD Extensions 3**

**Overview: Streaming SIMD Extensions 3**

The Intel® C++ intrinsics listed in this section are designed for the Intel® Pentium® 4 processor with Intel® Streaming SIMD Extensions 3 (SSE3). They will not function correctly on other IA-32 architecture-based processors. The new Intel® SSE3 intrinsics include:

- Floating-point Vector Intrinsics
- Integer Vector Intrinsics
- Miscellaneous Intrinsics
Macro Functions

The prototypes for these intrinsics are in the `pmmintrin.h` header file.

**Note**

You can also use the single `ia32intrin.h` header file for any IA-32 architecture-based intrinsics.

**Integer Vector Intrinsics for Streaming SIMD Extensions 3**

The integer vector intrinsic listed here is designed for the Intel® Pentium® 4 processor with Streaming SIMD Extensions 3 (SSE3).

The prototype for this intrinsic is in the `pmmintrin.h` header file.

\[ \text{R represents the register into which the returns are placed.} \]

```
__m128i _mm_lddqu_si128(__m128i const *p);
```

Loads an unaligned 128-bit value. This differs from `movdqu` in that it can provide higher performance in some cases. However, it also may provide lower performance than `movdqu` if the memory value being read was just previously written.

```
R
*p;
```

**Single-precision Floating-point Vector Intrinsics for Streaming SIMD Extensions 3**

The single-precision floating-point vector intrinsics listed here are designed for the Intel® Pentium® 4 processor with Streaming SIMD Extensions 3 (SSE3).

The results of each intrinsic operation are placed in the registers R0, R1, R2, and R3.

The prototypes for these intrinsics are in the `pmmintrin.h` header file.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE3 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_addsub_ps</td>
<td>Subtract and add</td>
<td>ADDSUBPS</td>
</tr>
<tr>
<td>_mm_hadd_ps</td>
<td>Add</td>
<td>HADDPS</td>
</tr>
<tr>
<td>_mm_hsub_ps</td>
<td>Subtracts</td>
<td>HSUBPS</td>
</tr>
<tr>
<td>_mm_movehdup_ps</td>
<td>Duplicates</td>
<td>MOVSHDUP</td>
</tr>
<tr>
<td>_mm_moveldup_ps</td>
<td>Duplicates</td>
<td>MOVSLDUP</td>
</tr>
</tbody>
</table>

extern __m128 _mm_addsub_ps(__m128 a, __m128 b);

Subtracts even vector elements while adding odd vector elements.

R0       R1       R2       R3
a0 - b0; a1 + b1; a2 - b2; a3 + b3;

extern __m128 _mm_hadd_ps(__m128 a, __m128 b);

Adds adjacent vector elements.

R0       R1       R2       R3
a0 + a1; a2 + a3; b0 + b1; b2 + b3;

extern __m128 _mm_hsub_ps(__m128 a, __m128 b);

Subtracts adjacent vector elements.

R0       R1       R2       R3
a0 - a1; a2 - a3; b0 - b1; b2 - b3;

extern __m128 _mm_movehdup_ps(__m128 a);

Duplicates odd vector elements into even vector elements.
Duplicates even vector elements into odd vector elements.

```
a0; a0; a2; a2;
```

### Double-precision Floating-point Vector Intrinsics for Streaming SIMD Extensions 3

The double-precision floating-point intrinsics listed here are designed for the Intel® Pentium® 4 processor with Streaming SIMD Extensions 3 (SSE3). The results of each intrinsic operation are placed in the registers R0 and R1. The prototypes for these intrinsics are in the pmmintrin.h header file.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE3 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_addsub_pd</td>
<td>Subtract and add</td>
<td>ADDSUBPD</td>
</tr>
<tr>
<td>_mm_hadd_pd</td>
<td>Add</td>
<td>HADDPD</td>
</tr>
<tr>
<td>_mm_hsub_pd</td>
<td>Subtract</td>
<td>HSUBPD</td>
</tr>
<tr>
<td>_mm_loadup_pd</td>
<td>Duplicate</td>
<td>MOVDDUP</td>
</tr>
<tr>
<td>_mm_movedup_pd</td>
<td>Duplicate</td>
<td>MOVDDUP</td>
</tr>
</tbody>
</table>

```
extern __m128d _mm_addsub_pd(__m128d a, __m128d b);
```

Adds upper vector element while subtracting lower vector element.

```
a0 - b0; a1 + b1;
```
extern __m128d _mm_hadd_pd(__m128d a, __m128d b);

Add adjacent vector elements.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 + a1; b0 + b1; \\
\end{array}
\]

Extern __m128d _mm_hsub_pd(__m128d a, __m128d b);

Subtracts adjacent vector elements.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0 - a1; b0 - b1; \\
\end{array}
\]

Extern __m128d _mm_loadpd(double const * dp);

Duplicates a double value into upper and lower vector elements.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
*dp; *dp; \\
\end{array}
\]

Extern __m128d _mm_moveldup_pd(__m128d a);

Duplicates lower vector element into upper vector element.

\[
\begin{array}{c|c}
R0 & R1 \\
\hline
a0; a0; \\
\end{array}
\]

**Macro Functions for Streaming SIMD Extensions 3**

The macro function intrinsics listed here are designed for the Intel® Pentium® 4 processor with Streaming SIMD Extensions 3 (SSE3).

The prototypes for these intrinsics are in the `pmmintrin.h` header file.

`_MM_SET_DENORMALS_ZERO_MODE(x)`
Macro arguments: one of __MM_DENORMALS_ZERO_ON,
__MM_DENORMALS_ZERO_OFF This causes "denormals are zero" mode to be
turned on or off by setting the appropriate bit of the control register.

__MM_GET_DENORMALS_ZERO_MODE()
No arguments. This returns the current value of the denormals are zero mode bit
of the control register.

Miscellaneous Intrinsics for Streaming SIMD Extensions 3

The miscellaneous intrinsics listed here are designed for the Intel® Pentium® 4
processor with Streaming SIMD Extensions 3 (SSE3).
The prototypes for these intrinsics are in the pmmintrin.h header file.

extern void _mm_monitor(void const *p, unsigned extensions,
unsigned hints);
Generates the MONITOR instruction. This sets up an address range for the
monitor hardware using p to provide the logical address, and will be passed to
the monitor instruction in register eax. The extensions parameter contains
optional extensions to the monitor hardware which will be passed in ecx. The
hints parameter will contain hints to the monitor hardware, which will be passed
in edx. A non-zero value for extensions will cause a general protection fault.

extern void _mm_mwait(unsigned extensions, unsigned hints);
Generates the MWAIT instruction. This instruction is a hint that allows the
processor to stop execution and enter an implementation-dependent optimized
state until occurrence of a class of events. In future processor designs
extensions and hints parameters may be used to convey additional information to
the processor. All non-zero values of extensions and hints are reserved. A non-zero value for extensions will cause a general protection fault.

Intrinsics for Supplemental Streaming SIMD Extensions 3

Overview: Supplemental Streaming SIMD Extensions 3
Intel's C++ intrinsics listed in this section correspond to the Supplemental Streaming SIMD Extensions 3 instructions. The prototypes for these intrinsics are in \texttt{tmmintrin.h}. You can also use the \texttt{ia32intrin.h} header file for these intrinsics.

- **Addition Intrinsics**
- **Subtraction Intrinsics**
- **Multiplication Intrinsics**
- **Absolute Value Intrinsics**
- **Shuffle Intrinsics**
- **Concatenate Intrinsics**
- **Negation Intrinsics**

### Addition Intrinsics

Use the following SSSE3 intrinsics for horizontal addition.

```c
extern __m128i _mm_hadd_epi16 (__m128i a, __m128i b);
```

Add horizontally packed signed words.

**Interpreting** \(a\), \(b\), and \(r\) as arrays of 16-bit signed integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = a[2*i] + a[2i+1];
    r[i+4] = b[2*i] + b[2*i+1];
}
```

```c
extern __m128i _mm_hadd_epi32 (__m128i a, __m128i b);
```

Add horizontally packed signed dwords.

**Interpreting** \(a\), \(b\), and \(r\) as arrays of 32-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = a[2*i] + a[2i+1];
    r[i+2] = b[2*i] + b[2*i+1];
}
```

```c
extern __m128i _mm_hadds_epi16 (__m128i a, __m128i b);
```

Add horizontally packed signed words with signed saturation.

**Interpreting** \(a\), \(b\), and \(r\) as arrays of 16-bit signed integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = signed_saturate_to_word(a[2*i] + a[2i+1]);
    r[i+4] = signed_saturate_to_word(b[2*i] + b[2*i+1]);
}
Add horizontally packed signed words.

Interpreting a, b, and r as arrays of 16-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = a[2*i] + a[2i+1];
    r[i+2] = b[2*i] + b[2*i+1];
}
```

Add horizontally packed signed dwords.

Interpreting a, b, and r as arrays of 32-bit signed integers:

```c
r[0] = a[1] + a[0];
r[1] = b[1] + b[0];
```

Add horizontally packed signed words with signed saturation.

Interpreting a, b, and r as arrays of 16-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = signed_saturate_to_word(a[2*i] + a[2i+1]);
    r[i+2] = signed_saturate_to_word(b[2*i] + b[2*i+1]);
}
```

Subtraction Intrinsics

Use the following SSSE3 intrinsics for horizontal subtraction.

Add horizontally packed signed words.

Interpreting a, b, and r as arrays of 16-bit signed integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = a[2*i] - a[2i+1];
    r[i+4] = b[2*i] - b[2*i+1];
}
```

Add horizontally packed signed dwords.

Interpreting a, b, and r as arrays of 32-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = a[2*i] - a[2i+1];
    r[i+2] = b[2*i] - b[2*i+1];
```
Subtract horizontally packed signed words with signed saturation.

Interpreting $a$, $b$, and $r$ as arrays of 16-bit signed integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = signed_saturate_to_word(a[2*i] - a[2*i+1]);
    r[i+4] = signed_saturate_to_word(b[2*i] - b[2*i+1]);
}
```

Subtract horizontally packed signed words.

Interpreting $a$, $b$, and $r$ as arrays of 16-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = a[2*i] - a[2*i+1];
    r[i+2] = b[2*i] - b[2*i+1];
}
```

Subtract horizontally packed signed dwords.

Interpreting $a$, $b$, and $r$ as arrays of 32-bit signed integers:

```c
r[0] = a[0] - a[1];
r[1] = b[0] - b[1];
```

Subtract horizontally packed signed words with signed saturation.

Interpreting $a$, $b$, and $r$ as arrays of 16-bit signed integers:

```c
for (i = 0; i < 2; i++) {
    r[i] = signed_saturate_to_word(a[2*i] - a[2*i+1]);
    r[i+2] = signed_saturate_to_word(b[2*i] - b[2*i+1]);
}
```

### Multiplication Intrinsics

Use the following SSSE3 intrinsics for multiplication.

```c
extern __m128i _mm_maddubs_epi16 (__m128i a, __m128i b);
```

Multiply signed and unsigned bytes, add horizontal pair of signed words, pack saturated signed words.
Interpreting \( a \) as array of unsigned 8-bit integers, \( b \) as arrays of signed 8-bit integers, and \( r \) as arrays of 16-bit signed integers:

```c
for (i = 0; i < 8; i++) {
    r[i] = signed_saturate_to_word(a[2*i+1] * b[2*i+1] + a[2*i]*b[2*i]);
}
```

```c
extern __m64 __mm_maddubs_pi16 (__m64 a, __m64 b);
```

Multiply signed and unsigned bytes, add horizontal pair of signed words, pack saturated signed words.

Interpreting \( a \) as array of unsigned 8-bit integers, \( b \) as arrays of signed 8-bit integers, and \( r \) as arrays of 16-bit signed integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = signed_saturate_to_word(a[2*i+1] * b[2*i+1] + a[2*i]*b[2*i]);
}
```

```c
extern __m128i __mm_mulhrs_epi16 (__m128i a, __m128i b);
```

Multiply signed words, scale and round signed dwords, pack high 16-bits.

Interpreting \( a, b, \) and \( r \) as arrays of signed 16-bit integers:

```c
for (i = 0; i < 8; i++) {
    r[i] = ( ((int32)((a[i] * b[i]) >> 14) + 1) >> 1) & 0xFFFF;
}
```

```c
extern __m64 __mm_mulhrs_pi16 (__m64 a, __m64 b);
```

Multiply signed words, scale and round signed dwords, pack high 16-bits.

Absolute Value Intrinsics

Use the following SSSE3 intrinsics to compute absolute values.

```c
extern __m128i __mm_abs_epi8 (__m128i a);
```

Compute absolute value of signed bytes.

Interpreting \( a \) and \( r \) as arrays of signed 8-bit integers:

```c
for (i = 0; i < 16; i++) {
    r[i] = abs(a[i]);
}
```

```c
extern __m128i __mm_abs_epi16 (__m128i a);
```
Compute absolute value of signed words.

Interpreting \(a\) and \(r\) as arrays of signed 16-bit integers:

```c
for (i = 0; i < 8; i++) {
    r[i] = abs(a[i]);
}
```

```c
extern __m128i __mm_abs_epi32 (__m128i a);
```

Compute absolute value of signed dwords.

Interpreting \(a\) and \(r\) as arrays of signed 32-bit integers:

```c
for (i = 0; i < 4; i++) {
    r[i] = abs(a[i]);
}
```

```c
extern __m64 __mm_abs_pi8 (__m64 a);
```

Compute absolute value of signed bytes.

Interpreting \(a\) and \(r\) as arrays of signed 8-bit integers:

```c
for (i = 0; i < 8; i++) {
    r[i] = abs(a[i]);
}
```

```c
extern __m64 __mm_abs_pi16 (__m64 a);
```

Shuffle Intrinsics

Use the following SSSE3 intrinsics for shuffle.

```c
extern __m128i __mm_shuffle_epi8 (__m128i a, __m128i b);
```

Shuffle bytes from \(a\) according to contents of \(b\).
Interpreting $a$, $b$, and $r$ as arrays of unsigned 8-bit integers:

```c
for (i = 0; i < 16; i++){
    if (b[i] & 0x80){
        r[i] = 0;
    } else {
        r[i] = a[b[i] & 0x0F];
    }
}
extern __m64 _mm_shuffle_pi8 (__m64 a, __m64 b);
```

Shuffle bytes from $a$ according to contents of $b$.

Interpreting $a$, $b$, and $r$ as arrays of unsigned 8-bit integers:

```c
for (i = 0; i < 8; i++){
    if (b[i] & 0x80){
        r[i] = 0;
    } else {
        r[i] = a[b[i] & 0x07];
    }
}
```

Concatenate Intrinsics

Use the following SSSE3 intrinsics for concatenation.

```c
extern __m128i _mm_alignr_epi8 (__m128i a, __m128i b, int n);
```

Concatenate $a$ and $b$, extract byte-aligned result shifted to the right by $n$.

Interpreting $t_1$ as 256-bit unsigned integer, $a$, $b$, and $r$ as 128-bit unsigned integers:

```c
t1[255:128] = a;
t1[127:0] = b;
t1[255:0] = t1[255:0] >> (8 * n); // unsigned shift
r[127:0] = t1[127:0];
extern __m64 _mm_alignr_pi8 (__m64 a, __m64 b, int n);
```

Concatenate $a$ and $b$, extract byte-aligned result shifted to the right by $n$.

Interpreting $t_1$ as 127-bit unsigned integer, $a$, $b$ and $r$ as 64-bit unsigned integers:

```c
t1[127:64] = a;
t1[63:0] = b;
t1[127:0] = t1[127:0] >> (8 * n); // unsigned shift
```
Negation Intrinsics

Use the following SSSE3 intrinsics for negation.

```c
extern __m128i _mm_sign_epi8 (__m128i a, __m128i b);
```

Negate packed bytes in `a` if corresponding sign in `b` is less than zero.

**Interpreting `a`, `b`, and `r` as arrays of signed 8-bit integers:**

```c
for (i = 0; i < 16; i++){
    if (b[i] < 0){
        r[i] = -a[i];
    } else
    if (b[i] == 0){
        r[i] = 0;
    } else
    { r[i] = a[i];
}
```

```c
extern __m128i _mm_sign_epi16 (__m128i a, __m128i b);
```

Negate packed words in `a` if corresponding sign in `b` is less than zero.

**Interpreting `a`, `b`, and `r` as arrays of signed 16-bit integers:**

```c
for (i = 0; i < 8; i++){
    if (b[i] < 0){
        r[i] = -a[i];
    } else
    if (b[i] == 0){
        r[i] = 0;
    } else
    { r[i] = a[i];
}
```

```c
extern __m128i _mm_sign_epi32 (__m128i a, __m128i b);
```

Negate packed dwords in `a` if corresponding sign in `b` is less than zero.

**Interpreting `a`, `b`, and `r` as arrays of signed 32-bit integers:**

```c
for (i = 0; i < 4; i++){
    if (b[i] < 0){
        r[i] = -a[i];
    }
```
else
  if (b[i] == 0) {
    r[i] = 0;
  } else {
    r[i] = a[i];
  }
}

extern __m64 _mm_sign_pi8 (__m64 a, __m64 b);

Negate packed bytes in \( a \) if corresponding sign in \( b \) is less than zero.

Interpreting \( a, b, \) and \( r \) as arrays of signed 8-bit integers:

```c
for (i = 0; i < 16; i++) {
  if (b[i] < 0) {
    r[i] = -a[i];
  } else {
    if (b[i] == 0) {
      r[i] = 0;
    } else {
      r[i] = a[i];
    }
  }
}
```

extern __m64 _mm_sign_pi16 (__m64 a, __m64 b);

Negate packed words in \( a \) if corresponding sign in \( b \) is less than zero.

Interpreting \( a, b, \) and \( r \) as arrays of signed 16-bit integers:

```c
for (i = 0; i < 8; i++) {
  if (b[i] < 0) {
    r[i] = -a[i];
  } else {
    if (b[i] == 0) {
      r[i] = 0;
    } else {
      r[i] = a[i];
    }
  }
}
```

extern __m64 _mm_sign_pi32 (__m64 a, __m64 b);

Negate packed dwords in \( a \) if corresponding sign in \( b \) is less than zero.

Interpreting \( a, b, \) and \( r \) as arrays of signed 32-bit integers:

```c
for (i = 0; i < 2; i++) {
```
if (b[i] < 0){
    r[i] = -a[i];
} else if (b[i] == 0){
    r[i] = 0;
} else {
    r[i] = a[i];
}

**Intrinsics for Streaming SIMD Extensions 4**

**Overview: Streaming SIMD Extensions 4**

The intrinsics in this section correspond to Intel® Streaming SIMD Extensions 4 (SSE4) instructions. Intel® SSE4 includes the following categories:

- **Vectorizing Compiler and Media Accelerators**. The prototypes for these intrinsics are in the `smmitnrin.h` file.

- **Efficient Accelerated String and Text Processing**. The prototypes for these intrinsics are in the `nmmintrin.h` file.

**Streaming SIMD Extensions 4: Vectorizing Compiler and Media Accelerators**

**Overview: Streaming SIMD Extensions 4 Vectorizing Compiler and Media Accelerators**

The intrinsics in this section correspond to Streaming SIMD Extensions 4 (SSE4) Vectorizing Compiler and Media Accelerators instructions.

- **Packed Blending Intrinsics for Streaming SIMD Extensions 4**
- **Floating Point Dot Product Intrinsics for Streaming SIMD Extensions 4**
- **Packed Format Conversion Intrinsics for Streaming SIMD Extensions 4**
- **Packed Integer Min/Max Intrinsics for Streaming SIMD Extensions 4**
- **Floating Point Rounding Intrinsics for Streaming SIMD Extensions 4**
- **DWORD Multiply Intrinsics for Streaming SIMD Extensions 4**
- **Register Insertion/Extraction Intrinsics for Streaming SIMD Extensions 4**
- **Test Intrinsics for Streaming SIMD Extensions 4**
- **Packed DWORD to Unsigned WORD Intrinsic for Streaming SIMD Extensions 4**
- **Packed Compare for Equal Intrinsics for Streaming SIMD Extensions 4**
- **Cacheability Support Intrinsic for Streaming SIMD Extension 4**

The prototypes for these intrinsics are in the `smmintrin.h` file.

**Packed Blending Intrinsics for Streaming SIMD Extensions 4**

These intrinsics pack multiple operations in a single instruction. Blending conditionally copies one field in the source onto the corresponding field in the destination.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__m128 _mm_blend_ps(__m128 v1, __m128 v2, const int mask)</code></td>
<td>Selects float single precision data from 2 sources using constant mask</td>
<td>BLENDPS</td>
</tr>
<tr>
<td><code>__m128d _mm_blend_pd(__m128d v1, __m128d v2, const int mask)</code></td>
<td>Selects float double precision data from 2 sources using constant mask</td>
<td>BLENDPD</td>
</tr>
<tr>
<td><code>__m128 _mm_blendv_ps(__m128 v1, __m128 v2, __m128 v3)</code></td>
<td>Selects float single precision data from 2 sources using variable mask</td>
<td>BLENDVPS</td>
</tr>
<tr>
<td><code>__m128d _mm_blendv_pd(__m128d v1, __m128d v2, __m128d v3)</code></td>
<td>Selects float double precision data from 2 sources using variable mask</td>
<td>BLENDVPD</td>
</tr>
<tr>
<td>Intrinsic Syntax</td>
<td>Operation</td>
<td>Corresponding SSE4 Instruction</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>__m128i</td>
<td>Selects integer bytes from 2 sources using variable mask</td>
<td>PBLENDVB</td>
</tr>
<tr>
<td>__m128i</td>
<td>Selects integer words from 2 sources using constant mask</td>
<td>PBLENDNW</td>
</tr>
</tbody>
</table>

**Floating Point Dot Product Intrinsics for Streaming SIMD Extensions 4**

These intrinsics enable floating point single precision and double precision dot products.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_dp_pd</td>
<td>Double precision dot product</td>
<td>DPPD</td>
</tr>
<tr>
<td>_mm_dp_ps</td>
<td>Single precision dot product</td>
<td>DPPS</td>
</tr>
</tbody>
</table>

```
__m128d _mm_dp_pd ( __m128d a, __m128d b, const int mask)
```

This intrinsic calculates the dot product of double precision packed values with mask-defined summing and zeroing of the parts of the result.

```
__m128 _mm_dp_ps ( __m128 a, __m128 b, const int mask)
```

This intrinsic calculates the dot product of single precision packed values with mask-defined summing and zeroing of the parts of the result.

**Packed Format Conversion Intrinsics for Streaming SIMD Extensions 4**
These intrinsics convert a packed integer to a zero-extended or sign-extended integer with wider type.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i _mm_cvtepi8_epi32(__m128i a)</td>
<td>Sign extend 4 bytes into 4 double words</td>
<td>PMOVVSXBD</td>
</tr>
<tr>
<td>__m128i _mm_cvtepi8_epi64 (__m128i a)</td>
<td>Sign extend 2 bytes into 2 quad words</td>
<td>PMOVVSXBQ</td>
</tr>
<tr>
<td>__m128i _mm_cvtepi8_epi16(__m128i a)</td>
<td>Sign extend 8 bytes into 8 words</td>
<td>PMOVVSXBW</td>
</tr>
<tr>
<td>__m128i _mm_cvtepi32_epi64(__m128i a)</td>
<td>Sign extend 2 double words into 2 quad words</td>
<td>PMOVVSXDQ</td>
</tr>
<tr>
<td>__m128i _mm_cvtepi16_epi32(__m128i a)</td>
<td>Sign extend 4 words into 4 double words</td>
<td>PMOVVSXWD</td>
</tr>
<tr>
<td>__m128i _mm_cvtepi16_epi64(__m128i a)</td>
<td>Sign extend 2 words into 2 quad words</td>
<td>PMOVVSXWQ</td>
</tr>
<tr>
<td>__m128i _mm_cvtepu8_epi32(__m128i a)</td>
<td>Zero extend 4 bytes into 4 double words</td>
<td>PMOVZXBBD</td>
</tr>
<tr>
<td>__m128i _mm_cvtepu8_epi64(__m128i a)</td>
<td>Zero extend 2 bytes into 2 quad words</td>
<td>PMOVZXBQ</td>
</tr>
<tr>
<td>__m128i</td>
<td>Zero extend 8 bytes into 8 words</td>
<td>PMOVZXBW</td>
</tr>
</tbody>
</table>
Intrinsic Syntax | Operation | Corresponding SSE4 Instruction
--- | --- | ---
_mm_cvtepu8_epi16(__m128i a) | 8 word | 
__m128i_mm_cvtepu32_epi64(__m128i a) | Zero extend 2 double words into 2 quad words | PMOVZXDQ
__m128i_mm_cvtepu16_epi32(__m128i a) | Zero extend 4 words into 4 double words | PMOVZXWD
__m128i_mm_cvtepu16_epi64(__m128i a) | Zero extend 2 words into 2 quad words | PMOVZXWQ

Packed Integer Min/Max Intrinsics for Streaming SIMD Extensions 4

These intrinsics compare packed integers in the destination operand and the source operand, and return the minimum or maximum for each packed operand in the destination operand.

Intrinsic Syntax | Operation | Corresponding SSE4 Instruction
--- | --- | ---
__m128i_mm_max_epi8( __m128i a, __m128i b) | Calculates maximum of signed packed integer bytes | PMAXSB
__m128i_mm_max_epi32( __m128i a, __m128i b) | Calculates maximum of signed packed integer double words | PMAXSD
<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i _mm_max_epu32( __m128i a, __m128i b)</td>
<td>Calculates maximum of unsigned packed integer double words</td>
<td>PMAXUD</td>
</tr>
<tr>
<td>__m128i _mm_max_epu16( __m128i a, __m128i b)</td>
<td>Calculates maximum of unsigned packed integer words</td>
<td>PMAXUW</td>
</tr>
<tr>
<td>__m128i _mm_min_epi8( __m128i a, __m128i b)</td>
<td>Calculates minimum of signed packed integer bytes</td>
<td>PMINSB</td>
</tr>
<tr>
<td>__m128i _mm_min_epi32( __m128i a, __m128i b)</td>
<td>Calculates minimum of signed packed integer double words</td>
<td>PMINSD</td>
</tr>
<tr>
<td>__m128i _mm_min_epu32( __m128i a, __m128i b)</td>
<td>Calculates minimum of unsigned packed integer double words</td>
<td>PMINUD</td>
</tr>
<tr>
<td>__m128i _mm_min_epu16( __m128i a, __m128i b)</td>
<td>Calculates minimum of unsigned packed integer words</td>
<td>PMINUW</td>
</tr>
</tbody>
</table>

Floating Point Rounding Intrinsics for Streaming SIMD Extensions 4

These rounding intrinsics cover scalar and packed single-precision and double precision floating-point operands.

The floor and ceil intrinsics correspond to the definitions of floor and ceil in the ISO 9899:1999 standard for the C programming language.
<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128d _mm_round_pd(__m128d s1, int iRoundMode)</td>
<td>Packed float double precision rounding</td>
<td>ROUNDPD</td>
</tr>
<tr>
<td>__m128d _mm_floor_pd(__m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128d _mm_ceil_pd(__m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_round_ps(__m128 s1, int iRoundMode)</td>
<td>Packed float single precision rounding</td>
<td>ROUNDPS</td>
</tr>
<tr>
<td>__m128 _mm_floor_ps(__m128 s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_ceil_ps(__m128 s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128d _mm_round_sd(__m128d dst, __m128d s1, int iRoundMode)</td>
<td>Single float double precision rounding</td>
<td>ROUNDSD</td>
</tr>
<tr>
<td>__m128d _mm_floor_sd(__m128d dst, __m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128d _mm_ceil_sd(__m128d dst, __m128d s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_round_ss(__m128 dst, __m128d s1, int iRoundMode)</td>
<td>Single float single precision rounding</td>
<td>ROUNDSS</td>
</tr>
<tr>
<td>__m128 _mm_floor_ss(__m128d dst, __m128 s1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>__m128 _mm_ceil_ss(__m128d dst, __m128 s1)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
These DWORD multiply intrinsics are designed to aid vectorization. They enable four simultaneous 32 bit by 32 bit multiplies.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i _mm_mul_epi32(__m128i a, __m128i b)</td>
<td>Packed integer 32-bit multiplication of 2 low pairs of operands producing two 64-bit results</td>
<td>PMULDQ</td>
</tr>
<tr>
<td>__m128i _mm_mullo_epi32(__m128i a, __m128i b)</td>
<td>Packed integer 32-bit multiplication with truncation of upper halves of results</td>
<td>PMULLD</td>
</tr>
</tbody>
</table>

Register Insertion/Extraction Intrinsics for Streaming SIMD Extensions 4

These intrinsics enable data insertion and extraction between general purpose registers and XMM registers.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128 _mm_insert_ps(__m128 dst, __m128 src, const int ndx)</td>
<td>Insert single precision float into packed single precision array element selected by index</td>
<td>INSERTPS</td>
</tr>
<tr>
<td>int _mm_extract_ps(__m128 src, const int ndx)</td>
<td>Extract single precision float from packed single</td>
<td>EXTRACTPS</td>
</tr>
</tbody>
</table>
### Intrinsic Syntax

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>int _mm_extract_epi8(__m128i src, const int ndx)</td>
<td>Extract integer byte from packed integer array element selected by index</td>
<td>PEXTRB</td>
</tr>
<tr>
<td>int _mm_extract_epi32(__m128i src, const int ndx)</td>
<td>Extract integer double word from packed integer array element selected by index</td>
<td>PEXTRD</td>
</tr>
<tr>
<td>__int64 _mm_extract_epi64(__m128i src, const int ndx)</td>
<td>Extract integer quad word from packed integer array element selected by index</td>
<td>PEXTRQ</td>
</tr>
<tr>
<td>int _mm_extract_epi16(__m128i src, int ndx)</td>
<td>Extract integer word from packed integer array element selected by index</td>
<td>PEXTRW</td>
</tr>
<tr>
<td>__m128i _mm_insert_epi8(__m128i s1, int s2, const int ndx)</td>
<td>Insert integer byte into packed integer array element selected by index</td>
<td>PINSRB</td>
</tr>
<tr>
<td>__m128i _mm_insert_epi32(__m128i s1, int s2, const int ndx)</td>
<td>Insert integer double word into packed integer array element selected by index</td>
<td>PINSRD</td>
</tr>
</tbody>
</table>
### Intrinsic Syntax

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__m128i</td>
<td>Insert integer quad word into packed integer array</td>
<td>PINSRQ</td>
</tr>
<tr>
<td>__mm_insert_epi64(__m128i s2, int s, const int ndx)</td>
<td>Insert integer quad word into packed integer array</td>
<td>element selected by index</td>
</tr>
</tbody>
</table>

### Test Intrinsics for Streaming SIMD Extensions 4

These intrinsics perform packed integer 128-bit comparisons.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>__mm_testc_si128</td>
<td>Check for all ones in specified bits of a 128-bit value</td>
<td>PTEST</td>
</tr>
<tr>
<td>__mm_testz_si128</td>
<td>Check for all zeros in specified bits of a 128-bit value</td>
<td>PTEST</td>
</tr>
<tr>
<td>__mm_testnzc_si128</td>
<td>Check for at least one zero and at least one one in specified bits of a 128-bit value</td>
<td>PTEST</td>
</tr>
</tbody>
</table>

```c
int _mm_testz_si128 (__m128i s1, __m128i s2)
```

Returns 1 if the bitwise AND of s1 and s2 is all zero, else returns 0

```c
int _mm_testc_si128 (__m128i s1, __m128i s2)
```

Returns 1 if the bitwise AND of s2 ANDNOT of s1 is all ones, else returns 0.
int _mm_testnzc_si128 (__m128i s1, __m128i s2)

Same as (!_mm_testz) && (!_mm_testc)

Packed DWORD to Unsigned WORD Intrinsic for Streaming SIMD Extensions 4
__m128i _mm_packus_epi32(__m128i m1, __m128i m2);
Converts 8 packed signed DWORDs into 8 packed unsigned WORDs, using unsigned saturation to handle overflow condition.
Corresponding SSE4 instruction: PACKUSDW

Packed Compare for Equal for Streaming SIMD Extensions 4
__m128i _mm_cmpeq_epi64(__m128i a, __m128i b)
Performs a packed integer 64-bit comparison for equality. The intrinsic zeroes or fills with ones the corresponding parts of the result.
Corresponding SSE4 instruction: PCMPEQQ

Cacheability Support Intrinsic for Streaming SIMD Extensions 4
extern __m128i _mm_stream_load_si128(__m128i* v1);
Loads _m128 data from a 16-byte aligned address (v1) to the destination operand (m128i) without polluting the caches.
Corresponding SSE4 instruction: MOVNTDQA

Streaming SIMD Extensions 4: Efficient Accelerated String and Text Processing

Overview: Streaming SIMD Extensions 4 Efficient Accelerated String and Text Processing
The intrinsics in this section correspond to Streaming SIMD Extensions 4 (SSE4) Efficient Accelerated String and Text Processing instructions. These instructions include:
- Packed Comparison Intrinsics for Streaming SIMD Extensions 4
- Application Targeted Accelerators Intrinsics
The prototypes for these intrinsics are in the nmmintrin.h file.
**Packed Comparison Intrinsics for Streaming SIMD Extensions 4**

These intrinsics perform packed comparisons. They correspond to SSE4 instructions. For intrinsics that could map to more than one instruction, the Intel® C++ Compiler selects the instruction to generate.

<table>
<thead>
<tr>
<th>Intrinsic Name</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>_mm_cmpestri</td>
<td>Packed comparison, generates index</td>
<td>PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestrm</td>
<td>Packed comparison, generates mask</td>
<td>PCMPESTRM</td>
</tr>
<tr>
<td>_mm_cmpistri</td>
<td>Packed comparison, generates index</td>
<td>PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistrm</td>
<td>Packed comparison, generates mask</td>
<td>PCMPISTRM</td>
</tr>
<tr>
<td>_mm_cmpestrz</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestrc</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestrs</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestro</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpestra</td>
<td>Packed comparison</td>
<td>PCMPESTRM or PCMPESTRI</td>
</tr>
<tr>
<td>_mm_cmpistrz</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistrc</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
<tr>
<td>Intrinsic Name</td>
<td>Operation</td>
<td>Corresponding SSE4 Instruction</td>
</tr>
<tr>
<td>-------------------</td>
<td>--------------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>_mm_cmpistri</td>
<td>Packed comparison</td>
<td>PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistro</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
<tr>
<td>_mm_cmpistra</td>
<td>Packed comparison</td>
<td>PCMPISTRM or PCMPISTRI</td>
</tr>
</tbody>
</table>

```c
int _mm_cmpestri(__m128i src1, int len1, __m128i src2, int len2, const int mode)
```

This intrinsic performs a packed comparison of string data with explicit lengths, generating an index and storing the result in ECX.

```c
__m128i _mm_cmpestrm(__m128i src1, int len1, __m128i src2, int len2, const int mode)
```

This intrinsic performs a packed comparison of string data with explicit lengths, generating a mask and storing the result in XMM0.

```c
int _mm_cmpestrz(__m128i src1, int len1, __m128i src2, int len2, const int mode);
```

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if ZFlag == 1, otherwise 0.
int _mm_cmpestrc(__m128i src1, int len1, __m128i src2, int len2, const int mode);

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if CFlag == 1, otherwise 0.

int _mm_cmpestro(__m128i src1, int len1, __m128i src2, int len2, const int mode);

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if OFlag == 1, otherwise 0.

int _mm_cmpestra(__m128i src1, int len1, __m128i src2, int len2, const int mode);

This intrinsic performs a packed comparison of string data with explicit lengths. Returns 1 if CFlag == 0 and ZFlag == 0, otherwise 0.

int _mm_cmpistrz(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (ZFlag == 1), otherwise 0.

int _mm_cmpistrc(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (CFlag == 1), otherwise 0.

int _mm_cmpistrs(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (SFlag == 1), otherwise 0.

int _mm_cmpistro(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (OFlag == 1), otherwise 0.
int _mm_cmpistra(__m128i src1, __m128i src2, const int mode);

This intrinsic performs a packed comparison of string data with implicit lengths. Returns 1 if (ZFlag == 0 and CFlag == 0), otherwise 0.

**Application Targeted Accelerators Intrinsics**

Application Targeted Accelerators extend the capabilities of Intel architecture by adding performance-optimized, low-latency, lower power fixed-function accelerators on the processor die to benefit specific applications. The prototypes for application targeted accelerator intrinsics are in the file `nmmintrin.h`.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>int _mm_popcnt_u32(unsigned int v)</td>
<td>Counts number of set bits in a data operation</td>
<td>POPCNT</td>
</tr>
<tr>
<td>int _mm_popcnt_u64(unsigned __int64 v)</td>
<td>Counts number of set bits in a data operation</td>
<td>POPCNT</td>
</tr>
<tr>
<td>unsigned int _mm_crc32_u8(unsigned int crc, unsigned char v)</td>
<td>Accumulates cyclic redundancy check</td>
<td>CRC32</td>
</tr>
<tr>
<td>unsigned int _mm_crc32_u16(unsigned int crc, unsigned short v)</td>
<td>Performs cyclic redundancy check</td>
<td>CRC32</td>
</tr>
<tr>
<td>unsigned int _mm_crc32_u32(unsigned int crc, unsigned int v)</td>
<td>Performs cyclic redundancy check</td>
<td>CRC32</td>
</tr>
</tbody>
</table>
### Intrinsic Syntax

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Operation</th>
<th>Corresponding SSE4 Instruction</th>
</tr>
</thead>
</table>
| unsigned __int64  
_mm_crc32_u64(unsigned __int64 crc, unsigned __int64 v) | Performs cyclic redundancy CRC32 check | |
| unsigned int _mm_crc32_u8(unsighned int crc, unsigned char v); | | |
| unsigned int _mm_crc32_u16(unsighned int crc, unsigned short v); | | |
| unsigned int _mm_crc32_u32(unsighned int crc, unsigned int v); | | |
| unsigned __int64 _mm_crc32_u64(unsigned __int64 crc, unsigned __int64 v); | | |

Starting with an initial value in the first operand, accumulates a CRC32 value for the second operand and stores the result in the destination operand.

Accumulates CRC32 on r/m8.

Accumulates CRC32 on r/m16.

Accumulates CRC32 on r/m32.

Accumulates CRC32 on r/m64.

### Intrinsics for Use Across All Intel Architectures

**Overview: Intrinsics For All Intel Architectures**
Most of the intrinsics documented in this section function across all Intel architectures, namely IA-32, Intel(R) 64, and IA-64 architectures. Some of them function across two Intel architectures.

The intrinsics are offered as a convenience to the programmer and are categorized as follows:

- **Integer Arithmetic Intrinsics**
- **Floating-Point Intrinsics**
- **String and Block Copy Intrinsics** (only for IA-32 and Intel(R) 64 architectures)
- **Miscellaneous Intrinsics**

### Integer Arithmetic Intrinsics

The following table lists and describes integer arithmetic intrinsics that you can use across all Intel architectures.

<table>
<thead>
<tr>
<th>Intrinsic Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int abs(int)</td>
<td>Returns the absolute value of an integer.</td>
</tr>
<tr>
<td>long labs(long)</td>
<td>Returns the absolute value of a long integer.</td>
</tr>
<tr>
<td>unsigned long _lrotl(unsigned long value, int shift)</td>
<td>Implements 64-bit left rotate of value by shift positions.</td>
</tr>
<tr>
<td>unsigned long _lrotr(unsigned long value, int shift)</td>
<td>Implements 64-bit right rotate of value by shift positions.</td>
</tr>
<tr>
<td>unsigned int _rotl(unsigned int value, int shift)</td>
<td>Implements 32-bit left rotate of value by shift positions.</td>
</tr>
<tr>
<td>unsigned int _rotr(unsigned int value, int shift)</td>
<td>Implements 32-bit right rotate of value by shift positions.</td>
</tr>
<tr>
<td>unsigned short _rotwl(unsigned short value,</td>
<td>Implements 16-bit left rotate of value by shift positions.</td>
</tr>
</tbody>
</table>
Intrinsic Syntax | Description
--- | ---
```c
int shift)
```
by `shift` positions. These intrinsics are not supported on IA-64 architecture-based platforms.

```c
unsigned short
_rotwr(unsigned short value,
int shift)
```
Implements 16-bit right rotate of `value` by `shift` positions. These intrinsics are not supported on IA-64 architecture-based platforms.

**Note**

Passing a constant shift value in the rotate intrinsics results in higher performance.

**Floating-point Intrinsics**

The following table lists and describes floating point intrinsics that you can use across all Intel architectures.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>double fabs(double)</code></td>
<td>Returns the absolute value of a floating-point value.</td>
</tr>
<tr>
<td><code>double log(double)</code></td>
<td>Returns the natural logarithm <code>ln(x)</code>, <code>x&gt;0</code>, with double precision.</td>
</tr>
<tr>
<td><code>float logf(float)</code></td>
<td>Returns the natural logarithm <code>ln(x)</code>, <code>x&gt;0</code>, with single precision.</td>
</tr>
<tr>
<td><code>double log10(double)</code></td>
<td>Returns the base 10 logarithm <code>log10(x)</code>, <code>x&gt;0</code>, with double precision.</td>
</tr>
<tr>
<td><code>float log10f(float)</code></td>
<td>Returns the base 10 logarithm <code>log10(x)</code>, <code>x&gt;0</code>, with single precision.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double exp(double)</td>
<td>Returns the exponential function with double precision.</td>
</tr>
<tr>
<td>float expf(float)</td>
<td>Returns the exponential function with single precision.</td>
</tr>
<tr>
<td>double pow(double, double)</td>
<td>Returns the value of x to the power y with double precision.</td>
</tr>
<tr>
<td>float powf(float, float)</td>
<td>Returns the value of x to the power y with single precision.</td>
</tr>
<tr>
<td>double sin(double)</td>
<td>Returns the sine of x with double precision.</td>
</tr>
<tr>
<td>float sinf(float)</td>
<td>Returns the sine of x with single precision.</td>
</tr>
<tr>
<td>double cos(double)</td>
<td>Returns the cosine of x with double precision.</td>
</tr>
<tr>
<td>float cosf(float)</td>
<td>Returns the cosine of x with single precision.</td>
</tr>
<tr>
<td>double tan(double)</td>
<td>Returns the tangent of x with double precision.</td>
</tr>
<tr>
<td>float tanf(float)</td>
<td>Returns the tangent of x with single precision.</td>
</tr>
<tr>
<td>double acos(double)</td>
<td>Returns the inverse cosine of x with double precision</td>
</tr>
<tr>
<td>float acosf(float)</td>
<td>Returns the inverse cosine of x with single precision</td>
</tr>
<tr>
<td>double acosh(double)</td>
<td>Compute the inverse hyperbolic cosine of the argument with double precision.</td>
</tr>
<tr>
<td>float acoshf(float)</td>
<td>Compute the inverse hyperbolic cosine of the argument with single precision.</td>
</tr>
<tr>
<td>double asin(double)</td>
<td>Compute inverse sine of the argument with double precision.</td>
</tr>
<tr>
<td>float asinf(float)</td>
<td>Compute inverse sine of the argument with single precision.</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>double asinh(double)</td>
<td>Compute inverse hyperbolic sine of the argument with double precision.</td>
</tr>
<tr>
<td>float asinhf(float)</td>
<td>Compute inverse hyperbolic sine of the argument with single precision.</td>
</tr>
<tr>
<td>double atan(double)</td>
<td>Compute inverse tangent of the argument with double precision.</td>
</tr>
<tr>
<td>float atanf(float)</td>
<td>Compute inverse tangent of the argument with single precision.</td>
</tr>
<tr>
<td>double atanh(double)</td>
<td>Compute inverse hyperbolic tangent of the argument with double precision.</td>
</tr>
<tr>
<td>float atanhf(float)</td>
<td>Compute inverse hyperbolic tangent of the argument with single precision.</td>
</tr>
<tr>
<td>double cabs(double complex z)</td>
<td>Computes absolute value of complex number. The intrinsic argument is a complex number made up of two double precision elements, one real and one imaginary. The input parameter z is made up of two values of double type passed together as a single argument.</td>
</tr>
<tr>
<td>float cabsf(float complex z)</td>
<td>Computes absolute value of complex number. The intrinsic argument is a complex number made up of two single precision elements, one real and one imaginary. The input parameter z is made up of two values of float type passed together as a single argument.</td>
</tr>
<tr>
<td>double ceil(double)</td>
<td>Computes smallest integral value of double</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>float ceilf(float)</td>
<td>Computes smallest integral value of single precision argument not less than the argument.</td>
</tr>
<tr>
<td>double cosh(double)</td>
<td>Computes the hyperbolic cosine of double precision argument.</td>
</tr>
<tr>
<td>float coshf(float)</td>
<td>Computes the hyperbolic cosine of single precision argument.</td>
</tr>
<tr>
<td>float fabsf(float)</td>
<td>Computes absolute value of single precision argument.</td>
</tr>
<tr>
<td>double floor(double)</td>
<td>Computes the largest integral value of the double precision argument not greater than the argument.</td>
</tr>
<tr>
<td>float floorf(float)</td>
<td>Computes the largest integral value of the single precision argument not greater than the argument.</td>
</tr>
<tr>
<td>double fmod(double)</td>
<td>Computes the floating-point remainder of the division of the first argument by the second argument with double precision.</td>
</tr>
<tr>
<td>float fmodf(float)</td>
<td>Computes the floating-point remainder of the division of the first argument by the second argument with single precision.</td>
</tr>
<tr>
<td>double hypot(double, double)</td>
<td>Computes the length of the hypotenuse of a right angled triangle with double precision.</td>
</tr>
<tr>
<td>float hypotf(float, float)</td>
<td>Computes the length of the hypotenuse of a right angled triangle with single precision.</td>
</tr>
<tr>
<td>double rint(double)</td>
<td>Computes the integral value represented as</td>
</tr>
</tbody>
</table>
Intel(R) C++ Compiler User and Reference Guides

Intrinsic | Description
--- | ---
float rintf(float) | Computes the integral value represented with single precision using the IEEE rounding mode.
double sinh(double) | Computes the hyperbolic sine of the double precision argument.
float sinhf(float) | Computes the hyperbolic sine of the single precision argument.
float sqrtf(float) | Computes the square root of the single precision argument.
double tanh(double) | Computes the hyperbolic tangent of the double precision argument.
float tanhf(float) | Computes the hyperbolic tangent of the single precision argument.

String and Block Copy Intrinsics

The following table lists and describes string and block copy intrinsics that you can use on systems based on IA-32 and Intel® 64 architectures.

**Note**

On systems based on IA-64 architectures, you can perform string and block copy operations using the String and Block Copy intrinsics implemented as regular function calls.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char * _strset(char *, _int32)</td>
<td>Sets all characters in a string to a fixed value.</td>
</tr>
</tbody>
</table>
### Intrinsic

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>int memcmp(const void *cs, const void *ct, size_t n)</code></td>
<td>Compares two regions of memory. Return &lt;0 if <code>cs&lt;ct</code>, 0 if <code>cs=ct</code>, or &gt;0 if <code>cs&gt;ct</code>.</td>
</tr>
<tr>
<td><code>void *memcpy(void *s, const void *ct, size_t n)</code></td>
<td>Copies from memory. Returns <code>s</code>.</td>
</tr>
<tr>
<td><code>void *memset(void * s, int c, size_t n)</code></td>
<td>Sets memory to a fixed value. Returns <code>s</code>.</td>
</tr>
<tr>
<td><code>char *strcat(char * s, const char * ct)</code></td>
<td>Appends to a string. Returns <code>s</code>.</td>
</tr>
<tr>
<td><code>int strcmp(const char *, const char *)</code></td>
<td>Compares two strings. Return &lt;0 if <code>cs&lt;ct</code>, 0 if <code>cs=ct</code>, or &gt;0 if <code>cs&gt;ct</code>.</td>
</tr>
<tr>
<td><code>char *strcpy(char * s, const char * ct)</code></td>
<td>Copies a string. Returns <code>s</code>.</td>
</tr>
<tr>
<td><code>size_t strlen(const char * cs)</code></td>
<td>Returns the length of string <code>cs</code>.</td>
</tr>
<tr>
<td><code>int strncmp(char *, char *, int)</code></td>
<td>Compare two strings, but only specified number of characters.</td>
</tr>
<tr>
<td><code>int strncpy(char *, char *, int)</code></td>
<td>Copies a string, but only specified number of characters.</td>
</tr>
</tbody>
</table>

### Miscellaneous Intrinsics

The following tables list and describe intrinsics that you can use across all Intel architectures, except where noted.

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>_abnormal_termination(void)</code></td>
<td>Can be invoked only by termination</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>__cpu1d</td>
<td>Queries the processor for information about processor type and supported features. The Intel(R) C++ Compiler supports the Microsoft* implementation of this intrinsic. See the Microsoft documentation for details.</td>
</tr>
<tr>
<td>void * __alloca(int)</td>
<td>Allocates memory in the local stack frame. The memory is automatically freed upon return from the function.</td>
</tr>
<tr>
<td>int _bit_scan_forward(int x)</td>
<td>Returns the bit index of the least significant set bit of x. If x is 0, the result is undefined.</td>
</tr>
<tr>
<td>int _bit_scan_reverse(int)</td>
<td>Returns the bit index of the most significant set bit of x. If x is 0, the result is undefined.</td>
</tr>
<tr>
<td>int _bswap(int)</td>
<td>Reverses the byte order of x. Bits 0-7 are swapped with bits 24-31, and bits 8-15 are swapped with bits 16-23.</td>
</tr>
<tr>
<td>int _BitScanForward64(int x)</td>
<td>Returns the bit index of the least significant set bit of x. If x is 0, the result is undefined.</td>
</tr>
<tr>
<td>int _BitScanReverse64(int x)</td>
<td>Returns the bit index of the most significant set bit of x. If x is 0, the result is undefined.</td>
</tr>
<tr>
<td>int _bswap64(int x)</td>
<td>Reverses the byte order of x.</td>
</tr>
<tr>
<td>unsigned int __cacheSize(unsigned int cacheLevel)</td>
<td>__cacheSize(n) returns the size in bytes of the cache at level n. 1 represents the first-level cache. 0 is returned for a non-existent</td>
</tr>
<tr>
<td>Intrinsic</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>_exception_code(void)</td>
<td>Returns the exception code.</td>
</tr>
<tr>
<td>_exception_info(void)</td>
<td>Returns the exception information.</td>
</tr>
<tr>
<td>void _enable(void)</td>
<td>Enables the interrupt.</td>
</tr>
<tr>
<td>void _disable(void)</td>
<td>Disables the interrupt.</td>
</tr>
<tr>
<td>int _in_byte(int)</td>
<td>Intrinsic that maps to the IA-32 instruction IN. Transfer data byte from port specified by argument.</td>
</tr>
<tr>
<td>int _in_dword(int)</td>
<td>Intrinsic that maps to the IA-32 instruction IN. Transfer double word from port specified by argument.</td>
</tr>
<tr>
<td>int _in_word(int)</td>
<td>Intrinsic that maps to the IA-32 instruction IN. Transfer word from port specified by argument.</td>
</tr>
<tr>
<td>int _inp(int)</td>
<td>Same as _in_byte</td>
</tr>
<tr>
<td>int _inpd(int)</td>
<td>Same as _in_dword</td>
</tr>
<tr>
<td>int _inpw(int)</td>
<td>Same as _in_word</td>
</tr>
<tr>
<td>int _out_byte(int, int)</td>
<td>Intrinsic that maps to the IA-32 instruction OUT. Transfer data byte in second argument to port specified by first argument.</td>
</tr>
<tr>
<td>int _out_dword(int, int)</td>
<td>Intrinsic that maps to the IA-32 instruction OUT. Transfer double word in second argument.</td>
</tr>
</tbody>
</table>
### Intrinsic

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>int _out_word(int, int)</td>
<td>Intrinsic that maps to the IA-32 instruction OUT. Transfer word in second argument to port specified by first argument.</td>
</tr>
<tr>
<td>int _outp(int, int)</td>
<td>Same as _out_byte</td>
</tr>
<tr>
<td>int _outpw(int, int)</td>
<td>Same as _out_word</td>
</tr>
<tr>
<td>int _outpd(int, int)</td>
<td>Same as _out_dword</td>
</tr>
<tr>
<td>int _popcnt32(int x)</td>
<td>Returns the number of set bits in x.</td>
</tr>
<tr>
<td>__int64 _rdpmc(int p)</td>
<td>Returns the current value of the 40-bit performance monitoring counter specified by p.</td>
</tr>
</tbody>
</table>

**Intrinsics for IA-32 and Intel® 64 Architectures Only**

<table>
<thead>
<tr>
<th>Intrinsic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__int64 _rdtsc(void)</td>
<td>Returns the current value of the processor’s 64-bit time stamp counter. This intrinsic is not implemented on systems based on IA-64 architecture.</td>
</tr>
<tr>
<td>int _setjmp(jmp_buf)</td>
<td>A fast version of setjmp(), which bypasses the termination handling. Saves the callee-save registers, stack pointer and return address. This intrinsic is not implemented on systems based on IA-64 architecture.</td>
</tr>
</tbody>
</table>
This reference for the Intel® C++ Compiler includes the following sections:

- Intel C++ Compiler Pragmas
- Intel® Math Library
- Intel C++ Class Libraries

Intel(R) C++ Compiler Pragmas

Overview: Intel® C++ Compiler Pragmas

Pragmas are directives that provide instructions to the compiler for use in specific cases. For example, you can use the `novector` pragma to specify that a loop should never be vectorized. The keyword `#pragma` is standard in the C++ language, but individual pragmas are machine-specific or operating system-specific, and vary by compiler.

Some pragmas provide the same functionality as compiler options. Pragmas override behavior specified by compiler options.

The Intel® C++ Compiler pragmas are categorized as follows:

- Intel-Specific Pragmas - pragmas developed or modified by Intel to work specifically with the Intel® C++ Compiler
- Intel Supported Pragmas - pragmas developed by external sources that are supported by the Intel® C++ Compiler for compatibility reasons

Using Pragmas

You enter pragmas into your C++ source code using the following syntax:

```
#pragma <pragma name>
```

IndividualPragma Descriptions

Each pragma description has the following details:

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Short Description</td>
<td>Contains a brief description of what the pragma does</td>
</tr>
<tr>
<td>Syntax</td>
<td>Contains the pragma syntax</td>
</tr>
</tbody>
</table>
### Intel-Specific Pragmas

The Intel-specific C++ compiler pragmas described in the Intel-SpecificPragma reference are listed below. Click on the pragmas for a more detailed description.

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_section</td>
<td>allocates variable in specified section</td>
</tr>
<tr>
<td>distribute_point</td>
<td>instructs the compiler to prefer loop distribution at the location indicated</td>
</tr>
<tr>
<td>intel_omp_task</td>
<td>specifies a unit of work, potentially executed by a different thread</td>
</tr>
<tr>
<td>intel_omp_taskq</td>
<td>specifies a unit of work, potentially executed by a different thread</td>
</tr>
<tr>
<td>ivdep</td>
<td>instructs the compiler to ignore assumed vector dependencies</td>
</tr>
<tr>
<td>loop_count</td>
<td>indicates the loop count is likely to be an integer</td>
</tr>
<tr>
<td>memref_control</td>
<td>provides a method for controlling load latency at the variable level</td>
</tr>
<tr>
<td>novector</td>
<td>specifies that the loop should never be vectorized</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>optimize</td>
<td>enables or disables optimizations for specific functions; provides some degree of compatibility with Microsoft's implementation of optimize pragma</td>
</tr>
<tr>
<td>optimization_level</td>
<td>enables control of optimization for a specific function</td>
</tr>
<tr>
<td>parallel/noparallel</td>
<td>facilitates auto-parallelization of an immediately following loop; using keyword [always] forces the compiler to auto-parallelize; noparallel pragma prevents auto-parallelization</td>
</tr>
<tr>
<td>prefetch/noprefetch</td>
<td>asserts that the data prefetches are generated or not generated for some memory references</td>
</tr>
<tr>
<td>swp/noswp</td>
<td>swp indicates preference for loops to be software pipelined; noswp indicates the loops not to be software pipelined</td>
</tr>
<tr>
<td>unroll/nounroll</td>
<td>instructs the compiler the number of times to unroll/not to unroll a loop</td>
</tr>
<tr>
<td>unroll_and_jam/nounroll_and_jam</td>
<td>instructs the compiler to partially unroll higher loops and the resulting loops back together. Specifying the nounroll_and_jam pragma prevents unrolling and jamming of loops.</td>
</tr>
<tr>
<td>unused</td>
<td>describes variables that are unused (warnings not generated)</td>
</tr>
<tr>
<td>vector</td>
<td>indicates to the compiler that the loop should be vectorized according to the arguments: always/aligned/unaligned/nontemporal/temporal</td>
</tr>
</tbody>
</table>

### Intel-specific Pragma Reference

**alloc_section**
Allocates variable in specified section. Controls section attribute specification for variables.

Syntax

```
#pragma alloc_section(var1, var2, ..., "r;attribute-list")
```

Arguments

- `var` variable that can be used to define a symbol in the section

"r;attribute-list" a comma-separated list of attributes; defined values are: 'short' and 'long'

Description

The `alloc_section` pragma places the listed variables, `var1`, `var2`, etc., in the specified section. This pragma controls section attribute specification for variables. The compiler decides whether the variable, as defined by `var1/var2/etc`, should go to a "data", "bss", or "rdata" section.
The section name is enclosed in double quotation marks. It should be previously introduced into the program using `#pragma section`. The list of comma-separated variable names follows the section name after a separating comma. All listed variables are necessarily defined before this pragma, in the same translation unit and in the same scope. The variables have static storage; their linkage does not matter in C modules, but in C++ modules they are defined with the extern "C" linkage specification.

Example

The following example illustrates how to use the pragma.

```
#pragma alloc_section(var1, "r;short")
int var1 = 20;
#pragma alloc_section(var2, "r;short")
extern int var2;
```

`distribute_point`

Instructs the compiler to prefer loop distribution at the location indicated.

Syntax

```
#pragma distribute_point
```

Arguments

None

Description

The `distribute_point` pragma is used to suggest to the compiler to split large loops into smaller ones; this is particularly useful in cases where optimizations like software-pipelining (SWP) or vectorization cannot take place due to excessive register usage.

Using `distribute_point` pragma for a loop distribution strategy enables software pipelining for the new, smaller loops in the IA-64 architecture. By splitting a loop into smaller segments, it is possible to get each smaller loop or at least one of the smaller loops to SWP or vectorize.
• When the pragma is placed inside a loop, the compiler distributes the loop at that point. All loop-carried dependencies are ignored.
• When inside the loop, pragmas cannot be placed within an `if` statement.
• When the pragma is placed outside the loop, the compiler distributes the loop based on an internal heuristic. The compiler determines where to distribute the loops and observes data dependency. If the pragmas are placed inside the loop, the compiler supports multiple instances of the pragma.

Example

Example 1: Using `distribute_point` pragma outside the loop

The following example uses the `distribute_point` pragma outside the loop.

```cpp
#define NUM 1024
void loop_distribution_pragma1(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
    int i;
    // Before distribution or splitting the loop
    #pragma distribute_point
    for (i=0; i< NUM; i++) {
        a[i] = a[i] + i;
        b[i] = b[i] + i;
        c[i] = c[i] + i;
        x[i] = x[i] + i;
        y[i] = y[i] + i;
        z[i] = z[i] + i;
    }
}
```

Example 2: Using `distribute_point` pragma inside the loop

The following example uses the `distribute_point` pragma inside the loop.

```cpp
#define NUM 1024
void loop_distribution_pragma2(
    double a[NUM], double b[NUM], double c[NUM],
    double x[NUM], double y[NUM], double z[NUM] )
{
    int i;
    // After distribution or splitting the loop.
    for (i=0; i< NUM; i++) {
        a[i] = a[i] + i;
        b[i] = b[i] + i;
        c[i] = c[i] + i;
        #pragma distribute_point
        x[i] = x[i] + i;
        y[i] = y[i] + i;
        z[i] = z[i] + i;
    }
}
Example 3: Using `distribute_point` pragma inside and outside the loop

The following example shows how to use the `distribute_point` pragma, first outside the loop and then inside the loop.

```cpp
void dist1(int a[], int b[], int c[], int d[])
{
    #pragma distribute_point
    // Compiler will automatically decide where to
distribute. Data dependency is observed.
    for (int i=1; i<1000; i++) {
        b[i] = a[i] + 1;
        c[i] = a[i] + b[i];
        d[i] = c[i] + 1;
    }
}

void dist2(int a[], int b[], int c[], int d[])
{
    for (int i=1; i<1000; i++) {
        b[i] = a[i] + 1;
        #pragma distribute_point
        // Distribution will start here,
        // ignoring all loop-carried dependency.
        c[i] = a[i] + b[i];
        d[i] = c[i] + 1;
    }
}
```

`intel_omp_task`

Specifies a unit of work, potentially executed by a different thread.

**Syntax**

```
#pragma intel_omp_task
[clause[,]clause]...
structured-block
```

**Where clause can be any of the following:**

- `private( variable list )`
- `captureprivate( variable-list )`

**Arguments**

<table>
<thead>
<tr>
<th><code>private(variable-list)</code></th>
<th>The <code>private</code> clause creates a</th>
</tr>
</thead>
</table>
private, default-constructed version for each object in `variable-list` for the task.

The original object referenced by the variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and has an indeterminate value upon exit from the construct.

`captureprivate(variable-list)`

The `captureprivate` clause creates a private, copy-constructed version for each object in `variable-list` for the task.
The `intel_omp_task` pragma specifies a unit of work, potentially executed by a different thread.

Example

For an example on how to use `intel_omp_task` pragma see topic Workqueueing Example Function in Optimizing Applications>Using Parallelism: OpenMP* Support>Intel(R) Workqueing Model.

See Also

Workqueueing Constructs in Optimizing Applications>Using Parallelism: OpenMP* Support>Intel(R) Workqueing Model

`intel_omp_taskq`

Specifies an environment for the while loop in which to enqueue the units of work specified by the enclosed `task` pragma.

Syntax
#pragma intel_omp_taskq
[clause[[],clause]...]  
structured-block

where clause can be any of the following:

- private (variable-list)
- firstprivate (variable-list)
- lastprivate (variable-list)
- reduction (operator : variable-list)
- ordered
- nowait

**Arguments**

| **private**(variable-list) | The private clause creates a private, default-constructed version for each object in variable-list for the taskq. It also implies captureprivate on each enclosed task. The original object referenced by each variable has an indeterminate value upon entry to the construct, must not be modified within |
**firstprivate (variable-list)**

The `firstprivate` clause creates a private, copy-constructed version for each object in `variable-list` for the taskq. It also implies `captureprivate` on each enclosed task. The original object referenced by each variable must not be modified within the dynamic extent of the construct and has an indeterminate value upon exit.

The dynamic extent of the construct, and has an indeterminate value upon exit from the construct.
The `lastprivate` clause creates a private, default-constructed version for each object in `variable-list` for the `taskq`. It also implies captureprivate on each enclosed task. The original object referenced by each variable has an indeterminate value upon entry to the construct, must not be modified within the dynamic extent of the construct, and is copy-assigned the value of the object from the last enclosed task after that.

<table>
<thead>
<tr>
<th>lastprivate (variable-list)</th>
</tr>
</thead>
</table>

from the construct.
The reduction clause performs a reduction operation with the given operator in enclosed task constructs for each object in `variable-list`. `operator` and `variable-list` are defined the same as in the OpenMP Specifications.

The ordered clause performs ordered constructs in enclosed task constructs in original sequential execution order. The `taskq` directive, to which the ordered is
**nowait**

The `nowait` clause removes the implied barrier at the end of the `taskq`. Threads may exit the `taskq` construct before completing all the `task` constructs queued within it.

---

**Description**

The `intel_omp_taskq/taskq` pragma specifies the environment within which the enclosed units of work (tasks) are to be executed. From among all the threads that encounter a `taskq` pragma, one is chosen to execute it initially. Conceptually, the `taskq` pragma causes an empty queue to be created by the chosen thread, and then the code inside the `taskq` block is executed single-threaded. All the other threads wait for work to be enqueued on the conceptual queue.

The `task` pragma specifies a unit of work, potentially executed by a different thread. When a `task` pragma is encountered lexically within a `taskq` block, the code inside the `task` block is conceptually enqueued on the queue associated with the `taskq`. The conceptual queue is disbanded when all work enqueued on it finishes, and when the end of the `taskq` block is reached.
Example

For an example on how to use `taskq` pragma see topic Workqueuing Example Function in *Optimizing Applications>Using Parallelism: OpenMP* Support>Intel(R) Workqueing Model.

See Also

Workqueing Constructs in *Optimizing Applications>Using Parallelism: OpenMP* Support>Intel(R) Workqueing Model

dep

Instructs the compiler to ignore assumed vector dependencies.

Syntax

```c
#pragma ivdep
```

Arguments

None

Description

The `ivdep` pragma instructs the compiler to ignore assumed vector dependencies. To ensure correct code, the compiler treats an assumed dependence as a proven dependence, which prevents vectorization. This pragma overrides that decision. Only use this pragma when you know that the assumed loop dependencies are safe to ignore.

Note

- The proven dependencies that prevent vectorization are not ignored, only assumed dependencies are ignored.
- When an `ivdep` pragma is specified for applications designed to run on IA-64 architectures, the `-ivdep-parallel` (Linux* systems) or `/Qivdep-parallel` (Windows* systems) option indicates there is no loop-carried memory
dependency in the loop. This technique is useful for some sparse matrix applications. See Example 2.

Example

**Example 1**
The loop in this example will not vectorize without the ivdep pragma, since the value of $k$ is not known; vectorization would be illegal if $k<0$.

```c
void ignore_vec_dep(int *a, int k, int c, int m)
{
    #pragma ivdep
    for (int i = 0; i < m; i++)
        a[i] = a[i + k] * c;
}
```

The pragma binds only the for loop contained in current function. This includes a for loop contained in a sub-function called by the current function.

**Example 2**
The following loop requires the parallel option in addition to the ivdep pragma to indicate there is no loop-carried dependencies:

```c
#pragma ivdep
for (i=1; i<n; i++)
{
    e[ix[2][i]] = e[ix[2][i]]+1.0;
    e[ix[3][i]] = e[ix[3][i]]+2.0;
}
```

**Example 3**
The following loop requires the parallel option in addition to the ivdep pragma to ensure there is no loop-carried dependency for the store into `a()`.

```c
#pragma ivdep
for (j=0; j<n; j++)
{
    a[b[j]] = a[b[j]] + 1;
}
```

See Also

- `vector` Indicates to the compiler that the loop should be vectorized according to the argument keywords always/aligned/unaligned/nontemporal/temporal.
- `novector` Specifies that the loop should never be vectorized.

In addition to the ivdep pragma, the vector pragma can be used to override the efficiency heuristics of the vectorizer.
loop_count

Specifies the iterations for the for loop.

Syntax

#pragma loop_count (n)
#pragma loop_count = n

or

#pragma loop_count (n1[, n2]...)
#pragma loop_count = n1[, n2]...

or

#pragma loop_count min(n), max(n), avg(n)
#pragma loop_count min=n, max=n, avg=n

Arguments

(n) or =n

Non-negative integer value. The compiler will attempt to iterate the next loop the number of times specified in \( n \); however, the number of iterations is not
guaranteed.

Non-negative integer values. The compiler will attempt to iterate the next loop the number of times specified by \( n1 \) or \( n2 \), or some other unspecified number of times. This behavior allows the compiler some flexibility in attempting to unroll the loop. The number of iterations is not guaranteed.
$min(n), max(n), avg(n)$ or $min=n, max=n, avg=n$

Non-negative integer values. Specify one or more in any order without duplication. The compiler insures the next loop iterates for the specified maximum, minimum, or average number ($n_1$) of times. The specified number of iterations is guaranteed for min and max.
The loop_count pragma affects heuristics in vectorization, loop-transformations, and software pipelining (IA-64 architecture). The pragma specifies the minimum, maximum, or average number of iterations for a for loop. In addition, a list of commonly occurring values can be specified to help the compiler generate multiple versions and perform complete unrolling. You can specify more than one pragma for a single loop; however, do not duplicate the pragma.

**Example**

**Example 1: Using #pragma loop_count (n)**
The following example illustrates how to use the pragma to iterate through the loop and enable software pipelining on IA-64 architectures.

```cpp
void loop_count(int a[], int b[])
{
    #pragma loop_count (10000)
    for (int i=0; i<1000; i++)
        a[i] = b[i] + 1.2;
}
```

**Example 2: Using #pragma loop_count min(n), max(n), avg(n)**
The following example illustrates how to use the pragma to iterate through the loop a minimum of three, a maximum of ten, and average of five times.

```cpp
#include <stdio.h>
int i;
int main()
{
    #pragma loop_count min(3), max(10), avg(5)
    for (i=1;i<=15;i++)
        printf("i=%d\n",i);
}
```

**memref_control**

Provides a method to control load latency and temporal locality at the variable level.

**Syntax**

```
#pragma memref_control
[name1[::<locality>][::<latency>]],[name2...]
```
Arguments

*name1, name2*

Specifies the name of array or pointer. You must specify at least one name; however, you can specify names with associated locality and latency values.

*locality*

An optional integer value that indicates the desired cache level to store data for future access. This will determine the load/store hint (or `prefetch` hint) to be used for this reference. The value can be one of the following:

- `1 = 0`
- `2 = 1`
- `3 = 2`
- `mem = 3`

To use this
argument, you must also specify name.

An optional integer value that indicates the load (or the latency that has to be overlapped if a prefetch is issued for this address). The value can be one of the following:

- \( \text{1\_latency} = 0 \)
- \( \text{2\_latency} = 1 \)
- \( \text{3\_latency} = 2 \)
- \( \text{mem\_latency} = 3 \)

To use this argument, you must also specify name and locality.

Description

The `memref_control` pragma is supported on Itanium® processors only. This pragma provides a method for controlling load latency and temporal locality at the variable level. The `memref_control` pragma allows you to specify locality.
and latency at the array level. For example, using this pragma allows you to control the following:

- The location (cache level) to store data for future access.
- The most appropriate latency value to be used for a load, or the latency that has to be overlapped if a `prefetch` is issued for this reference.

When you specify source-level and the data locality information at a high level for a particular data access, the compiler decides how best to use this information. If the compiler can prefetch profitably for the reference, then it issues a `prefetch` with a distance that covers the specified latency specified and then schedules the corresponding load with a smaller latency. It also uses the hints on the prefetch and load appropriately to keep the data in the specified cache level.

If the compiler cannot compute the address in advance, or decides that the overheads for prefetching are too high, it uses the specified latency to separate the load and its use (in a pipelined loop or a Global Code Scheduler loop). The hint on the load/store will correspond to the cache level passed with the locality argument.

You can use this with the `prefetch` and `noprefetch` to further tune the hints and prefetch strategies. When using the `memref_control` with `noprefetch`, keep the following guidelines in mind:

- **Specifying `noprefetch` along with the `memref_control` causes the compiler to not issue prefetches; instead the latency values specified in the `memref_control` is used to schedule the load.**
- **There is no ordering requirements for using the two pragmas together.** Specify the two pragmas in either order as long as both are specified consecutively just before the loop where it is to be applied. Issuing a `prefetch` with one hint and loading it later using a different hint can provide greater control over the hints used for specific architectures.
- **`memref_control` is handled differently from the `prefetch` or `noprefetch`.** Even if the load cannot be prefetched, the reference can still be loaded using a non-default load latency passed to the `latency` argument.
Example

Example 1: Using #pragma memref_control when prefetching is not possible

The following example illustrates a case where the address is not known in advance, so prefetching is not possible. The compiler, in this case, schedules the loads of the tab array with an L3 load latency of 15 cycles (inside a software pipelined loop or GCS loop).

```c++
#pragma memref_control tab : l2 : l3_latency
for (i=0; i<n; i++)
{
    x = <generate 64 random bits inline>;
    dum += tab[x&mask]; x>>=6;
    dum += tab[x&mask]; x>>=6;
    dum += tab[x&mask]; x>>=6;
}
```

Example 2: Using #pragma memref_control with prefetch and noprefetch pragmas [sparse matrix]

The following example illustrates one way of using memref_control, prefetch, and noprefetch together.

```c++
if( size <= 1000 ) {
    v#pragma noprefetch cp, vp
    #pragma memref_control x:l2:13_latency

    #pragma noprefetch yp, bp, rp
    #pragma noprefetch xp
    for (iii=0; iii<rag1m0; iii++) {
        if( ip < rag2 ) {
            sum -= vp[ip]*x[cp[ip]];
            ip++;
        } else {
            xp[i] = sum*yp[i];
            i++;
            sum = bp[i];
            rag2 = rp[i+1];
        }
    }
    xp[i] = sum*yp[i];
} else {
    #pragma prefetch cp, vp
    #pragma memref_control x:l2:mem_latency

    #pragma prefetch yp, bp, rp
    #pragma noprefetch xp
    for (iii=0; iii<rag1m0; iii++) {
        if( ip < rag2 ) {
            sum -= vp[ip]*x[cp[ip]];
        }
    }
```
ip++;
} else {
    xp[i] = sum*yp[i];
i++;
    sum = bp[i];
    rag2 = rp[i+1];
}
}
xp[i] = sum*yp[i];

**novector**

Specifies that the loop should never be vectorized.

**Syntax**

```c
#pragma novector
```

**Arguments**

None

**Description**

The **novector** pragma specifies that a particular loop should never be
vectorized, even if it is legal to do so. When avoiding vectorization of a loop is
desirable (when vectorization results in a performance regression rather than
improvement), the **novector** pragma can be used in the source text to disable
vectorization of a loop. This behavior is in contrast to the **vector always**
pragma.

**Example**

**Example: Using the novector pragma**

When you know the trip count \((ub - lb)\) is too low to make vectorization
worthwhile, you can use novector to tell the compiler not to vectorize, even if the
loop is considered vectorizable.

```c
void foo(int lb, int ub)
{
    #pragma novector
    for(j=lb; j<ub; j++)
    {
        a[j]=a[j]+b[j];
    }
}
```
See Also

vector pragma Indicates to the compiler that the loop should be vectorized according to the argument keywords always/aligned/unaligned/nontemporal/temporal.

optimize

Enables or disables optimizations for specific functions.

Syntax

```c
#pragma optimize("", on|off)
```

Arguments

The compiler ignores first argument values. Valid second arguments for optimize are given below.

- `off` disables optimization
- `on` enables optimization

Description

The optimize pragma is used to enable or disable optimizations for specific functions. Specifying `#pragma optimize("", off)` disables optimization until either the compiler finds a matching `#pragma optimize("", on)` statement or until the compiler reaches the end of the translation unit.

Example

Example 1: Disabling optimization for a single function using the `#pragma optimize`

In the following example, optimizations are disabled for the `alpha()` function but not for `omega()`.
Example 2: Disabling optimization for all functions using the \texttt{#pragma optimize}

In the following example, optimizations are disabled for both the \texttt{alpha()} and \texttt{omega()} functions.

```
#pragma optimize("", off)
alpha() {
    ...
}
#pragma optimize("", on)
omega() {
    ...
}
```

\texttt{optimization\_level}

Controls optimization for one function or all functions after its first occurrence.

\textbf{Syntax}

```
#pragma [intel|GCC] optimization\_level n
```

\textbf{Arguments}

\texttt{intel|GCC} indicates the interpretation to use

\texttt{n} an integer value specifying an optimization level; valid values are:

- \texttt{0 same}
optimizations as -O0
- 1 same optimizations as -O1
- 2 same optimizations as -O2
- 3 same optimizations as -O3

Note
For more information on the optimizations levels, see Enabling Automatic Optimizations in Optimizing Applications>Using Compiler Optimizations

Description
The optimization_level pragma is used to restrict optimization for a specific function while optimizing the remaining application using a different, higher optimization level. For example, if you specify -O3 (Linux* and Mac OS* X systems) for the application and specify #pragma optimization_level 1, the marked function will be optimized at the -O1 option level, while the remaining application will be optimized at the higher level.

In general, the pragma optimizes the function at the level specified as $n$; however, certain compiler optimizations, like Inter-procedural Optimization (IPO), are not enabled or disabled during translation unit compilation. For example, if you enable IPO and a specific optimization level, IPO is enabled even for the function targeted by this pragma; however, IPO might not be fully implemented
regardless of the optimization level specified at the command line. The reverse is also true.

**Scope of optimization restriction**

On Linux* and Mac OS* X systems, the scope of the optimization restriction can be affected by arguments passed to the `-pragma-optimization-level` compiler option as explained in the following table.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>#pragma intel optimization_level n</td>
<td>Applies pragma only to the next function, using the specified optimization level, regardless of the argument passed to the <code>-pragma-optimization-level</code> option.</td>
</tr>
<tr>
<td>#pragma GCC optimization_level n</td>
<td>Applies pragma to all subsequent functions, using the specified optimization level, regardless of the argument passed to the <code>-pragma-optimization-level</code> option.</td>
</tr>
<tr>
<td>#pragma GCC optimization_level reset</td>
<td>Specifying reset reverses the effect of the most recent #pragma GCC optimization_level statement, by returning to the optimization level previously specified.</td>
</tr>
<tr>
<td>#pragma optimization_level n</td>
<td>Applies either the intel or GCC interpretation. Interpretation depends on argument passed to the <code>-pragma-optimization-level</code> option.</td>
</tr>
</tbody>
</table>

**Note**

On Windows* systems, the pragma always uses the `intel` interpretation; the pragma is applied only to the next function.

**Using the intel interpretation of #pragma optimization_level**

Place the pragma immediately before the function being affected.

**Example: intel interpretation of pragma**

```cpp
#pragma intel optimization_level 1
gamma() {  
  ...
}
```
Using the GCC* interpretation of `#pragma optimization_level`
Place the pragma in any location prior to the functions being affected.

**parallel/noparallel**

The `parallel` pragma helps the compiler to resolve dependencies thereby facilitating auto-parallelization of an immediately following FOR loop. The `noparallel` pragma prevents auto-parallelization of an immediately following FOR loop.

**Syntax**

```c
#pragma parallel
#pragma parallel {always}
#pragma noparallel
```

**Arguments**

`always`

Overrides compiler heuristics that estimate the likelihood that parallelization of a loop would increase performance. Using this keyword allows a loop to be
parallelized
even when
the compiler
estimates
that
parallelization
might not
improve
performance.

Description

The `parallel` pragma instructs the compiler to ignore potential dependencies that it assumes could exist and which would prevent correct parallelization in the immediately following loop. However, if dependencies are proven, they are not ignored.

Using `#pragma parallel always` overrides the compiler heuristics that estimate the likelihood that parallelization of a loop would increase performance. It allows a loop to be parallelized even when the compiler estimates that parallelization might not improve performance.

The `#pragma noparallel` prevents autoparallelization of immediately following DO loops.

These pragmas take effect only if autoparallelization is enabled by the switch `/Qparallel (Windows* operating system) or -parallel (Linux* or Mac OS* X operating systems).

⚠️ Caution

The `#pragma parallel always` should be used with care. Overriding the heuristics of the compiler should only be done if you are absolutely sure the parallelization will improve performance.

Example

The following example illustrates how to use the `#pragma parallel`.  

1518
void add(int k, float *a, float *b)
{
    #pragma parallel
    for (int i = 0; i < 10000; i++)
        a[i] = a[i+k] + b[i];
}

See Also

Programming for Multithread Platform Consistency in Optimizing Applications>
Using Parallelism: Automatic Parallelization

**prefetch/noprefetch**

Invites the compiler to issue data prefetches from memory (prefetch) or disables data prefetching (noprefetch).

**Syntax**

```cpp
#pragma prefetch
#pragma prefetch [var1 [: hint1 [: distance1]] [, var2 [: hint2 [: distance2]]]...]
#pragma noprefetch [var1 [, var2]...]
```

**Arguments**

- **var**
  - Optional memory reference (data to be prefetched)

- **hint**
  - Optional hint to the compiler to specify the type of prefetch. Possible values are the constants defined in the header `xmmintrin.h`:
- \_MM\_HINT\_T0
  - for integer data that will be reused
- \_MM\_HINT\_NT1
  - for integer and floating point data that will be reused from L2 cache
- \_MM\_HINT\_NT2
  - for data that will be reused from L3 cache
- \_MM\_HINT\_NTA
  - for data that will not be reused

To use this argument, you must also specify \textit{var}.

\textit{distance}

Optional integer argument with a value greater than 0. It indicates the number of loop iterations ahead of which a prefetch is issued, before the
corresponding load
or store instruction.
To use this
argument, you
must also specify
\textit{var} and \textit{hint}.

\textbf{Description}

The \texttt{prefetch} pragma is supported by Intel® Itanium® processors only. The \texttt{prefetch} pragma hints to the compiler to generate data prefetches for some memory references. This affects the heuristics used in the compiler. Prefetching data can minimize the effects of memory latency.

If you specify \#pragma prefetch with no arguments, all arrays accessed in the immediately following loop are prefetched.

If the loop includes the expression \texttt{A(j)}, placing \#pragma prefetch \texttt{A} in front of the loop asks the compiler to insert prefetches for \texttt{A(j + d)} within the loop. Here, \textit{d} is the number of iterations ahead of which to prefetch the data, and is determined by the compiler.

The \texttt{prefetch} pragma affects the immediately following loop provided that the compiler general optimization level is \texttt{-O1} (Linux* operating systems) or \texttt{/O1} (Windows* operating systems) or higher. Remember that \texttt{-O2} or \texttt{/O2} is the default optimization level.

The \texttt{noprefetch} pragma is also supported by Intel® Itanium® processors only. This pragma hints to the compiler not to generate data prefetches for some memory references. This affects the heuristics used in the compiler.

\textbf{Example}

\textbf{Example 1: using noprefetch and prefetch pragmas}

The following example demonstrates how to use the \texttt{noprefetch} and \texttt{prefetch} pragmas together:

\begin{verbatim}
#pragma noprefetch b

\end{verbatim}
Example 2: using noprefetch and prefetch pragmas

The following is yet another example of how to use the **noprefetch** and **prefetch** pragmas:

```cpp
for (i=0; i!=i1; i+=is) {
    float sum = b[i];
    int ip = srow[i];
    int c = col[ip];

    #pragma noprefetch col
    #pragma prefetch value:1:80
    #pragma prefetch x:1:40

    for(; ip<srow[i+1]; c=col[++ip])
        sum -= value[ip] * x[c];
    y[i] = sum;
}
```

Example 3: using noprefetch, prefetch, memref_control pragmas

The following example, which is for IA-64 architecture only, demonstrates how to use the **prefetch**, **noprefetch**, and **memref_control** pragmas together:

```cpp
#define SIZE 10000
int prefetch(int *a, int *b)
{
    int i, sum = 0;
    #pragma memref_control a:l2
    #pragma noprefetch a
    #pragma prefetch b
    for (i = 0; i<SIZE; i++)
        sum += a[i] * b[i];
    return sum;
}
```

```cpp
#include <stdio.h>
int main()
{
    int i, arr1[SIZE], arr2[SIZE];
    for (i = 0; i<SIZE; i++)
        arr1[i] = i;
    arr2[i] = i;
    printf("Demonstrating the use of prefetch, noprefetch, \n""
           "and memref_control pragma together.\n")
    prefetch(arr1, arr2);
    return 0;
}
```

**See Also**
**swp/noswp**

Indicates a preference for loops to be software pipelined or not pipelined.

**Syntax**

```cpp
#pragma swp
#pragma noswp
```

**Arguments**

None

**Description**

The `swp` pragma indicates a preference for loops to be software pipelined. The pragma does not help data dependency, but overrides heuristics based on profile counts or unequal control flow. The software pipelining optimization triggered by the `swp` pragma applies instruction scheduling to certain innermost loops, allowing instructions within a loop to be split into different stages, allowing increased instruction-level parallelism.

This strategy can reduce the impact of long-latency operations, resulting in faster loop execution. Loops chosen for software pipelining are always the innermost loops that do not contain procedure calls that are not inlined. Because the optimizer no longer considers fully unrolled loops as innermost loops, fully unrolling loops can allow an additional loop to become the innermost loop. The `noswp` pragma is used to instruct the compiler not to software pipeline that loop. This may be advantageous for loops that iterate few times, as pipelining introduces overhead in executing the prolog and epilog code sequences.

**Example**

**Example: using swp pragma**
The following example demonstrates one way of using the pragma to instruct the compiler to attempt software pipeling.

```cpp
t void swp(int a[], int b[]) {
    #pragma swp
    for (int i = 0; i < 100; i++)
        if (a[i] == 0)
            b[i] = a[i] + 1;
        else
            b[i] = a[i] * 2;
}
```

See Also

Loop unrolling options in Optimizing Applications>Using High-Level Optimizations (HLO) >Loop Unrolling
Optimizer Report Generation in Optimizing Applications>Evaluating Performance>Using Compiler Reports>Generating Reports

unroll/nounroll

Indicates to the compiler to unroll or not to unroll a counted loop.

Syntax

```cpp
#pragma unroll
#pragma unroll(n)
#pragma nounroll
```

Arguments

n is the unrolling factor representing the number of times to unroll a
Description

The \texttt{unroll[n]} pragma tells the compiler how many times to unroll a counted loop.

The \texttt{unroll} pragma must precede the \texttt{FOR} statement for each \texttt{FOR} loop it affects. If \texttt{n} is specified, the optimizer unrolls the loop \texttt{n} times. If \texttt{n} is omitted or if it is outside the allowed range, the optimizer assigns the number of times to unroll the loop.

This pragma is supported only when option \texttt{O3} is set. The \texttt{unroll} pragma overrides any setting of loop unrolling from the command line.

The pragma can be applied for the innermost loop nest as well as for the outer loop nest. If applied to outer loop nests, the current implementation supports complete outer loop unrolling. The loops inside the loop nest are either not unrolled at all or completely unrolled. The compiler generates correct code by comparing \texttt{n} and the loop count.

When unrolling a loop increases register pressure and code size it may be necessary to prevent unrolling of a loop. In such cases, use the \texttt{nounroll} pragma. The \texttt{nounroll} pragma instructs the compiler not to unroll a specified loop.

Example

Example 1: using \texttt{unroll} pragma for innermost loop unrolling

```cpp
void unroll(int a[], int b[], int c[], int d[]) {
    #pragma unroll(4)
    for (int i = 1; i < 100; i++) {
        b[i] = a[i] + 1;
        d[i] = c[i] + 1;
    }
}
```
Example 2: using unroll pragma for outer loop unrolling

In Example 2, placing the `#pragma unroll` before the first `for` loop causes the compiler to unroll the outer loop completely. If a `#pragma unroll` is placed before the inner `for` loop as well as before the outer `for` loop, the compiler ignores the inner `for` loop `unroll pragma`. If the `#pragma unroll` is placed only for the innermost loop, the compiler unrolls the innermost loop according to some factor.

```c
int m = 0;
int dir[4]= {1,2,3,4};
int data[10];
#pragma unroll (4)  // outer loop unrolling
for (int i = 0; i < 4; i++)
{
    for (int j = dir[i]; data[j]==N ; j+=dir[i])
        m++;
}
```

See Also

Loop unrolling options in Optimizing Applications>Using High-Level Optimizations (HLO) >Loop Unrolling

Optimizer Report Generation in Optimizing Applications>Evaluating Performance>Using Compiler Reports>Generating Reports

Applying Optimization Strategies in Optimizing Applications>Programming Guidelines>Applying Optimization Strategies

unroll_and_jam/nounroll_and_jam

Hints to the compiler to enable or disable loop unrolling and jamming. These pragmas can only be applied to iterative `FOR` loops.

Syntax

```c
#pragma unroll_and_jam
#pragma unroll_and_jam (n)
#pragma nounroll_and_jam
```

Arguments
$n$ is the unrolling factor representing the number of times to unroll a loop; it is an integer constant from 0 through 255.

**Description**

The `unroll_and_jam` pragma partially unrolls one or more loops higher in the nest than the innermost loop and fuses/jams the resulting loops back together. This transformation allows more reuses in the loop. This pragma is not effective on innermost loops. Ensure that the immediately following loop is not the innermost loop after compiler-initiated interchanges are completed.

Specifying this pragma is a hint to the compiler that the unroll and jam sequence is legal and profitable. The compiler enables this transformation whenever possible.

The `unroll_and_jam` pragma must precede the `FOR` statement for each `FOR` loop it affects. If $n$ is specified, the optimizer unrolls the loop $n$ times. If $n$ is omitted or if it is outside the allowed range, the optimizer assigns the number of times to unroll the loop. The compiler generates correct code by comparing $n$ and the loop count.

This pragma is supported only when compiler option `O3` is set. The `unroll_and_jam` pragma overrides any setting of loop unrolling from the command line.
When unrolling a loop increases register pressure and code size it may be necessary to prevent unrolling of a nested/imperfect nested loop. In such cases, use the `nounroll_and_jam` pragma. The `nounroll_and_jam` pragma hints to the compiler not to unroll a specified loop.

**Example**

**Example: using unroll_and_jam pragma**

```cpp
int a[10][10];
ext b[10][10];
ext c[10][10];
ext d[10][10];
void unroll(int n)
{
    int i,j,k;
    #pragma unroll_and_jam (6)
    for (i = 1; i < n; i++) {
        #pragma unroll_and_jam (6)
        for (j = 1; j < n; j++) {
            for (k = 1; k < n; k++){
                a[i][j] += b[i][k]*c[k][j];
            }
        }
    }
}
```

**See Also**

Loop unrolling options in *Optimizing Applications>*Using High-Level Optimizations (HLO) >Loop Unrolling

Optimizer Report Generation in *Optimizing Applications>*Evaluating Performance>Using Compiler Reports>Generating Reports

Applying Optimization Strategies in *Optimizing Applications>*Programming Guidelines>Applying Optimization Strategies

**unused**

Describes variables that are unused (warnings not generated).

**Syntax**

```cpp
#pragma unused
```

**Arguments**
The unused pragma is implemented for compatibility with Apple* implementation of GCC.

**vector {always|aligned|unaligned|nontemporal|temporal}**

Indicates to the compiler that the loop should be vectorized according to the argument keywords always/aligned/unaligned/nontemporal/temporal.

**Syntax**

```
#pragma vector
{always|aligned|unaligned|nontemporal|temporal}
#pragma vector nontemporal[(var1[, var2, ...])]  
```

**Arguments**

- **always**
  
  instructs the compiler to override any efficiency heuristic during the decision to vectorize or not, and vectorize non-unit strides or very unaligned memory
accesses; controls the vectorization of the subsequent loop in the program

aligned

instructs compiler to use aligned data movement instructions for all array references when vectorizing

unaligned

instructs compiler to use unaligned data movement instructions for all array references when vectorizing

nontemporal
directs the compiler to
use non-temporal (that is, streaming) stores on systems based on IA-32 and Intel(R) 64 architectures; optionally takes a comma separated list of variables

temporal
directs the compiler to use temporal (that is, non-streaming) stores on systems based on IA-32 and Intel(R) 64 architectures

Description

The **vector** pragma indicates that the loop should be vectorized, if it is legal to do so, ignoring normal heuristic decisions about profitability. The **vector**
pragma takes several argument keywords to specify the kind of loop vectorization required. These keywords are aligned, unaligned, always, and nontemporal.

**Using aligned/unaligned keywords**

When the aligned/unaligned argument keyword is used with this pragma, it indicates that the loop should be vectorized using aligned/unaligned data movement instructions for all array references. Specify only one argument keyword: aligned or unaligned.

⚠️ Caution

If you specify aligned as an argument, you must be sure that the loop is vectorizable using this pragma. Otherwise, the compiler generates incorrect code.

**Using always keyword**

When the always argument keyword is used, the pragma controls the vectorization of the subsequent loop in the program. As the compiler does not apply the vector pragma to nested loops, each nested loop needs a preceding pragma statement. Place the pragma before the loop control statement.

⚠️ Note

The pragma vector{always|aligned|unaligned} should be used with care. Overriding the efficiency heuristics of the compiler should only be done if the programmer is absolutely sure that vectorization will improve performance. Furthermore, instructing the compiler to implement all array references with aligned data movement instructions will cause a run-time exception in case some of the access patterns are actually unaligned.

**Using nontemporal/temporal keywords**

The nontemporal and temporal argument keywords are used to control how the "stores" of register contents to storage are performed (streaming versus non-streaming) on systems based on IA-32 and Intel® 64 architectures.
By default, the compiler automatically determines whether a streaming store should be used for each variable. Streaming stores may cause significant performance improvements over non-streaming stores for large numbers on certain processors. However, the misuse of streaming stores can significantly degrade performance.

Example

**Example 1: vector aligned pragma**
The loop in the following example uses the `aligned` argument keyword to request that the loop be vectorized with aligned instructions, as the arrays are declared in such a way that the compiler could not normally prove this would be safe to do so.

```cpp
void vec_aligned(float *a, int m, int c) {
    int i;
    // Instruct compiler to ignore assumed vector dependencies.
    #pragma vector aligned
    for (i = 0; i < m; i++)
        a[i] = a[i] * c;
    // Alignment unknown but compiler can still align.
    for (i = 0; i < 100; i++)
        a[i] = a[i] + 1.0f;
}
```

**Example 2: vector always pragma**
The following example illustrates how to use the `vector always` pragma.

```cpp
void vec_always(int *a, int *b, int m) {
    #pragma vector always
    for(int i = 0; i <= m; i++)
        a[32*i] = b[99*i];
}
```

**Example 3a: vector nontemporal pragma**
A float-type loop together with the generated assembly are shown in the following example. For large $N$, significant performance improvements result on Pentium 4 systems over a non-streaming implementation.

```cpp
float a[1000];
void foo(int N){
    int i;
    #pragma vector nontemporal
    for (i = 0; i < N; i++) {
        a[i] = 1;
    }
}
Example 3b: ASM code for the loop body

```
.B1.2:
movntps XMMWORD PTR _a[eax], xmm0
movntps XMMWORD PTR _a[eax+16], xmm0
add eax, 32
cmp eax, 4096
jl .B1.2
```

Example 4: vector nontemporal pragma with variables

The following example illustrates how to use the `#pragma vector nontemporal` with variables for implementing streaming stores.

```c
double A[1000];
double B[1000];
void foo(int n){
    int i;
    #pragma vector nontemporal (A, B)
    for (i=0; i<n; i++){
        A[i] = 0;
        B[i] = i;
    }
}
```

Intel Supported Pragmas

The Intel® C++ Compiler supports the following pragmas:

<table>
<thead>
<tr>
<th>Pragma</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>alloc_text</td>
<td>names the code section where the specified function definitions are to reside</td>
</tr>
<tr>
<td>auto_inline</td>
<td>excludes any function defined within the range where off is specified from being considered as candidates for automatic inline expansion</td>
</tr>
<tr>
<td>bss_seg</td>
<td>indicates to the compiler the segment where uninitialized variables are stored in the .obj file</td>
</tr>
<tr>
<td>check_stack</td>
<td>on argument indicates that stack checking should be enabled for functions that follow and off argument indicates that stack checking should be disabled for functions that follow.</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>code_seg</td>
<td>specifies a code section where functions are to be allocated</td>
</tr>
<tr>
<td>comment</td>
<td>places a comment record into an object file or executable file</td>
</tr>
<tr>
<td>component</td>
<td>controls collecting of browse information or dependency information from within source files</td>
</tr>
<tr>
<td>conform</td>
<td>specifies the run-time behavior of the /Zc:forScope compiler option</td>
</tr>
<tr>
<td>const_seg</td>
<td>specifies the segment where functions are stored in the .obj file</td>
</tr>
<tr>
<td>data_seg</td>
<td>specifies the default section for initialized data</td>
</tr>
<tr>
<td>deprecated</td>
<td>indicates that a function, type, or any other identifier may not be supported in a future release or indicates that a function, type, or any other identifier should not be used any more</td>
</tr>
<tr>
<td>poison</td>
<td>labels the identifiers you want removed from your program; an error results when compiling a &quot;poisoned&quot; identifier; #pragma POISON is also supported.</td>
</tr>
<tr>
<td>float_control</td>
<td>specifies floating-point behavior for a function</td>
</tr>
<tr>
<td>fp_contract</td>
<td>allows or disallows the implementation to contract expressions</td>
</tr>
<tr>
<td>include_directory</td>
<td>appends the string argument to the list of places to search for #include files; HP compatible pragma</td>
</tr>
<tr>
<td>init_seg</td>
<td>specifies the section to contain C++ initialization code for the translation unit</td>
</tr>
<tr>
<td>Pragma</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>message</td>
<td>displays the specified string literal to the standard output device</td>
</tr>
<tr>
<td>optimize</td>
<td>specifies optimizations to be performed on a function-by-function basis; implemented to partly support Microsoft's implementation of same pragma; <a href="#">click here</a> for Intel's implementation</td>
</tr>
<tr>
<td>options</td>
<td>GCC-compatible (MacOS) pragma; sets the alignment of fields in structures</td>
</tr>
<tr>
<td>pointers_to_members</td>
<td>specifies whether a pointer to a class member can be declared before its associated class definition and is used to control the pointer size and the code required to interpret the pointer</td>
</tr>
<tr>
<td>pop_macro</td>
<td>sets the value of the macro_name macro to the value on the top of the stack for this macro</td>
</tr>
<tr>
<td>push_macro</td>
<td>saves the value of the macro_name macro on the top of the stack for this macro</td>
</tr>
<tr>
<td>region/endregion</td>
<td>specifies a code segment in the Microsoft Visual Studio* 2005 Code Editor that expands and contracts by using the outlining feature</td>
</tr>
<tr>
<td>section</td>
<td>creates a section in an .obj file. Once a section is defined, it remains valid for the remainder of the compilation</td>
</tr>
<tr>
<td>start_map_region</td>
<td>used in conjunction with the stop_map_region pragma</td>
</tr>
<tr>
<td>stop_map_region</td>
<td>used in conjunction with the start_map_region pragma</td>
</tr>
</tbody>
</table>
Pragma | Description
---|---
fenv_access | informs an implementation that a program may test status flags or run under a non-default control mode
vtordisp | on argument enables the generation of hidden vtordisp members and off disables them
warning | allows selective modification of the behavior of compiler warning messages
weak | declares symbol you enter to be weak

See Also

Intel-specific Pragmas

Intel(R) Math Library

Overview: Intel® Math Library for Linux

The Intel® C++ Compiler includes a mathematical software library containing highly optimized and very accurate mathematical functions. These functions are commonly used in scientific or graphic applications, as well as other programs that rely heavily on floating-point computations. To include support for C99 _Complex data types, use the -std=c99 compiler option.

The mathimf.h header file includes prototypes for the library functions. For a complete list of the functions available, refer to the Function List.

Math Libraries

The math library linked to an application depends on the compilation or linkage options specified.

<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>libimf.a</td>
<td>Default static math library.</td>
</tr>
<tr>
<td>libimf.so</td>
<td>Default shared math library.</td>
</tr>
</tbody>
</table>
Using the Intel Math Library

To use the Intel math library, include the header file, \texttt{mathimf.h}, in your program. Here are two example programs that illustrate the use of the math library.

Example Using Real Functions

```c
// real_math.c

#include <stdio.h>
#include <mathimf.h>

int main() {
    float fp32bits;
    double fp64bits;
    long double fp80bits;
    long double pi_by_four = 3.141592653589793238/4.0;
    // pi/4 radians is about 45 degrees
    fp32bits = (float) pi_by_four; // float approximation to pi/4
    fp64bits = (double) pi_by_four; // double approximation to pi/4
    fp80bits = pi_by_four; // long double (extended) approximation to pi/4

    // The sin(pi/4) is known to be 1/sqrt(2) or approximately .7071067
    printf("When x = %8.8f, sinf(x) = %8.8f \n", fp32bits, 
sinf(fp32bits));
    printf("When x = %16.16f, sin(x) = %16.16f \n", fp64bits, 
sin(fp64bits));
    printf("When x = %20.20Lf, sinl(x) = %20.20f \n", fp80bits, 
sinl(fp80bits));

    return 0;
}
```

The command for compiling \texttt{real_math.c} is:

```
icc real_math.c
```

The output of \texttt{a.out} will look like this:

When $x = 0.78539816$, $\sin(x) = 0.70710678$

When $x = 0.7853981633974483$, $\sin(x) = 0.7071067811865475$
When \( x = 0.78539816339744827900 \), \( \sin(x) = 0.70710678118654750275 \)

**Example Using Complex Functions**

```c
// complex_math.c

#include <stdio.h>
#include <complex.h>

int main()
{

    float _Complex c32in,c32out;
    double _Complex c64in,c64out;
    double pi_by_four = 3.141592653589793238/4.0;

    c64in = 1.0 + I* pi_by_four;
    // Create the double precision complex number 1 + (pi/4) * i
    // where I is the imaginary unit.

    c32in = (float _Complex) c64in;
    // Create the float complex value from the double complex value.

    c64out = cexp(c64in);
    c32out = cexpf(c32in);
    // Call the complex exponential,
    // \( \text{cexp}(z) = e^{x + iy} = e^x \cdot (\cos(y) + i \sin(y)) \)

    printf("When z = %7.7f + %7.7f i, cexpf(z) = %7.7f + %7.7f i \n",
            crealf(c32in), cimagf(c32in), crealf(c32out), cimagf(c32out));
    printf("When z = %12.12f + %12.12f i, cexp(z) = %12.12f + %12.12f i \n",
            creal(c64in), cimag(c64in), creal(c64out), cimagf(c64out));

    return 0;
}
```

The command to compile `complex_math.c` is:

```bash
icc -std=c99 complex_math.c
```

The output of `a.out` will look like this:

When \( z = 1.0000000 + 0.7853982 i \), \( \text{cexpf}(z) = 1.9221154 + 1.9221156 i \)

When \( z = 1.000000000000 + 0.785398163397 i \), \( \text{cexp}(z) = 1.922115514080 + 1.922115514080 i \)
Note

_CComplex data types are supported in C but not in C++ programs. It is necessary to include the -std=c99 compiler option when compiling programs that require support for _Complex data types.

Exception Conditions

If you call a math function using argument(s) that may produce undefined results, an error number is assigned to the system variable errno. Math function errors are usually domain errors or range errors.

**Domain errors** result from arguments that are outside the domain of the function. For example, **acos** is defined only for arguments between -1 and +1 inclusive. Attempting to evaluate **acos(-2)** or **acos(3)** results in a domain error, where the return value is QNaN.

**Range errors** occur when a mathematically valid argument results in a function value that exceeds the range of representable values for the floating-point data type. Attempting to evaluate **exp(1000)** results in a range error, where the return value is INF.

When domain or range error occurs, the following values are assigned to errno:

- domain error (EDOM): errno = 33
- range error (ERANGE): errno = 34

The following example shows how to read the errno value for an EDOM and ERANGE error.

```c
#include <errno.h>
#include <mathimf.h>
#include <stdio.h>

int main(void)
{
    double neg_one=-1.0;
    double zero=0.0;

    // The natural log of a negative number is considered a domain error - EDOM
    printf("log(%e) = %e and errno(EDOM) = %d \n",neg_one,log(neg_one),errno);
```
// The natural log of zero is considered a range error - ERANGE
printf("log(%e) = %e and errno(ERANGE) = %d
\n", zero, log(zero), errno);
}
The output of errno.c will look like this:
log(-1.000000e+00) = nan and errno(EDOM) = 33
log(0.000000e+00) = -inf and errno(ERANGE) = 34

For the math functions in this section, a corresponding value for errno is listed when applicable.

Other Considerations

Some math functions are inlined automatically by the compiler. The functions actually inlined may vary and may depend on any vectorization or processor-specific compilation options used. For more information, see Inline Function Expansion in Optimizing Applications>Using Interprocedural Optimizations>Inline Expansion of Functions.

A change of the default precision control or rounding mode may affect the results returned by some of the mathematical functions. See Overview: Tuning Performance in Floating-point Operations>Tuning Performance.

Math Functions

Function List

The Intel Math Library functions are listed here by function type.

<table>
<thead>
<tr>
<th>Function Type</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Trigonometric Functions</strong></td>
<td>acos</td>
</tr>
<tr>
<td></td>
<td>acosd</td>
</tr>
<tr>
<td></td>
<td>asin</td>
</tr>
<tr>
<td></td>
<td>asind</td>
</tr>
<tr>
<td></td>
<td>atan</td>
</tr>
<tr>
<td></td>
<td>atan2</td>
</tr>
<tr>
<td></td>
<td>atand</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td><code>atand2</code></td>
</tr>
<tr>
<td></td>
<td><code>cos</code></td>
</tr>
<tr>
<td></td>
<td><code>cosd</code></td>
</tr>
<tr>
<td></td>
<td><code>cot</code></td>
</tr>
<tr>
<td></td>
<td><code>cotd</code></td>
</tr>
<tr>
<td></td>
<td><code>sin</code></td>
</tr>
<tr>
<td></td>
<td><code>sincos</code></td>
</tr>
<tr>
<td></td>
<td><code>sincosd</code></td>
</tr>
<tr>
<td></td>
<td><code>sind</code></td>
</tr>
<tr>
<td></td>
<td><code>tan</code></td>
</tr>
<tr>
<td></td>
<td><code>tand</code></td>
</tr>
<tr>
<td><strong>Hyperbolic Functions</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>acosh</code></td>
</tr>
<tr>
<td></td>
<td><code>asinh</code></td>
</tr>
<tr>
<td></td>
<td><code>atanh</code></td>
</tr>
<tr>
<td></td>
<td><code>cosh</code></td>
</tr>
<tr>
<td></td>
<td><code>sinh</code></td>
</tr>
<tr>
<td></td>
<td><code>sinhcosh</code></td>
</tr>
<tr>
<td></td>
<td><code>tanh</code></td>
</tr>
<tr>
<td><strong>Exponential Functions</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>cbrt</code></td>
</tr>
<tr>
<td></td>
<td><code>exp</code></td>
</tr>
<tr>
<td></td>
<td><code>exp10</code></td>
</tr>
<tr>
<td></td>
<td><code>exp2</code></td>
</tr>
<tr>
<td></td>
<td><code>expm1</code></td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>--------------</td>
<td>--------</td>
</tr>
<tr>
<td></td>
<td>frexp</td>
</tr>
<tr>
<td></td>
<td>hypot</td>
</tr>
<tr>
<td></td>
<td>invsqr</td>
</tr>
<tr>
<td></td>
<td>ilogb</td>
</tr>
<tr>
<td></td>
<td>ldexp</td>
</tr>
<tr>
<td></td>
<td>log</td>
</tr>
<tr>
<td></td>
<td>log10</td>
</tr>
<tr>
<td></td>
<td>log1p</td>
</tr>
<tr>
<td></td>
<td>log2</td>
</tr>
<tr>
<td></td>
<td>logb</td>
</tr>
<tr>
<td></td>
<td>pow</td>
</tr>
<tr>
<td></td>
<td>scalb</td>
</tr>
<tr>
<td></td>
<td>scalbln</td>
</tr>
<tr>
<td></td>
<td>scalbn</td>
</tr>
<tr>
<td></td>
<td>sqrt</td>
</tr>
<tr>
<td>Special</td>
<td>annuity</td>
</tr>
<tr>
<td>Functions</td>
<td>compound</td>
</tr>
<tr>
<td></td>
<td>erf</td>
</tr>
<tr>
<td></td>
<td>erfc</td>
</tr>
<tr>
<td></td>
<td>erfinv</td>
</tr>
<tr>
<td></td>
<td>gamma</td>
</tr>
<tr>
<td></td>
<td>gamma_r</td>
</tr>
<tr>
<td></td>
<td>j0</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>---------------------</td>
<td>--------</td>
</tr>
<tr>
<td>j1</td>
<td>j1</td>
</tr>
<tr>
<td>jn</td>
<td>jn</td>
</tr>
<tr>
<td>lgamma</td>
<td>lgamma</td>
</tr>
<tr>
<td>lgamma_r</td>
<td>lgamma_r</td>
</tr>
<tr>
<td>tgamma</td>
<td>tgamma</td>
</tr>
<tr>
<td>y0</td>
<td>y0</td>
</tr>
<tr>
<td>y1</td>
<td>y1</td>
</tr>
<tr>
<td>yn</td>
<td>yn</td>
</tr>
<tr>
<td>Nearest Integer Functions</td>
<td>ceil</td>
</tr>
<tr>
<td></td>
<td>floor</td>
</tr>
<tr>
<td></td>
<td>llrint</td>
</tr>
<tr>
<td></td>
<td>llround</td>
</tr>
<tr>
<td></td>
<td>lrint</td>
</tr>
<tr>
<td></td>
<td>lround</td>
</tr>
<tr>
<td></td>
<td>modf</td>
</tr>
<tr>
<td></td>
<td>nearbyint</td>
</tr>
<tr>
<td></td>
<td>rint</td>
</tr>
<tr>
<td></td>
<td>round</td>
</tr>
<tr>
<td></td>
<td>trunc</td>
</tr>
<tr>
<td>Remainder Functions</td>
<td>fmod</td>
</tr>
<tr>
<td></td>
<td>remainder</td>
</tr>
<tr>
<td></td>
<td>remquo</td>
</tr>
<tr>
<td></td>
<td>copysign</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>Miscellaneous Functions</td>
<td>copysign</td>
</tr>
<tr>
<td></td>
<td>fabs</td>
</tr>
<tr>
<td></td>
<td>fdim</td>
</tr>
<tr>
<td></td>
<td>finite</td>
</tr>
<tr>
<td></td>
<td>fma</td>
</tr>
<tr>
<td></td>
<td>fmax</td>
</tr>
<tr>
<td></td>
<td>fmin</td>
</tr>
<tr>
<td></td>
<td>fpclassify</td>
</tr>
<tr>
<td></td>
<td>isfinite</td>
</tr>
<tr>
<td></td>
<td>isgreater</td>
</tr>
<tr>
<td></td>
<td>isgreaterequal</td>
</tr>
<tr>
<td></td>
<td>isinf</td>
</tr>
<tr>
<td></td>
<td>isless</td>
</tr>
<tr>
<td></td>
<td>islessequal</td>
</tr>
<tr>
<td></td>
<td>islessgreater</td>
</tr>
<tr>
<td></td>
<td>isnan</td>
</tr>
<tr>
<td></td>
<td>isnormal</td>
</tr>
<tr>
<td></td>
<td>isunordered</td>
</tr>
<tr>
<td></td>
<td>nextafter</td>
</tr>
<tr>
<td></td>
<td>nexttoward</td>
</tr>
<tr>
<td></td>
<td>signbit</td>
</tr>
<tr>
<td></td>
<td>significand</td>
</tr>
<tr>
<td></td>
<td>cabs</td>
</tr>
<tr>
<td>Function Type</td>
<td>Name</td>
</tr>
<tr>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>Complex Functions</td>
<td>cabs</td>
</tr>
<tr>
<td></td>
<td>cacos</td>
</tr>
<tr>
<td></td>
<td>cacosh</td>
</tr>
<tr>
<td></td>
<td>carg</td>
</tr>
<tr>
<td></td>
<td>casin</td>
</tr>
<tr>
<td></td>
<td>casinh</td>
</tr>
<tr>
<td></td>
<td>catan</td>
</tr>
<tr>
<td></td>
<td>catanh</td>
</tr>
<tr>
<td></td>
<td>ccos</td>
</tr>
<tr>
<td></td>
<td>cexp</td>
</tr>
<tr>
<td></td>
<td>cexp2</td>
</tr>
<tr>
<td></td>
<td>cimag</td>
</tr>
<tr>
<td></td>
<td>cis</td>
</tr>
<tr>
<td></td>
<td>clog</td>
</tr>
<tr>
<td></td>
<td>clog10</td>
</tr>
<tr>
<td></td>
<td>conj</td>
</tr>
<tr>
<td></td>
<td>ccosh</td>
</tr>
<tr>
<td></td>
<td>cpow</td>
</tr>
<tr>
<td></td>
<td>cproj</td>
</tr>
<tr>
<td></td>
<td>creal</td>
</tr>
<tr>
<td></td>
<td>csin</td>
</tr>
<tr>
<td></td>
<td>csinh</td>
</tr>
<tr>
<td></td>
<td>csqrt</td>
</tr>
</tbody>
</table>
Trigonometric Functions

The Intel Math Library supports the following trigonometric functions:

**acos**

**Description:** The `acos` function returns the principal value of the inverse cosine of \( x \) in the range \([0, \pi]\) radians for \( x \) in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**

double acos(double x);
long double acosl(long double x);
float acosf(float x);

**acosd**

**Description:** The `acosd` function returns the principal value of the inverse cosine of \( x \) in the range \([0,180]\) degrees for \( x \) in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**

double acosd(double x);
long double acosdl(long double x);
float acosdf(float x);

**asin**

**Description:** The `asin` function returns the principal value of the inverse sine of \( x \) in the range \([-\pi/2, +\pi/2]\) radians for \( x \) in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)
**Calling interface:**

double asin(double x);
long double asinl(long double x);
float asinf(float x);

**asind**

**Description:** The `asind` function returns the principal value of the inverse sine of `x` in the range \([-90,90]\) degrees for `x` in the interval \([-1,1]\).

**errno:** EDOM, for \(|x| > 1\)

**Calling interface:**

double asind(double x);
long double asindl(long double x);
float asindf(float x);

**atan**

**Description:** The `atan` function returns the principal value of the inverse tangent of `x` in the range \([-\pi/2, +\pi/2]\) radians.

**Calling interface:**

double atan(double x);
long double atanl(long double x);
float atanf(float x);

**atan2**

**Description:** The `atan2` function returns the principal value of the inverse tangent of \(y/x\) in the range \([-\pi, +\pi]\) radians.

**errno:** EDOM, for \(x = 0\) and \(y = 0\)

**Calling interface:**

double atan2(double y, double x);
long double atan2l(long double y, long double x);
float atan2f(float y, float x);

**atand**
**Description:** The `atand` function returns the principal value of the inverse tangent of \( x \) in the range [-90,90] degrees.

**Calling interface:**

```c
double atand(double x);
long double atandl(long double x);
float atandf(float x);
```

**atan2d**

**Description:** The `atan2d` function returns the principal value of the inverse tangent of \( y/x \) in the range [-180, +180] degrees.

**errno:** EDOM, for \( x = 0 \) and \( y = 0 \).

**Calling interface:**

```c
double atan2d(double x, double y);
long double atan2dl(long double x, long double y);
float atan2df(float x, float y);
```

**cos**

**Description:** The `cos` function returns the cosine of \( x \) measured in radians. This function may be inlined by the Itanium® compiler.

**Calling interface:**

```c
double cos(double x);
long double cosl(long double x);
float cosf(float x);
```

**cosd**

**Description:** The `cosd` function returns the cosine of \( x \) measured in degrees.

**Calling interface:**

```c
double cosd(double x);
long double cosdl(long double x);
float cosdf(float x);
```
Description: The `cot` function returns the cotangent of $x$ measured in radians.

errio: ERANGE, for overflow conditions at $x = 0$.

Calling interface:

double cot(double x);
long double cotl(long double x);
float cotf(float x);

cotd

Description: The `cotd` function returns the cotangent of $x$ measured in degrees.

errio: ERANGE, for overflow conditions at $x = 0$.

Calling interface:

double cotd(double x);
long double cotdl(long double x);
float cotdf(float x);

sin

Description: The `sin` function returns the sine of $x$ measured in radians. This function may be inlined by the Itanium® compiler.

Calling interface:

double sin(double x);
long double sinl(long double x);
float sinf(float x);

sincos

Description: The `sincos` function returns both the sine and cosine of $x$ measured in radians. This function may be inlined by the Itanium® compiler.

Calling interface:

void sincos(double x, double *sinval, double *cosval);
void sincosl(long double x, long double *sinval, long double *cosval);
void sincosf(float x, float *sinval, float *cosval);
**sincosd**

**Description:** The **sincosd** function returns both the sine and cosine of \( x \) measured in degrees.

**Calling interface:**

```c
void sincosd(double x, double *sinval, double *cosval);
void sincosdl(long double x, long double *sinval, long double *cosval);
void sincosdf(float x, float *sinval, float *cosval);
```

**sind**

**Description:** The **sind** function computes the sine of \( x \) measured in degrees.

**Calling interface:**

```c
double sind(double x);
long double sindl(long double x);
float sindf(float x);
```

**tan**

**Description:** The **tan** function returns the tangent of \( x \) measured in radians.

**Calling interface:**

```c
double tan(double x);
long double tanl(long double x);
float tanf(float x);
```

**tand**

**Description:** The **tand** function returns the tangent of \( x \) measured in degrees.

**errno:** **ERANGE**, for overflow conditions

**Calling interface:**

```c
double tand(double x);
long double tandl(long double x);
float tandf(float x);
```

**Hyperbolic Functions**
The Intel Math Library supports the following hyperbolic functions:

**acosh**

**Description:** The `acosh` function returns the inverse hyperbolic cosine of `x`.

**errno:** EDOM, for `x < 1`

**Calling interface:**

```c
double acosh(double x);
long double acoshl(long double x);
float acoshf(float x);
```

**asinh**

**Description:** The `asinh` function returns the inverse hyperbolic sine of `x`.

**Calling interface:**

```c
double asinh(double x);
long double asinhl(long double x);
float asinhf(float x);
```

**atanh**

**Description:** The `atanh` function returns the inverse hyperbolic tangent of `x`.

**errno:** EDOM, for `x > 1`

**errno:** ERANGE, for `x = 1`

**Calling interface:**

```c
double atanh(double x);
long double atanhl(long double x);
float atanhf(float x);
```

**cosh**

**Description:** The `cosh` function returns the hyperbolic cosine of `x`, \((e^x + e^{-x})/2\).

**errno:** ERANGE, for overflow conditions

**Calling interface:**

```c
double cosh(double x);
```
long double coshl(long double x);
float coshf(float x);

sinh

**Description:** The `sinh` function returns the hyperbolic sine of \( x \), \((e^x - e^{-x})/2\).

**errno:** ERANGE, for overflow conditions

**Calling interface:**

double sinh(double x);
long double sinh1(long double x);
float sinhf(float x);

sinhcosh

**Description:** The `sinhcosh` function returns both the hyperbolic sine and hyperbolic cosine of \( x \).

**errno:** ERANGE, for overflow conditions

**Calling interface:**

void sinhcosh(double x, double *sinval, double *cosval);
void sinhcosh1(long double x, long double *sinval, long double *cosval);
void sinhcoshf(float x, float *sinval, float *cosval);

tanh

**Description:** The `tanh` function returns the hyperbolic tangent of \( x \), \((e^x - e^{-x}) / (e^x + e^{-x})\).

**Calling interface:**

double tanh(double x);
long double tanhl(long double x);
float tanhf(float x);

**Exponential Functions**

The Intel Math Library supports the following exponential functions:
cbrt

**Description:** The cbrt function returns the cube root of x.

**Calling interface:**

double cbrt(double x);
long double cbrtl(long double x);
float cbrtf(float x);

exp

**Description:** The exp function returns e raised to the x power, e^x. This function may be inlined by the Itanium® compiler.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

double exp(double x);
long double expl(long double x);
float expf(float x);

exp10

**Description:** The exp10 function returns 10 raised to the x power, 10^x.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

double exp10(double x);
long double exp10l(long double x);
float exp10f(float x);

exp2

**Description:** The exp2 function returns 2 raised to the x power, 2^x.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

double exp2(double x);
long double exp2l(long double x);
float exp2f(float x);
expm1

**Description:** The `expm1` function returns $e$ raised to the $x$ power minus 1, $e^x - 1$.

**errno:** ERANGE, for overflow conditions

**Calling interface:**

double expm1(double x);
long double expm1l(long double x);
float expmlf(float x);

frexp

**Description:** The `frexp` function converts a floating-point number $x$ into signed normalized fraction in $[1/2, 1)$ multiplied by an integral power of two. The signed normalized fraction is returned, and the integer exponent stored at location `exp`.

**Calling interface:**

double frexp(double x, int *exp);
long double frexpl(long double x, int *exp);
float frexpf(float x, int *exp);

hypot

**Description:** The `hypot` function returns the square root of $(x^2 + y^2)$.

**errno:** ERANGE, for overflow conditions

**Calling interface:**

double hypot(double x, double y);
long double hypotl(long double x, long double y);
float hypotf(float x, float y);

ilogb

**Description:** The `ilogb` function returns the exponent of $x$ base two as a signed int value.

**errno:** ERANGE, for $x = 0$

**Calling interface:**

int ilogb(double x);
int ilogbl(long double x);
int ilogbf(float x);

invsqrt

**Description:** The `invsqrt` function returns the inverse square root. This function may be inlined by the Itanium® compiler.

**Calling interface:**

double invsqrt(double x);
long double invsqrtl(long double x);
float invsqrtf(float x);

ldexp

**Description:** The `ldexp` function returns \( x \cdot 2^{\text{exp}} \), where \( \text{exp} \) is an integer value.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

double ldexp(double x, int exp);
long double ldexpl(long double x, int exp);
float ldexpf(float x, int exp);

log

**Description:** The `log` function returns the natural log of \( x \), \( \ln(x) \). This function may be inlined by the Itanium® compiler.

**errno:** EDOM, for \( x < 0 \)

**errno:** ERANGE, for \( x = 0 \)

**Calling interface:**

double log(double x);
long double logl(long double x);
float logf(float x);

log10

**Description:** The `log10` function returns the base-10 log of \( x \), \( \log_{10}(x) \). This function may be inlined by the Itanium® compiler.
**errno**: EDOM, for \( x < 0 \)

**errno**: ERANGE, for \( x = 0 \)

**Calling interface**:

double log10(double x);
long double log10l(long double x);
float log10f(float x);

log1p

**Description**: The \texttt{log1p} function returns the natural log of \((x+1)\), \( \ln(x + 1) \).

**errno**: EDOM, for \( x < -1 \)

**errno**: ERANGE, for \( x = -1 \)

**Calling interface**:

double log1p(double x);
long double log1pl(long double x);
float log1pf(float x);

log2

**Description**: The \texttt{log2} function returns the base-2 log of \( x \), \( \log_2(x) \).

**errno**: EDOM, for \( x < 0 \)

**errno**: ERANGE, for \( x = 0 \)

**Calling interface**:

double log2(double x);
long double log2l(long double x);
float log2f(float x);

logb

**Description**: The \texttt{logb} function returns the signed exponent of \( x \).

**errno**: EDOM, for \( x = 0 \)

**Calling interface**:

double logb(double x);
long double logbl(long double x);
float logbf(float x);

pow

**Description:** The `pow` function returns $x$ raised to the power of $y$, $x^y$. This function may be inlined by the Itanium® compiler.

**Calling interface:**
- `errno`: EDOM, for $x = 0$ and $y < 0$
- `errno`: EDOM, for $x < 0$ and $y$ is a non-integer
- `errno`: ERANGE, for overflow and underflow conditions

**Calling interface:**
- `double pow(double x, double y);`
- `long double powl(double x, double y);`
- `float powf(float x, float y);`

scalb

**Description:** The `scalb` function returns $x \times 2^y$, where $y$ is a floating-point value.

`errno`: ERANGE, for underflow and overflow conditions

**Calling interface:**
- `double scalb(double x, double y);`
- `long double scalbl(long double x, long double y);`
- `float scalbf(float x, float y);`

scalbn

**Description:** The `scalbn` function returns $x \times 2^n$, where $n$ is an integer value.

`errno`: ERANGE, for underflow and overflow conditions

**Calling interface:**
- `double scalbn(double x, int n);`
- `long double scalbnn (long double x, int n);`
- `float scalbnf(float x, int n);`

scalbin
**Description:** The `scalbln` function returns \( x \times 2^n \), where \( n \) is a long integer value.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

```c
double scalbln(double x, long int n);
long double scalblnl (long double x, long int n);
float scalblnf(float x, long int n);
```

**sqrt**

**Description:** The `sqrt` function returns the correctly rounded square root.

**errno:** EDOM, for \( x < 0 \)

**Calling interface:**

```c
double sqrt(double x);
long double sqrtl(long double x);
float sqrtf(float x);
```

**Special Functions**

The Intel Math Library supports the following special functions:

**annuity**

**Description:** The `annuity` function computes the present value factor for an annuity, \( \frac{1 - (1+x)^{-y}}{x} \), where \( x \) is a rate and \( y \) is a period.

**errno:** ERANGE, for underflow and overflow conditions

**Calling interface:**

```c
double annuity(double x, double y);
long double annuityl(long double x, long double y);
float annuityf(float x, float y);
```

**compound**

**Description:** The `compound` function computes the compound interest factor, \( (1+x)^y \), where \( x \) is a rate and \( y \) is a period.

**errno:** ERANGE, for underflow and overflow conditions
Calling interface:
double compound(double x, double y);
long double compoundl(long double x, long double y);
float compoundf(float x, float y);

erf

Description: The erf function returns the error function value.

Calling interface:
double erf(double x);
long double erfl(long double x);
float erff(float x);

erfc

Description: The erfc function returns the complementary error function value.
errno: EDOM, for finite or infinite |x| > 1

Calling interface:
double erfc(double x);
long double erfcl(long double x);
float erfcf(float x);

erfinv

Description: The erfinv function returns the value of the inverse error function of x.
errno: EDOM, for finite or infinite |x| > 1

Calling interface:
double erfinv(double x);
long double erfinvl(long double x);
float erfinvf(float x);

gamma

Description: The gamma function returns the value of the logarithm of the absolute value of gamma.
errno: ERANGE, for overflow conditions when x is a negative integer.

**Calling interface:**

double gamma(double x);
long double gammal(long double x);
float gammaf(float x);

gamma_r

**Description:** The gamma_r function returns the value of the logarithm of the absolute value of gamma. The sign of the gamma function is returned in the integer signgam.

**Calling interface:**

double gamma_r(double x, int *signgam);
long double gammal_r(long double x, int *signgam);
float gammaf_r(float x, int *signgam);

j0

**Description:** Computes the Bessel function (of the first kind) of x with order 0.

**Calling interface:**

double j0(double x);
long double j0l(long double x);
float j0f(float x);

j1

**Description:** Computes the Bessel function (of the first kind) of x with order 1.

**Calling interface:**

double j1(double x);
long double j1l(long double x);
float j1f(float x);

jn

**Description:** Computes the Bessel function (of the first kind) of x with order n.
**Calling interface:**

double jn(int n, double x);
long double jnl(int n, long double x);
float jnf(int n, float x);

**lgamma**

**Description:** The lgamma function returns the value of the logarithm of the absolute value of gamma.

**errno:** ERANGE, for overflow conditions, x=0 or negative integers.

**Calling interface:**

double lgamma(double x);
long double lgammal(long double x);
float lgammaf(float x);

**lgamma_r**

**Description:** The lgamma_r function returns the value of the logarithm of the absolute value of gamma. The sign of the gamma function is returned in the integer signgam.

**errno:** ERANGE, for overflow conditions, x=0 or negative integers.

**Calling interface:**

double lgamma_r(double x, int *signgam);
long double lgammal_r(long double x, int *signgam);
float lgammaf_r(float x, int *signgam);

**tgamma**

**Description:** The tgamma function computes the gamma function of x.

**errno:** EDOM, for x=0 or negative integers.

**Calling interface:**

double tgamma(double x);
long double tgammaf(long double x);
float tgammaf(float x);
y0

**Description:** Computes the Bessel function (of the second kind) of \( x \) with order 0.

**errno:** EDOM, for \( x \leq 0 

**Calling interface:**

double y0(double x);
long double y0l(long double x);
float y0f(float x);

y1

**Description:** Computes the Bessel function (of the second kind) of \( x \) with order 1.

**errno:** EDOM, for \( x \leq 0 

**Calling interface:**

double y1(double x);
long double y1l(long double x);
float y1f(float x);

yn

**Description:** Computes the Bessel function (of the second kind) of \( x \) with order \( n \).

**errno:** EDOM, for \( x \leq 0 

**Calling interface:**

double yn(int n, double x);
long double ynl(int n, long double x);
float ynf(int n, float x);

**Nearest Integer Functions**

The Intel Math Library supports the following nearest integer functions:
**Description:** The **ceil** function returns the smallest integral value not less than \( x \) as a floating-point number. This function may be inlined by the Itanium® compiler.

**Calling interface:**
```c
double ceil(double x);
long double ceill(long double x);
float ceillf(float x);
```

**floor**

**Description:** The **floor** function returns the largest integral value not greater than \( x \) as a floating-point value. This function may be inlined by the Itanium® compiler.

**Calling interface:**
```c
double floor(double x);
long double floorl(long double x);
float floorf(float x);
```

**llrint**

**Description:** The **llrint** function returns the rounded integer value (according to the current rounding direction) as a **long long int**.

**errno:** **ERANGE**, for values too large

**Calling interface:**
```c
long long int llrint(double x);
long long int llrintl(long double x);
long long int llrintf(float x);
```

**llround**

**Description:** The **llround** function returns the rounded integer value as a **long long int**.

**errno:** **ERANGE**, for values too large

**Calling interface:**
```c
long long int llround(double x);
```
long long int llroundl(long double x);
long long int llroundf(float x);

\textbf{llrint}

\textbf{Description:} The \texttt{llrint} function returns the rounded integer value (according to the current rounding direction) as a \texttt{long int}.

\textbf{errno:} ERANGE, for values too large

\textbf{Calling interface:}
long int llrint(double x);
long int llrintl(long double x);
long int llrintf(float x);

\textbf{lround}

\textbf{Description:} The \texttt{lround} function returns the rounded integer value as a \texttt{long int}. Halfway cases are rounded away from zero.

\textbf{errno:} ERANGE, for values too large

\textbf{Calling interface:}
long int lround(double x);
long int lroundl(long double x);
long int lroundf(float x);

\textbf{modf}

\textbf{Description:} The \texttt{modf} function returns the value of the signed fractional part of \textit{x} and stores the integral part at \texttt{*iptr} as a floating-point number.

\textbf{Calling interface:}
double modf(double x, double *iptr);
long double modfl(long double x, long double *iptr);
float modff(float x, float *iptr);

\textbf{nearbyint}

\textbf{Description:} The \texttt{nearbyint} function returns the rounded integral value as a floating-point number, using the current rounding direction.
Calling interface:
double nearbyint(double x);
long double nearbyintl(long double x);
float nearbyintf(float x);

rint

Description: The rint function returns the rounded integral value as a floating-point number, using the current rounding direction.

Calling interface:
double rint(double x);
long double rintl(long double x);
float rintf(float x);

round

Description: The round function returns the nearest integral value as a floating-point number. Halfway cases are rounded away from zero.

Calling interface:
double round(double x);
long double roundl(long double x);
float roundf(float x);

trunc

Description: The trunc function returns the truncated integral value as a floating-point number.

Calling interface:
double trunc(double x);
long double truncl(long double x);
float truncf(float x);

Remainder Functions

The Intel Math Library supports the following remainder functions:

fmod
Description: The \texttt{fmod} function returns the value $x-n\times y$ for integer $n$ such that if $y$ is nonzero, the result has the same sign as $x$ and magnitude less than the magnitude of $y$.

\texttt{errno}: EDOM, for $y = 0$

Calling interface:

\begin{verbatim}
double fmod(double x, double y);
long double fmodl(long double x, long double y);
float fmodf(float x, float y);
\end{verbatim}

remainder

Description: The \texttt{remainder} function returns the value of $x \text{ REM } y$ as required by the IEEE standard.

Calling interface:

\begin{verbatim}
double remainder(double x, double y);
long double remainderl(long double x, long double y);
float remainderf(float x, float y);
\end{verbatim}

remquo

Description: The \texttt{remquo} function returns the value of $x \text{ REM } y$. In the object pointed to by \texttt{quo} the function stores a value whose sign is the sign of $x/y$ and whose magnitude is congruent modulo $2^n$ of the integral quotient of $x/y$. $N$ is an implementation-defined integer. For systems based on IA-64 architecture, $N$ is equal to 24. For all other systems, $N$ is equal to 31.

Calling interface:

\begin{verbatim}
double remquo(double x, double y, int *quo);
long double remquol(long double x, long double y, int *quo);
float remquof(float x, float y, int *quo);
\end{verbatim}

Miscellaneous Functions

The Intel Math Library supports the following miscellaneous functions:
copysign

**Description:** The `copysign` function returns the value with the magnitude of \( x \) and the sign of \( y \).

**Calling interface:**

```c
double copysign(double x, double y);
long double copysignl(long double x, long double y);
float copysignf(float x, float y);
```

fabs

**Description:** The `fabs` function returns the absolute value of \( x \).

**Calling interface:**

```c
double fabs(double x);
long double fabsl(long double x);
float fabsf(float x);
```

fdim

**Description:** The `fdim` function returns the positive difference value, \( x - y \) (for \( x > y \)) or +0 (for \( x \leq y \)).

**errno:** ERANGE, for values too large

**Calling interface:**

```c
double fdim(double x, double y);
long double fdiml(long double x, long double y);
float fdimf(float x, float y);
```

finite

**Description:** The `finite` function returns 1 if \( x \) is not a NaN or +/- infinity. Otherwise 0 is returned.

**Calling interface:**

```c
int finite(double x);
int finitel(long double x);
int finitef(float x);
```
fma

**Description:** The fma functions return \((x \times y) + z\).

**Calling interface:**

double fma(double x, double y, double z);
long double fmal(long double x, long double y, long double z);
float fmaf(float x, float y, float double z);

fmax

**Description:** The fmax function returns the maximum numeric value of its arguments.

**Calling interface:**

double fmax(double x, double y);
long double fmaxl(long double x, long double y);
float fmaxf(float x, float y);

fmin

**Description:** The fmin function returns the minimum numeric value of its arguments.

**Calling interface:**

double fmin(double x, double y);
long double fminl(long double x, long double y);
float fminf(float x, float y);

fpclassify

**Description:** The fpclassify function returns the value of the number classification macro appropriate to the value of its argument.

**Return Value**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>(NaN)</td>
</tr>
<tr>
<td>1</td>
<td>(Infinity)</td>
</tr>
</tbody>
</table>
Return Value

2 (Zero)

3 (Subnormal)

4 (Finite)

**Calling interface:**

double fpclassify(double x);
long double fpclassifyl(long double x);
float fpclassifyf(float x);

*isfinite*

**Description:** The *isfinite* function returns 1 if *x* is not a NaN or +/- infinity. Otherwise 0 is returned.

**Calling interface:**

int isfinite(double x);
int isfinitel(long double x);
int isfinitef(float x);

*isgreater*

**Description:** The *isgreater* function returns 1 if *x* is greater than *y*. This function does not raise the invalid floating-point exception.

**Calling interface:**

int isgreater(double x, double y);
int isgreaterl(long double x, long double y);
int isgreaterf(float x, float y);

*isgreaterequal*

**Description:** The *isgreaterequal* function returns 1 if *x* is greater than or equal to *y*. This function does not raise the invalid floating-point exception.

**Calling interface:**

int isgreaterequal(double x, double y);
int isgreatequeall(long double x, long double y);
int isgreatequealf(float x, float y);

isinf

**Description:** The `isinf` function returns a non-zero value if and only if its argument has an infinite value.

**Calling interface:**

```c
int isinf(double x);
int isinfl(long double x);
int isinff(float x);
```

isless

**Description:** The `isless` function returns 1 if `x` is less than `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**

```c
int isless(double x, double y);
int islessl(long double x, long double y);
int islesssf(float x, float y);
```

islessequal

**Description:** The `islessequal` function returns 1 if `x` is less than or equal to `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**

```c
int islessequal(double x, double y);
int islessequall(long double x, long double y);
int islessequalf(float x, float y);
```

islessgreater

**Description:** The `islessgreater` function returns 1 if `x` is less than or greater than `y`. This function does not raise the invalid floating-point exception.

**Calling interface:**

```c
int islessgreater(double x, double y);
```
int islessgreaterl(long double x, long double y);
int islessgreate rf(float x, float y);

isnan

**Description:** The `isnan` function returns a non-zero value if and only if `x` has a NaN value.

**Calling interface:**
int isnan(double x);
int isnanl(long double x);
int isnanf(float x);

isnormal

**Description:** The `isnormal` function returns a non-zero value if and only if `x` is normal.

**Calling interface:**
int isnormal(double x);
int isnormall(long double x);
int isnormalf(float x);

isunordered

**Description:** The `isunordered` function returns 1 if either `x` or `y` is a NaN. This function does not raise the invalid floating-point exception.

**Calling interface:**
int isunordered(double x, double y);
int isunorderedl(long double x, long double y);
int isunorderedf(float x, float y);

nextafter

**Description:** The `nextafter` function returns the next representable value in the specified format after `x` in the direction of `y`.

**errno:** ERANGE, for overflow and underflow conditions
**Calling interface:**

double nextafter(double x, double y);
long double nextafterl(long double x, long double y);
float nextafterf(float x, float y);

**nexttoward**

**Description:** The `nexttoward` function returns the next representable value in the specified format after `x` in the direction of `y`. If `x` equals `y`, then the function returns `y` converted to the type of the function. Use the `/Qlong-double` option on Windows* operating systems for accurate results.

**errno**: ERANGE, for overflow and underflow conditions

**Calling interface:**

double nexttoward(double x, long double y);
long double nexttowardl(long double x, long double y);
float nexttowardf(float x, long double y);

**signbit**

**Description:** The `signbit` function returns a non-zero value if and only if the sign of `x` is negative.

**Calling interface:**

int signbit(double x);
int signbitl(long double x);
int significandf(float x);

**significand**

**Description:** The `significand` function returns the significand of `x` in the interval [1,2). For `x` equal to zero, NaN, or +/- infinity, the original `x` is returned.

**Calling interface:**

double significand(double x);
long double significandl(long double x);
float significandf(float x);
Complex Functions

The Intel Math Library supports the following complex functions:

**cabs**

**Description:** The `cabs` function returns the complex absolute value of $z$.

**Calling interface:**

```c
double cabs(double _Complex z);
long double cabsl(long double _Complex z);
float cabsf(float _Complex z);
```

**cacos**

**Description:** The `cacos` function returns the complex inverse cosine of $z$.

**Calling interface:**

```c
double _Complex cacos(double _Complex z);
long double _Complex cacosl(long double _Complex z);
float _Complex cacosf(float _Complex z);
```

**cacosh**

**Description:** The `cacosh` function returns the complex inverse hyperbolic cosine of $z$.

**Calling interface:**

```c
double _Complex cacosh(double _Complex z);
long double _Complex cacoshl(long double _Complex z);
float _Complex cacoshf(float _Complex z);
```

**carg**

**Description:** The `carg` function returns the value of the argument in the interval $[-\pi, +\pi]$.

**Calling interface:**

```c
double carg(double _Complex z);
long double cargl(long double _Complex z);
float cargf(float _Complex z);
```
casin

**Description:** The *casin* function returns the complex inverse sine of *z*.

**Calling interface:**

double _Complex casin(double _Complex z);
long double _Complex casinl(long double _Complex z);
float _Complex casinf(float _Complex z);

casinh

**Description:** The *casinh* function returns the complex inverse hyperbolic sine of *z*.

**Calling interface:**

double _Complex casinh(double _Complex z);
long double _Complex casinhl(long double _Complex z);
float _Complex casinhf(float _Complex z);

catan

**Description:** The *catan* function returns the complex inverse tangent of *z*.

**Calling interface:**

double _Complex catan(double _Complex z);
long double _Complex catanl(long double _Complex z);
float _Complex catanf(float _Complex z);

catanh

**Description:** The *catanh* function returns the complex inverse hyperbolic tangent of *z*.

**Calling interface:**

double _Complex catanh(double _Complex z);
long double _Complex catanhl(long double _Complex z);
float _Complex catanhf(float _Complex z);

ccos

**Description:** The *ccos* function returns the complex cosine of *z*.
**Calling interface:**

double _Complex ccos(double _Complex z);  
long double _Complex ccosl(long double _Complex z);  
float _Complex ccosf(float _Complex z);

**ccosh**

**Description:** The `ccosh` function returns the complex hyperbolic cosine of `z`.

**Calling interface:**

double _Complex ccosh(double _Complex z);  
long double _Complex ccoshl(long double _Complex z);  
float _Complex ccoshf(float _Complex z);

**cexp**

**Description:** The `cexp` function returns \(e^z\) (\(e\) raised to the power `z`).

**Calling interface:**

double _Complex cexp(double _Complex z);  
long double _Complex cexpl(long double _Complex z);  
float _Complex cexpf(float _Complex z);

**cexp2**

**Description:** The `cexp` function returns \(2^z\) (2 raised to the power `z`).

**Calling interface:**

double _Complex cexp2(double _Complex z);  
long double _Complex cexp2l(long double _Complex z);  
float _Complex cexp2f(float _Complex z);

**cexp10**

**Description:** The `cexp10` function returns \(10^z\) (10 raised to the power `z`).

**Calling interface:**

double _Complex cexp10(double _Complex z);  
long double _Complex cexp10l(long double _Complex z);  
float _Complex cexp10f(float _Complex z);
cimag

**Description:** The `cimag` function returns the imaginary part value of \( z \).

**Calling interface:**

```c
double cimag(double _Complex z);
long double cimagl(long double _Complex z);
float cimagf(float _Complex z);
```

cis

**Description:** The `cis` function returns the cosine and sine (as a complex value) of \( z \) measured in radians.

**Calling interface:**

```c
double _Complex cis(double x);
long double _Complex cisl(long double z);
float _Complex cisf(float z);
```

cisd

**Description:** The `cisd` function returns the cosine and sine (as a complex value) of \( z \) measured in degrees.

**Calling interface:**

```c
double _Complex cisd(double x);
long double _Complex cisdl(long double z);
float _Complex cisdf(float z);
```

clog

**Description:** The `clog` function returns the complex natural logarithm of \( z \).

**Calling interface:**

```c
double _Complex clog(double _Complex z);
long double _Complex clogl(long double _Complex z);
float _Complex clogf(float _Complex z);
```

clog2

**Description:** The `clog2` function returns the complex logarithm base 2 of \( z \).
Calling interface:

double _Complex clog2(double _Complex z);
long double _Complex clog2l(long double _Complex z);
float _Complex clog2f(float _Complex z);

clog10

Description: The clog10 function returns the complex logarithm base 10 of z.

Calling interface:

double _Complex clog10(double _Complex z);
long double _Complex clog10l(long double _Complex z);
float _Complex clog10f(float _Complex z);

conj

Description: The conj function returns the complex conjugate of z by reversing the sign of its imaginary part.

Calling interface:

double _Complex conj(double _Complex z);
long double _Complex conjl(long double _Complex z);
float _Complex conjf(float _Complex z);

cpow

Description: The cpow function returns the complex power function, \( x^y \).

Calling interface:

double _Complex cpow(double _Complex x, double _Complex y);
long double _Complex cpowl(long double _Complex x, long double _Complex y);
float _Complex cpowf(float _Complex x, float _Complex y);

cproj

Description: The cproj function returns a projection of z onto the Riemann sphere.
Calling interface:

```c
double _Complex cproj(double _Complex z);
long double _Complex cprojl(long double _Complex z);
float _Complex cprojf(float _Complex z);
```

creal

**Description:** The `creal` function returns the real part of \( z \).

**Calling interface:**

```c
double creal(double _Complex z);
long double creall(long double _Complex z);
float crealf(float _Complex z);
```

csin

**Description:** The `csin` function returns the complex sine of \( z \).

**Calling interface:**

```c
double _Complex csin(double _Complex z);
long double _Complex csinl(long double _Complex z);
float _Complex csinf(float _Complex z);
```

csinh

**Description:** The `csinh` function returns the complex hyperbolic sine of \( z \).

**Calling interface:**

```c
double _Complex csinh(double _Complex z);
long double _Complex csinhl(long double _Complex z);
float _Complex csinhf(float _Complex z);
```

csqrt

**Description:** The `csqrt` function returns the complex square root of \( z \).

**Calling interface:**

```c
double _Complex csqrt(double _Complex z);
long double _Complex csqrtl(long double _Complex z);
float _Complex csqrf(float _Complex z);
```
ctan

**Description:** The `ctan` function returns the complex tangent of `z`.

**Calling interface:**

double _Complex ctan(double _Complex z);
long double _Complex ctanl(long double _Complex z);
float _Complex ctanf(float _Complex z);

cthanh

**Description:** The `ctanh` function returns the complex hyperbolic tangent of `z`.

**Calling interface:**

double _Complex ctanh(double _Complex z);
long double _Complex ctanhl(long double _Complex z);
float _Complex ctanhf(float _Complex z);

**C99 Macros**

The Intel Math Library and `mathimf.h` header file support the following C99 macros:

```c
int fpclassify(x);
int isfinite(x);
int isgreater(x, y);
int isgreaterequal(x, y);
int isinf(x);
int isless(x, y);
int islessequal(x, y);
int isnan(x);
int isnormal(x);
int isunordered(x, y);
int signbit(x);
```

**See Also**

[Miscellaneous Functions](#)
Intel C++ Class Libraries

Introduction to the Class Libraries

Overview: Intel C++ Class Libraries

The Intel C++ Class Libraries enable Single-Instruction, Multiple-Data (SIMD) operations. The principle of SIMD operations is to exploit microprocessor architecture through parallel processing. The effect of parallel processing is increased data throughput using fewer clock cycles. The objective is to improve application performance of complex and computation-intensive audio, video, and graphical data bit streams.

Hardware and Software Requirements

The Intel C++ Class Libraries are functions abstracted from the instruction extensions available on Intel processors as specified in the table that follows:

Processor Requirements for Use of Class Libraries

<table>
<thead>
<tr>
<th>Header File</th>
<th>Extension Set</th>
<th>Available on These Processors</th>
</tr>
</thead>
<tbody>
<tr>
<td>ivec.h</td>
<td>MMX™ technology</td>
<td>Intel® Pentium® processor with MMX™ technology, Intel® Pentium® II processor, Intel® Pentium® III processor, Intel® Pentium® 4 processor, Intel® Xeon® processor, and Intel® Itanium® processor</td>
</tr>
<tr>
<td>fvec.h</td>
<td>Streaming SIMD Extensions</td>
<td>Intel Pentium III processor, Intel Pentium 4 processor, Intel Xeon processor, and Intel Itanium processor</td>
</tr>
<tr>
<td>dvec.h</td>
<td>Streaming SIMD Extensions 2</td>
<td>Intel Pentium 4 processor and Intel Xeon processors</td>
</tr>
</tbody>
</table>

About the Classes
The Intel C++ Class Libraries for SIMD Operations include:

- Integer vector (Ivec) classes
- Floating-point vector (Fvec) classes

You can find the definitions for these operations in three header files: ivec.h, fvec.h, and dvec.h. The classes themselves are not partitioned like this. The classes are named according to the underlying type of operation. The header files are partitioned according to architecture:

- ivec.h is specific to architectures with MMX™ technology
- fvec.h is specific to architectures with Streaming SIMD Extensions
- dvec.h is specific to architectures with Streaming SIMD Extensions 2

Streaming SIMD Extensions 2 intrinsics cannot be used on IA-64 architecture based systems. The mmclass.h header file includes the classes that are usable on the IA-64 architecture.

This documentation is intended for programmers writing code for the Intel architecture, particularly code that would benefit from the use of SIMD instructions. You should be familiar with C++ and the use of C++ classes.

Details About the Libraries

The Intel C++ Class Libraries for SIMD Operations provide a convenient interface to access the underlying instructions for processors as specified in Processor Requirements for Use of Class Libraries. These processor-instruction extensions enable parallel processing using the single instruction-multiple data (SIMD) technique as illustrated in the following figure.

**SIMD Data Flow**

```
<table>
<thead>
<tr>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3</td>
<td>B2</td>
<td>B1</td>
<td>B0</td>
</tr>
</tbody>
</table>
```

```
A3 op B3 A2 op B2 A1 op B1 A0 op B0
```
Performing four operations with a single instruction improves efficiency by a factor of four for that particular instruction. These new processor instructions can be implemented using assembly inlining, intrinsics, or the C++ SIMD classes. Compare the coding required to add four 32-bit floating-point values, using each of the available interfaces:

<table>
<thead>
<tr>
<th>Assembly Inlining</th>
<th>Intrinsics</th>
<th>SIMD Class Libraries</th>
</tr>
</thead>
<tbody>
<tr>
<td>... __m128 a,b,c;</td>
<td>#include &lt;mmintrin.h&gt;</td>
<td>#include &lt;fvec.h&gt;</td>
</tr>
<tr>
<td>__asm{ movaps __m128 a,b,c; a = b +c; ... }</td>
<td>... F32vec4 a,b,c; a = b +c; ...</td>
<td></td>
</tr>
</tbody>
</table>

This table shows an addition of two single-precision floating-point values using assembly inlining, intrinsics, and the libraries. You can see how much easier it is to code with the Intel C++ SIMD Class Libraries. Besides using fewer keystrokes and fewer lines of code, the notation is like the standard notation in C++, making it much easier to implement over other methods.

**C++ Classes and SIMD Operations**

The use of C++ classes for SIMD operations is based on the concept of operating on arrays, or vectors of data, in parallel. Consider the addition of two vectors, A and B, where each vector contains four elements. Using the integer vector (Ivec) class, the elements A[i] and B[i] from each array are summed as shown in the following example.

**Typical Method of Adding Elements Using a Loop**

```cpp
short a[4], b[4], c[4];
```
for (i=0; i<4; i++) /* needs four iterations */
c[i] = a[i] + b[i]; /* returns c[0], c[1], c[2], c[3] */

The following example shows the same results using one operation with Ivec Classes.

SIMD Method of Adding Elements Using Ivec Classes

sIs16vec4 ivecA, ivecB, ivec C; /*needs one iteration*/
ivecC = ivecA + ivecB; /*returns ivecC0, ivecC1, ivecC2, ivecC3 */

Available Classes

The Intel C++ SIMD classes provide parallelism, which is not easily implemented using typical mechanisms of C++. The following table shows how the Intel C++ SIMD classes use the classes and libraries.

### SIMD Vector Classes

<table>
<thead>
<tr>
<th>Instruction Set</th>
<th>Class</th>
<th>Signedness</th>
<th>Data Type</th>
<th>Size</th>
<th>Elements</th>
<th>Header File</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX™ technology</td>
<td>I64vec1</td>
<td>unspecified</td>
<td>__m64</td>
<td>64</td>
<td>1</td>
<td>ivec.h</td>
</tr>
<tr>
<td>I32vec2</td>
<td></td>
<td>unspecified</td>
<td>int</td>
<td>32</td>
<td>2</td>
<td>ivec.h</td>
</tr>
<tr>
<td>Is32vec2</td>
<td>signed</td>
<td>int</td>
<td>32</td>
<td>2</td>
<td>ivec.h</td>
<td></td>
</tr>
<tr>
<td>Iu32vec2</td>
<td>unsigned</td>
<td>int</td>
<td>32</td>
<td>2</td>
<td>ivec.h</td>
<td></td>
</tr>
<tr>
<td>I16vec4</td>
<td></td>
<td>unspecified</td>
<td>short</td>
<td>16</td>
<td>4</td>
<td>ivec.h</td>
</tr>
<tr>
<td>Is16vec4</td>
<td>signed</td>
<td>short</td>
<td>16</td>
<td>4</td>
<td>ivec.h</td>
<td></td>
</tr>
<tr>
<td>Iu16vec4</td>
<td>unsigned</td>
<td>short</td>
<td>16</td>
<td>4</td>
<td>ivec.h</td>
<td></td>
</tr>
<tr>
<td>I8vec8</td>
<td></td>
<td>unspecified</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>ivec.h</td>
</tr>
<tr>
<td>Is8vec8</td>
<td>signed</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>ivec.h</td>
<td></td>
</tr>
<tr>
<td>Iu8vec8</td>
<td>unsigned</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>ivec.h</td>
<td></td>
</tr>
<tr>
<td>Instruction Set</td>
<td>Class</td>
<td>Signedness</td>
<td>Data Type</td>
<td>Size</td>
<td>Elements</td>
<td>Header File</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
<td>------------</td>
<td>-----------</td>
<td>------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>Streaming</td>
<td>F32vec4</td>
<td>signed</td>
<td>float</td>
<td>32</td>
<td>4</td>
<td>fvec.h</td>
</tr>
<tr>
<td>SIMD Extensions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Streaming</td>
<td>F32vec1</td>
<td>signed</td>
<td>float</td>
<td>32</td>
<td>1</td>
<td>fvec.h</td>
</tr>
<tr>
<td>SIMD Extensions</td>
<td>F64vec2</td>
<td>signed</td>
<td>double</td>
<td>64</td>
<td>2</td>
<td>dvec.h</td>
</tr>
<tr>
<td>Extensions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I128vec1</td>
<td>unspecified</td>
<td>_m128i</td>
<td>128</td>
<td>1</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I64vec2</td>
<td>unspecified</td>
<td>long</td>
<td>64</td>
<td>4</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I64vec2</td>
<td>unspecified</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is64vec2</td>
<td>signed</td>
<td>long</td>
<td>64</td>
<td>4</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>Is64vec2</td>
<td>signed</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Iu64vec2</td>
<td>unsigned</td>
<td>long</td>
<td>32</td>
<td>4</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>Iu64vec2</td>
<td>unsigned</td>
<td>int</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I32vec4</td>
<td>unspecified</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I32vec4</td>
<td>signed</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I32vec4</td>
<td>unsigned</td>
<td>int</td>
<td>32</td>
<td>4</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I16vec8</td>
<td>unspecified</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I16vec8</td>
<td>signed</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I16vec8</td>
<td>unsigned</td>
<td>int</td>
<td>16</td>
<td>8</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I8vec16</td>
<td>unspecified</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I8vec16</td>
<td>signed</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td></td>
<td>dvec.h</td>
</tr>
<tr>
<td>I8vec16</td>
<td>signed</td>
<td>char</td>
<td>8</td>
<td>16</td>
<td></td>
<td>dvec.h</td>
</tr>
</tbody>
</table>
Most classes contain similar functionality for all data types and are represented by all available intrinsics. However, some capabilities do not translate from one data type to another without suffering from poor performance, and are therefore excluded from individual classes.

**Note**

Intrinsics that take immediate values and cannot be expressed easily in classes are not implemented.

(For example, _mm_shuffle_ps, _mm_shuffle_pi16, _mm_shuffle_ps, _mm_extract_pi16, _mm_insert_pi16).

---

### Access to Classes Using Header Files

The required class header files are installed in the include directory with the Intel® C++ Compiler. To enable the classes, use the `#include` directive in your program file as shown in the table that follows.

#### Include Directives for Enabling Classes

<table>
<thead>
<tr>
<th>Instruction Set Extension</th>
<th>Include Directive</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMX Technology</td>
<td><code>#include &lt;ivec.h&gt;</code></td>
</tr>
<tr>
<td>Streaming SIMD Extensions</td>
<td><code>#include &lt;fvec.h&gt;</code></td>
</tr>
<tr>
<td>Streaming SIMD Extensions 2</td>
<td><code>#include &lt;dvec.h&gt;</code></td>
</tr>
</tbody>
</table>

Each succeeding file from the top down includes the preceding class. You only need to include `fvec.h` if you want to use both the Ivec and Fvec classes. Similarly, to use all the classes including those for the Streaming SIMD Extensions 2, you need only to include the `dvec.h` file.

**Usage Precautions**
When using the C++ classes, you should follow some general guidelines. More detailed usage rules for each class are listed in Integer Vector Classes, and Floating-point Vector Classes.

**Clear MMX Registers**

If you use both the Ivec and Fvec classes at the same time, your program could mix MMX instructions, called by Ivec classes, with Intel x87 architecture floating-point instructions, called by Fvec classes. Floating-point instructions exist in the following Fvec functions:

- **fvec constructors**
- **debug functions (cout and element access)**
- **rsqrt_nr**

⚠️ **Note**

MMX registers are aliased on the floating-point registers, so you should clear the MMX state with the EMMS instruction intrinsic before issuing an x87 floating-point instruction, as in the following example.

```cpp
ivecA = ivecA & ivecB;  // Ivec logical operation that uses MMX instructions
empty ();                // clear state
cout << f32vec4a;       // F32vec4 operation that uses x87 floating-point instructions
```

⚠️ **Caution**

Failure to clear the MMX registers can result in incorrect execution or poor performance due to an incorrect register state.

**Follow EMMS Instruction Guidelines**

Intel strongly recommends that you follow the guidelines for using the EMMS instruction. Refer to this topic before coding with the Ivec classes.

**Capabilities**

The fundamental capabilities of each C++ SIMD class include:
Understanding each of these capabilities and how they interact is crucial to achieving desired results.

### Computation

The SIMD C++ classes contain vertical operator support for most arithmetic operations, including shifting and saturation. Computation operations include: $+,-,\times,/,\text{reciprocal}$ ($\text{rcp}$ and $\text{rcp}_{\text{nr}}$), square root ($\text{sqrt}$), reciprocal square root ($\text{rsqrt}$ and $\text{rsqrt}_{\text{nr}}$). Operations $\text{rcp}$ and $\text{rsqrt}$ are new approximating instructions with very short latencies that produce results with at least 12 bits of accuracy. Operations $\text{rcp}_{\text{nr}}$ and $\text{rsqrt}_{\text{nr}}$ use software refining techniques to enhance the accuracy of the approximations, with a minimal impact on performance. (The "nr" stands for Newton-Raphson, a mathematical technique for improving performance using an approximate result.)

### Horizontal Data Support

The C++ SIMD classes provide horizontal support for some arithmetic operations. The term "horizontal" indicates computation across the elements of one vector, as opposed to the vertical, element-by-element operations on two different vectors. The $\text{add}_{\text{horizontal}}$, $\text{unpack}_{\text{low}}$, and $\text{pack}_{\text{sat}}$ functions are examples of horizontal data support. This support enables certain algorithms that cannot exploit the full potential of SIMD instructions. Shuffle intrinsics are another example of horizontal data flow. Shuffle intrinsics are not expressed in the C++ classes due to their immediate arguments. However, the C++ class implementation enables you to mix shuffle intrinsics with the other C++ functions. For example:
F32vec4 fveca, fvecb, fvecd;
fveca += fvecb;
fvecd = _mm_shuffle_ps(fveca, fvecb, 0);

Typically every instruction with horizontal data flow contains some inefficiency in the implementation. If possible, implement your algorithms without using the horizontal capabilities.

Branch Compression/Elimination

Branching in SIMD architectures can be complicated and expensive, possibly resulting in poor predictability and code expansion. The SIMD C++ classes provide functions to eliminate branches, using logical operations, max and min functions, conditional selects, and compares. Consider the following example:

```c
short a[4], b[4], c[4];
for (i=0; i<4; i++)
c[i] = a[i] > b[i] ? a[i] : b[i];
```

This operation is independent of the value of $i$. For each $i$, the result could be either A or B depending on the actual values. A simple way of removing the branch altogether is to use the `select_gt` function, as follows:

```c
Isl6vec4 a, b, c
c = select_gt(a, b, a, b)
```

Caching Hints

Streaming SIMD Extensions provide prefetching and streaming hints. Prefetching data can minimize the effects of memory latency. Streaming hints allow you to indicate that certain data should not be cached. This results in higher performance for data that should be cached.

**Integer Vector Classes**

**Overview: Integer Vector Classes**
The Ivec classes provide an interface to SIMD processing using integer vectors of various sizes. The class hierarchy is represented in the following figure.

**Ivec Class Hierarchy**

The M64 and M128 classes define the _m64 and _m128i data types from which the rest of the Ivec classes are derived. The first generation of child classes are derived based solely on bit sizes of 128, 64, 32, 16, and 8 respectively for the I128vec1, I64vec1, I64vec2, I32vec2, I32vec4, I16vec4, I16vec8, I8vec16, and I8vec8 classes. The latter seven of these classes require specification of signedness and saturation.

⚠️ Caution

Do not intermix the M64 and M128 data types. You will get unexpected behavior if you do.

The signedness is indicated by the s and u in the class names:

Is64vec2
Iu64vec2
Is32vec4
Iu32vec4
Is16vec8
Iu16vec8
Is8vec16
Iu8vec16
Is32vec2
Iu32vec2
Terms, Conventions, and Syntax Defined

The following are special terms and syntax used in this chapter to describe functionality of the classes with respect to their associated operations.

Ivec Class Syntax Conventions

The name of each class denotes the data type, signedness, bit size, number of elements using the following generic format:

\(<type><signedness><bits>vec<elements>\)

\({ F | I } { s | u } { 64 | 32 | 16 | 8 } vec { 8 | 4 | 2 | 1 }\)

where

- **type** indicates floating point (\(F\)) or integer (\(I\))
- **signedness** indicates signed (\(s\)) or unsigned (\(u\)). For the Ivec class, leaving this field blank indicates an intermediate class. There are no unsigned Fvec classes, therefore for the Fvec classes, this field is blank.
- **bits** specifies the number of bits per element
- **elements** specifies the number of elements

Special Terms and Conventions

The following terms are used to define the functionality and characteristics of the classes and operations defined in this manual.

- **Nearest Common Ancestor** -- This is the intermediate or parent class of two classes of the same size. For example, the nearest common ancestor of
Iu8vec8 and Is8vec8 is I8vec8. Also, the nearest common ancestor between Iu8vec8 and I16vec4 is M64.

- **Casting** -- Changes the data type from one class to another. When an operation uses different data types as operands, the return value of the operation must be assigned to a single data type. Therefore, one or more of the data types must be converted to a required data type. This conversion is known as a typecast. Sometimes, typecasting is automatic, other times you must use special syntax to explicitly typecast it yourself.

- **Operator Overloading** -- This is the ability to use various operators on the same user-defined data type of a given class. Once you declare a variable, you can add, subtract, multiply, and perform a range of operations. Each family of classes accepts a specified range of operators, and must comply by rules and restrictions regarding typecasting and operator overloading as defined in the header files. The following table shows the notation used in this documentation to address typecasting, operator overloading, and other rules.

### Class Syntax Notation Conventions

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[s</td>
<td>u][N]vec[N]</td>
</tr>
<tr>
<td>I64vec1</td>
<td>__m64 data type</td>
</tr>
<tr>
<td>I[s</td>
<td>u]64vec2</td>
</tr>
<tr>
<td>I[s</td>
<td>u]32vec4</td>
</tr>
<tr>
<td>I[s</td>
<td>u]8vec16</td>
</tr>
<tr>
<td>I[s</td>
<td>u]16vec8</td>
</tr>
<tr>
<td>I[s</td>
<td>u]32vec2</td>
</tr>
<tr>
<td>I[s</td>
<td>u]16vec4</td>
</tr>
<tr>
<td>I[s</td>
<td>u]8vec8</td>
</tr>
</tbody>
</table>
Rules for Operators

To use operators with the Ivec classes you must use one of the following three syntax conventions:

```
```

Example 1: `I64vec1 R = I64vec1 A & I64vec1 B;`
```
```

Example 2: `I64vec1 R = andnot(I64vec1 A, I64vec1 B);`
```
[ Ivec_Class ] R [ operator ]= [ Ivec_Class ] A
```

Example 3: `I64vec1 R &= I64vec1 A;`
```
[ operator ] an operator (for example, &, |, or ^ )
```

[ Ivec_Class ] an Ivec class

R, A, B variables declared using the pertinent Ivec classes

The table that follows shows automatic and explicit sign and size typecasting.

"Explicit" means that it is illegal to mix different types without an explicit typecasting. "Automatic" means that you can mix types freely and the compiler will do the typecasting for you.

Summary of Rules Major Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Sign Typecasting</th>
<th>Size Typecasting</th>
<th>Other Typecasting Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>Logical</td>
<td>Automatic</td>
<td>Automatic (to left)</td>
<td>Explicit typecasting is required for different types used in non-logical expressions on the right side of the assignment.</td>
</tr>
</tbody>
</table>
Addition and Subtraction                      | Automatic | Explicit | N/A
Multiplication                              | Automatic | Explicit | N/A
Shift                                        | Automatic | Explicit | Casting Required to ensure arithmetic shift.
Compare                                      | Automatic | Explicit | Explicit casting is required for signed classes for the less-than or greater-than operations.
Conditional Select                          | Automatic | Explicit | Explicit casting is required for signed classes for less-than or greater-than operations.

Data Declaration and Initialization

The following table shows literal examples of constructor declarations and data type initialization for all class sizes. All values are initialized with the most significant element on the left and the least significant to the right.

Declaration and Initialization Data Types for Ivec Classes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Class</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>M128</td>
<td>I128vec1 A; Iu8vec16 A;</td>
</tr>
</tbody>
</table>

1594
<table>
<thead>
<tr>
<th>Operation</th>
<th>Class</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declaration</td>
<td>M64</td>
<td>I64vec1 A; Iu8vec16 A;</td>
</tr>
<tr>
<td>__m128 Initialization</td>
<td>M128</td>
<td>I128vec1 A(__m128 m); Iu16vec8(__m128 m);</td>
</tr>
<tr>
<td>__m64 Initialization</td>
<td>M64</td>
<td>I64vec1 A(__m64 m); Iu8vec8 A(__m64 m);</td>
</tr>
<tr>
<td>__int64 Initialization</td>
<td>M64</td>
<td>I64vec1 A = __int64 m; Iu8vec8 A = __int64 m;</td>
</tr>
<tr>
<td>int i Initialization</td>
<td>M64</td>
<td>I64vec1 A = int i; Iu8vec8 A = int i;</td>
</tr>
<tr>
<td>int Initialization</td>
<td>I32vec2</td>
<td>I32vec2 A(int A1, int A0); Is32vec2 A(signed int A1, signed int A0); Iu32vec2 A(unsigned int A1, unsigned int A0);</td>
</tr>
<tr>
<td>int Initialization</td>
<td>I32vec4</td>
<td>I32vec4 A(short A3, short A2, short A1, short A0); Is32vec4 A(signed short A3, ..., signed short A3, ...); Iu32vec4 A(unsigned int A1, unsigned int A0);</td>
</tr>
<tr>
<td>Operation</td>
<td>Class</td>
<td>Syntax</td>
</tr>
<tr>
<td>--------------------</td>
<td>----------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>short int Initialization</td>
<td>I16vec4</td>
<td><code>I16vec4 A(short A3, short A2, short A1, short A0);</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Is16vec4 A(signed short A3, ..., signed short A0);</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Iu16vec4 A(unsinged short A3, ..., unsinged short A0);</code></td>
</tr>
<tr>
<td>short int Initialization</td>
<td>I16vec8</td>
<td><code>I16vec8 A(short A7, short A6, ..., short A1, short A0);</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Is16vec8 A(signed A7, ..., signed short A0);</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Iu16vec8 A(unsinged short A7, ..., unsinged short A0);</code></td>
</tr>
<tr>
<td>char Initialization</td>
<td>I8vec8</td>
<td><code>I8vec8 A(char A7, char A6, ..., char A1, char A0);</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Is8vec8 A(signed char A7, ..., signed char A0);</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>Iu8vec8 A(unsinged char A7, ..., unsinged char A0);</code></td>
</tr>
<tr>
<td>Operation</td>
<td>Class</td>
<td>Syntax</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>char Initialization</td>
<td>I8vec16</td>
<td>I8vec16 A(char A15, ..., char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Is8vec16 A(signed char A15, ..., signed char A0);</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Iu8vec16 A(unsigned char A15, ..., unsigned char A0);</td>
</tr>
</tbody>
</table>

**Assignment Operator**

Any `Ivec` object can be assigned to any other `Ivec` object; conversion on assignment from one `Ivec` object to another is automatic.

**Assignment Operator Examples**

```cpp
Is16vec4 A;
Is8vec8 B;
I64vec1 C;
A = B; /* assign Is8vec8 to Is16vec4 */
B = C; /* assign I64vec1 to Is8vec8 */
B = A & C; /* assign M64 result of '&' to Is8vec8 */
```

**Logical Operators**

The logical operators use the symbols and intrinsics listed in the following table.

<table>
<thead>
<tr>
<th>Bitwise Operation</th>
<th>Operator Symbols</th>
<th>Syntax Usage</th>
<th>Corresponding Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standard</td>
<td>w/assign</td>
<td>Standard</td>
</tr>
<tr>
<td>AND</td>
<td>&amp;</td>
<td></td>
<td>R = A &amp; R &amp; = A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>OR</td>
<td></td>
<td></td>
<td>R = A</td>
</tr>
</tbody>
</table>
Bitwise Operation | Operator Symbols | Syntax Usage | Corresponding Intrinsic
--- | --- | --- | ---
| | Standard | w/assign | Standard | w/assign |
| XOR | ^ | ^= | B | _mm_and_si128 |
| ANDNOT | andnot | N/A | R = A ^ B | _mm_and_si64 |
| | | | R ^= A | _mm_and_si128 |
| | | | R = A | _mm_and_si64 |
| | | | N/A | _mm_and_si128 |
| | | | andnot B | _mm_and_si128 |

Logical Operators and Miscellaneous Exceptions

A and B converted to M64. Result assigned to Iu8vec8.

```c
I64vec1 A;
Is8vec8 B;
Iu8vec8 C;
C = A & B;
```

Same size and signedness operators return the nearest common ancestor.

```c
I32vec2 R = Is32vec2 A ^ Iu32vec2 B;
A&B returns M64, which is cast to Iu8vec8.
C = Iu8vec8(A&B) + C;
```

When A and B are of the same class, they return the same type. When A and B are of different classes, the return value is the return type of the nearest common ancestor.

The logical operator returns values for combinations of classes, listed in the following tables, apply when A and B are of different classes.

Ivec Logical Operator Overloading

<table>
<thead>
<tr>
<th>Return (R)</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>NAND</th>
<th>A Operand</th>
<th>B Operand</th>
</tr>
</thead>
<tbody>
<tr>
<td>I64vec1 &amp;</td>
<td></td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s</td>
<td>u]64vec2 I[s</td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td>B</td>
</tr>
<tr>
<td>I64vec2 &amp;</td>
<td></td>
<td></td>
<td>^</td>
<td>andnot</td>
<td>I[s</td>
<td>u]64vec2 I[s</td>
</tr>
</tbody>
</table>

1598
For logical operators with assignment, the return value of R is always the same data type as the pre-declared value of R as listed in the table that follows.

Ivec Logical Operator Overloading with Assignment

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Left Side (R)</th>
<th>AND</th>
<th>OR</th>
<th>XOR</th>
<th>Right Side (Any Ivec Type)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I128vec1</td>
<td>I128vec1</td>
<td>R &amp;=</td>
<td></td>
<td></td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec1</td>
<td>I64vec1</td>
<td>R &amp;=</td>
<td></td>
<td></td>
<td>I[s</td>
</tr>
<tr>
<td>I64vec2</td>
<td>I64vec2</td>
<td>R &amp;=</td>
<td></td>
<td></td>
<td>I[s</td>
</tr>
<tr>
<td>Return Type</td>
<td>Left Side (R) AND</td>
<td>OR</td>
<td>XOR</td>
<td>Right Side (Any Ivec Type)</td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>------------------</td>
<td>----</td>
<td>-----</td>
<td>---------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I[x]32vec2</td>
<td>I[x]32vec2 &amp;</td>
<td>=   ^=</td>
<td>I[s</td>
<td>u][N]vec[N] A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I[x]16vec8</td>
<td>I[x]16vec8 &amp;</td>
<td>=   ^=</td>
<td>I[s</td>
<td>u][N]vec[N] A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I[x]8vec16</td>
<td>I[x]8vec16 &amp;</td>
<td>=   ^=</td>
<td>I[s</td>
<td>u][N]vec[N] A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I[x]8vec8</td>
<td>I[x]8vec8 &amp;</td>
<td>=   ^=</td>
<td>I[s</td>
<td>u][N]vec[N] A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Addition and Subtraction Operators**

The addition and subtraction operators return the class of the nearest common ancestor when the right-side operands are of different signs. The following code provides examples of usage and miscellaneous exceptions.

**Syntax Usage for Addition and Subtraction Operators**

Return nearest common ancestor type, I16vec4.

Iu16vec4 A;
Iu16vec4 B;
Iu16vec4 C;
C = A + B;

Returns type left-hand operand type.

Is16vec4 A;
Is16vec4 B;
A += B;
B -= A;

Explicitly convert B to Is16vec4.

Is16vec4 A,C;
Iu32vec24 B;
C = A + C;
C = A + (Is16vec4)B;

**Addition and Subtraction Operators with Corresponding Intrinsics**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbols</th>
<th>Syntax</th>
<th>Corresponding Intrinsics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>+</td>
<td>R = A + B</td>
<td>_mm_add_epi64</td>
</tr>
<tr>
<td></td>
<td>+=</td>
<td>R += A</td>
<td>_mm_add_epi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_epi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_epi8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_add_pi8</td>
</tr>
<tr>
<td>Subtraction</td>
<td>-</td>
<td>R = A - B</td>
<td>_mm_sub_epi64</td>
</tr>
<tr>
<td></td>
<td>-=</td>
<td>R -= A</td>
<td>_mm_sub_epi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_epi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_epi8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_pi32</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_pi16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sub_pi8</td>
</tr>
</tbody>
</table>

The following table lists addition and subtraction return values for combinations of classes when the right side operands are of different signedness. The two operands must be the same size, otherwise you must explicitly indicate the typecasting.

**Addition and Subtraction Operator Overloading**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Available Operators</th>
<th>Right Side Operands</th>
</tr>
</thead>
</table>
The following table shows the return data type values for operands of the addition and subtraction operators with assignment. The left side operand determines the size and signedness of the return value. The right side operand must be the same size as the left operand; otherwise, you must use an explicit typecast.

**Addition and Subtraction with Assignment**

<table>
<thead>
<tr>
<th>Return Value (R)</th>
<th>Left Side (R)</th>
<th>Add</th>
<th>Sub</th>
<th>Right Side (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>I[x]32vec4 R</td>
<td>I[x]32vec2</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A;</td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>I[x]32vec2 R</td>
<td>I[x]32vec2</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A;</td>
<td></td>
<td>A;</td>
</tr>
<tr>
<td>Return Value (R)</td>
<td>Left Side (R)</td>
<td>Add</td>
<td>Sub</td>
<td>Right Side (A)</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------</td>
<td>-----</td>
<td>-----</td>
<td>----------------</td>
</tr>
<tr>
<td>I[x]16vec8</td>
<td>I[x]16vec8</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]16vec4</td>
<td>I[x]16vec4</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]8vec16</td>
<td>I[x]8vec16</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
<tr>
<td>I[x]8vec8</td>
<td>I[x]8vec8</td>
<td>+=</td>
<td>-=</td>
<td>I[s</td>
</tr>
</tbody>
</table>

**Multiplication Operators**

The multiplication operators can only accept and return data types from the I[s|u]16vec4 or I[s|u]16vec8 classes, as shown in the following example.

**Syntax Usage for Multiplication Operators**

Explicitly convert B to Is16vec4.

Is16vec4 A,C;
Iu32vec2 B;
C = A * C;
C = A * (Is16vec4)B;

Return nearest common ancestor type, I16vec4

Is16vec4 A;
Iu16vec4 B;
I16vec4 C;
C = A + B;

The mul_high and mul_add functions take Is16vec4 data only.

Is16vec4 A,B,C,D;
C = mul_high(A,B);
D = mul_add(A,B);
Multiplication Operators with Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Symbols</th>
<th>Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>R = A * B</td>
<td>_mm_mullo_pi16</td>
</tr>
<tr>
<td></td>
<td>R *= A</td>
<td>_mm_mullo_epi16</td>
</tr>
<tr>
<td>mul_high</td>
<td>N/A</td>
<td>R = _mm_mulhi_pi16</td>
</tr>
<tr>
<td></td>
<td>mul_high(A, B)</td>
<td>_mm_mulhi_epi16</td>
</tr>
<tr>
<td>mul_add</td>
<td>N/A</td>
<td>R = _mm_madd_pi16</td>
</tr>
<tr>
<td></td>
<td>mul_high(A, B)</td>
<td>_mm_madd_epi16</td>
</tr>
</tbody>
</table>

The multiplication return operators always return the nearest common ancestor as listed in the table that follows. The two operands must be 16 bits in size, otherwise you must explicitly indicate typecasting.

Multiplication Operator Overloading

<table>
<thead>
<tr>
<th>R</th>
<th>Mul</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>I16vec4</td>
<td>*</td>
<td>I[s</td>
<td>u]16vec4 A</td>
</tr>
<tr>
<td>I16vec8</td>
<td>*</td>
<td>I[s</td>
<td>u]16vec8 A</td>
</tr>
<tr>
<td>Is16vec4</td>
<td>mul_add</td>
<td>Is16vec4 A</td>
<td>Is16vec4 B</td>
</tr>
<tr>
<td>Is16vec8</td>
<td>mul_add</td>
<td>Is16vec8 A</td>
<td>Is16vec8 B</td>
</tr>
<tr>
<td>Is32vec2</td>
<td>mul_high</td>
<td>Is16vec4 A</td>
<td>Is16vec4 B</td>
</tr>
<tr>
<td>Is32vec4</td>
<td>mul_high</td>
<td>s16vec8 A</td>
<td>Is16vec8 B</td>
</tr>
</tbody>
</table>

The following table shows the return values and data type assignments for operands of the multiplication operators with assignment. All operands must be 16 bytes in size. If the operands are not the right size, you must use an explicit typecast.

Multiplication with Assignment

<table>
<thead>
<tr>
<th>Return Value (R)</th>
<th>Left Side (R)</th>
<th>Mul</th>
<th>Right Side (A)</th>
</tr>
</thead>
</table>

Shift Operators

The right shift argument can be any integer or Ivec value, and is implicitly converted to a M64 data type. The first or left operand of a << can be of any type except I[s|u]8vec[8|16].

Example Syntax Usage for Shift Operators

Automatic size and sign conversion.

```c++
Is16vec4 A, C;
Iu32vec2 B;
C = A;
```

A&B returns I16vec4, which must be cast to Iu16vec4 to ensure logical shift, not arithmetic shift.

```c++
Is16vec4 A, C;
Iu16vec4 B, R;
R = (Iu16vec4)(A & B) C;
```

A&B returns I16vec4, which must be cast to Is16vec4 to ensure arithmetic shift, not logical shift.

```c++
R = (Is16vec4)(A & B) C;
```

Shift Operators with Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Operation</th>
<th>Symbols</th>
<th>Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shift Left</td>
<td>&lt;&lt;</td>
<td>R = A &lt;&lt; B</td>
<td>_mm_sll_si64</td>
</tr>
<tr>
<td></td>
<td>&amp;=</td>
<td>R &amp;= A</td>
<td>_mm_slli_si64</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>_mm_sll_pi32</td>
</tr>
</tbody>
</table>
Right shift operations with signed data types use arithmetic shifts. All unsigned and intermediate classes correspond to logical shifts. The following table shows how the return type is determined by the first argument type.

**Shift Operator Overloading**

<table>
<thead>
<tr>
<th>Option</th>
<th>R</th>
<th>Right Shift</th>
<th>Left Shift</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>I64vec1</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;=</td>
<td>I64vec1</td>
<td>I64vec1 B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;=</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>I32vec2</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;=</td>
<td>I32vec2</td>
<td>I32vec2 B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>&lt;=</td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Is32vec2</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;=</td>
<td>Is32vec2</td>
<td>I[s</td>
</tr>
<tr>
<td>Logical</td>
<td>Iu32vec2</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;=</td>
<td>Iu32vec2</td>
<td>I[s</td>
</tr>
<tr>
<td>Logical</td>
<td>I16vec4</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;=</td>
<td>I16vec4</td>
<td>I16vec4 B</td>
</tr>
<tr>
<td>Option</td>
<td>R</td>
<td>Right Shift</td>
<td>Left Shift</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>----------</td>
<td>---</td>
<td>-------------</td>
<td>------------</td>
<td>--------------</td>
<td>--------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td>Arithmetic</td>
<td>Is16vec4</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;=</td>
<td>&lt;&lt;=</td>
<td>Is16vec4</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td>B;</td>
<td></td>
</tr>
<tr>
<td>Logical</td>
<td>Iu16vec4</td>
<td>&gt;&gt;</td>
<td>&gt;&gt;=</td>
<td>&lt;&lt;=</td>
<td>Iu16vec4</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td>B;</td>
<td></td>
</tr>
</tbody>
</table>

**Comparison Operators**

The equality and inequality comparison operands can have mixed signedness, but they must be of the same size. The comparison operators for less-than and greater-than must be of the same sign and size.

**Example of Syntax Usage for Comparison Operator**

The nearest common ancestor is returned for compare for equal/not-equal operations.

```cpp
Iu8vec8 A;
Is8vec8 B;
I8vec8 C;
C = cmpeq(A,B);
```

Type cast needed for different-sized elements for equal/not-equal comparisons.

```cpp
Iu8vec8 A, C;
Is16vec4 B;
C = cmpeq(A,(Iu8vec8)B);
```

Type cast needed for sign or size differences for less-than and greater-than comparisons.

```cpp
Iu16vec4 A;
Is16vec4 B, C;
C = cmpge((Is16vec4)A,B);
C = cmpgt(B,C);
```

**Inequality Comparison Symbols and Corresponding Intrinsics**
### Comparison operators have the restriction that the operands must be the size and sign as listed in the Compare Operator Overloading table.

#### Compare Operator Overloading

<table>
<thead>
<tr>
<th>R</th>
<th>Comparison</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>I32vec2</code></td>
<td>cmpeq</td>
<td>`I[s</td>
<td>u]32vec2`</td>
</tr>
<tr>
<td></td>
<td>cmpne</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conditional Select Operators

For conditional select operands, the third and fourth operands determine the type returned. Third and fourth operands with same size, but different signedness, return the nearest common ancestor data type.

Conditional Select Syntax Usage

Return the nearest common ancestor data type if third and fourth operands are of the same size, but different signs.

I16vec4 R = select_neq(Is16vec4, Is16vec4, Is16vec4, Iu16vec4);

Conditional Select for Equality

R0 := (A0 == B0) ? C0 : D0;
R1 := (A1 == B1) ? C1 : D1;
R2 := (A2 == B2) ? C2 : D2;
R3 := (A3 == B3) ? C3 : D3;

Conditional Select for Inequality

R0 := (A0 != B0) ? C0 : D0;
R1 := (A1 != B1) ? C1 : D1;
R2 := (A2 != B2) ? C2 : D2;
R3 := (A3 != B3) ? C3 : D3;
## Conditional Select Symbols and Corresponding Intrinsics

<table>
<thead>
<tr>
<th>Conditional Select For:</th>
<th>Operators</th>
<th>Syntax</th>
<th>Corresponding Intrinsic</th>
<th>Additional Intrinsic (Applies to All)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>select_eq</td>
<td>R =</td>
<td>_mm_cmpeq_pi32 _mm_and_si64</td>
<td>_mm_and_si64 _mm_or_si64 _mm_andnot_si64</td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_eq(A, B, C, D)</td>
<td>_mm_cmpeq_pi16</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_eq(A, B, C, D)</td>
<td>_mm_cmpeq_pi8</td>
<td></td>
</tr>
<tr>
<td>Inequality</td>
<td>select_neq</td>
<td>R =</td>
<td>_mm_cmpeq_pi32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_neq(A, B, C, D)</td>
<td>_mm_cmpeq_pi16</td>
<td></td>
</tr>
<tr>
<td>Greater Than</td>
<td>select_gt</td>
<td>R =</td>
<td>_mm_cmpl_gt_pi32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_gt(A, B, C, D)</td>
<td>_mm_cmpl_gt_pi16</td>
<td></td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>select_ge</td>
<td>R =</td>
<td>_mm_cmpl_ge_pi32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_ge(A, B, C, D)</td>
<td>_mm_cmpl_ge_pi16</td>
<td></td>
</tr>
<tr>
<td>Less Than</td>
<td>select_lt</td>
<td>R =</td>
<td>_mm_cmplt_pi32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_lt(A, B, C, D)</td>
<td>_mm_cmplt_pi16</td>
<td></td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>select_le</td>
<td>R =</td>
<td>_mm_cmplt_pi32</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>select_le(A, B, C, D)</td>
<td>_mm_cmplt_pi16</td>
<td></td>
</tr>
</tbody>
</table>

All conditional select operands must be of the same size. The return data type is the nearest common ancestor of operands C and D. For conditional select operations using greater-than or less-than operations, the first and second operands must be signed as listed in the table that follows.

### Conditional Select Operator Overloading

<table>
<thead>
<tr>
<th>R</th>
<th>Comparison</th>
<th>A and B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>

1610
The following table shows the mapping of return values from $R_0$ to $R_7$ for any number of elements. The same return value mappings also apply when there are fewer than four return values.

**Conditional Select Operator Return Value Mapping**

<table>
<thead>
<tr>
<th>Return A Value</th>
<th>Available Operators</th>
<th>B Operands</th>
<th>C and D Operands</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C0 : D0;</td>
</tr>
<tr>
<td>R1:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C1 : D1;</td>
</tr>
<tr>
<td>R2:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C2 : D2;</td>
</tr>
<tr>
<td>R3:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C3 : D3;</td>
</tr>
<tr>
<td>R4:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C4 : D4;</td>
</tr>
<tr>
<td>R5:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C5 : D5;</td>
</tr>
<tr>
<td>R6:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C6 : D6;</td>
</tr>
<tr>
<td>R7:= A0</td>
<td>== != &gt; &gt;= &lt; &lt;=</td>
<td>B0</td>
<td>C7 : D7;</td>
</tr>
</tbody>
</table>

**Debug**

The debug operations do not map to any compiler intrinsics for MMX™ instructions. They are provided for debugging programs only. Use of these
operations may result in loss of performance, so you should not use them outside of debugging.

Output

The four 32-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
\[
\text{cout} \ll \text{is32vec4 A;} \\
\text{cout} \ll \text{iu32vec4 A;} \\
\text{cout} \ll \text{hex} \ll \text{iu32vec4 A;} /* print in hex format */ \\
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"
\]
Corresponding Intrinsics: none

The two 32-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
\[
\text{cout} \ll \text{is32vec2 A;} \\
\text{cout} \ll \text{iu32vec2 A;} \\
\text{cout} \ll \text{hex} \ll \text{iu32vec2 A;} /* print in hex format */ \\
"[1]:A1 [0]:A0"
\]
Corresponding Intrinsics: none

The eight 16-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
\[
\text{cout} \ll \text{is16vec8 A;} \\
\text{cout} \ll \text{iu16vec8 A;} \\
\text{cout} \ll \text{hex} \ll \text{iu16vec8 A;} /* print in hex format */ \\
"[7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2 [1]:A1 [0]:A0"
\]
Corresponding Intrinsics: none

The four 16-bit values of A are placed in the output buffer and printed in the following format (default in decimal):
\[
\text{cout} \ll \text{is16vec4 A;} \\
\text{cout} \ll \text{iu16vec4 A;} \\
\text{cout} \ll \text{hex} \ll \text{iu16vec4 A;} /* print in hex format */ \\
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"
\]
Corresponding Intrinsics: none

The sixteen 8-bit values of $A$ are placed in the output buffer and printed in the following format (default is decimal):

```c
cout << Is8vec16 A; cout << Iu8vec16 A; cout << hex << Iu8vec8 A;
/* print in hex format instead of decimal*/
```

Corresponding Intrinsics: none

The eight 8-bit values of $A$ are placed in the output buffer and printed in the following format (default is decimal):

```c
cout << Is8vec8 A; cout << Iu8vec8 A; cout << hex << Iu8vec8 A;
/* print in hex format instead of decimal*/
"[7]:A7 [6]:A6 [5]:A5 [4]:A4 [3]:A3 [2]:A2 [1]:A1 [0]:A0"
```

Corresponding Intrinsics: none

**Element Access Operators**

```c
int R = Is64vec2 A[i];
unsigned int R = Iu64vec2 A[i];
int R = Is32vec4 A[i];
unsigned int R = Iu32vec4 A[i];
int R = Is32vec2 A[i];
unsigned int R = Iu32vec2 A[i];
short R = Is16vec8 A[i];
unsigned short R = Iu16vec8 A[i];
short R = Is16vec4 A[i];
unsigned short R = Iu16vec4 A[i];
signed char R = Is8vec16 A[i];
unsigned char R = Iu8vec16 A[i];
signed char R = Is8vec8 A[i];
```
unsigned char R = Iu8vec8 A[i];

Access and read element i of A. If DEBUG is enabled and the user tries to access an element outside of A, a diagnostic message is printed and the program aborts.

Corresponding Intrinsics: none

**Element Assignment Operators**

Is64vec2 A[i] = int R;
Is32vec4 A[i] = int R;
Iu32vec4 A[i] = unsigned int R;
Is32vec2 A[i] = int R;
Iu32vec2 A[i] = unsigned int R;
Is16vec8 A[i] = short R;
Iu16vec8 A[i] = unsigned short R;
Is16vec4 A[i] = short R;
Iu16vec4 A[i] = unsigned short R;
Is8vec16 A[i] = signed char R;
Iu8vec16 A[i] = unsigned char R;
Is8vec8 A[i] = signed char R;
Iu8vec8 A[i] = unsigned char R;

Assign R to element i of A. If DEBUG is enabled and the user tries to assign a value to an element outside of A, a diagnostic message is printed and the program aborts.

Corresponding Intrinsics: none

**Unpack Operators**

Interleave the 64-bit value from the high half of A with the 64-bit value from the high half of B.

I364vec2 unpack_high(I64vec2 A, I64vec2 B);
Is64vec2 unpack_high(Is64vec2 A, Is64vec2 B);
Iu64vec2 unpack_high(Iu64vec2 A, Iu64vec2 B);
R0 = A1;
R1 = B1;

**Corresponding intrinsic:** `_mm_unpackhi_epi64`

Interleave the two 32-bit values from the high half of A with the two 32-bit values from the high half of B.

```cpp
I32vec4 unpack_high(I32vec4 A, I32vec4 B);
Is32vec4 unpack_high(Is32vec4 A, Is32vec4 B);
Iu32vec4 unpack_high(Iu32vec4 A, Iu32vec4 B);
R0 = A1;
R1 = B1;
R2 = A2;
R3 = B2;
```

**Corresponding intrinsic:** `_mm_unpackhi_epi32`

Interleave the 32-bit value from the high half of A with the 32-bit value from the high half of B.

```cpp
I32vec2 unpack_high(I32vec2 A, I32vec2 B);
Is32vec2 unpack_high(Is32vec2 A, Is32vec2 B);
Iu32vec2 unpack_high(Iu32vec2 A, Iu32vec2 B);
R0 = A1;
R1 = B1;
```

**Corresponding intrinsic:** `_mm_unpackhi_pi32`

Interleave the four 16-bit values from the high half of A with the two 16-bit values from the high half of B.

```cpp
I16vec8 unpack_high(I16vec8 A, I16vec8 B);
Is16vec8 unpack_high(Is16vec8 A, Is16vec8 B);
Iu16vec8 unpack_high(Iu16vec8 A, Iu16vec8 B);
R0 = A2;
R1 = B2;
R2 = A3;
R3 = B3;
```

**Corresponding intrinsic:** `_mm_unpackhi_epi16`
Interleave the two 16-bit values from the high half of \( \text{A} \) with the two 16-bit values from the high half of \( \text{B} \).

\[
\text{I16vec4 unpack_high(I16vec4 A, I16vec4 B);} \\
\text{I16vec4 unpack_high(Is16vec4 A, Is16vec4 B);} \\
\text{I16vec4 unpack_high(Iu16vec4 A, Iu16vec4 B);} \\
\text{R0 = A2; R1 = B2;} \\
\text{R2 = A3; R3 = B3;}
\]

**Corresponding intrinsic:** \_mm\_unpackhi\_pi16

Interleave the four 8-bit values from the high half of \( \text{A} \) with the four 8-bit values from the high half of \( \text{B} \).

\[
\text{I8vec8 unpack_high(I8vec8 A, I8vec8 B);} \\
\text{I8vec8 unpack_high(Is8vec8 A, I8vec8 B);} \\
\text{I8vec8 unpack_high(Iu8vec8 A, I8vec8 B);} \\
\text{R0 = A4;}
\text{R1 = B4;} \\
\text{R2 = A5;}
\text{R3 = B5;} \\
\text{R4 = A6;}
\text{R5 = B6;} \\
\text{R6 = A7;}
\text{R7 = B7;}
\]

**Corresponding intrinsic:** \_mm\_unpackhi\_pi8

Interleave the sixteen 8-bit values from the high half of \( \text{A} \) with the four 8-bit values from the high half of \( \text{B} \).

\[
\text{I8vec16 unpack_high(I8vec16 A, I8vec16 B);} \\
\text{I8vec16 unpack_high(Is8vec16 A, I8vec16 B);} \\
\text{I8vec16 unpack_high(Iu8vec16 A, I8vec16 B);} \\
\text{R0 = A8;} \\
\text{R1 = B8;} \\
\text{R2 = A9;} \\
\text{R3 = B9;}
\]
R4 = A10;
R5 = B10;
R6 = A11;
R7 = B11;
R8 = A12;
R8 = B12;
R2 = A13;
R3 = B13;
R4 = A14;
R5 = B14;
R6 = A15;
R7 = B15;

**Corresponding intrinsic: **_mm_unpackhi_epi16

Interleave the 32-bit value from the low half of A with the 32-bit value from the low half of B

R0 = A0;
R1 = B0;

**Corresponding intrinsic: **_mm_unpacklo_epi32

Interleave the 64-bit value from the low half of A with the 64-bit values from the low half of B

I64vec2 unpack_low(I64vec2 A, I64vec2 B);
I64vec2 unpack_low(Is64vec2 A, Is64vec2 B);
I64vec2 unpack_low(Iu64vec2 A, Iu64vec2 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;

**Corresponding intrinsic: **_mm_unpacklo_epi32

Interleave the two 32-bit values from the low half of A with the two 32-bit values from the low half of B

I32vec4 unpack_low(I32vec4 A, I32vec4 B);
Is32vec4 unpack_low(Is32vec4 A, Is32vec4 B);
Iu32vec4 unpack_low(Iu32vec4 A, Iu32vec4 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;

**Corresponding intrinsic:** `_mm_unpacklo_epi32`

Interleave the 32-bit value from the low half of A with the 32-bit value from the low half of B.

I32vec2 unpack_low(I32vec2 A, I32vec2 B);
Is32vec2 unpack_low(Is32vec2 A, Is32vec2 B);
Iu32vec2 unpack_low(Iu32vec2 A, Iu32vec2 B);
R0 = A0;
R1 = B0;

**Corresponding intrinsic:** `_mm_unpacklo_pi32`

Interleave the two 16-bit values from the low half of A with the two 16-bit values from the low half of B.

I16vec8 unpack_low(I16vec8 A, I16vec8 B);
Is16vec8 unpack_low(Is16vec8 A, Is16vec8 B);
Iu16vec8 unpack_low(Iu16vec8 A, Iu16vec8 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;

**Corresponding intrinsic:** `_mm_unpacklo_epi16`

Interleave the two 16-bit values from the low half of A with the two 16-bit values from the low half of B.
I16vec4 unpack_low(I16vec4 A, I16vec4 B);
Is16vec4 unpack_low(Is16vec4 A, Is16vec4 B);
Iu16vec4 unpack_low(Iu16vec4 A, Iu16vec4 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;

**Corresponding intrinsic:** _mm_unpacklo_pi16

Interleave the four 8-bit values from the high low of A with the four 8-bit values from the low half of B.

I8vec16 unpack_low(I8vec16 A, I8vec16 B);
Is8vec16 unpack_low(Is8vec16 A, Is8vec16 B);
Iu8vec16 unpack_low(Iu8vec16 A, Iu8vec16 B);
R0 = A0;
R1 = B0;
R2 = A1;
R3 = B1;
R4 = A2;
R5 = B2;
R6 = A3;
R7 = B3;
R8 = A4;
R9 = B4;
R10 = A5;
R11 = B5;
R12 = A6;
R13 = B6;
R14 = A7;
R15 = B7;

**Corresponding intrinsic:** _mm_unpacklo_epi8
Interleave the four 8-bit values from the high low of $A$ with the four 8-bit values from the low half of $B$.

\[
\begin{align*}
&I8vec8 \text{ unpack\_low}(I8vec8 A, I8vec8 B); \\
&Is8vec8 \text{ unpack\_low}(Is8vec8 A, Is8vec8 B); \\
&Iu8vec8 \text{ unpack\_low}(Iu8vec8 A, Iu8vec8 B);
\end{align*}
\]

\[
\begin{align*}
R0 &= A0; \\
R1 &= B0; \\
R2 &= A1; \\
R3 &= B1; \\
R4 &= A2; \\
R5 &= B2; \\
R6 &= A3; \\
R7 &= B3;
\end{align*}
\]

**Corresponding intrinsic:** \_mm\_unpacklo\_pi8

**Pack Operator**

Pack the eight 32-bit values found in $A$ and $B$ into eight 16-bit values with signed saturation.

\[
\begin{align*}
&Is16vec8 \text{ pack\_sat}(Is32vec2 A, Is32vec2 B);
\end{align*}
\]

**Corresponding intrinsic:** \_mm\_packs\_epi32

Pack the four 32-bit values found in $A$ and $B$ into eight 16-bit values with signed saturation.

\[
\begin{align*}
&Is16vec4 \text{ pack\_sat}(Is32vec2 A, Is32vec2 B);
\end{align*}
\]

**Corresponding intrinsic:** \_mm\_packs\_pi32

Pack the sixteen 16-bit values found in $A$ and $B$ into sixteen 8-bit values with signed saturation.

\[
\begin{align*}
&Is8vec16 \text{ pack\_sat}(Is16vec4 A, Is16vec4 B);
\end{align*}
\]

**Corresponding intrinsic:** \_mm\_packs\_epi16

Pack the eight 16-bit values found in $A$ and $B$ into eight 8-bit values with signed saturation.
Is8vec8 pack_sat(Is16vec4 A, Is16vec4 B);
 Corresponding intrinsic: _mm_packs_pi16
Pack the sixteen 16-bit values found in A and B into sixteen 8-bit values with unsigned saturation.
Iu8vec16 packu_sat(Is16vec4 A, Is16vec4 B);
 Corresponding intrinsic: _mm_packus_epi16
Pack the eight 16-bit values found in A and B into eight 8-bit values with unsigned saturation.
Iu8vec8 packu_sat(Is16vec4 A, Is16vec4 B);
 Corresponding intrinsic: _mm_packs_pu16

Clear MMX™ State Operator
Empty the MMX™ registers and clear the MMX state. Read the guidelines for using the EMMS instruction intrinsic.
void empty(void);
 Corresponding intrinsic: _mm_empty

Integer Functions for Streaming SIMD Extensions

⚠️ Note
You must include fvec.h header file for the following functionality.
Compute the element-wise maximum of the respective signed integer words in A and B.
Is16vec4 simd_max(Is16vec4 A, Is16vec4 B);
 Corresponding intrinsic: _mm_max_pi16
Compute the element-wise minimum of the respective signed integer words in A and B.
Is16vec4 simd_min(Is16vec4 A, Is16vec4 B);
 Corresponding intrinsic: _mm_min_pi16
Compute the element-wise maximum of the respective unsigned bytes in A and B.
Iu8vec8 simd_max(Iu8vec8 A, Iu8vec8 B);
Corresponding intrinsic: _mm_max_pu8
Compute the element-wise minimum of the respective unsigned bytes in A and B.
Iu8vec8 simd_min(Iu8vec8 A, Iu8vec8 B);
Corresponding intrinsic: _mm_min_pu8
Create an 8-bit mask from the most significant bits of the bytes in A.
int move_mask(I8vec8 A);
Corresponding intrinsic: _mm_movemask_p18
Conditionally store byte elements of A to address p. The high bit of each byte in the selector B determines whether the corresponding byte in A will be stored.
void mask_move(I8vec8 A, I8vec8 B, signed char *p);
Corresponding intrinsic: _mm_maskmove_si64
Store the data in A to the address p without polluting the caches. A can be any Ivec type.
void store_nta(__m64 *p, M64 A);
Corresponding intrinsic: _mm_stream_pi
Compute the element-wise average of the respective unsigned 8-bit integers in A and B.
Iu8vec8 simd_avg(Iu8vec8 A, Iu8vec8 B);
Corresponding intrinsic: _mm_avg_pu8
Compute the element-wise average of the respective unsigned 16-bit integers in A and B.
Iu16vec4 simd_avg(Iu16vec4 A, Iu16vec4 B);
Corresponding intrinsic: _mm_avg_pu16

Conversions between Fvec and Ivec

Convert the lower double-precision floating-point value of A to a 32-bit integer with truncation.
int F64vec2ToInt(F64vec42 A);
\[ r := \text{(int)} A0; \]
Convert the four floating-point values of \( A \) to two the two least significant double-precision floating-point values.

\[
\text{F64vec2 F32vec4ToF64vec2(F32vec4 A);} \\
r0 := \text{(double)}A0; \\
r1 := \text{(double)}A1;
\]

Convert the two double-precision floating-point values of \( A \) to two single-precision floating-point values.

\[
\text{F32vec4 F64vec2ToF32vec4(F64vec2 A);} \\
r0 := \text{(float)}A0; \\
r1 := \text{(float)}A1;
\]

Convert the signed \text{int} \text{in} \ B \text{to a double-precision floating-point value and pass the upper double-precision value from} \ A \text{through to the result.}

\[
\text{F64vec2 InttoF64vec2(F64vec2 A, int B);} \\
r0 := \text{(double)}B; \\
r1 := A1;
\]

Convert the lower floating-point value of \( A \) to a 32-bit integer with truncation.

\[
\text{int F32vec4ToInt(F32vec4 A);} \\
r := \text{(int)}A0;
\]

Convert the two lower floating-point values of \( A \) to two 32-bit integer with truncation, returning the integers in packed form.

\[
\text{Is32vec2 F32vec4ToIs32vec2(F32vec4 A);} \\
r0 := \text{(int)}A0; \\
r1 := \text{(int)}A1;
\]

Convert the 32-bit integer value \( B \) to a floating-point value; the upper three floating-point values are passed through from \( A \).

\[
\text{F32vec4 IntToF32vec4(F32vec4 A, int B);} \\
r0 := \text{(float)}B; \\
r1 := A1; \\
r2 := A2; \\
r3 := A3;
\]
Convert the two 32-bit integer values in packed form in \( B \) to two floating-point values; the upper two floating-point values are passed through from \( A \).

\[
\text{F32vec4 } \text{Is32vec2ToF32vec4}(\text{F32vec4 } A, \text{ Is32vec2 } B);
\]

\[
r0 := (\text{float})B0;
\]

\[
r1 := (\text{float})B1;
\]

\[
r2 := A2;
\]

\[
r3 := A3;
\]

**Floating-point Vector Classes**

**Overview: Floating-point Vector Classes**

The floating-point vector classes, \( \text{F64vec2}, \text{F32vec4}, \) and \( \text{F32vec1} \), provide an interface to SIMD operations. The class specifications are as follows:

\[
\text{F64vec2 } A(\text{double } x, \text{ double } y);
\]

\[
\text{F32vec4 } A(\text{float } z, \text{ float } y, \text{ float } x, \text{ float } w);
\]

\[
\text{F32vec1 } B(\text{float } w);
\]

The packed floating-point input values are represented with the right-most value lowest as shown in the following table.

**Single-Precision Floating-point Elements**

<table>
<thead>
<tr>
<th>High Value</th>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operands</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operations</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Return Value</td>
<td>127</td>
<td>83</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\( \text{F32vec4} \) returns four packed **single-precision floating point** values (\( R0, R1, R2, \) and \( R3 \)).

\( \text{F32vec2} \) returns one **single-precision floating point** value (\( R0 \)).

**Fvec Notation Conventions**
This reference uses the following conventions for syntax and return values.

**Fvec Classes Syntax Notation**

Fvec classes use the syntax conventions shown the following examples:

**Example 1:**

\[
\text{Fvec\_Class} \ R = \text{Fvec\_Class} \ A \ \text{operator} \ \text{Ivec\_Class} \ B;
\]

**Example 2:**

\[
\text{Fvec\_Class} \ R = \text{operator}(\text{Fvec\_Class} \ A, \text{Fvec\_Class} \ B);
\]

**Example 3:**

\[
\text{Fvec\_Class} \ R = \text{operator} = \text{Fvec\_Class} \ A;
\]

where

- \([\text{operator}]\) is an operator (for example, \&, \|, or ^)
- \([\text{Fvec\_Class}]\) is any Fvec class (F64vec2, F32vec4, or F32vec1)
- \(R, A, B\) are declared Fvec variables of the type indicated.

**Return Value Notation**

Because the Fvec classes have packed elements, the return values typically follow the conventions presented in the Return Value Convention Notation Mappings table. F32vec4 returns four single-precision, floating-point values (R0, R1, R2, and R3); F64vec2 returns two double-precision, floating-point values, and F32vec1 returns the lowest single-precision floating-point value (R0).

**Return Value Convention Notation Mappings**

<table>
<thead>
<tr>
<th>Example 1:</th>
<th>Example 2:</th>
<th>Example 3:</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 := A0 &amp; R0 := A0</td>
<td>R0 &amp; A0; x x x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B0; andnot B0;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1 := A1 &amp; R1 := A1</td>
<td>R1 &amp; A1; x x</td>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td>B1; andnot B1;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2 := A2 &amp; R2 := A2</td>
<td>R2 &amp; A2; x</td>
<td></td>
<td></td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>B2; andnot B2;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Data Alignment

Memory operations using the Streaming SIMD Extensions should be performed on 16-byte-aligned data whenever possible. 

F32vec4 and F64vec2 object variables are properly aligned by default. Note that floating point arrays are not automatically aligned. To get 16-byte alignment, you can use the alignment __declspec:

```
__declspec( align(16) ) float A[4];
```

Conversions

All Fvec object variables can be implicitly converted to __m128 data types. For example, the results of computations performed on F32vec4 or F32vec1 object variables can be assigned to __m128 data types.

```
__m128d mm = A & B; /* where A,B are F64vec2 object variables */
__m128 mm = A & B; /* where A,B are F32vec4 object variables */
__m128 mm = A & B; /* where A,B are F32vec1 object variables */
```

Constructors and Initialization

The following table shows how to create and initialize F32vec objects with the Fvec classes.

```
Example Intrinsic Returns
R3 := A3 & B3 R3 := A3 R3 &= A3; x N/A N/A
    andhot B3;
```

```
Constructors and Initialization for Fvec Classes
```

```
Example Intrinsic Returns
R3 := A3 & B3 R3 := A3 R3 &= A3; x N/A N/A
    andhot B3;
```

```
Data Alignment

Memory operations using the Streaming SIMD Extensions should be performed on 16-byte-aligned data whenever possible. 

F32vec4 and F64vec2 object variables are properly aligned by default. Note that floating point arrays are not automatically aligned. To get 16-byte alignment, you can use the alignment __declspec:

```
__declspec( align(16) ) float A[4];
```

Conversions

All Fvec object variables can be implicitly converted to __m128 data types. For example, the results of computations performed on F32vec4 or F32vec1 object variables can be assigned to __m128 data types.

```
__m128d mm = A & B; /* where A,B are F64vec2 object variables */
__m128 mm = A & B; /* where A,B are F32vec4 object variables */
__m128 mm = A & B; /* where A,B are F32vec1 object variables */
```

Constructors and Initialization

The following table shows how to create and initialize F32vec objects with the Fvec classes.

```
Example Intrinsic Returns
R3 := A3 & B3 R3 := A3 R3 &= A3; x N/A N/A
    andhot B3;
```

```
Constructors and Initialization for Fvec Classes
```

```
Example Intrinsic Returns
R3 := A3 & B3 R3 := A3 R3 &= A3; x N/A N/A
    andhot B3;
```
<table>
<thead>
<tr>
<th>Constructor Declaration</th>
</tr>
</thead>
<tbody>
<tr>
<td>F64vec2 A;</td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td>F32vec4 B;</td>
</tr>
<tr>
<td>F32vec1 C;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>_m128 Object Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>F64vec2 A(__m128d mm);</td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td>N/A</td>
</tr>
<tr>
<td>F32vec4 B(__m128 mm);</td>
</tr>
<tr>
<td>F32vec1 C(__m128 mm);</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Double Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>/* Initializes two _mm_set_pd doubles. */</td>
</tr>
<tr>
<td>A0 := d0;</td>
</tr>
<tr>
<td>A1 := d1;</td>
</tr>
<tr>
<td>F64vec2 A(double d0, double d1);</td>
</tr>
<tr>
<td>F64vec2 A = F64vec2(double d0, double d1);</td>
</tr>
<tr>
<td>F64vec2 A(double _mm_set1_pd d0);</td>
</tr>
<tr>
<td>A0 := d0;</td>
</tr>
<tr>
<td>A1 := d0;</td>
</tr>
<tr>
<td>/* Initializes both return values with the same double</td>
</tr>
<tr>
<td>precision value */.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Float Initialization</th>
</tr>
</thead>
<tbody>
<tr>
<td>F32vec4 A(float f3, _mm_set_ps float f2, float f1, float f2;</td>
</tr>
</tbody>
</table>
Example

Intrinsic

Returns

**Constructor Declaration**

```c
f0);
F32vec4 A =
F32vec4(float f3,
float f2,
float f1, float
f0);
F32vec4 A(float _mm_set1_ps
f0);
/* Initializes all
return values
with the same
floating point
value. */
F32vec4 A(double _mm_set1_ps(d)
d0);
/* Initialize all
return values with
the same double-
precision value. */
F32vec1 A(double _mm_set_ss(d)
d0);
/* Initializes the
lowest value of A
with d0 and the
other values with
0. */
F32vec1 B(float _mm_set_ss
B0 := f0;
```
Example

Intrinsic

Returns

Constructor Declaration

f0);
/* Initializes the
lowest value of B
with f0 and the
other values with
0.*/

F32vec1 B(int I);  _mm_cvtsi32_ss
/* Initializes the
lowest value of B
with f0, other
values are
undefined.*/

Arithmetic Operators

The following table lists the arithmetic operators of the Fvec classes and generic syntax. The operators have been divided into standard and advanced operations, which are described in more detail later in this section.

Fvec Arithmetic Operators

<table>
<thead>
<tr>
<th>Category</th>
<th>Operation</th>
<th>Operators</th>
<th>Generic Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard</td>
<td>Addition</td>
<td>+</td>
<td>R = A + B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>+=</td>
<td>R += A;</td>
</tr>
<tr>
<td></td>
<td>Subtraction</td>
<td>-</td>
<td>R = A - B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-=</td>
<td>R -= A;</td>
</tr>
<tr>
<td></td>
<td>Multiplication</td>
<td>*</td>
<td>R = A * B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*=</td>
<td>R *= A;</td>
</tr>
</tbody>
</table>
### Standard Arithmetic Operator Usage

The following two tables show the return values for each class of the standard arithmetic operators, which use the syntax styles described earlier in the Return Value Notation section.

#### Standard Arithmetic Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>A</th>
<th>Operators</th>
<th>B</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 := A0</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B0</td>
<td></td>
</tr>
<tr>
<td>R1 := A1</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B1</td>
<td>N/A</td>
</tr>
<tr>
<td>R2 := A2</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B2</td>
<td>N/A</td>
</tr>
<tr>
<td>R3 := A3</td>
<td>+</td>
<td>-</td>
<td>*</td>
<td>/</td>
<td>B3</td>
<td>N/A</td>
</tr>
</tbody>
</table>

#### Arithmetic with Assignment Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>Operators</th>
<th>A</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0 := +=</td>
<td>-=</td>
<td>*=</td>
<td>/=</td>
<td>A0</td>
<td></td>
</tr>
<tr>
<td>R1 := +=</td>
<td>-=</td>
<td>*=</td>
<td>/=</td>
<td>A1</td>
<td>N/A</td>
</tr>
<tr>
<td>R2 := +=</td>
<td>-=</td>
<td>*=</td>
<td>/=</td>
<td>A2</td>
<td>N/A</td>
</tr>
</tbody>
</table>
This table lists standard arithmetic operator syntax and intrinsics.

### Standard Arithmetic Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
<th>Example Syntax</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Addition</strong></td>
<td>4 floats</td>
<td>F32vec4 R = F32vec4 A + F32vec4 B; F32vec4 R += F32vec4 A;</td>
<td>_mm_add_ps</td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = F64vec2 A + F64vec2 B; F64vec2 R += F64vec2 A;</td>
<td>_mm_add_pd</td>
</tr>
<tr>
<td></td>
<td>1 float</td>
<td>F32vec1 R = F32vec1 A + F32vec1 B; F32vec1 R += F32vec1 A;</td>
<td>_mm_add_ss</td>
</tr>
<tr>
<td><strong>Subtraction</strong></td>
<td>4 floats</td>
<td>F32vec4 R = F32vec4 A - F32vec4 B; F32vec4 R -= F32vec4 A;</td>
<td>_mm_sub_ps</td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = F64vec2 A +</td>
<td>_mm_sub_pd</td>
</tr>
<tr>
<td>Operation</td>
<td>Returns</td>
<td>Example Syntax Usage</td>
<td>Intrinsic</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------</td>
<td>----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec2 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R -=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A;</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td></td>
<td>F32vec1 R = _mm_sub_ss</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 A -</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 R -=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 A;</td>
<td></td>
</tr>
<tr>
<td>Multiplication</td>
<td>4 floats</td>
<td>F32vec4 R = _mm_mul_ps</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A *</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 R *=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A;</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td></td>
<td>F64vec2 R = _mm_mul_pd</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A *</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F364vec2 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R *=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A;</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td></td>
<td>F32vec1 R = _mm_mul_ss</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 A *</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 R *=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 A;</td>
<td></td>
</tr>
<tr>
<td>Division</td>
<td>4 floats</td>
<td>F32vec4 R = _mm_div_ps</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A /</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 R /=</td>
<td></td>
</tr>
</tbody>
</table>
## Advanced Arithmetic Operator Usage

The following table shows the return values classes of the advanced arithmetic operators, which use the syntax styles described earlier in the Return Value Notation section.

### Advanced Arithmetic Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>Operators</th>
<th>A</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Advanced Arithmetic Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>Operators</th>
<th>A</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R1:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>=</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R3:</td>
<td>sqrt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
This table shows examples for advanced arithmetic operators.

### Advanced Arithmetic Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Square Root</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = sqrt(F32vec4 A);</td>
<td>_mm_sqrt_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = sqrt(F64vec2 A);</td>
<td>_mm_sqrt_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = sqrt(F32vec1 A);</td>
<td>_mm_sqrt_ss</td>
</tr>
</tbody>
</table>

<p>| <strong>Reciprocal</strong> |                   |                    |
| 4 floats       | F32vec4 R = rcp(F32vec4 A); | _mm_rcp_ps         |
| 2 doubles      | F64vec2 R = rcp(F64vec2 A); | _mm_rcp_pd         |
| 1 float        | F32vec1 R = _mm_rcp_ss  |                    |</p>
<table>
<thead>
<tr>
<th></th>
<th>Function Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reciprocal</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>rcp(F32vec1 A);</code></td>
</tr>
<tr>
<td><strong>Reciprocal Square Root</strong></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td><code>F32vec4 R = _mm_rsqrt_ps rsqrt(F32vec4 A);</code></td>
</tr>
<tr>
<td></td>
<td><code>F64vec2 R = _mm_rsqrt_pd rsqrt(F64vec2 A);</code></td>
</tr>
<tr>
<td></td>
<td><code>F32vec1 R = _mm_rsqrt_ss rsqrt(F32vec1 A);</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td><code>F64vec2 R = _mm_rsqrt_pd rsqrt(F64vec2 A);</code></td>
</tr>
<tr>
<td></td>
<td><code>F32vec1 R = _mm_rsqrt_ss rsqrt(F32vec1 A);</code></td>
</tr>
<tr>
<td><strong>Reciprocal Newton Raphson</strong></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td><code>F32vec4 R = _mm_sub_ps _mm_add_ps _mm_mul_ps _mm_rcp_ps</code> rcp_nr(F32vec4 A);`</td>
</tr>
<tr>
<td></td>
<td><code>F64vec2 R = _mm_sub_pd _mm_add_pd _mm_mul_pd _mm_rcp_pd</code> rcp_nr(F64vec2 A);`</td>
</tr>
<tr>
<td></td>
<td><code>F32vec1 R = _mm_sub_ss _mm_add_ss _mm_mul_ss _mm_rcp_ss</code> rcp_nr(F32vec1 A);`</td>
</tr>
<tr>
<td>2 doubles</td>
<td><code>F64vec2 R = _mm_sub_pd _mm_add_pd _mm_mul_pd _mm_rcp_pd</code> rcp_nr(F64vec2 A);`</td>
</tr>
<tr>
<td><strong>Reciprocal Square Root Newton Raphson</strong></td>
<td></td>
</tr>
<tr>
<td>4 float</td>
<td><code>F32vec4 R = _mm_sub_pd rsqrt_nr(F32vec4 A); _mm_mul_pd _mm_rsqrt_ps</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td><code>F64vec2 R = _mm_sub_pd rsqrt_nr(F64vec2 A); _mm_mul_pd _mm_rsqrt_ps</code></td>
</tr>
</tbody>
</table>
Reciprocal Square Root Newton Raphson

\[
\text{1 float} \quad \text{F32vec1 } R = \text{rsqrt_nr(F32vec1 } A) \text{; } \_\text{mm_rsqrt_pd}
\]

Horizontal Add

\[
\text{1 float} \quad \text{float } f = \text{add_horizontal(F32vec4 } \_\text{mm_shuffle_ss } A) \text{; } \_\text{mm_add_ss}
\]

\[
\text{1 double} \quad \text{double } d = \text{add_horizontal(F64vec2 } \_\text{mm_shuffle_sd } A) \text{; } \_\text{mm_add_sd}
\]

Minimum and Maximum Operators

Compute the minimums of the two double precision floating-point values of \( A \) and \( B \).

\[
\text{F64vec2 } R = \text{simd_min(F64vec2 } A, \text{F64vec2 } B) \\
R0 := \min(A0, B0); \\
R1 := \min(A1, B1);
\]

Corresponding intrinsic: \_\text{mm_min_pd}

Compute the minimums of the four single precision floating-point values of \( A \) and \( B \).

\[
\text{F32vec4 } R = \text{simd_min(F32vec4 } A, \text{F32vec4 } B) \\
R0 := \min(A0, B0); \\
R1 := \min(A1, B1); \\
R2 := \min(A2, B2); \\
R3 := \min(A3, B3);
\]

Corresponding intrinsic: \_\text{mm_min_ps}
Compute the minimum of the lowest single precision floating-point values of A and B.

\[
\begin{align*}
F32vec1 \ R &= \ simd\_min(F32vec1 \ A, \ F32vec1 \ B) \\
R0 &= \ \min(A0, B0);
\end{align*}
\]

**Corresponding intrinsic:** \texttt{__mm\_min\_ss}

Compute the maximums of the two double precision floating-point values of A and B.

\[
\begin{align*}
F64vec2 \ simd\_max(F64vec2 \ A, \ F64vec2 \ B) \\
R0 &= \ \max(A0, B0); \\
R1 &= \ \max(A1, B1);
\end{align*}
\]

**Corresponding intrinsic:** \texttt{__mm\_max\_pd}

Compute the maximums of the four single precision floating-point values of A and B.

\[
\begin{align*}
F32vec4 \ R &= \ simd\_man(F32vec4 \ A, \ F32vec4 \ B) \\
R0 &= \ \max(A0, B0); \\
R1 &= \ \max(A1, B1); \\
R2 &= \ \max(A2, B2); \\
R3 &= \ \max(A3, B3);
\end{align*}
\]

**Corresponding intrinsic:** \texttt{__mm\_max\_ps}

Compute the maximum of the lowest single precision floating-point values of A and B.

\[
\begin{align*}
F32vec1 \ simd\_max(F32vec1 \ A, \ F32vec1 \ B) \\
R0 &= \ \max(A0, B0);
\end{align*}
\]

**Corresponding intrinsic:** \texttt{__mm\_max\_ss}

**Logical Operations**

The following table lists the logical operators of the \texttt{Fvec} classes and generic syntax. The logical operators for \texttt{F32vec1} classes use only the lower 32 bits.

**Fvec Logical Operators Return Value Mapping**

<table>
<thead>
<tr>
<th>Bitwise Operation</th>
<th>Operators</th>
<th>Generic Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1637
### Bitwise Operation

<table>
<thead>
<tr>
<th>Operation</th>
<th>Operators</th>
<th>Generic Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>&amp;</td>
<td>R = A &amp; B;</td>
</tr>
<tr>
<td></td>
<td>&amp;</td>
<td>R &amp;= A;</td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>XOR</strong></td>
<td>^</td>
<td>R = A ^ B;</td>
</tr>
<tr>
<td></td>
<td>^=</td>
<td>R ^= A;</td>
</tr>
<tr>
<td><strong>andnot</strong></td>
<td>andnot</td>
<td>R = andnot(A);</td>
</tr>
</tbody>
</table>

The following table lists standard logical operators syntax and corresponding intrinsics. Note that there is no corresponding scalar intrinsic for the `F32vec1` classes, which accesses the lower 32 bits of the packed vector intrinsics.

### Logical Operations for Fvec Classes

<table>
<thead>
<tr>
<th>Operation</th>
<th>Returns</th>
<th>Example Syntax</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AND</strong></td>
<td>4 floats</td>
<td>F32vec4 &amp; = _mm_and_ps</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A &amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 &amp; &amp;=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A;</td>
<td></td>
</tr>
<tr>
<td><strong>OR</strong></td>
<td>2 doubles</td>
<td>F64vec2 R = _mm_and_pd</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A &amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec2 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R &amp;=</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A;</td>
<td></td>
</tr>
<tr>
<td><strong>XOR</strong></td>
<td>1 float</td>
<td>F32vec1 R = _mm_and_ps</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 A &amp;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 B;</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>Returns</td>
<td>Example Syntax Usage</td>
<td>Intrinsic</td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>----------------------</td>
<td>-----------</td>
</tr>
<tr>
<td>OR</td>
<td>4 floats</td>
<td>F32vec1 R &amp;= F32vec1 A; F32vec4 R = _mm_or_ps F32vec4 A</td>
<td>_mm_or_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 R</td>
<td>= F32vec4 A;</td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = _mm_or_pd F64vec2 A</td>
<td>_mm_or_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R</td>
<td>= F64vec2 A;</td>
</tr>
<tr>
<td>XOR</td>
<td>1 float</td>
<td>F32vec1 R = _mm_or_ps F32vec1 A</td>
<td>_mm_or_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec1 R</td>
<td>= F32vec1 A;</td>
</tr>
<tr>
<td></td>
<td>4 floats</td>
<td>F32vec4 R = _mm_xor_ps F32vec4 A ^ F32vec4 B;</td>
<td>_mm_xor_ps</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F32vec4 R ^= F32vec4 A;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 doubles</td>
<td>F64vec2 R = _mm_xor_pd F64vec2 A ^ F364vec2 B;</td>
<td>_mm_xor_pd</td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 B;</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>F64vec2 R ^= F64vec2 A;</td>
<td></td>
</tr>
<tr>
<td>Operation</td>
<td>Returns</td>
<td>Example Syntax</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>---------</td>
<td>----------------</td>
<td></td>
</tr>
</tbody>
</table>
| 1 float   |         | F32vec1 R = _mm_xor_ps  
|           |         | F32vec1 A ^  
|           |         | F32vec1 B;  
|           |         | F32vec1 R ^=  
|           |         | F32vec1 A;  |
| ANDNOT    | 2 doubles | F64vec2 R = _mm_andnot_pd  
|           |         | andnot(F64vec2  
|           |         | A,  
|           |         | F64vec2 B); |

**Compare Operators**

The operators described in this section compare the single precision floating-point values of A and B. Comparison between objects of any Fvec class return the same class being compared.

The following table lists the compare operators for the Fvec classes.

**Compare Operators and Corresponding Intrinsics**

<table>
<thead>
<tr>
<th>Compare For:</th>
<th>Operators</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>cmpeq</td>
<td>R = cmpeq(A, B)</td>
</tr>
<tr>
<td>Inequality</td>
<td>cmpneq</td>
<td>R = cmpneq(A, B)</td>
</tr>
<tr>
<td>Greater Than</td>
<td>cmpgt</td>
<td>R = cmpgt(A, B)</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>cmpge</td>
<td>R = cmpge(A, B)</td>
</tr>
<tr>
<td>Not Greater Than</td>
<td>cmpngt</td>
<td>R = cmpngt(A, B)</td>
</tr>
<tr>
<td>Not Greater Than or Equal To</td>
<td>cmpnge</td>
<td>R = cmpnge(A, B)</td>
</tr>
</tbody>
</table>
Compare For:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmplt</td>
<td>R = cmplt(A, B)</td>
</tr>
<tr>
<td>cmple</td>
<td>R = cmple(A, B)</td>
</tr>
<tr>
<td>cmpnlt</td>
<td>R = cmpnlt(A, B)</td>
</tr>
<tr>
<td>cmpnle</td>
<td>R = cmpnle(A, B)</td>
</tr>
</tbody>
</table>

Compare Operators

The mask is set to **0xffffffff** for each floating-point value where the comparison is true and **0x00000000** where the comparison is false. The following table shows the return values for each class of the compare operators, which use the syntax described earlier in the [Return Value Notation](#) section.

Compare Operator Return Value Mapping

<table>
<thead>
<tr>
<th>R</th>
<th>A0 For Any Operators</th>
<th>B If True</th>
<th>If False</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R0:=</td>
<td>(A1 cmp[eq</td>
<td>lt</td>
<td>le</td>
<td>gt B1) 0xffffffff 0x00000000 X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>!(A1</td>
<td>ge] B1) cmp[ne</td>
<td>nlt</td>
<td>nle</td>
<td>ngt</td>
<td>nge]</td>
<td></td>
</tr>
</tbody>
</table>

| R1:= | (A1 cmp[eq | lt | le | gt B2) 0xffffffff 0x00000000 X | X | N/A |
| !(A1 | ge) B2) cmp[ne | nlt | nle | ngt | nge] |

<p>| R2:= | (A1 cmp[eq | lt | le | gt B3) 0xffffffff 0x00000000 X | N/A | N/A |
| !(A1 | ge) B3) cmp[ne | nlt | nle | ngt | nge] |</p>
<table>
<thead>
<tr>
<th>R</th>
<th>A0</th>
<th>For Any Operators</th>
<th>B</th>
<th>If True</th>
<th>If False</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>R3:= A3</td>
<td>cmp[eq</td>
<td>lt</td>
<td>le</td>
<td>gt</td>
<td>ge]</td>
<td>B3) 0xffffffff 0x00000000 X</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Compare Operations for Fvec Classes**

The following table shows examples for arithmetic operators and intrinsics.

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare for Equality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = cmpeq(F32vec4 A);</td>
<td>_mm_cmpeq_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = cmpeq(F64vec2 A);</td>
<td>_mm_cmpeq_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = cmpeq(F32vec1 A);</td>
<td>_mm_cmpeq_ss</td>
</tr>
<tr>
<td><strong>Compare for Inequality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = cmpneq(F32vec4 A);</td>
<td>_mm_cmpneq_ps</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 R = cmpneq(F64vec2 A);</td>
<td>_mm_cmpneq_pd</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 R = cmpneq(F32vec1 A);</td>
<td>_mm_cmpneq_ss</td>
</tr>
<tr>
<td><strong>Compare for Less Than</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 R = cmplt(F32vec4 A);</td>
<td>_mm_cmplt_ps</td>
</tr>
</tbody>
</table>
| Compare for Less Than | 2 doubles | F64vec2 R = _mm_cmplt_pd  
cmplt(F64vec2 A); |
|-----------------------|-----------|--------------------------|
|                       | 1 float   | F32vec1 R = _mm_cmplt_ss  
cmplt(F32vec1 A); |
| Compare for Less Than or Equal | 4 floats | F32vec4 R = _mm_cmple_ps  
cmple(F32vec4 A); |
|                       | 2 doubles | F64vec2 R = _mm_cmple_pd  
cmple(F64vec2 A); |
|                       | 1 float   | F32vec1 R = _mm_cmple_pd  
cmple(F32vec1 A); |
| Compare for Greater Than | 4 floats | F32vec4 R = _mm_cmpgt_ps  
cmpgt(F32vec4 A); |
|                       | 2 doubles | F64vec2 R = _mm_cmpgt_pd  
cmpgt(F64vec2 A); |
|                       | 1 float   | F32vec1 R = _mm_cmpgt_ss  
cmpgt(F32vec1 A); |
| Compare for Greater Than or Equal To | 4 floats | F32vec4 R = _mm_cmpge_ps  
cmpge(F32vec4 A); |
|                       | 2 doubles | F64vec2 R = _mm_cmpge_pd  
cmpge(F64vec2 A); |
|                       | 1 float   | F32vec1 R = _mm_cmpge_ss  
cmpge(F32vec1 A); |
| Compare for Not Less Than | 4 floats | F32vec4 R = _mm_cmpne_ps  
cmpne(F32vec4 A); |
|                       | 2 doubles | F64vec2 R = _mm_cmpne_pd  
cmpne(F64vec2 A); |
|                       | 1 float   | F32vec1 R = _mm_cmpne_ss  
cmpne(F32vec1 A); |
### Compare for Not Less Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4</td>
<td><code>_mm_cmpnlt_ps</code> cmpnlt(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td><code>_mm_cmpnlt_pd</code> cmpnlt(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td><code>_mm_cmpnlt_ss</code> cmpnlt(F32vec1 A);</td>
</tr>
</tbody>
</table>

### Compare for Not Less Than or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4</td>
<td><code>_mm_cmpnle_ps</code> cmpnle(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td><code>_mm_cmpnle_pd</code> cmpnle(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td><code>_mm_cmpnle_ss</code> cmpnle(F32vec1 A);</td>
</tr>
</tbody>
</table>

### Compare for Not Greater Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4</td>
<td><code>_mm_cmpngt_ps</code> cmpngt(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td><code>_mm_cmpngt_pd</code> cmpngt(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td><code>_mm_cmpngt_ss</code> cmpngt(F32vec1 A);</td>
</tr>
</tbody>
</table>

### Compare for Not Greater Than or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Result Type</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec4</td>
<td><code>_mm_cmpnge_ps</code> cmpnge(F32vec4 A);</td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td><code>_mm_cmpnge_pd</code> cmpnge(F64vec2 A);</td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td><code>_mm_cmpnge_ss</code> cmpnge(F32vec1 A);</td>
</tr>
</tbody>
</table>

1644
Compare for Not Greater Than or Equal

\[ \text{cmpnge(F32vec1 A)}; \]

Conditional Select Operators for Fvec Classes

Each conditional function compares single-precision floating-point values of A and B. The C and D parameters are used for return value. Comparison between objects of any Fvec class returns the same class.

### Conditional Select Operators for Fvec Classes

<table>
<thead>
<tr>
<th>Conditional Select for:</th>
<th>Operators</th>
<th>Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equality</td>
<td>select_eq</td>
<td>R = select_eq(A, B)</td>
</tr>
<tr>
<td>Inequality</td>
<td>select_neq</td>
<td>R = select_neq(A, B)</td>
</tr>
<tr>
<td>Greater Than</td>
<td>select_gt</td>
<td>R = select_gt(A, B)</td>
</tr>
<tr>
<td>Greater Than or Equal To</td>
<td>select_ge</td>
<td>R = select_ge(A, B)</td>
</tr>
<tr>
<td>Not Greater Than</td>
<td>select_gt</td>
<td>R = select_gt(A, B)</td>
</tr>
<tr>
<td>Not Greater Than or Equal To</td>
<td>select_ge</td>
<td>R = select_ge(A, B)</td>
</tr>
<tr>
<td>Less Than</td>
<td>select_lt</td>
<td>R = select_lt(A, B)</td>
</tr>
<tr>
<td>Less Than or Equal To</td>
<td>select_le</td>
<td>R = select_le(A, B)</td>
</tr>
<tr>
<td>Not Less Than</td>
<td>select_nlt</td>
<td>R = select_nlt(A, B)</td>
</tr>
<tr>
<td>Not Less Than or Equal To</td>
<td>select_nle</td>
<td>R = select_nle(A, B)</td>
</tr>
</tbody>
</table>

**Conditional Select Operator Usage**

For conditional select operators, the return value is stored in C if the comparison is true or in D if false. The following table shows the return values for each class.
of the conditional select operators, using the Return Value Notation described earlier.

**Compare Operator Return Value Mapping**

<table>
<thead>
<tr>
<th>R</th>
<th>A0</th>
<th>Operators</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>F32vec4</th>
<th>F64vec2</th>
<th>F32vec1</th>
</tr>
</thead>
</table>
| R0:= (A1 select_[eq | lt | le | gt | ge] select_[ne | nlt | nle | ngt | nge]
     ! (A1 le | gt | ge]
                 B0) C0 D0 X X X             |
| R1:= (A2 select_[eq | lt | le | gt | ge] select_[ne | nlt | nle | ngt | nge]
     ! (A2 le | gt | ge]
                 B1) C1 D1 X X N/A             |
| R2:= (A2 select_[eq | lt | le | gt | ge] select_[ne | nlt | nle | ngt | nge]
     ! (A2 le | gt | ge]
                 B2) C2 D2 X N/A N/A             |
| R3:= (A3 select_[eq | lt | le | gt | ge] select_[ne | nlt | nle | ngt | nge]
     ! (A3 le | gt | ge]
                 B3) C3 D3 X N/A N/A             |

The following table shows examples for conditional select operations and corresponding intrinsics.

**Conditional Select Operations for Fvec Classes**

<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare for Equality</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 4 floats | F32vec4 R = _mm_cmpeq_ps
<pre><code>      | select_eq(F32vec4 A); | |
</code></pre>
<table>
<thead>
<tr>
<th>Returns</th>
<th>Example Syntax Usage</th>
<th>Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Compare for Equality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 ( R = ) _mm_cmpeq_pd select_eq(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 ( R = ) _mm_cmpeq_ss select_eq(F32vec1 A);</td>
<td></td>
</tr>
<tr>
<td><strong>Compare for Inequality</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 ( R = ) _mm_cmpneq_ps select_neq(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 ( R = ) _mm_cmpneq_pd select_neq(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 ( R = ) _mm_cmpneq_ss select_neq(F32vec1 A);</td>
<td></td>
</tr>
<tr>
<td><strong>Compare for Less Than</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 floats</td>
<td>F32vec4 ( R = ) _mm_cmplt_ps select_lt(F32vec4 A);</td>
<td></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2 ( R = ) _mm_cmplt_pd select_lt(F64vec2 A);</td>
<td></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1 ( R = ) _mm_cmplt_ss select_lt(F32vec1 A);</td>
<td></td>
</tr>
<tr>
<td>Compare for Less Than or Equal</td>
<td>4 floats</td>
<td>$F32\text{vec}4\ R = _\text{mm}_\text{cmple}_\text{ps}$</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>----------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{le}(F32\text{vec}4\ A)$</td>
</tr>
<tr>
<td>2 doubles</td>
<td>$F64\text{vec}2\ R = _\text{mm}_\text{cmple}_\text{pd}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{le}(F64\text{vec}2\ A)$</td>
</tr>
<tr>
<td>1 float</td>
<td>$F32\text{vec}1\ R = _\text{mm}_\text{cmple}_\text{ps}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{le}(F32\text{vec}1\ A)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Greater Than</th>
<th>4 floats</th>
<th>$F32\text{vec}4\ R = _\text{mm}_\text{cmpgt}_\text{ps}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{gt}(F32\text{vec}4\ A)$</td>
</tr>
<tr>
<td>2 doubles</td>
<td>$F64\text{vec}2\ R = _\text{mm}_\text{cmpgt}_\text{pd}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{gt}(F64\text{vec}2\ A)$</td>
</tr>
<tr>
<td>1 float</td>
<td>$F32\text{vec}1\ R = _\text{mm}_\text{cmpgt}_\text{ss}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{gt}(F32\text{vec}1\ A)$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Compare for Greater Than or Equal To</th>
<th>4 floats</th>
<th>$F32\text{vec}1\ R = _\text{mm}_\text{cmpge}_\text{ps}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{ge}(F32\text{vec}4\ A)$</td>
</tr>
<tr>
<td>2 doubles</td>
<td>$F64\text{vec}2\ R = _\text{mm}_\text{cmpge}_\text{pd}$</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\text{select}_\text{ge}(F64\text{vec}2\ A)$</td>
</tr>
<tr>
<td>1 float</td>
<td>$F32\text{vec}1\ R = _\text{mm}_\text{cmpge}_\text{ss}$</td>
<td></td>
</tr>
</tbody>
</table>
### Compare for Greater Than or Equal To

```cpp
select_ge(F32vec1 A);
```

### Compare for Not Less Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Vector Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1</td>
<td><code>select_nlt(F32vec4 A);</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td><code>select_nlt(F64vec2 A);</code></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td><code>select_nlt(F32vec1 A);</code></td>
</tr>
</tbody>
</table>

### Compare for Not Less Than or Equal

<table>
<thead>
<tr>
<th>Type</th>
<th>Vector Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1</td>
<td><code>select_nle(F32vec4 A);</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td><code>select_nle(F64vec2 A);</code></td>
</tr>
<tr>
<td>1 float</td>
<td>F32vec1</td>
<td><code>select_nle(F32vec1 A);</code></td>
</tr>
</tbody>
</table>

### Compare for Not Greater Than

<table>
<thead>
<tr>
<th>Type</th>
<th>Vector Type</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 floats</td>
<td>F32vec1</td>
<td><code>select_ngt(F32vec4 A);</code></td>
</tr>
<tr>
<td>2 doubles</td>
<td>F64vec2</td>
<td><code>_mm_cmpngt_pd</code></td>
</tr>
</tbody>
</table>

1649
### Compare for Not Greater Than

```cpp
select_ngt(F64vec2 A);
```

**1 float**

```cpp
F32vec1 R = _mm_cmpngt_ss select_ngt(F32vec1 A);
```

### Compare for Not Greater Than or Equal

**4 floats**

```cpp
F32vec1 R = _mm_cmpnge_ps select_nge(F32vec4 A);
```

**2 doubles**

```cpp
F64vec2 R = _mm_cmpnge_pd select_nge(F64vec2 A);
```

**1 float**

```cpp
F32vec1 R = _mm_cmpnge_ss select_nge(F32vec1 A);
```

### Cacheability Support Operations

Stores (non-temporal) the two double-precision, floating-point values of \( A \).

Requires a 16-byte aligned address.

```cpp
void store_nta(double *p, F64vec2 A);
```

**Corresponding intrinsic:** `_mm_stream_pd`

Stores (non-temporal) the four single-precision, floating-point values of \( A \).

Requires a 16-byte aligned address.

```cpp
void store_nta(float *p, F32vec4 A);
```

**Corresponding intrinsic:** `_mm_stream_ps`

### Debugging
The debug operations do not map to any compiler intrinsics for MMX™ technology or Streaming SIMD Extensions. They are provided for debugging programs only. Use of these operations may result in loss of performance, so you should not use them outside of debugging.

Output Operations

The two single, double-precision floating-point values of A are placed in the output buffer and printed in decimal format as follows:

cout << F64vec2 A;
"[1]:A1 [0]:A0"

Corresponding intrinsics: none

The four, single-precision floating-point values of A are placed in the output buffer and printed in decimal format as follows:

cout << F32vec4 A;
"[3]:A3 [2]:A2 [1]:A1 [0]:A0"

Corresponding intrinsics: none

The lowest, single-precision floating-point value of A is placed in the output buffer and printed.

cout << F32vec1 A;

Corresponding intrinsics: none

Element Access Operations

double d = F64vec2 A[int i]

Read one of the two, double-precision floating-point values of A without modifying the corresponding floating-point value. Permitted values of i are 0 and 1. For example:

If DEBUG is enabled and i is not one of the permitted values (0 or 1), a diagnostic message is printed and the program aborts.

double d = F64vec2 A[1];

Corresponding intrinsics: none
Read one of the four, single-precision floating-point values of $A$ without modifying the corresponding floating point value. Permitted values of $i$ are 0, 1, 2, and 3. For example:

```cpp
defloat f = F32vec4 A[int i]
```

If DEBUG is enabled and $i$ is not one of the permitted values (0-3), a diagnostic message is printed and the program aborts.

```cpp
defloat f = F32vec4 A[2];
```

Corresponding intrinsics: none

**Element Assignment Operations**

```cpp
F64vec4 A[int i] = double d;
```

Modify one of the two, double-precision floating-point values of $A$. Permitted values of $i$ are 0 and 1. For example:

```cpp
F32vec4 A[int i] = float f;
```

Modify one of the four, single-precision floating-point values of $A$. Permitted values of $i$ are 0, 1, 2, and 3. For example:

```cpp
```

Corresponding intrinsics: none.

**Load and Store Operators**

Loads two, double-precision floating-point values, copying them into the two, floating-point values of $A$. No assumption is made for alignment.

```cpp
void loadu(F64vec2 A, double *p)
```

Corresponding intrinsic: `_mm_loadu_pd`

Stores the two, double-precision floating-point values of $A$. No assumption is made for alignment.

```cpp
void storeu(float *p, F64vec2 A);
```

Corresponding intrinsic: `_mm_storeu_pd`
Loads four, single-precision floating-point values, copying them into the four floating-point values of \( A \). No assumption is made for alignment.

```cpp
void loadu(F32vec4 A, double *p)
```

**Corresponding intrinsic:** 
\[
_mm_loadu_ps
\]

Stores the four, single-precision floating-point values of \( A \). No assumption is made for alignment.

```cpp
void storeu(float *p, F32vec4 A);
```

**Corresponding intrinsic:** 
\[
_mm_storeu_ps
\]

### Unpack Operators for Fvec Operators

Selects and interleaves the lower, double-precision floating-point values from \( A \) and \( B \).

```cpp
F64vec2 R = unpack_low(F64vec2 A, F64vec2 B);
```

**Corresponding intrinsic:** 
\[
_mm_unpacklo_pd(a, b)
\]

Selects and interleaves the higher, double-precision floating-point values from \( A \) and \( B \).

```cpp
F64vec2 R = unpack_high(F64vec2 A, F64vec2 B);
```

**Corresponding intrinsic:** 
\[
_mm_unpackhi_pd(a, b)
\]

Selects and interleaves the lower two, single-precision floating-point values from \( A \) and \( B \).

```cpp
F32vec4 R = unpack_low(F32vec4 A, F32vec4 B);
```

**Corresponding intrinsic:** 
\[
_mm_unpacklo_ps(a, b)
\]

Selects and interleaves the higher two, single-precision floating-point values from \( A \) and \( B \).

```cpp
F32vec4 R = unpack_high(F32vec4 A, F32vec4 B);
```

**Corresponding intrinsic:** 
\[
_mm_unpackhi_ps(a, b)
\]

### Move Mask Operator

Creates a 2-bit mask from the most significant bits of the two, double-precision floating-point values of \( A \), as follows:
int i = move_mask(F64vec2 A)
i := sign(a1)<<1 | sign(a0)<<0

**Corresponding intrinsic:** `_mm_movemask_pd`

Creates a 4-bit mask from the most significant bits of the four, single-precision floating-point values of A, as follows:

int i = move_mask(F32vec4 A)
i := sign(a3)<<3 | sign(a2)<<2 | sign(a1)<<1 | sign(a0)<<0

**Corresponding intrinsic:** `_mm_movemask_ps`

### Classes Quick Reference

This appendix contains tables listing operators to perform various SIMD operations, corresponding intrinsics to perform those operations, and the classes that implement those operations. The classes listed here belong to the Intel C++ Class Libraries for SIMD Operations.

In the following tables,

- **N/A** indicates that the operator is not implemented in that particular class.
  
  Thus, in the Logical Operations table, the **Andnot** operator is not implemented in the **F32vec4** and **F32vec1** classes.

- All other entries under **Classes** indicate that those operators are implemented in those particular classes, and the entries under the **Classes** columns provide the suffix for the corresponding intrinsic. For example, consider the Arithmetic Operations Part1 table, where the corresponding intrinsic is `_mm_add_[x]` and the entry `epi16` is under the **I16vec8** column. It means that the **I16vec8** class implements the addition operators and the corresponding intrinsic is `_mm_add_epi16`.

#### Logical Operations:

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Classes</th>
</tr>
</thead>
</table>

1654
<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I64vec2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>si128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>si128</td>
</tr>
<tr>
<td></td>
<td></td>
<td>si128</td>
</tr>
</tbody>
</table>

Arithmetic Operations: Part 1
### Arithmetic Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I32vec2</td>
</tr>
<tr>
<td>+, +=</td>
<td><em>mm_add</em>[x]</td>
<td>pi32</td>
</tr>
<tr>
<td>- , -=</td>
<td><em>mm_sub</em>[x]</td>
<td>pi32</td>
</tr>
<tr>
<td>*, *=</td>
<td><em>mm_mullo</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>/, /=</td>
<td><em>mm_div</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>mul_high</td>
<td><em>mm_mulhi</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>mul_add</td>
<td><em>mm_madd</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>sqrt</td>
<td><em>mm_sqrt</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rcp</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>rcp_nr</td>
<td><em>mm_rcp</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

### Shift Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I64vec2</td>
</tr>
<tr>
<td>rsqrt_nr</td>
<td><em>mm_rsqrt</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_sub</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td><em>mm_mul</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Shift Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>128vec1</th>
<th>64vec2</th>
<th>32vec4</th>
<th>16vec8</th>
<th>8vec16</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;&gt;, &gt;&gt;==</td>
<td><em>mm_srl</em>[x]</td>
<td>N/A</td>
<td>epi64</td>
<td>epi32</td>
<td>epi16</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_srli</em>[x]</td>
<td>N/A</td>
<td>epi64</td>
<td>epi32</td>
<td>epi16</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_sra</em>_[x]</td>
<td>N/A</td>
<td>N/A</td>
<td>epi32</td>
<td>epi16</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_srai</em>_[x]</td>
<td>N/A</td>
<td>N/A</td>
<td>epi32</td>
<td>epi16</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>&lt;&lt;, &lt;&lt;=</td>
<td><em>mm_sll</em>[x]</td>
<td>N/A</td>
<td>epi64</td>
<td>epi32</td>
<td>epi16</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_slli</em>_[x]</td>
<td>N/A</td>
<td>epi64</td>
<td>epi32</td>
<td>epi16</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

### Comparison Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>32vec4</th>
<th>16vec8</th>
<th>8vec16</th>
<th>32vec2</th>
<th>16vec4</th>
<th>8vec8</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmpeq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>p32</td>
<td>p16</td>
<td>p8</td>
<td></td>
</tr>
<tr>
<td>cmpneq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>p32</td>
<td>p16</td>
<td>p8</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td>cmpgt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>p32</td>
<td>p16</td>
<td>p8</td>
<td></td>
</tr>
<tr>
<td>cmpge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>p32</td>
<td>p16</td>
<td>p8</td>
<td></td>
</tr>
</tbody>
</table>
**Comparison Operations: Part 2**

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F64vec2</td>
</tr>
<tr>
<td>cmpeq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td>cmpneq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td>cmpgt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td>cmpge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td>cmplt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td>cmple</td>
<td><em>mm_cmple</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td>cmpngt</td>
<td><em>mm_cmpngt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td>cmpnge</td>
<td><em>mm_cmpnge</em>[x]</td>
<td>pd</td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.
### Conditional Select Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>F64vec2</th>
<th>F32vec4</th>
<th>F32vec1</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmnplnt</td>
<td><em>mm_cmnplnt</em>[x]</td>
<td>pd</td>
<td>ps</td>
<td>ss</td>
<td></td>
</tr>
<tr>
<td>cmpnle</td>
<td><em>mm_cmpnle</em>[x]</td>
<td>pd</td>
<td>ps</td>
<td>ss</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>I32vec4</th>
<th>I16vec8</th>
<th>I8vec16</th>
<th>I32vec2</th>
<th>I16vec4</th>
<th>I8vec8</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_eq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>pi32</td>
<td>pi16</td>
<td>pi8</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>I32vec4</th>
<th>I16vec8</th>
<th>I8vec16</th>
<th>I32vec2</th>
<th>I16vec4</th>
<th>I8vec8</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_neq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>pi32</td>
<td>pi16</td>
<td>pi8</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>I32vec4</th>
<th>I16vec8</th>
<th>I8vec16</th>
<th>I32vec2</th>
<th>I16vec4</th>
<th>I8vec8</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_gt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>pi32</td>
<td>pi16</td>
<td>pi8</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>I32vec4</th>
<th>I16vec8</th>
<th>I8vec16</th>
<th>I32vec2</th>
<th>I16vec4</th>
<th>I8vec8</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_ge</td>
<td><em>mm_cmpege</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>pi32</td>
<td>pi16</td>
<td>pi8</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
<th>I32vec4</th>
<th>I16vec8</th>
<th>I8vec16</th>
<th>I32vec2</th>
<th>I16vec4</th>
<th>I8vec8</th>
</tr>
</thead>
<tbody>
<tr>
<td>select_lt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>epi32</td>
<td>epi16</td>
<td>epi8</td>
<td>pi32</td>
<td>pi16</td>
<td>pi8</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td>si128</td>
<td>si128</td>
<td>si128</td>
<td>si64</td>
<td>si64</td>
<td>si64</td>
<td></td>
</tr>
</tbody>
</table>
### Conditional Select Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F64vec2</td>
</tr>
<tr>
<td>select_eq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_neq</td>
<td><em>mm_cmpeq</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_gt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
</tbody>
</table>

* Note that _mm_andnot_[y] intrinsics do not apply to the fvec classes.
## Classes

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>F64vec2</td>
</tr>
<tr>
<td>select_ge</td>
<td><em>mm_cmpge</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_lt</td>
<td><em>mm_cmplt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_le</td>
<td><em>mm_cmple</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_and</em>[y]</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_andnot</em>[y]*</td>
<td></td>
</tr>
<tr>
<td></td>
<td><em>mm_or</em>[y]</td>
<td></td>
</tr>
<tr>
<td>select_nlt</td>
<td><em>mm_cmpgt</em>[x]</td>
<td>pd</td>
</tr>
<tr>
<td></td>
<td><em>mm_cmpge</em>[x]</td>
<td></td>
</tr>
<tr>
<td>select_nge</td>
<td><em>mm_cmpge</em>[x]</td>
<td></td>
</tr>
<tr>
<td>select_nlt</td>
<td><em>mm_cmplt</em>[x]</td>
<td></td>
</tr>
<tr>
<td>select_nle</td>
<td><em>mm_cmple</em>[x]</td>
<td></td>
</tr>
</tbody>
</table>

## Packing and Unpacking Operations: Part 1

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>I64vec2</td>
</tr>
<tr>
<td>unpack_high</td>
<td><em>mm_unpackhi</em>[x]</td>
<td>epi64</td>
</tr>
<tr>
<td>unpack_low</td>
<td><em>mm_unpacklo</em>[x]</td>
<td>epi64</td>
</tr>
<tr>
<td>pack_sat</td>
<td><em>mm_packss</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>packu_sat</td>
<td><em>mm_packuss</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>
### Operators

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>sat_add</td>
<td><em>mm_adds</em>[x]</td>
<td>N/A</td>
</tr>
<tr>
<td>sat_sub</td>
<td><em>mm_subs</em>[x]</td>
<td>N/A</td>
</tr>
</tbody>
</table>

#### Packing and Unpacking Operations: Part 2

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
<th>Classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>unpack_high</td>
<td><em>mm_unpackhi</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>unpack_low</td>
<td><em>mm_unpacklo</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>pack_sat</td>
<td><em>mm_packs</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>packu_sat</td>
<td><em>mm_packus</em>[x]</td>
<td>pu16</td>
</tr>
<tr>
<td>sat_add</td>
<td><em>mm_adds</em>[x]</td>
<td>pi16</td>
</tr>
<tr>
<td>sat_sub</td>
<td><em>mm_subs</em>[x]</td>
<td>pi16</td>
</tr>
</tbody>
</table>

#### Conversions Operations:

Conversion operations can be performed using intrinsics only. There are no classes implemented to correspond to these intrinsics.

<table>
<thead>
<tr>
<th>Operators</th>
<th>Corresponding Intrinsic</th>
</tr>
</thead>
<tbody>
<tr>
<td>F64vec2ToInt</td>
<td>_mm_cvtsd_si32</td>
</tr>
<tr>
<td>F32vec4ToF64vec2</td>
<td>_mm_cvtps_pd</td>
</tr>
<tr>
<td>F64vec2ToF32vec4</td>
<td>_mm_cvtpd_ps</td>
</tr>
<tr>
<td>IntToF64vec2</td>
<td>_mm_cvtsi32_sd</td>
</tr>
<tr>
<td>F32vec4ToInt</td>
<td>_mm_cvtt_ss2si</td>
</tr>
<tr>
<td>F32vec4ToIs32vec2</td>
<td>_mm_cvttps_pi32</td>
</tr>
</tbody>
</table>
Programming Example

This sample program uses the F32vec4 class to average the elements of a 20 element floating point array.

```c
// Include Streaming SIMD Extension Class Definitions
#include <fvec.h>

// Shuffle any 2 single precision floating point from a
// into low 2 SP FP and shuffle any 2 SP FP from b
// into high 2 SP FP of destination
#define SHUFFLE(a,b,i) (F32vec4)_mm_shuffle_ps(a,b,i)
#include <stdio.h>
#define SIZE 20

// Global variables
float result;
_MM_ALIGN16 float array[SIZE];

void Add20ArrayElements (F32vec4 *array, float *result)
{
    F32vec4 vec0, vec1;
    vec0 = _mm_load_ps ((float *) array); // Load array's first 4 floats

    // Add all elements of the array, 4 elements at a time
    vec0 += array[1]; // Add elements 5-8
    vec0 += array[2]; // Add elements 9-12
    vec0 += array[3]; // Add elements 13-16
    vec0 += array[4]; // Add elements 17-20

    // There are now 4 partial sums.
    // Add the 2 lowers to the 2 raises,
    // then add those 2 results together
```
vec1 = SHUFFLE(vec1, vec0, 0x40);
vec0 += vec1;
vec1 = SHUFFLE(vec1, vec0, 0x30);
vec0 += vec1;
vec0 = SHUFFLE(vec0, vec0, 2);
_mm_store_ss (result, vec0); // Store the final sum
}

void main(int argc, char *argv[])
{
    int i;
    // Initialize the array
    for (i=0; i < SIZE; i++)
    {
        array[i] = (float) i;
    }

    // Call function to add all array elements
    Add20ArrayElements (array, &result);

    // Print average array element value
    printf ("Average of all array values = %f\n", result/20.);
    printf ("The correct answer is %f\n\n", 9.5);
}

C/C++ Language Extensions

Introduction

This section contains descriptions of Intel's C/C++ Language Extensions that assist users in parallel programming. The following C/C++ language extensions are included:

- **C++ Lambda Expressions or Functions**: these language extensions enable easy usage of Intel®) Threading Building Blocks (TBB), making parallelism more available to developers.

C++ Lambda Extensions

Introduction


C++ Lambda expressions are primary expressions that define function objects. Such expressions can be used wherever a function object is expected; for example, as arguments to Standard Template Library (STL) algorithms.
This section contains the following topics:

- About Using C++ Lambda Expressions
- Understanding Lambda-Capture
- Lambda Function Object

Lambda Functions

This section explains in some detail about using Intel's implementation of C++ Lambda expressions. In order to use lambda expressions, you need to request c++0x with the command-line option -std or /Qstd:

On Windows* systems: /Qstd=c++0x
On Linux* systems: -std=c++0x

Introducing Lambda Expressions

Lambda expressions are introduced in the code by [lambda-captureopt]. If the expression does not capture any local variables or references with automatic storage duration, lambda-captureopt can be empty, leaving [] as the introducer.

For example, the lambda expression

```cpp
[](int x) {return x%3==0;}
```

is equivalent to the expression `unique()`, where `unique` is a secret identifier generated and defined by the compiler as

```cpp
class unique {
    public:
        bool operator() (int x) const {return x%3==0;}
};
```

Parameter List in Lambda Expressions

The `lambda-parameter-declaration` follows the `lambda-introducer`. If the parameter list is empty, the parentheses can be omitted. For example, the following lambda expressions are equivalent.

```cpp
[] {return rand();}
[]( ) {return rand();}
```

Body and Return Type of Lambda Expressions

The body of a lambda expression is a compound statement. A return type $T$ for a lambda expression can be specified by writing $\rightarrow T$ after the parameter list.
For example, the following lambda expressions are equivalent:

\[
\begin{align*}
&[](\text{int } x) \{ \text{return } x \% 3 == 0; \} \\
&[](\text{int } x) \to \text{bool} \{ \text{return } x \% 3 == 0; \} \\
&[](\text{int } x) \to \text{bool} \{ \text{if} ( x \% 3 == 0 ) \text{ return true; else return false; } \}
\end{align*}
\]

If the return type is not explicitly specified, the return type is \texttt{void} unless the body has the form \{\text{return expression;}\}. In such a case the return type is the type of the expression. Consider the following example:

\[
[](\text{int } x) \{ \text{if} ( x \% 3 == 0 ) \text{ return true; else return false; } \}
\]

This expression is an error because you can't return a boolean value from a function returning void.

### Exception Specification

A lambda expression may include an exception specification after the parameter list and before the return type specification. The parameter list is necessary if there is an exception specification.

The following lambda expressions specify that they do not throw exceptions:

\[
\begin{align*}
&[](\text{int } x) \text{ throw()} \to \text{bool} \{ \text{return } x \% 3 == 0; \} \\
&[]() \text{ throw()} \{ \text{return } \text{rand}(); \}
\end{align*}
\]

### See Also

- [Understanding \texttt{[lambda-capture}\texttt{opt}]]

### Understanding Lambda-Capture

A lambda expression can refer to identifiers declared outside the lambda expression. If the identifier is a local variable or a reference with automatic storage duration, it is an up-level reference and must be "captured" by the lambda expression. Such a lambda expression must be introduced by \texttt{[lambda-capture}\texttt{opt}], where \texttt{lambda-capture}\texttt{opt} specifies whether identifiers are captured by reference or by copy. The table below summarizes forms of \texttt{lambda-capture}\texttt{opt}.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Indicates</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Symbol & Indicates

[] Nothing to capture: an up-level reference is an error

[&x, y, ...] Capture as specified: identifiers prefixed by & are captured by reference; other identifiers are captured by copy. An up-level reference to any variable not explicitly listed is an error

[&] Capture by reference: an up-level reference implicitly captures the variable by reference

[,] Capture by copy: an up-level reference implicitly captures the variable by copy

[, x, y, ...] Capture by reference with exceptions: listed variables are captured by value/copy (no listed variable may be prefixed by &)

[=, &x, &y, ...] Capture by copy with exceptions: listed variables are captured by reference only (every listed variable must be prefixed by &)

No identifier may appear twice in the list. In the following code that sets area to the sum of the areas of four circles, the notation [&area,pi] specifies that area is captured by reference and pi by copy.

```cpp
float radius[] = {2,3,5,7};
float area=0;
float pi = 3.14f;
for_each(radius, radius+4, [&area,pi](float r) {
  return area+=pi*r*r;
})
```

### Specifying Default Capture

When a default capture is specified, the list must specify only the other kind of capture. In other words, if you specify that the default capture is by reference, then you must list (and only list) the variables that should be captured by copy. For example, if your intent is to capture x by reference and y by copy, and both x and y appear in the function body, the following code illustrates what is correct and incorrect code:

```cpp
[&,&x,y] // ERROR - default is capture-by-reference; you must list only capture by copy
```
Default Binding Modes

The following lambda expressions demonstrate default binding modes. All three expressions are semantically equivalent. Each captures $x$ and $y$ by reference, and captures $a$ and $b$ by copy.

```cpp
 [&x,&a,b](float r) {x=a; y=b;}
 [&,a,b](float r) {x=a; y=b;}
 [=,&x,&a,b](float r) {x=a; y=b;}
```

Referring to this

If a lambda expression occurs inside a member function, it can refer to this. Because this is not a variable, it cannot be captured by reference. Even when it is captured implicitly in a lambda expression introduced by `[&]`, it is captured by copy.

Lambda Function Object

The compiler creates an anonymous function object upon evaluating a lambda expression. This function object, created by a lambda expression, may live longer than the block in which it is created. You must ensure that it does not use variables that were destroyed before the function object is used.

The following example shows how a function object created by a lambda expression can outlive the function that creates it.

```cpp
 struct Base {
   virtual bool test(int x) = 0;
};
 template<typename F>
 struct Derived: Base {
   F f;
   bool test(int x) {return f(x);}
   Derived(F f_) : f(f_) {}
};
```
In the above example, Bar invokes Foo, which copies the function object generated by a lambda expression into an instance of template class Derived. The lambda expression refers to a local variable k. Although the code destroys k before using the copied function object, the code is safe because k was captured by copy.