FUJITSU

FUJITSU Software
Technical Computing Suite V2.0

Fortran User's Guide (PRIMEHPC FX100)

J2UL-1865-02ENZ0(00)
November 2015

Preface

Purpose of This Manual

This manual explains how to use the Fortran system (called "this system" in this manual) that runs on the Linux operating system. It also describes the following aspects of the system:

- Fortran programming
- Restrictions

Intended Readers

This manual was written for people who write Fortran programs and process them using this system.

This manual assumes that readers know how to use Linux commands, file manipulation and shell programming.

This manual assumes an understanding of the Fortran language specification. For details of the Fortran language specification, refer to "Fortran Language Reference" on online manual, the Fortran language standard, or commercial books.

Organization of This Manual

This manual is organized as follows:

Chapter 12 Multiprocessing

This chapter describes how parallel processing is performed for a Fortran program in this system.

Appendix A Limits and Restrictions

This appendix describes limits and restrictions on programing using this system.

Appendix B Printing Control Command

This appendix describes printing control commands provided by this system.

Appendix C Endian Convert Command

This appendix describes endian convert commands provided by this system.

Appendix D Job Operation Software

This appendix describes information on the compilation and execution processes for using the high-speed facility on the Job Operation Software.

Appendix E Preprocessor

This appendix describes the C language preprocessor and the Fortran preprocessor.

Appendix F Notes on Migration from FX10 System to FX100 System

This appendix describes notes on migrating from FX10 system to FX100 system.

Appendix G Compatibility Information (FX10 System)

This appendix describes compatibility information as notes on migrating.

Appendix H Compatibility Information (FX100 System)

This appendix describes compatibility information as notes on migrating.

Related Manuals

The following manuals are related to this manual.

- Fortran Language Reference
- Fortran Compiler Messages
- Fortran/C/C++ Runtime Messages
- Fortran User's Guide Additional Volume COARRAY

Notes of This Manual

The code examples of optimization in this manual are conceptual source that complements each explanation of functions. When the code examples are compiled and executed, optimizations may not work as expected. This is because optimizations depend on compilation options and other conditions.

Notation Used in This Manual

Syntax Description Symbols

A syntax description symbol is a symbol that has a specific meaning when used to describe syntax. The following symbols are used in this manual:

Symbol name	Symbol	Explanation
Selection symbols	{ }	Only one of the items enclosed in the braces must be selected (items are listed vertically).
	|	Multiple items are enumerated by this delimiter (items are listed horizontally).
Option symbol	[ ]	An item enclosed in brackets can be omitted. This symbol includes the meaning of the selection symbol "{}".

Symbol name	Symbol	Explanation
Repeat symbol	...	The item immediately preceding the ellipsis can be specified repeatedly in the syntax.

Parallel

The way of parallelization is thread parallelization unless otherwise described.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademarks

- Oracle and Java are registered trademarks of Oracle and/or its affiliates.
- All SPARC trademarks are used under license from SPARC International, Inc. and are trademarks or registered trademarks of that company in the United States and other countries.
- OpenMP is a trademark of OpenMP Architecture Review Board.
- Linux is a trademark or registered trademark of Linus Torvalds in the United States and other countries.
- All other trademarks and product names are the property of their respective owners. The trademark notice symbol (TM, (R)) is not necessarily added in the system name and the product name, etc. published in this material.

Date of Publication and Version

Version	Manual code
November 2015, 2nd Version	J2UL-1865-02ENZ0(00)
February 2015, Version 1.1	J2UL-1865-01ENZ0(01)
October 2014, 1st Version	J2UL-1865-01ENZ0(00)

Copyright

Copyright FUJITSU LIMITED 2014-2015

Update History

Changes	Location	Version
Added the article "Parallel" to "Notation Used in This Manual".	Preface	2nd Version
Added the article on the following option: - -K{ openmp_tls | openmp_notls }	2.2 12.3.1.1.1	
Added articles on the following options: - -K{ FLTLD | NOFLTLD } - -K{ HPC_ACE | HPC_ACE2 } - -K{ intentopt | nointentopt } - -K{ lto | nolto } - -K{ simd_separate_stride | simd_noseparate_stride } - -N{ coarray | nocoarray }	2.2	
Changed articles on the following options:		

Changes	Location	Version
- -g - -K{ fed | nofed } - -Kfp_relaxed - -K{ nf | nonf } - -Knoalias[=*spec*] - -Ksimd=auto - -NRtrap - -Nsetvalue=struct - -K{ uxsimd | nouxsimd } - -S - -W		
Changed the article of the -Lu option.	3.3	
Added the article on the environment variable FLIB_USE_STOPCODE.	3.8	
Added the article on "Notes for Execution"	3.9.1	
Added articles of the following notes: - Link Time Optimization - #line directives	4.1.2.3	
Changed articles in response to the specification addition of "ERROR STOP statement".	4.2 4.2.1 6.5.7 8.1.8	
Changed the article on the STATUS= specifier.	7.5.1	
Changed the article.	8.2.1.3	
Corrected the code example of List Vector Conversion.	9.1.1.7.3	
Added the article "SIMD with Redundant Executions for the SIMD Width".	9.1.1.7.4	
Changed the article.	9.1.2.6	
Added the article "Link Time Optimization".	9.1.2.10	
Changed the article.	9.7.2	
Added articles on the following optimization control specifiers: - { FLTLD | NOFLTLD } - SIMD(ALIGNED) - SIMD(UNALIGNED) - { SIMD_REDUNDANT_VL(*n*) | SIMD_NOREDUNDANT_VL } - UXSIMD(ALIGNED) - UXSIMD(UNALIGNED)	9.10.4	
Changed articles on the following optimization control specifiers: - FP_RELAXED - { NF | NONF } - NOALIAS		

Changes	Location	Version
- SIMD - UXSIMD		
Corrected the code example of the optimization control specifier NOVREC.		
Added the description in restrictions of the target variable of the dimension shift of the array declaration.	9.14.2	
Added the description in effects of merging array variables.	9.15.1	
Added the description in restrictions of merging local array variables.	9.15.2	
Added the article "Example of SIMD(ALIGNED) specifier".	9.18.1	
Corrected the article "Notes when using service routines".	12.2.1.2.7 12.3.1.2.3	
Corrected the article on the environment variable OMP_PROC_BIND.	12.3.2	
Changed the article on BIND attribute.	12.3.3.7	
Added the article "Derived type variable with length type parameter" to OpenMP specifications and restrictions.	12.3.3.11	
Added BLOCK construct and CRITICAL construct in limits of total nesting level.	A.2	
Updated notes on migration.	Appendix F	
Added that the specification changes of the -K{ FLTLD | NOFLTLD } options and the optimization control specifiers { FLTLD | NOFLTLD } are only influenced when the -KHPC_ACE2 option is set.	F.7	
Added that the specification changes of the optimization control specifiers SIMD and UXSIMD are only influenced when the -KHPC_ACE2 option is set.	F.12	
Added the compatibility information.	G.1 H.1	
Reviewed the code examples and command execution examples.	-	
Added the following articles: - Related Manuals - Notes of This Manual	Preface	Version 1.1
Corrected the article of -Ay option.	2.2	
Added articles on the following options: - -K{ openmp_assume_norecurrence | openmp_noassume_norecurrence } - -N{ cancel_overtime_compilation | nocancel_overtime_compilation }		
Changed the option name from -K{ ordered_omp_reduction | noordered_omp_reduction } to -K{ openmp_ordered_reduction | openmp_noordered_reduction }.	2.2 12.3.1.1.1	
Changed the article of -Lu option.	3.3	
Changed the output of SIMD information.	4.1.2.1 4.1.2.2	
Added the article on Fortran record which length is more than 2G bytes.	7.3.2.1	
Deleted articles on the error number 1164 because the upper size limit of the output item list was changed.	Chapter 8	
Added the article of software pipelining.	9.1.1.6	
Changed the article on the optimization control specifier ARRAY_MERGE.	9.10.4	
Corrected the code example of the optimization control specifier NOVREC.		

Changes	Location	Version
Added the article of loop distribution and automatic loop slicing.	12.2.3.1.6	
Added the article on linkage with other multi-thread programs.	12.2.4 12.3.5	
Changed the term from "hardware barrier in a cpu chip" to "on-chip hardware barrier".	Appendix D	
Updated notes on migration.	Appendix F	
Added the compatibility information.	G.1	
Added the compatibility information on FX100 system as the appendix.	Appendix H	

All rights reserved.
The information in this manual is subject to change without notice.

Contents

Chapter 1 Overview of the Fortran System

This chapter provides an overview of the Fortran system (or simply "the system") and illustrates the most common ways to use the system to develop and run a Fortran program.

1.1 Configuration of the Fortran System

The Fujitsu Fortran system runs under the Operating System based on Linux.

The system consists of the following components. There are two types of compile commands, one is the cross compile command that is used on the login node, and the other is the own compile command that is used on the compute node.

- The Fortran driver frtpx (cross compiler command) or frt (own compiler command), calls the Fortran compiler, the preprocessor, the assembler, and the linker.

  The frtpx is used as the name of the driver at the following. Replace the frtpx as the frt when you use the own compiler.

- The Fortran compiler creates object programs from Fortran source programs.
- The Fortran library provides commonly used start-up routines, input/output facilities, and intrinsic procedures.
- Printing control command fot/fotpx and endian convert command fcvendian/fcvendianpx are included in the system.
- Online manual pages, accessed by the man command, provide information about frtpx(1), frt(1), fot(1), fotpx(1), fcvendian(1), fcvendianpx(1), intrinsic(3) and service(3).

The following is cross compiler command and own compiler command. Please read frtpx in a different way as frt when you use own compiler.

Component	Cross Compiler	Own Compiler
Fortran driver	frtpx	frt
Printing control command	fotpx	fot
Endian convert command	fcvendianpx	fcvendian

1.2 How to use Fortran System

1.2.1 Preparation

To use the system, following environmental variables must be set correctly.

Environment variable	Value
PATH	/opt/FJSVmxlang/bin
LD_LIBRARY_PATH	/opt/FJSVmxlang/lib64
MANPATH	/opt/FJSVmxlang/man

Examples:

```
$ PATH=/opt/FJSVmxlang/bin:$PATH ; export PATH
$ LD_LIBRARY_PATH=/opt/FJSVmxlang/lib64:$LD_LIBRARY_PATH ; export LD_LIBRARY_PATH
$ MANPATH=/opt/FJSVmxlang/man:$MANPATH ; export MANPATH
```

1.2.2 Compilation and Linking

The frtpx command is used for compilation and linking.

The simplest case is to compile a Fortran program, stored in a file such as prog.f95. The command

```
$ frtpx prog.f95
```

compiles and links the program, producing an executable file named a.out. Specifying -c option causes only object file to be made, without linking.

```
$ frtpx -c prog.f95
```

1.2.3 File Names

The frtpx command may specify source files, object files, or assembly language files. Source files may have a .f90, .f95, .f, .for, .f03, .F90, .F95, .F, .FOR or .F03 suffix. If it is .f90, .f95 or .f03, the statements are assumed to be in free source form; if it is .f or .for, the statements are assumed to be in fixed source form; in either case, this assumption may be overridden with the -Free or -Fixed compiler option. If the suffix is .F90, .F95, .F, .FOR or .F03, the preprocessor is invoked before the compiler and the assumed source form is fixed and free, respectively.

Assembly language files have a .s suffix; they are not processed by the compiler but passed on to the assembler. An assembly language file is produced when the -S option is used.

Object files have a .o suffix. Such files are not processed by the compiler, but passed on to the linker.

1.2.4 Some Compilation Options

There are many compilation options; they are described in detail in Section "2.2 Compiler Options". Some of the most important and commonly used options are:

-c	Produce object files only and do not link to create an executable.
-fw	Output only warning and serious error messages (no informative messages).
-fs	Output only serious error messages.
-f*msg_num*	Suppress particular error message specified by *msg_num*.
-Nmaxserious=*maxnum*	Stop the compilation if s-level (serious) error detected more *maxnum*.
-o*exe_file*	The executable is named *exe_file* instead of a.out. If the -c option is also specified, the object file is named *exe_file*.
-Fixed	The program is in fixed source form.
-Free	The program is in free source form.
-H[a|e|f|o|s|u|x]	Check for argument mismatches (a), shape conformance (e), simultaneous OPEN and I/O recursive call (f), overlapping dummy arguments and extend undefined (o), subscript and substring values out of bounds (s), undefined variables (u), or module, common, and pointer undefined check (x). These may be combined: -Haesu or -Hsu.
-M*directory*	Names a *directory* to hold module information files
-I*directory*	Names a *directory* to search for include and module information files.
-O[{0|1|2|3}]	Specifies the optimization level. The default is the -O2.
-Kfast	Sets the many optimization parameters to reasonable values to achieve optimized performance
-X6	Compiles Fortran source program as FORTRAN66 source.
-X7	Compiles Fortran source program as FORTRAN77 source.
-X9	Compiles Fortran source program as Fortran 95 source.
-X03	Compiles Fortran source program as Fortran 2003 source.

1.2.5 Execution

You can run an executable program created by frtpx command. An example of executing a.out file follows:

```
$ ./a.out
```

You can specify instructions to Fortran library on invoking an executable file (see "Chapter 3 Executing Fortran Programs").

1.2.6 Debugging

There are two methods of debugging Fortran programs; one is batch debug function in this Fortran system, another is source-level interactive debug function.

See Section "8.2 Debugging Functions" for batch debug function.

See the "Debugger User's Guide" for source-level interactive debug function.

1.2.7 Tuning

There is a sampling function provided by the Profiler for tuning of Fortran programs.

See the "Profiler User's Guide" for the Profiler.

Chapter 2 Compiling and Linking Fortran Programs

This chapter tells you how to compile and link Fortran programs.

2.1 Compile Command

Use the compile command frtpx to compile and prepare Fortran programs for execution. The frtpx command analyzes its arguments. It then calls the cpp command for C preprocessing, the fpp command for Fortran preprocessing, Fortran compiler, the as command for the assembler, and the ld command for the linker, as necessary.

2.1.1 Compile Command Format

Specify the filename list and option list as arguments for the Fortran compiler, the as command, and the ld command. See Section "2.2 Compiler Options" for information on Fortran compiler options. For information on the as, and ld commands, use the man command to refer to the online manual.

The format of the frtpx command is as follows:

Command	Operand
frtpx	[_option-list]_file-name-list

_: At least one blank is required.

Notes:

1. The items in the option-list and file-name-list can be specified in any order. For example, you may specify the option-list after the file-name-list or mix option-list and file-name-list specifications.
2. At least one file name must be specified in the file-name-list, unless the entire operand is -V.
3. See Section "2.1.2 Compile Command Input File" for details of file-name-list.

2.1.2 Compile Command Input File

The following table lists the suffixes allowed for frtpx(1) file names.

File type	File suffix	Program
Fortran source program	.f	Fortran compiler
	.for	
	.f90	
	.f95	
	.f03	
Preprocessing directive Fortran source program (*1)	.F	Preprocessor followed by Fortran compiler
	.FOR	
	.F90	
	.F95	
	.F03	
Assembler source program	.s	Assembler
Object program	.o	Linker
Other than above	Other than above	Linker

*1: Preprocessing directives may be included in Fortran source programs. They have the same form as directives for the C language; each line begins with the symbol "#".

2.2 Compiler Options

Compiler options may be specified as option arguments of frtpx or by the compilation profile file or by the compiler directive lines or by the environment variable FORT90CPX or FORT90C.

See Section "2.3 Compile Command Environment Variable" for information about the environment variable FORT90CPX or FORT90C.

See Section "2.4 Compilation Profile File" for information about the compilation profile file.

See Section "2.5 Compiler Directive Lines" for information about the compiler directive lines.

This section explains the format and function of compiler options.

Each of the system components may be affected by the options.

The following table shows the relation between a file suffix and some compiler options.

File suffix	Option
.f	-X03 -Fixed (*1)
.for	
.F	
.FOR	
.f90	-X9 -Free
.f95	
.F90	
.F95	
.f03	-X03 -Free
.F03	

*1: If the -Xd7 option is specified, activates -X7 -Fixed options instead.

Compiler Options

Table 2.1 [Options for preprocessor]

Function overview	Option name
The -Ccpp option specifies to use the C preprocessor.	-Ccpp
The -Cfpp option specifies to use the Fortran preprocessor.	-Cfpp
The -Cpp option specifies to use preprocessor.	-Cpp
The -D option specifies to replace name by value.	-D*name*[=*tokens*]
The -P option specifies to make preprocessor store results in the current directory.	-P
The -E option specifies to perform only preprocessing to the specified source file. The result of preprocessing is output to the standard-output.	-E
The -U option specifies to invalidate the definition of name.	-U*name*

Table 2.2 [Options for source program]

Function overview	Option name
The -AU option specifies to treat characters in case sensitive manner.	-AU
The -Fixed option specifies to compile a Fortran source program as a fixed-form source.	-Fixed

Function overview	Option name
The -Free option specifies to compile a Fortran source program as a free-form source.	-Free
The -Fwide option specifies to compile a Fortran source program as a fixed-form source whose length of all lines is 255.	-Fwide
The -Xd7 option specifies that -X7 option is the default of language-version option when the suffix of the Fortran source file is .f, .for, .F or .FOR.	-Xd7

Table 2.3 [Options for Fortran standards and language specifications]

Function overview	Option name
The -AE option specifies not to treat special characters such as backslash (\) as control-characters.	-AE
The -Ae option specifies to treat special characters such as backslash (\) as control-characters.	-Ae
The -AT option indicates that it does not apply implicit typing and specifies the null mapping for all the letters.	-AT
The -Ap option specifies to treat the default access permission of modules as PRIVATE.	-Ap
The -Aw option specifies to compile IACHAR, ACHAR, IBITS and ISHFTC functions as intrinsic procedures.	-Aw
The -Ay option specifies to assume type of an unsigned real constant on the right side of an assignment statement as type of a variable on the left side.	-Ay
The -Az option specifies to append a null character (\0) to the end of each string constant argument of an external procedure.	-Az
These options specify whether or not to enable external name which appears only in EXTERNAL statements.	-N{allextput | noallextput}
These options specify whether or not to process the allocatable assignment of the Fortran 2003 standard.	-N{alloc_assign | noalloc_assign}
These options specify whether or not to enable COARRAY specifications.	-N{ coarray | nocoarray }
These options specify whether or not to copy constant actual argument of procedure.	-N{copyarg | nocopyarg}
These options specify whether or not to process the character assignment statements in Fortran standard.	-N{f90move | nof90move}
These options specify whether or not to assume MALLOC and FREE as intrinsic procedures.	-N{mallocfree | nomallocfree}
These options specify whether or not to compile particular functions as intrinsic functions.	-N{obsfun | noobsfun}
These options control an initial state for a list item with the ALLOCATABLE attribute in a private clause of OpenMP.	-N{ privatealloc | noprivatealloc }
These options specify whether or not to add RECURSIVE keyword in each SUBROUTINE and FUNCTION statement.	-N{recursive | norecursive}
These options specify whether or not to add SAVE statement in each program unit except main program.	-N{save | nosave}

Function overview	Option name
These options specify whether or not to automatically set zero value by the procedure entrance and ALLOCATE statement in each variable.	-N{ setvalue[=*set_arg*] | nosetvalue } *set_arg*: { { heap | noheap } | { stack | nostack } | { scalar | noscalar } | { array | noarray } | { struct | nostruct } }
These options specify version of the language.	-X{6 | 7 | 9 | 03}

Table 2.4 [Options for degree of accuracy]

Function overview	Option name
These options specify to improve the precision of the variables.	-A{d | i | q}
	-C{ cdII8 | cI4I8 | cdLL8 | cL4L8 | cdRR8 | cR4R8 | cd4d8 | ca4a8 | cdDR16 | cR8R16 }
The -C*l* option specifies to verify whether or not precision of rounding errors is in specified range.	-C*l* 0 <= *l* <= 15

Table 2.5 [Options for data allocation]

Function overview	Option name
The -AA option specifies not to align data in common blocks.	-AA
These options specify whether or not to free allocatable array.	-N{freealloc | nofreealloc}
These options specify whether or not to allocate data to write inhibit area.	-N{use_rodata | nouse_rodata}

Table 2.6 [Options for math functions]

Function overview	Option name
These options specify whether or not to inline intrinsic functions.	-K{ilfunc | noilfunc}
These options specify whether or not to use multi-operation function.	-K{mfunc[=*level*] | nomfunc} *level*: {1 | 2 | 3}
These options specify whether or not to create executable program which the Fortran intrinsic function library is linked in statically.	-K{ static_fjlib | nostatic_fjlib }
The -SSL2 option specifies to link SSL-II and sequential BLAS/ LAPACK libraries.	-SSL2
The -SSL2 option specifies to link SSL-II and parallelized BLAS/ LAPACK libraries.	-SSL2BLAMP

Table 2.7 [Options for performance and optimization]

Function overview	Option name
The -O option specifies the optimization level.	-O{ 0 | 1 | 2 | 3 }
The -x option specifies to apply inline expansion for user-defined procedures.	-x{ - | *proc_name*[,*proc_name*]... | *stmt_no* | *dat_sz*K | dir=*dir_name*[,dir=*dir_name*]... | 0 }
The -Kadr44 option specifies to create object files in which absolute address is expressed as 44-bit data.	-Kadr44
The -Kadr64 option specifies to create object files in which absolute address is expressed as 64-bit data.	-Kadr64
The -Knoalias option specifies to perform optimization under the assumption that pointers in the source code do not refer to the same memory area as	-Knoalias[=*spec*] *spec*: 's'

Function overview	Option name
other pointers or variables. 's' can be specified in *spec*.	
These options specify to pad arrays.	-Karraypad_const[=*N*] 1 <= *N* <= 2147483647
	-Karraypad_expr=*N* 1 <= *N* <= 2147483647
These options specify to merge arrays.	-Karray_merge
	-Karray_merge_common[=*name*]
	-Karray_merge_local
	-Karray_merge_local_size=*N* 2 <= *N* <= 2147483647
These options specify to shift dimensions of arrays.	-Karray_subscript
	-Karray_subscript element=*N* 2 <= *N* <= 2147483647
	-Karray_subscript_elementlast=*N* 2 <= *N* <= 2147483647
	-Karray_subscript_rank=*M* 2 <= *M* <= 30
	-Karray_subscript_variable='*ary_nm*(*rank*,*rank*[,*rank*]...)' 1 <= *rank* <=30
The -Karray_transform option is equivalent to -Karraypad_const,array_merge, array_subscript.	-Karray_transform
These options specify whether or not to allocate local variable on the stack.	-K{auto | noauto}
These options specify whether to allocate automatic data on the stack or on the heap.	-K{autoobjstack | noautoobjstack}
These options specify whether to allocate the result area of user-defined array function statically in the caller program unit or dynamically in execution.	-K{calleralloc[=*level*] | nocalleralloc} *level*: {1 | 2}
The -Kcommonpad option specifies to pad array in common blocks or module.	-Kcommonpad[=*N*] 4 <= *N* <= 2147483644
These options specify whether or not to assume that the particular data is aligned on the eight-byte boundary.	-K{dalign | nodalign}
These options specify whether or not to perform optimization that modifies how operations are evaluated for object programs.	-K{eval | noeval}
The -Kfast option specifies to create an object program which runs at high speed on the target machine.	-Kfast
These options specify whether or not to create an executable program which uses floating-point exception disable mode.	-K{ fed | nofed }

Function overview	Option name
These options specify whether or not the program may access floating point rounding mode, and also specify whether or not the rounding mode is the default.	-K{ fenv_access | nofenv_access }
These options specify whether or not to apply optimizations which use non-faulting load instructions.	-K{FLTLD | NOFLTLD}
These options direct whether or not to perform optimization using "Floating-Point Multiply-Add/ Subtract" floating point operation instructions on object programs.	-K{fp_contract | nofp_contract}
These options specify whether or not to convert a floating point division or a SQRT function into reciprocal approximation instructions.	-K{fp_relaxed | nofp_relaxed}
These options specify whether or not to apply optimizations which simplify floating-point operations.	-K{fsimple | nofsimple}
These options specify whether to generate the object file using instruction set for either HPC-ACE or HPC-ACE2.	-K{HPC_ACE | HPC_ACE2}
These options direct whether the optimization which uses the intent is performed in the program unit of procedure call which includes the actual argument associated with the dummy argument to which INTENT attribute is specified.	-K{intentopt | nointentopt}
These options specify whether or not to apply loop blocking optimization.	-K{loop_blocking[=*N*] | loop_noblocking} 2 <= *N* <= 10000
These options specify whether or not to perform loop fission: the optimization which divides a loop into plural loops.	-K{ loop_fission | loop_nofission }
These options specify whether or not to perform the loop fission optimization as much as possible before and after IF constructs in loops.	-K{ loop_fission_if | loop_nofission_if }
These options specify whether or not to fuse loops into one loop.	-K{loop_fusion | loop_nofusion}
These options specify whether or not to exchange loops.	-K{ loop_interchange | loop_nointerchange }
These options specify whether or not to perform optimization which divides loops and partially uses SIMD extensions.	-K{ loop_part_simd | loop_nopart_simd }
These options specify whether or not to perform optimization by the loop versioning.	-K{ loop_versioning | loop_noversioning }
These options specify whether or not to perform link time optimization.	-K{lto|nolto}
These options specify whether or not to apply optimizations which use the Non-Faulting mode.	-K{ nf| nonf }
These options specify whether or not to let FPU be nonstandard mode.	-K{ns | nons}

Function overview	Option name
These options specify whether or not to make OCLs (Optimization Control Lines) effective.	-K{ocl | noocl}
	-Kvppocl
These options specify whether or not to keep the frame pointer in a register from procedure calling.	-K{omitfp | noomitfp}
The -Koptions specifies to treat "!options" statements as compiler directive lines.	-Koptions
These options specify to generate position-independent code (PIC).	-K{pic | PIC}
These options specify whether or not to evaluate invariant expressions in advance.	-K{preex | nopreex}
These options specify how to generate prefetch instructions.	-Kprefetch_cache_level={1 | 2 | all}
	-K{prefetch_conditional | prefetch_noconditional}
	-K{prefetch_indirect | prefetch_noindirect}
	-K{prefetch_infer | prefetch_noinfer}
	-Kprefetch_iteration=*N* 1 <= *N* <= 10000
	-Kprefetch_iteration_L2=*N* 1 <= *N* <= 10000
	-Kprefetch_line=*M* 1 <= *M* <= 100
	-Kprefetch_line_L2=*M* 1 <= *M* <= 100
	-K{ prefetch_sequential[=*kind*] | prefetch_nosequential } *kind*: {auto | soft}
	-K{prefetch_stride | prefetch_nostride}
	-Knoprefetch
	-K{ prefetch_strong | prefetch_nostrong }
	-K{ prefetch_strong_L2 | prefetch_nostrong_L2 }
These options specify whether some optimizations for short loops are applied to all innermost loops in the program.	-K{shortloop=*N* | noshortloop} 2 <= *N* <= 10
These options specify whether or not to use SIMD extensions.	-K{simd[=*level*] | nosimd} *level*: {1 | 2 | auto}
These options specify to determine whether to use SIMD indirect instructions or scalar instructions.	-K{simd_separate_stride | simd_noseparate_stride}
These options specify whether or not to perform the striping optimization.	-K{ striping[=*N*] | nostriping } 2 <= *N* <= 100
These options specify whether or not to apply software pipelining.	-K{swp | noswp}
This option specifies to perform software pipelining for more loops by easing its applicable condition.	-Kswp_strong

Function overview	Option name
These options specify whether or not to allocate temporary array data object on the stack.	-K{temparraystack | notemparraystack}
These options specify whether or not to apply loop unrolling .	-K{unroll[=*N*] | nounroll} 2 <= *N* <= 100
These options specify whether or not to use SIMD extensions by UXSIMD optimization.	-K{uxsimd | nouxsimd}
These options specify whether or not to generate XFILL instructions.	-K{ XFILL[=*N*] | NOXFILL } 1 <= *N* <= 100

Table 2.8 [Options for thread parallelization]

Function overview	Option name
These options specify whether or not to localize arrays in loops.	-K{array_private | noarray_private}
These options specify whether or not to select a loop to be parallelized considering the number of threads when both an inner loop and an outer loop are candidates in a nested loop.	-K{dynamic_iteration | nodynamic_iteration}
The -Kindependent option specifies that the procedure named *proc_name* behave just the same regardless whether DO loops are parallelized or not.	-Kindependent=*proc_name*
The -Kinstance option specifies to create a parallelized object program executed in N threads.	-Kinstance=*N* 2 <= *N* <= 512
These options specify whether or not to perform automatic parallelization that requires dividing loops.	-K{ loop_part_parallel | loop_nopart_parallel }
These options specify whether or not to enable directives of OpenMP standard.	-K{openmp | noopenmp}
These options direct whether or not to promote optimization by assuming that array elements of the chunk size have no data dependency over iteration.	-K{ openmp_assume_norecurrence | openmp_noassume_norecurrence }
These options specify whether or not to fix the operation order of the reduction operations same as in the numerical order of the threads at the end of the region for which the reduction clause of OpenMP was specified.	-K{ openmp_ordered_reduction | openmp_noordered_reduction }
These options specify whether or not to allocate the threadprivate variable on the Thread-Local Storage.	-K{ openmp_tls | openmp_notls }
These options specify whether or not to apply automatic parallelization.	-K{parallel | noparallel}
These options specify whether or not to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads.	-K{ parallel_fp_precision | parallel_nofp_precision }
The -Kparallel_iteration option specifies that only the loops which are analyzed to iterate *N* times or more are targets of parallelization.	-Kparallel_iteration=*N* 1 <= *N* <= 2147483647
The -Kparallel_strong option specifies to parallelize all loops which can be parallelized automatically without estimating parallelization effects.	-Kparallel_strong
These options specify whether or not to apply reduction optimization.	-K{reduction | noreduction}

Function overview	Option name
These options specify whether or not to make the parallel region extended.	-K{region_extension | noregion_extension}
These options specify whether or not to create the thread-safe object program.	-K{threadsafe | nothreadsafe}
The -Kvisimpact option specifies to create object programs optimized for VISIMPACT (Virtual Single Processor by Integrated Multicore Parallel Architecture).	-Kvisimpact

Table 2.9 [Options for debug and tools]

Function overview	Option name
These options specify whether or not to output information for debug into object programs.	{ -g | -g0 }
These options specify to debug a serial program.	-H{a | e | f | o | s | u | x}
The -Ncheck_global option specifies to check following characteristics in program units during compilation. - The procedure characteristics between external procedure definitions and references. - The procedure characteristics between external procedure definitions and interface body. - Size of common blocks.	-Ncheck_global
The -Ncheck_intrfunc option specifies to perform intrinsic function error processing.	-Ncheck_intrfunc
These options specify whether or not to use the hook function called from a specified location.	-N{ hook_func | nohook_func }
These options specify whether or not to use the hook function called at regular time interval.	-N{ hook_time | nohook_time }
These options specify whether or not to output the statement number where the user defined procedure which causes an error is called and the statement number where an error occurred in the procedure.	-N{line | noline}
These options specify to debug Fortran source code.	-Eg
	-Nquickdbg [=*dbg_arg*] *dbg_arg*: {{ argchk | noargchk }| { subchk | nosubchk }| { undef | undefnan | noundef } | { inf_detail | inf_simple }}
These options specify whether or not to output the runtime information.	-N{ rt_tune | rt_notune }
	-Nrt_tune_func
	-Nrt_tune_loop[=*kind*] *kind*: {all|innermost}
These options specify whether or not to trap floating-point exceptions.	-N{Rtrap | Rnotrap}

Table 2.10 [Options for messages]

Function overview	Option name
The -f option specifies the lowest error level of diagnostic messages.	-f{i | w | s | *msg_num*}

Function overview	Option name
These options specify to check whether a source code conforms to a specific Fortran standard.	-v{03d | 03e | 03o | 03s}
The -Ec option specifies to output diagnostic messages when expressions may not be evaluated accurately.	-Ec
These options specify whether or not to output information of applied optimizations.	-K{optmsg[=*level*] | nooptmsg} *level*: {1 | 2}
These options specify whether to cancel the compilation if the compiler forecasts that it takes a long time to compile the program.	-N{ cancel_overtime_compilation | nocancel_overtime_compilation }
The -Ncheck_cache_arraysize option specifies to inspect the sizes of arrays during the compilation and issue a level i diagnostic message if the sizes could be a cause an impact to execution performance.	-Ncheck_cache_arraysize
These options specify whether or not to output the name of file and program unit which are currently processed.	-N{compdisp | nocompdisp}
The -Nmaxserious option specifies to stop compilation when serious errors are detected more than *maxnum* times.	-Nmaxserious=*maxnum*
These options specify to output information of compilation into .lst file.	-Nlst[={a | d | i | m | p | t | x}]
	-Nlst_out=*file*
	-Q[{a | d | i | m | o*file* | p | t | x}]

Table 2.11 [Options for linker]

Function overview	Option name
The -shared option specifies to create shared objects.	-shared
The -l option specifies to link lib*name*.so or lib*name*.a	-l*name*
These options specify whether or not to use largepage function.	-K{largepage | nolargepage}
The -L option specifies to search libraries from *directory*.	-L*directory*

Table 2.12 [Other options]

Function overview	Option name
The -c option specifies to suppress calling a linker.	-c
These options specify to change method to rename external procedures.	-ml={cdecl|frt}
	-mldefault={cdecl | frt}
The -mlcmain option specifies to select C main program name.	-mlcmain={main | MAIN__}
The -o option specifies to name output file *exe_file*.	-o*exe_file*
The -I option specifies to search include files and module information (.mod) files from *directory*.	-I*directory*
The -M option specifies to output module information (.mod) files to *directory*, or to search .mod files from *directory*.	-M*directory*
The -P option specifies to perform only preprocessing.	-P
The -S option specifies to output assembly source.	-S
The -V option specifies to output the version information of each tool.	-V
The -W option specifies to use *arg1*, a*rg2*, ... as options for *tool*.	-W*tool*,*arg1*[,*arg2*]...

Function overview	Option name
These options specify to output absolute path of tools and options used by frtpx command.	-#
	-###

Option List

[-c] [-f*msg_lvl*] [{ -g | -g0 }] [-l*name*] [-ml=*target*] [-mlcmain=*main_program*] [-mldefault=*target*]
[-o*exe_file*] [-shared] [-v*msg_arg*] [-x*inl_arg*]
[-A*obj_arg*] [-C*typ_arg*] [-D*name*[=*tokens*]] [-E] [-E*msg_arg*] [-Fixed] [-Free] [-Fwide]
[-H*dbg_arg*] [-I*directory*] [-K*opt_arg*] [-L*directory*] [-M*directory*]
[-N*src_arg*] [-O[*opt_lvl*]] [-P] [-Q[*lst_arg*]] [-S] [-SSL2] [-SSL2BLAMP]
[-U*name*] [-V] [-W*tool*,*arg1*[,*arg2*]...] [-X*lan_lvl*] [-#] [-###]

-c

The -c (compile only) option suppresses calling the linking phase. If the -c option is specified, object programs are created, but an executable program is not created. It is ignored if the -P and -S option is specified. If only object files are specified by file, it is meaningless to invoke the compile command with the -c option.

-f *msg_lvl* *msg_lvl*: { i | w | s | *msg_num* }

The -f option specifies the lowest error level of diagnostic messages output for Fortran compiler errors.

Each diagnostic message has a prefix of the form jwd*xxxxz-y* message, where *xxxx* is the message serial number, *z* is the message kind, and *y* is the message error level. The *y* can be either i (information message), w (warning message), s (serious error), or u (unrecoverable error). The -f option is the default.

See Section "4.1 Compilation Output" for information about diagnostic messages output during compilation.

If -f*msg_num* is specified, the particular error messages whose message number is *msg_lvl are* suppressed.

The *msg_num* argument can coexist with other arguments (including other *msg_num* arguments). To specify *msg_num* and other arguments for the -f option, separate the arguments by a comma (,).

Example:

```
$ frtpx -fw,1040,2005 a.f95
```

i

The -fi option specifies to output all levels of diagnostic messages.

w

The -fw option specifies to output diagnostic messages only for error levels w, s, and u.

s

The -fs option specifies to output diagnostic messages only for error levels s and u.

msg_num

The -f*msg_num* suppresses particular information and warning diagnostic messages specified by the *msg_num* argument. The *msg_num* argument is the four-digit message number; more than one *msg_num* may be specified. The *msg_num* argument can coexist with any other arguments, but is ignored when -fs is specified.

{ -g | -g0 }

These options direct whether to generate the debugging information used by the debugger.

-g

When this option is effective, the debugging information is generated in object programs.

The user can perform source-level debugging of programs compiled with this option.

When this option is effective, and the option which makes the -O1 or higher option effective is not specified, the -O0 option is effective.

When this option is effective, the -Karray_merge_common[=*name*], -Karray_merge_local, -Karray_subscript, and -Klto options are ignored.

Note the following restrictions when using debugger with program compiled with -O1 or higher option:

- Debugging of the variables is guaranteed at the entry of each procedure, and debugging of only the dummy arguments, variables declared in modules, and variables declared in common blocks is guaranteed. And note the following restrictions:

  - Debugging of the dummy arguments of which type is quadruple precision REAL, single precision COMPLEX, double precision COMPLEX, quadruple precision COMPLEX, derived type, or polymorphic is restricted.

  - Debugging of the dummy arguments of which type is CHARACTER whose length type parameter is not constant, array whose lower or upper bound is not constant, or assumed-shape array is restricted.

  - Debugging of the dummy arguments which are not referred in its procedure is restricted.

- Debugging of the local variables is not guaranteed.

- Debugging of the inlined procedures is restricted.

- The displaying of line number is not guaranteed.

Compile the program with the compiler option -O0 when using debugger without the restrictions above.

See "Debugger User's Guide" for the debugger specification.

-g0

This option invalidates the -g option.

-l*name*

The -l option specifies to search the library lib*name*.so or lib*name*.a. The position of this option within the command line is important. This is because libraries are searched for in the order in which the other libraries and object files appear within the command line. This option and its argument are passed to the linker.

-ml=*target* *target*: { cdecl | frt }

The -ml option specifies to change how to process external procedure name. cdecl or frt can be specified for *target*. See Section "11.3 Rules for Processing Fortran Procedure Names" for details.

-mlcmain=*main_program* *main_program*: { main | MAIN__ }

The -mlcmain option specifies C main program name. C main program name is a function name specified for *main_program*. main or MAIN__ can be specified for the function name. See Section "11.5.1 Passing Control First to a C Program" for details.

-mldefault=*target* *target*: { cdecl | frt }

The -mldefault option specifies to change how to process default external procedure names according to *target*. cdecl or frt can be specified for *target*. See Section "11.3 Rules for Processing Fortran Procedure Names" for details.

-o *exe_file*

The -o option specifies to name the output file *exe_file*. Unless either the -c or -S option is specified, the output file is the executable program. If the -c option is specified, the output file is the object program. If the -S option is specified, the output file is the assembly program.

-shared

The -shared option specifies to create a shared library. This option is passed for linker program.

-v *msg_arg* *msg_arg*: { 03d | 03e | 03o | 03s }

The -v option specifies to diagnose whether a source code conforms to a specific Fortran standard.

To specify plural arguments for the -v option, separate the arguments by a comma (,).

03d

The -v03d option specifies to issue diagnostic messages if the source code contains any deleted features in Fortran 2003.

03e

The -v03e option specifies to issue diagnostic messages if the source code contains any Fortran 2003 features not also in Fortran 95.

03o

The -v03o option specifies to issue diagnostic messages if the source code contains any obsolescent features in Fortran 2003.

03s

The -v03s option specifies to issue diagnostic messages if the source code contains any nonstandard features, other than decremental features in Fortran 2003.

-xinl_arg *inl_arg*: { - | *proc_name*[,*proc_name*]... | *stmt_no* | *dat_sz*K | dir=*dir_name*[,dir=*dir_name*]... | 0 }

The -x option specifies to apply inline expansion to user-defined procedures. Inline-expansion improves the execution performance as -K series options. Either of the -O1 option or higher must be specified together. To specify plural arguments for the -x option, separate the arguments by a comma.

The -x0 option is default.

This optimization increases the compilation time and memory requirements. The more optimizations are applied, the larger the object program size is. Specify the -Koptmsg=2 option to know whether or not this optimization is applied.

-

The -x- option specifies to choose the targets of inline expansion automatically from the user defined functions.

proc_name[,*proc_name*]...

The -x*proc_name*[,*proc_name*]... option specifies to apply inline expansion only to user-defined procedures which is specified by *proc_name* arguments. Specifying only the names of frequently called procedures can reduce compilation time and memory requirements significantly. To specify *proc_name* and other arguments (including other *proc_name* arguments) for the -x option, separate the arguments by a comma. If *stmt_no* or *dat_sz*K is specified in addition to *proc_name*, the compiler applies inline expansion for procedures which meet either condition.

To specify module procedure name as *proc_name*, the module name and a period must be specified immediately before without intervening blanks.

Similarly, to specify internal procedure name as *proc_name*, the host procedure name and a period must be specified immediately before without intervening blanks.

Moreover, if the host procedure is module procedure, the module name and a period must be specified immediately before the host name without intervening blanks.

Example: Specifying internal procedure name

```
$ frtpx -xsub.insub a.f90
```

stmt_no

The -x*stmt_no* option specifies to apply inline expansion for user-defined procedures having *stmt_no* or fewer executable statements. The argument *stmt_no* must be an integer value between 1 and 2147483647.

*dat_sz*K

The -x*dat_sz*K option specifies to apply inline expansion only to user-defined procedures whose all local arrays are up to *dat_sz*K. The argument *dat_sz* must be an integer value between 1 to 2147483. The letter K, represents kilobytes (not kibibytes), must be added after the value of *dat_sz*. The default *dat_sz* is unlimited.

The -x*dat_sz*K option applies inline expansion for user-defined procedures whose local array size is less than the specified *dat_sz*K value.

If inline expansion for user-defined procedures is applied, the generated object program includes the data referenced in the user-defined procedure. If this data includes a huge local array, the generated object program can be quite large. To control this, specify the -x *dat_sz*K option.

dir=*dir_name*[,dir=*dir_name*]...

The-xdir=*dir_name* option specifies to apply inline expansion for procedures defined in the file under the directory specified by the argument *dir_name* or referenced in the file currently compiled. However, these files are necessary to be the following conditions:

- These files whose suffix is .f, .for, .f90, .f95, or .f03. And,

- These files must not need to specify -Cpp option.

When the -xdir=*dir_name* option is specified, files under the *dir_name* directory are compiled temporarily by the compiler options which are enabled during the compilation of input file, and these are target of inline expansion.

To specify dir=*dir_name* and other arguments (including other dir=*dir_name* arguments) for the -x option, separate the arguments by a comma.

In that case, the other sub options are effective on the all files under the specified directory.

Specifying the -xdir=*dir_name* option, needs plenty time and large work area for compilation.

0

The -x0 option specifies not to apply inline expansion.

-A *obj_arg* *obj_arg*: { A | E | e | T | U | d | i | p | q | w | y | z }

The arguments of the -A option are A, E, e, T, U, d, i, m, p, q, w, y and z. These arguments can be specified at the same time.

A

The -AA option specifies not to align data and derived type in common blocks. See Section "5.2 Correct Data Boundaries" for details.

E

The -AE option specifies not to treat special characters such as backslash (\) character as control-characters. The -AE option is default.

e

The -Ae option specifies to treat special characters such as backslash (\) as control-characters.

T

The -AT option does not apply implicit typing and specifies the null mapping for all the letters. This does not affect any IMPLICIT statements.

U

The -AU option indicates that names are to be interpreted in a case-sensitive manner. Be careful of the following:

- This causes the program to be interpreted in a nonstandard manner. In ISO standard Fortran, letters are not case sensitive.
- References to intrinsic module entities, service subroutines, service functions or the external procedure names of the multi-operation functions must be in lowercase letters.
- The spelling of all declarations and references to an intrinsic procedure must be the same.
- In the IMPLICIT statement, the lowercase letters specified in a letter specifier list are equivalent to the corresponding uppercase letters.
- When you debug using the debugger, user-defined procedure names are case sensitive and variable names are not case sensitive.

d

The -Ad option improves constants, variables, and functions (intrinsic, statement, external, module and internal functions) of default real and default complex types to double precision. See Section "5.3.1 Precision Improving" for information about this function.

The -CcR8R16, -CcR4R8, -CcdRR8, -CcdDR16, -Ccd4d8, and -Cca4a8 options cannot be used with the -Ad option.

i

The -Ai option changes the default for integer type data from four-byte to two-byte. Two-byte integer type data will be used for:

- Integer constants with absolute values from 0 to 32767
- Data typed using the default implicit typing rules and having symbolic names beginning with I, J, K, L, M, or N
- Data without kind selector declared in an INTEGER statement
- Data resulting from intrinsic functions INT (without one-byte integer, four-byte integer, and eight-type integer argument), NINT, IDNINT, ICHAR, MAX1, MIN1, LEN, and INDEX

The -CcI4I8, -CcdII8, -Ccd4d8, and -Cca4a8 options cannot be used with the -Ai option.

p

The -Ap option specifies a default of private accessibility. It is same as explicitly specifies PRIVATE statement.

q

The -Aq option improves constants, variables, and functions (intrinsic, statement, external, module and internal functions) of double-precision real and double-precision complex types to quadruple precision. See Section "5.3.1 Precision Improving" for information about this function.

The -CcR8R16, -CcR4R8, -CcdRR8, -CcdDR16, -Ccd4d8, and -Cca4a8 options cannot be specified with this option.

w

The -Aw option causes the IACHAR, ACHAR, IBITS, and ISHFTC to be intrinsic procedures even if the -X6 or -X7 option is in effect.

y

The -Ay option improves the kind (precision) of an unsigned real literal constant on the right side of an assignment statement to the kind of the variable on the left side. For example,

```
REAL(8) A
A=1.23456789012345
```

The above program is equivalent to the following program if the -Ay option is specified.

```
REAL(8) A
A=1.23456789012345_8
```

z

The -Az option appends the null character (\0) to the end of each character argument passed to an external procedure. This allows a character string to be passed correctly to a C function, such as strlen. The length of the character string argument does not include the null character. For example, the result of following program is 1234 4 , even if the -Az option is specified.

```
CALL SUB('1234')
END
SUBROUTINE SUB(ARG)
CHARACTER(LEN=*) ARG
PRINT *,ARG , LEN(ARG)
END
```

-C *typ_arg* *typ_arg*: { *l* | cpp | fpp | pp | cdII8 | cI4I8 | cdLL8 | cL4L8 | cdRR8 | cR4R8 | cd4d8 | ca4a8 | cdDR16 | cR8R16 }

The -C option controls numerical accuracy and data alignment, the C preprocessor, the Fortran preprocessor, and the evaluation of data types. To specify more than one argument for the -C option, separate the arguments by a comma (,).

l 0 <= *l* <= 15

The *l* arguments of the -C option determine the effect of rounding errors on the results of floating-point data operations. The *l* must be a number between 0 to15. When the -C*l* option is specified, the right most *l* bits of the mantissa are set to zero each time a value is assigned to a default real, double-precision real, quadruple-precision real, default complex, double-precision complex, or quadruple-precision complex type variable by an assignment statement. (For complex types, these bits are set to zero in both the real and imaginary parts.) See Section "5.3.2 Precision Lowering and Analysis of Errors" for information about this function.

cpp

When it meets either of the following requirements, the -Ccpp option calls the C language preprocessor.

- The suffix of Fortran source file is F, FOR, F90, F95, or .F03.
- The -Cpp option is specified.

The C language preprocessor processes Fortran source as follows.

- The names are to be interpreted in a case-sensitive manner.

- The preprocessor ignores the comment for Fortran, the continuing line for Fortran, and the column number in fixed source form for Fortran.

If you want to know more detail, see "Appendix E Preprocessor".

fpp

When it meets either of the following requirements, the -Cfpp option calls the Fortran preprocessor.

- The suffix of Fortran source file is F, FOR, F90, F95, or .F03.
- The -Cpp option is specified.

The Fortran source is processed as follows.

- The names are to be interpreted in a case-sensitive manner.
- The comment for Fortran is given to priority more than the comment for C language.
- The preprocessor recognizes the comment for Fortran, the continuing line for Fortran, and the column number in fixed source form for Fortran.

If you want to know more detail, see "Appendix E Preprocessor".

pp

The -Cpp option calls the C language or Fortran preprocessor. The preprocessor of default is Fortran. If you want to use the C preprocessor, specify -Cpp and -Ccpp options.

cdII8

The -CcdII8 option interprets the default integer variables, constants, and functions as eight-byte integers.

If the -CcdII8 option is specified, INT, IFIX, IDINT, IQINT, NINT, IDNINT, and IQNINT specific intrinsic functions must not be used as an actual argument.

If the -CcdII8 option is specified, the specific intrinsic functions NINT and IDNINT must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The -Ai option cannot be used with this option. For example,

```
INTEGER     ::A
INTEGER(4) ::B
A=2
B=2_4
```

is evaluated as follows if the -CcdII8 option is specified.

```
INTEGER(8) ::A
INTEGER(4) ::B
A=2_8
B=2_4
```

cI4I8

The -CcI4I8 option interprets any four-byte integer type (whether default four-byte or explicitly declared four-byte) as eight-byte integer type. It applies to variables, constants, and functions.

If the -CcI4I8 option is specified, INT, IFIX, IDINT, IQINT, NINT, IDNINT, and IQNINT specific intrinsic functions must not be used as an actual argument.

If the -CcI4I8 option is specified, the specific intrinsic functions NINT and IDNINT must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The -Ai option cannot be used with this option. For example,

```
INTEGER     ::A
INTEGER(4) ::B
A=2
B=2_4
```

is evaluated as follows if the -CcI4I8 option is specified.

```
INTEGER(8) ::A
INTEGER(8) ::B
A=2_8
B=2_8
```

cdLL8

The -CcdLL8 option interprets default logical variables, constants, and functions as eight-byte logicals.

If the -CcdLL8 option is specified, the specific intrinsic function BTEST must not be used as an actual argument.

For example,

```
LOGICAL     ::A
LOGICAL(4) ::B
A=.TRUE.
B=.TRUE._4
```

is evaluated as follows if the -CcdLL8 option is specified.

```
LOGICAL(8) ::A
LOGICAL(4) ::B
A=.TRUE._8
B=.TRUE._4
```

cL4L8

The -CcL4L8 option interprets any four-byte logical type (whether default four-byte or explicitly declared four-byte) as an eight-byte logical type. It applies to variables, constants, and functions.

If the -CcL4L8 option is specified, the specific intrinsic function BTEST must not be used as an actual argument. For example,

```
LOGICAL     ::A
LOGICAL(4) ::B
A=.TRUE.
B=.TRUE._4
```

is evaluated as follows if the -CcL4L8 option is specified.

```
LOGICAL(8) ::A
LOGICAL(8) ::B
A=.TRUE._8
B=.TRUE._8
```

cdRR8

The -CcdRR8 option interprets the default real type as double-precision real type and interprets the default complex type as double-precision complex type. It applies to variables, constants, and functions.

If the -CcdRR8 option is specified, the specific intrinsic functions REAL, FLOAT, SNGL, and SNGLQ must not be used as an actual argument.

The -Ad and -Aq option cannot be used with this option. For example,

```
REAL         ::A
REAL(4)     ::B
COMPLEX     ::C
COMPLEX(4) ::D
A=2.0
B=2.0_4
C=(2.0,2.0)
D=(2.0_4,2.0_4)
```

is evaluated as follows if the -CcdRR8 option is specified.

```
REAL(8)     ::A
REAL(4)     ::B
COMPLEX(8) ::C
COMPLEX(4) ::D
A=2.0_8
B=2.0_4
C=(2.0_8,2.0_8)
D=(2.0_4,2.0_4)
```

cR4R8

The -CcR4R8 option is equivalent to specifying the -Ad option. It interprets any default real or complex type (whether default four-byte or explicitly declared four byte) as double-precision real or complex type. It applies to variables, constants, and functions.

If the -CcR4R8 option is specified, the specific intrinsic functions REAL, FLOAT, SNGL, and SNGLQ must not be used as an actual argument.

The -Ad and -Aq option cannot be used with this option. For example,

```
REAL         ::A
REAL(4)     ::B
COMPLEX     ::C
COMPLEX(4) ::D
A=2.0
B=2.0_4
C=(2.0,2.0)
D=(2.0_4,2.0_4)
```

is evaluated as follows if the -CcR4R8 option is specified.

```
REAL(8)     ::A
REAL(8)     ::B
COMPLEX(8) ::C
COMPLEX(8) ::D
A=2.0_8
B=2.0_8
C=(2.0_8,2.0_8)
D=(2.0_8,2.0_8)
```

cd4d8

The -Ccd4d8 option is equivalent to specifying all of the -CcdII8, -CcdLL8, and -CcdRR8 options. For example,

```
INTEGER     ::A
INTEGER(4) ::B
LOGICAL     ::C
LOGICAL(4) ::D
REAL         ::E
REAL(4)     ::F
COMPLEX     ::G
COMPLEX(4) ::H
A=2
B=2_4
C=.TRUE.
D=.TRUE._4
E=2.0
F=2.0_4
G=(2.0,2.0)
H=(2.0_4,2.0_4)
```

is evaluated as follows if the -Ccd4d8 option is specified.

```
INTEGER(8) ::A
INTEGER(4) ::B
```

```
LOGICAL(8) ::C
LOGICAL(4) ::D
REAL(8)     ::E
REAL(4)     ::F
COMPLEX(8) ::G
COMPLEX(4) ::H
A=2_8
B=2_4
C=.TRUE._8
D=.TRUE._4
E=2.0_8
F=2.0_4
G=(2.0_8,2.0_8)
H=(2.0_4,2.0_4)
```

ca4a8

The -Cca4a8 option is equivalent to specifying all of the -CcI4I8, -CcL4L8, and -CcR4R8 options. For example,

```
INTEGER     ::A
INTEGER(4) ::B
LOGICAL     ::C
LOGICAL(4) ::D
REAL         ::E
REAL(4)     ::F
COMPLEX     ::G
COMPLEX(4) ::H
A=2
B=2_4
C=.TRUE.
D=.TRUE._4
E=2.0
F=2.0_4
G=(2.0,2.0)
H=(2.0_4,2.0_4)
```

is evaluated as follows if the -Cca4a8 option is specified.

```
INTEGER(8) ::A
INTEGER(8) ::B
LOGICAL(8) ::C
LOGICAL(8) ::D
REAL(8)     ::E
REAL(8)     ::F
COMPLEX(8) ::G
COMPLEX(8) ::H
A=2_8
B=2_8
C=.TRUE._8
D=.TRUE._8
E=2.0_8
F=2.0_8
G=(2.0_8,2.0_8)
H=(2.0_8,2.0_8)
```

cdDR16

The -CcdDR16 option interprets the double-precision real type as quadruple-precision real type. It applies to variables, constants, and functions.

If the -CcdDR16 option is specified, the specific intrinsic functions DFLOAT, DBLE, DBLEQ, DREAL, and DPROD must not be used as an actual argument.

If the -CcdDR16 option is specified, the specific intrinsic function DPROD must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The -Ad, and -Aq option cannot be used with this option. For example,

```
DOUBLE PRECISION ::A
REAL(8)            ::B
A=2.0D0
B=2.0_8
```

is evaluated as follows if the -CcdDR16 option is specified.

```
REAL(16) ::A
REAL(8)   ::B
A=2.0_16
B=2.0_8
```

cR8R16

The -CcR8R16 option is equivalent to specifying the -Aq option. It interprets any double-precision real type as quadruple-precision type and interprets any double-precision complex type as quadruple-precision complex type. It applies to variables, constants, and functions.

If the -CcR8R16 option is specified, the specific intrinsic functions DFLOAT, DBLE, DBLEQ, DREAL, and DPROD must not be used as an actual argument.

If the -CcR8R16 option is specified, the specific intrinsic function DPROD must not be used as right side expression of pointer assignment statement and component of the structure constructor.

The -Ad, and -Aq option cannot be used with this option. For example,

```
DOUBLE PRECISION ::A
REAL(8)            ::B
COMPLEX(8)         ::C
A=2.0D0
B=2.0_8
C=(2.0_8,2.0_8)
```

is evaluated as follows if the -CcR8R16 option is specified.

```
REAL(16)     ::A
REAL(16)     ::B
COMPLEX(16) ::C
A=2.0_16
B=2.0_16
C=(2.0_16,2.0_16)
```

-D *name*[=*tokens*]

The -D option specifies to define name as #define directives do. If *tokens* is omitted, name is defined to be 1.

If both -D*name* and -U*name* are specified, the -U*name* option invalidate -D*name* option regardless of the order. Even if -D is not specified, the following macros are defined by default.

Macro	Value	Note
__FUJITSU	1	
__arch64__	1	
__unix	1	
__sparc	1	
__sparcv9	1	
__linux	1	
unix	1	
sparc	1	

Macro	Value	Note
linux	1	
__frt_version	700	
_OPENMP	201107	This macro is defined when the -Kopenmp option is effective

-E

The -E option specifies to perform only preprocessing to the specified source file.

The result of preprocessing is output to the standard-output.

The -Cpp option must be specified together unless a suffix of the source file is .F, .FOR, .F90, .F95 or .F03.

The -P option cannot be specified together.

-E *msg_arg* *msg_arg*: { c | g }

The -E option tells the Fortran compiler to output diagnostic messages about operations for which round off may affect the results.

The arguments of the -E option are c and g. These arguments may be combined, e.g., -Ecg.

c

The -Ec option specifies to output diagnostic messages when expressions may not be evaluated accurately. During compilation, this option also outputs level i diagnostic messages for arithmetic IF statements with real expressions.

See Section "4.1.1 Compilation Diagnostic Messages" for information about diagnostic messages output during compilation. See Section "6.3.2 Real Data Operations" for information about relational operations.

g

This function increases the compilation time. When the -Eg option is specified, the -Ncheck_global, -Haefosux and -O0 options are in effect.

-Fixed

The -Fixed option specifies to compile a Fortran source program as a fixed-form source.

-Free

The -Free option specifies to compile a Fortran source program as a free-form source.

-Fwide

The -Fwide option specifies to compile a Fortran source program as a fixed-form source whose length of all lines is 255.

-H *dbg_arg* *dbg_arg*: { a | e | f | o | s | u | x }

The -H option specifies the level of run-time error checking. During compilation some errors can be detected in Fortran object programs and error messages are generated. Program execution automatically detects some errors, and the arguments a, e, f, o, s, u and x can be specified to broaden the range of errors checked. Arguments a, e, f, o, s, u and x can be combined. These arguments can be specified at the same time as -Haefosu.

The -Eg option is supported to check the program error, too. It checks characteristics of a procedure between procedure definition and reference, between procedure definition and interface body, and size of common block during compilation.

If the -H option is specified, the default optimization is -O0. The -O option may be specified after the -H option to make the -O option effective. See Section "8.2 Debugging Functions" for information about the debug function.

a

The -Ha option compares the numbers, types, attributes, and sizes of actual and dummy arguments. The types of function results are also checked during execution. If a mismatch occurs during execution, the corresponding diagnostic message is output. See Section "8.2.1.1 Checking Argument Validity (ARGCHK)" for information about the debug function.

e

The -He options checks shape conformance when an array assignment statement is executed. If a mismatch of shape conformance occurs during execution, the corresponding diagnostic message is output. See Section "8.2.1.4 Shape Conformance Check (SHAPECHK)" for information about the debug function.

f

The -Hf option checks whether the file is connected with two devices or more at the same time in the input/output statement, and whether the input/output statement is called in addition is inspected by the function while executing the input/output statement.

If a mismatch occurs during execution, the corresponding diagnostic message is output. See Section "8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)" for information about the debug function.

o

The -Ho option checks the follows:

- Two dummy arguments are overlapped and the part of overlap is changed.
- When an assumed-size array with INTENT(OUT) attribute is referenced during execution, checks to see if the variable is defined.
- When a variable with SAVE attribute is referenced during execution, checks to see if the variable is defined.

If a mismatch occurs during execution, the corresponding diagnostic message is output. For information about the debug function, see Section "8.2.1.6 Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP)".

When the -Ho option is specified, -Ha option and -Hu option are in effect. See Section "8.2.1.1 Checking Argument Validity (ARGCHK)" for information about the -Ha debug function. See Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)" for information about the -Hu debug function.

s

When an array section, array element, or substring is referenced during execution, the -Hs option checks the value of each subscript or substring expression to see if it is within the declared range. The same check is done during compilation when possible. If the value is not within the declared range, a diagnostic message is output. See Section "8.2.1.2 Checking Subscript and Substring Values (SUBCHK)" for information about the debug function.

u

When a variable outside a common block and a module is referenced during execution, the -Hu option checks to see if the variable is defined, If the referenced variable is undefined, a diagnostic message is output. See Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)" for information about the u argument.

x

The -Hx option checks to see if the variable declared in a module or in a common block is defined when the variable is referenced during execution, to see if the pointer is associated when a pointer is referenced during execution, and to see if the incrementation parameter of the DO construct, stride of the FORALL, incrementation parameter of array constructor implied do control and stride of subscript-triplet is not 0.

If the -Hx option is specified, the -Hu option is in effect.

Be careful to the following when the -Hx option using:

- All program units that have a definition or initialization for common block object shall be compiled with -Hx option.
- All program units that uses the module shall be compiled with -Hx option, if the module is compiled with -Hx option.

See Section "8.2.1.5 Extended Checking of UNDEF (EXTCHK)" for information about the x argument.

-I *directory*

The -I option specifies directory names to be searched for files specified in an INCLUDE line or for module information (.mod) files.

More than one -I option may be specified. If multiple -I options are specified, the path names are searched in the order specified.

INCLUDE file retrieval is performed in the following order:

1. The directory of the files containing the INCLUDE lines
2. The directory in which frtpx is being executed
3. Directories specified as the argument of an -I option

The compilation information of a module is searched in the following order:

1. Directories specified as the argument of a -M option

2. The directory in which frtpx is being executed

3. Directories specified as the argument of an -I option

-K*opt_arg*

opt_arg: { adr44 | adr64 | noalias[=*spec*] | arraypad_const[=*N*] | arraypad_expr=*N* | array_merge | array_merge_common[=*name*] | array_merge_local | array_merge_local_size=*N* | { array_private | noarray_private } | array_subscript | array_subscript_element=*N* | array_subscript_elementlast=*N* | array_subscript_rank=*N* | array_subscript_variable='*ary_nm*(*rank*,*rank*[,*rank*]...)' | array_transform | { auto | noauto } | { autoobjstack | noautoobjstack } | { calleralloc[=*level*] | nocalleralloc } | commonpad[=*N*] | dalign | nodalign } | { dynamic_iteration | nodynamic_iteration } | { eval | noeval } | fast | { fed | nofed } | { fenv_access | nofenv_access } | { FLTLD | NOFLTLD } | { fp_contract | nofp_contract } | { fp_relaxed | nofp_relaxed } | { fsimple | nofsimple } | { HPC_ACE | HPC_ACE2 } | { ilfunc | noilfunc } | independent=proc_name | instance=*N* | { intentopt| nointentopt } | { largepage | nolargepage } | { loop_blocking[=*N*]} | loop_noblocking } | { loop_fission | loop_nofission } | { loop_fission_if | loop_nofission_if } | { loop_fusion | loop_nofusion } | { loop_interchange | loop_nointerchange } | { loop_part_parallel | loop_nopart_parallel } | { loop_part_simd | loop_nopart_simd } | { loop_versioning | loop_noversioning } | { lto | nolto } | { mfunc[=*level*] | nomfunc } | { nf | nonf } | { ns | nons } | { ocl | noocl } | { omitfp | noomitfp } | { openmp | noopenmp } | {openmp_assume_norecurrence | openmp_noassume_norecurrence } | { openmp_ordered_reduction | openmp_noordered_reduction } | { openmp_tls | openmp_notls } | options | { optmsg[=*level*] | nooptmsg } | { parallel | noparallel } | { parallel_fp_precision | parallel_nofp_precision } | parallel_iteration=*N* | parallel_strong | { pic | PIC } | { preex | nopreex } | noprefetch | prefetch_cache_level=*N* | { prefetch_conditional prefetch_noconditional } | { prefetch_indirect | prefetch_noindirect } | { prefetch_infer | refetch_noinfer } | prefetch_iteration=*N* | prefetch_iteration_L2=*N* | prefetch_line=*N* | prefetch_line_L2=*N* | { prefetch_sequential[=*kind*] | prefetch_nosequential } | { prefetch_sequential | prefetch_nosequential } | { prefetch_stride | refetch_nostride } | { prefetch_strong | prefetch_nostrong } | { prefetch_strong_L2 | prefetch_nostrong_L2 } | { reduction | noreduction } | { region_extension | noregion_extension } | { shortloop=*N* | noshortloop } | { simd[=*level*] | nosimd } | { simd_separate_stride | simd_noseparate_stride } | { static_fjlib | nostatic_fjlib } | { striping[=*N*] | nostriping } | { swp | noswp } | swp_strong | { temparraystack | notemparraystack } | { threadsafe | nothreadsafe } | { unroll[=*N*] | nounroll } | { uxsimd | nouxsimd } | visimpact | vppocl | { XFILL[=*N*] | NOXFILL } }

The -K options controls various aspects of optimization in order to improve execution performance of object programs. It is specified using arguments, each controlling a different area of optimization. Multiple arguments can be specified, separated with commas.

adr44

The -Kadr44 option specifies to create object files in which absolute address is expressed as 44-bit data.

adr64

The -Kadr64 option specified to create object files in which absolute address is expressed as 64-bit data.

noalias[=*spec*]

The -Knoalias directs the compiler to perform optimization under the assumption that different pointers or target variables in the source code do not assign values to the same memory area. 's' can be specified in *spec*.

When 's' is specified for *spec*, the compiler performs optimization only on variables with pointer attributes that meet Fortran standards. When -Knoalias is specified, the compiler performs optimization and automatic parallelization under the above conditions, promoting optimization and automatic parallelization of pointer variables. Note that the optimizations may not be promoted when the association status of the pointer variables is changed in the loop.

If this option is specified, and the pointers in the source program do not meet the above conditions, the executable program may output an incorrect result.

arraypad_const[=*N*] 1 <= *N* <= 2147483647

If the first dimension of an array has explicit bounds and the bounds expression is constant, -Karraypad_const directs to execute padding to align *N* elements. When *N* is not specified, the compiler will determine the padding volume for each array. Padding is the act of creating spaces within arrays.

This cannot be specified at the same time as the -Karraypad_expr=*N*.

See Section "9.12 Effects of Applying Optimization to Change Shape of Array" for points to note when specifying this option.

arraypad_expr=*N* 1 <= *N* <= 2147483647

Regardless of whether the bounds expression is a constant, -Karraypad_expr directs to executes padding to align *N* elements in arrays where the first dimension has explicit bounds.

This cannot be specified at the same time as the -Karraypad_const[=*N*] option.

See Section "9.12 Effects of Applying Optimization to Change Shape of Array" for points to note when specifying this option.

array_merge

The -Karray_merge option is equivalent to specifying the -Karray_merge_local and -Karray_merge_common options.

See Section "9.15 Merging Array Variables" for points to note when specifying this option.

array_merge_common[=*name*]

The -Karray_merge_common option directs that multiple arrays within common blocks are to be merged. If the -Ncheck_global, -Nquickdbg=argchk,-Nquickdbg=undef, -Nquickdbg=undefnan, -Nquickdbg=subchk, -g or -H option is set, -Karray_merge_common[=*name*] option will be ignored. Specify the common block name in the *name*.

If this *name* is omitted, the operation will target all arrays within named common blocks.

array_merge_local

The -Karray_merge_local option specifies to whether multiple local arrays are to be merged. If the -Ncheck_global, -Nquickdbg=argchk,-Nquickdbg=undef, -Nquickdbg=undefnan, -Nquickdbg=subchk, -g or -H option is set, the -Karray_merge_local option will be invalid. -Karray_merge_local_size=1000000 will also take effect at the same time.

array_merge_local_size=*N* 2 <= *N* <= 2147483647

The -Karray_merge_local_size option directs that the size of the local arrays to be merged are *N* bytes or greater. The -Karray_merge_local option must be specified together.

{ array_private | noarray_private }

These options direct whether or not to localize arrays in loops. This option is effective when -Kparallel option is effective. The default is -Knoarray_private.

array_private

The -Karray_private option directs to localize arrays in loops. The -Karray_private option is effective when -Kparallel option is effective.

noarray_private

The -Knoarray_private option directs not to localize arrays in loops.

array_subscript

The -Karray_subscript option causes an array dimension shift to be performed on:

- Allocatable arrays with 4 or more dimensions.
- Arrays with 4 or more dimensions that have 10 or fewer elements in the last dimension but more than 100 elements in the other dimensions.

See Section "9.14 The Dimension Shift of the Array Declaration" for points to note when specifying this option. If the -Ncheck_global, -Nquickdbg=argchk,-Nquickdbg=undef, -Nquickdbg=undefnan, -Nquickdbg=subchk, -g or -H option is set, the -Karray_subscript option will be invalid.

-Karray_subscript_element=100, -Karray_subscript_elementlast=10, and -Karray_subscript_rank=4 will take effect at the same time.

array_subscript_element=*N* 2 <= *N* <= 2147483647

The -Karray_subscript_element option directs that the number of elements other than the last dimension of the array (for which dimension shift is to be performed) is *N* or more. -Karray_subscript_element is effective only when -Karray_subscript is set. This option is not effective in allocatable arrays.

array_subscript_elementlast=*N* 2 <= *N* <= 2147483647

The -Karray_subscript_elementlast option directs that the number of elements of the last dimension of the array (for which dimension shift is to be performed) is *N* or fewer. -Karray_subscript_elementlast is effective only when the -Karray_subscript option is set. This option is not effective in allocatable arrays.

array_subscript_rank=*N* 2 <= *N* <= 30

The -Karray_subscript_rank option directs that the number of elements of the array (for which dimension is to be performed) is *N* or more. This is effective only when the -Karray_subscript option is set.

array_subscript_variable='*ary_nm*(*rank*,*rank*[,*rank*]...)' 1 <= *rank* <= 30

The -Karray_subscript_variable option specifies to perform dimension shift on the array specified by '*ary_nm*(*rank*,*rank*[,*rank*]...)'. The argument *ary_nm* is the name of array to optimize, and the argument "*rank*" is the position of the dimension to be shifted. The number of specified dimension positions must be the same as the dimension number of the argument *ary_nm*. The same position of the dimension cannot be specified two times or more. This is effective only when the -Karray_subscript option is set, and will invalidate the values specified in the -Karray_subscript_element=*N*, -Karray_subscript_elementlast=*N*, and -Karray_subscript_rank=*N* options.

array_transform

The -Karray_transform option is equivalent to specifying the -Karraypad_const, -Karray_merge, and -Karray_subscript.

{ auto | noauto }

These options direct whether to treat local variables (other than those with the SAVE attribute or initial values) as automatic variables. The automatic variables become undefined when the procedure ends. The default is -Knoauto.

auto

Local variables, other than those with the SAVE attribute or initial values, are treated as automatic variables and allocated to the stack. See Section "9.11 Effects of Allocating on Stack" for points to note when specifying this option.

noauto

Local variables, other than those with the SAVE attribute or initial values, are not treated as automatic variables.

{ autoobjstack | noautoobjstack }

These options specify whether to allocate automatic data on the stack or the heap. The default is -Knoautoobjstack.

autoobjstack

The -Kautoobjstack option directs that automatic data objects are allocated to the stack. See Section "9.11 Effects of Allocating on Stack" for points to note when specifying this option.

noautoobjstack

The -Knoautoobjstack option directs that automatic data objects are allocated to the heap.

{ calleralloc[=*level*] | nocalleralloc } *level*: { 1 | 2 }

These options direct whether the function result area is allocated by the caller or called, when referencing user-defined array function. The default is -Knocalleralloc.

Specify 1 or 2 for *level*.The default value is 1. Function results matching the conditions are allocated by the caller according to *level*. With function results that do not meet the condition, the area is allocated dynamically during execution of the function, and the caller deallocates the area after return.

Specifying this option may reduce overhead at execution caused by the dynamic allocation and deallocation of the function result area.

The same option must be specified for all files in which the array function is defined or referenced. If this option is specified differently or different *level*s are specified across the caller and called, the program may not work correctly.

calleralloc=1

When a user-defined array function is referenced, the caller of the function allocates the area for function results that meet the following conditions:

- The result variables of the function are primitive data types.
- The result variables of the function are explicit-shape arrays and all explicit bounds declarations are constant expressions.
- When the result variables of the function are character types, the character length parameter of those variables are constant expressions.

calleralloc=2

In addition to the conditions of calleralloc=1, the result area that meets the following conditions will be allocated by the caller of the function.

- -Karraypad_expr=*N* is not valid.

- The result variables of the function are primitive data types.
- When the result variables of the function are character types, the character length parameter of those variables are constant expressions.
- At the declaration of the function, all the explicit bounds declarations are constant expressions or intrinsic function SIZE references.
- The argument of the intrinsic function SIZE is a derived array name without an OPTIONAL attribute.
- The intrinsic function SIZE does not have a second argument.

nocalleralloc

The -Knocalleralloc option directs to allocate the area of the function result dynamically whenever the function is executed with user-defined array function references. It also deallocates the area at the caller after return.

commonpad[=*N*] 4 <= *N* <= 2147483644

The -Kcommonpad option specifies to pad array in common blocks or declaration part of module to use cache efficiently.

If the source file which includes the common block or module is compiled with the -Kcommonpad option, other source files which include the same common block or using module must be compiled with same -Kcommonpad. *N* is specified by bytes.

When *N* is omitted, the compiler determines automatically suitable value. When the common block or declaration part of module contain only array variables, the pads are effective.

{ dalign | nodalign }

These options direct whether to generate instructions assuming that eight-byte integer data, double-precision real data, quadruple-precision real data, double-precision complex data, or quadruple-precision complex data referred to by common blocks, dummy arguments, or pointers are aligned on eight-byte boundaries. The default is -Knodalign.

dalign

Instructions are generated assuming that the object program being generated is aligned on eight-byte boundaries. If -Kdalign is used, all the procedures comprising the executable program must be compiled with -Kdalign specified.

nodalign

Instructions are generated assuming that the object program being generated is not aligned on eight-byte boundaries.

{ dynamic_iteration | nodynamic_iteration }

These options direct whether to dynamically select a loop to be parallelized considering the number of threads when both the inner loop and the outer loop are candidates in a nested loop. This option is effective only when the -Kparallel option is set. The default is -Knodynamic_iteration.

dynamic_iteration

Loops are dynamically selected to be parallelized considering the number of threads when both the inner loop and the outer loop are candidates in a nested loop. The nest level for the loops is up to three.

nodynamic_iteration

Loops are not dynamically selected to be parallelized when both the inner loop and the outer loop are candidates in a nested loop.

{ eval | noeval }

These options direct whether to perform optimization that modifies how operations are evaluated for object programs. Note that performing this optimization may cause side effects in the execution result. The numerical operations do not conform to IEEE 754. The default is -Knoeval.

eval

The -Keval option modifies how operations are evaluated for object programs. When -Keval is set, -Kfsimple is valid. In addition, when this option and -Kparallel are set, -Kfsimple and -Kreduction are valid.

This option is effective only when the -O1 option or higher is set.

noeval

The -Knoeval option does not modify how operations are evaluated for object programs. When -Knoeval is set, -Knofsimple and -Knoreduction option are valid.

fast

The -Kfast option is for creating object programs to be executed at the highest speed on the target machine.

This is equivalent to specifying the -O3 -Kdalign,eval,fp_contract,fp_relaxed,ilfunc,mfunc,ns,omitfp,prefetch_conditional options. As -Kfast includes the -Keval, -Kfp_contract, -Kfp_relaxed, -Kilfunc, and -Kmfunc options, there may be side effects on the calculation results. The numerical operations do not conform to IEEE 754.

{ fed | nofed }

These options direct whether to create an executable program that uses floating-point exception disable mode.

These options must be set at linking. The default is -Knofed.

These options are valid only when the -KHPC_ACE2 option is set.

fed

An executable program that uses floating-point exception disable mode is created.

The program can be executed at high speed owing to disabling the floating-point exception.

The numerical operations do not conform to IEEE 754 and programs that handle floating-point exceptions run incorrectly. It is necessary to use this option carefully enough because all functions and libraries that are called from the executable program operate in floating-point exception disable mode. This option includes the function of the -Kns option regardless of whether to specify the -Kns or -Knons option.

The -Kfed option is valid only when the -O1 option or higher is set. If both the -Kfed option and the -NRtrap option or the -Nquickdbg=undefnan option are specified, the one specified last is valid.

nofed

An executable program that does not use floating-point exception disable mode is created.

{ fenv_access | nofenv_access }

These options direct whether or not the program may access floating point rounding mode and also direct whether or not the rounding mode is the default. Some floating point optimization may be suppressed when this option is valid.

The default is -Knofenv_access.

fenv_access

The -Kfenv_access specifies that the program can access floating point rounding mode or the rounding mode is not the default. Some floating point optimization may be suppressed.

nofenv_access

The -Knofenv_access specifies that the program does not access the floating point rounding mode and this directs that the rounding mode is the default.

{ FLTLD | NOFLTLD }

These options direct whether to perform optimization using non-faulting load instructions on object programs. The non-faulting load instruction is one of the load instructions which do not cause an exception whether or not the given address is correct. Using this instruction extends the range of optimization and will promote further optimization. The default is -KFLTLD.

These options are valid only when the -KHPC_ACE option is set.

FLTLD

The -KFLTLD option specifies not to perform optimizations using non-faulting-load instructions.

NOFLTLD

The -KNOFLTLD option specifies to perform optimizations using non-faulting-load instructions. -KNOFLTLD is effective only when the -O1 option or higher is set.

{ fp_contract | nofp_contract }

These options direct whether or not to perform optimization using "Floating-Point Multiply-Add/Subtract" floating point operation instructions on object programs.

Rounding errors may occur when these instructions are generated. The default is -Knofp_contract.

fp_contract

Optimization using Floating-Point Multiply-Add/Subtract floating point operation instructions is performed on object programs.

This option is effective only when the -O1 option or higher is set.

nofp_contract

Optimization using Floating-Point Multiply-Add/Subtract floating point operation instructions is not performed on object programs.

{ fp_relaxed | nofp_relaxed }

These options specify whether or not to perform optimizations using reciprocal approximation operation instructions on single-precision or double precision floating point divisions or SQRT functions. This optimization may introduce rounding errors, compared to when the normal division instructions or SQRT instructions are used. The numerical operations do not conform to IEEE 754. The default is -Knofp_relaxed.

fp_relaxed

The -Kfp_relaxed option directs that reciprocal approximation operation instructions and Floating-Point Multiply-Add/Subtract floating point operation instructions are used on single-precision or double precision floating point division or SQRT functions. -Kfp_relaxed is effective only when the -O1 option or higher is set.

nofp_relaxed

The -Knofp_relaxed option directs that normal division instructions or SQRT instructions are used for single-precision or double precision floating point division or SQRT functions.

{ fsimple | nofsimple }

These options direct whether to perform simplification of floating point operation on object programs. For example, operations such as "x*0" will be simplified to "0".The default is -Knofsimple. Using this optimization the numerical operations do not conform to IEEE 754.

fsimple

Simplification of floating point operation on object programs is performed. -Kfsimple is effective only when the -O1 option or higher is set.

nofsimple

Simplification of floating point operation on object programs is not performed.

{ HPC_ACE | HPC_ACE2 }

These options specify whether to generate the object file using instruction set for either HPC-ACE or HPC-ACE2. The -KHPC_ACE2 option is default.

HPC_ACE

-KHPC_ACE option specifies to generate the object file using HPC-ACE instruction set.

If this option is specified, note the following:

- The object file generated by this option cannot operate on FX10 system.

- If this option is specified to the main program, the dynamic library linked at runtime is the library for HPC-ACE. So, the execution performance may decrease.

- The SIMD instruction for HPC-ACE2 executes 4 operations at the same time. On the other hand, the SIMD instruction for HPC-ACE executes 2 operations at the same time, so the execution performance of the loop which SIMD optimization is applied to may decrease by this option.

HPC_ACE2

-KHPC_ACE2 option specifies to generate the object file using HPC-ACE2 instruction set.

{ ilfunc | noilfunc }

These options direct whether to perform inline expansion for intrinsic functions and operations. The compiler determines whether to perform inline expansion, based on this option, the data type, the code size, etc. Note that performing this optimization may cause rounding errors in the result. Further, if the argument of the intrinsic function corresponds to the "Cause of error" described in Section "8.1.5 Intrinsic Function Error Processing", error messages are not output and the execution results are not guaranteed. The default is -Knoilfunc. Using this optimization the numerical operations do not conform to IEEE 754.

The following shows the intrinsic functions and operations for which inline expansion can be performed.

- Intrinsic functions of the single-precision and double-precision real type: ATAN, ATAN2, COS, EXP, EXP10, LOG, LOG10, SIN, and TAN
- Exponentiations of the single-precision and double-precision real type (Both the base and exponent must be the same type)
- Intrinsic function of the single-precision complex type: EXP
- Intrinsic functions of the double-precision complex type: ABS and EXP

ilfunc

Inline expansion is performed for intrinsic functions and operations. -Kilfunc is effective only when the -O1 option or higher is set. In addition, reciprocal approximation operation and Floating-Point Multiply-Add/Subtract floating point operation may be used, even if -Knofp_contract and -Knofp_relaxed option are set.

noilfunc

Inline expansion is not performed for intrinsic functions and operations.

independent=*pgm_nm*

The -Kindependent option directs that the procedure reference specified in *pgm_nm* always performs as a sequential process, even if the procedure reference is specified within the parallelized DO-loop, and would be subject to automatic parallelization. -Kindependent is effective only when the -Kparallel option is set.

Procedure *pgm_nm* must be compiled with the -Kthreadsafe option specified. If the -Kthreadsafe option is not specified, the results of the execution cannot be guaranteed.

To specify module procedure name as *pgm_nm*, the module name and a period must be specified immediately before without intervening blanks.

Similarly, to specify internal procedure name as *pgm_nm*, the host procedure name and a period must be specified immediately before without intervening blanks.

For an internal procedure-name that is within module procedures, the module-name must also be specified.

Example: To specify the internal procedure-name "insub" that is within the parent procedure sub

```
$ frtpx -Kparallel,independent=sub.insub a.f90
```

This option shares points to note with the optimization indicator INDEPENDENT. See Section "12.2.3.3.2 Nested Parallel Processing" and Section "12.2.3.3.4 Notes on Using Optimization Control Line" for details.

instance=*N* 2 <= *N* <= 512

The -Kinstance=*N* option specifies the number of threads at execution. Specify a value between 2 and 512 for *N*. -Kinstance is effective only when the -Kparallel option is set. If the value of *N* differs from the number of threads at execution, execution will be aborted. See Section "12.2.3.3.1 Caution when specifying compile-time option -Kparallel,instance=N" for details.

{ intentopt | nointentopt }

These options direct whether the optimization which uses the intent is performed in the program unit including procedure call, which procedure call includes the actual argument associated with the dummy argument to which INTENT attribute is specified.

When the -O0 option is set, the default is -Knointentopt. When the -O1 option or higher is set, the default is -Kintentopt.

intentopt

The optimization which uses the intent is performed in the program unit including procedure call, which procedure call includes the actual argument associated with the dummy argument to which INTENT attribute is specified.

This option is effective only when the -O1 option or higher is set.

nointentopt

The optimization which uses the intent is not performed in the program unit including procedure call, which procedure call includes the actual argument associated with the dummy argument to which INTENT attribute is specified.

{ largepage | nolargepage }

These options specifies whether or not to create executable program where use large-page function. These options must be set at linking.

See Section "D.5 Large-Page Function" for information about large-page function. The -Klargepage option is default.

largepage

-Klargepage option specifies to create the executable program which the large-page function is applied.

nolargepage

-Knolargepage option specifies to create the executable program which the large-page function isn't applied.

{ loop_blocking[=*N*] | loop_noblocking } 2 <= *N* <= 10000

These options direct whether to perform the loop blocking optimization. Specify the block size between 2 and 10000 for *N*. If a value was not specified for *N*, the compiler will automatically determine a suitable value. The default is -Kloop_blocking, when the -O2 option or higher is set.

loop_blocking[=*N*]

Loop blocking is performed. -Kloop_blocking is valid only when the -O2 option or higher is set.

loop_noblocking

Loop blocking is not performed.

{ loop_fission | loop_nofission }

These options direct whether to perform loop fission:

- The optimization which divides a loop into plural loops.

The default is -Kloop_fission, when the -O2 option or higher is set.

loop_fission

Loop fission is performed. -Kloop_fission is valid only when the -O2 option or higher is set.

loop_nofission

Loop fission is not performed.

{ loop_fission_if | loop_nofission_if }

These options direct whether to perform loop fission optimization as much as possible before and after IF constructs in loops. The -Kloop_fission_if option is valid when the -Kloop_fission option is set. The -Kloop_nofission_if is default.

See Section "9.1.2.7 Loop Fission before and after IF constructs in loops(LFI)" for details.

loop_fission_if

The -Kloop_fission_if option specifies to perform the loop fission optimization as much as possible before and after IF constructs in loops.

loop_nofission_if

The -Kloop_nofission_if option specifies not to perform the loop fission optimization as much as possible before and after IF constructs in loops, but to perform normal loop fission.

{ loop_fusion | loop_nofusion }

These options direct whether to perform loop fusion, which combines adjacent loops. The default when the -O2 option or higher is set is -Kloop_fusion.

loop_fusion

Loop fusion is performed. -Kloop_fusion is valid only when the -O2 option or higher is set.

loop_nofusion

Loop fusion is not performed.

{ loop_interchange | loop_nointerchange }

These options specify whether or not to exchange loops.

When either the -O2 option or higher is specified, -Kloop_interchange option is effective and derived.

When the -Kloop_nointerchange option is specified, -Kloop_interchange option is ignored.

{ loop_part_parallel | loop_nopart_parallel }

When a loop contains statements to which parallelization can be applied and statements to which parallelization cannot be applied, the loop is divided into two or more loops and automatic parallelization is applied to one of the loops. This optimization is applied to the innermost loop. Note that compile time and execution time may increase. The -Kloop_part_parallel option is effective only when both the -Kloop_fission option and the -Kparallel option are set. The default is -Kloop_nopart_parallel.

loop_part_parallel

The automatic parallelization which needs dividing loops is performed.

loop_nopart_parallel

The automatic parallelization which needs dividing loops is not performed.

{ loop_part_simd | loop_nopart_simd }

When a loop contains instructions to which SIMD extensions can be applied and instructions to which SIMD extensions cannot be applied, the loop is divided into two or more loops and SIMD extensions are applied to one of the loops. This optimization is applied to the innermost loop. Note that compile time and execution time may increase. The -Kloop_part_simd option is effective only when both the -Kloop_fission option and the -Ksimd option are set. The default is -Kloop_nopart_simd option.

loop_part_simd

The optimization using SIMD extensions which needs dividing loops is performed.

loop_nopart_simd

The optimization using SIMD extensions which needs dividing loops is not performed.

{ loop_versioning | loop_noversioning }

These options specify whether to perform optimization by the loop versioning.

The loop versioning may promote optimizations such as SIMD, software pipelining, or automatic parallelization.

Note that the size of the object program and compile time may increase because the loop versioning generates two loops. Moreover, the execution performance of the program may decrease due to the overhead of judgement for the choice of generated loops.

These options are effective only when the -O2 option or higher is set. The default is -Kloop_noversioning.

For details about the loop versioning, see Section "9.1.2.8 Loop Versioning".

loop_versioning

The -Kloop_versioning option specifies to perform optimization by the loop versioning.

loop_noversioning

The -Kloop_noversioning option specifies not to perform optimization by the loop versioning.

{ lto | nolto }

These options direct whether to perform link time optimization. The default is -Knolto.

If the -g, -S or -Wa option is set, -Klto option will be invalid.

If -Klto option and -xdir=*dir_name* option are specified, -xdir=*dir_name* option will be invalid.

See Section "9.1.2.10 Link Time Optimization" for this function.

lto

Link time optimization is performed. -Klto option is effective only when the -O1 option or higher is set.

nolto

Link time optimization is not performed.

{ mfunc[=*level*] | nomfunc } *level*: { 1 | 2 | 3 }

These options direct whether to perform optimization by converting intrinsic functions and operations to multi-operation functions. "Multi-operation functions" are optimized intrinsic functions which perform same operations for multiple arguments at a single calling to improve performance. However, they may introduce potential rounding errors due to a difference in algorithms. This option is effective only when the optimization *level* -O2 option or higher is set. Specify 1, 2, or 3 for *level*. The default value for *level* is 1. The default is -Knomfunc.

The following shows the intrinsic functions and operations that can be converted.

Function/Operation name	Single precision real type	Double precision real type	Single precision complex type	Double precision complex type	Integer type
ACOS	o (*1)	o (*1)	o (*2)	o (*2)	-
ACOSH	x	x	o (*2)	o (*2)	-
ASIN	o (*1)	o (*1)	o (*2)	o (*2)	-
ASINH	x	x	o (*2)	o (*2)	-
ATAN	o	o	o (*2)	o (*2)	-
ATAN2	o	o	-	-	-
ATANH	x	x	o (*2)	o (*2)	-
COS	o	o	o (*2)	o (*2)	-
COSH	o (*2)	o (*2)	o (*2)	o (*2)	-
COSQ	x	x	o (*2)	o (*2)	-
ERF	o (*1)	o (*1)	-	-	-
ERFC	o (*1)	o (*1)	-	-	-
EXP	o	o	o (*2)	o	-
EXP10	o	o	-	-	-
LOG	o	o	o (*2)	o (*2)	-
LOG10	o	o	-	-	-
SIN	o	o	o (*2)	o (*2)	-
SINH	o (*2)	o (*2)	o (*2)	o (*2)	-
SINQ	x	x	o (*2)	o (*2)	-
TAN	x	x	o (*2)	o (*2)	-
TANH	o (*2)	o (*2)	o (*2)	o (*2)	-
exponentiation	o	o	o (*2)	o (*2)	o (*2)
ISHFT	-	-	-	-	o (*2)

o: To be target
x: Not to be target
-: No Intrinsic function

*1) These multi-operation functions execute sequential instructions internally. The execution performance improves because optimizations, such as branch optimization, are applied to the internal instructions. The improvement is inferior to that of the normal multi-operation functions.

*2) These multi-operation functions call the corresponding intrinsic functions four times internally. Although the performance of Multi-operation functions is the same as when calling the intrinsic functions four times, SIMD optimization to loops including intrinsic functions is promoted by using Multi-operation functions.

mfunc=1

The -Kmfunc=1 option specifies to use multi-operation functions with 4 multiplicities.

mfunc=2

In addition to the functions used by "-Kmfunc=1", this option specifies to use multi-operation functions with 8 or more multiplicities. This function requires a large stack area.

mfunc=3

In addition to "-Kmfunc=2", this option specifies to also use the multi-operation functions with 8 or more multiplicities for loops including IF constructs. Note that the performance will be worse than when "-Kmfunc=1" or "-Kmfunc=2" is specified, when the TRUE rate of the IF construct is low. Note that performing this optimization may cause side effects in the execution result. See Section "9.16.2 Effects of Compiler Option -Kmfunc=3" for information on the side effects. This function requires a large stack area.

nomfunc

Optimization by converting intrinsic functions and operations to multi-operation functions is not performed.

{ nf | nonf }

These options direct whether to perform optimization using the Non-Faulting mode on object programs. Using the Non-Faulting mode extends the range of speculative execution optimization, and optimization can be promoted.

Even if the given address is not correct, the load instruction in the Non-Faulting mode does not cause an exception.

When -Ksimd={2|auto} option and -Knf option are specified at the same time, the load instructions that are not speculative execution in the loop either might operate in the Non-Faulting mode. The default is -Knonf.

These options are valid only when the -KHPC_ACE2 option is set.

nf

Optimization using the Non-Faulting mode is done to the object program.

-Knf option is valid only when the -O1 option or higher is set.

nonf

Optimization using the Non-Faulting mode is not done to the object program.

{ ns | nons }

These options direct whether to initialize the FPU with non-standard floating-point mode. These options must be set at linking. The default is -Knons. When the -Kfed option is effective, the function of the -Kns option always becomes effective.

ns

The FPU is initialized with non-standard floating-point mode. When the FPU is initialized with non-standard floating-point mode, underflow value may become zero to get more performance. Moreover, the numerical operations do not conform to IEEE 754. -Kns is effective only when the -O1 option or higher is set.

nons

The FPU is not initialized with non-standard floating-point mode.

{ ocl | noocl }

These options direct whether to enable optimization control lines. See Section "9.10 Using Optimization control line (OCL)" for information on optimization control lines. The default is -Knoocl.

ocl

Optimization control lines are enabled. -Kocl is effective only when the -O1 option or higher is set.

noocl

Optimization control lines are disabled.

{ omitfp | noomitfp }

These options specify whether or not to keep the frame pointer in a register from procedure calling. The -Knoomitfp is default.

omitfp

When the -Komitfp option is effective, the frame pointer in a register is not kept. When this option is set, the trace back information is not kept. Either of the -O1 option or higher must be specified together.

noomitfp

When the -Knoomitfp option is effective, the frame pointer in a register is kept.

{ openmp | noopenmp }

These options direct whether to enable OpenMP directives. See Section "12.3 Parallelization by OpenMP Specifications" for information on this function. The default is -Knoopenmp.

openmp

The OpenMP directives are enabled. When the -Kopenmp option is set, the -Kthreadsafe and -Kauto options will also be set. However, the local variable of the main program will not be allocated to the stack even if the -Kauto option is set.

The -Knoauto and -Knothreadsafe options are valid if they are specified after the -Kopenmp option. Caution should be taken when the -Knoauto and -Knothreadsafe options are specified at the same time as the -Kopenmp option. (See Section "12.3 Parallelization by OpenMP Specifications").

Even when only object programs are specified as file names (that is, when only linking is to be performed), -Kopenmp must be specified if an object program that was compiled by with the -Kopenmp option specified is included.

noopenmp

OpenMP directives are treated as comments.

{ openmp_assume_norecurrence | openmp_noassume_norecurrence }

These options direct whether or not to promote optimizations by assuming that array elements of the chunk size have no data dependency over iteration for the loop with the OpenMP DO directive.

When -Kopenmp_assume_norecurrence is effective, SIMD extensions, software pipelining, and other optimizations are promoted.

These options are applied to the innermost loops with the OpenMP DO directive.

These options are effective only when the -Kopenmp option and -O2 option or higher is set. The default is -Kopenmp_noassume_norecurrence.

openmp_assume_norecurrence

This option directs to assume that array elements of the chunk size have no data dependency over iteration to promote optimizations.

openmp_noassume_norecurrence

This option directs not to assume that array elements of the chunk size have no data dependency over iteration.

{ openmp_ordered_reduction | openmp_noordered_reduction }

These options specify whether or not to fix the operation order of the reduction operations same as in the numerical order of the threads at the end of the region for which the reduction clause of OpenMP was specified. If the number of threads used is identical, by fixing the order of the reduction operations same as in the numerical order of the threads, identical results will be always obtained. However, rounding errors may occur when the operation order is changed by the effect of the scheduling of a loop construct with a dynamic or a guided schedule kind, or sections construct.

The -Kopenmp_noordered_reduction option is default.

openmp_ordered_reduction

The -Kopenmp_ordered_reduction option specifies to fix the operation order of the reduction operations same as in the numerical order of the threads at the end of the region for which the reduction clause was specified. Note that execution performance may decrease compared to when the -Kopenmp_ordered_reduction is invalidated. This option is valid only when the -Kopenmp option is in effect.

openmp_noordered_reduction

The -Kopenmp_noordered_reduction option specifies not to fix the operation order of the reduction operation at the end of the region for which the reduction clause was specified.

{ openmp_tls | openmp_tls }

These options specify whether or not to allocate the threadprivate variable on the Thread-Local Storage. The default is -Kopenmp_notls.

If the threadprivate variable is passed between Fortran and C/C++ programs, it is necessary to specify the -Kopenmp_tls option.

If the -Kopenmp_tls option is used, all the source programs comprising the executable program must be compiled with the -Kopenmp_tls option.

openmp_tls

The -Kopenmp_tls option directs that threadprivate variables are allocated to the Thread-Local Storage. This option is effective only when the -Kopenmp option is set.

openmp_notls

The -Kopenmp_notls option directs that threadprivate variables are allocated to the heap.

options

The -Koptions directs that lines starting with "!options" are treated as compiler directives. See Section "2.5 Compiler Directive Lines" for information on compiler directives.

{ optmsg[=*level*] | nooptmsg } *level*: { 1 | 2 }

These options direct whether to output messages about optimization status. Specify 1 or 2 for level. The default value is 1. The default is -Koptmsg=1.

optmsg=1

Messages are output indicating that the optimization performed may cause side effects in the execution results.

optmsg=2

Along with the messages from optmsg=1, messages are output indicating that optimization functions for automatic parallelization, SIMD optimization, and loop unrolling have performed.

nooptmsg

Output of optimization status messages is suppressed.

{ parallel | noparallel }

These options direct to performed automatic parallelization. However, if the effect of parallel execution is not expected, automatic parallelization is not performed. This option induces the -O2 and -Kregion_extension option. If the -O3 option is set, however, the optimization level will be 3. If the -H or -Eg option is set, the -Kparallel option will be ignored. The default is -Knoparallel.

See Section "12.2 Automatic Parallelization" for information on this option.

Even when only object programs are specified as file names (that is, when only linking is to be performed), this option must be specified if an object program that was compiled with the -Kparallel option specified is included.

parallel

The -Kparallel option directs that automatic parallelization is performed.

noparallel

The -Knoparallel option directs that automatic parallelization is not performed.

{ parallel_fp_precision | parallel_nofp_precision }

These options specify whether to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads. The default is -Kparallel_nofp_precision.

parallel_fp_precision

The -Kparallel_fp_precision option specifies to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads.

This option is effective when -Kparallel or -Kopenmp option is effective.

When this option and -Kopenmp option are set, -Kopenmp_ordered_reduction option is valid.

When this option is set, the execution performance may decrease because the part of optimization is restricted.

Note that the calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads even if this option is valid when the reduction clause of OpenMP is specified.

parallel_nofp_precision

The -Kparallel_nofp_precision option specifies not to avoid calculation error of a real type or a complex type operation that is caused by the difference of the parallel number of threads.

parallel_iteration=*N* 1 <= *N* <= 2147483647

The -Kparallel_iteration option directs that only loops determined at compilation to have a number of iterations exceeding *N* will be subject to parallelization. -Kparallel_iteration is effective only when the -Kparallel option is set.

Example :

```
REAL(KIND=4),DIMENSION(10000,5) :: A,B,C
DO J=1,5
   DO I=2,10000
     A(I,J) = B(I,J) + C(I,J)
   END DO
END DO
```

When -Kparallel_iteration=6 option is set, the outer loop will not be subject to parallelization because its iteration is 5.

parallel_strong

The -Kparallel_strong option directs to paralellize all loops for which it is detected that paralellization can be performed without estimating the effects of the paralellization. This option includes the -Keval and -Kpreex options.

Aside from this point, the functions and notes are the same as for the -Kparallel option.

{ pic | PIC }

These options direct to generate position-independent code (PIC).

If the -KPIC option is specified, a slower object with a longer instruction sequence is generated, however the number of unique external symbols that can be referenced when linking increases. If the -Kpic option is specified, a faster object with a shorter instruction sequence is generated, but the number of unique external symbols that can be referenced when linking decreases. In these cases, the number of unique external symbols is the total of all libraries that are referenced simultaneously.

If both -KPIC and -Kpic are specified, the one specified last is valid. This option is valid only when specified during compilation.

{ preex | nopreex }

These options direct whether to evaluate invariant expressions in advance. Note that this optimization may cause errors, as instructions that may not have been executed based on the logic of the program are likely to be executed. To identify whether or not this optimization was performed, see the diagnostic messages at compilation.

The default is -Knopreex.

preex

Invariant expressions are evaluated in advance. -Kpreex is effective only when the -O1 option or higher is set.

nopreex

Invariant expressions are not evaluated in advance.

noprefetch

The -Knoprefetch option directs that objects using prefetch instructions are not created.

prefetch_cache_level=*N* *N*: { 1 | 2 | all }

The -Kprefetch_cache_level option directs which cache level is to have data prefetched. -Kprefetch_cache_level is effective only when the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect option is set. Specify 1, 2, or all for *N*. The default is -Kprefetch_cache_level=all.

prefetch_cache_level=1

The -Kprefetch_cache_level=1 option directs to prefetch data in the first level cache. Normal prefetch instructions are used.

prefetch_cache_level=2

The -Kprefetch_cache_level=2 option directs to prefetch data only using the second level cache.

prefetch_cache_level=all

Functionality of prefetch_cache_level=1 and prefetch_cache_level=2 are valid simultaneously. Faster prefetch can be achieved by using the two types of prefetch instruction in combination.

{ prefetch_conditional | prefetch_noconditional }

These options specify whether or not to generate prefetch instructions for array data used in blocks in IF constructs or CASE constructs. This option requires that the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect option is set. The default is -Kprefetch_noconditional.

prefetch_conditional

Prefetch instructions are generated for array data used in blocks in IF constructs and CASE constructs.

prefetch_noconditional

Prefetch instructions are not generated for array data used in blocks in IF constructs and CASE constructs.

{ prefetch_indirect | prefetch_noindirect }

These options direct whether to create an object that uses prefetch instructions for array data that accessed indirectly (list access) within a loop. This option is effective only when the -O1 option or higher is set. The default is -Kprefetch_noindirect. Specifying this option may cause non-faulting load instructions to be created.

prefetch_indirect

Prefetch instructions are generated for array data accessed indirectly (list access) within a loop.

prefetch_noindirect

Prefetch instructions are not generated for array data accessed indirectly (list access) within a loop.

{ prefetch_infer | prefetch_noinfer }

These options direct whether to create prefetch instructions even if the prefetching distance is unknown. This option requires that the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect option is set. The default is -Kprefetch_noinfer.

prefetch_infer

Prefetch instructions for sequential access are created, even if the prefetching distance is unknown.

prefetch_noinfer

Prefetch instructions for sequential access are not created if the prefetching distance is unknown.

prefetch_iteration=*N* 1 <= *N* <= 10000

The -Kprefetch_iteration option specifies to prefetch data that is referenced or defined after *N* iterations of the loop.

As this option is only for prefetch instructions that prefetch to the first level cache, it is valid only when any option of the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=1 or the -Kprefetch_cache_level=all option is set.

This cannot be specified with the -Kprefetch_line=*N* option.

prefetch_iteration_L2=*N* 1 <= *N* <= 10000

The pretech_iteration_L2 option specifies to prefetch data that is referenced or defined after *N* iterations of the loop.

As this option is only for prefetch instructions that prefetch to the second level cache, it is valid only when any option of the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=2 or the -Kprefetch_cache_level=all option is set.

This cannot be specified at the same time as the -Kprefetch_line_L2=*N* option.

prefetch_line=*N*　　1 <= *N* <= 100

The -Kprefetch_line option specifies to prefetch data which are referenced or defined after *N* cache line(s).

As this option is only for prefetch instructions that prefetch to the first level cache, it is valid only when the -Kprefetch_sequential or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=1 or the -Kprefetch_cache_level=all option is set.

This cannot be specified at the same time as the -Kprefetch_iteration=*N* option.

prefetch_line_L2=*N*　　1 <= *N* <= 100

The -Kprefetch_line_L2 option specifies to prefetch data that is referenced to or defined in a line after *N* lines.

As this option is only for prefetch instructions that prefetch to the second level cache, it is valid only when the -Kprefetch_sequential or -Kprefetch_indirect option is valid, and, the -Kprefetch_cache_level=2 or the -Kprefetch_cache_level=all option is set.

This cannot be specified at the same time as the -Kprefetch_iteration_L2=*N* option.

{ prefetch_sequential[=*kind*] | prefetch_nosequential }　　*kind*: {auto|soft}

These options direct whether to generate prefetch instructions for array data accessed sequentially within a loop. Specify auto or soft for *kind*. The default value of *kind* is auto. The default when the -O0 or -O1 option is set is -Kprefetch_nosequential. The default when the -O2 option or higher is set is -Kprefetch_sequential.

prefetch_sequential=auto

The compiler automatically selects whether to use hardware-prefetch or to create prefetch instructions for array data that is accessed sequentially within a loop. -Kprefetch_sequential=auto is effective only when the -O1 option or higher is set.

prefetch_sequential=soft

The compiler does not use hardware-prefetch, but rather creates prefetch instructions for array data that is accessed sequentially within a loop. -Kprefetch_sequential=soft is effective only when the -O1 option or higher is set.

prefetch_nosequential

Prefetch instructions are not generated for array data that is accessed sequentially within a loop.

{ prefetch_stride | prefetch_nostride }

These options direct whether to generate prefetch instructions for array data that is accessed with a stride larger than the cache line size used in the loop. This option is effective only when the -O1 option or higher is set. The default is -Kprefetch_nostride.

prefetch_stride

Prefetch instructions are generated for array data that is accessed with a stride larger than the cache line size used in a loop. This includes loops with prefetch addresses not defined at compilation.

prefetch_nostride

Prefetch instructions are not generated for array data that is accessed with a stride larger than the cache line size used in a loop.

{ prefetch_strong | prefetch_nostrong }

These options direct whether to create strong prefetch instructions for the first level cache.

As this option is only for prefetch instructions that prefetch to the first level cache, it is valid only when any option of the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=1 or the -Kprefetch_cache_level=all option is set.

The default is -Kprefetch_nostrong.

prefetch_strong

Prefetch instructions created for the first dimension cache are strong prefetch.

prefetch_nostrong

Prefetch instructions created for the first dimension cache are not strong prefetch.

{ prefetch_strong_L2 | prefetch_nostrong_L2 }

These options direct whether to create strong prefetch instructions for the second level cache.

As this option is only for prefetch instructions that prefetch to the second level cache, it is valid only when any option of the -Kprefetch_sequential, -Kprefetch_stride, or -Kprefetch_indirect is set, and, the -Kprefetch_cache_level=2 or the -Kprefetch_cache_level=all option is set.

The default is -Kprefetch_strong_L2.

prefetch_strong_L2

Prefetch instructions created for the second level cache are strong prefetch.

prefetch_nostrong_L2

Prefetch instructions created for the second level cache are not strong prefetch.

{ reduction | noreduction }

These options direct whether to perform the reduction optimization. This option is effective only when the -Kparallel option is set. The default is -Knoreduction. Using this optimization the numerical operations do not conform to IEEE 754. See Section "12.2 Automatic Parallelization" and Section "12.2.3.1.8 Loop Reduction" for information about this function.

reduction

Reduction optimization is performed.

noreduction

Reduction optimization is not performed.

{ region_extension | noregion_extension }

These options specify whether or not to expand parallel-region to reduce parallelization overhead. This option is effective only when the -Kparallel option is set. See Section "12.2.3.1.11 Extension of Parallel Region" for information on this function. The default is -Knoregion_extension.

region_extension

Parallelization overhead is reduced by expanding parallelization.

noregion_extension

Parallelization is not expanded.

{ shortloop=*N* | noshortloop } 2 <= *N* <= 10

These options direct whether some optimizations for short loops are applied to all innermost loops in the program.

-Kshortloop=*N* specifies to modify some optimization functions at innermost loops assuming its iteration is few as the specified *N*. *N* can be specified from 2 to 10. The default is -Knoshortloop. -Kshortloop=*N* is valid only when the -O2 option or higher is set.

shortloop=*N*

-Kshortloop=*N* directs that some optimizations for short loops are applied to all innermost loops. -Kshortloop=*N* is valid only when the -O2 option or higher is set.

At the innermost loop with *N* iterations, -Kshortloop=*N* performs the followings:

- controlling the number of loop unrolling

- suppression of software pipelining

- suppression of loop blocking

- suppression of loop striping

- suppression of automatic parallelization

- suppression of XFILL instructions using

Note that when the -Kshortloop=*N* option is specified, performance may be reduced because of assumption that the iteration at the all innermost loops in the program is few.

noshortloop=*N*

-Knoshortloop directs that some optimizations are applied without assuming that the iteration at the all innermost loops is few.

{ simd[=*level*] | nosimd } *level*: { 1 | 2 | auto }

These options direct whether to create objects that use SIMD extension instructions. The -Ksimd option is effective only when the optimization level -O2 option or higher is set. Specify 1, 2, or auto for *level*. The default value for *level* is auto. The default when the -O2 option or higher is set is the -Ksimd option.

simd=1

Objects that use SIMD extension instructions are created.

simd=2

In addition to the functions of the simd=1 option, objects that use SIMD extension instructions are created for loops that include IF constructs. In order to redundantly execute instructions within IF constructs, performance may be reduced depending on the true ratio of the IF construct. In addition, as with the -Kpreex option, speculative executions are performed within an expression in IF constructs, so this optimization may execute an instruction which would not be executed based on the logic of the program, causing an error. (*1)

simd=auto

The compiler automatically determines whether to use SIMD extension for the loop. SIMD extension is promoted for loops that contain IF construct. (*1)

When the -KHPC_ACE option is set, -Ksimd=1 is assumed.

nosimd

Objects are created that do not use SIMD extension instructions.

*1) When the -Knf option is effective, it is possible to avoid the error by speculative execution of load instructions in IF construct. See the explanation of the -Knf option in this section for notes when specifying the -Knf option.

{ simd_separate_stride | simd_noseparate_stride }

When the stride width of memory access in the loop of SIMD-target is out of the range of 1 to 7, the compiler determines whether to use SIMD indirect instructions or scalar instructions at the memory access.

When the -O2 option or higher is effective and the -Ksimd option or the SIMD optimization control specifier is effective, the -Ksimd_separate_stride option is effective. The default is -Ksimd_noseparate_stride.

-Ksimd_separate_stride

When the stride width of memory access is out of the range of 1 to 7, the -Ksimd_separate_stride option specifies to use scalar instructions at the memory access.

-Ksimd_noseparate_stride

The -Ksimd_noseparate_stride option specifies to use SIMD indirect instructions.

{ static_fjlib | nostatic_fjlib }

These options specify whether or not to create executable program which the Fortran intrinsic function library is linked in statically. These options must be set at linking. The -Knostatic_fjlib option is default.

static_fjlib

-Kstatic_fjlib option specifies to create the executable program which the Fortran intrinsic function library is linked in statically. The size of executable program may increase when specifying this option.

nostatic_fjlib

-Knostatic_fjlib option specifies to create the executable program which the Fortran intrinsic function library is linked in dynamically.

{ striping[=*N*] | nostriping } 2 <= *N* <= 100

These options specify whether or not to perform the striping optimization. The striped length (number of expansions) can be specified for *N* as a value between 2 and 100. If *N* is omitted, the compiler will automatically determine the best value. When the iteration count of a loop in the source program is apparent, the expansion number determined by the compiler will be used even if a number that exceeds the iteration count is specified for *N*. The default is -Knostriping.

See Section "9.1.2.4 Loop Striping" for information on striping.

In striping, the statements in a loop are unrolled many times, so, as with loop unrolling, the size of the object module will increase, along with the required compilation time. Caution should be taken when using it, as performance may also be reduced due to an increased number of registers used.

striping[=*N*]

The striping optimization is performed. -Kstriping is effective only when the -O2 option or higher is set.

nostriping

The striping optimization is not performed.

{ swp | noswp }

These options direct whether to perform the software pipelining optimization. However, if the effect of software pipelining is not expected, software pipelining is not performed. When the -O0 or -O1 option is set, the default is -Knoswp. When the -O2 option or higher is set, the default is -Kswp. See Section "9.1.1.6 Software Pipelining" for information on software pipelining.

swp

Software pipelining is performed. -Kswp is effective only when the -O2 option or higher is set.

noswp

Software pipelining is not performed.

swp_strong

The -Kswp_strong option specifies to perform software pipelining for more loops by easing its applicable condition.

This option may increase compilation time and memory requirement significantly.

Aside from this point, the functions and notes are the same as for the -Kswp option.

{ temparraystack | notemparraystack }

These options direct whether to allocate the following results of the stack area. The default is -Knotemparraystack.

- Intermediate results of the array operation
- Mask expression evaluation result when iteration count of DO CONCURRENT is constant.

temparraystack

Intermediate results of the array operation and mask expression evaluation result are allocated to the stack area. See Section "9.11 Effects of Allocating on Stack" for points to note when specifying this option.

notemparraystack

Intermediate results of the array operation and mask expression evaluation result are not allocated to the stack area.

{ threadsafe | nothreadsafe }

These options direct whether or not to create thread-safe object programs. The default is -Knothreadsafe.

threadsafe

Thread-safe object programs are created.

nothreadsafe

Created object programs not thread-safe.

{ unroll[=*N*] | nounroll } 2 <= *N* <= 100

These options direct that the loop unrolling optimization is performed. Specify the upper bound for loop unrolling in *N*, as a value between 2 and 100. If a value was not specified for *N*, the compiler will automatically determine the best value. If the -O0 or -O1 option is set, the default is -Knounroll. If the -O2 option or higher is set, the default is -Kunroll.

unroll[=*N*]

Loop unrolling is performed. -Kunroll is effective only when the -O1 option or higher is set.

nounroll

Loop unrolling is not performed.

{ uxsimd | nouxsimd }

These options specify whether or not to use SIMD extensions by UXSIMD optimization. The -Ksimd option and the -KHPC_ACE option must be specified together. The -Knouxsimd option is default.

SIMD extensions are not used for intrinsic procedures. However, when the inline expansion is applied by optimization, the expanded instructions become the target of SIMD extensions.

See Section "9.1.2.6 UXSIMD" for information about UXSIMD.

uxsimd

The -Kuxsimd option specifies to use SIMD extension by UXSIMD optimization.

nouxsimd

The -Knouxsimd option invalidates -Kuxsimd option.

visimpact

The -Kvisimpact option directs that the most appropriate automatically parallelized object for VISIMPACT (Virtual Single Processor by Integrated Multicore Parallel Architecture) is created. This performs the same optimization as when the -Kfast,parallel option is specified. Using this optimization the numerical operations do not conform to IEEE 754.

vppocl

The -Kvppocl option directs that the optimization indicator NOVREC is treated equally to the optimization indicator NORECURRENCE.

This option is effective only when the -Kocl option is set.

{ XFILL[=*N*] | NOXFILL } 1 <= *N* <= 100

These options direct that for array data to be written only in a loop, an instruction is created that allocates a cache line for writing to the cache without loading data from the memory.

XFILL instructions targeting the cache lines specified in *N* are generated.

Specify a value between 1 and 100 for *N*. If a value is not specified for *N*, the compiler will automatically determine a value. This option is effective only when the -O2 option or higher is set. The default is -KNOXFILL. See Section "9.1.2.5 XFILL" for information on XFILL.

XFILL[=*N*]

XFILL instructions are output.

NOXFILL

XFILL instructions are not output.

-L *directory*

The -L option add directory to the list of directories in which the linker searches for libraries. This option and its argument are passed to the linker.

-M *directory*

The -M option specifies an alternate directory for module information files (.mod files).

.mod files are created in directory during compilation. For information about the compilation of modules, see "Chapter 10 Fortran Modules".

-N *src_arg*

src_arg: { { allextput | noallextput } | { alloc_assign | noalloc_assign } | { cancel_overtime_compilation | nocancel_overtime_compilation } | check_cache_arraysize | check_global | check_intrfunc | { coarray | nocoarray } | { compdisp | nocompdisp } | { copyarg | nocopyarg } | { f90move | nof90move } | { freealloc | nofreealloc } | { hook_func | nohook_func } | { hook_time | nohook_time } | { line | noline } | lst[=*lst_arg*] | lst_out=*file* | { mallocfree | nomallocfree } | maxserious=*maxnum* | { obsfun | noobsfun } | { privatealloc | noprivatealloc } | quickdbg[=*dbg_arg*] | { recursive | norecursive } | { rt_tune | rt_notune } | rt_tune_func | rt_tune_loop[=*kind*] | { save | nosave } | { setvalue[=*set_arg*] | nosetvalue } | { use_rodata | nouse_rodata } | { Rtrap | Rnotrap } }

{ allextput | noallextput }

The -Nallextput option creates an external name which appears only on EXTERNAL statement. The -Nallextput option is default.

The -Nnoallextput option does not create an external name which appears only on EXTERNAL statement.

{ alloc_assign | noalloc_assign }

The -Nalloc_assign is specified, the allocatable assignment (Refer to Section "6.4.2 Allocatable Assignment") is processed in Fortran 2003 standard when the -X9 or -X03 option is in effect. If the -X6 or -X7 is in effect, the -Nalloc_assign is not in effect. -Nnoalloc_assign is default.

alloc_assign

The -Nalloc_assign is specified, the allocatable assignment is processed in Fortran 2003 standard when the -X9 or -X03 option is in effect. The program has less effect on execution performance in Fortran 95 standard program when the -Nalloc_assign is specified.

noalloc_assign

The -Nnoalloc_assign is specified, the allocatable assignment is not processed in Fortran 2003 standard.

{ cancel_overtime_compilation | nocancel_overtime_compilation }

These options specify whether to cancel the compilation if the compiler forecasts that it takes a long time (24 hours or more as a guide) to compile the program.

The -Ncancel_overtime_compilation option is default.

cancel_overtime_compilation

The -Ncancel_overtime_compilation specifies to cancel the compilation if the compiler forecasts that it takes a long time to compile the program.

nocancel_overtime_compilation

The -Nnocancel_overtime_compilation specifies not to cancel the compilation even if it takes a long time to compile the program.

check_cache_arraysize

The -Ncheck_cache_arraysize options specifies to check the sizes of arrays during compilation and issue level-i diagnostic messages if the array sizes may cause execution performance decrease.

This function decides that an array could be a cause of a cache conflict if its size is a multiple of that of second level cache.

The array shape could be rewritten so as not to have a size of multiple of that of the cache in order to avoid the impact.

The function cannot judge correctly when the sector cache is active.

When the compiler option -Karraypad_const[=*N*] or -Karraypad_expr=*N* is specified for notes when this function is used, the size of the array becomes the size which adds the size of the padding.

See Section "4.1.1 Compilation Diagnostic Messages" for information about diagnostic messages output during compilation.

check_global

The -Ncheck_global option specifies to check the size of common blocks, and the procedure characteristics between external procedure definitions and references, and between external procedure definitions and interface body in program units during compilation.

If an error occurs, a diagnostic message is output. When -Ncheck_global is valid, compilation time may increase.

check_intrfunc

The -Ncheck_intrfunc option specifies to make the intrinsic function error processing effective for object programs.

If this option is in effect, error processing is performed when the object program is executed.

When it suits the error as the result of checking, when the following processing is executed:

- The output of the diagnostic message, traceback and error summaries
- user control of error by ERRSET and ERRSAV subroutine

- execution of user-defined subroutine

See Section "8.1.5 Intrinsic Function Error Processing" for details about the error number is detected by -Ncheck_intrfunc option and intrinsic function error processing.

{ coarray | nocoarray }

These options direct whether to enable COARRAY specifications. See the "Fortran User's Guide Additional Volume COARRAY" for information on this function. The default is -Nnocoarray.

coarray

The -Ncoarray option validates COARRAY specification.

Even when only object programs are specified as file names (that is, when only linking is performed), -Ncoarray must be specified if an object program that was compiled by with the -Ncoarray option specified is included.

nocoarray

The -Nnocoarray option invalidates COARRAY specification.

If you compile COARRAY programs with this option, error messages are output and the compilation stops.

{ compdisp | nocompdisp }

These options specify whether or not to output file name and program name which are currently compiled. The -Nnocompdisp option is default.

{ copyarg | nocopyarg }

The -Ncopyarg option specify to copy scalar constant argument to generated variable argument. When the -Ncopyarg is specified, even if the value of dummy argument is updated in procedure, it doesn't affected by the constant value of actual argument.

The program is not Fortran standard compliant. However, the program that can be operated by specifying this option does not conform to with the Fortran 95 standard.

Example:

```
CALL SUB(1)
PRINT *,1
END
SUBROUTINE SUB(I)
I=2
END
```

When -Ncopyarg is specified, this program outputs 1.

{ f90move | nof90move }

The -Nf90move option is specified, the character assignment statements are processed in Fortran standard even if the -X7 or -X6 option is in effect.

The -Nnof90move option is specified, the character assignment statements are not processed in Fortran standard.

If the -X9 or -X03 option is in effect, the -Nf90move is always in effect and the -Nnof90move option cannot be specified. If the -X7 or -X6 option is in effect, the -Nnof90move option is default.

Example:

```
CHARACTER(LEN=5) C
C='12345'
C(2:5)=C(1:4)
PRINT *,C ! output is 11234
END
```

When the -Nf90move option is specified, this program outputs "11234" even if the -X7 or -X6 option is in effect.

{ freealloc | nofreealloc }

The -Nfreealloc option specifies to deallocate arrays which do not have save attribute and are allocated when the procedure exits. The -Nfreealloc option is default.

If -Nnofreealloc option is specified and an unsaved allocatable array has a status of currently allocated when a procedure is exited, the array is not deallocated. This is not the behavior specified by the Fortran 95 standard.

{ hook_func | nohook_func }

These options specify whether or not to use the hook function that is called from a specified location.

See Section "8.2.3 Hook Function", for information about the hook function.

The default is -Nnohook_func.

This option must be set if an object program compiled with the -Nhook_func option set is included, even if only the object program name appears (that is, if only linking is performed).

hook_func

The -Nhook_func option is specified, a user-defined subroutine is called from the following locations:

- Program entry and exit

- Procedure entry and exit

- Parallel region (OpenMP or automatic parallelization) entry and exit

nohook_func

This option specifies not to use the hook function.

{ hook_time | nohook_time }

These options specify whether or not to use the hook function called at regular time interval.

Refer to Section "8.2.3 Hook Function", for information about the hook function.

The default is -Nnohook_time. This option must be set at linking.

hook_time

If the -Nhook_time option is enabled, a user-defined subroutine is called at regular time interval.

The time interval can be specified using the environment variable FLIB_HOOK_TIME. If the environment variable FLIB_HOOK_TIME is not specified, the user-defined subroutine is called once every minute.

Refer to Section "3.8 Using Environment Variables for Execution", for information about the environment variable FLIB_HOOK_TIME.

nohook_time

This option specifies not to use the hook function.

{ line | noline }

The -Nline option specifies to output the statement number where the user defined procedure which caused an error is called and the statement number where an error occurred in the procedure. These numbers are output into the trace back map. The -Nline option is default.

The -Nnoline option specifies that even if the error occurs in the procedure, the statement number is not displayed in the trace back map.

This information is output to standard error output when an error occurs during execution of Fortran programs. See Section "4.2.2 Trace Back Map for information about the trace back map.

lst[=*lst_arg*] *lst_arg*:{a|d|i|m|p|t|x}

The function of this option is equivalent to the -Q{a|d|i|m|p|t|x} option. For more details about the arguments, see -Q option.

Note that usage of comma to specify multiple arguments is different from the -Q option.

Each argument delimited by comma should be started from "lst=".

Example: "-Nlst=a,lst=d,lst=x"

lst_out=*file*

The function of this option is equivalent to the -Qo*file* option.

The compilation information is output to the *file*.

{ mallocfree | nomallocfree }

These options specify whether or not to evaluate MALLOC and FREE as intrinsic procedures. The -Nnomallocfree is default.

-Nmallocfree is specified to evaluate MALLOC and FREE as intrinsic procedures.

When -Nnomallocfree option is specified, evaluates MALLOC and FREE as service routines.

maxserious=*maxnum*

Stop the compilation if s-level (serious) errors are detected more than *maxnum* times. *maxnum* must be greater than or equal to 1. When this option is omitted, compilation does not stop even if s-level (serious) errors are detected.

{ obsfun | noobsfun }

The -Nobsfun option specifies that the under mentioned names are interpreted as intrinsic functions.

The -Nnoobsfun option specifies that the under mentioned names are not interpreted as intrinsic functions. The -Nnoobsfun option is default.

```
AIMAX0, AJMAX0, I2MAX0, IMAX0, JMAX0, IMAX1, JMAX1, AIMIN0, AJMIN0,
I2MIN0, IMIN0, JMIN0, IMIN1, JMIN1, FLOATI, FLOATJ, DFLOTI,
DFLOTJ,IIABS, JIABS, I2ABS, IIDIM, JIDIM, I2DIM,IIFIX, JIFIX, JFIX,
INT1, INT2, INT4, INT8, IINT, JINT,ININT, JNINT, IIDNNT, I2NINT,
JIDNNT, IIDINT, JIDINT,IMOD, JMOD, I2MOD, IISIGN, JISIGN,
I2SIGN,BITEST, BJTEST, IIBCLR, JIBCLR, IIBITS, JIBITS,

IIBSET, JIBSET,IBCHNG, ISHA, ISHC, ISHL,IIAND, JIAND, IIEOR, JIEOR,
IIOR, JIOR, INOT, JNOT,IISHFT, JISHFT, IISHFTC, JISHFTC,IZEXT, JZEXT,
IZEXT2, JZEXT2, JZEXT4,VAL
```

{ privatealloc | noprivatealloc }

For a list item with the ALLOCATABLE attribute in a private clause of OpenMP: if the list item is "currently allocated", the new list item will have an initial state of "not currently allocated". If the -Nprivatealloc option is specified, the program does not conform to the OpenMP 3.1 standard. The -Kopenmp option must be specified together.

When the -Nnoprivatealloc option is specified, the -Nprivatealloc option is ignored. The -Nnoprivatealloc option is default.

quickdbg[=*dbg_arg*] *dbg_arg*: {{ argchk | noargchk } | { subchk | nosubchk } | { undef | undefnan | noundef } | { inf_detail | inf_simple }}

This function is used to debug Fortran source code. If the debug function is enabled, it checks for Fortran source errors during compilation. It also embeds in object programs the information required for debugging and performs automatic checks during execution.

Checks are performed if the -Nquickdbg=argchk, -Nquickdbg=subchk, -Nquickdbg=undef, or -Nquickdbg=undefnan options are set.

For *dbg_arg*, either argchk, noargchk, subchk, nosubchk, undef, undefnan, noundef, inf_detail, or inf_simple can be specified. These options can be used in combination by specifying the -Nquickdbg[=*dbg_arg*] option multiple times.

The -Nquickdbg=inf_detail and -Nquickdbg=inf_simple options are enabled when set together with either the -Nquickdbg, -Nquickdbg=argchk, -Nquickdbg=subchk, -Nquickdbg=undef, or -Nquickdbg=undefnan option.

Omitting *dbg_arg* is equivalent to specifying -Nquickdbg=argchk,fastdbg=subchk,fastdbg=undef.

If the -Nquickdbg=argchk, -Nquickdbg=subchk, -Nquickdbg=undef, or the -Nquickdbg=undefnan option is enabled, the -Nquickdbg=inf_detail and -Ncheck_global options are also enabled.

See "8.2 Debugging Functions" for detail.

If the -H option is enabled, disable the -Nquickdbg option.

{ argchk | noargchk }

This option specifies whether or not to check the validity of the called and calling of procedure references.

This check is performed at compilation and at execution. See Section "8.2.1.1 Checking Argument Validity (ARGCHK)" for detail.

-Nquickdbg=argchk

Checks the validity of arguments and result of procedure. If an error is detected, a diagnostic message is output.

-Nquickdbg=noargchk

The validity of arguments and result of procedure are not checked.

{ subchk | nosubchk }

This option specifies whether or not the validity of the declared expression and referenced expression are checked in array section, array element, and substring declarations and references.

This check is performed at execution. See Section "8.2.1.2 Checking Subscript and Substring Values (SUBCHK)" for details.

-Nquickdbg=subchk

Checks the subscript and substring value. If an error is detected, a diagnostic message is output.

-Nquickdbg=nosubchk

The subscript and substring value are not checked.

{ undef | undefnan | noundef }

This option specifies whether or not to check that data is defined in references to module declaration parts and variables outside of common blocks.

This check is performed when the program is executed. See Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)" for details.

-Nquickdbg=undef

Checks undefined data references. If an error is detected, a diagnostic message is output.

-Nquickdbg=undefnan

Checks undefined data references. When this option is specified, real type and complex type errors are detected as invalid operation exceptions. When an error is detected, either a diagnostic message or an invalid operation exception message is output, depending on the variable type.

If both the -Nquickdbg=undefnan option and the -Kfed option are specified, the one specified last is valid.

-Nquickdbg=noundef

Undefined data reference checks are not performed.

{ inf_detail | inf_simple }

This option specifies the information to be included in diagnostic messages output when errors are detected.

See Section "8.2.1 Debugging Check Functions" for details.

-Nquickdbg=inf_detail

In addition to the message and line number where the error occurred, information is output such as the variable name, procedure name, and element location in order to identify the cause.

However, when -Nquickdbg=undefnan option is specified, for errors detected as invalid operation exceptions, only the error and the line number where it occurred are output in diagnostic messages.

-Nquickdbg=inf_simple

The error and the line number where it error occurred are output in diagnostic messages.

{ recursive | norecursive }

These options are specified, the RECURSIVE keyword is added in each SUBROUTINE and FUNCTION statement.

Example:

```
SUBROUTINE SUB
```

is evaluated as follows if the -Nrecursive option is specified.

```
RECURSIVE SUBROUTINE SUB
```

When -Nrecursive option is effective, the program does not conform to the Fortran 95 standard.

When -Nnorecursive option is specified, the RECURSIVE keyword is not added in SUBROUTINE and FUNCTION statements. The -Nnorecursive option is default.

{ rt_tune | rt_notune }

The -Nrt_tune option specifies to output the runtime information (parallelization information, cost information, input-output information and hardware monitor information).

The runtime information can be used for the tuning of user program. See "Runtime Information Output Function User's Guide" for details.

When the -Nrt_notune option is specified, the -Nrt_tune option is ignored. The -Nrt_notune option is default.

rt_tune_func

In addition to the -Nrt_tune output, runtime information about a user-defined functions is output.

This option induces the -Nrt_tune option. When the -Nrt_notune option is specified after -Nrt_tune_func option, -Nrt_tune_func option will be invalid.

See "Runtime Information Output Function User's Guide" for details.

rt_tune_loop[=*kind*] *kind*: { all | innermost }

In addition to the -Nrt_tune output, runtime information about loops is output.

All or innermost can be specified in the argument *kind*. If the *kind* is not specified, all is used.

This option induces the -Nrt_tune option. When the -Nrt_notune option is specified after -Nrt_tune_loop option, -Nrt_tune_loop option will be invalid.

See "Runtime Information Output Function User's Guide" for details.

rt_tune_loop=all

Runtime information about all loops is output.

rt_tune_loop=innermost

Runtime information about the innermost loops is output.

{ save | nosave }

When -Nsave option is specified, the SAVE statement without saved entity list is added in each program unit except main program. It is ignored if the -Nrecursive or -Kauto option is specified.

When -Nnosave option is specified, The SAVE statement without saved entity list is not added in program unit. The -Nnosave option is default.

{ setvalue[=*set_arg*] | nosetvalue }
set_arg: { { heap | noheap } | { stack | nostack } | { scalar | noscalar } | { array | noarray } | { struct | nostruct } }

The options specify whether or not to automatically set zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to stack or heap. The -Nnosetvalue is default.

Note that execution time may increase for the initialization.

If the -Eg, -H or -Nquickdbg option is specified, the -Nsetvalue option is ignored.

setvalue[=*set_arg*]

The -Nsetvalue option specifies to automatically set zero value in variables.

Either heap, noheap, stack, nostack, scalar, noscalar, array, noarray, struct or nostruct can be specified in the *set_arg*.

These options can be used in combination by specifying the -Nsetvalue=*set_arg* option multiple times.

If the *set_arg* is omitted, it is equivalent to the following options:

-Nsetvalue=heap,setvalue=stack,setvalue=scalar,setvalue=array,setvalue=struct

{ heap | noheap }

These options specify whether or not to automatically set zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to heap.

The zero value is not set to the allocated area by service subroutine MALLOC and intrinsic function MALLOC. The zero value is not set in the allocatable variable at head of the OpenMP directive block when the allocatable variable is specified by PRIVATE or LASTPRIVATE clause of OpenMP directive.

The value is set in the following variables:

- Automatic data objects when -Knoautoobjstack option is effective.
- Allocatable variables.
- Pointer variables.

setvalue=heap

The -Nsetvalue=heap option specifies to automatically set the zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to heap.

When the setvalue=heap is specified, the -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct options are in effect.

setvalue=noheap

The -Nsetvalue=noheap option does not apply to automatically set the zero value by the procedure entrance and ALLOCATE statement in variables that are allocated to heap.

{ stack | nostack }

These options specify whether or not to automatically set the zero value by the procedure entrance in variables that allocated to stack.

The value is set in the following variables:

- Automatic data objects when -Kautoobjstack option is effective.
- Local variables without SAVE attribute when -Kauto option is effective.
- Local variables in program unit that has RECURSIVE or PURE.
- Local variables that declare by AUTOMATIC attribute or AUTOMATIC statement.
- Local variables that specify by PRIVATE clause or LASTPRIVATE clause in OpenMP directive.

setvalue=stack

The -Nsetvalue=stack option specifies to automatically set the zero value by the procedure entrance in variables that are allocated to stack.

But, the zero value is set at head of the OpenMP directive block when the specifying variables by PRIVATE clause or LASTPRIVATE clause.

When the setvalue=stack is specified, the -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct options are in effect.

setvalue=nostack

The -Nsetvalue=nostack option does not apply to automatically set the zero value by the procedure entrance in variables that are allocated to stack.

{ scalar | noscalar }

These options specify whether or not to automatically set the zero value in scalar variables of numeric types and logical types.

setvalue=scalar

The -Nsetvalue=scalar option specifies to automatically set the zero value in scalar variables of numeric types and logical types.

setvalue=noscalar

The -Nsetvalue=noscalar option does not apply to automatically set the zero value in scalar variables of numeric types and logical types.

{ array | noarray }

These options specify whether or not to automatically set the zero value in array variables of numeric types and logical types.

setvalue=array

The -Nsetvalue=array option specifies to automatically set the zero value in array variables of numeric types and logical types.

setvalue=noarray

The -Nsetvalue=noarray option does not apply to automatically set the zero value in array variables of numeric types and logical types.

{ struct | nostruct }

These options specify whether or not to automatically set the zero value in scalar and array variables of derived type and character type.

setvalue=struct

The -Nsetvalue=struct option specifies to automatically set the zero value in scalar and array variables of derived type and character type.

The variable of derived type having length type parameter is not applied.

setvalue=nostruct

The -Nsetvalue=nostruct option does not apply to automatically set the zero value in scalar and array variables of derived type and character type.

nosetvalue

The -Nnosetvalue option does not apply to automatically set zero value in variables.

The option is equivalent to the following options:

-Nsetvalue=noheap,setvalue=nostack,setvalue=noscalar,setvalue=noarray,setvalue=nostruct

Be careful of the following:

1. When the -Nsetvalue option is in effect, it is necessary to be the following conditions:
   - The -Nsetvalue=heap or -Nsetvalue=stack is in effect, and
   - The -Nsetvalue=scalar, -Nsetvalue=array or -Nsetvalue=struct is in effect.
2. If the -Nsetvalue=heap or -Nsetvalue=stack is specified, -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct are generated. Therefore, when the -Nsetvalue=noscalar, -Nsetvalue=noarray or -Nsetvalue=nostruct will be in effect, the option should be specified after -Nsetvalue=heap or -Nsetvalue=stack.
3. When the zero value is set in only scalar variable of numeric types and logical types for stack, it is necessary to be the following options:
   -Nsetvalue=stack,setvalue=noarray,setvalue=nostruct

   When the -Nsetvalue=stack is specified, the -Nsetvalue=scalar, -Nsetvalue=array and -Nsetvalue=struct are generated. Therefore, if -Nsetvalue=stack,setvalue=scalar are specified, the option specifying is not limited to scalar variables alone.

{ use_rodata | nouse_rodata }

The -Nuse_rodata option is specified, string constant, floating point constant and initialization value of aggregate type local storage variable are allocated to read-only data section. The -Nuse_rodata option is default.

When the -Nnouse_rodata option is specified, the -Nuse_rodata option is ignored.

{ Rtrap | Rnotrap }

The -NRtrap option specifies to trap the intrinsic instructions errors and floating point exceptions in execution, and if the intrinsic instructions errors or floating point exceptions are trapped, the runtime messages are outputted. -NRnotrap option is the default. This option is not effective in an initial constant expression that is operated at compilation time.

Rtrap

If a main program unit is compiled with -NRtrap option, the intrinsic instructions errors (from jwe0259i-e to jwe0282i-e, jwe1397i-e, jwe1398i-e, jwe1413i-e, jwe1414i-e, jwe1416i-e, and jwe1417i-e) and the floating point exceptions (jwe0011i-u,

jwe0012i-u, jwe0013i-u and jwe0292i-u) are trapped and runtime messages are output. The floating-point underflow exception is trapped when the environment variable FLIB_EXCEPT=u have been specified.

If both the -NRtrap option and the -Kfed option are specified, the one specified last is valid.

If -NRtrap option is used together with -Kpreex option and -Ksimd=2 option, the speculative execution may cause exceptions that would not normally occur.

For the messages, see "Fortran/C/C++ Runtime Messages" for details.

Rnotrap

If a main program is compiled with -NRnotrap option, the intrinsic instructions errors and the floating point exceptions are not trapped and runtime messages are not output.

-O [*opt_lvl*] *opt_lvl*: { 0 | 1 | 2 | 3 }

The -O option specifies the optimization level used by the compiler.

The system provides four optimization levels: 0, 1, 2 and 3. If the argument is omitted, level 3 is used. If the -O option is not specified, level 2 is used.

See "Chapter 9 Optimization Functions" for information about the optimization of object programs.

In addition, the following optimization facilities can be specified:

- The advance evaluation of invariant expressions and the evaluation strategy can be changed by specifying -K series option
- It can indicate the inline expansion of user-defined procedure by specifying -x option.

0

The -O0 option creates an object program without applying optimizations. A program compiled with the -O0 option requires the least compilation time and memory. Specify this argument to debug compilation errors in Fortran source programs.

1

The -O1 option creates an object program by applying basic optimization. The run time of an executable program will be shorter and the size of it will be smaller if it is created with the -O1 option than if it is created with the -O0 option.

2

When the -O2 option is effective, following optimizations are applied in addition to the optimizations applied by the -O1 option.

- Loop unrolling
- Software pipelining
- Loop blocking
- Loop fusion
- Loop fission
- Exchange Loop (-Kloop_interchange)
- Using prefetch instructions(Equivalents to -Kprefetch_sequencial, prefetch_cachelevel=all)
- Using SIMD instructions
- Repeat basic optimization

The -O2 option repeats optimizations applied by the -O1 option until it admits of no optimization.

Note that compile time and object program size may be increase compared to the -O1 option. To determine whether each optimization was applied, specify the -Koptmsg=2 option.

3

When the -O3 option is effective, following optimizations are applied in addition to the optimizations applied by the -O2 option.

- Unrolling nested loops
- Splitting for promoting loop exchange

- Loop unswitching

Note that compile time and object program size may be increase compared to the -O2 option.

To determine whether each optimization was applied, specify the -Koptmsg=2 option.

-P

The -P option specifies to perform only preprocessing to the specified source file. The result of preprocessing is output to the temporary file.

The -Cpp option must be specified together unless a suffix of the source file is .F, .FOR, .F90, .F95 or .F03. The -E option cannot be specified together.

If this option is set, only preprocessor is performed. This option cannot be specified with -E.

The temporary files are generated are as follows:

Source file	Result file of preprocessing
file.F	*file*.cpp.f
file.f	
file.FOR	*file*.cpp.for
file.for	
file.F90	*file*.cpp.f90
file.f90	
file.F95	*file*.cpp.f95
file.f95	
file.F03	*file*.cpp.f03
file.f03	

-Q [*lst_arg*] *lst_arg*: { a | d | i | m | o*file* | p | t | x }

The -Q option outputs compilation information to a file with suffix .lst. If more than one Fortran source program files is specified, it is output to the first-file-name.lst.

The -Q option can be specified with arguments a, d, i, m, o*file*, p, t, and x. The arguments may be specified at the same time, separated by comma as in -Qa,o*file*,x. If arguments are omitted, the source program list and diagnostic error messages are output.

a

If the -Qa option is specified, the attributes of names are output in addition to the compilation information produced by the -Q option.

The function of this option is equivalent to the -Nlst=a option.

d

If the -Qd option is specified, the layout of derived types is output in addition to the compilation information produced by the -Q option.

The function of this option is equivalent to the -Nlst=d option.

i

If the -Qi option is specified, include files are listed in addition to the compilation information produced by the -Q option.

The function of this option is equivalent to the -Nlst=i option.

m

If the -Qm option is specified, the Fortran program which expresses the situation of automatic parallelization with OpenMP directives is output in addition to the compilation information produced by the -Qp option. Fortran program to be output is stored in the file having name that .omp is inserted before suffix (.f, .F, etc.) of an input file. It is ignored if the -Qi option is specified, or if -Kparallel option is not effective.

The function of this option is equivalent to the -Nlst=m option.

o*file*

The file name must be specified immediately after o. The compilation information is output to the file.

The function of this option is equivalent to the -Nlst_out=*file* option.

p

The -Qp option specifies optimization information to be output along with listing of -Q option. Optimization information shows the situation of automatic parallelization, inlining and unrolling.

The function of this option is equivalent to the -Nlst=p option.

t

If the -Qt option is specified, the details optimization information and the statistics information is output in addition to the compilation produced by the -Qp option.

The function of this option is equivalent to the -Nlst=t option.

x

If the -Qx option is specified, the cross reference list of name and label is output in addition to the compilation produced by the -Q option.

The function of this option is equivalent to the -Nlst=x option.

-S

The -S option suppresses creation of the corresponding object programs and does not call the linker. Instead, assembly programs (files with the suffix .s) are created. If only assembly source files or object files are specified by file, it is meaningless to invoke the compile command. If the -S option is effective, the -Klto option will be invalid.

{ -SSL2 | -SSL2BLAMP }

These options relate to linking with Fujitsu's math libraries (SSL II, BLAS, LAPACK). The details are described in "SSL II Online Documents", "SSL II Thread-Parallel Capabilities Online Documents" and "BLAS, LAPACK, ScaLAPACK Online Documents." Following is just an outline.

-SSL2

The whole set of routines from SSL II, SSL II Thread-Parallel Capabilities and BLAS/LAPACK becomes part of link libraries.

-SSL2BLAMP

The whole set of routines from SSL II, SSL II Thread-Parallel Capabilities and BLAS/LAPACK Thread-Parallel versions becomes part of link libraries (i.e. -SSL2BLAMP just replaces the sequential BLAS/LAPACK from -SSL2 with the corresponding thread-parallel versions.)

-U *name*

The -U option undefines name, which has the same effect as an #undef preprocessing directive. If the same name is specified for both -D and -U, name is not defined regardless of the order of the options.

-V

The -V option displays the version and release information on each Fortran compiler component.

-W *tool*,*arg1*[,*arg2*]...

Passes each argument *arg1*[,*arg2*]... as a separate argument to a tool. The arguments must be separated by commas (A comma can be part of an argument by using a backslash as an escape character before it. The backslash is removed from the resulting argument).

tool can be one of the following:

Value	Meaning
p	Preprocessor
0	Compiler
a	Assembler

Value	Meaning
l	Linker

For example, -Wa,-oobjfile passes -o and objfile to the assembler, in that order. Also, -Wl,-Iname causes the linking phase to override the default name of the dynamic linker. For other -W options, arguments passed to a tool may not be in order.

If the -Wa option is specified, the -Klto option will be invalid.

-X *lan_lvl lan_lvl*: { 6 | 7 | 9 | 03 | d7 }

The -X option directs the level of language specification. Either 6, 7, 9, 03 or d7 can be specified as the argument of the -X option.

The interpretation of Fortran source programs is different for each level of language specification. To compile Fortran source programs, indicate the level of language specification used by the Fortran compiler. See Section "6.1 Fortran Versions" for information on the difference between languages.

6

The -X6 option specifies to compile Fortran source programs as FORTRAN66 source.

7

The -X7 option specifies to compile Fortran source programs as FORTRAN77 source. If the suffix of the Fortran source file is .f or .F, -X7 is the default.

9

The -X9 option specifies to compile Fortran source programs as Fortran 95 or Fortran 90 source. Activates -Nf90move option at the same time. If the suffix of Fortran source file is .f90, .F90, .f95 or .F95, -X9 is the default.

03

The -X03 option specifies to compile Fortran source programs as Fortran 2003 source.

Activates -Nf90move option at the same time. If the suffix of Fortran source file is .f03 or .F03, -X03 is the default.

d7

If the version of the language is not specified, and the suffix of the Fortran source file is .f, .for, .F or .FOR, -Xd7 is the default.

-#

The -# option specifies to output the name of each path and options used by frtpx command, but does not execute any of the tools.

-###

The -### option specifies to output the name of each path and options used by frtpx command as it executes.

2.3 Compile Command Environment Variable

It explains the environment variables that the compile commands (frtpx command and frt command) recognize as follows.

FORT90C

This is the environment variable that the frt command (own compiler) recognizes.

The compiler options specified in the environment variable FORT90C are combined with the frt command option arguments. The frt command option arguments have precedence if there is a conflict with options specified in the environment variable FORT90C.

An example to use the environment variable FORT90C is shown in the following.

```
$ export FORT90C="-fw -I/usr/prv/usri"
$ frt a.f90
```

This is interpreted as

```
$ frt -fw -I/usr/prv/usri a.f90
```

It explains the environment variable that the frtpx command (cross compile command) recognizes as follows.

FORT90CPX

This is the environment variable that the frtpx command (corss compiler) recognizes.

The compiler options specified in the environment variable FORT90CPX are combined with the frtpx command option arguments. The frtpx command option arguments have precedence if there is a conflict with options specified in the environment variable FORT90CPX.

An example to use the environment variable FORT90CPX is shown in the following.

```
$ export FORT90CPX="-fw -I/usr/prv/usri"
$ frtpx a.f90
```

This is interpreted as

```
$ frtpx -fw -I/usr/prv/usri a.f90
```

TMPDIR

This is the environment variable that the frtpx command (corss compiler) and the frt command (own compiler) recognize.

The temporary directory used by the frtpx command or frt command can be changed by specifying a value for the environment variable TMPDIR.

If the environment variable TMPDIR is not set, /tmp is used. To avoid to output in common directory, set the environment variable TMPDIR to local directory.

An example to use the environment variable TMPDIR is shown in the following.

```
$ export TMPDIR=/usr/local/tmp
```

2.4 Compilation Profile File

The default of compilation options can be changed by specifying the compilation profile file (/etc/opt/FJSVmxlang/jwd_prof).

A compilation profile file must be used according to the following format:

- Blank characters not enclosed quotation mark (") or single quotation mark (') are not significant.
- Either single or double quotation marks can be used to begin and end character strings. A character string is terminated be the same delimiter with which the string began. A character string is also terminated by the end of a line.
- A symbol (#) that is not part of a character string starts a comment. Any characters from the # to the end of the line are treated as part of the comment.

The following is an example of a compilation profile file specification:

```
#Default options
-fW -Kocl
```

The priority of compilation options is as follows, highest first:

1. Compiler directive lines(the -Koptions is specified)
2. Compilation command operand
3. Environment variable FORT90CPX or FORT90C
4. Profile file
5. Default value

2.5 Compiler Directive Lines

If !options is specified the first in Fortran source program and compiler option -Koptions is effective, options in !options become effective.

Form of compiler directive lines:

```
!options opt[,opt] ....
```

opt: compiler options

List of effective compiler options:

-O[1-3]

-Keval/noeval/loop_blocking[=*N*]/loop_noblocking/loop_fusion/loop_nofusion/ilfunc/noilfunc/mfunc[=level]/nomfunc/ocl/noocl/preex/nopreex/noprefetch/prefetch_infer/prefetch_noinfer/prefetch_iteration=*N*/prefetch_iteration_L2=*N*/prefetch_line=*N*/prefetch_line_L2=*N*/striping[=*N*]/nostriping/unroll[=*N*]/nounroll/noalias[=*spec*]/vppocl

Example:

```
1---5----10---15---20---25
!options -O2 -Knounroll
    subroutine sub
```

Notes:

- !options cannot be specified the first in module procedure or internal subprogram.
- If compiler option -O3 effective, -O[1|2] option cannot be specified in compiler directive lines.

2.6 Return Values during Compilation

The following table lists the return values set by the frtpx(1) command.

Return value	Status
0	Normal termination
1	Compile or linker error
Other than the above	Linker error

Chapter 3 Executing Fortran Programs

This chapter describes the procedures for executing Fortran programs.

3.1 Execution Command

An execution command is used to execute the executable program file created by the frtpx compile command.

3.2 Execution Command Format

Runtime options and user-defined executable program options may be specified as command option arguments of an execution command. The runtime options use functions supported by the Fortran library.

See Section "3.3 Runtime Options", for information on the runtime option.

The format of runtime options is as follows:

```
exe_file [-Wl,-runtime option[,-runtime option]...] [-user-defined
          executable program option[,-user-defined executable program
          option]...]
```

Notes:

1. *exe_file* indicates the executable program file.
2. The GETPARM and GETARG service subroutines fetch user-defined executable program options other than runtime options.
3. If an option is specified more than once with different arguments, the last occurrence is used.

The following example shows how to specify the runtime option -Wl,-i and user-defined executable program options -K and -L as command option arguments of a command named a.out.

```
$ ./a.out -Wl,-i -K,-L
```

3.3 Runtime Options

Runtime options may be specified as arguments for execution commands or in the FORT90L environment variable. This section explains the format and functions of the runtime options.

The runtime option format is as follows:

-Wl[,-a] [,-d*num*] [,-e*num*] [,-g*num*] [,-i] [,-l*elvl*] [,-m*u_no*] [,-n] [,-p*u_no*] [,-q] [,-r*u_no*] [,-t*sec*] [,-u] [,-x] [,-C*u_no*] [,-G*u_no*] [,-Lb] [,-Li] [,-Lr] [,-Lu] [,-M] [,-Q] [,-Re] [,-Rp] [,-Ry] [,-T*u_no*]

When runtime options are specified, -Wl (l is lowercase) is required at the beginning of the runtime options, and the options must be separated by commas. If the same runtime option is specified more than once with different arguments, the last occurrence is used.

Example: Runtime option

```
$ ./a.out -Wl,-a,-p10,-x
```

-a

When the -a option is specified, an abend is executed forcibly following normal program termination. This processing is executed immediately before closing external files.

Example: Runtime option -a

```
$ ./a.out -Wl,-a
```

-d*num* 1 <= *num* <= 32767

The -d option determines the size of the input/output buffer used by a direct access input/output statement. The -d option improves input/output performance when data is read from or written to files a record at a time in sequential record-number order. If the -d option is specified, the input/output buffer size is used for all units used during execution.

To specify the size of the input/output buffer for individual units, specify the number of Fortran records in the environment variable fu*xx*bf (*xx* is the unit number). See Section "3.8 Using Environment Variables for Execution" for details. When the -d option and the environment variable are specified at the same time, the environment variable becomes effective. The option argument *num* specifies the number of Fortran records, in fixed-block format, included in one block. The option argument *num* must be an integer from 1 to 32767. The other integer value is specified, the -d option is invalid. To obtain the input/output buffer size, multiply *num* by the value specified in the RECL= specifier of the OPEN statement. If the files are shared by several processes, the number of Fortran records per block must be 1. If the -d option is omitted, the size of the input/output buffer is 8M bytes.

Example: Runtime option -d

```
$ ./a.out -Wl,-d10
```

-e*num* 0 <= *num* <= 32767

The -e option controls termination based on the total number of execution errors. The option argument *num*, specifies the error limit as an integer from 0 to 32767. When *num* is greater than or equal to 1, execution terminates when the total number of errors reaches the limit. If -e*num* is omitted or *num* is zero, execution is not terminated based on the error limit. However, program execution still terminates if the Fortran system error limit is reached.

Example: Runtime option -e

```
$ ./a.out -Wl,-e10
```

-g*num* 0 <= *num* <= 2097151

The -g option sets the size of the input/output buffer used by a sequential access input/output statement. This size is set in units of kilobytes for all unit numbers used during execution. The argument *num* must be an integer from 1 to 2097151. The other value is specified, the -g option is invalid. If the -g option is omitted, the size of the input/output defaults to 8M bytes.

The -g option improves input/output performance when a large amount of data is read from or written to files by an unformatted sequential access input/output statement. The argument *num* is used as the size of the input/output buffer for all units. To avoid using excessive memory, specify the size of the input/output buffer for individual units by specifying the size in the environment variable fu*xx*bf (*xx* is the unit number). See Section "3.8 Using Environment Variables for Execution" for details. When the -g option is and the environment variable are specified at the same time, the environment variable becomes effective.

Example: Runtime option -g

```
$ ./a.out -Wl,-g10
```

-i

The -i option controls processing of runtime interrupts. When the -i option is specified, the Fortran library is not used to process interrupts. When the -i option is not specified, the Fortran library is used to process interrupts. Diagnostic messages are output. For the error check, see Section "8.2.2 Debugging Programs for Abend" for details.

For the interrupt, see Section "8.1.7 Exception Handling Processing" for details.

Example: Runtime option -i

```
$ ./a.out -Wl,-i
```

-l*elvl elvl*: { i | w | e | s }

The -l option controls the output of diagnostic messages during execution. The option argument *elvl*, specifies the lowest error level, i, w, e, or s, for which diagnostic messages are to be output. If the -l option is not specified, diagnostic messages are output for error levels w, e, and s. However, messages beyond the print limit are not printed.

Example: Runtime option -l

```
$ ./a.out -Wl,-le
```

-li

The -li option outputs diagnostic messages for all error levels.

-lw

The -lw option outputs diagnostic messages for error levels w, e, s, and u.

-le

The -le option outputs diagnostic messages for error levels e, s, and u.

-ls

The -ls option outputs diagnostic messages for error levels s and u.

-m*u_no* 0 <= *u_no* <= 2147483647

The -m option connects the specified unit number *u_no* to the standard error output file where diagnostic messages are to be written. If the -m option is omitted, unit number 0, the system default, is connected to the standard error output file. See Section "3.8 Using Environment Variables for Execution", and Section "7.2 Unit Numbers and File Connection" for details.

Example: Runtime option -m

```
$ ./a.out -Wl,-m10
```

-n

The -n option controls whether prompt messages are sent to standard input. When the -n option is specified, prompt messages are output when data is to be entered from standard input using formatted sequential READ statements, including list-directed and namelist statements. If the -n option is omitted, prompt messages are not generated when data is to be entered from standard input using a formatted sequential READ statement.

Example: Runtime option -n

```
$ ./a.out -Wl,-n
```

-p*u_no* 0 <= *u_no* <= 2147483647

The -p option connects the unit number *u_no* to the standard output file. If the -p option is omitted, unit number 6, the system default, is connected to the standard output file. See Section "3.8 Using Environment Variables for Execution", and Section "7.2 Unit Numbers and File Connection" for details.

Example: Runtime option -p

```
$ ./a.out -Wl,-p10
```

-q

The -q option specifies whether to capitalize the E, EN, ES, D, Q, G, L and Z edit output characters produced by formatted output statements. This option also specifies whether to capitalize the alphabetic characters in the character constants used by the inquiry specifier (excluding the NAME specifier) in the INQUIRE statement. If the -q option is specified, the characters appear in uppercase letters. If the -q option is omitted, the characters appear in lowercase letters. In Fortran95 and Fortran 2003, the characters appear in uppercase letters so the -q option is not required.

Example: Runtime option -q

```
$ ./a.out -Wl,-q
```

-r*u_no* 0 <= *u_no* <= 2147483647

The -r option connects the unit number *u_no* to the standard input file during execution. If the -r option is omitted, unit number 5, the system default, is connected to the standard input file. See Section "3.8 Using Environment Variables for Execution", and Section "7.2 Unit Numbers and File Connection" for details.

Example: Runtime option -r

```
$ ./a.out -Wl,-r10
```

-tsec 1 <= sec <= 9999

The -t option sets the time limit in seconds for program execution. If -t20 is specified, for example, execution is canceled after 20 seconds.

Example : Runtime option -t

```
$ ./a.out -Wl,-t20
```

-x

The -x option determines whether blanks in numeric edited input data are ignored or treated as zeros. If the -x option is specified, blanks are changed to zeros during numeric editing with formatted sequential input statements for which no OPEN statement has been executed. The result is the same as when the BLANK= specifier in an OPEN statement is set to zero. If the -x option is omitted, blanks in the input field are treated as null and ignored. The result is the same as if the BLANK= specifier in an OPEN statement is set to NULL or if the BLANK= specifier is omitted.

Example: Runtime option -x

```
$ ./a.out -Wl,-x
```

-C or -C*u_no* 0 <= *u_no* <= 2147483647

The -C option specifies how to process an unformatted file of IBM370-format floating-point data using an unformatted input/output statement. When the -C option is specified, the data of an unformatted file associated with the specified unit number is regarded as IBM370-format floating-point data in an unformatted input/output statement. The option argument *u_no* specifies an integer from 0 to 147483647 as the unit number. If option argument *u_no* is omitted, the -C option is valid for all unit numbers connected to unformatted files. When the specified unit number is connected to a formatted file, the option is ignored for the file. When the -C option is not specified, the data of an unformatted file associated with unit number *u_no* is regarded as IEEE-format floating-point data in an unformatted input-output statement.

Example: Runtime option -C

```
$ ./a.out -Wl,-C10
```

-G or -G*u_no* 0 <= *u_no* <= 2147483647

The -G option specifies how to process an unformatted file of IBM370 EBCIDC character code data using an unformatted input/output statement. When the -G option is specified, the data of an unformatted file associated with the specified unit number is regarded as IBM370 EBCDIC character code data in an unformatted input/output statement. The option argument *u_no* specifies an integer from 0 to 2147483647 as the unit number. If option argument *u_no* is omitted, the -G option is valid for all unit numbers connected to unformatted files. When the specified unit number is connected to a formatted file, the option is ignored for the file. When the -G option is not specified, the data of an unformatted file associated with unit number *u_no* is regarded as ASCII character code data in an unformatted input-output statement.

Example: Runtime option -G

```
$ ./a.out -Wl,-G10
```

-Lb

Service routines use eight-byte logicals instead of four-byte logicals as arguments and function results.

Example: Runtime option -Lb

```
$ ./a.out -Wl,-Lb
```

-Li

Service routines use eight-byte integers instead of four-byte integers as arguments and function results.

Example: Runtime option -Li

```
$ ./a.out -Wl,-Li
```

-Lr

Service routines use eight-byte reals instead of four-byte reals as arguments and function results.

Example: Runtime option -Lr

```
$ ./a.out -Wl,-Lr
```

-Lu

If the length of the unformatted Fortran record to be transferred in a sequential access unformatted input/output statement is 2G bytes or more, input/output is by dividing it into multiple Fortran records. If the -Lu option is specified, the size of the field which exists in the top and the end of the logical record and which is assigned the length of Fortran record is expanded to 8 bytes from 4 bytes, and can be input/output as single Fortran record even if the length of the Fortran record is 2G bytes or more.

The data which has written to the -Lu option can be read only by unformatted sequential input statements with the -Lu option. And the data which has written without the -Lu option can be read only by that statements without the -Lu option.

If the -Lu option is specified differently across writing and reading the data, the unformatted sequential input statement may not work.

Example: Runtime option -Lu

```
$ ./a.out -Wl,-Lu
```

-M

The -M option specifies whether to output the diagnostic message (jwe0147i-w) when bits of the mantissa are lost during conversion of IBM370-IEEE-format floating-point data. If the -M option is specified, a diagnostic message is output if conversion of IBM370-IEEE-format floating-point data results in a bits of the mantissa being lost. When the -M option is omitted, the diagnostic message (jwe0147i-w) is not output.

Example: Runtime option -M

```
$ ./a.out -Wl,-M
```

-Q

The -Q option suppressed padding of an input field with blanks when a formatted input statement is used to read a Fortran record. This option applies to cases where the field width needed in a formatted input statement is longer than the length of the Fortran record and the file was not opened with and OPEN statement. The result is the same as if the PAD= specifier in an OPEN statement is set to NO. If the -Q option is omitted, the input record is padded with blanks. The result is the same as when the PAD= specifier in an OPEN statement is set to YES or when the PAD= specifier is omitted.

Example: Runtime option -Q

```
$ ./a.out -Wl,-Q
```

-Re

Disables the runtime error handler. The diagnostic message, traceback, error summaries, user control of errors by ERRSET and ERRSAV, and execution of user code for error correction are suppressed. The standard correction is processed if an error occurs.

Example : Runtime option -Re

```
$ ./a.out -Wl,-Re
```

-Rp

The -Rp option issues a diagnostic message under certain circumstances for programs running on multiple processors.

If the -Rp option is specified, a diagnostic message (jwe1034i-w) is output when a program compiled with the compiler option -Kparallel runs on only one processor.

If the -Rp option is not specified, the diagnostic message (jwe1034i-w) is not output.

Example : Runtime option -Rp

```
$ ./a.out -Wl,-Rp
```

-Ry

A diagnostic message may be output when a program used service routines that returns the year using only the last two digits.

The last two digits of the year are returned when one of the following procedures is used.

- DATE service subroutine
- JDATE service function

If the return values are compared by a user program, a malfunction may occur. Thus the program should receive proper consideration. By specifying the runtime option -Ry,-li, the diagnostic message is output when these routines are executed.

The routines that return a four-digit number for the year are listed below:

- CTIME service function
- FDATE service subroutine
- IDATE service subroutine
- GETDAT service subroutine
- DATE_AND_TIME intrinsic subroutine

Example: Runtime option -Ry

```
$ ./a.out -Wl,-Ry,-li
```

-T or -T*u_no* 0 <= *u_no* <= 2147483647

Little endian integer data, logical data, and IEEE floating-point data is transferred in an unformatted input/output statement.

The argument *u_no* is a unit number connected with an unformatted file. If *u_no* is omitted, -T takes effect for all unit numbers. If both -T and -T*u_no* are specified as in the example, "-Wl,-T,-T10", -T takes effect for all unit numbers.

Example: Runtime option -T

```
$ ./a.out -Wl,-T10
```

3.4 Execution Command Environment variable

The environment variable FORT90L may be used to specify runtime options. The runtime options specified in FORT90L are combined with the execution command option arguments. The execution command option arguments take precedence over the corresponding options specified in the environment variable FORT90L.

The following examples show how to use the environment variable FORT90L.

Example 1:

a. When using the environment variable FORT90L

```
$ export FORT90L='-Wl,-e99,-le'
$ ./a.out -Wl,-m99 -k
```

b. When not the using environment variable FORT90L

```
$ ./a.out -Wl,-e99,-le,-m99 -k
```

Both two above examples have the same meaning. When executing the command a.out, runtime options -e99, -le, and -m99, and user-defined executable program option -k are in effect.

Example 2:

```
$ export FORT90L='-Wl,-e10'
$ ./a.out -Wl,-e99
```

When executing execution command a.out, runtime option -e99 is in effect.

3.5 Execution Profile File

The default of runtime options can be changed by specifying the execution profile file (/etc/opt/FJSVmxlang/jwe_prof).

An execution profile file must be used according to the following format:

- Lines beginning with the pound sign (#) are treated as comment lines.
- A single runtime option cannot extend over multiple lines.
- The format of runtime options that can be specified must follow the conventions explained in Section 3.3 Runtime Options.

The following is an example of an execution profile file specification:

```
#Default options
-Wl,-x,-le
```

The priority of runtime options is as follows, highest first:

1. Execution command operand
2. Environment variable
3. Profile file
4. Default value

3.6 Execution Command Return Values

The following table lists return values set by the execution command.

Return value	Status
0	No error or level i (information message)
4	Level w error (warning)
8	Level e error (medium)
12	Level s error (serious)
16	Limit exceeded for level w, e, s error, or a level u error (unrecoverable) was detected
240	Abnormal termination
Others	Forcible termination

3.7 Standard Input, Output, and Error Output for Execution

The default unit numbers for standard input, output, and error output for execution commands are:

Standard input : Unit number 5

Standard output : Unit number 6

Standard error output : Unit number 0

3.8 Using Environment Variables for Execution

This section describes environment variables that control execution.

fu*xx filen*

The fu*xx* environment variable connects units and files. The value *xx* is a unit number. The value *filen* is a filename to be connected to a unit number. See Section "7.2 Unit Numbers and File Connection" for information on connecting units and files. The standard input and output files (fu05 and fu06) and error file (fu00) must not be specified.

The following example shows how to connect data.file to unit number 10 prior to the start of execution.

```
$ export fu10="data.file"
```

fu*xx*bf *size*

The fu*xx*bf environment variable specifies the size of the input/output buffer used by a sequential or direct access input/output statement for every unit number. The value *xx* in the fu*xx*bf environment variable specifies the unit number. The *size* argument used for sequential access input/output statements is in kilobytes; the *size* argument used for direct access input/output statements is in Fortran records. The *size* argument must be an integer with a value of 1 to 2097151. The *size* argument for direct access input/output statements is the number of Fortran records per block in fixed-block format. The *size* argument must be an integer from 1 to 2147483747/RECL= specifier for an OPEN statement that indicates the number of Fortran records per block. If the value of *size* argument is incorrect, the input/output buffer size is the default size. The default size is 8M bytes. If the environment variable and the runtime options -d or -g are specified at the same time, the environment variable is effective. However, if the *size* argument is not effective, the value specified with the runtime options is effective.

When sequential access input/output statements are executed for unit number 10, the statements use an input/output buffer of 64 kilobytes.

```
$ export fu10bf="64"
```

When direct access input/output statements are executed for unit number 10, the number of Fortran records included in one block is 50. The input/output buffer size is obtained by multiplying 50 by the value specified in the RECL= specifier of the OPEN statement.

```
$ export fu10bf="50"
```

FLIB_ALLOC_ALIGN *size*

The boundary of allocated memory by an ALLOCATE statement is aligned on a specified alignment boundary size. The value of *size* must be 16 or more and must be a power of two.

The following example shows how to align 128 byte boundary to allocated memory.

```
$ export FLIB_ALLOC_ALIGN=128
```

FLIB_EXCEPT u

The environment variable FLIB_EXCEPT=u controls floating point underflow interrupt processing.

If the program was compiled with the -NRtrap compiler option and the program was executed setting the environment variable FLIB_EXCEPT=u, the system performs floating point underflow interrupt processing. The system outputs diagnostic message jwe0012i-u during execution.

If the program was compiled without the -NRtrap compiler option or the program was executed without setting the environment variable FLIB_EXCEPT=u, the system ignores the floating point underflow.

The following example shows how to specify FLIB_EXCEPT.

```
$ export FLIB_EXCEPT=u
```

FLIB_HOOK_TIME *time*

Specifies the interval at which the user-defined subroutine is called at regular time interval. When *time* is specified in the environment variable FLIB_HOOK_TIME, the user-defined subroutine is called at interval of *time* milliseconds.

The unit for specifying *time* is milliseconds, and the valid value range is 0 <= *time* <= 2147483647. If 0 is specified for *time*, calling at regular time interval is disabled. If the environment variable is not specified, the Fortran system default value takes effect. The default value is 60000 milliseconds (one minute).

Refer to Section "8.2.3 Hook Function", for information about the hook function.

An example of specifying the environment variable FLIB_HOOK_TIME is shown below.

```
$ export FLIB_HOOK_TIME=600000
```

In this example, the user-defined subroutine is called at 10 minutes interval.

FLIB_IOBUFCPY

FLIB_IOBUFCPY is used for I/O buffer parallel transfer function. Only "MP" can be specified to FLIB_IOBUFCPY. See Section "12.4 I/O Buffer Parallel Transfer" for details.

```
$ export FLIB_IOBUFCPY="MP"
```

FLIB_IOBUFCPY_SIZE *size*

The environment variable FLIB_IOBUFCPY_SIZE is used to change the condition to execute I/O buffer parallel transfer function. When the *size* is specified to the environment variable FLIB_IOBUFCPY_SIZE, an unformatted I/O statement transfers data in parallel if data transfer size and I/O buffer size are more than *size* Kbytes. The *size* argument is in Kilobytes. The *size* argument must be an integer with a value of 1 or more.

The following example shows how to specify FLIB_IOBUFCPY_SIZE.

```
$ export FLIB_IOBUFCPY_SIZE="16"
```

See Section "12.4 I/O Buffer Parallel Transfer" for information on I/O buffer parallel transfer.

FLIB_UNDEF_VALUE *hex*

This environment variable change the values used for undefined data reference checks. Using *hex*, hexadecimal values can be specified. The valid range of values is 00 <= *hex* <= FF. If the environment variable is not specified or if *hex* value is disabled, the Fortran system default value takes effect. The default value is 8B.

See Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)" for details on undefined data reference checks.

An example of specifying the FLIB_UNDEF_VALUE environment variable is shown below.

```
$ export FLIB_UNDEF_VALUE=8C
```

In this example, the value 8C is applied in each byte of the data value used in the undefined data reference check.

FLIB_USE_STOPCODE *n*

This environment variable specifies whether to reflect the stop-code of the STOP/ERROR STOP statement in the return code. The least significant byte (0 to 255) in a stop-code is reflected in the return code if the value of *n* is 1. The stop-code is not reflected in the return code if anything other than 1 is specified to *n* or this environment variable is not specified.

The following example shows how to specify the environment variable FLIB_USE_STOPCODE.

```
$ export FLIB_USE_STOPCODE=1
```

In this example, the stop-code of the STOP/ERROR STOP statement is reflected in the return code.

TMPDIR *dir*

The TMPDIR environment variable changes the directory where an unnamed file is created. If this is not effective, an unnamed file is created on the current directory.

The following example shows how to specify TMPDIR.

```
$ export TMPDIR='/tmp'
```

In this example, an unnamed file is created on /tmp.

3.9 Notes for Execution

3.9.1 Notes for Execution of the Non-process Parallel Job on the FX100 System

Blurring or deterioration in the execution performance might happen because the access time of the memory is not uniform with the NUMA node where the memory is allocated in the NUMA architecture.

The problem can be evaded by fixing the NUMA node used by using the numactl command as follows for non-process parallel job whose number of threads is 16 (the number of all CPUs on one NUMA node) or less.

Example: Start of program using numactl command

```
$ numactl -C 0 ./a.out
```

The cpuid of NUMA node used is adjusted to 0 by the -C option.

Chapter 4 Messages and Listings

This chapter describes the system messages and listing produced as a result of compiling a Fortran source program or executing a Fortran program.

4.1 Compilation Output

4.1.1 Compilation Diagnostic Messages

The output from the frtpx command and the Fortran compiler includes diagnostic messages. If frtpx command line arguments contain errors or if errors or warnings are detected during the compilation of Fortran source programs, diagnostic messages are written.

Compiler Messages

Formats

Messages issued by frtpx are in the following format:

```
$ frtpx: Message-text
```

Messages issued by the Fortran compiler are in the following format:

```
jwdxxxxz-y filename line column Message-text
```

The elements of such a compiler message are defined as follows:

jwd*xxxxz-y*

jwd: A diagnostic message from the Fortran compiler

xxxx: Message serial number

z: Message type (i, o, s, or p)

Message type	Explanation
i	This message is unrelated to the optimization.
o	This message reports the optimization status.
s	This message reports the optimization of using SIMD instructions.
p	This message reports the optimization status of automatic parallelization.

y: Message error level (i, w, s, or u) and return code

Error level	Return code	Explanation
i	0	No error. The message is information.
w		The message is a warning. Processing continues.
s	1	A serious error. The system ignores the erroneous statement and continues processing other statements.
u		An unrecoverable error. The system terminates processing.

filename

The filename where the message event was detected

line

The line number where the message event was detected

column

The column number of the line where the message event was detected. The column number may be added to diagnostic messages.

Message-text

The message text is issued in either English or Japanese depending on user environment variables, such as LC_MESSAGES and LANG. Using compiler options, the user can specify either long or short message text.

4.1.2 Compilation Information

4.1.2.1 Compilation Information Options

Compilation information is output by specifying the -Q, -Nlst or -Nlst_out=*file* compiler option. The compilation information is output to a file with suffix .lst. If more than one Fortran source program file is specified, it is output to the first-file-name.lst. The compilation information can be controlled by specifying the arguments of -Q, -Nlst or -Nlst_out=*file*. See Section 4.1.2.2 Example of Compilation Information. The options and their meanings are as follows:

-Q or -Nlst

Outputs the source program list and diagnostic error messages.

-Qa or -Nlst=a

Outputs attributes of names in addition to the compilation information specified by the -Q or -Nlst option.

-Qd or -Nlst=d

Outputs information about derived type layout in addition to the compilation information specified by the -Q or -Nlst option.

-Qi or -Nlst=i

Outputs include file listings and a list of the names of include files in addition to the compilation information specified by the -Q or -Nlst option.

-Qm or -Nlst=m

If the -Qm or -Nlst=m option is specified, the Fortran program which expresses the situation of automatic parallelization with OpenMP directives is output in addition to the compilation information produced by the -Qp or -Nlst=p option. "!FRT" is added at the last of directives output by this function in order to distinguish from directives already described.

In addition, when the situation of automatic parallelization cannot be correctly expressed with OpenMP directive, the line which starts not in "!$OMP" but in "!!!!! PARALLEL INFO:" is output. The situation of the optimization performed simultaneously with automatic parallelization may be added to the line.

1. The kind of OpenMP directive output by this function is shown below.

   - PARALLEL DO
   - FIRSTPRIVATE(var_name,...)
   - LASTPRIVATE(var_name,...)
   - PRIVATE(var_name,...)
   - REDUCTION(+:var_name)
   - REDUCTION(-:var_name)
   - REDUCTION(*:var_name)
   - REDUCTION(MAX:var_name)
   - REDUCTION(MIN:var_name)
   - REDUCTION(.OR.:var_name)
   - REDUCTION(.AND.:var_name)

2. The kind of information output to the line which starts in "!!!!! PARALLEL INFO:" is shown below. Referring to these information, user can easily rewrite the program into OpenMP form.

   - LOOP INTERCHANGED
     It means that loop was interchanged.
   - LOOP FUSED
     It means that loop was fused.
   - LOOP SPLIT
     It means that loop was split.
   - COLLAPSED
     It means that nested do loops were collapsed into a single loop was performed.
   - OTHER METHODS
     It means that the situation of automatic parallelization cannot be correctly expressed with OpenMP directive by a reason other than the above. The example of the main factors is shown below.
     - The loop is not DO loop.
     - The loop including an induction variable.
     - The loop including a variable of derived type.
     - The loop including a definition in the array element with an invariable subscript.
     - The variable of reduction operation is the array element with an invariable subscript.
     - The initial value,terminal value or incrementation value of loop including the array element, function reference, and so on.
   - UNKNOWN VARIABLE(var_name)
     It means that it is the variable that kind of OpenMP directive is not decided correctly. The example of the main factors is shown below.
     - The variable that kind of OpenMP directive is not decided correctly because of optimization.
     - The variable of complex type.
   - Clause of OpenMP(var_name)
     When the kind of OpenMP directive is not decided correctly, the information for the clause of OpenMP might be output as a reference information. The kind of clause is shown below:
     - FIRSTPRIVATE(var_name,...)
     - LASTPRIVATE(var_name,...)
     - PRIVATE(var_name,...)
     - REDUCTION(+:var_name)
     - REDUCTION(-:var_name)
     - REDUCTION(*:var_name)
     - REDUCTION(MAX:var_name)
     - REDUCTION(MIN:var_name)
     - REDUCTION(.OR.:var_name)
     - REDUCTION(.AND.:var_name)

3. Notes on use are shown below.

   - When -Qm or -Nlst=m option is specified to file created by -P option, the file having name that .omp is added to the file name before preprocessing is created.

```
$ frtpx -P file.F95
  # file.cpp.f95 was created.
$ frtpx -Qm -Kparallel file.cpp.f95
```

```
  # file.omp.F95 was created.
$ frtpx -Nlst=m -Kparallel file.cpp.f95
  # file.omp.F95 was created.
```

- If OpenMP directive REDUCTION([MAX|MIN]:var_name) is created from Fortran program containing a variable or a program unit with the name MAX or MIN, the Fortran program can issue the error of S level and cannot be compiled. Correct Fortran program not to use the name MAX or MIN in that case.
- Even when parallelized, neither OpenMP directive nor "!!!!! PARALLEL INFO:" line may be output by the influence of optimization.

-Qo*file* or -Nlst_out=*file*

Outputs the information to a file named '*file*'.

-Qp or -Nlst=p

This option specifies optimization information and automatic parallelization information to be output along with listing of -Q or -Nlst option. Optimization information shows status of parallelization, inlining and unrolling. To know the output on statements specified with the OpenMP Fortran directives, see 12.3 Parallelization by OpenMP Specifications.

1. Automatic parallelization information

   The meanings of marks for DO statement or array expression are as follows.

   These describe the parallelization status of whole DO loop or array expression.

p	:	DO loop or array expression is parallelized
m	:	DO loop or array expression is partially parallelized
s	:	DO loop or array expression is not parallelized
blank	:	DO loop or array expression is not target of parallelization

   When the above mark "p", "m" or "s" is displayed for DO statement, "p", "m" or "s" is displayed for executable statement in loop. The meaning of marks for executable statements is as follows.

p	:	Executable statements can be parallelized
m	:	Executable statements can be partially parallelized
s	:	Executable statements cannot be parallelized

   The following condition applies to statements targeted for parallelization. Non-executable statements and executable statements that process only transfer of control are displayed by symbols determined by the parallelization information of executable statements before and after these statements. Transfer of control statements include ELSE, END IF, CONTINUE, ENDDO, ELSEWHERE, and so on. Also parallelization information for these statements may be blank. Even when a DO loop or array expression is enabled for parallelization, for the sake of performance, the system may not parallelize it. In this case it displays "s" for the DO statement and displays the unchanged analysis results for statements inside the DO loop.

2. Optimization information

   The meaning of optimization marks are as follows.

i	:	inlined the reference to procedure
number	:	unrolled expansion number
f	:	full unrolling the loop

-Qt or -Nlst=t

If the -Qt or -Nlst=t option is specified, the details optimization information and the statistics information is output in addition to the compilation produced by the -Qp or -Nlst=p option.

The optimization information is output in loop units. The parallel dimension information is output in statement units. The statistics information is output approximate stack size is used in a reference of procedure.

1. The outputs information in loop units is defined as follows:

   a. Displaying about automatic parallelization

      - Standard iteration count : *N*
        When the iteration count of a loop is *N* or more, the loop is brought to parallel execution. When less than *N*, it is brought to serial execution.

   b. Displaying optimization information in loop units

      - INTERCHANGED(nest:*nest-no*)
        It means that loop was interchanged.
        nest-no is the nesting level of the loop that was moved by loop interchange optimization.
        (nest:) may not be displayed if the other optimization like loop collapsing was applied.

      - FUSED(lines:*num1*,*num2*[,*num3*]...)
        It means that loops were fused.
        Loops at line *num2* and subsequent lines were fused into a loop at line *num1*.
        (lines:) isn't displayed at line *num2* and subsequent lines, that were fused to line *num1*.

      - SPLIT
        It means that loop was split.

      - COLLAPSED
        It means that nested loops were collapsed into a single loop.

      - SOFTWARE PIPELINING
        It means that loop was performed software pipelining.

      - SIMD(VL: *length*[, *length*]...)
        It means that SIMD instructions were generated for the loop.
        - A SIMD instruction treats *length* elements of array. When loop fission is applied to the loop and the values of *length* for the each loop are different, two or more *length*s are displayed.

      - UXSIMD
        It means that SIMD instructions were generated for the loop by UXSIMD.

      - STRIPING
        It means that loop was performed striping.

      - MULTI-OPERATION FUNCTION
        It means that loop includes multi-operation function.

      - PATTERN MATCHING(matmul)
        It means that loop was changed library function call(matmul).

      - UNSWITCHING
        It means that loop was performed loop unswitching.

      - FULL UNROLLING
        It means that loop was fully unrolled.

      - CLONE
        It means that CLONE optimization was applied to the loop.

      - LOOP VERSIONING
        It means that loop versioning was applied to the loop.

   c. Information that relates to the prefetch instruction is displayed.

      - PREFETCH: *N*
        Indicates the number of prefetch instructions that exist in the loop.

      - variable name: *N*,...
        Indicates the number of prefetch instructions for the variable name.

      - OTHER PREFETCH
        Indicates the number of prefetch instructions for the variable which was generated by compiler.

d. Information that relates to the xfill instruction is displayed.

- XFILL: *N*
  Indicates the number of xfill instructions that exist in the loop.
- variable name: *N*,...
  Indicates the number of xfill instructions for the variable name.
- OTHER XFILL
  Indicates the number of xfill instructions for the variable which was generated by compiler.

2. Marks for DO statement or array expression are as follows:

p	:	It means the beginning of parallelized range.(The mark "p" of -Qp or -Nlst=p option is output together, so not p but "pp" is displayed)

3. The statistics information is defined as follows:

a. The statistics information is output approximate stack size is used in a reference of procedure.

Note:

- The stack size in internal procedure is displayed with stack size in parent procedure.
- The following data is allocated stack area, but the size does not include outputs in this function.
  - The data is specified THREADPRIVATE directive.
  - The data is automatic data object and -Kautoobjstack option in effect.

b. The number of prefetch instructions procedure is displayed.

4. SIMD information

The meanings of marks for DO statement or array expression are as follows.

These describe the SIMD optimization status of whole DO loop or array expression.

x	:	DO loop or array expression can be applied SIMD optimization by UXSIMD
v	:	DO loop or array expression is applied SIMD optimization
m	:	DO loop or array expression is applied partially SIMD optimization
s	:	DO loop or array expression is not applied SIMD optimization
blank	:	DO loop or array expression is not target of SIMD optimization

When the above mark "v", "m" or "s" is displayed for DO statement, "v", "m" or "s" is displayed for executable statement in loop.

The meaning of marks for executable statements is as follows.

x	:	Executable statements can be applied SIMD optimization by UXSIMD
v	:	Executable statements can be applied SIMD optimization (*1)
m	:	Executable statements can be applied partially SIMD optimization (*1)
s	:	Executable statements cannot be applied SIMD optimization

The following condition applies to statements targeted for SIMD optimization.

Non-executable statements and executable statements that process only transfer of control are displayed by symbols determined by the SIMD optimization information of executable statements before and after these statements.

Transfer of control statements include ELSE, END IF, CONTINUE, ENDDO, ELSEWHERE, and so on.

Also SIMD optimization information for these statements may be blank.

Even when a DO loop or array expression is enabled for SIMD optimization, for the sake of performance, the system may not SIMD optimization it. In this case it displays "s" for the DO statement and displays the unchanged analysis results for statements inside the DO loop.

*1: When SIMD optimization is applied to a DO loop and the mark "v" or "m" is displayed for the DO statement, the mark "v" could be displayed for statements in the loop though SIMD extensions are not used for them.
The mark "s" is displayed for a statement preventing SIMD optimization, which is a clue to tune programs.

-Qx or -Nlst=x

Outputs a cross reference listing of names and labels in addition to the compilation information specified by the -Q or -Nlst option.

These arguments can be specified at the same time, separated by comma as in -Qa,o*file*,x or -Nlst=a,lst_out=*file*,lst=x.

4.1.2.2 Example of Compilation Information

The following is an example of the output produced by the -Q or -Nlst option.

```
Main program "CHECK"
 (line-no.)(nest)
         1              PROGRAM CHECK
         2              INTEGER,DIMENSION(10,10) :: A = 0
         3              INTERFACE
         4                   SUBROUTINE SUB(I,J,A)
         5                   INTEGER,DIMENSION(:,:) :: A
         6                   END SUBROUTINE
         7              END INTERFACE
         8     1        DO I=1,10
         9     2          DO J=MOD(I,2)+1,10,2
        10     2              CALL SUB(I,J,A)
        11     2          END DO
        12     1        END DO
        13              WRITE (*,"(10I3)") (A(:,I),I=1,10)
        14              END PROGRAM

Procedure information
  Lines      : 14
  Statements : 14

 External subroutine subprogram "SUB"
 (line-no.)(nest)
        15
        16              SUBROUTINE SUB(I,J,A)
        17              INTEGER,DIMENSION(:,:) :: A
        18              A(I,J) = 1
        19              END SUBROUTINE

Procedure information
  Lines      : 5
  Statements : 4

Total information
  Procedures       : 2
  Total lines      : 19
  Total statements : 18
```

The following is an example of diagnostic error messages that are output.

```
 Main program "ERROR"
 (line-no.)(nest)
         1              PROGRAM ERROR
         2     1         DO 10 I=1,3
         3     2          DO 20 J=1,3
         4     2             CALL SUB(I*J)
         5     2     10   ENDDO
         6     1        END PROGRAM

Diagnostic messages: program name(ERROR)
```

```
   jwd1027i-s  "error.f", line 3: Undefined statement label '   20' referenced.
   jwd1131i-s  "error.f", line 3, column 9: The program unit contains unmatched nest in DO construct,
IF construct, CASE construct, SELECT TYPE construct, ASSOCIATE construct, CRITICAL construct, or
BLOCK construct.

 Procedure information
   Lines      : 6
   Statements : 6

  External subroutine subprogram "SUB"
  (line-no.)(nest)
          7
          8              SUBROUTINE SUB(I)
          9               INTEGER,INTENT(IN) :: I
         10               PRINT *,I()
         11              END SUBROUTINE

 Diagnostic messages: program name(SUB)
   jwd2008i-i  "error.f", line 8: Dummy argument 'I' not used in this subprogram.
   jwd1771i-s  "error.f", line 10, column 16: 'I' already has the INTENT attribute.

 Procedure information
   Lines      : 5
   Statements : 4

 Total information
   Procedures       : 2
   Total lines      : 11
   Total statements : 10
```

The following is an example from the -Qi or -Nlst=i option.

```
  Module "MOD"
  (inc)(line-no.)(nest)
               1              MODULE MOD
               2                CHARACTER(LEN=15) CC
               3              END MODULE

 Procedure information
   Lines      : 3
   Statements : 3

Main program "INCLUDE"
  (inc)(line-no.)(nest)
               4
               5              PROGRAM INCLUDE
               6              INCLUDE "INC1"
     1         1        use mod,only:cc
               7              CALL INIT()
               8              PRINT *,CC
               9              END PROGRAM

 Procedure information
   Lines      : 7
   Statements : 5

External subroutine subprogram "INIT"
  (inc)(line-no.)(nest)
              10
              11              SUBROUTINE INIT()
              12              INCLUDE "INC2"
     2         1        include "inc1"
     1         1        use mod,only:cc
```

```
    2         2
             13              CC = "FUJITSU FORTRAN"
             14              END SUBROUTINE

 Procedure information
   Lines      : 5
   Statements : 3

 Total information
   Procedures       : 3
   Total lines      : 14
   Total statements : 10

 Include file name list
     1 : ./inc1
     2 : ./inc2
```

The following is an example of the output from the -Qp -Kparallel -x- or -Nlst=p -Kparallel -x- option.

```
  Main program "MAIN"
  (line-no.)(nest)(optimize)
          1                            INTEGER,PARAMETER ::N=10000
          2                            REAL,DIMENSION(N)::A
          3     1   p   6               DO I=1,N
          4     1   pi  6                 A(I) = A(I) * FUNC(I)
          5     1   p   6               ENDDO
          6                            END

 Procedure information
   Lines      : 6
   Statements : 6

External function subprogram "FUNC"
  (line-no.)(nest)(optimize)
          7                            FUNCTION FUNC(I)
          8                               FUNC=I+1
          9                            END

 Procedure information
   Lines      : 3
   Statements : 3

 Total information
   Procedures       : 2
   Total lines      : 9
   Total statements : 9
```

The following is an example of the output from the -Qx or -Nlst=x option.

```
  Module "COMPLEX"
  (line-no.)(nest)
          1                          MODULE COMPLEX
          2                            TYPE TYPE1
          3                              SEQUENCE
          4                              REAL :: R_PART
          5                              REAL :: I_PART
          6                           END TYPE TYPE1
          7                            INTERFACE OPERATOR(+)
          8                              MODULE PROCEDURE PLUS
          9                            END INTERFACE
         10                            PRIVATE PLUS
         11                            CONTAINS
         12                              TYPE(TYPE1) FUNCTION PLUS(OP1,OP2)
         13                                TYPE(TYPE1),INTENT(IN) :: OP1,OP2
```

```
        14                                  PLUS%R_PART = OP1%R_PART + OP2%R_PART
        15                                  PLUS%I_PART = OP1%I_PART + OP2%I_PART
        16                              END FUNCTION PLUS
        17                            END MODULE

 Procedure information
   Lines      : 17
   Statements : 17

 Scoping unit of module : COMPLEX
   Cross reference of name
     COMPLEX
      |(Class and Type) : module name
      |(Declaration)    : 1
      |(Definition)     :
      |(Reference)      :
     PLUS
      |(Class and Type) : module function name, TYPE(TYPE1)
      |(Declaration)    : 8  10
      |(Definition)     :
      |(Reference)      :
     TYPE1
      |(Class and Type) : type name
      |(Declaration)    :
      |(Definition)     : 2
      |(Reference)      :
     [OPERATOR(+)]
      |(Class and Type) : user defined operator
      |(Declaration)    : 7
      |(Definition)     :
      |(Reference)      :

 Scoping unit of module sub-program : PLUS
   Cross reference of name
     OP1
      |(Class and Type) : variable name, TYPE(TYPE1)
      |(Declaration)    : 12  13
      |(Definition)     :
      |(Reference)      : 14  15
     OP2
      |(Class and Type) : variable name, TYPE(TYPE1)
      |(Declaration)    : 12  13
      |(Definition)     :
      |(Reference)      : 14  15
     PLUS
      |(Class and Type) : variable name, TYPE(TYPE1)
      |(Declaration)    :
      |(Definition)     : 14  15
      |(Reference)      :
     PLUS
      |(Class and Type) : module function name, TYPE(TYPE1)
      |(Declaration)    :
      |(Definition)     : 12
      |(Reference)      : 16
     TYPE1
      |(Class and Type) : type name
      |(Declaration)    :
      |(Definition)     :
      |(Reference)      : 12  13

Main program "MAIN"
  (line-no.)(nest)
        18
```

```
        19                              PROGRAM MAIN
        20                                USE COMPLEX
        21                                TYPE(TYPE1) CPX1, CPX2, RESULT
        22                                READ (*,*) CPX1, CPX2
        23                                RESULT = CPX1 + CPX2
        24                                PRINT *,RESULT
        25                              END PROGRAM MAIN

Procedure information
  Lines      : 8
  Statements : 7

Scoping unit of program : MAIN
  Cross reference of name
    COMPLEX
     |(Class and Type) : module name
     |(Declaration)    :
     |(Definition)     :
     |(Reference)      : 20
    CPX1
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Declaration)    : 21
     |(Definition)     : 22
     |(Reference)      : 23
    CPX2
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Declaration)    : 21
     |(Definition)     : 22
     |(Reference)      : 23
    MAIN
     |(Class and Type) : program name
     |(Declaration)    : 19
     |(Definition)     :
     |(Reference)      : 25
    RESULT
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Declaration)    : 21
     |(Definition)     : 23
     |(Reference)      : 24
    TYPE1
     |(Class and Type) : type name
     |(Declaration)    :
     |(Definition)     :
     |(Reference)      : 21
    [OPERATOR(+)]
     |(Class and Type) : user defined operator
     |(Declaration)    :
     |(Definition)     :
     |(Reference)      : 23

Total information
  Procedures       : 2
  Total lines      : 25
  Total statements : 24
```

The following is an example of the output from the -Qa,x or -Nlst=a,lst=x option.

```
 Module "COMPLEX"
 (line-no.)(nest)
        1                          MODULE COMPLEX
        2                            TYPE TYPE1
        3                              SEQUENCE
        4                              REAL :: R_PART
```

```
         5                                REAL :: I_PART
         6                              END TYPE TYPE1
         7                              INTERFACE OPERATOR(+)
         8                                MODULE PROCEDURE PLUS
         9                              END INTERFACE
        10                              PRIVATE PLUS
        11                              CONTAINS
        12                                TYPE(TYPE1) FUNCTION PLUS(OP1,OP2)
        13                                  TYPE(TYPE1),INTENT(IN) :: OP1,OP2
        14                                  PLUS%R_PART = OP1%R_PART + OP2%R_PART
        15                                  PLUS%I_PART = OP1%I_PART + OP2%I_PART
        16                              END FUNCTION PLUS
        17                            END MODULE

Procedure information
  Lines      : 17
  Statements : 17

Scoping unit of module : COMPLEX
  Attribute and Cross reference of name
    COMPLEX
     |(Class and Type) : module name
     |(Attributes)     :
     |(Declaration)    : 1
     |(Definition)     :
     |(Reference)      :
    PLUS
     |(Class and Type) : module function name, TYPE(TYPE1)
     |(Attributes)     : PUBLIC
     |(Declaration)    : 8  10
     |(Definition)     :
     |(Reference)      :
    TYPE1
     |(Class and Type) : type name
     |(Attributes)     : PUBLIC
     |(Declaration)    :
     |(Definition)     : 2
     |(Reference)      :
    [OPERATOR(+)]
     |(Class and Type) : user defined operator
     |(Attributes)     : PUBLIC
     |(Declaration)    : 7
     |(Definition)     :
     |(Reference)      :

Scoping unit of module sub-program : PLUS
  Attribute and Cross reference of name
    OP1
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Attributes)     : INTENT(IN), dummy-argument
     |(Declaration)    : 12  13
     |(Definition)     :
     |(Reference)      : 14  15
    OP2
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Attributes)     : INTENT(IN), dummy-argument
     |(Declaration)    : 12  13
     |(Definition)     :
     |(Reference)      : 14  15
    PLUS
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Attributes)     : result-value
     |(Declaration)    :
```

```
     |(Definition)     : 14  15
     |(Reference)      :
    PLUS
     |(Class and Type) : module function name, TYPE(TYPE1)
     |(Attributes)     :
     |(Declaration)    :
     |(Definition)     : 12
     |(Reference)      : 16
    TYPE1
     |(Class and Type) : type name
     |(Attributes)     : host-associated
     |(Declaration)    :
     |(Definition)     :
     |(Reference)      : 12  13

Main program "MAIN"
  (line-no.)(nest)
        18
        19                          PROGRAM MAIN
        20                            USE COMPLEX
        21                            TYPE(TYPE1) CPX1, CPX2, RESULT
        22                            READ (*,*) CPX1, CPX2
        23                            RESULT = CPX1 + CPX2
        24                            PRINT *,RESULT
        25                          END PROGRAM MAIN

 Procedure information
   Lines      : 8
   Statements : 7

 Scoping unit of program : MAIN
   Attribute and Cross reference of name
    COMPLEX
     |(Class and Type) : module name
     |(Attributes)     :
     |(Declaration)    :
     |(Definition)     :
     |(Reference)      : 20
    CPX1
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Attributes)     :
     |(Declaration)    : 21
     |(Definition)     : 22
     |(Reference)      : 23
    CPX2
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Attributes)     :
     |(Declaration)    : 21
     |(Definition)     : 22
     |(Reference)      : 23
    MAIN
     |(Class and Type) : program name
     |(Attributes)     :
     |(Declaration)    : 19
     |(Definition)     :
     |(Reference)      : 25
    RESULT
     |(Class and Type) : variable name, TYPE(TYPE1)
     |(Attributes)     :
     |(Declaration)    : 21
     |(Definition)     : 23
     |(Reference)      : 24
    TYPE1
```

```
     |(Class and Type) : type name
     |(Attributes)     : use-associated
     |(Declaration)    :
     |(Definition)     :
     |(Reference)      : 21
    [OPERATOR(+)]
     |(Class and Type) : user defined operator
     |(Attributes)     : use-associated
     |(Declaration)    :
     |(Definition)     :
     |(Reference)      : 23

 Total information
   Procedures       : 2
   Total lines      : 25
   Total statements : 24
```

The following is an example of the output from the -Qd or -Nlst=d option.

```
  Module "COMPLEX"
  (line-no.)(nest)
          1                            MODULE COMPLEX
          2                              TYPE TYPE1
          3                                SEQUENCE
          4                                REAL :: R_PART
          5                                REAL :: I_PART
          6                             END TYPE TYPE1
          7                              INTERFACE OPERATOR(+)
          8                                MODULE PROCEDURE PLUS
          9                              END INTERFACE
         10                              PRIVATE PLUS
         11                              CONTAINS
         12                                TYPE(TYPE1) FUNCTION PLUS(OP1,OP2)
         13                                  TYPE(TYPE1),INTENT(IN) :: OP1,OP2
         14                                  PLUS%R_PART = OP1%R_PART + OP2%R_PART
         15                                  PLUS%I_PART = OP1%I_PART + OP2%I_PART
         16                              END FUNCTION PLUS
         17                            END MODULE

 Procedure information
   Lines      : 17
   Statements : 17

 Scoping unit of module : COMPLEX
   Derived type construction
     TYPE1
      |  (Attributes) : SEQUENCE
      |R_PART
      |  (Type and Attributes) : REAL(4), PUBLIC
      |I_PART
      |  (Type and Attributes) : REAL(4), PUBLIC

 Scoping unit of module sub-program : PLUS

Main program "MAIN"
  (line-no.)(nest)
         18
         19                            PROGRAM MAIN
         20                              USE COMPLEX
         21                              TYPE(TYPE1) CPX1, CPX2, RESULT
         22                              READ (*,*) CPX1, CPX2
         23                              RESULT = CPX1 + CPX2
         24                              PRINT *,RESULT
```

```
        25                          END PROGRAM MAIN

 Procedure information
   Lines      : 8
   Statements : 7

 Scoping unit of program : MAIN

 Total information
   Procedures       : 2
   Total lines      : 25
   Total statements : 24
```

The following is an example of the output from the -Kparallel,reduction -Qm or -Kparallel,reduction -Nlst=m option.(.omp file)

```
       SUBROUTINE SUB(N,M)
      REAL,DIMENSION(10000) :: A
      REAL,DIMENSION(10000,100) :: C,D
      REAL::RRES=0.0

      CALL INIT(A,B,C)

!$OMP PARALLEL DO REDUCTION(MAX:RRES) !FRT
      DO I=1,N
        IF (RRES<A(I)) RRES = A(I)
      END DO
      CALL OUTPUT(RRES)
!$OMP PARALLEL DO PRIVATE(I) !FRT
      DO J=1,M
        DO I=1,N
          C(I,J) = C(I,J) + D(I,J)
        END DO
      END DO
      CALL OUTPUT(C)

      DO I=1,N
!!!!! PARALLEL INFO: LOOP INTERCHANGED
        DO J=1,M
          D(I,J) = D(I,J) + C(I,J)
        END DO
      END DO
      CALL OUTPUT(D)
      END
```

The following is an example of the output produced by the -Kfast -Kparallel -Qt or -Kfast -Kparallel -Nlst=t option.

```
External subroutine subprogram "SUB"
  (line-no.)(nest)(optimize)
          1                           SUBROUTINE SUB(A,B,C,N)
          2                           IMPLICIT NONE
          3                           INTEGER(KIND=4) :: N
          4                           REAL(KIND=8),DIMENSION(N,N) :: A,B
          5                           REAL(KIND=8) :: C
          6                           INTEGER(KIND=4) :: I,J
                                <<< Loop-information Start >>>
                                <<<  [OPTIMIZATION]
                                <<<    INTERCHANGED(nest: 2)
                                <<<    SIMD(VL: 4)
                                <<<    SOFTWARE PIPELINING
                                <<< Loop-information  End >>>
          7     1   p   8v            DO I=2,N
                                <<< Loop-information Start >>>
                                <<<  [PARALLELIZATION]
                                <<<    Standard iteration count: 3
```

```
                            <<<  [OPTIMIZATION]
                            <<<    INTERCHANGED(nest: 1)
                            <<< Loop-information  End >>>
         8     2  pp   8               DO J=2,N
         9     2   p   8v                A(I,J) = A(I,J) + B(I,J) * C
        10     2   p   8               END DO
        11     1   p                 END DO
        12                           END SUBROUTINE

 Procedure information
   Lines      : 12
   Statements : 12
   Stack(byte): 32
   Prefetch num: 0

 Total information
   Procedures       : 1
   Total lines      : 12
   Total statements : 12
   Total stack(byte): 32
   Total prefetch num: 0
```

The following is an example of the output from the -Kparallel,uxsimd,eval -Qt or -Kparallel,uxsimd,eval -Nlst=t option.

```
External subroutine subprogram "SUB"
  (line-no.)(nest)(optimize)
         1                           SUBROUTINE SUB(NN,E,D)
         2                           IMPLICIT NONE
         3                           INTEGER,PARAMETER:: TN=8  ! THREAD NUM
         4                           INTEGER :: NN,K,I,J
         5                           REAL*8 :: D(8,NN,TN),E(8,NN,TN)
         6
                            <<< Loop-information Start >>>
                            <<<  [PARALLELIZATION]
                            <<<    Standard iteration count: 2
                            <<< Loop-information  End >>>
         7     1  pp                  DO K=1,TN
                            <<< Loop-information Start >>>
                            <<<  [OPTIMIZATION]
                            <<<    SOFTWARE PIPELINING
                            <<< Loop-information  End >>>
         8     2   p                     DO I=1,NN
                            <<< Loop-information Start >>>
                            <<<  [OPTIMIZATION]
                            <<<    SIMD(VL: 4)
                            <<<    UXSIMD
                            <<< Loop-information  End >>>
         9     3   p   4vx                 DO J=1,8
        10     3   p   4vx                       D(J,I,K) = 0.0000001D0*DBLE(I)/10000.D0
        11     3   p   4vx                       E(J,I,K) = 0.0000002D0*DBLE(I)/10000.D0
        12     3   p   4v                  ENDDO
        13     2   p                     ENDDO
        14     1   p                   ENDDO
        15                             END
```

4.1.2.3 Notes on Compilation Information

Notes on Compilation Information are shown in the following.

- The -Qp (-Nlst=p) option and -Qt (-Nlst=t) option may output wrong compilation information (optimization information) in the following cases.
  - When a function is inline expanded, different optimizations may be applied to each line. In this case, the compiler may output information incorrectly as follows.
    - Output messages redundantly.
    - Output optimization messages inconsistent with compilation information (optimization information).
    - Output no compilation information (optimization information).

    In addition, PREFETCH: *N*, which is the number of prefetch instructions in the inlined function, includes all prefetch instructions expanded in multiple lines.
  - The compiler applies multiple optimizations to one loop. In this case, the compiler may output information incorrectly as follows.
    - Output optimization information on line number to an incorrect line.
    - Output optimization messages inconsistent with compilation information (optimization information).
  - When multiple loops are written in the same line in a program, the compiler outputs optimization information for only one of the loops. Write one loop in one line in order to output optimization information.
  - Detailed optimization information for a loop which consists of IF constructs and GO TO statements is not output. Optimization information on line number may be output to an incorrect line.
- The compiler may generate loops when any of the following conditions is met. The optimization messages may be output for the generated loops, and compilation information (optimization information) may be output for the generated loops when -Qp (-Nlst=p) or -Qt (-Nlst=t) option is specified.
  - The actual argument is a pointer array, assumed shape array, or array section, and the corresponding dummy argument is a nonpointer and nonassumed shape array.
  - -Nquickdbg=undef, -Nquickdbg=undefnan, or -Nsetvalue=array option is effective.
  - An array variable name appears in FIRSTPRIVATE, LASTPRIVATE, REDUCTION, COPYIN, or COPYPRIVATE clause of OpenMP.
  - There is a loop parallelized automatically by -Karray_private option or optimization control specifier ARRAY_PRIVATE, FIRST_PRIVATE or LAST_PRIVATE.
- When link time optimization is applied, the optimization which is different from the optimization displayed in compilation information may be applied at runtime.
- When a #line directive is included in the source program, compilation information (optimization information) output by -Qp (-Nlst=p) or -Qt (-Nlst=t) option is output based on the line number specified by the #line directive. Suppose, for example, a #line directive specifying the line 3 in the source program is attached to a DO statement which is in the line 10. In this case, the compilation information (optimization information) about the DO statement is output to the line 3 in the source program.

4.2 Executable Program Output

Program execution generates the following output:

- Print data from output statements
- Messages from PAUSE or STOP/ERROR STOP statements
- Diagnostic messages
- Trace back map
- Error summary information
- Debug output

This section describes the diagnostic messages, trace back map, and error summary information output when errors are detected during Fortran program execution. See 7.7 Data Transfer Input/output Statements for output statements. See 6.5.6 PAUSE Statement, and 6.5.7 STOP/ERROR STOP Statement for PAUSE or STOP/ERROR STOP statements. See 8.2 Debugging Functions for debugging function.

4.2.1 Diagnostic Messages

A diagnostic message is output if PAUSE statements or STOP/ERROR STOP statements are executed or an error is detected during Fortran program execution.

Messages issued by the Fortran library are in the following format:

```
jwexxxxt-y Additional-information Message-text
```

a. jwe*xxxxt-y*
   jwe : A diagnostic messages from the Fortran library
   xxxx : A message serial number
   t: Output is 'i' or 'a'. 'i' is output for messages not requiring a response. 'a' is output by a PAUSE statement, which requires a response. See 6.5.6 PAUSE Statement for the PAUSE statement.

y	Explanation
i	Not an error. The message is information message.
w	The message is a warning. Processing continues.
e	A medium error. The system ignores the erroneous statement and continues processing. Up to 10 erroneous statements can be detected before the system terminates processing. The error number for system terminates can be controlled -enum runtime option and runtime error processing library.
s	A serious error. The system ignores the erroneous statement and terminates processing.
u	An unrecoverable error. The system terminates processing.

b. Additional-information

   Additional information may be added to diagnostic messages. The additional information indicates the line number of the Fortran source program where the error was detected.

c. Message-text

   The message text is issued in English.

4.2.2 Trace Back Map

A trace back map is generated as specified in the error control table. This information indicates how the main program calls the program in which the error was detected. See 8.1 Error Processing for information about the error control table. See 8.1.7 Exception Handling Processing, and 8.2.2 Debugging Programs for Abend for exception handling. See 2.2 Compiler Options for a compiler option -Nnoline.

The trace back map is generated in the following format after a diagnostic message:

```
error occurs at name1 line s1 loc wwwwwwww offset xxxxxxxx
name1         at loc yyyyyyyy called from loc zzzzzzzz in name2 line s2
name2         at loc yyyyyyyy called from loc zzzzzzzz in name3 line s2
name3         at loc yyyyyyyy called from o.s
```

name1

Intrinsic function name or entry name of the program unit where the error was detected. If the name is unknown, '????????' is assigned to *name1*.

name2

Intrinsic function name or entry name of the program unit where the error was detected. If the name is unknown, '????????' is assigned to *name2*.

name3

Entry name of the main program; o.s indicates the control program. If the name is unknown, '????????' is assigned to *name3*.

s1

Line number of the statement where the error was detected. If the compiler option -Nnoline is specified, it does not appear.

s2

Line number of the statement where the program unit is called. If the compiler option -Nnoline is specified, it does not appear.

wwwwwwww

Absolute address (hexadecimal) of the statement where the error was detected.

xxxxxxxx

Relative address (hexadecimal) of the statement where the error was detected, starting at the beginning of the program unit; not output for intrinsic function errors; if this address is unknown, it does not appear.

yyyyyyyy

Absolute address (hexadecimal) of the beginning of the program unit; if this address is unknown, it is '0000000000000000'.

zzzzzzzz

Absolute address (hexadecimal) of the statement that called the program unit; if this address is unknown, it is '0000000000000000'.

name1, *name2*, and *name3* are processing names of the procedure names. The processing names and corresponding procedure names are shown in the following table:

Kind of procedure	Processing name
Main-program	MAIN__
Internal-subprogram of main-program	Main-program-name.internal-subprogram-name_
External-subprogram	External-subprogram-name_
Internal-subprogram of external-subprogram	External-subprogram-name.internal-subprogram-name_
Module-subprogram	Module-name.module-subprogram-name_
Internal-subprogram of module-subprogram	Module-name.module-subprogram-name.internal-subprogram-name_

Note1 : Processing names are uniformly lowercase unless the compiler option -AU is specified.

Note2 : The information in shared objects may be not correct.

4.2.3 Error Summary

The following error summary information is output when program execution is completed.

However, the system errors whose error identification number is 900 to 1000 are not output.

```
error summary (Fortran)
error number error level error count
 jwennnni         y            zzz
     :             :             :
total error count = xxx
```

nnnn

Message serial number

y

Error level *y*(*y*:i, w, e, s, u) detected

zzz

Error count

xxx

Total error count of diagnostic messages

Chapter 5 Data Types, Kinds, and Representations

This section describes the Fortran data types and kinds, and describes the machine representations of these data.

5.1 Internal Data Representation

The following describes the internal representation for each data type.

5.1.1 Integer Type Data

Kind 1 (one-byte) integer type data is represented internally as an 8-bit two's complement binary number with values ranging from -128 to 127.

Kind 2 (two-byte) integer type data is represented internally as a 16-bit two's complement binary number with values ranging from -32768 to 32767.

Kind 4 (four-byte) integer type data is represented internally as a 32-bit two's complement binary number with values ranging from -2147483648 to 2147483647.

Kind 8 (eight-byte) integer type data is represented internally as a 64-bit two's complement binary number with values ranging from -9223372036854775808 to 9223372036854775807.

In two's complement representation, the first bit of each integer is the sign. For a positive integer, the sign bit is zero; for a negative integer, the sign bit is 1. For example, the kind 1 representation of -1 is Z'FF'. For integer zero, all bits are set to zero.

5.1.2 Logical Type Data

Kind 1 (one-byte) logical type data is represented internally as an 8-bit binary number. For TRUE, the value is Z'01'. For FALSE, all bits are zero.

Kind 2 (two-byte) logical type data is represented internally as a 16-bit binary number. For TRUE, the value is Z'0001'. For FALSE, all bits are zero.

Kind 4 (four-byte) logical type data is represented internally as a 32-bit binary number. For TRUE, the value is Z'00000001'. For FALSE, all bits are zero.

Kind 8 (eight-byte) logical type data is represented internally as a 64-bit binary number. For TRUE, the value is Z'0000000000000001'. For FALSE, all bits are zero.

5.1.3 Real and Complex Type Data

The internal representations of all kinds of real type data conform to IEEE specifications.

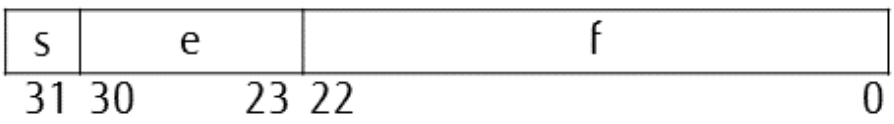
The internal representations of the real and imaginary parts of all kinds of complex type data are the same as the internal representations of the corresponding kinds of real type data. A complex type value requires twice the storage of a real type value of the same kind.

The following describes in detail how floating-point data is stored in memory.

Kind 4 (single precision) real and complex

The figure below shows the bit composition for sign (s), exponent (e), and fraction (f).

Default real storage format:



The table below shows the bit patterns for numbers stored as default real values.

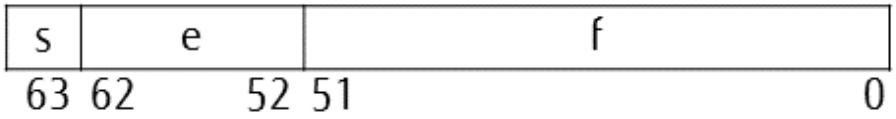
Table 5.1 Default real values

Common name	Storage format	Value
0	00000000 0.	0e+00
+ maximum normal number	7f7fffff	3.40282347e+38
- maximum normal number	ff7fffff	-3.40282347e+38
+ minimum normal number	00800000	1.17549435e-38
- minimum normal number	80800000	-1.17549435e-38
+ maximum denormalized number	007fffff	1.17549421e-38
- maximum denormalized number	807fffff	-1.17759421e-38
+ minimum denormalized number	00000001	1.40129846e-45
- minimum denormalized number	80000001	-1.40129846e-45
+infinity	7f800000	Inf (Infinity)
-infinity	ff800000	-Inf
+quiet_NaN	7fc00000-7fffffff	NaN (Not a number)
-quiet_NaN	ffc00000-ffffffff	-NaN
+signaling_NaN	7f800001-7fbfffff	NaN
-signaling_NaN	ff800001-ffbfffff	-NaN

Kind 8 (double precision) real and complex

The figure below shows the bit composition of sign (s), exponent (e), and fraction (f).

Double-precision real storage format:



The table below shows the bit patterns for numbers stored as double-precision real values.

Double-precision real values

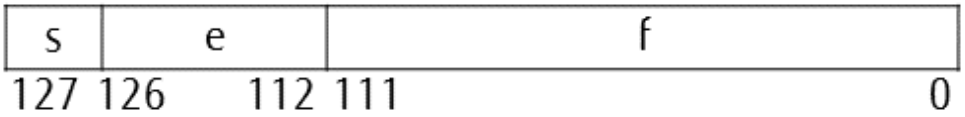
Common name	Storage format	Value
0	00000000 00000000	0.0e+00
+ maximum normal number	7feffff fffffff	1.797693134862316d+308
- maximum normal number	ffeffff fffffff	-1.797693134862316d+308
+ minimum normal number	00100000 00000000	2.225073858507201d-308
- minimum normal number	80100000 00000000	-2.225073858507021d-308
+ maximum denormalized number	000fffff fffffff	2.225073858507201d-308
- maximum denormalized number	800fffff fffffff	-2.225073858507021d-308
+ minimum denormalized number	00000000 00000001	4.940656458412465d-324
- minimum denormalized number	80000000 00000001	-4.940656458412465d-324
+infinity	7ff00000 00000000	Inf
-infinity	fff00000 00000000	-Inf
+quiet_NaN	7ff80000 00000000	NaN

Common name	Storage format	Value
	- 7fffffff ffffffff	
-quiet_NaN	fff80000 00000000 - ffffffff ffffffff	-NaN
+signaling_NaN	7ff00000 00000001 - 7ff7ffff ffffffff	NaN
-signaling_NaN	fff00000 00000001 - fff7ffff ffffffff	-NaN

Kind 16 (quadruple precision) real and complex

The figure below shows the bit composition of sign (s), exponent (e), and fraction (f).

Quadruple-precision real storage format:



The table below shows the bit patterns for numbers stored as quadruple-precision real values.

Quadruple-precision real values

Common name	Storage format	Value
0	00000000 00000000 00000000 00000000	0.0e+00
+ maximum normal number	7ffeffff ffffffff ffffffff ffffffff	1.189731495357231765085759326628007 0q+4932
- maximum normal number	fffeffff ffffffff ffffffff ffffffff	-1.189731495357231765085759326628007 0q+4932
+ minimum normal number	00010000 00000000 00000000 00000000	3.362103143112093506262677817321752 6q-4932
- minimum normal number	80010000 00000000 00000000 00000000	-3.362103143112093506262677817321752 6q-4932
+ maximum denormalized number	0000ffff ffffffff ffffffff ffffffff	3.362103143112093506262677817321751 9q-4932
- maximum denormalized number	8000ffff ffffffff ffffffff ffffffff	-3.362103143112093506262677817321751 9q-4932
+ minimum denormalized number	00000000 00000000 00000000 00000001	6.475175119438025110924438958227646 5q-4966
- minimum denormalized number	80000000 00000000 00000000 00000001	-6.475175119438025110924438958227646 5q-4966
+infinity	7fff0000 00000000 00000000 00000000	Inf
-infinity	ffff0000 00000000 00000000 00000000	-Inf
+quiet_NaN	7fff8000 00000000 00000000 00000000 -7fffffff ffffffff ffffffff ffffffff	NaN
-quiet_NaN	ffff8000 00000000 00000000 00000000 - ffffffff ffffffff ffffffff ffffffff	-NaN

Common name	Storage format	Value
+signaling_NaN	7fff0000 00000000 00000000 00000001 -7fff7fff ffffffff ffffffff ffffffff	NaN
-signaling_NaN	ffff0000 00000000 00000000 00000001 - ffff7fff ffffffff ffffffff ffffffff	-NaN

5.1.4 Character Type Data

Character data is represented internally using ASCII codes. Each character is represented as an 8-bit binary number without a parity bit.

All ASCII characters, both printable and nonprintable, are allowed in character constants and comments.

5.1.5 Derived Type Data

A derived type is a data type derived from the intrinsic data types, whose ultimate components are of intrinsic type. The derived type scalar data, which is a set of intrinsic type data, is called a structure.

If a derived type is not a sequence type, the storage sequence of the structure is undefined.

5.2 Correct Data Boundaries

Data must be aligned on the correct boundaries. The compiler usually aligns the data on the correct boundaries.

For elements of derived type and variable in a common block, the compiler assigns the data to the correct boundaries. However, if the compiler option -AA is specified, the compiler may not align data on the correct boundaries. In this case, the boundaries are those specified by the user for variables that share storage space because of an EQUIVALENCE statement. For a derived type definition, the boundary that follows the last component must be a correct boundary for the derived type.

For example, consider the following derived type definition when the compiler option -AA is specified:

```
TYPE TAG
   REAL(8)       R
   INTEGER(4)   I
END TYPE
```

The correct boundary for this derived type (TAG) is an address that is a multiple of 8. But because the boundary for the component I is a multiple of 4, but not 8, this derived type definition is incorrect.

Table 5.1 lists the correct data boundaries.

Table 5.2 Correct boundaries for data types

Data type	Correct boundary
One-byte integer type	Arbitrary address
Two-byte integer type	Address in multiples of 2
Four-byte integer type	Address in multiples of 4
Eight-byte integer type	Address in multiples of 8
One-byte logical type	Arbitrary address
Two-byte logical type	Address in multiples of 2
Four-byte logical type	Address in multiples of 4
Eight-byte logical type	Address in multiples of 8
Single-precision real type	Address in multiples of 4
Double-precision real type	Address in multiples of 8
Quadruple-precision real type	Address in multiples of 8

Data type	Correct boundary
Single-precision complex type	Address in multiples of 4
Double-precision complex type	Address in multiples of 8
Quadruple-precision complex type	Address in multiples of 8
Character type	Arbitrary address
Derived type	Address in multiples of the largest required multiple for all of the components

Boundary of real type array

The first element of a single-precision real type array is allocated on 8-byte boundary, and double-precision real type array's is allocated on 16-byte boundary. This is because these arrays may be applied SIMD optimizations.

However, arrays passed as arguments, arrays in a COMMON block which are not first element of the block and pointer arrays may not be allocated on above boundary.

5.3 Precision Conversion

The precision of the arithmetic data that Fortran handles is restricted by the hardware. This section explains how to use the precision-improving and precision-lowering functions that the system provides.

5.3.1 Precision Improving

The precision-improving function increases the precision in programs. This function converts constants, variables, and functions of the specified type to the next higher precision.

5.3.1.1 Precision-Raising Options

Precision improving can be executed for real and complex constants, variables, and functions.

If compiler option -Ad is specified, the data types are converted as follows:

```
Single-precision real      ->      Double-precision real
Single-precision complex   ->      Double-precision complex
```

If compiler option -Aq is specified, the data types are converted as follows:

```
Double-precision real      ->      Quadruple-precision real
Double-precision complex   ->      Quadruple-precision complex
```

An example of precision improving for constants, variables, and functions when compiler option -Ad is specified as follows.

Example: Precision improving for constants, variables, and functions

```
IMPLICIT REAL(8)(D)
REAL KNOTE,KIN
COMPLEX ARY(10),CDATA
KNOTE = 1.0
KIN = 3.111-2.5D0*KNOTE
DEX = 0.5D0-1.111111111111
ARY(1) = (1.1,1.0)
KIN = KIN-SQRT(KIN/2.0)
```

The above program is equivalent to the following program.

```
IMPLICIT REAL(8)(D)
REAL(8) KNOTE,KIN
COMPLEX(8) ARY(10),CDATA
KNOTE =1.0_8
KIN =3.111_8-2.5D0*KNOTE
DEX =0.5D0-1.111111111111_8
ARY(1) =(1.1_8,1.0_8)
KIN =KIN-DSQRT(KIN/2.0_8)
```

If precision is improved for an intrinsic function, the type of the function result is changed to the higher precision type. The function name is not changed if an EXTERNAL statement is used to declare a user-defined function.

For user-defined external functions, precision is improved for the function result. If the precision is improved for a function reference, functions must also be recompiled using the precision-improving function.

For intrinsic functions, precision improving affects function names, the types of arguments, and function results. Intrinsic functions with specific names are changed to the appropriate name. If an intrinsic function has a generic name, the precision-improving function is executed for the arguments, and an appropriate version of the function is selected. If an intrinsic function is used as an actual argument, the name is changed to reflect precision improving.

5.3.2 Precision Lowering and Analysis of Errors

The precision-lowering function determines the effect that a rounding error has on the result. Rounding

errors occur because of the limit on floating-point data precision.

Numerical programs can be debugged using the precision-lowering function to check whether algorithm

errors are tolerable.

For example, consider programs that solve simultaneous linear equations or obtain eigenvalues or eigenvectors of matrices. In these programs, a slight input error in the matrix may cause an extremely large error. Thus, the user may not be able to determine whether the result is correct. The precision-lowering function is provided to improve and debug numerical programs by detecting sections easily affected by errors. This facility helps the user select the appropriate solution to resolve the errors and minimize the effect of input errors.

Generally, the following relationship can be determined for the number of arithmetic digits and the precision of the answer:

```
d = (p - a)/m
```

```
d : Precision of answer (digits)
p : Number of arithmetic digits
a : Missing digits
m : Values for the problem and solution method
```

The value of m is usually 1 and corresponds to the order of multiple roots when roots of algebraic equations are obtained. The preceding equation expresses the problem conditions precisely. However, the equation does not apply when:

- Input data or a constant is erroneous.
- A problem equation contains an error.
- A calculation terminates inappropriately.

That is, the above expression comes into existence when rounding error a prime cause. Incrementing and decrementing the digits by one increments and decrements the solution precision by 1/m digits. Thus, a slight shifting of the number of digits shifts the solution precision by the same amount. The precision-lowering function uses this principle to obtain the solution precision.

Compiler option -C*l* enables the precision-lowering function.

5.3.2.1 Defining the Precision-Lowering Function

Specifying the following option enables the precision-lowering function:

```
-Cl   0 ≤ l ≤ 15
```

The precision-lowering function can be used on all kinds (precisions) of real and complex data.

If the precision-lowering function is effect, when an assignment statement defines the all kinds of real and complex data, the low-order *l* bits are set to zero.

The precision-lowering number must be between 0 and 15, inclusive. The result when *l* is zero is equivalent to specifying compiler option -C*l*. The PRNSET service subroutine can be called to change l dynamically. Thus, the precision-lowering function operates when *l* is changed dynamically to a value other than zero.

The precision-lowering number *l* is determined for each executable program. Therefore, when the PRNSET service subroutine is called from parallel region, the dynamic change by a specific thread may affect the execution of the another threads. The initial value of *l* is specified when the main program is compiled. Compiler option -C*l* only starts the precision-lowering function in subprograms; *l* itself is meaningless in this case.

5.3.2.2 Using the Precision-Lowering Function

In floating-point calculations, *l* can be changed, the same calculation re-executed, and the results compared.

First time:

The program is compiled and executed using -C0.

Second time:

The program is compiled and executed using -C1.

If the digits from two consecutive attempts are identical, don't assume that either result is correct. If the digits from two consecutive attempts are not identical, assume that one or the other of the two results is incorrect.

The number of executions can be increased to improve the accuracy. If you change the value of *l* to a number from 2 to 10 and increase the number of executions, the precision of the result can be determined accurately. Generally, two executions are sufficient.

However, it is more efficient to use the computer for only one execution to determine the precision. Source code to determine the precision from only one execution must be created. This program must use compiler option -C*l* and include the PRNSET service subroutine. Use the following procedure to create this source code:

1. Save the data to be changed in the check program section.
2. Execute CALL PRNSET(*l*).
3. Use the saved data in the calculations. Store the result in *l*.
4. Change *l*. Return to step 2 if the specified count is not reached.
5. Compare the results for each *l*. Print the information that matches.

An example of a program that uses the precision-lowering function as follows.

Example: Program that uses the precision-lowering function

```
      REAL,DIMENSION(10,11)::A,B
      REAL,DIMENSION(10,10)::X
 1    READ (5,FMT='(I2,I2)',END=99) N,M ! Enter data
      PRINT '(I2,I2)',N,M       ! Print data
      DO I=1,N
         READ (5,'(5F12.8)') (A(I,J),J=1,N+1)
         WRITE(6,'(5F12.8)') (A(I,J),J=1,N+1)
      END DO
      DO L=1,M
         CALL PRNSET(L-1)
         CALL SWEEP(A,B,N) ! Calculate while changing
!     precision lowering
         X(:N,L)=B(:N,N+1)
      END DO
      WRITE(UNIT=*,FMT='(//,A,5X,3(A,10X),A)') &
      & ' -Cl ','X1=1.0','X2=2.0','X3=1.0','X4=-1.0'
       DO L=1,M
          WRITE(6,'(A,I1,A,1P8E16.7)') &
          & ' -C',L-1,' ',X(:N,L) ! Print the result
       END DO
```

```
       WRITE(6,'(/,A,7X,3(A,14X),A)') ' -Cl:-C0','X1','X2','X3','X4'
      DO L=2,M
       X(:N,L)=X(:N,L)-X(:N,1)
      END DO
      DO L=2,M ! Result of -Cl and -C0
         WRITE(6,'(A,I1,A,1P8E16.7)') ' -C',L-1,':-C0',X(:N,L)
      END DO
      GO TO 1
 99   END
      SUBROUTINE SWEEP(A,Z,N)
      REAL,DIMENSION(10,11)::A,Z
      Z(:N,:N+1)=A(:N,:N+1)
      DO K=1,N
         Z(K,K+1:N+1) = Z(K,K+1:N+1)/Z(K,K)
         DO I=1,N              ! Use the sweep method to solve the
!     simultaneous linear equations
            IF(K.EQ.I) CYCLE
            Z(I,K+1:N+1)=Z(I,K+1:N+1)-Z(I,K)*Z(K,K+1:N+1)
         END DO
      END DO
      END SUBROUTINE SWEEP
```

In the printed input data, the first line indicates the original values and test number. The second to fifth lines indicate the four coefficients and one constant.

```
4 9
3.20000005  1.25979996  -2.01999998 5.13980007  -1.44019997
1.02010000  -1.35010004 3.09999990  -2.12010002 3.53999996
-2.02979994 2.54999995  -1.37020004 3.64000010  -1.94000006
3.21000004  1.11020005  2.80999994  4.53999996  3.70040011
```

```
 -Cl      X1=1.0          X2=2.0          X3=1.0          X4=-1.0
 -C0    9.9909294E-01   1.9976463E+00   1.0000743E+00  -9.9882913E-01
 -C1    9.9581611E-01   1.9891458E+00   1.0003431E+00  -9.9459982E-01
 -C2    9.9198890E-01   1.9792151E+00   1.0006566E+00  -9.8966026E-01
 -C3    1.0101604E+00   2.0263634E+00   9.9916744E-01  -1.0131168E+00
 -C4    1.0316334E+00   2.0820694E+00   9.9740791E-01  -1.0408325E+00
 -C5    1.0174370E+00   2.0452271E+00   9.9856758E-01  -1.0225105E+00
 -C6    8.9227295E-01   1.7205429E+00   1.0088348E+00  -8.6096954E-01
 -C7    1.0482483E+00   2.1251831E+00   9.9604034E-01  -1.0623016E+00
 -C8    9.1392517E-01   1.7766724E+00   1.0070496E+00  -8.8891602E-01
```

Difference between -C*I* and -C0

```
 -Cl:-C0       X1              X2              X3              X4
 -C1:-C0  -3.2768250E-03  -8.5005760E-03   2.6881695E-04   4.2293072E-03
 -C2:-C0  -7.1040392E-03  -1.8431187E-02   5.8233738E-04   9.1688633E-03
 -C3:-C0   1.1067390E-02   2.8717041E-02  -9.0682507E-04  -1.4287710E-02
 -C4:-C0   3.2540321E-02   8.4423065E-02  -2.6663542E-03  -4.2002678E-02
 -C5:-C0   1.8343925E-02   4.7580719E-02  -1.5066862E-03  -2.3681164E-02
 -C6:-C0  -1.0681915E-01  -2.7709961E-01   8.7604523E-03   1.3785934E-01
 -C7:-C0   4.9155235E-02   1.2753677E-01  -4.0339231E-03  -6.3470840E-02
 -C8:-C0  -8.5166931E-02  -2.2097397E-01   6.9752932E-03   1.0991287E-01
```

Notes:

1. This problem uses the sweep method to solve simultaneous linear equations that have poor characteristics (conditions).

2. When the number of precision-lowering bits varies from 0 to 8, this program prints the difference between the solution and the result of -C0.

The difference in the result of using -C0 almost indicates an error, which shows that debugging is sufficient.

Chapter 6 Notes on Programming

This chapter describes some of notes on programming on the Fujitsu Fortran system.

Input/output features are discussed in "Chapter 7 Input/output Processing".

6.1 Fortran Versions

Four versions of Fortran, all compiled using the frtpx command, are available:

1. FORTRAN66: Programs written in this version are called FORTRAN66 programs. Use compiler option -X6.
2. FORTRAN77: Programs written in this version are called FORTRAN77 programs. The compiler treats programs as FORTRAN77 programs under the following conditions:
   - The compiler option -X7 is specified.
   - The file suffix is ".f", ".for", ".F" or ".FOR", and the compiler option -Xd7 is specified.

   Example:

   The compiler treats the following program as a FORTRAN77 program:

   ```
   $ frtpx a.f90 -X7
   ```

   ```
   $ frtpx b.f -Xd7
   ```
3. Fortran 95 (including Fortran 90): Programs written in this version are called Fortran 95 programs. The compiler treats programs as Fortran 95 programs under the following conditions:
   - The compiler option -X9 is specified.
   - The file suffix is ".f90", ".f95", ".F90" or ".F95", and the compiler option -X is not specified.

   Example:

   The compiler treats the following program as a Fortran 95 program:

   ```
   $ frtpx b.f -X9
   ```

   ```
   $ frtpx a.f90
   ```

   ```
   $ frtpx a.f95
   ```
4. Fortran 2003: Programs written in this version are called Fortran 2003 programs. The compiler treats programs as Fortran 2003 programs under the following conditions:
   - The compiler option -X03 is specified.
   - The file suffix is ".f03" or ".F03", and the compiler option -X is not specified.
   - The file suffix is ".f", ".for", ".F" or ".FOR", and the compiler option -Xd7 is not specified.

   Example:

   ```
   $ frtpx b.f
   ```

   ```
   $ frtpx a.f90 -X03
   ```

   The compiler treats this program as a Fortran 2003 program:

Each Fortran Version, some items have different interpretations.

They are interpreted in accordance with any of the following Fortran version, and run.

- The Fortran version when the main program is compiled.
- The Fortran version of a program that includes a specification that is fixed when compiling a program

This compiler compiles all FORTRAN66, FORTRAN77, Fortran 95 (including Fortran 90) and Fortran 2003 programs. However, differences in the four versions can cause incorrect results.

The following items are interpreted differently, depending on whether compiler option -X6, -X7, -X9 or -X03 is specified.

The items and their interpretations are shown in the following.

These items are interpreted differently by language specification.	The interpretation is decided when the each program is compiled	The interpretation is decided when the main program is compiled
EXTERNAL statement that specifies an intrinsic function (Refer to Section "6.6.2 An Intrinsic Function that Specifies EXTERNAL Attribute")	*	
DO loop iteration count method (Refer to Section "6.5.4.1 DO Loop Iteration Count Method")	*	
Effect of the X edit descriptor (Refer to Section "7.17.5 Effect of the X Edit Descriptor")		*
REAL and CMPLX used as generic names (Refer to Section "6.6.3 REAL and CMPLX Used as Generic Names")	*	
Intrinsic procedures (Refer to Section "6.6.4 Intrinsic Procedures")	*	
Intrinsic function names with the type declared (Refer to Section "6.6.5 Intrinsic Function Names with the Type Declared")	*	
Assignment statements with overlapping character positions (Refer to Section "6.4.1 Assignment Statements with Overlapping Character Positions")	*	
allocatable assignment (Refer to Section "6.4.2 Allocatable Assignment".)	*	
SAVE attribute for initialized variables (Refer to Section "6.2.5 SAVE Attribute for Initialized Variables")	*	
OPEN statement with RECL= and with no ACCESS= specifier (Refer to Section "7.4.4 Default Value of ACCESS= Specifier with RECL=Specifier")	*	*
INQUIRE statement (Refer to Section "7.6.2 Difference in Language Version of the INQUIRE statement")		*
Generalized integer editing (Refer to Section "7.17.1 Generalized Integer Editing")		*
Output character in formatted output statement (Refer to Section "7.17.2 Exponent Character in Formatted Output")		*
Result of G editing (Refer to Section "7.17.3 Result of G Editing")		*
NAMELIST input/output (Refer to Section "7.7.4.3 NAMELIST Input/output Version Differences")		*
Diagnostic messages for a character string edit descriptor (Refer to Section "7.17.4 Diagnostic Messages for the Character String Edit Descriptor")		*
The form of nondelimited character constants produced by list-directed output (Refer to Section "7.7.5.2.1 The Form of Nondelimited Character Constants Produced by List-Directed Outputt")		*
The specified value of the OPEN statement (Refer to Section "7.4.2 Executing an OPEN Statement on a Connected File")		*

These items are interpreted differently by language specification.	The interpretation is decided when the each program is compiled	The interpretation is decided when the main program is compiled
Input/Output of Fortran records (Refer to Section "7.4.3 Input/output of Fortran Records")		*
Data transfer in direct access I/O (Refer to Section "7.7.3.3 The Error of Data Transfer in a DIRECT Access Input/output")		*
Input result of L editing (Refer to Section "7.17.6 L Edit Descriptor")		*
Variable enclosed by parenthesis (Refer to Section "6.3.8 Variable enclosed by parenthesis")	*	
The form of simple output list enclosed in parentheses (Refer to Section "6.5.9 Simple Output List Enclosed in Parentheses")	*	
The output statement that starts by TYPE keyword (Refer to Section "6.5.10 The Output Statement that starts by TYPE Keyword")	*	
List-Directed Input/Output Statements (Refer to Section "7.7.5.3 List-Directed Input/output Version Differences")		*
The returned values of ATAN2, ATAN2D and ATAN2Q when negative zero is specified for the argument (Refer to Section "6.6.7 The returned values of the intrinsic function")		*
The returned values of LOG and SQRT when negative zero is specified for imaginary part of the argument of complex type (Refer to Section "6.6.7 The returned values of the intrinsic function")		*

6.2 Data

This section discusses features related to data.

6.2.1 COMMON Statement

When using a COMMON statement, note the following items:

6.2.1.1 Boundaries of Variables in Common Blocks

The variables in common blocks must be aligned on the correct boundaries. For more information, see Section "5.2 Correct Data Boundaries". If data is not aligned on the correct boundaries, the compiler inserts space in the storage area and aligns the data correctly. However, there are exceptions, as described below.

- If the compiler option -AA is specified, the compiler does not insert space.
- Variables of common entities with BIND attribute have interoperable boundary.
- When double precision or quadruple precision real or complex variables are not aligned on addresses that are multiples of eight, these variables are aligned on addresses that are multiples of four. But if the compiler option -Kdalign is specified, they are aligned on addresses that are multiples of eight.

The beginning address of a common block is always a multiple of eight.

6.2.1.2 Common Blocks that Have Initial Values

When either of the following is specified, an error is detected at linking:

1. In different files, common blocks with the same name or blank commons are initialized more than once.

   Example 1:

   Error 1 at linking:

```
! file name : a.f                    ! file name : b.f
PROGRAM MAIN                         SUBROUTINE SUB
COMMON /COM/A,B                      COMMON /COM/A,B
DATA A/1.0/                          DATA B/2.0/
CALL SUB                             PRINT *,A,B
END                                  END
```

2. In different files, common blocks with the same name or blank commons that have different sizes are used, and the common block that does not have the maximum size of the same common blocks is initialized.

   Example 2:

   Error 2 at linking:

```
! file name : a.f                    ! file name : b.f
PROGRAM MAIN                         BLOCK DATA BLK
COMMON /COM/A,B                      COMMON /COM/A
PRINT *,A                            DATA A/1.0/
END                                  END
```

6.2.2 EQUIVALENCE Statement

Each equivalence object must be aligned on the correct boundary.

6.2.3 Initialization

This section describes items related to initialization.

6.2.3.1 Character Initialization

If a character constant is specified for the corresponding noncharacter type variable, the system outputs a level w diagnostic message. If a character string that exceeds the length of an element is specified as an initial value, the excess is not used to initialize the subsequent element.

Example1: Initializing array names

```
CHARACTER(3) A(2)
DATA A/'ABCDEF'/
```

Initial values in this case:

```
A(1): 'ABC'
A(2): No initial value (undefined)
```

Example2: Initializing array element

```
CHARACTER(LEN=2) B(4)
DATA B(2)/'ABCDE'/
```

Initial values in this case:

```
B(1) No initial value (undefined)
B(2) 'AB'
B(3) No initial value (undefined)
B(4) No initial value (undefined)
```

The following examples give 'CD' and 'E' as initial values to B(3) and B(4).

```
CHARACTER(LEN=2) B(4)
DATA B(2),B(3),B(4)/'AB','CD','E'/
```

or

```
CHARACTER(LEN=2) B(4)
CHARACTER(LEN=6) BB
EQUIVALENCE (B(2),BB)
DATA BB/'ABCDE'/
```

6.2.3.2 Initialization with Hexadecimal, Octal, and Binary Constants

To initialize an array, the number of hexadecimal, octal, or binary constants must match the number of array elements. If these numbers do not match, the remaining part is undefined. Thus, one hexadecimal, octal, or binary constant corresponds to one array element. If a constant exceeds an element in size, the excess is truncated.

Example: Initializing array elements with hexadecimal values

```
INTEGER A(3)
DATA A/Z'F0F0A100F2F2A202'/
```

Initial value of A(1)	:	F2F2A202
Initial value of A(2)	:	No initial value (undefined)
Initial value of A(3)	:	No initial value (undefined)

If a hexadecimal, octal, or binary constant that exceeds the size of an array element is specified, the high-order excess is truncated.

If a hexadecimal, octal, or binary constant is shorter than an array element, the space to the left of the constant is padded with zeroes.

6.2.3.3 Overlapping Initialization

If multiple initial values are specified for the same variable, the system outputs a level w diagnostic message and uses the last value specified in the source code. However, the code should be corrected.

6.2.4 Using CRAY Pointers

When using CRAY pointers, note the following items:

a. Do not save the address of a dummy argument in a subprogram.

Example: Incorrect use of a pointer (1)

```
SUBROUTINE SUB(A)
POINTER (IP,IPP)
INTEGER A(10),IPP(10)
SAVE IP
:
IP = LOC(A(2))
:
END
```

b. Do not return the address of an actual argument as a result.

Example: Incorrect use of a pointer (2)

```
FUNCTION ADDR(I)
INTEGER ADDR,I(5)
ADDR = LOC(I(2))
END
POINTER (IP,POINTER)
INTEGER POINTER,DATA(5),ADDR
IP = ADDR(DATA)
```

```
POINTER = 1
END
```

c. The data referenced by a pointer must not overwrite other data, except when the address is explicitly defined in a program unit by the intrinsic function LOC. Do not use the address in an assignment statement for which the value is copied.

Example: Incorrect use of a pointer (3)

```
POINTER (IP1,ARRAY1)
POINTER (IP2,ARRAY2)
INTEGER DATA(5),ARRAY1(5),ARRAY2(5)
IP1 = LOC(DATA)
ARRAY1 = 1
IP2 = IP1
ARRAY2 = 2 ! cannot use the address defined
END ! in assignment statement
```

6.2.5 SAVE Attribute for Initialized Variables

In Fortran 95 and Fortran 2003, initialized variables have the SAVE attributes.

In FORTRAN66 and FORTRAN77, initialized variables do not have the SAVE attribute unless they are given the SAVE attribute explicitly.

Example: The SAVE attribute for a variable with an initial value

```
SUBROUTINE S
INTEGER::I=1
I=I+1
END
```

In Fortran 95, the previous program is the same as the following:

```
SUBROUTINE S
INTEGER::I=1
SAVE I
I=I+1
END
```

6.2.6 Operation of Real or Complex type for Initialization Expression

When real or complex type is operated in initialization expression of nonexecutable statement, the result may have some difference within the margin of error even if the same operation appears in executable statement.

6.3 Expressions

This section discusses features related to expressions and operations.

6.3.1 Integer Data Operations

If integer data is assigned, the size is not checked. If the result of integer operations exceeds the range allowed, the result is undefined. This rule is especially critical for integer comparisons.

Example:

```
IF(I-J)1,2,3
```

Note:

In this example, the result of I-J can overflow. Change the expression to a comparison using relational operators in a logical IF statement.

If overflow occurs when evaluating the constant expression that defines a parameter, the value of the named constant is undefined.

6.3.2 Real Data Operations

Real data of any precision is processed as accurately as possible. However, machine language does not always represent data exactly.

Example: Decisions with no tolerance are allowed, but the result may not be what is expected

```
IF (A-0.1) 1,2,1
3 IF (B.EQ.C) GO TO 4
```

Note:

Even if 0.1 is the expected result of calculating A, processing does not always go to the statement with label 2. Even if B and C are expected to be equal, processing does not always go to the statement with label 4.

Do not execute equality or inequality value comparisons with relational operations using real or complex data. Level i diagnostic messages are produced for relational expressions that do not allow tolerance if the -Ec compiler option is specified. Relational expressions with no tolerance allowed are "e1 relop e2", where relop (relational operator) is .EQ., ==, .NE., or /=, and variables e1 and e2 are type complex or real of any kind (precision). See Section "2.2 Compiler Options" for information on the -Ec option. The following is an example.

Example: Preferred decision

```
IF (ABS(A-0.1)-0.00001) 2,1,1
3 IF (ABS(B-C).LT.D) GO TO 4
```

In the constant expression defining a parameter, do not specify a constant expression whose evaluated result might be an denormalized number (see Section "5.1.3 Real and Complex Type Data"). Such a number may cause unexpected results.

6.3.3 Logical Data Operations

Do not assign a value other than the result of a logical expression (.TRUE. or .FALSE.) for logical data and treat that value as logical. For example, do not initialize logical variable with a hexadecimal number and treat the value as logical. Also, do not equivalence a logical and integer variable and treat the integer as logical.

Example: Logical operation using other than logical values

```
LOGICAL A/Z'00000002'/,B
INTEGER I/2/
EQUIVALENCE (I,B)
IF (A) GO TO 10
IF (B) GO TO 20
IF (A.AND.B) GO TO 30
IF (A.OR.B) GO TO 40
```

Note:

In this case, do not expect the logical IF statement to execute the GO TO statement.

The result of a logical expression is TRUE or FALSE. Logical operators do not perform logical operations for each bit.

Example:

```
LOGICAL(1) L1,L2
DATA L1/Z'FF'/
L2=.NOT.L1
```

Note:

In this example, the result of L2 is not necessarily false.

6.3.4 Evaluation of Operations

The result type of operation +, -, *, / , and ** between eight-byte integer and single-precision real or complex is different in Fortran versions.

For Fortran 95 and Fortran 2003 program, the result type of these operations is of type single-precision real or single precision-complex.

For FORTRAN66 and FORTRAN77 program, the result type of these operations is of type double precision-real or double-precision complex.

6.3.5 Order of Evaluation of Data Entities

The order of evaluation of data entities connected by an operator is undefined.

For example, it is not allowed in a program to assume that one of F(X) and F(Y) should be evaluated previous to the other in the expression F(X) + F(Y).

6.3.6 Evaluation of a part of Expression

When an expression is evaluated, some parts of the expression may not be evaluated.

For example, in evaluating the logical expression L1.AND.L2 L2, L2 may not be evaluated if L1 is false because the value of the expression is clearly false with out evaluating L2.

Therefore, such a function reference as a part of an expression must be written carefully.

6.3.7 Definition and Undefinition by Function Reference

Evaluation of a function reference may not change the value of the data entity referred outside the function reference in the same statement.

For example, in the expression F(I).AND.I==1, evaluation of the function reference F(I) may not change the value of I.

Evaluation of the function reference in the logical expression of the logical IF statement can exceptionally change the value of the data entity referred in the executable statement of the logical IF statement.

Additionally, evaluation of a function reference in a statement may not change the value of the common block element whose value might affect evaluation of any other function references in the same statement.

For example, it is not allowed evaluating the expression FA( ) + FB( ) where functions FA and FB are defined.

Example:

```
FUNCTION FA()
COMMON /C/A
FA=A
END
FUNCTION FB()
COMMON /C/A
A=1
FB=A
END
```

6.3.8 Variable enclosed by parenthesis

An actual argument is different in Fortran versions when a variable is enclosed by parenthesis in an actual argument of user's procedure.

In FORTRAN66 and FORTRAN77, the actual argument has the described variable.

In Fortran95 and Fortran 2003, the value of the variable is copied to the temporary area, and the actual argument has temporary area.

6.4 Assignment Statement

6.4.1 Assignment Statements with Overlapping Character Positions

In FORTRAN66 and FORTRAN77, if the -Nf90move compiler option is not specified, overlapping data (in a storage area) may not be specified on both sides of a character assignment statement. If the -Nf90move compiler option is specified, overlapping character data is permitted.

In Fortran 95 and Fortran 2003, overlapping character data is permitted.

Example: In Fortran 95 and Fortran 2003, assignment of data with overlapping character positions

```
CHARACTER(LEN=5) C
C='12345'
C(1:4)=C(2:5)
PRINT *,C ! output is '23455'
```

Note:

In FORTRAN66 and FORTRAN77, if the -Nf90move option is not specified, the assignment is not permitted.

6.4.2 Allocatable Assignment

When -Nalloc_assign compiler option is effective and left side of the assignment statement is an allocatable variable, diagnostic message jwd2881i-i is output at compilation time and the allocatable assignment of the Fortran 2003 standard is operated at run time.

The execution performance decreases when the -Nalloc_assign compiler option is specified for the source program of the Fortran 95 language specification.

The execution performance never decreases when the -Nnoalloc_assign compiler option is specified or the left side does not become an allocatable variable as the following example:

Example: No allocatable variable whose the section subscript

```
Before: left part is allocatable variable
```

```
INTEGER,ALLOCATABLE::ARRAY(:)
ALLOCATE(ARRAY(2))
ARRAY=[1,2]
END
```

```
After: left part is not allocatable variable
```

```
INTEGER,ALLOCATABLE::ARRAY(:)
ALLOCATE(ARRAY(2))
ARRAY(:)=[1,2]
END
```

6.5 Control Statements

This section describes considerations related to control statements.

6.5.1 GO TO Statement

If a statement label list is present and the destination of an assigned GO TO statement is a statement label other than one in the list, no error is output during compilation. However, the program may not operate correctly.

To improve execution efficiency, specify all labels of statements that can be branch targets of the assigned GO TO statement.

6.5.2 Arithmetic IF Statement

If the arithmetic expression in an arithmetic IF statement is an integer expression and fixed-point overflow occurs, the next statement executed is unpredictable. For example, if the following IF statement is executed, the statement executed next could be 10, 20, or 30.

Example:

```
I=3
IF(2147483647+I)10,20,30
```

6.5.3 Logical IF Statement

If an expression in a logical IF statement has a value that does not conform to the internal logical data form, execution is unpredictable. For more information, see Section "5.1.2 Logical Type Data", and Section "6.3.3 Logical Data Operations".

Example:

```
LOGICAL TR
EQUIVALENCE(TR,JR)
DATA JR/16/
IF(TR) CALL S
```

Note:

The CALL statement may or may not be executed.

6.5.4 DO Statement

If there is a statement that branches into the range of a DO loop from outside the range of the DO loop, the system outputs a warning diagnostic message. In this case, program operation is unpredictable.

For the extended range of a DO loop defined using FORTRAN66, the system outputs a diagnostic message as a warning, and the program operates normally. However, the program should be corrected to ensure future compatibility.

6.5.4.1 DO Loop Iteration Count Method

The following formula defines the DO loop iteration count *r7* used in FORTRAN77, Fortran 95 and Fortran 2003.

```
r7 = MAX ( INT ( ( m2 - m1 + m3 ) / m3 ) , 0 )
```

In this formula, *m1*, *m2*, and *m3* are the initial, terminal, and incrementation parameters of the DO loop. The body of the DO loop is not always executed.

The following formula defines the DO loop iteration count *r6* used in FORTRAN66.

```
r6 = MAX ( ( m2 - m1 ) / m3 + 1 , 1 )
```

For FORTRAN66, the body of the DO loop is executed at least once.

6.5.4.2 DO CONCURRENT

If a loop control of DO construct has CONCURRENT, the DO construct (DO CONCURRENT) is equal to DO construct with NORECURRENCE specifier in optimization control specifier. See Section "9.10.4 Optimization Control Specifiers" for information about NORECURRENCE specifier.

6.5.5 CASE Statement

If a case expression is an integer expression, the case value shall have the value that can be represent of case expression type and type parameter.

Example:

```
INTEGER(2)::I
...
SELECT CASE (I)
CASE (40000)          ! THE CASE VALUE HAS THE VALUE BETWEEN -32768 TO
   ...                ! 32767, BECAUSE THE CASE EXPRESSION IS OF TYPE
                      ! TWO-BYTE INTEGER
END SELECT
```

6.5.6 PAUSE Statement

The PAUSE statement writes diagnostic message jwe0001a to the standard error file. If the message is sent to the terminal and the standard input is from the terminal, execution of the program is suspended awaiting a response. Entering any standard input data at the terminal restarts the program. In cases other than the above, the PAUSE statement is ignored and the program continues.

Example: PAUSE statement diagnostic messages

```
PAUSE
   jwe0001a pause
```

```
PAUSE n
   jwe0001a pause n
```

n: A decimal number of one to five digits

```
PAUSE c
   jwe0001a pause c
```

c: A character constant enclosed with apostrophes with a length not exceeding 255

6.5.7 STOP/ERROR STOP Statement

The STOP/ERROR STOP statement causes termination of the executing program, and writes the diagnostic message as follows to the standard error output file.

Statement kind	Stop-code specification	Diagnostic message
STOP	None	-
	Yes	jwe0002i
ERRO STOP	None	jwe0003i
	Yes	

-: No message

The stop-code of the STOP/ERROR STOP statement is reflected in the return code as follows if the environment variable FLIB_USE_STOPCODE has the value 1.

Statement kind	Stop-code	Return code
STOP	(Not specified)	0
	scalar-int-initialization-expr	The least significant byte specified by a stop-code (0 to 255)
	scalar-default-char-initialization-expr	0
ERROR STOP	(Not specified)	1
	scalar-int-initialization-expr	The least significant byte specified by a stop-code (0 to 255)
	scalar-default-char-initialization-expr	1

Example: STOP/ERROR STOP statement diagnostic messages

- STOP *scalar-int-initialization-expr*

```
STOP 123456
   jwe0002i stop 123456
```

```
STOP -3 + 4
   jwe0002i stop 1
```

- STOP *scalar-default-char-initialization-expr*

```
STOP 'NORMAL'
   jwe0002i stop NORMAL
```

```
STOP ('AB'//'CD')//'EF'
   jwe0002i stop ABCDEF
```

- ERROR STOP *scalar-int-initialization-expr*

```
ERROR STOP 123456
   jwe0003i error stop 123456
```

```
ERROR STOP -3 + 2
   jwe0003i error stop -1
```

- ERROR STOP *scalar-default-char-initialization-expr*

```
ERROR STOP 'ERROR'
   jwe0003i stop ERROR
```

```
ERROR STOP ('AB'//'CD')//'EF'
   jwe0003i stop ABCDEF
```

Example: Return code from the STOP/ERROR STOP statement

Statement kind	Stop-code	Example sentence	Diagnostic message	Return code
STOP	(Not specified)	`STOP`	-	0
	scalar-int-initialization-expr	`STOP -3 + 4`	`jwe0002i stop 1`	1
		`STOP 400`	`jwe0002i stop 400`	144
	scalar-default-char-initialization-expr	`STOP 'ABC'`	`jwe0002i stop ABC`	0
		`STOP ('AB'//'CD')//'EF'`	`jwe0002i stop ABCDEF`	0
ERROR STOP	(Not specified)	`ERROR STOP`	`jwe0003i error stop`	1
	scalar-int-initialization-expr	`ERROR STOP 222`	`jwe0003i error stop 222`	222
		`ERROR STOP -3 + 2`	`jwe0003i error stop -1`	255
	scalar-default-char-initialization-expr	`ERROR STOP 'ERROR'`	`jwe0003i error stop ERROR`	1
		`ERROR STOP ('AB'//'CD')//'EF'`	`jwe0003i error stop ABCDEF`	1

-: No message

6.5.8 ALLOCATE/DEALLOCATE Statement

If the STAT= specifier is specified for an ALLOCATE or DEALLOCATE statement, the following value is returned and execution continues.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end
912	Available memory was insufficient when the ALLOCATE statement was executed.
1001	The allocatable variable specified for an ALLOCATE statement was already allocated.
1003	The allocatable variable specified for a DEALLOCATE statement was not allocated. The deallocation is ignored.
1004	The pointer specified for a DEALLOCATE statement was not associated.
1005	The pointer specified for a DEALLOCATE statement is associated with a target that has not been created by an ALLOCATE statement.

6.5.9 Simple Output List Enclosed in Parentheses

In FORTRAN66, when a simple output list that is enclosed in parentheses and contains named constant is specified, it is output as a simple output list enclosed in parentheses.

In FORTRAN77, Fortran 95 and Fortran 2003, it is output as a COMPLEX literal constant.

Example:

```
REAL A,B
PARAMETER (A=1.0,B=2.0)
WRITE (*,*) (A,B)
END
```

In FORTRAN66, two REAL literals as a simple output list enclosed in parentheses are output as follows:

```
$ ./a.out
 1.00000000 2.00000000
$
```

In FORTRAN77, Fortran 95 and Fortran 2003, a COMPLEX literal constant is output as follows:

```
$ ./a.out
 (1.00000000,2.00000000)
$
```

6.5.10 The Output Statement that starts by TYPE Keyword

The output statement that starts by TYPE keyword can be complied and executed in FORTRAN66 and FORTRAN77. In Fortran 95 and Fortran 2003, the output statement that starts by TYPE keyword is interpreted as a TYPE statement of derived type definition or a TYPE type declaration statement, and the diagnostic message at s level is output. Change the keyword TYPE to PRINT.

Example: The output statement that starts by TYPE keyword

```
A = 1.0
TYPE*, A ! The diagnostic message at s level is output in Fortran 95
             and Fortran 2003.
END
```

6.6 Procedures

This section describes various properties of procedures.

6.6.1 Statement Function Reference

If the types of actual and dummy arguments do not match in a statement function reference, the system outputs a diagnostic message. The actual argument type is used for evaluation. If an arithmetic or logical expression is used for the actual argument, however, the actual argument type must follow the expression conversion rules.

In the following example, the actual arguments are double-precision real for statement function SF. However, statement function ESF causes an error.

Example: Statement function reference

```
REAL(8) X,Y,Z
LOGICAL L,M,N,ESF
SF(I,J,K)=I**2+J**3+K**4
ESF(L,M,N)=.NOT. L .AND. M .AND. N
:
XX=3.0E1+SF(X,Y,Z)        ! Actual arguments of SF are
:                            ! double-precision real
L=X==Y .AND. ESF(X,Y,Z)  ! These arguments not allowed
                             ! by conversion rules
```

6.6.2 An Intrinsic Function that Specifies EXTERNAL Attribute

In FORTRAN77, Fortran 95 and Fortran 2003, if the EXTERNAL attribute is specified to an intrinsic function name, the intrinsic function characteristic is lost, and the name is processed as a user-defined external procedure or block data program unit.

In FORTRAN66, if a basic external function name is specified in an EXTERNAL statement, the name is processed as a basic external function.

Note:

The FORTRAN77, Fortran 95 and Fortran 2003 intrinsic function is the generic name for a FORTRAN66 basic external or intrinsic functions.

6.6.3 REAL and CMPLX Used as Generic Names

The generic function REAL converts the data to type real in FORTRAN77, Fortran 95 and Fortran 2003, but it only fetches the real part from a complex argument in FORTRAN66.

In Fortran 95 and Fortran 2003, if the first argument is of type complex and the second argument is not present in the generic function REAL, the result type parameter is the kind type parameter of the first argument.

The generic function CMPLX converts the date to type complex in FORTRAN77, Fortran 95 and Fortran 2003, but it only creates a complex number in FORTRAN66.

6.6.4 Intrinsic Procedures

In FORTRAN77, Fortran 95 and Fortran 2003, the following names are interpreted as intrinsic functions. In FORTRAN66, the same names are treated as user-defined functions and not as intrinsic functions (basic external functions).

```
ICHAR, CHAR, ANINT, DNINT, QNINT, NINT, IDNINT, IQNINT, DPROD, QPROD, LEN, INDEX, DASIN, QASIN,
DACOS, QACOS, LGE, LGT, LLE, LLT, NOT, IAND, IOR, IEOR, ISHFT, IBSET, IBCLR, BTEST
```

In Fortran 95 and Fortran 2003, the following names are interpreted as intrinsic procedures. In FORTRAN66 and FORTRAN77, the same names are treated as user-defined procedures.

```
ACHAR, ADJUSTL, ADJUSTR, ALL, ALLOCATED, ANY, ASSOCIATED, BIT_SIZE, CEILING,
COMMAND_ARGUMENT_COUNT , COUNT, CPU_TIME, CSHIFT, DATE_AND_TIME, DIGITS, DOT_PRODUCT, EOSHIFT,
EPSILON, EXPONENT, FLOOR, FRACTION, HUGE, GET_COMMAND, GET_COMMAND_ARGUMENT,
GET_ENVIRONMENT_VARIABLE, IACHAR, IBITS, ISHFTC, KIND, LBOUND, LEN_TRIM, LOGICAL, MATMUL,
MAXEXPONENT, MAXLOC, MAXVAL, MERGE, MINEXPONENT, MINLOC, MINVAL, MODULO, MOVE_ALLOC, MVBITS,
NEAREST, NEW_LINE , NULL , PACK, PRECISION, PRESENT, PRODUCT, RADIX, RANDOM_NUMBER, RANDOM_SEED,
RANGE, REPEAT, RESHAPE, RRSPACING, SAME_TYPE_AS, SCALE, SCAN, SELECTED_CHAR_KIND,
```

```
SELECTED_INT_KIND, SELECTED_REAL_KIND, SET_EXPONENT, SHAPE, SIZE, SPACING, SPREAD, SUM, TINY,
SYSTEM_CLOCK, TRANSFER, TRANSPOSE, TRIM, UBOUND, UNPACK, VERIFY
```

If the -Nobsfun compiler option is specified, the following names are interpreted as intrinsic functions.

```
AIMAX0, AJMAX0, I2MAX0, IMAX0, JMAX0, IMAX1, JMAX1, AIMIN0, AJMIN0, I2MIN0, IMIN0, JMIN0, IMIN1,
JMIN1, FLOATI, FLOATJ, DFLOTI, DFLOTJ, IIABS, JIABS, I2ABS, IIDIM, JIDIM, I2DIM, IIFIX, JIFIX,
IFIX, INT1, INT2, INT4, INT8, IINT, JINT, ININT, JNINT, IIDNNT, I2NINT, JIDNNT, IIDINT, JIDINT,
IMOD, JMOD, I2MOD, IISIGN, JISIGN, I2SIGN, BITEST, BJTEST, IIBCLR, JIBCLR, IIBITS, JIBITS, IIBSET,
JIBSET, IBCHNG, ISHA, ISHC, ISHL, IIAND, JIAND, IIEOR, JIEOR, IIOR, JIOR, INOT, JNOT, IISHFT,
JISHFT, IISHFTC, JISHFTC, IZEXT, JZEXT, IZEXT2, JZEXT2, JZEXT4, VAL
```

6.6.5 Intrinsic Function Names with the Type Declared

In FORTRAN77, Fortran 95 and Fortran 2003, even if an intrinsic function is declared with a different type, the intrinsic function characteristic is not lost.

In FORTRAN66, if an intrinsic function is declared with a different type, the intrinsic function characteristic is lost. In addition, the intrinsic function is treated as a user-defined function whenever it is used.

6.6.6 Service Routines

If compiler option -AU is specified, a service subroutine name must be entered in lowercase characters.

The validity of the number and type of arguments is checked if the module SERVICE_ROUTINES that includes all service routines explicit interface is used. There are following notes to use the SERVICE_ROUTINES:

- Specify ONLY option and service routines that only using in program because SERVICE_ROUTINES includes all service subroutines and service functions explicit interface. For example,

```
USE SERVICE_ROUTINES,ONLY:ACCESS
```

- If compiler option -AU is specified, module name SERVICE_ROUTINES and each service routine name in USE statement must be in lowercase characters.
- If compiler option -CcdII8, -CcI4I8, -CcdRR8, -CcR4R8, -CcdLL8, -CcL4L8, -Ccd4d8, -Cca4a8, -Ad or -Aq is specified, the corresponding types of argument of service routines is changed. Specify the correspondence runtime option -Lb, -Li, or -Lr.

The validity of the number and type of arguments is not checked if the module SERVICE_ROUTINES is not used. For about module and use a module, see "Chapter 10 Fortran Modules".

If the -Nmallocfree compiler option is specified, the MALLOC and the FREE service routines are interpreted as intrinsic procedures.

If the compiler option -NRtrap and the runtime option -Wl,-i or the environment variable FLIB_EXCEPT=u are specified, DVCHK service subroutine and OVERFL service subroutine return the value which shows that an exception doesn't occur.

6.6.7 The returned values of the intrinsic function

When negative zero is specified for the argument, the following cases become the different returned value between Fortran2003 and Fortran95 or before.

- ATAN2(Y,X) : When X < 0 and Y is negative zero, the returned value is -PI in Fortran2003 and PI in Fortran95 or before.
- ATAN2D(Y,X) : When X < 0 and Y is negative zero, the returned value is -180.0 in Fortran2003 and 180.0 in Fortran95 or before.
- ATAN2Q(Y,X) : When X < 0 and Y is negative zero, the returned value is -2.0 in Fortran2003 and 2.0 in Fortran95 or before.
- LOG(X) : When X is complex with REAL(X) < 0 and the imaginary part of X is negative zero, the imaginary part of the returned value is -PI in Fortran2003 and PI in Fortran95 or before.
- SQRT(X) : When X is complex with REAL(X) < 0 and the imaginary part of X is negative zero, the imaginary part of the returned value is negative value in Fortran2003 and positive value in Fortran95 or before.

Chapter 7 Input/output Processing

This section describes the basic properties of Fortran records and files and the Fortran input/output statements used to process files. Features that are ISO standard Fortran are not discussed in general, except to describe their specific characteristics in the Fujitsu Fortran system. Unless indicated otherwise, "file" means external file in this section.

7.1 Files

Input/output statements process internal and external files. Internal files are stored in main memory.

Internal files consist of scalar character variables, character array elements, character arrays, or substrings.

Internal input/output statements may read or write internal files. External files are stored on external devices and are connected for sequential, direct, and stream access.

Fortran programs process the following types of files:

- Standard input files (stdin)
- Standard output files (stdout)
- Standard error output files (stderr)
- Regular files

Files other than standard files are called regular files. Either sequential, direct, or stream access methods can be used to access regular files. Only the sequential access method can be used to access standard files. Unformatted input/output statements may not be used to access standard files.

7.1.1 Old and New Files

New files may be created when an executable program is started or at any time during its execution.

Old and new files may be accessed when connected to a unit number specified in an input/output statement.

7.1.2 File Connection

A file and unit must be connected to execute a data transfer or file positioning input/output statement.

7.1.2.1 Connection

Files are connected in two ways:

- Dynamic connection (during execution of a program)
- Preconnection

Dynamic connection is performed when an OPEN, READ, WRITE, BACKSPACE, ENDFILE, or PRINT statement is executed.

However, dynamic connection for direct or stream access may be performed only by an OPEN statement.

Preconnection is performed by an external specification for a Fortran program before an executable program is started.

The standard input file (stdin, unit 5), the standard output file (stdout, unit 6), and the standard error output files (stderr, unit 0) are always preconnected. Executing an OPEN statement with a FILE= specifier disconnects a standard file.

7.1.2.2 File disconnection

Files are disconnected in the following cases.

- Execution of an CLOSE statement
- Termination of the execution program
- Execution of an OPEN statement with different file name to the same unit number.

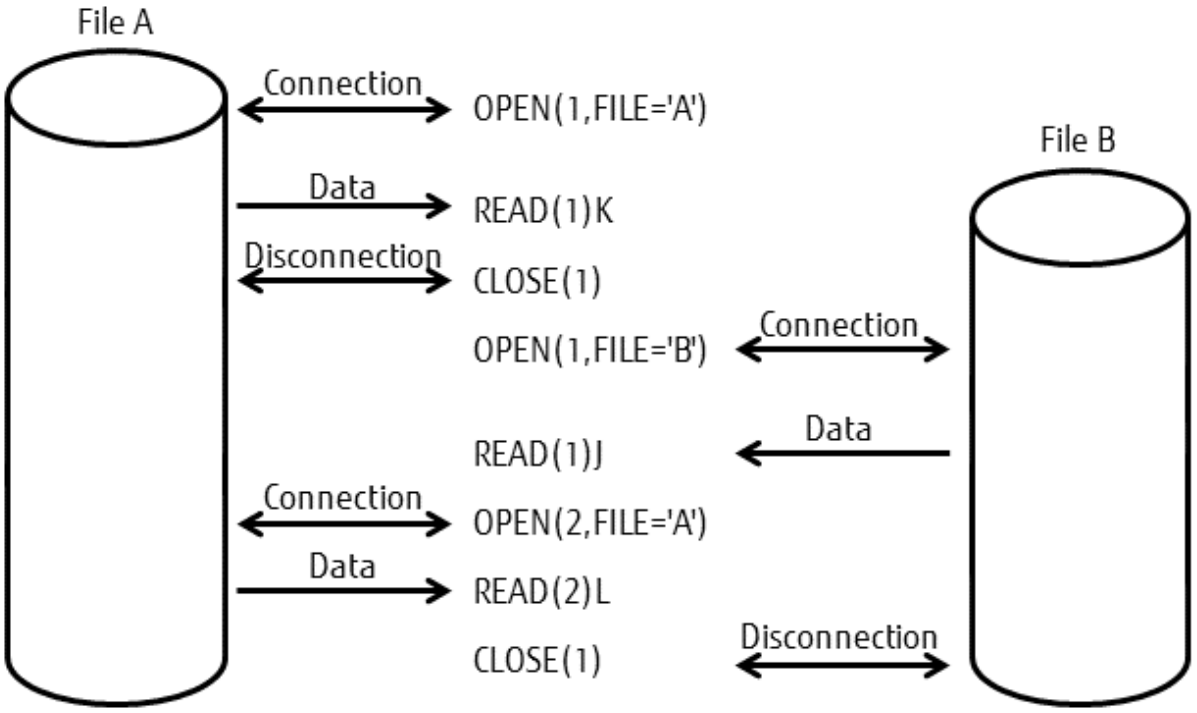
The following processes are done in the disconnection of the file.

- Termination of the connection of specified unit to an external file
- Deletion or keeping of the file that is connected to the specified unit

A file that has been disconnected may be connected by an OPEN statement.

The only means of connecting a file that has been disconnected is by appearance of its name in an OPEN statement.

There may be no means if reconnecting an unnamed file once it is disconnected.



7.2 Unit Numbers and File Connection

Units and files are connected as follows:

- Execute an OPEN statement with a filename specified.
- Use a environment variable outside the program to pre-connect a unit and file.

7.2.1 Priority of Unit and File Connection

The way of connecting it in the case of standard input, standard output and the standard error output file and that order of priority are shown in the following.

How to specify	Explanation	Priority
An OPEN statement with FILE=specifier	Connection by an OPEN statement with FILE=specifier	High
Runtime options	Connection by the runtime options. -p*u_no* (standard output) -r*u_no* (standard input) -m*u_no* (standard error output)	↕
Default value of Fortran system	Connection by Fortran System. Standard output (unit number 6)	Low

How to specify	Explanation	Priority
	Standard input (unit number 5) Standard error output (unit number 0)	

The way of connecting it and that order of priority are shown in the following in the case as the named file and the unnamed file.

How to specify	Explanation	Priority
An OPEN statement with FILE=specifier	Connection by an OPEN statement with FILE=specifier	High
The environment variable	Connection by the environment variable (fu*xx*).	Low

7.2.2 Environment variables to Connect Units and Files

Environment variables may be used to connect units and files. How to specify file name by using an environment variable is shown in the following.

Name of Environment variable	Value
fu*xx*	*File-name*

fu: Fixed

xx: Unit number (00 to 2147483647)

File-name: Filename (full) pathname or filename

Unit numbers 0, 5, and 6 (set by runtime options -m, -r, and -p) are preconnected for error output, standard input, and standard output, respectively. These unit numbers may not be connected using a environment variable.

Environment variable file connections are disabled when a file is closed and remain disabled thereafter. The input/output statements that may close any of the standard files are as follows:

- CLOSE statement
- OPEN statement with a file specified

Example 1: Connect "test.data" to unit number 1

Environment variable setting:

```
$export fu01="test.data"
```

Example 2: Disabled connect by environment variable

Fortran program:

```
OPEN(1,FILE='b.file')
   ...
```

Commands:

```
$ export fu01=a.file
$ ./a.out
```

When a.out is executed, file connection by environment variable (connection to a.file) is disabled, and the filename specified an OPEN statement (connection to b.file) is enabled.

7.3 Records

Fortran records contain data in internal machine format or character strings. One Fortran record does not always correspond to one physical record. The Fortran record types are as follows:

- Formatted (see Section 7.3.1 Formatted Fortran Records)

- Unformatted (see Section 7.3.2 Unformatted Fortran Records)
- List-Directed (see Section 7.3.3 List-Directed Fortran Records)
- Namelist (see Section 7.3.4 Namelist Fortran Records)
- Endfile (see Section 7.3.5 Endfile Records)
- Binary (see Section 7.3.6 Binary Fortran Records)
- Stream (see Section 7.3.7 STREAM Fortran Records)

Fortran record characteristics are shown below.

Fortran record type	Input/output statement	Record format	One Fortran record
Formatted	Formatted sequential	Undefined length	One logical record
	Formatted direct	Fixed length	
	Internal file		One scalar character variable, array element, or substring. For an array, each element is a Fortran record.
Unformatted	Unformatted sequential	Variable length	One logical record
	Unformatted direct	Fixed length	
List-directed	List-directed, Print statement	Undefined length	One logical record
	Internal file	Fixed length	One scalar character variable, array element, or substring. For an array, each element is a Fortran record.
Namelist	Namelist	Undefined length	One logical record from the &namelist name up to / or &end.
	Internal file	Fixed length	One scalar character variable, array element, or substring. For an array, each element is a Fortran record.
Endfile	Input/output statement other than direct	----	The last Fortran record does not have the length attribute.
Binary	Unformatted sequential	Fixed length(*1)	One logical record
	Unformatted direct	Fixed length(*1)	
Stream	formatted stream	Undefined length	One logical record
	Unformatted stream	Fixed length (*1)	One logical record

*1. The Fortran record length is determined according to the size of data for which input/output is to be executed.

Fortran records exist as logical records on external devices. An undefined length record consists of a Fortran record and the line-feed character (\n). A fixed length record consists of a Fortran record. A variable length record consists of a Fortran record and 8 bytes (a 4-byte length before and after the Fortran record).

7.3.1 Formatted Fortran Records

Formatted Fortran records can consist of any character string. Formatted Fortran records are accessed by formatted sequential, formatted direct, and internal file input/output statements.

7.3.1.1 Formatted Sequential Record

Files controlled by formatted sequential input/output statements have an undefined length record format.

One Fortran record corresponds to one logical record. The length of the undefined length record depends on the Fortran record to be processed. The max length may be assigned in OPEN statement RECL= specifier.

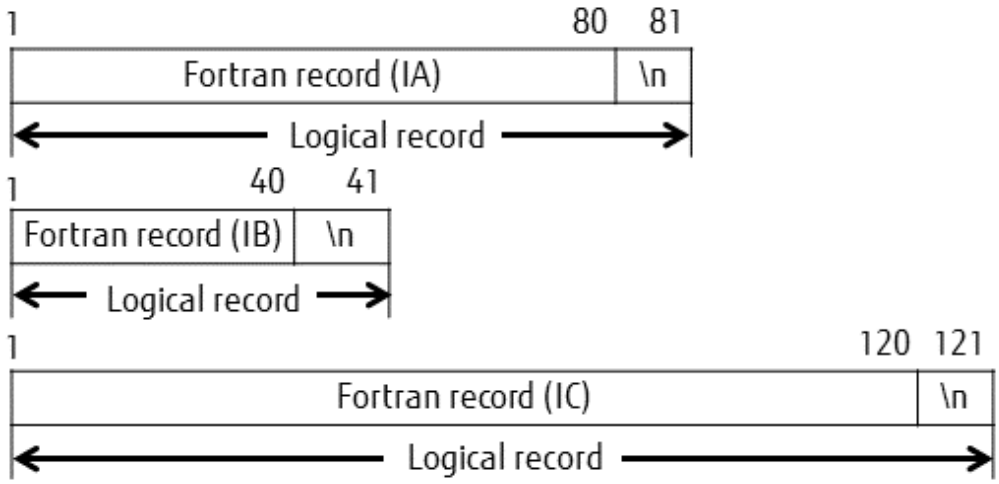
The line-feed character (\n) terminates the logical record. If the $ edit descriptor or \ edit descriptor is specified for the format of the format of the formatted sequential output statement, the Fortran record does not include the line feed character (\n).

The relationship between formatted Fortran records and the undefined length record format is shown below.

Fortran program:

```
      INTEGER,DIMENSION(20) :: IA
      INTEGER,DIMENSION(10) :: IB
      INTEGER,DIMENSION(30) :: IC
      WRITE(2,10) IA,IB
10    FORMAT(20I4,/,10I4)
      WRITE(2,20) IC
20    FORMAT(30I4)
```

Fortran records:



7.3.1.2 Formatted Direct Record

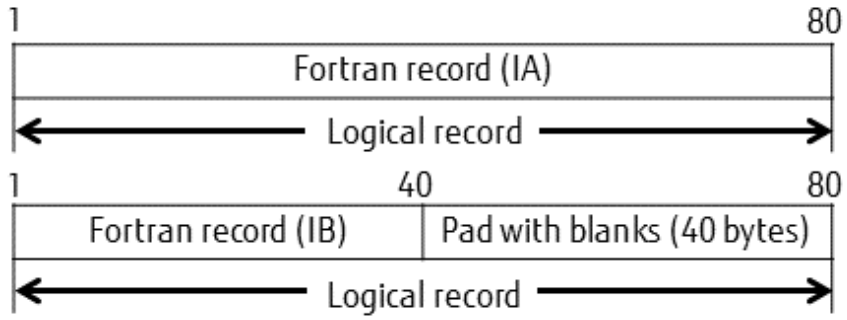
Files processed by formatted direct input/output statements have a fixed length record format. One Fortran record corresponds to one logical record. The length of the logical record must be assigned in the OPEN statement RECL= specifier. If the Fortran record is shorter than the logical record, the remaining part is padded with blanks. The length of the Fortran record must not exceed the logical record. This fixed length record format is unique to Fortran.

The relationship between formatted Fortran records and the fixed length record format is shown below.

Fortran program:

```
     INTEGER,DIMENSION(20) :: IA
     INTEGER,DIMENSION(10) :: IB
     OPEN(UNIT=3,ACCESS='DIRECT',RECL=80,FORM='FORMATTED')
     WRITE(3,FMT=10,REC=1) IA,IB
10   FORMAT(20I4)
```

Fortran record:



7.3.1.3 Internal File Record

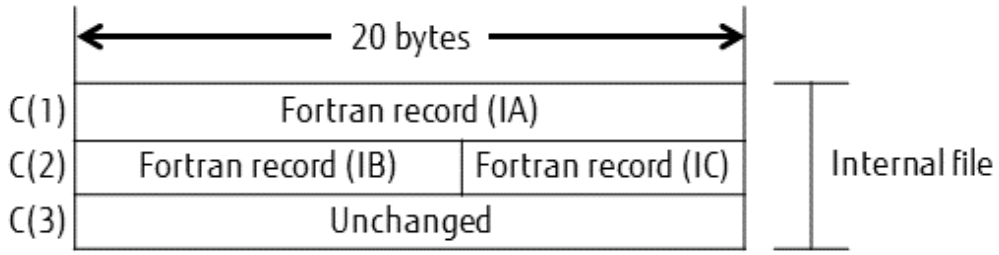
The fixed length record are processed by internal file input/output statements. The records are stored in internal files. Internal files that are a scalar character variable, character array element or substring contain only one Fortran record. The length of the Fortran record must not exceed the size of the scalar character variable, character array element, or substring. When an internal file is a character array, the length of a Fortran record must not exceed the size of an array element of the character array. If the length of the Fortran record is shorter than the size of the scalar character variable, character array element or substring, the remaining part is padded with blanks. For input, the lengths must be equal.

The relationship between formatted Fortran records and internal files is shown below.

Fortran program:

```
      CHARACTER(LEN=20) C(3)
      INTEGER,DIMENSION(5) :: IA
      INTEGER,DIMENSION(3) :: IB
      INTEGER,DIMENSION(2) :: IC
      WRITE(C,10) IA,IB,IC
10    FORMAT(5I4)
```

Fortran record (Internal file: C):



7.3.2 Unformatted Fortran Records

An unformatted Fortran record consists of a string of values (character or noncharacter data, and a string can be empty). The length depends on the input/output source program statement (zero length can be used).

Both unformatted sequential and unformatted direct input/output statements can use unformatted Fortran records.

7.3.2.1 Unformatted Sequential Record

Files processed using unformatted sequential input/output statements have a variable length record format.

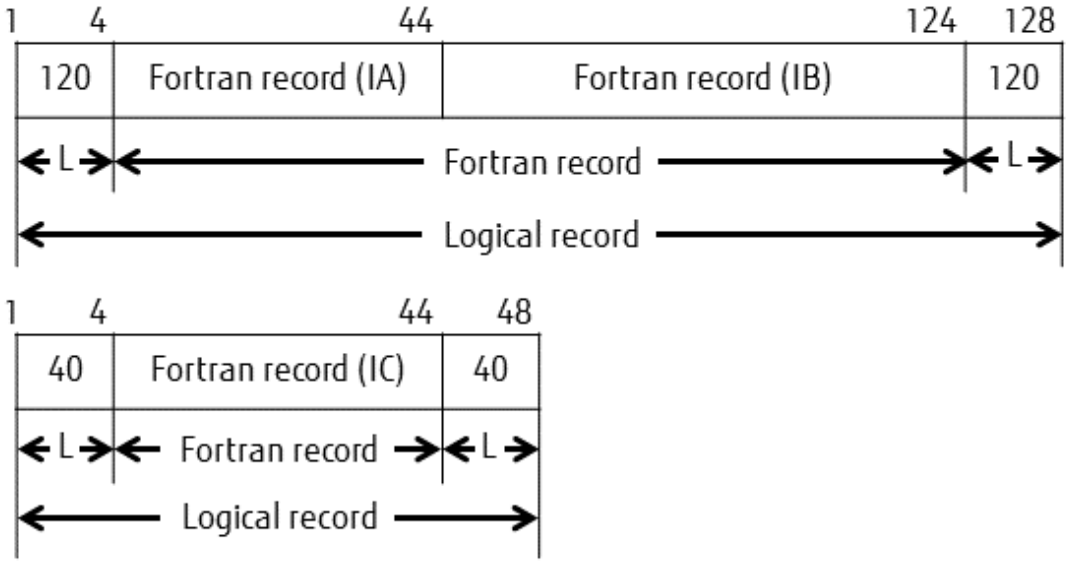
One Fortran record corresponds to one logical record. The length of the variable length record depends on the length of the Fortran record. The length of the Fortran record includes 4 bytes added to the beginning and end of the logical record. If the runtime option -Wl,-Lu is specified, the size of the field to put the length has 8 bytes. The max length may be assigned in the OPEN statement RECL= specifier. The beginning area is used when an unformatted sequential statement is executed. The end area is used when a BACKSPACE statement

is executed. This variable length record format is unique to Fortran. The relationship between unformatted Fortran records and the variable length record format is shown below.

Fortran program:

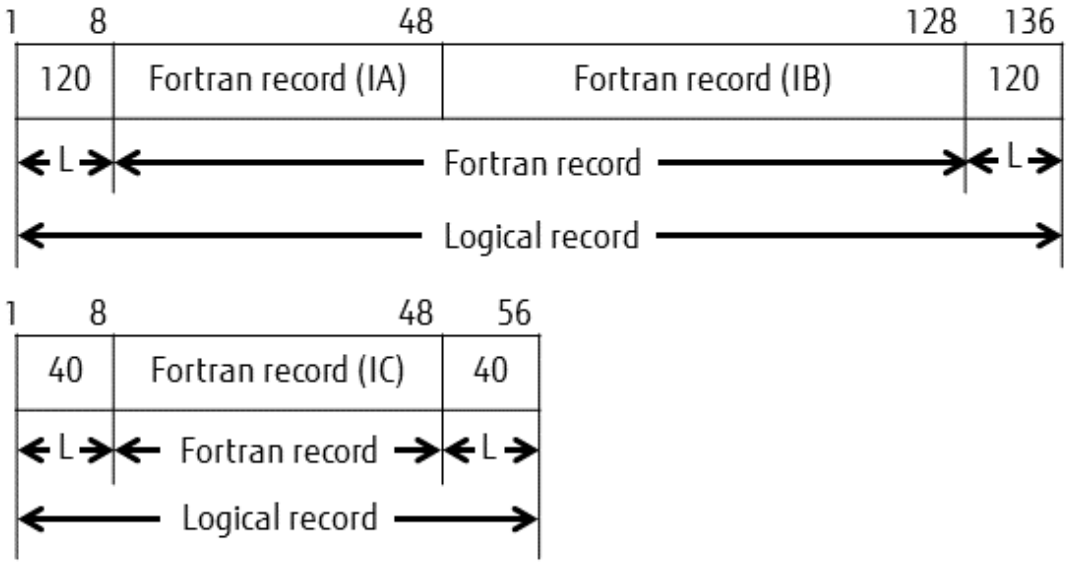
```
INTEGER,DIMENSION(10) :: IA,IC
INTEGER,DIMENSION(20) :: IB
WRITE(2) IA,IB
WRITE(2) IC
```

Fortran record (the runtime option -Wl,-Lu is not specified):



L: Length of Fortran record

Fortran record (the compiler option is valid and the runtime option -Wl,-Lu is specified):



L: Length of Fortran record

The Fortran record that length is 2G bytes (2147483648 bytes) or more is input/output by dividing the record. The length of each divided Fortran record is from 1 to 2147483647 bytes.

The value set to the top and the end of a logical record is as follows because the divided Fortran records are considered single Fortran record.

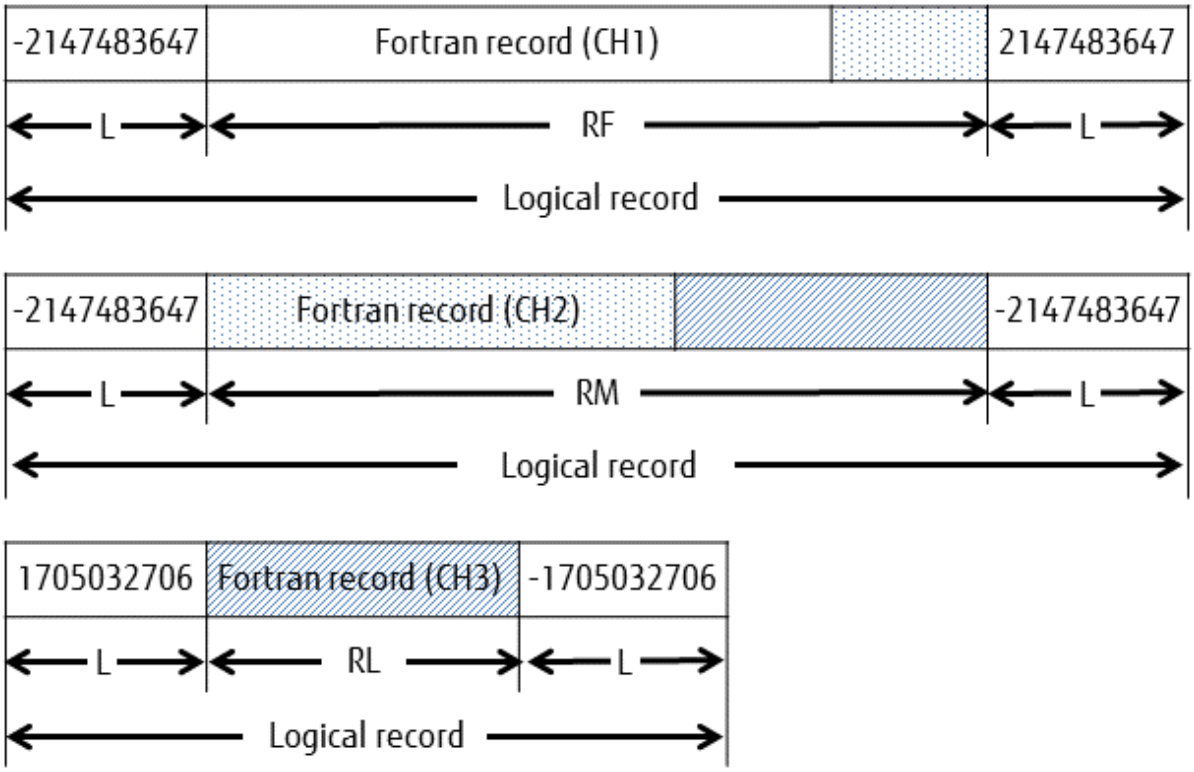
Divided Fortran record	The value of top of logical record	The value of end of logical record
First	A negative value that the absolute value is equal to the length of the divided Fortran record is set. (-2147483647 is set)	The length of the divided Fortran record is set. (2147483647 is set)
Middle	A negative value that the absolute value is equal to the length of the divided Fortran record is set. (-2147483647 is set)	A negative value that the absolute value is equal to the length of the divided Fortran record is set. (-2147483647 is set)
Last	The length of the divided Fortran record is set.	A negative value that the absolute value is equal to the length of the divided Fortran record is set.

The relationship between unformatted Fortran records that length is 2G bytes or more and the variable length record format is shown below.

Fortran program:

```
CHARACTER(LEN=2000000000) :: CH1,CH2,CH3
OPEN(10,FILE="work.data",FORM="UNFORMATTED")
WRITE(10) CH1,CH2,CH3
```

Fortran record:



L: Length of Fortran record
RF: Divided Fortran record (First)
RM: Divided Fortran record (Middle)
RL: Divided Fortran record (Last)

7.3.2.2 Unformatted Direct Record

Files processed by unformatted direct input/output statements have a fixed length record format. One Fortran record can correspond to more than one logical record. See Section 7.3.1.2 Formatted Direct Record for details.

The record length must be assigned in the OPEN statement RECL= specifier. One Fortran record can consist of more than one logical record. If the Fortran record terminates within a logical record, the remaining part is padded with binary zeros. If the length of the Fortran record exceeds the logical record, the remaining data goes into the next record. This fixed length record format is unique to Fortran.

7.3.3 List-Directed Fortran Records

List-directed input/output statements, print statement, or internal file input/output statements are used to process list-directed Fortran records. List-directed Fortran records consist of data items and value separators. The data items are input/output character strings. The length of the Fortran record depends on the number and type of items. Data items from the beginning of a logical record up to the end of the input are handled as one Fortran record. Files processed using list-directed input/output statements have an undefined length record format or a fixed length record format. See Section 7.3.1.1 Formatted Sequential Record for details. See Section 7.3.1.3 Internal File Record for details.

7.3.4 Namelist Fortran Records

Namelist input/output statements and internal input/output statements access namelist Fortran records.

Namelist Fortran records consist of data items (character strings specified by a namelist name) from the &namelist name up to / or &END. The correspondence between namelist Fortran records and logical records is the same as for Fortran records handled using list-directed input/output statements. Files accessed by namelist input/output statements have an undefined length record format. See Section 7.3.1.1 Formatted Sequential Record for details. See Section 7.3.1.3 Internal File Record for details.

7.3.5 Endfile Records

An endfile record must be the last record of a file connected for sequential access. Endfile records do not have a length attribute.

The ENDFILE statement writes endfile records in files connected for sequential access. After at least one WRITE statements is executed, endfile records are output under the following conditions:

- A REWIND statement is executed.
- A BACKSPACE statement is executed.
- A CLOSE statement is executed.

7.3.6 Binary Fortran Records

A binary Fortran record consists of a string of values (character or noncharacter data). The length of the binary Fortran record depends on the input/output list (it may be zero length). One Fortran record corresponds to one logical record. An unformatted sequential access or an unformatted direct access input/output statement may access binary Fortran records. See Section 7.3.1.2 Formatted Direct Record for details.

7.3.7 STREAM Fortran Records

A STREAM Fortran record is composed of the file storage units, and its storage position can be uniquely identified using positive integers.

STREAM Fortran records can be accessed by stream input/output statements.

7.3.7.1 Formatted Stream Record

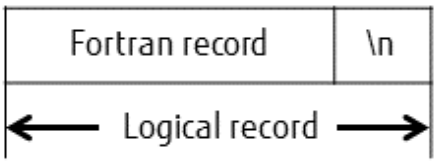
Files controlled by formatted stream input/output statements have an undefined length record format.

One Fortran record corresponds to one logical record. The length of the undefined length record depends on the Fortran record to be processed. The line-feed character (\n) terminates the logical record. If the $ edit descriptor or \ edit descriptor is specified for the format of the formatted stream output statement, the Fortran record does not include the line feed character (\n).

Fortran program:

```
OPEN(10, FILE='x', ACCESS='stream', FORM='formatted')
DO I=1,10
   INQUIRE(10,POS=N)
   WRITE(10,'(I4)',POS=N) I
ENDDO
CLOSE(10)
```

Fortran record:



7.3.7.2 Unformatted Stream Record

Files controlled by unformatted stream input/output statements have a fixed length record format.

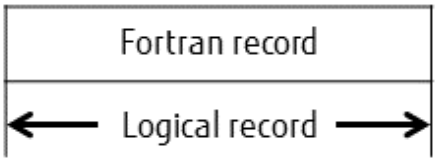
The length of the undefined length record depends on the Fortran record to be processed.

The relationship between unformatted Fortran records and the fixed length record format is shown below.

Fortran program:

```
OPEN(10, FILE='x', ACCESS='stream', FORM='unformatted')
DO I=1,10
WRITE(10,POS=4*I ) I
ENDDO
CLOSE(10)
```

Fortran record:



7.4 OPEN Statement

7.4.1 OPEN Statement Specifiers

These topics explain the parameters that can be specified in an OPEN statement. The parameters are:

- FILE= Specifier
- STATUS= Specifier
- RECL= Specifier
- ACTION= Specifier
- BLOCKSIZE= Specifier
- CONVERT= Specifier
- CARRIAGECONTROL= Specifier

The character strings that can be specified with STATUS= , ACCESS= , FORM= , BLANK= , ACTION= , PAD= , POSITION= , DELIM= , ASYNCHRONOUS= , DECIMAL= , ROUND= , SIGN= , CARRIAGECONTROL= , and CONVERT= specifiers depend on the parameter.

The characters in the specifier may be any mixture of upper and lower case; however, ignoring case, they must match a legal specifier exactly. Blanks at the beginning are ignored. If the result is not an allowed value, diagnostic message jwe0086i-e is output, and the parameter is ignored. A specifier may contain trailing blanks.

Example values for the OPEN statement ACCESS= specifier are shown below. The other parameters are handled similarly.

Example:

Correct character expressions are:

```
OPEN(10,ACCESS='SEQUENTIAL')
OPEN(10,ACCESS='sequential')
OPEN(10,ACCESS='SeQuEnTiAl')
OPEN(10,ACCESS='SEQUENTIAL     ')
OPEN(10,ACCESS='     SEQUENTIAL')
```

Incorrect character expressions are:

```
OPEN(10,ACCESS='SEQUENTIALAB')
OPEN(10,ACCESS='SEQUEN')
OPEN(10,ACCESS='S EQUENTIAL')
OPEN(10,ACCESS='             ')
```

7.4.1.1 FILE= Specifier

The FILE= specifier specifies a file. Blanks at the beginning of the FILE= specifier are ignored. If a blank is detected in the character string, the value of the string up to the blank is read.

Filename specified in a FILE= specifier is shown below, where it is assumed that the current directory is /home/v1954.

Interpreted as /home/v1954/file.data1 are:

```
OPEN(10,FILE='/home/v1954/file.data1')
OPEN(10,FILE='file.data1')
OPEN(10,FILE=' file.data1')
```

Interpreted as /home/c1121/file.data1 is:

```
OPEN(10,FILE='../c1121/file.data1')
```

Filename is specified with the FILE= specifier and the STATUS= specifier. The tables below list the relationship between FILE= and STATUS= specifiers.

FILE= specifier	STATUS= specifier		
	NEW, OLD, SHR. REPLACE	SCRATCH	UNKNOWN
FILE= specifier	The system treats the specified value as a named file.	An Error.	The system treats the specified value as NEW or OLD, depending on the file status.
No FILE= specifier (no environment variable)	The system treats the specification as a named file with file name fort.*nn* (*nn*: unit number).	The system treats the specification as an unnamed file.	The system treats the specification as a named file with file name fort.*nn* (*nn*: unit number).
No FILE= specifier (environment variable allocated)	The system treats the file name specified by the shell variable as a named file.	An Error.	The system treats the file name specified by the environment variable as a named file.

Notes:

- If a FILE= specifier is not specified, processing depends on whether the units and file are connected by an environment variable (See Section 7.2 Unit Numbers and File Connection).

- If a name is not given, the name used is 'tF*nnn*', where *nnn* is a unique number. See "tempnam(3s)".

7.4.1.2 STATUS= Specifier

The STATUS= specifier must correspond to the file status. That is, NEW or SCRATCH must not be specified for an existing file, and if OLD or SHR is specified, the file must exist. When UNKNOWN is specified, the Fortran system determines the file status. If the STATUS= specifier is omitted, UNKNOWN is used by default.

If UNKNOWN and the FILE= specifier are specified, and the file does not already exist, the status is defined as NEW. If UNKNOWN and the FILE= specifier are specified, and the file does exist, the status is defined as OLD. If UNKNOWN and the environment variable are specified, and a FILE= specifier is not specified, the status is defined as OLD. If UNKNOWN is specified, and a FILE=specifier and the environment variable are not specified, the status is defined as SCRATCH. The FILE= specifier may be present when STATUS is SCRATCH.

The tables below list the relationship between file and STATUS= specifier.

SHR is the same as OLD. If SHR is specified, the file must exist, and a FILE= specifier must be specified at the same time. When SHR is specified, the connected file can be shared with other executable programs.

Relationship between File and STATUS= specifier

STATUS= specifier	File
NEW	Specified for a file that does not exist.
OLD, SHR	Specified for a file that exists.
SCRATCH	Specified for a file that does not exist. The file is deleted when the program is finished or the file is closed.
UNKNOWN	Specified if the file status is unknown. The Fortran system determines the status as follows: - If the FILE= specifier is specified, the status is NEW if the file does not exist. The status is OLD if the file exists. - If no FILE= specifier is specified, the status is SCRATCH if the file was not assigned using an environment variable. The status is OLD if the file was assigned using an environment variable.
REPLACE	If the file does exist, the file is deleted, and a new file is created with the same name.

7.4.1.3 RECL= Specifier

The RECL= specifier specifies the length of each Fortran record in files in which DIRECT is specified by the ACCESS= specifier. An OPEN statement with RECL= specified must be executed before any direct input/output statements are executed for that file. The value specified for an existing file must be the same as for each record in the file.

The RECL= specifier indicates the maximum length of a Fortran record in files with SEQUENTIAL specified by the ACCESS= specifier. If the RECL= specifier is omitted, 2147483647 is used by default.

The value of the RECL= specifier must be positive.

The RECL= specifier must not be specified for a file connected for stream access.

The tables below list the relationship between ACCESS= specifier and RECL= specifier.

Relationship between ACCESS= specifier and RECL= specifier

ACCESS= specifier	with RECL= specifier		without RECL= specifier
	in Fortran95 and Fortran 2003	in FORTRAN77 and FORTRAN66	
SEQUENTIAL	Sequential access	Sequential access (*1)	Sequential access
DIRECT	Direct access	Direct access	Error
STREAM	Error	Error	Stream access
None	Sequential access	Direct access (*2)	Sequential access

*1: The diagnostic message (jwe0097i-w) is generated, and RECL= specifier is ignored.

*2: The compiler message (jwd1449i-i) is generated.

7.4.1.4 ACTION= Specifier

When READ is specified, "r" is the permission of the file; that is, WRITE and PRINT statements may not be executed for the file.

When WRITE is specified, "w" is the permission of the file; that is, READ, BACKSPACE, and REWIND statements cannot be executed for the file. For direct access, READWRITE or BOTH is assumed even if WRITE is specified.

BOTH is the same as READWRITE.

When READWRITE or BOTH is specified, "r+" or "w+" is the permission of the file; that is, all input/output is allowed for the file. When a file with read only permission is opened, READ must be specified. Therefore, when the WRITE statement is executed, a diagnostic message is output even if READWRITE or BOTH is specified. If the ACTION= specifier is omitted, READWRITE is used by default.

The tables below list the relationship between ACTION= specifier and file status.

Relationship between ACTION= specifier and Files

Access method	File STATUS	ACTION=specifier	Open mode	Notes
Sequential	NEW	READ	Error	
		WRITE	w	
		READWRITE or BOTH	w+	
	OLD, SHR	READ	r	
		WRITE	w	
		READWRITE or BOTH	r+ (*1)	
	SCRATCH	READ	r	ACTION='READ' is not allowed for temporary files and new files.
		WRITE	w	
		READWRITE or BOTH	w+ (*1)	
	REPLACE	READ	r	
		WRITE	w+	
		READWRITE or BOTH	W+ (*2)	
Direct	NEW	READ	Error	
		WRITE	w	
		READWRITE or BOTH	w+	
	OLD, SHR	READ	r	

Access method	File STATUS	ACTION=specifier	Open mode	Notes
		WRITE	r+	
		READWRITE or BOTH	r+ (*2)	
	SCRATCH	READ	r	ACTION='READ' is not allowed for temporary files and new files.
		WRITE	w+	
		READWRITE or BOTH	r+ (*2)	
	REPLACE	READ	r	
		WRITE	w+	
		READWRITE or BOTH	w+ (*2)	
Stream	NEW	READ	Error	
		WRITE	w	
		READWRITE or BOTH	w+	
	OLD, SHR	READ	r	
		WRITE	w	
		READWRITE or BOTH	r+ (*1)	
	SCRATCH	READ	r	ACTION='READ' is not allowed for temporary files and new files.
		WRITE	w	
		READWRITE or BOTH	w+ (*1)	
	REPLACE	READ	r	
		WRITE	w	
		READWRITE or BOTH	w+ (*2)	

*1: A file with the read-only attribute is opened in 'r' mode.

*2: A file with the read-only attribute causes an error.

7.4.1.5 BLOCKSIZE= Specifier

The BLOCKSIZE= specifier specifies the sequential input/output buffer size in bytes. The default buffer size is 8M bytes.

7.4.1.6 CONVERT= Specifier

When LITTLE ENDIAN or IBM is specified in the CONVERT= specifier, no asynchronous data transfer is performed when an asynchronous input/output statement is executed on that unit.

7.4.1.7 CARRIAGECONTROL= Specifier

When STREAM has been specified in the ACCESS= specifier, FORTRAN cannot be specified in the CARRIAGECONTROL= specifier.

7.4.2 Executing an OPEN Statement on a Connected File

If the file connected to a unit is the same as the file to be connected, only BLANK=, DELIM=, PAD=, DECIMAL=, ROUND=, SIGN=, ERR=, and IOSTAT= specifiers may have values different from those currently in effect. If the other specifiers have different value, the

specified value is ignored in FORTRAN66 and FORTRAN77. In Fortran 95 and Fortran 2003, the runtime message (jwe1115i-w) is output and the specified value is ignored.

In addition, if the value of the STATUS= specifier is other than OLD, in the Fortran 2003 language specifications the diagnostic message jwe1115i-w is output, and the status of the external device that was connected immediately beforehand takes effect.

Example:

```
$ cat ap.f
      OPEN(10,FILE='XX.DAT',POSITION='APPEND')
      WRITE(10,*)123
      OPEN(10,FILE='XX.DAT',POSITION='REWIND')
      WRITE(10,*)456
      END
```

In FORTRAN66 and FORTRAN77:

```
$ ./a.out
$ more XX.DAT
 123
 456
$
```

In Fortran95 and Fortran 2003:

```
$ ./a.out
jwe1115i-w line 7 The value of POSITION specifier in an OPEN statement is different from that
currently in effect (unit=10).
 error occurs at MAIN__   line 7 loc 0000000000400b51 offset 0000000000000041
 MAIN__       at loc 0000000000400b10 called from o.s.
taken to (standard) corrective action, execution continuing.
error summary (Fortran)
error number  error level  error count
  jwe1115i         w           1
total error count = 1
$ more XX.DAT
 123
 456
$
```

7.4.3 Input/output of Fortran Records

If an input/output statement is executed that is inconsistent with the file properties of the connection (open), the specifications in the input/output statement are used in FORTRAN66 and FORTRAN77. In Fortran 95 and Fortran 2003, the runtime message (jwe0113i-e) is output and the input/output statement is ignored.

Example:

```
$ cat fm.f
      OPEN(10,FILE='XX.DAT',FORM='UNFORMATTED')
      WRITE(10,*)123
      END
```

In FORTRAN66 and FORTRAN77:

```
$ ./a.out
$ more XX.DAT
 123
$
```

In Fortran 95 and Fortran 2003:

```
$ ./a.out
jwe0113i-e line 2 A(an) formatted I/O statement cannot be executed for a unit connected to
unformatted (unit=10).
```

```
 error occurs at MAIN__   line 2 loc 0000000000400b42 offset 0000000000000032
 MAIN__       at loc 0000000000400b10 called from o.s.
taken to (standard) corrective action, execution continuing.
error summary (Fortran)
error number  error level  error count
  jwe0113i         e           1
total error count = 1
$ more XX.DAT
$
```

7.4.4 Default Value of ACCESS= Specifier with RECL=Specifier

In FORTRAN66 and FORTRAN77, if RECL= is specified and ACCESS= is not specified in an OPEN statement, ACCESS='DIRECT' is used by default. In Fortran 95 and Fortran 2003, if RECL= is specified and ACCESS= is not, ACCESS='SEQUENTIAL' is used by default.

Example: OPEN statement with RECL= and with no ACCESS= specifier:

```
OPEN (1, RECL=80)
```

7.5 CLOSE Statement STATUS= Specifier

This section explains the STATUS= specifier used in the CLOSE statement. See Section 7.7.1 Control Information for Input/output Data Transfer Statements for information on the other specifiers.

7.5.1 STATUS= Specifier

The STATUS= specifier indicates whether to keep or delete the file connected to the unit number when it is closed. This specifier and the OPEN statement STATUS= specifier determine whether the file is retained or deleted. The table below lists the relationship between STATUS= specifiers of OPEN and CLOSE statements.

Relationship between STATUS= specifiers of OPEN and CLOSE statements

OPEN statement	CLOSE statement		
	KEEP/FSYNC	DELETE	Default specification
NEW	Keep the file after CLOSE statement execution.	Delete the file after CLOSE statement execution.	Keep the file after CLOSE statement execution.
OLD, SHR, REPLACE			
SCRATCH	Delete an unnamed file after CLOSE statement execution.		Delete the file after CLOSE statement execution.

Note:

If UNKNOWN is specified in the OPEN statement, the file status is based on whether the file is handled as NEW, OLD, or SCRATCH.

In case the 'FSYNC' is specified to the CLOSE statement, the data which is in memory for the file is synchronized with storage device by invoking the fsync(2) system call only if the following conditions are met:

1. The connected file is a named regular file.
2. The file listed in 1. above exists.
3. The file listed in 1. above is being used.
4. File system supports synchronous operation by invoking the fsync(2) system call.

If the condition listed in 1. above is not met, an error occurs when FSYNC is specified to CLOSE statement.

If the condition listed in 2. or 3. above is not met, the fsync(2) system call is not invoked, and processing is continued.

If the condition listed in 4. is not met, an error occurs.

7.6 INQUIRE Statement

7.6.1 INQUIRE Statement Parameters

The following table indicates the way the system treats returned INQUIRE values, depending on whether the length of the returned character value is less than the scalar character variable name or array element name to be set.

Character Strings:

Returned length ≥ Character variable length		Returned length < Character variable length	
NAME= specifier	Other specifier	NAME= specifier	Other specifier
The returned value is padded with blanks to fill the character variable.		A message is output indicating that the length of the character variable is too short. The character variable is set to all blanks.	A message is output indicating that the length of the character variable is too short. The returned value truncated to the length of the character variable.

The tables below list the returned values for the INQUIRE statement.

Returned values when a filename is specified

Inquiry specifier	File exists		File does not exist
	Being used (*1)	Not being used (*2)	
EXIST	True	True	False
OPENED	True	False	False
NUMBER	o	-1	-1
NAMED	True	True	True
NAME	o	o	o
FLEN	o	0	0
ACCESS	o	"UNDEFINED"	"UNDEFINED" (*8)
SEQUENTIAL	o	"UNKNOWN"	"UNKNOWN" (*8)
DIRECT	o	"UNKNOWN"	"UNKNOWN" (*8)
ACTION	o	"UNDEFINED"	"UNDEFINED" (*8)
READ	o	"UNKNOWN"	"UNKNOWN" (*8)
WRITE	o	"UNKNOWN"	"UNKNOWN" (*8)
READWRITE	o	"UNKNOWN"	"UNKNOWN" (*8)

Inquiry specifier	File exists		File does not exist
	Being used (*1)	Not being used (*2)	
FORM	o	"UNDEFINED"	"UNDEFINED" (*8)
FORMATTED	o	"UNKNOWN"	"UNKNOWN" (*8)
UNFORMATTED	o	"UNKNOWN"	"UNKNOWN" (*8)
BINARY	o	"UNKNOWN"	"UNKNOWN" (*8)
RECL	o (*3)	0	0
NEXTREC	o (*4)	0	0
BLANK	o (*5)	"UNDEFINED"	"UNDEFINED" (*8)
PAD	o (*5)	"UNDEFINED" (*7)	"UNDEFINED" (*13)
POSITION	o (*6)	"UNDEFINED"	"UNDEFINED" (*8)
DELIM	o (*5)	"UNDEFINED"	"UNDEFINED" (*8)
BLOCKSIZE	o (*9)	0	0
CARRIAGECONTROL	o (*5)	"UNDEFINED"	"UNDEFINED" (*8)
CONVERT	o (*10)	"UNKNOWN"	"UNKNOWN" (*8)
ASYNCHRONOUS	o (*11)	“UNDEFINED”	“UNDEFINED”
DECIMAL	o	“UNDEFINED”	“UNDEFINED”
ENCODING	o	“UNKNOWN”	“UNKNOWN”
ROUND	o	“UNDEFINED”	“UNDEFINED”
SIGN	o	“UNDEFINED”	“UNDEFINED”
ID	o	0	0
PENDING	o	FALSE	FALSE
POS	o (*12)	0	0
SIZE	o	o	-1
STREAM	o	“UNKNOWN”	“UNKNOWN”

True or False: Four-byte logical value

o : A defined value is assigned to the variable

*1. This column indicates the processing when input/output statements are being executed for the relevant unit number.

*2. This column indicates the processing when input/output statements are not being executed for the relevant unit number.

*3. The value of the RECL= specifier in the OPEN statement is assigned to the variable. If the RECL= specifier in the OPEN statement is omitted, the value 2147483647 in Fortran95 and Fortran 2003 or the value 0 in FORTRAN66 and FORTRAN77 is assigned to the variable.

*4. Set only for direct access; otherwise, zero.

*5. Set only for formatted input/output; otherwise, it is assigned the value UNDEFINED.

*6. Set only for sequential access; otherwise, it is assigned the value UNDEFINED.

*7. Set the value YES in Fortran95.

*8. Set blank characters in FORTRAN66 and FORTRAN77.

*9. Set only for sequential access; otherwise, zero.

*10. Set only for unformatted input/output; otherwise, it is assigned the value NATIVE.

*11. For a file or unit for which access other than unformatted sequential is specified, or when a special file has been connected, it is assigned the value NO.

*12. Set only for sequential access; otherwise, it is assigned the value 0.

*13. Set the value YES in Fortran95. Set blank characters in FORTRAN66 and FORTRAN77.

Note:

1. In FORTRAN66 and FORTRAN77, the return value of the character type is returned by the small letter.

2. The undefined value is the blank character or the value 0. The blank character is assigned to the character variable. 0 is assigned to the integer variable.

3. If the Fortran system cannot define a meaningful value, -1, 0, blanks, "UNKNOWN" or "UNDEFINED" is assigned to the variable.

Returned values when unit is specified.

Inquiry specifier	Unit number within the range (*1)			Unit specifier number outside the range (*2)
	File connected (*3)		File not connected (*4)	
	Being used (*5)	Not being used (*6)		
EXIST	True	True	True	False
OPENED	True	True	False	False
NUMBER	o	-1	-1	-1
NAMED	True (*7)	False	False	False
NAME	o (*8)	" "	" "	" "
FLEN	o	0	0	0
ACCESS	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
SEQUENTIAL	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
DIRECT	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN"

Inquiry specifier	Unit number within the range (*1)			Unit specifier number outside the range (*2)
	File connected (*3)		File not connected (*4)	
	Being used (*5)	Not being used (*6)		
				(*14)
ACTION	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
READ	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
WRITE	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
READWRITE	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
FORM	o	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
FORMATTED	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
UNFORMATTED	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
BINARY	o	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
RECL	o (*9)	0	0	0 (*14)
NEXTREC	o (*10)	0	0	0 (*14)
BLANK	o (*11)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
PAD	o (*11)	"UNDEFINED" (*19)	"UNDEFINED" (*19)	"UNDEFINED" (*20)
POSITION	o (*12)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
DELIM	o (*11)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
BLOCKSIZE	o (*13)	0	0	0
CARRIAGECONTROL	o (*15)	"UNDEFINED"	"UNDEFINED"	"UNDEFINED" (*14)
CONVERT	o (*16)	"UNKNOWN"	"UNKNOWN"	"UNKNOWN" (*14)
ASYNCHRONOUS	o (*17)	“UNDEFINED”	“UNDEFINED”	“UNDEFINED”

Inquiry specifier	Unit number within the range (*1)			Unit specifier number outside the range (*2)
	File connected (*3)		File not connected (*4)	
	Being used (*5)	Not being used (*6)		
DECIMAL	o	“UNDEFINED”	“UNDEFINED”	“UNDEFINED”
ENCODING	o	“UNKNOWN”	“UNKNOWN”	“UNKNOWN”
ROUND	o	“UNDEFINED”	“UNDEFINED”	“UNDEFINED”
SIGN	o	“UNDEFINED”	“UNDEFINED”	“UNDEFINED”
ID	o	0	0	0
PENDING	o	FALSE	FALSE	FALSE
POS	o (*18)	0	0	0
SIZE	o	o	-1	-1
STREAM	o	“UNKNOWN”	“UNKNOWN”	“UNKNOWN”

True or False: Four-byte logical value

o : A defined value is assigned to the variable

" ": Indicates blanks.

*1. The unit number range is 0 to 2147483647.

*2. Outside the unit number means less than 0 or greater than 2147483647.

*3. This column indicates the processing when a unit number and file are connected.

*4. This column indicates the processing when a unit number and file are not connected.

*5. This column indicates the processing when input/output statements are being executed for the relevant unit number.

*6. This column indicates the processing when input/output statements are not being executed for the relevant unit number.

*7. For an unnamed file, false is set.

*8. For an unnamed file, a blank is set.

*9. The value of the RECL= specifier in the OPEN statement is assigned to the variable. If the RECL= specifier in the OPEN statement is omitted, the value 2147483647 in Fortran95 and Fortran 2003 or the value 0 in FORTRAN66 and FORTRAN77 is assigned to the variable.

*10. Set only for direct access; otherwise, zero.

*11. Set only for formatted input/output; otherwise, it is assigned the value UNDEFINED.

*12. Set only for sequential access; otherwise, it is assigned the value UNDEFINED.

*13. Set only for sequential access; otherwise, zero.

*14. Set blank characters in FORTRAN66 and FORTRAN77.

*15. Set only for formatted sequential access, otherwise, UNDEFINED.

*16. Set only for unformatted input/output; otherwise, it is assigned the value NATIVE.

*17. For a file or unit for which access other than unformatted sequential is specified, or when a special file has been connected, it is assigned the value NO.

*18. Set only for sequential access; otherwise, it is assigned the value 0.

*19. Set the value YES in Fortran95.

*20. Set the value YES in Fortran95. Set blank characters in FORTRAN66 and FORTRAN77.

Note:

1. In FORTRAN66 and FORTRAN77, the return value of the character type is returned by the small letter.

2. The undefined value is the blank character or the value 0. The blank character is assigned to the character variable. 0 is assigned to the integer variable.

3. If the Fortran system cannot define a meaningful value, -1, 0, blanks, "UNKNOWN" or "UNDEFINED" is assigned to the variable.

7.6.2 Difference in Language Version of the INQUIRE statement

See Section 7.6.1 INQUIRE Statement Parameters for details of the value returned by the INQUIRE statement.

7.6.2.1 Return value of ACTION= specifier

In FORTRAN66 and FORTRAN77, the variable in the ACTION= specifier is assigned the value BOTH if the file is connected for both input and output.

In Fortran95 and Fortran 2003, it is assigned the value READWRITE.

7.6.2.2 Char-constant returned by inquiry specifiers of INQUIRE statement

In Fortran66 and FORTRAN77, char-constant returned by an inquiry specifier except NAME= specifier is returned by the small letter. However, when run-time option -q is specified, it is returned by the capital letter.

In Fortran95 and Fortran 2003, it is returned by the capital letter.

Example:

```
CHARCTER(LEN=10) CH
OPEN(10,ACCESS='SEQUENTIAL')
INQUIRE(10,ACCESS=CH)
```

In FORTRAN66 and FORTRAN77, CH is assigned the value sequential.

In Fortran95 and Fortran 2003, CH is assigned the value SEQUENTIAL.

7.6.2.3 Return value of PAD= specifier

In FORTRAN66, FORTRAN77 and Fortran2003, the variable of PAD= specifier is assigned the value undefined or the blank character if the file is not being used.

In Fortran95, it is assigned the value YES.

7.6.2.4 Return value of the inquire by file of INQUIREstatement

In FORTRAN66 and FORTRAN77, the variable of the following specifiers is assigned the blank character if the file does not exist.

In Fortran95 and Fortran 2003, it is assigned the value UNDEFINED or UNKNOWN.

Specifiers are ACCESS=, SEQUENTIAL=, DIRECT=, ACTION=, READ=, WRITE=, READWRITE=, FORM=, FORMATTED=,

UNFORMATTED=, BINARY=, BLANK=, POSITION=, DELIM=, ASYNCHRONOUS=, DECIMAL=, ROUND=, SIGN= and STREAM=.

7.6.2.5 Return value of the inquire by unit of INQUIRE statement

In FORTRAN66 and FORTRAN77, the variable of the following specifiers is assigned the value the blank character if UNIT= specifier is out of range from 0 to 2147483647.

In Fortran95 and Fortran 2003, it is assigned UNDEFINED or UNKNOWN.

Specifiers are ACCESS=, SEQUENTIAL=, DIRECT=, ACTION=, READ=, WRITE=, READWRITE=, FORM=, FORMATTED=,

UNFORMATTED=, BINARY=, BLANK=, POSITION=, DELIM=, ASYNCHRONOUS=, DECIMAL=, ROUND=, SIGN= and STREAM=.

7.7 Data Transfer Input/output Statements

Not every input/output statement may be used with all input/output devices. For example, a terminal device must not be accessed using unformatted I/O.

Input/output statement type	Terminal	Direct access storage device
Formatted sequential (including namelist and list-directed)	Allowed	
Unformatted sequential	Not allowed	Allowed
Direct		
Stream		

7.7.1 Control Information for Input/output Data Transfer Statements

This section describes the control information (configuration elements) used in data transfer input/output statements.

The control specifiers are the UNIT=, FMT=, IOSTAT=, ERR=, END=, EOR=, and IOMSG= specifiers.

7.7.1.1 Input/output UNIT Specifier

When the user specifies an asterisk as the unit number or omits the UNIT= specifier, the system uses a unit based on a runtime option. If no runtime option is specified, the system uses the default specification. The table below lists the standard unit numbers used in the conditions just described.

Standard unit numbers:

Input/output statement	Runtime option	Standard value
Diagnostic message output by the Fortran system	Value of *u_no* specified by runtime option -m*u_no*	0
READ statement with * specified. READ statement with the unit specifier omitted.	Value of *u_no* specified by runtime option -r*u_no*	5
WRITE statement with * specified. PRINT statement.	Value of *u_no* specified by runtime option -p*u_no*	6

Note:

Values specified using runtime options have priority over the default values.

7.7.1.2 Format Specifier

Data is edited as real, complex, integer, logical, character, binary, octal, or hexadecimal according to an edit descriptor. For example, specifying the I edit descriptor for type real output edits data to produce integer data. The table below lists the edit descriptors and types of input/output.

Table 7.1 Edit descriptors and types of input/output

Types of input	Edit descriptor output fields							
	L	I	F,E,D,Q, EN,ES	A	G	B	Z	O
One-byte logical Two-byte logical Four-byte logical Eight-byte logical	o	+	+	o	o	o	o	o

Types of input	Edit descriptor output fields							
	L	I	F,E,D,Q, EN,ES	A	G	B	Z	O
One-byte integer Two-byte integer Four-byte integer Eight-byte integer	+	o	+	o	o	o	o	o
Default real Double-precision real Quadruple-precision real	+	+	o	o	o	o	o	o
Default complex Double-precision complex Quadruple-precision complex	+	+	o	o	o	o	o	o
Character	x	x	x	o	o	o	o	o

o: Combination is allowed.

+: Conflicting combination; a warning message is output and editing is performed according to the edit descriptor.

x: Combination is not allowed. A message is output, and processing terminates.

7.7.1.3 IOSTAT= Specifier

When an data transfer input/output statement is executed, the IOSTAT= specifier is set indicating one of the following:

- An error condition occurs.
- An end-of-file condition occurs.
- An end-of-record condition occurs.
- Neither an error condition, an end-of-file condition nor an end-of-record condition occurred.

The tables below list the value assigned to the IOSTAT= specifier after an error condition, an end-of-file condition, or an end-of-record condition occurs. The IOSTAT= specifier is set to zero if neither an error condition, an end-of-file condition, nor an end-of-record condition occurs. If an end-of-file condition or an end-of-record condition occurs for an input statement containing an IOSTAT= specifier, a diagnostic message is not output. The standard system and user-defined corrections for errors do not occur. See Section 8.1 Error Processing for details.

Error, end-of-file and end-of-record condition

Input/output error conditions	Error cases	Corresponding input/output statement	Value of IOSTAT= specifier
Error condition (Unrecoverable error)	An unrecoverable error occurred during file input/output.	OPEN statement CLOSE statement Data transfer statements File positioning statements FLUSH statement	1
Error condition (except unrecoverable error)	An I/O error was detected.	OPEN statement CLOSE statement Data transfer statements	Error number

Input/output error conditions	Error cases	Corresponding input/output statement	Value of IOSTAT= specifier
		File positioning statements FLUSH statement	
End-of-file condition	An end-of-file record was detected.	Sequential READ List-directed READ Namelist READ Stream READ	-1
End-of-file condition	A Fortran record exceeded the end of the internal file.	Internal file READ	-1
End-of-record condition	The length of input list exceeds the length of a Fortran record.	Nonadvancing formatted sequential input	-2

7.7.1.4 Error Branch

If an error condition occurs during the execution of a data transfer input/output statement, execution continues with the statement specified in the ERR= specifier. If the input/output statement also contains an IOSTAT= specifier, the value of its error number is assigned to the variable. See Section 7.7.1.3 IOSTAT= Specifier for details.

If an error condition occurs for an input/output statement that specifies the ERR= specifier, a diagnostic message is not output, and the standard system and user-defined error correction do not occur.

7.7.1.5 End-of-File Branch

If an end-of-file condition occurs and no error condition occurs during the execution of a data transfer input/output statement, execution continues with the statement specified in the END= specifier. If the input/output statement also contains an IOSTAT= specifier, the value -1 is assigned to the variable.

If an end-of-file condition occurs for an input statement that specifies the END= specifier, a diagnostic message is not output, and the standard system and user-defined error correction do not occur.

7.7.1.6 End-of-Record Branch

If an end-of-record condition occurs and no error condition occurs during the execution of a data transfer input/output statement, execution continues with the statement specified in the EOR= specifier. If the input/output statement also contains an IOSTAT= specifier, the value -2 is assigned to the variable.

If an end-of-record condition occurs for an input statement that specifies the EOR= specifier, a diagnostic message is not output, and the standard system and user-defined error correction do not occur.

7.7.1.7 IOMSG= Specifier

The IOMSG= specifier sets an error message when an error condition, end-of-file condition, or end-of-record condition occurs.

Note, however, that if an IOSTATE= specifier or an ERR= specifier is not specified simultaneously with the IOMSG= specifier, a message is not set. Further, if an error condition, end-of-file condition, or end-of-record condition does not occur, the value specified in the IOMSG= specifier is not changed.

7.7.2 Sequential Access Input/output Statements

Sequential input/output statements read and write formatted and unformatted Fortran records from sequential files. Formatted sequential input/output statements are used to read and write formatted Fortran records. Unformatted sequential input/output statements are used to read and write unformatted Fortran records.

7.7.2.1 Formatted Sequential Access Input/output Statements

Formatted sequential input/output statements sequentially read and write characters based on a format specification. Generally, these statements are used when the data must be displayed.

Formatted sequential READ statements read data from the current Fortran record based on a format. When an input list is specified, the Fortran record is edited into internal representations based on edit descriptors in the format.

Formatted sequential WRITE and PRINT statements write Fortran records to the current record based on a format. When an output list is specified, each item is edited based on the edit descriptors in the format. If the $ edit descriptor is specified, a line-feed character is not written after the edited data.

Example: A formatted sequential input/output statement used to read and write data

```
      REAL A,B
      INTEGER I,J
      A=1.5                      !A is assigned the hexadecimal value 3fc00000
      I=100                      !I is assigned the hexadecimal value 00000064
      WRITE(1,10) I,A           !A Fortran record becomes the hexadecimal
                                 !     value 2031303020312e35
      REWIND 1                   !Position the file at the beginning.
      READ(1,10) J,B            !J is assigned the hexadecimal value 00000064.
                                 !     B is assigned the hexadecimal value 3fc00000.
 10   FORMAT(1X,I3,1X,F3.1)
      STOP
      END
```

7.7.2.2 Unformatted Sequential Access Input/output Statements

Unformatted sequential input/output statements read and write Fortran records consisting of internal representations. Generally, these statements are used when the data need not be displayed.

If unformatted sequential input/output statements are executed for devices (such as terminals) that process character codes, the results are not predictable.

Files written with variable-length Fortran records by Fujitsu Fortran implementation can be accessed by an unformatted input statement.

Unformatted sequential READ statements read the current record. When an input list is specified, the data is stored unchanged into each item in the input list in the order specified. An error occurs if the number of bytes in the record is less than in the input list items. If the number of bytes in the record is greater, the excess data is ignored. If an input list is not specified, the record is skipped.

Unformatted sequential WRITE statements write the output list items as a Fortran record in the order they are specified. The statement writes the record in the current file position. If an output list is not specified, a Fortran record of length zero is written.

The length of the Fortran record is the sum of the bytes in the output list items.

Example: An unformatted sequential input/output statement used to read and write data

```
REAL A,B
INTEGER I,J
A=1.5                     !A is assigned the hexadecimal value 3fc00000
I=100                     !I is assigned the hexadecimal value 00000064
WRITE(1) A,I              !A Fortran record becomes the hexadecimal value
                          !     3fc0000000000064
REWIND 1                  !Position the file at the beginning.
READ(1) B,J               !B is assigned the hexadecimal value 3fc00000
                          !     J is assigned the hexadecimal value 00000064
STOP
END
```

7.7.3 Direct Access Input/output Statements

Direct input/output statements are used to read and write formatted or unformatted Fortran records to or form the record of the file indicated by the REC= specifier.

An OPEN statement must be executed before any direct input/output statements may be executed.

When TOTALREC is used in an OPEN statement, reading and writing can be executed only for the number of records specified. Sequential input/output statements must not be used to add records to direct files.

7.7.3.1 Direct Access READ Statement

A direct access READ statement read Fortran records which is uniquely identified by the record number.

During formatted data transfer, data are transferred with editing between the file and the entities specified by the input list.

During unformatted data transfer, data are transferred without editing between current record and the entities specified by the input list. The number of values required by the input list shall be less than or equal to the number of values in the record.

Data are not transferred if the input list does not exist.

7.7.3.2 Direct Access WRITE Statement

A direct access WRITE statement writes Fortran records which is uniquely identified by the record number.

During formatted data transfer, data are transferred with editing between the file and the entities specified by the output list. Blank values are transferred if the output list does not exist.

During unformatted data transfer, data are transferred without editing between current record and the entities specified by the output list. The output list shall not specify more values than can fit into the record. If the values specified by the output list do not fill the record, the remainder of the record is undefined. Zero values are transferred if the output list does not exist.

7.7.3.3 The Error of Data Transfer in a DIRECT Access Input/output

The execution of Fortran program is different by the compiler option -X if the output list specify more values than can fit into the record and the number of values required by the input list are more than the number of values in the record.

In FORTRAN66 and FORTRAN77, the program ends normally.

In Fortran 95 and Fortran 2003, a diagnostic message of w level is generated and the multiple records are written or read.

Example:

program di.f

```
OPEN(10,FILE='XX.DATA',FORM='UNFORMATTED',ACCESS='DIRECT',RECL=4)
WRITE(10,REC=1)1.0_8
INQUIRE(10,NEXTREC=i)
PRINT*,'NEXTREC=',i
END
```

result

In FORTRAN66 and FORTRAN77:

```
$ ./a.out
 NEXTREC=3
$
```

In Fortran 95 and Fortran 2003:

```
$ ./a.out
jwe1162i-w line 2 On output to a file connected for unformatted direct access, the output list must
not specify more values than can fit into the Fortran record (unit=10).
 error occurs at MAIN__   line 2 loc 0000000000400be6 offset 0000000000000036
 MAIN__       at loc 0000000000400bb0 called from o.s.
taken to (standard) corrective action, execution continuing.
 NEXTREC= 3
error summary (Fortran)
error number  error level  error count
  jwe1162i         w           1
```

```
total error count = 1
$
```

7.7.4 Namelist Input/output Statements

Namelist input/output statements transfer data between memory and sequential formatted records in an internal or external file. The namelist names and their associated variable and array names used for data transfer must have been declared previously in a NAMELIST statement.

7.7.4.1 Namelist READ Statements

It is necessary for the type of the item in the namelist to match the type of the data in the record. The correspondence for type of element and type of constant is indicated in the following table.

Type of namelist group object	Type of constant					
	Logical	Integer	Real	Complex	Character	Hexadecimal
One-byte logical Two-byte logical Four-byte logical Eight-byte logical	o	+ (*1)	+ (*1)	x	+ (*2)	+ (*7) (*8)
One-byte integer Two-byte integer Four-byte integer Eight-byte integer	+ (*3)	o	+ (*3)	x	+ (*2)	o (*8)
Default real Double-precision real Quadruple-precision real	+ (*4)	o (*5)	o	x	+ (*2)	+ (*7) (*8)
Default complex Double-precision complex Quadruple-precision complex	+ (*4) (*6) (*9)	+ (*5) (*6) (*9)	+ (*6) (*9)	o	+ (*2)	+ (*7) (*8)
Character	x	x	x	x	o	x (*7) (*8)

o: Operation is normal.

+: Combination is not allowed, but operation continues; interprets to revised for element type.

x: Input is terminated because the combination is not allowed.

*1. Logical editing is performed.

*2. Character editing is performed.

*3. Integer editing is performed.

*4. Real editing is performed.

*5. Real editing is performed. The decimal point is assumed to be to the right of the constant.

*6. The edit result is entered in the real part. Zero is entered in the imaginary part.

*7. In FORTRAN66 and FORTRAN77, the operation for hexadecimal types is normal.

*8. In Fortran 95 and Fortran 2003, hexadecimal is assumed to be character. Therefore, if the type of element is character, the operation is normal. If the type of element is not character, the combination is not allowed but operation continues. See Section "2.2 Compiler Options" for compiler options.

*9. In Fortran 95 and Fortran 2003, a diagnostic message of w level is generated.

Note:

For input processing, editing is based on the type of namelist element. For hexadecimal or character types, editing is based on the type of constant in the file.

a. Processing of Character Constants for Array Name.
The processing of character constants for array names depends on whether the type of the array is character or not. If the type of the array is character, one character constant corresponds to one array element. Even if the character constant exceeds the array element length, there is no overflow to the next array element.

Example: Character constant processing for elements of type character Fortran program:

```
CHARACTER(4) HA(5)
NAMELIST/NAMEH/HA
READ(1,NML=NAMEH)   !READ statement for namelist NAMEH
```

If the record to be read from the file connected to unit 1 is

```
&NAMEH HA=2*'NAMELIST READ STATEMENT' /
```

then the elements of HA will have the following values:

HA(1)	'NAME '
HA(2)	'NAME '
HA(3)	Unchanged
HA(4)	Unchanged
HA(5)	Unchanged

If the type of the array is not character, the character constant corresponding to the first array element is entered for the entire array. That is, if the length of the character constant exceeds one element, the character constant is stored in succeeding elements. If the length of the character constant is shorter than the entire array, the remaining elements are unchanged. If input ends within an element, the remaining part of the element is padded with blanks.

For the second and subsequent array elements, each character constant corresponds to a single element.

Even if the length of the character constant exceeds the array element length, no overflow occurs.

Example: The processing of character constants for elements of other than type character

Fortran program:

```
INTEGER IA(5)
NAMELIST/NAMEI/IA
READ(1,NML=NAMEI)   !READ statement for namelist NAMEI
```

The namelist record for unit number 1 is set as follows:

```
&NAMEI IA=2*'NAMELIST READ STATEMENT' /
```

The following is the result in character notation:

IA(1)	'NAME'
IA(2)	'NAME'
IA(3)	' REA'
IA(4)	'D ST'
IA(5)	'ATEM'

b. Input Editing Operation

The format of the namelist READ statement is determined based on the element type, constant type and constant size, as follows:

- If the type of constant and type of element are other than character, editing is performed according to the element type. The edit descriptor is Gw. The letter w indicates the size (between delimiters) of the constant to be entered.
- If the constant type is character, character editing is performed. The edit descriptor is Aw. The letter w indicates the number of characters.
- For noncharacter constants with elements of type character, character editing is performed. The edit descriptor is Aw.

7.7.4.1.1 Namelist data in Namelist input

The namelist data in the namelist input has one of following formats:

```
&name [v=c[,c]...[,v=c[,c]...]...] /
```

```
&name [v=c[,c]...[,v=c[,c]...]...] &end
```

name: The namelist-group name specified in the corresponding NAMELIST input/output statement.

v: The variable name or corresponding array element, array section, or character substring specified in the NAMELIST statement corresponding to the name.

c: NULL or one of the following forms:

```
const
```

```
r*const
```

```
r*
```

const: A literal constant other than a Hollerith, binary, octal, or hexadecimal constant.

r: A repeat specification. The argument is an unsigned integer constant.

*r*const*: It is equivalent to the sequential appearance of the *const r* times.

*r**: It is equivalent to the sequential appearance of NULL *r* times.

If the input item is a character type and all following conditions are satisfied, the character constant need not be enclosed with apostrophes or quotation marks.

- A blank, comma, slash, or ampersand, which is considered to be a value separator, is not included in the character constant.
- A semicolon, which is considered to be a value separator, is not included in the character constant, when the decimal point symbol is a comma.
- Left parentheses, right parentheses, and percent sign are not included in the character constant.
- The character constant does not extend over more than one Fortran record.
- The first non-blank character is a character other than an apostrophe or quotation mark.

- The form of the input item is not *r***const*.

When a character constant does not end with an delimiter character (apostrophe or quotation mark), it is assumed to end with the first blank, comma, or slash. Moreover, an apostrophe or quotation mark in the character string is not treated as two consecutive apostrophes or two consecutive quotation marks.

7.7.4.2 Namelist WRITE Statements

Namelist WRITE statements are used for output editing. These statements write a record based on the types of variables and arrays specified in the namelist. The output format for variable names consists of a variable name, equal sign, and edit result, in that order. A comma separates constants. The output format for array names consists of an array name, equal sign, and edit result for each array element, in that order. A comma separates the edit results.

The output format for a namelist group object is as follows:

Type of namelist group object	Output format (Edit descriptor)
One-byte logical	G1,1H, (*1)
Two-byte logical	G1,1H, (*1)
Four-byte logical	G1,1H, (*1)
Eight-byte logical	G1,1H, (*1)
One-byte integer	G4,1H, (*1)
Two-byte integer	G6,1H, (*1)
Four-byte integer	G11,1H, (*1)
Eight-byte integer	G20,1H, (*1)
Default real	1P,E15.8,1H, (*1)
	0P,G16.9,1H, (*1)
Double-precision real	1P,E22.15,1H, (*1)
	0P,G23.16,1H, (*1)
Quadruple-precision real	1P,E43.34E4,1H, (*1)
	0P,G44.35E4,1H, (*1)
Default complex	1H(,1P,E15.8,1H,,E15.8,2H), (*2)
	1H(,0P,G16.9,1H,,G16.9,2H), (*2)
Double-precision complex	1H(,1P,E22.15,1H,,E22.15,2H), (*2)
	1H(,0P,G23.16,1H,,G23.16,2H), (*2)
Quadruple-precision complex	1H(,1P,E43.34E4,1H,,E43.34E4,2H), (*2)
	1H(,0P,G44.35E4,1H,,G44.35E4,2H), (*2)
n-byte character	1H',character-contents,2H', (*3)
	1H'',character-contents,2H'', (*4)
	character-contents,1H, (*5)

*1. If an &end or a / comes immediately after the value generated by the G or E edit descriptor, 1H, is omitted.

*2. If an &end or a / comes immediately after the value generated by the G or E edit descriptor, 1H) is generated instead of 2H),.

*3. If an &end or a / comes immediately after the value delimited by quotes, 1H' is generated instead of 2H',.

*4. If an &end or a / comes immediately after the value delimited by apostrophes, 1H'' is generated instead of 2H'',.

*5. If an &end or a / comes immediately after the nondelimited character constants, 1H, is omitted.

Notes:

1. For types other than character, meaningless blanks in the edit result are deleted before output. The actual output length can therefore be smaller than the field width of the edit descriptor.

2. Whether 1P,Ew1.d1 or 0P,Gw2.d2 is used for type real or complex depends on the absolute value of the data. 0P,Gw2.d2 is used if the absolute value of x is 0 or 0.1 <= x < 10**d2. Otherwise, 1P,Ew1.d1 is used.

7.7.4.3 NAMELIST Input/output Version Differences

a. Character type output
In FORTRAN66 and FORTRAN77, character constants are written using apostrophes as delimiters. In Fortran 95 and Fortran 2003, DELIM= can be specified as the delimiter of character constants in an OPEN statement. If DELIM= specifier is omitted in an OPEN statement or an OPEN statement is not executed, the DELIM= specifier is treated as NONE. In this case, character constants are written without delimiters.

Example:

```
CHARACTER(LEN=5)::CH='XXXXX'
NAMELIST /X/CH
OPEN(10)
WRITE(10,NML=X)
```

In FORTRAN66 and FORTRAN77, the output is as follows:

```
&X
CH='XXXXX'
&end
```

In Fortran 95 and Fortran 2003, the output is as follows:

```
&X CH=XXXXX/
```

b. Format for terminating the namelist output statement
In FORTRAN66 and FORTRAN77, &end is written to terminate the namelist output statement. In Fortran 95 and Fortran 2003, / is written.

Example:

```
INTEGER::IH=123
NAMELIST /X/IH
WRITE(10,NML=X)
```

In FORTRAN66 and FORTRAN77, the output is as follows:

```
&X
IH=123
&end
```

In Fortran 95 and Fortran 2003, the output is as follows:

```
&X IH=123/
```

c. Blanks in namelist input/output
FORTRAN66 and FORTRAN77 treat blanks as part of the input data. Fortran 95 and Fortran2003 treat a blank as a delimiter. However, blanks between delimiters are treated as input data.

Example:

```
INTEGER,DIMENSION(3) :: IH
NAMELIST /X/IH
READ(*,NML=X)
```

Assume that the input data is IH=1_2_3, (_ indicates a blank).

FORTRAN66 and FORTRAN77 treat this data as follows:

```
IH(1)=10203
IH(2)=0
IH(3)=0
```

Fortran 95 and Fortran 2003 treat this data as follows:

```
IH(1)=1
IH(2)=2
IH(3)=3
```

d. Hexadecimal constants specified as namelist input data
FORTRAN66 and FORTRAN77 treat hexadecimal constants specified as namelist input data as hexadecimal constants. Fortran 95 and Fortran2003 treat these constants as character constants.

Example:

```
CHARACTER(LEN=5) CH
NAMELIST /X/CH
READ(*,NML=X)
```

Assume that the input data is CH=Z3132333435,.

FORTRAN66 and FORTRAN77 treat this data as follows:

```
CH='12345'
```

Fortran 95 and Fortran 2003 treat this data as follows:

```
CH='Z3132'
```

e. Hollerith constants specified as namelist input data.
FORTRAN66 and FORTRAN77 treat Hollerith constants specified as namelist input data as Hollerith constants. Fortran 95 and Fortran 2003 treat Hollerith constants specified as namelist input data as character constants.

Example:

```
CHARACTER(LEN=5) CH
NAMELIST /X/CH
READ(*,NML=X)
```

Assume that the input data is CH=3H123,.

FORTRAN66 and FORTRAN77 treat this data as follows:

```
CH='123 '
```

Fortran 95 and Fortran 2003 treat this data as follows:

```
CH='3H123'
```

f. Output of the first byte of the second and subsequent records when Fortran records contain character constants.
In FORTRAN66 and FORTRAN77, a blank is written in the first byte of the second record. In Fortran 95 and Fortran 2003, the first byte contains a value.

Example:

```
CHARACTER(LEN=1000) CH
NAMELIST /X/CH
WRITE(10,NML=X)
```

In FORTRAN66 and FORTRAN77, a blank is written as follows:

```
CH='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX <- the first record
*XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' <- the second record
^
blank is written
```

In Fortran 95 and Fortran 2003, a value is written as follows:

```
CH='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX <- the first record
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX' <- the second record
^
value is written
```

g. Input data with the *r** form or *r**const form.
In FORTRAN66 and FORTRAN77, when a *r** form appears in input data, it does not mean 'repeat the null value r times'. For those versions, it means the repeat is applied to the following input data. If the r* constitutes the last characters of a Fortran record, the repeat is applied to the first input data of the next Fortran record. In Fortran 95 and Fortran 2003, *r** means 'repeat the null value r times'.

Example:

```
INTEGER::I(4)=(/1,2,3,4/)
NAMELIST /X/I
READ(*,NML=X)
```

Assume that the input data is I=2*_30,40, (_ indicates a blank).

FORTRAN66 and FORTRAN77 treat this data as follows:

```
I(1)=30
I(2)=30
I(3)=40
I(4)=4
```

Fortran 95 and Fortran 2003 treat this data as follows:

```
I(1)=1
I(2)=2
I(3)=30
I(4)=40
```

h. The output of a group name or an object name of namelist output statement.
In FORTRAN66 and FORTRAN77, a group name or an object name in the output is case sensitive.
In Fortran 95 and Fortran 2003, it is in uppercase.

Example:

```
INTEGER::Iu=1
NAMELIST /Xu/Iu
WRITE(10,NML=Xu)
```

In FORTRAN66 and FORTRAN77, the output is as follows:

```
&Xu
Iu=1
&end
```

In Fortran 95 and Fortran 2003, the output is as follows:

```
&XU IU=1/
```

i. The output of a namelist Fortran record.
In FORTRAN66 and FORTRAN77, the output is as follows.
The first record : _&*name* (_ indicates a blank, *name* indicates namelist group object name)
The second record : _*v*=*c*[,*c*]...[,*v*=*c*[,*c*]...]...
(*v* indicates namelist group object list item, *c* indicates namelist output value)
The last record : _&*end*
In Fortran 95 and Fortran 2003, the output is as follows.
The first record : _&*name v*=*c*[,c]...[,*v*=*c*[,*c*]...].../

j. Output of real number zero.

The result of executing a real number type of variable group output with a value of 0 varies according to the language specification level.

In the FORTRAN66, FORTRAN77, and Fortran 95 language specifications, output of real number zero is equivalent to the result output when E edit is used.

In the Fortran 2003 language specifications, output of real number zero is equivalent to the result output when F edit is used.

Example:

```
REAL :: R = 0
NAMELIST /DATA/R
WRITE(*, NML=DATA)
```

When the Fortran program above is executed, in the Fortran 95 language specifications, &DATA R=0.00000000E+00/ is output.

When the Fortran program above is executed, in the Fortran 2003 language specifications, &DATA R=0.00000000/ is output.

7.7.5 List-Directed Input/output Statements

List-directed input/output statements transfer data between memory and formatted records based on the type of the data. Internal files may be used for list-directed formatting.

7.7.5.1 List-Directed READ Statements

List-directed READ statements transfer records from external and internal files to elements in input lists.

The types of elements in input lists must correspond to the types of elements in data items as indicated by the following table.

Type of element	Type of constant				
	Logical	Integer	Real	Complex	Character
One-byte logical Two-byte logical Four-byte logical Eight-byte logical	o	+ (*1)	+ (*1)	x	+ (*2)
One-byte integer Two-byte integer Four-byte integer Eight-byte integer	+ (*3)	o	+ (*3)	x	+ (*2)
Default real Double-precision real Quadruple-precision real	+ (*4)	o (*5)	o	x	+ (*2)
Default complex Double-precision complex Quadruple-precision complex	+ (*4) (*6) (*7)	+ (*5) (*6) (*7)	+ (*6) (*7)	o	+ (*2)
Character	+ (*2)	+ (*2)	+ (*2)	+ (*2)	o

o: Operation is normal.

+: Combination is not allowed, but operation continues; interprets to revised for element type.

x: Input is terminated because the combination is not allowed.

*1. Logical editing is performed.

*2. Character editing is performed.

*3. Integer editing is performed.

*4. Real editing is performed.

*5. Real editing is performed. The decimal point is assumed to be to the right of the constant.

*6. The edit result is set in the real part. Zero is set in the imaginary part.

*7. In Fortran95 and Fortran 2003, a diagnostic message of w level is generated.

a. Processing of Character Constants for Array Names

The processing of character constants for array names depends on whether the element is type character.

If the element is type character, input is performed so that one character constant corresponds to one array element. Even if the length of the character constant exceeds the array element, there is no overflow to the next array element.

If the element is not type character, the character constant corresponding to the first array element is entered for the entire array. That is, if the length of the character constant exceeds one element, the succeeding elements are also read. If the length of the character constant does not exceed the length of the entire array, the remaining elements are unchanged. If input ends within an element, the remaining part of the element is padded with blanks.

For second and subsequent array elements, the character constant corresponds to each element. Even if the length of the character constant exceeds that of the array element, the character constant does not overflow to the next element.

Example: How to enter data of type character for array names

Fortran program:

```
CHARACTER(LEN=4),DIMENSION(5) :: CHAR5
INTEGER,DIMENSION(5) :: I4
    ...
READ(5,*) CHAR5        ! List-directed READ statement for
                       ! character type array elements
READ(5,*) I4           ! List-directed READ statement for
                       ! other than character type array elements
```

The following is the format of the list-directed record for unit number 5:

```
3*'ABCDEFGHIJKLMNOPQ'/
3*'ABCDEFGHIJKLMNOPQ'/
```

The following is the result for CHAR5 and I4:

CHAR5(1)	'A B C D '	I4(1)	'A B C D '
CHAR5(2)	'A B C D '	I4(2)	'A B C D '
CHAR5(3)	'A B C D '	I4(3)	'A B C D '
CHAR5(4)	Unchanged	I4(4)	'M N O P '
CHAR5(5)	Unchanged	I4(5)	'Q '

b. Input Editing Operation

The format used by a list-directed READ statement is determined according to the element type, constant type, and constant size, as follows:

- If the constant and element are other than type character, editing is performed according to the element type. The edit descriptor is G*w*. The letter w indicates the size (between delimiters) of the constant to be entered.
- If the constant is type character, character editing is performed. The edit descriptor is A*w*. The letter *w* indicates the size between apostrophes.

- If the constant is other than type character and the element is type character, character editing is performed. The edit descriptor is A*w*.

7.7.5.2 List-Directed WRITE Statements

List-directed WRITE statements generate formatted list-directed records. The following edit descriptors are used for output with a list-directed WRITE statement.

Type of element	Output format (edit descriptor)
One-byte logical	G1,1H_ (*1)
Two-byte logical	G1,1H_ (*1)
Four-byte logical	G1,1H_ (*1)
Eight-byte logical	G1,1H_ (*1)
One-byte integer	G4,1H_ (*1)
Two-byte integer	G6,1H_ (*1)
Four-byte integer	G11,1H_ (*1)
Eight-byte integer	G20,1H_ (*1)
Default real	1P,E15.8,1H_ or 0P,G16.9,1H_ (*1)
Double-precision real	1P,E22.15,1H_ or 0P,G23.16,1H_ (*1)
Quadruple-precision real	1P,E43.34E4,1H_ or 0P,G44.35E4,1H_ (*1)
Default complex	1H(,1P,E15.8,1H,,E15.8,2H)_ or (*2) 1H(,0P,G16.9,1H,,G16.9,2H)_ (*2)
Double-precision complex	1H(,1P,E22.15,1H,,E22.15,2H)_ or (*2) 1H(,0P,G23.16,1H,,G23.16,2H)_ (*2)
Quadruple-precision complex	1H(,1P,E43.34E4,1H,,E43.34E4,2H)_ or (*2) 1H(,0P,G44.35E4,1H,,G44.35E4,2H)_ (*2)
n-byte character	Gn,1H_ or (*3)(*6) 1H',Gn,2H'_ or (*4)(*6) 1H'',Gn,2H''_ (*5)(*6)

_: Indicates blank

*1. If the record terminates immediately after the value generated by the G or E edit descriptor, 1H_ is omitted.

*2. If the record terminates 1 byte after the value generated by the G or E edit descriptor, 1H) is generated instead of 2H)_.

*3. If the record terminates immediately after the nondelimited character constants generated by the G edit descriptor, 1H_ is omitted. If the compiler option -X9 is not in effect, the output format is Gn.

*4. If the record terminates 1 byte after the value delimited by quotes generated by the G or E edit descriptor, 1H' is generated instead of 2H'_.

*5. If the record terminates 1 byte after the value delimited by apostrophes generated by the G or E edit descriptor, 1H'' is generated instead of 2H''_.

*6. In Fortran95 and Fortran 2003, is not in effect, the last blank is not output.

Notes:

1. For types other than character, meaningless blanks in the edit result are deleted before output. The actual output length can therefore be smaller than the field width of the edit descriptor.

2. Whether 1P,Ew1.d1 or 0P,Gw2.d2 is used for type real or complex depends on the absolute value of the data. 0P,Gw2.d2 is used if the absolute value of x is 0 or $0.1 \leq x < 10^{**}d2$. Otherwise, 1P,Ew1.d1 is used.

Example: List-directed WRITE statement

Fortran program:

```
INTEGER :: YY=2002, MM=09, DD=10
CHARACTER(15) :: CH='YEAR MONTH DATE'
INTEGER,DIMENSION(2,3) :: TABLE
WRITE(6,*) YY,CH(1:4) ,MM,CH(6:10) ,DD,CH(12:15)
OPEN(6,DELIM='QUOTE')
WRITE(6,*) YY,CH(1:4) ,MM,CH(6:10) ,DD,CH(12:15)
OPEN(6,DELIM='APOSTROPHE')
WRITE(6,*) YY,CH(1:4) ,MM,CH(6:10) ,DD,CH(12:15)
TABLE = RESHAPE((/1,2,3,4,5,6/) ,(/2,3/))
WRITE(6,*) ((TABLE(I,J) ,I=2,1,-1) ,J=3,1,-1)
END
```

The output is as follows:

```
$ ./a.out
 2002 YEAR 9 MONTH 10 DATE
 2002 "YEAR" 9 "MONTH" 10 "DATE"
 2002 'YEAR' 9 'MONTH' 10 'DATE'
 6 5 4 3 2 1
$
```

7.7.5.2.1 The Form of Nondelimited Character Constants Produced by List-Directed Output

In FORTRAN66 and FORTRAN77, the values are not separated.

In Fortran 95 and Fortran 2003, the values are separated by one blank.

Example:

```
CHARACTER(10)::CH=' '
WRITE(*,*)'Year',2002
WRITE(CH,*)'Month',9
WRITE(*,'(A10)')CH
END
```

In FORTRAN66 and FORTRAN77, the output is as follows:

```
$ ./a.out
 Year2002
 Month9
$
```

In Fortran 95 and Fortran 2003, the output is as follows:

```
$ ./a.out
 Year 2002
 Month 9
$
```

7.7.5.3 List-Directed Input/output Version Differences

Output of real number zero

The result of executing a real number type of list output with a value of 0 varies according to the language specification level.

In the FORTRAN66, FORTRAN77, and Fortran 95 language specifications, output of real number zero is equivalent to the result output when E edit is used.

In the Fortran 2003 language specifications, output of real number zero is equivalent to the result output when F edit is used.

Example:

```
REAL :: R = 0
WRITE(*,*) R
```

When the Fortran program above is executed, in the Fortran 95 language specifications, 0.00000000E+00 is output.

When the Fortran program above is executed, in the Fortran 2003 language specifications, 0.00000000 is output.

7.7.6 Internal File Input/output Statements

Internal file input/output statements treat a character string as a file. See Section 7.7.2 Sequential Access Input/output Statements, 7.7.4 Namelist Input/output Statements, and 7.7.5 List-Directed Input/output Statements for details.

7.7.7 Nonadvancing Input/output Statements

Nonadvancing input/output lets you control when the next record will be processed, and a single Fortran record can be read and written as many times as necessary. The statement may position the file at a character position within the current record. You can then use a SIZE=, IOSTAT=, or EOR= specifier to control the Fortran program. The variable specified in the SIZE= specifier becomes defined with the count of the characters transferred by data edit descriptors during execution of the current input statement. If an end of record condition occurs, the control may be processed by an IOSTAT variable or EOR= specifier.

Example: Nonadvancing input statements with a SIZE= and EOR= specifier

```
      CHARACTER(LEN=20) NAME
      INTEGER NO
 !      1-20 : NAME
 !      21-28 : NO
      OPEN(10,FORM='FORMATTED')
      DO
         READ(10,FMT='(A20) ',ADVANCE='NO',EOR=10,SIZE=IS,END=20) NAME
         IF (IS < 20) CYCLE
         READ(10,FMT='(I8) ',ADVANCE='NO',SIZE=IS,END=20,IOSTAT=IO) NO
         IF (IS < 8) CYCLE
         GO TO 30
10       CYCLE
30       PRINT *,NAME,' ',NO,' ',IS
      END DO
      GO TO 40
20    PRINT *,'END OF DATA'
40    END
```

7.8 Combining Input/output Statements

When input/output statements are executed, processing depends on the type of input/output statement executed previously.

The combinations of input/output statements explained here use the same unit number.

Processing according to input/output statement order

Current I/O	Immediately preceding I/O											
	Formatted sequential		Unformatted sequential		Formatted direct		Unformatted direct		Formatted stream		Unformatted stream	
	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE
Formatted sequential READ	o	x (*1)	x	x	x	x	x	x	x	x	x	x
Formatted sequential WRITE	o (*2)	o	x	x	x	x	x	x	x	x	x	x

Current I/O	Immediately preceding I/O											
	Formatted sequential		Unformatted sequential		Formatted direct		Unformatted direct		Formatted stream		Unformatted stream	
	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE	READ	WRITE
Unformatted sequential READ	x	x	o	x (*1)	x	x	x	x	x	x	x	x
Unformatted sequential WRITE	x	x	o (*2)	o	x	x	x	x	x	x	x	x
Formatted direct access	x	x	x	x	o	o	x (*5)	x (*5)	x	x	x	x
Unformatted direct access	x	x	x	x	x (*5)	x (*5)	o	o	x	x	x	x
Formatted stream READ	x	x	x	x	x	x	x	x	o	o (*6)	x	x
Formatted stream WRITE	x	x	x	x	x	x	x	x	o	o	x	x
Unformatted stream READ	x	x	x	x	x	x	x	x	x	x	o	o (*6)
Unformatted stream WRITE	x	x	x	x	x	x	x	x	x	x	o	o
OPEN	o	o	o	o	o	o	o	o	o	o	o	o
CLOSE	o	o	o	o	o	o	o	o	o	o	o	o
FLUSH	-	o	-	o	-	o	-	o	-	o	-	o
WAIT	o	o	o	o	o	o	o	o	o	o	o	o
BACKSPACE	o	o (*3)	o	o (*3)	x	x	x	x	o	o	x	x
REWIND	o	o	o	o	x	x	x	x	o	o	o	o
ENDFILE	o (*4)	o	o (*4)	o	x	x	x	x	o	o	o	o

o: The operation is normal.

x: A diagnostic message is output. Correction processing is performed.

-: No operation is performed.

*1. The READ statement cannot execute, because the Fortran record that follows the one written by the immediately preceding WRITE statement is undefined.

*2. The Fortran record that follows the one written by the current WRITE statement is undefined.

*3. The endfile record is generated before BACKSPACE is executed.

*4. The endfile record is generated immediately after the READ statement reads the Fortran record.

*5. In FORTRAN66 and FORTRAN77, the operation is normal.

*6. Execution of a stream input statements omitted POS= specifier after the execution of a stream output statements is not allowed.

Note:

Formatted READ and WRITE statements also include list-directed and namelist input/output statements.

Current I/O	Immediately preceding I/O								
	OPEN	CLOSE	FLUSH	WAIT	BACKSPACE	REWIND	END	EOF (*1)	None
Formatted sequential READ	o	o (*6)	o	o	o	o (*2)	o (*7)	x (*7)	o (*8)
Formatted sequential WRITE	o	o (*6)	o	o	o	o (*2)	o (*6)	x (*11)	o (*8)
Unformatted sequential READ	o	o (*6)	o	o	o	o (*2)	o (*7)	x (*7)	o (*8)
Unformatted sequential WRITE	o	o (*6)	o	o	o	o (*2)	o (*6)	x (*11)	o (*8)
Formatted direct access	o	x	o	o	x	x	x	x	x (*3)
Unformatted direct access	o	x	o	o	x	x	x	x	x (*3)
Formatted stream access	o	x	o	o	o	o (*2)	o	o	x (*3)
Unformatted stream access	o	x	o	o	x	o (*2)	o	o	x (*3)
OPEN	o	o	o	o	o	o	o	o	o
CLOSE	o	o	o	o	o	o	o	o	o
FLUSH	-	-	-	-	-	-	-	-	-
WAIT	o	o	o	o	o	o	o	o	o
BACKSPACE	o	o (*6)	o	o	o	-	o (*5)	o (*4)	o (*10)
REWIND	o	o (*9)	o	o	o	-	o	o	-
ENDFILE	o (*5)	o (*6)	o	o	o	o (*5)	- (*7)	- (*9)	o (*5)

o: The operation is normal.

x: A diagnostic message is output. Correction processing is performed.

-: No operation is performed.

*1. EOF means the end-of-file condition occurs.

*2. A combination of formatted and unformatted sequential input/output statements before or after a REWIND statement is not allowed.

*3. OPEN statement shall be executed for old file.

*4. BACKSPACE is executed for the endfile record.

*5. An empty file is created.

*6. Input/output is executed for the endfile record. In Fortran 2003 and Fortran 95, a diagnostic message (jwe0111i-e) is output. In FORTRAN766 and FORTRAN77, a diagnostic message (jwe0111i-e) is output.

*7. Input/output is executed for the endfile record.

*8. Input/output is executed for files named 'fort. nn' (nn:unit number.)

*9. Error.

*10. BACKSPACE is executed for the endfile record.

*11. Input/output is executed for the endfile record. In Fortran 2003 and Fortran 95, a diagnostic message (jwe0115i-e) is output. In FORTRAN766 and FORTRAN77, a diagnostic message (jwe0115i-e) is output.

Note:

Formatted READ and WRITE statements also include list-directed and namelist input/output statements.

7.8.1 Combinations of Input/output Statements Not Allowed

Some combinations of input/output statements are not allowed because of the access method or file status.

An input/output statement combination that is not allowed is considered an error. After the error is corrected, execution continues.

7.8.1.1 Formatted Sequential Access Input/output Statements

If the current statement is a formatted sequential (including list-directed and namelist) data transfer statement, the immediately preceding input/output statement must not be one of the following statements:

- Unformatted sequential access data transfer
- Direct access data transfer
- Stream access data transfer

If the current statement is READ, the immediately preceding input/output statement must not be WRITE, except the input/output is a terminal.

7.8.1.2 Unformatted Sequential Access Input/output Statements

If the current input/output statement is an unformatted sequential data transfer statement, the immediately preceding input/output statement must not be any of the following statements:

- Formatted sequential data transfer
- List-Directed
- Namelist
- Direct access data transfer
- Stream access data transfer

If the current input/output statement is READ, the immediately preceding input/output statement must not be a WRITE statement.

7.8.1.3 Direct Access Input/output Statements

If the current input/output statement is a direct access data transfer, the immediately preceding input/output statement must not be any of the following statements:

- Sequential data transfer

- List-Directed data transfer
- Namelist data transfer
- Stream access data transfer
- File positioning statement
- CLOSE

7.8.1.4 Stream Access Input/output Statements

If the current input/output statement is a stream access data transfer, the immediately preceding input/output statement must not be any of the following statements:

- Sequential data transfer
- Direct access data transfer
- List-Directed data transfer
- Namelist data transfer
- File positioning statement
- CLOSE

7.8.1.5 File Positioning Statements

By combining sequential input/output statements with file positioning statements, you can control file positioning. The file positioning statements are BACKSPACE, ENDFILE, and REWIND.

When a file is connected for sequential access, the initial file position is immediately before the first record in the file unless POSITION is specified in the OPEN statement.

If the current input/output statement is BACKSPACE, which are used for file positioning, the immediately preceding input/ output statement must not be either of the following statements:

- Direct access data transfer
- Unformatted stream access data transfer

If the current input/output statement is REWIND, which are used for file positioning, the immediately preceding input/

output statement must not be either of the following statements:

- Direct access data transfer

If the current input/output statement is ENDFILE, which are used for file positioning, the immediately preceding input/ output statement must not be either of the following statements:

- Direct access data transfer

7.9 File Positioning Statements

By combining sequential input/output statements with file positioning statements, you can control file positioning. The file positioning statements are BACKSPACE, ENDFILE, and REWIND.

- BACKSPACE Statement (see Section 7.9.1 BACKSPACE Statement)
- ENDFILE Statement (see Section 7.9.2 ENDFILE Statement)
- REWIND Statement (see Section 7.9.3 REWIND Statement)

7.9.1 BACKSPACE Statement

Execution of a BACKSPACE statement causes the file connected to the specified unit to be positioned before the current record if there is a current record, or before the preceding record if there is no current record. If there is no current record and no preceding record, the

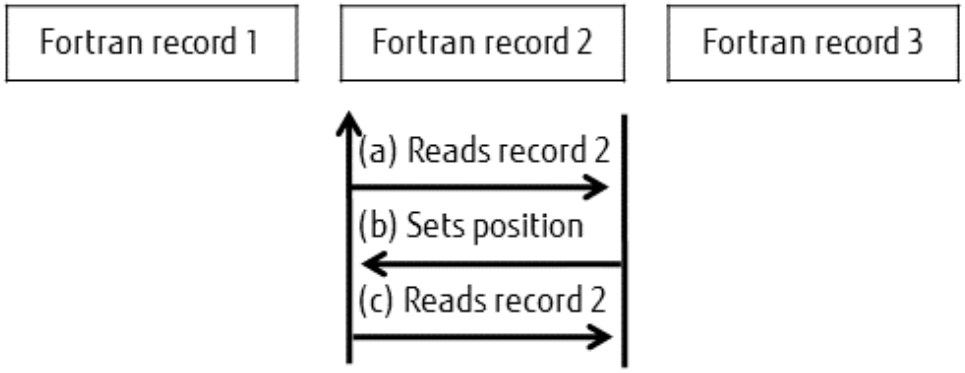
position of the file is not changed. If the preceding record is an endfile record, the file is positioned before the endfile record. If a BACKSPACE statement causes the implicit writing of an endfile record, the file is positioned before the record that precedes the endfile record.

Example 1: A BACKSPACE statement after a READ statement

Fortran program:

```
READ(1) IDATA  ! (a) Reads Fortran record 2
BACKSPACE 1    ! (b) Position is before record 2
READ(1) NDATA  ! (c) Reads Fortran record 2
END
```

File position:



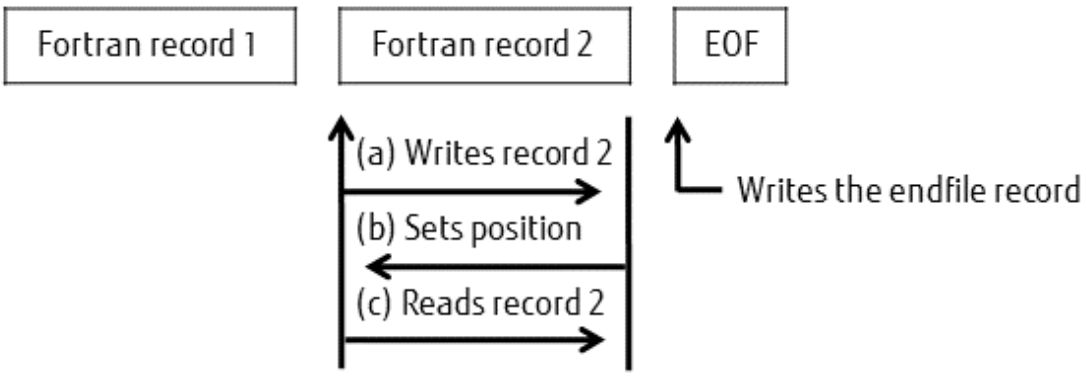
As a result, the same data is read into IDATA and NDATA.

Example 2: A BACKSPACE statement after a WRITE statement

Fortran program:

```
WRITE(1) IDATA     ! (a) Writes Fortran record 2
BACKSPACE 1        ! (b) Position is before record 2
READ(1) NDATA      ! (c) Reads Fortran record 2
END
```

File position:



As a result, the data written by IDATA is read into NDATA.

7.9.2 ENDFILE Statement

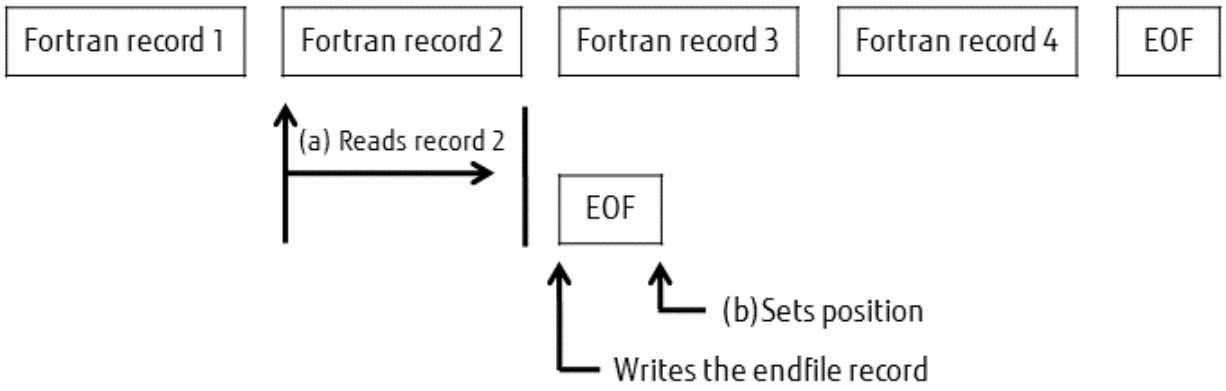
The execution of an ENDFILE statement writes an endfile record as the next record of the file. The file is then positioned after the endfile record, which becomes the last record of the file.

Example 1: An ENDFILE statement after executing a READ statement

Fortran program:

```
READ(1)IDATA   ! (a) Reads Fortran record 2.
ENDFILE 1       ! (b) Writes the endfile record.
```

File position:

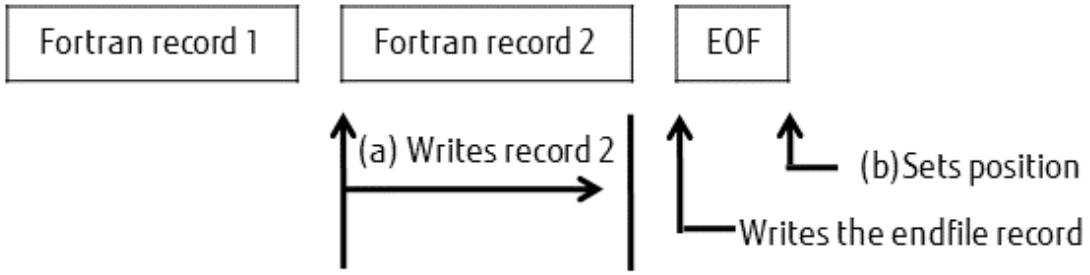


Example 2: An ENDFILE statement after executing a WRITE statement

Fortran program:

```
WRITE(1) IDATA     ! (a) Writes Fortran record 2.
ENDFILE 1           ! (b) Writes the endfile record.
```

File position:

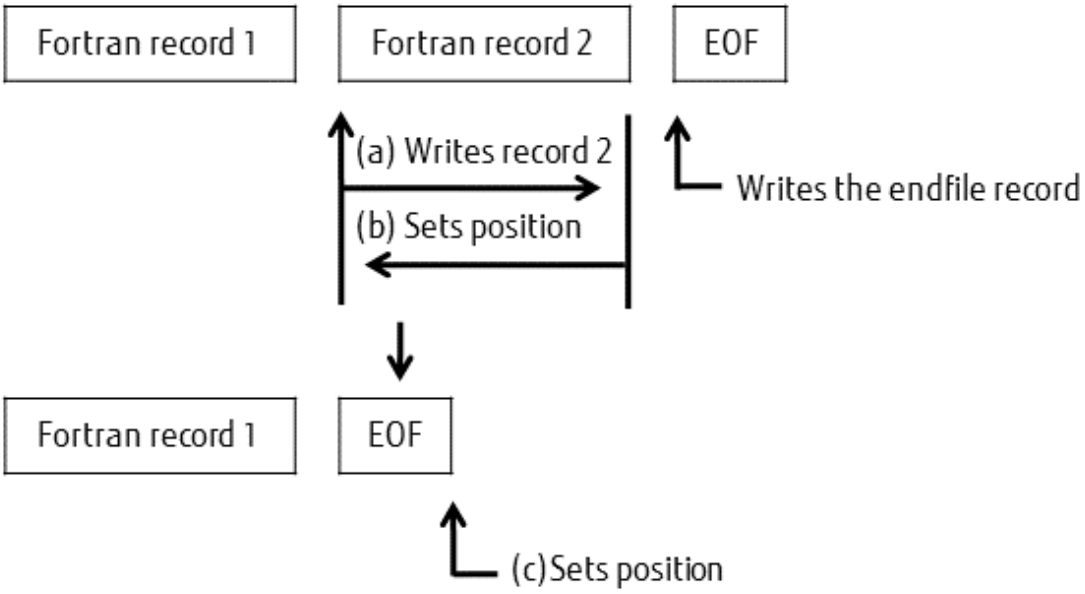


Example 3:An ENDFILE statement after executing a BACKSPACE statement

Fortran program:

```
WRITE(1)IDATA  ! (a) Write Fortran record 2.
BACKSPACE 1    ! (b) Write the endfile record.
ENDFILE 1       ! (c) Erase Fortran record 2.
```

File position:

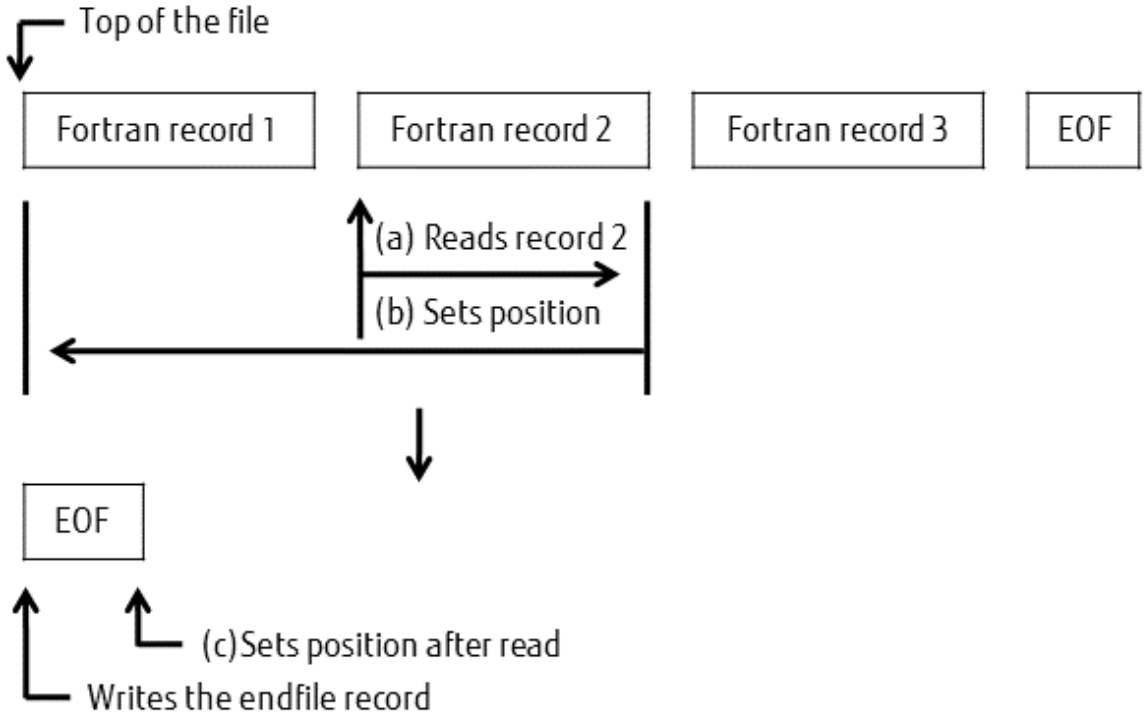


Example 4: An ENDFILE statement after executing a REWIND statement

Fortran Program:

```
READ(1)IDATA   ! (a) Reads Fortran record 2.
REWIND 1        ! (b) The file position is set to the beginning
                  !of the file.
ENDFILE 1       ! (c) Deletes all Fortran records and write the
                 !endfile record.
```

File position:



As a result, the file consists of only an endfile record.

7.9.3 REWIND Statement

When the file is connected using sequential or stream access, the file position is set to immediately before the first record in the file.

7.10 FLUSH statement

Execution of a FLUSH statement causes data written to an external file to be available to other processes, or causes data placed in an external file by means other than Fortran to be available to a READ statement.

Execution of a FLUSH statement for a file that is connected but does not exist has no effect on any file. And when an immediately preceding I/O is not a write statement, it has no effect on any file in the same way.

The UNIT=, IOSTAT= and ERR= specifiers can be specified for the FLUSH statement. See Section 7.7.1 Control Information for Input/output Data Transfer Statements for information on these specifiers.

7.11 Conversion between IBM370-Format Floating-Point Data and IEEE-Format Floating-Point Data on Input/output

This section describes the conversion between IBM370-format floating-point data and IEEE-format floating-point data.

7.11.1 Relationship between Floating-Point Data and Input/output Statements

If the runtime option -C is specified, the following operations are executed.

Unformatted READ statements read IBM370-format floating-point data in the unformatted Fortran records by converting them into internal IEEE-format floating-point data.

Unformatted WRITE statements convert internal IEEE-format floating-point data of the output list items into IBM370-format floating-point data and write the unformatted Fortran records. See Section 3.3 Runtime Options for details on runtime options.

The figures below show how IBM370-format floating-point data is stored in memory. See Section 5.1.3 Real and Complex Type Data for details on how IEEE-format floating-point data is stored in memory.

Default Real (Kind 4)

The figure below shows the bit composition of sign (s), exponent (e), fraction (f).

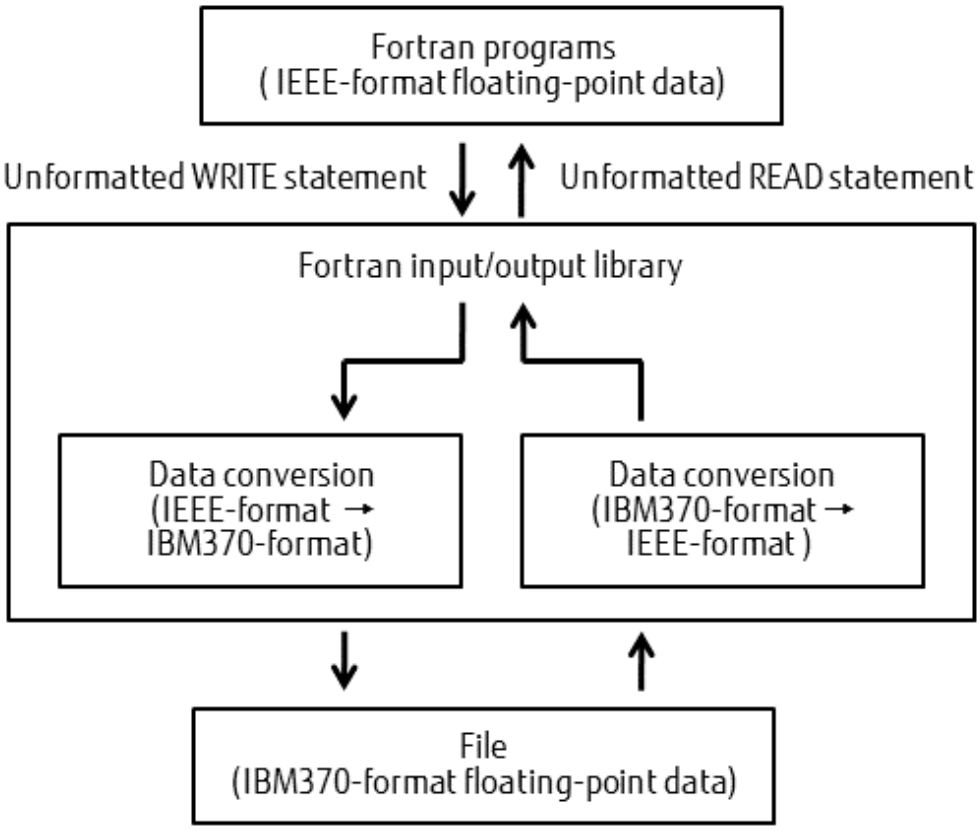
Real Storage Format

Double-Precision (Kind 8)

The figure below shows the bit composition of sign (s), exponent (e), fraction (f).

Double-Precision Storage Format

The flow of IBM370-format floating-point data



The following input/output statements are allowed:

- Unformatted sequential READ/WRITE statement
- Unformatted direct READ/WRITE statement

Files written with variable-length Fortran records by Fujitsu Fortran implementation can be converted.

See Section 7.3.2 Unformatted Fortran Records for details on unformatted records in this Fortran system.

The following types of input/output list items are allowed:

- Default real
- Double-precision real
- Default complex
- Double-precision complex

However, when a type of default complex or double-precision complex is specified for the input/output list items, the real and imaginary parts are converted, respectively, to types of default real and double-precision real. Moreover, conversion processing is not performed when types other than default real, double-precision real, default complex, and double-precision complex are specified for input/output list items. The degree of accuracy when the floating-point data is converted is as follows:

Type of input/ output list items	Range of values			READ: Convert IBM370-format to IEEE-format	WRITE: Convert IEEE-format to IBM370-format
Real	Exponent	IBM370-formats	From 10**(-78) to 10**75	Convertible within the following ranges.	Convertible
		IEEE-formats	From 10**(-37) to 10**38	From 10**(-37) to 10**38 (*1)	
	Fraction	IBM370-formats	24 bits	Convertible	Often, the conversion

Type of input/ output list items	Range of values			READ: Convert IBM370-format to IEEE-format	WRITE: Convert IEEE-format to IBM370-format
		IEEE-formats	24 bits (include a hidden bit)		caused a loss of three bits or less. (*2)
Double-precision real	Exponent	IBM370-formats	From 10**(-78) to 10**75	Convertible	Convertible within the following ranges. From 10**(-78) to 10**75 (*1)
		IEEE-formats	From 10**(-307) to 10**308		
	Fraction	IBM370-formats	56 bits	Often, the conversion caused a loss of three bits or less. (*3)	Convertible
		IEEE-formats	53 bits (include a hidden bit)		

*1: The conversion may cause an overflow or underflow, if the value is outside the range.

*2: The conversion may cause a loss of up to three bits, when up to three bits are added to the top of the fraction bit field depending on the exponent value.

*3: The conversion may cause a loss of up to three bits when the number of bits that the fraction occupies is different from ranges.

Note:

The conversion always sets the least significant bit of the fraction to 1 in order to minimize the error when the conversion causes a loss of bits. A diagnostic message is output if runtime option -M is specified.

7.11.2 Error Processing in the Conversion of Floating-Point Data

Conversion	Data types	Error event	Diagnostic message	Standard correction value
Convert IEEE-format data to IBM370-format data: WRITE	Default real	Inf is detected	jwe0145i-w	Maximum value
		NaN is detected	jwe0145i-w	Maximum value
		Missing bit is detected	jwe0147i-w (*1)	-------
	Double-precision real	Exponent overflow is detected	jwe0142i-w	Maximum value
		Exponent underflow is detected	jwe0144i-w	0
		Inf is detected	jwe0146i-w	Maximum value
		NaN is detected	jwe0146i-w	Maximum value
Convert IBM370-format data to IEEE-format data: READ	Default real	Exponent overflow is detected	jwe0141i-w	Maximum value
		Exponent underflow is detected	jwe0143i-w	0
	Double-precision real	Missing bit is detected	jwe0147i-w (*1)	-------

*1: A diagnostic message is output if the runtime option -M is specified.

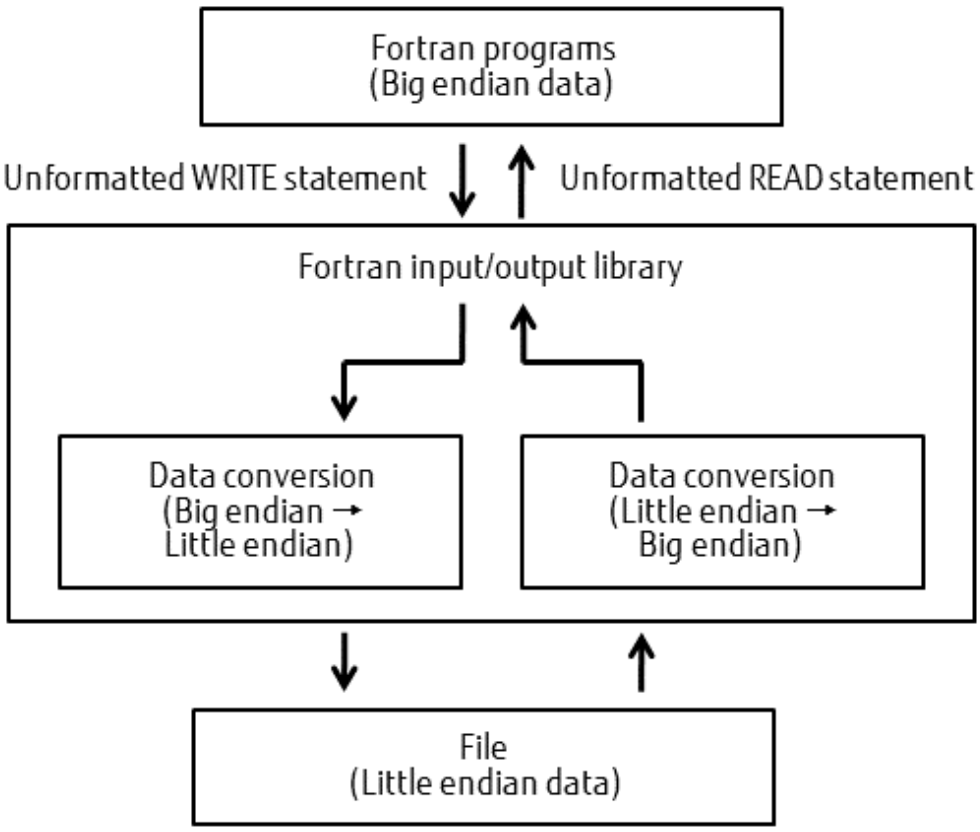
7.12 Conversion between Little Endian Data and Big Endian Data on Input/output

This section describes the conversion between little endian data and big endian data.

7.12.1 Relationship between Little Endian Data and Input/output Statements

You may specify the runtime option -T to read and write little endian data. If the runtime option -T is specified, the following operations are executed. See Section 3.3 Runtime Options for information on the runtime option -T.

The flow of little endian data



The following input/output statements are allowed:

- Unformatted sequential READ/WRITE statement
- Unformatted direct READ/WRITE statement
- Unformatted stream READ/WRITE statement

Files written with variable-length Fortran records by Fujitsu Fortran implementation can be converted.

See Section 7.3.2 Unformatted Fortran Records for details on unformatted records in this Fortran system.

The following types of input/output list items are allowed:

- 1-byte integer
- 2-byte integer
- 4-byte integer
- 8-byte integer
- Default real

- Double-precision real
- Quadruple-Precision real
- Default complex
- Double-precision complex
- Quadruple-Precision complex
- 1-byte logical
- 2-byte logical
- 4-byte logical
- 8-byte logical

7.13 Standard Files

7.13.1 Standard Files and Open Modes

The table below shows the relationship between standard files and open modes.

Input/output file	Specification method (terminal open mode if no specification)	Open mode
Standard input	command < file-name	r
	command << eof-sym	r
Standard output	command > file-name	w
	command >> file-name	a
Standard error	command 2> file-name	w
	command 2>> file-name	a

7.13.2 Restrictions on Input/output Statements for Standard Files

Only sequential access data transfer and non-data-transfer input/output statements may be used for standard input, standard output, and error output files. Standard input files are processed as ACTION='READ'.

Standard output and standard error output files are processed as ACTION='WRITE'.

Executing an OPEN statement with a FILE= specifier for a standard input, standard output, or standard error output unit terminates any existing connection and opens a new file. The unit number of the standard input, standard output, or standard error output file is processed in the same way as a regular file.

7.13.3 Relationship between Input/output Statements and Seekable and Nonseekable Files

Seekable files are files for which seek operations can be performed. The differences between seekable and nonseekable files are as follows:

Seekable or nonseekable	File type
Seekable file	Regular file
Nonseekable file	Block-type special file
	Character-type special file
	FIFO special file

Note:

File type refers to information obtained from stat or fstat system calls.

The input/output statements and edit descriptors listed here may be executed for seekable files. If they are executed for nonseekable files, a diagnostic message is output, and the statement is ignored.

- REWIND statement
- BACKSPACE statement
- ENDFILE statement (INPUT system)
- Direct access statement
- Stream access statement
- Unformatted sequential statement
- TL and T edit descriptors (only when the same operation as that using the TL edit descriptors is performed)

7.14 Restrictions Related to the ACTION= Specifier

Input/output statements with files that have been connected using the ACTION= specifier have the following restrictions.

Input/output statement	ACTION= specifier		
	READ	WRITE	READWRITE
READ	o	x	o
WRITE	x	o	o
PRINT	x	o	o
REWIND	o	x	o
BACKSPACE	o	x	o
OPEN	o	o	o
CLOSE	o	o	o
ENDFILE	--	o	o

o: Normal operation.

x: An error message is generated. Respond accordingly.

--: No operation is performed.

7.15 Number of Significant Digits

The number of significant digits for real type is defined in this system as follows:

Type	Number of significant digits
Default real	9
Double-precision real	16
Quadruple-precision real	35
Default complex	9
Double-precision complex	16
Quadruple-precision complex	35

7.16 I/O Buffer

The table below shows how to change the size of a sequential I/O buffer. The precedence order is summarized in the below table.

Operators	Size of buffer	Precedence order
BLOCKSIZE= Specifier	Defined Value	High
Environment variable (fuxxbf)	Defined Value	.
Runtime option (-g)	Defined Value	.
Default	8M bytes	Low

7.17 Edit Descriptors

7.17.1 Generalized Integer Editing

In FORTRAN66 and FORTRAN77, the Gw.d and Gw.dEe edit descriptors follow the rules for the Iw.m edit descriptor when used to specify the input/output of integer data.

In Fortran 95 and Fortran 2003, the Gw.d and Gw.dEe edit descriptors follow the rules for the Iw edit descriptor.

Example:

```
WRITE(*,'(G10.5)')123
```

In FORTRAN66 and FORTRAN77, the output is -----00123 (- indicates a blank).

In Fortran 95, the output is -------123.

7.17.2 Exponent Character in Formatted Output

In FORTRAN66 and FORTRAN77, the exponent character is output in lowercase for E, EN, ES, D, G, L, Q, and Z editing. If runtime option -q is specified, the character is output in uppercase.

In Fortran 95 and Fortran 2003, the character is output in uppercase whether runtime option -q is specified or not.

Example:

```
WRITE(*,FMT='(1H E15.5)')6.5432
```

In FORTRAN66 and FORTRAN77, the output is 0.65432e+01.

In Fortran 95, the output is 0.65432E+01.

7.17.3 Result of G Editing

In FORTRAN66 and FORTRAN77, a zero value is edited using the E edit descriptor. In Fortran 95 and Fortran 2003, a zero value is edited using the F edit descriptor.

7.17.4 Diagnostic Messages for the Character String Edit Descriptor

In FORTRAN66 and FORTRAN77, the system outputs a level w diagnostic message (jwe0159i-w) for a character string edit descriptor in a format used for input. The character string edit descriptor is replaced by input data.

In Fortran 95 and Fortran 2003, the system outputs a level e diagnostic message (jwe1181i-e). The input list is ignored.

7.17.5 Effect of the X Edit Descriptor

If the X edit descriptor (nX) is used for output, the result differs between FORTRAN66 and other language version.

In FORTRAN77, Fortran 95 and Fortran 2003, the record character position is advanced by n characters.

Blanks are not inserted.

In FORTRAN66, n blanks are inserted in the record.

This operation differs only if a left-positioning tab is used:

Example:

```
FORMAT (5H FFFF,I4,T2,4X,I2)
```

The Fortran program operates as follows:

```
FORTRAN77, Fortran 95 and Fortran 2003 program: -FFFFjjii
FORTRAN66 program:                   -----jjii
```

The two low-order digits of I4 are represented by ii, I2 is represented by jj, and - indicates a blank.

For FORTRAN77, Fortran 95 and Fortran 2003 programs, the Fortran record is first padded with blanks.

7.17.6 L Edit Descriptor

In FORTRAN66 or the FORTRAN77, if the type of the input item corresponding to the Lw type edit descriptor is not a logical constant, the input item is assigned the value FALSE.

In Fortran 95 and Fortran 2003, a diagnostic message (jwe1202i-w) is output.

Example:

Program a.f:

```
LOGICAL :: L=.TRUE.
OPEN (10,FILE='a.file')
READ (10,FMT='(L1)') L
PRINT *,L
END
```

file a.file:

```
1
```

In FORTRAN66 and FORTRAN77:

```
$ ./a.out
 f
$
```

In Fortran 95 and Fortran 2003:

```
$ ./a.out
jwe1202i-w line 3 Invalid LOGICAL input (unit=10).
 error occurs at MAIN__   line 3 loc 0000000000400b96 offset 0000000000000036
 MAIN__       at loc 0000000000400b60 called from o.s.
taken to (standard) corrective action, execution continuing.
 T
error summary (Fortran)
error number  error level  error count
  jwe1202i         w           1
total error count = 1
$
```

7.17.7 D, E, F, G, I, L, B, O, and Z Edit Descriptor without w or d Indicators

The D, E, F, G, I, L, B, O, or Z edit descriptor may omit the width (*w*) or the number of decimals (*d*).

For the I*w*, B*w*, O*w*, Z*w*, F*w*, E*w*, D*w*, L*w*, and G*w* edit descriptors, the Fortran system selects the field width when *w* is omitted.

The table below shows the selected width.

Edit descriptor	Type													
	I1	I2	I4	I8	R4	R8	R16	C8	C16	C32	L1	L2	L4	L8
B	9	17	33	65	33	65	129	33	65	129	9	17	33	65
O	4	7	12	23	12	23	44	12	23	44	4	7	12	23
Z	3	5	9	17	9	17	33	9	17	33	3	5	9	17
I	4	6	11	20	11	11	11	11	11	11	4	6	11	20
L	2	2	2	2	2	2	2	2	2	2	2	2	2	2
F	15	15	15	22	15	22	43	15	22	43	15	15	15	22
E	15	15	15	22	15	22	43	15	22	43	15	15	15	22
D	15	15	15	22	15	22	43	15	22	43	15	15	15	22
G	4	6	11	20	15	22	43	15	22	43	2	2	2	2

I1: 1 byte integer

I2: 2 byte integer

I4: 4 byte integer

I8: 8 byte integer

R4: default real

R8: double-precision real

R16: quadruple-precision real

C8: default complex

C16: double-precision complex

C32: quadruple-precision complex

L1: 1 byte logical

L2: 2 byte logical

L4: 4 byte logical

L8: 8 byte logical

7.17.8 Editing of Special Real-Type Data

When using one of the E, D, Q, G, F, EN, or ES edit descriptors, Inf or NaN are output right-justified within width w of the field when E, D, Q, G, F, EN, and ES edit descriptors are specified in the format Xw.d in Fortran 2003. In FORTRAN66, FORTRAN77 and Fortran 95,Inf or NaN are output left-justified.

If the output width has fewer characters than the field width w, it will be padded with spaces. -Inf or -NaN is output for a negative value.

Moreover, the sign can be edited with the S/SP/SS edit descriptors.

Character strings (Inf/-Inf, Infinity/-Infinity, NaN/-NaN, NaN()/-NaN()) can be used as the input data of a real number. It is zero or more alphanumeric characters parentheses of NaN(). The positive sign "+" can be omitted.

The table below shows the correspondence between strings in a file and the values set in the variables.

Correspondence between Strings and Values Set in Variables

Strings	Default real	Double-precision	Quadruple-precisio
+NaN or +NAN()	0x7fffffff	0x7fffffff ffffffff	0x7fffffffffffffff ffffffffffffffff
-NaN	0xffffffff	0xffffffff ffffffff	0xffffffffffffffff ffffffffffffffff

Strings	Default real	Double-precision	Quadruple-precisio
or -NAN()			
+Inf or +Infinity	0x7f800000	0x7ff00000 00000000	0x7fff000000000000 0000000000000000
-Inf or -Infinity	0xff800000	0xfff00000 00000000	0xffff000000000000 0000000000000000

7.18 Magnetic Tape File I/O

The magnetic tape file I/O is explained here.

When the unit number is connected with the magnetic tape file by the environment variable or the OPEN statement, tape file I/O is possible.

The following I/O statement can be used.

- Unformatted sequential I/O statement
- Formatted sequential I/O statement
- File connection statement
- File positioning statement
- File inquiry statement

7.18.1 Connection of Magnetic Tape File and Unit Number

The method of connecting the magnetic tape file and the unit number is explained.

7.18.1.1 Connection by Environment Variable

The magnetic tape file can be connected with the unit number by the environment variable.

Example: The magnetic tape file is connected with unit number 1.

Program:

```
OPEN(1)
  ...
```

Command string:

```
$ export fu01="/dev/rmt/0n"
$ ./a.out
```

7.18.1.2 Connection by FILE Specifier of OPEN Statement

A FILE specifier of the OPEN statement can connect magnetic tape file with the unit number.

Example: The magnetic tape file is connected with unit number 10.

Program:

```
OPEN(10,FILE='/dev/rmt/0n')
 ...
```

7.18.2 Note of Magnetic Tape File I/O Function

Here, the note in the magnetic tape file I/O function use is explained.

- When the program terminates abnormally, the place where the tape is located might become uncertain.
  Please locate the tape at an appropriate position by using the mt(1) command.
- Executing option -g is ineffectual.

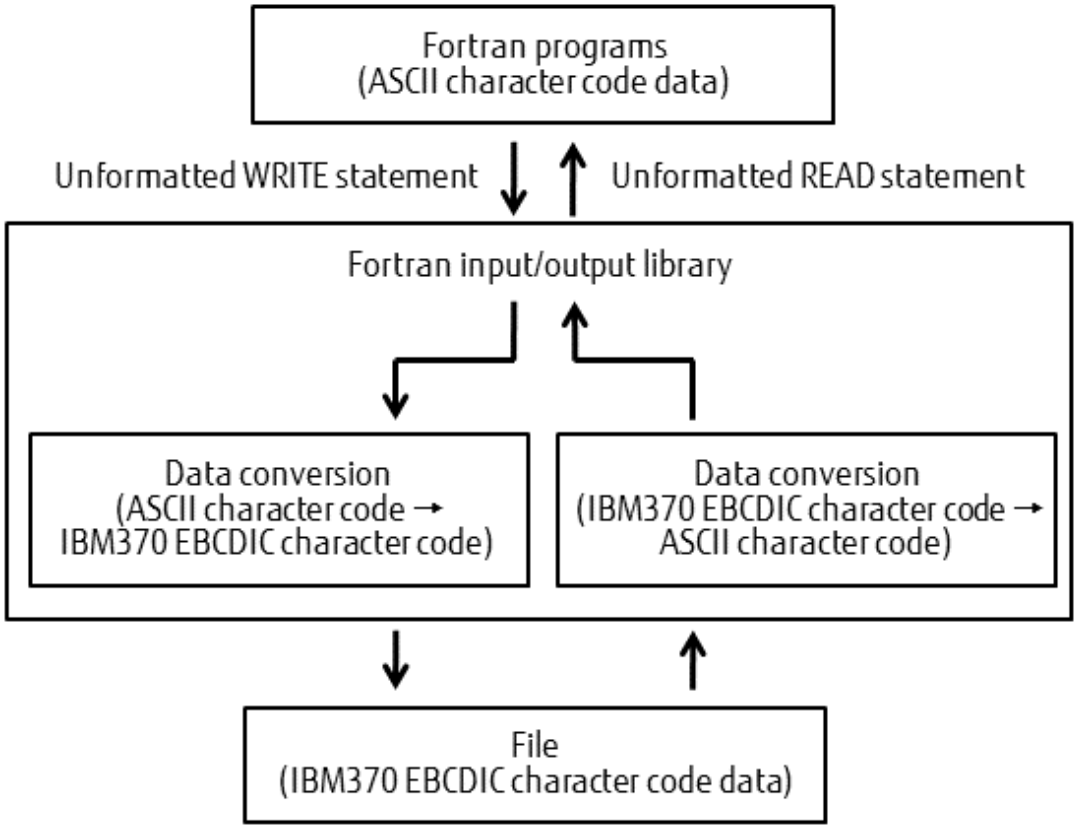
7.19 Conversion between EBCDIC Character Code and ASCII Character Code on Input/output

This section describes the conversion between IBM370 EBCDIC character code data and ASCII character code data.

7.19.1 Relationship between EBCDIC Character Code Data and Input/output Statements

You may specify the runtime option -G to read and write IBM370 EBCDIC character code data. If the runtime option -G is specified, the following operations are executed. See Section 3.3 Runtime Options for information on the runtime option -G.

The flow of IBM370 EBCDIC character code data.



The following input/output statements are allowed:

- Unformatted sequential READ/WRITE statement
- Unformatted direct READ/WRITE statement

Files written with variable-length Fortran records by Fujitsu Fortran implementation can be accessed by an unformatted input statement.

Character type of input/output list items are allowed.

7.20 Asynchronous Input/output Statements

This section describes asynchronous input/output statements.

7.20.1 Execution of Asynchronous Input/output Statements

An input/output statement that includes an ASYNCHRONOUS= specifier with a YES value is executed as an asynchronous input/output statement when it is used in relation to a file connected via an ASYNCHRONOUS= specifier with a YES value specified in an OPEN Statement.

An asynchronous input/output statement is processed by making data transfer asynchronous for any data that can be transferred asynchronously within the system.

Note, however, that asynchronous data transfer is not enabled for input/output statements other than those for unformatted sequential access or when the following compiler options or runtime options are specified:

- Compiler option -H (see Section 2.2 Compiler Options)
- Compiler option -Eg (see Section 2.2 Compiler Options)
- Runtime option -G (see Section 3.3 Runtime Options)
- Runtime option -C (see Section 3.3 Runtime Options)
- Runtime option -T (see Section 3.3 Runtime Options)

In addition, asynchronous data transfer may not be enabled depending on the number, size, kind, type, or attributes of items that have been specified in an input/output list.

7.20.2 ID= Specifier in Data Transfer Input/output Statements

A literal value less than 9223372036854775808 more than 0 that uniquely identifies an asynchronous input/output statement executed during the course of a program is assigned to an ID= specifier in a data transfer input/output statement. When a value that has been assigned to an ID= specifier exceeds the allowed range of ID= specifier values, that value is not guaranteed.

Chapter 8 Debugging

This chapter describes the system debugging functions provided for finding and correcting errors in source programs.

8.1 Error Processing

If an error occurs during the execution of a Fortran program, the error monitor controls the following operations:

- Generation of a diagnostic message
- Continuation of execution if allowed by the error type

By using the error monitor, the user can control the following:

- The error count limit for canceling program execution
- The printed message count limit
- Whether to print the trace back map
- Whether to call user-defined error subroutines

If the user does not specify these control operations, the system uses its default operations. For details, see "Table 8.1 Error control table standard values" in the next section.

8.1.1 Error Monitor

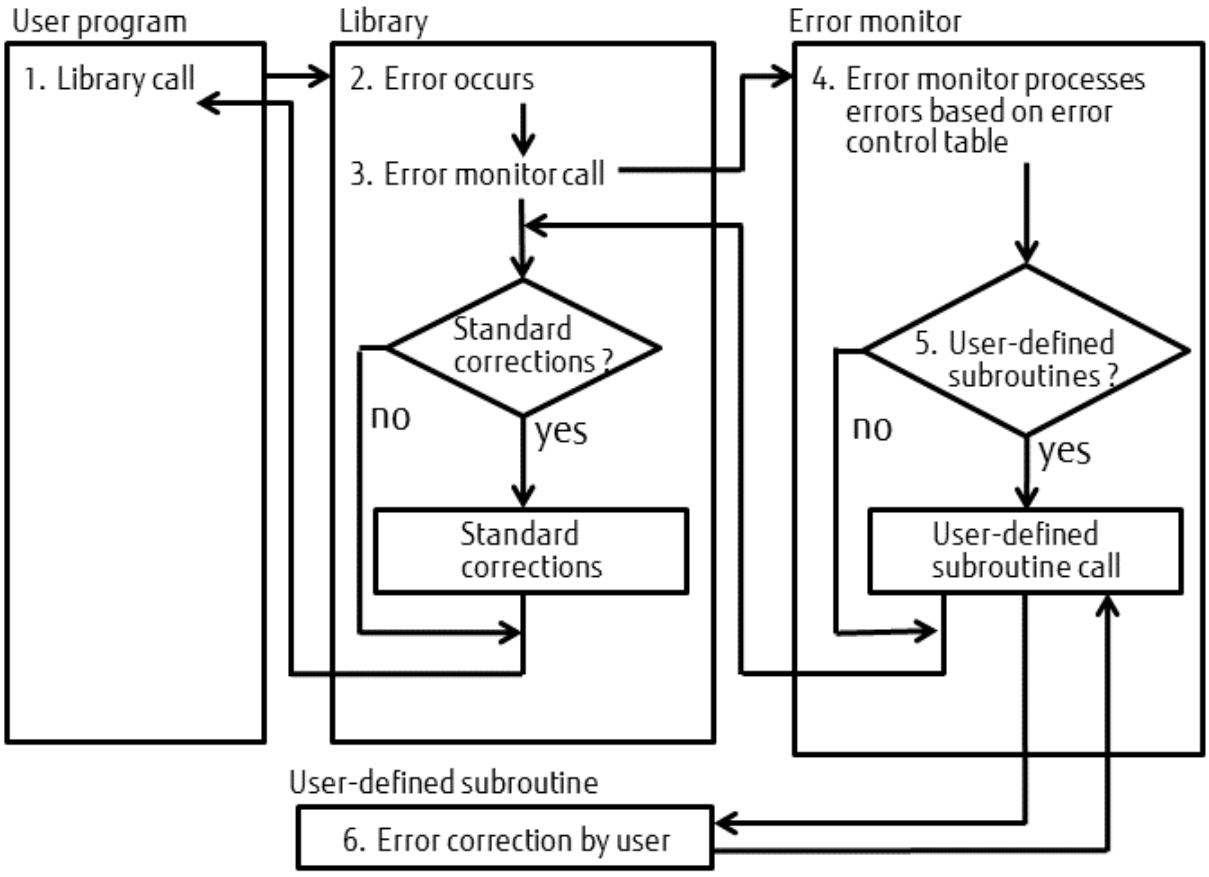
If an error occurs during execution, the error monitor is called. The error monitor processes errors based on the error control table.

The error monitor does the following:

- Checks whether the error count limit has been reached and notifies the user if it has.
- Checks whether the printed message count limit has been reached and prints a diagnostic message if it has not.
- Generates a trace back map if requested. If the Fortran program is linked to a program written in another language, some trace back map information may not be generated.
- Terminates the job if the error count limit has been reached. If the limit has not been reached, corrective action is taken and execution continues.

Corrective action for errors consists of standard corrections provided by the system and the execution of user-defined subroutines.

The figure below shows how the error monitor is called after an error.



1 - 6 : Call flow when an error occurs in libraries

Error monitor operations are based on data in the error control table. The error control table consists of 12-byte error items, as follows:

0	*estop*	*mprint*	*ecount*	*inf*	(unused)
8	*uexit*				
16	*ecode*		(unused)		
24					

The error items are:

estop : At byte 0, an error count limit in the range 0 to 255. Zero indicates that unlimited errors are allowed.

mprint : At byte 1, a printed message limit in the range 0 to 255. Zero indicates that no messages are printed.

ecount : At byte 2, an error count during execution, in the range 0 to 255.

inf : At byte 3, an error control item, as follows:

- Bit 0 : (1) The control character is set to the diagnostic messages. (0) The control character is not set to the diagnostic messages. See Section "8.1.2.3 ERRSET Subroutine".
- Bit 1 : (1) The user can correct the error item. (0) The user cannot correct the error item.
- Bit 2 : (1) The error count exceeds 255. (0) The error count does not exceed 255.
- Bit 3 : (1) The buffer is printed. (0) The buffer is not printed.
- Bit 4 : Not used
- Bit 5 : (1) Messages are printed. (0) Message printing is based on the value of *mprint*.
- Bit 6 : (1) The trace back map is printed. (0) The trace back map is not printed.

Bit 7 : Not used

uexit : At byte 4, the address of a user-defined error subroutine. If standard error correction is used, the last bit is set to 1.

ecode : At byte 8, an error level, as follows:

0 : i level. Fortran program processing continues after the message is printed.

4 : w level. Fortran program processing continues after the error is corrected.

8 : e level. The error statement is ignored. Corrective action is taken and execution continues if the error count limit has not been reached. The default error count limit is 10.

12 : s level. The error statement is ignored. Corrective action is taken and execution continues if the error count limit has not been reached. The default error count limit is 1.

16 : u level. Fortran program processing terminates.

Some error items cannot be edited. For editable items, use the ERRSET subroutine to change the standard values in the error control table. The Error Control Table Standard Values indicates whether error items are editable, and gives the default values for each item in the table.

Table 8.1 Error control table standard values

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
11 to 14	1	1	Not Editable	Not Printed	Not Printed	None	u
17 to 18	1	1	Not Editable	Not Printed	Not Printed	None	u
19	1	1	Not Editable	Not Printed	Printed	None	u
20	1	1	Not Editable	Not Printed	Not Printed	None	u
21 to 22	1	1	Editable	Not Printed	Printed	Yes	s
28	1	1	Editable	Not Printed	Printed	Yes	s
42	10	5	Editable	Not Printed	Printed	Yes	e
55	10	5	Editable	Not Printed	Printed	Yes	e
62	10	5	Editable	Not Printed	Printed	Yes	e
64 to 65	10	5	Editable	Not Printed	Printed	Yes	e
71 to 72	10	5	Editable	Not Printed	Printed	Yes	e
74	10	5	Editable	Not Printed	Printed	Yes	e
80 to 82	10	5	Editable	Not Printed	Printed	Yes	e
83 to 86	Unlimited	5	Editable	Not Printed	Printed	Yes	w
87 to 89	10	5	Editable	Not Printed	Printed	Yes	e
97	Unlimited	5	Editable	Not Printed	Printed	Yes	w
102 to 103	10	5	Editable	Not Printed	Printed	Yes	e
104	Unlimited	5	Editable	Not Printed	Printed	Yes	w
105	10	5	Editable	Not Printed	Printed	Yes	e
109	Unlimited	5	Editable	Not Printed	Printed	Yes	w
111 to 115	10	5	Editable	Not Printed	Printed	Yes	e
120	10	5	Editable	Not Printed	Printed	Yes	e
122	10	5	Editable	Not Printed	Printed	Yes	e
131 to 134	10	5	Editable	Not Printed	Printed	Yes	e
140 to 147	Unlimited	5	Editable	Not Printed	Printed	Yes	w

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
151	10	5	Editable	Not Printed	Printed	Yes	e
152	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w
153 to 154	Unlimited	5	Editable	Not Printed	Printed	Yes	w
155 to 157	10	5	Editable	Not Printed	Printed	Yes	e
158 to 159	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w
162	Unlimited	5	Editable	Not Printed	Printed	Yes	w
171 to 172	10	5	Editable	Not Printed	Printed	Yes	e
173 to 175	Unlimited	5	Editable	Printed	Printed	Yes	w
176	10	5	Editable	Not Printed	Printed	Yes	e
182 to 183	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w
184 to 189	10	5	Editable	Printed	Printed	Yes	e
190	Unlimited	1	Editable	Not Printed	Not Printed	Yes	w
202 to 255	10	5	Editable	Not Printed	Printed	Yes	e
256	Unlimited	1	Editable	Not Printed	Printed	Yes	w
257	10	5	Editable	Not Printed	Printed	Yes	s
259 to 282	10	5	Editable	Not Printed	Printed	Yes	e
291	10	5	Editable	Not Printed	Printed	None	e
292	1	1	Not Editable	Not Printed	Not Printed	None	u
301	10	5	Editable	Not Printed	Printed	Yes	w
304	Unlimited	5	Editable	Not Printed	Printed	Yes	w
306	1	1	Not Editable	Not Printed	Not Printed	None	u
308	1	1	Editable	Not Printed	Printed	Yes	s
311 to 318	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
320	10	5	Editable	Not Printed	Printed	Yes	w
322	10	5	Editable	Not Printed	Printed	Yes	w
323 to 324	Unlimited	5	Editable	Not Printed	Printed	Yes	w
329	1	1	Editable	Not Printed	Printed	Yes	s
330	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
1001	1	1	Editable	Not Printed	Printed	Yes	s
1003 to 1008	1	1	Editable	Not Printed	Printed	Yes	s
1031	1	1	Editable	Not Printed	Printed	Yes	s
1032	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1033	Unlimited	5	Editable	Not Printed	Printed	Yes	i
1034	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1035	1	1	Not Editable	Not Printed	Printed	None	u
1036	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1038	1	1	Editable	Not Printed	Printed	Yes	s
1039	1	1	Not Editable	Not Printed	Printed	None	u
1040 to 1041	1	1	Editable	Not Printed	Printed	Yes	s

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
1042	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1043	Unlimited	5	Editable	Not Printed	Not Printed	Yes	i
1044 to 1046	1	1	Not Editable	Not Printed	Printed	None	u
1047	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1048	1	1	Not Editable	Not Printed	Printed	None	u
1049	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1050	Unlimited	5	Editable	Not Printed	Not Printed	Yes	w
1071 to 1072	10	5	Editable	Not Printed	Printed	Yes	e
1073	10	5	Editable	Not Printed	Printed	Yes	e
1111 to 1113	10	5	Editable	Not Printed	Printed	Yes	e
1114 to 1115	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1117 to 1122	10	5	Editable	Not Printed	Printed	Yes	e
1141 to 1143	10	5	Editable	Not Printed	Printed	Yes	e
1146	10	5	Editable	Not Printed	Printed	Yes	e
1147	10	5	Editable	Not Printed	Printed	Yes	e
1148	10	5	Editable	Not Printed	Printed	Yes	e
1149	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1150	10	5	Editable	Not Printed	Printed	Yes	e
1161	1	1	Editable	Not Printed	Printed	Yes	s
1162 to 1163	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1181	10	5	Editable	Not Printed	Printed	Yes	e
1201 to 1203	Unlimited	5	Editable	Printed	Printed	Yes	w
1231	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1232	1	1	Editable	Not Printed	Printed	Yes	s
1301	10	5	Editable	Not Printed	Printed	Yes	e
1302	Unlimited	5	Editable	Not Printed	Printed	Yes	i
1303 to 1304	10	5	Editable	Not Printed	Printed	Yes	e
1305 to 1306	1	1	Editable	Not Printed	Printed	Yes	s
1355 to 1370	10	5	Editable	Not Printed	Printed	Yes	e
1371	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1372	10	5	Editable	Not Printed	Printed	Yes	e
1373	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1375 to 1378	1	1	Editable	Not Printed	Printed	Yes	s
1381 to 1387	1	1	Editable	Not Printed	Printed	Yes	s
1390 to 1392	1	1	Editable	Not Printed	Printed	Yes	s
1393	10	5	Editable	Not Printed	Printed	Yes	e
1394 to 1395	1	1	Editable	Not Printed	Printed	Yes	s
1396 to 1414	10	5	Editable	Not Printed	Printed	Yes	e

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
1415	1	1	Editable	Not Printed	Printed	Yes	s
1416 to 1417	10	5	Editable	Not Printed	Printed	Yes	e
1418 to 1419	1	1	Editable	Not Printed	Printed	Yes	s
1420	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1421	10	5	Editable	Not Printed	Printed	Yes	e
1422 to 1423	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1425 to 1454	10	5	Editable	Not Printed	Printed	Yes	e
1455	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1551	10	5	Editable	Not Printed	Printed	Yes	e
1561	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
1563 to 1564	1	1	Not Editable	Not Printed	Not Printed	None	u
1565 to 1566	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
1569 to 1571	Unlimited	Unlimited	Editable	Not Printed	Printed	Yes	w
1572 to 1575	1	1	Editable	Not Printed	Printed	Yes	s
1576 to 1577	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1578	1	1	Editable	Not Printed	Printed	Yes	s
1579	Unlimited	5	Editable	Not Printed	Printed	Yes	w
1582	Unlimited	5	Editable	Not Printed	Printed	Yes	i
1652	Unlimited	Unlimited	Editable	Not Printed	Not Printed	Yes	w

If an error occurs, the error monitor prints the following message (indicating correction status) after the trace back map:

```
taken to (corrector) corrective action, execution continuing.
```

corrector: {standard|user}

standard: indicates that an error is corrected with standard correction.

user: indicates that an error is corrected with user-defined correction.

8.1.2 Error Subroutines

Several service subroutines provide user control of error processing during execution. These subroutines can be used to optimize error monitor use.

With these subroutines, you can dynamically change items in the error control table, generate a trace back map, and execute user-defined error subroutines.

8.1.2.1 ERRSAV Subroutine

ERRSAV stores the leading 16 bytes of the specified error item in 16 bytes user area.

The ERRSAV service subroutine calling format is as follows:

CALL ERRSAV (*errno* , *darea*)

errno

Default integer scalar. Error number.

darea

16 bytes character scalar. The error item is saved.

If the type of *darea* is not character, this result may be undefined.

8.1.2.2 ERRSTR Subroutine

ERRSTR moves an error item to the error control table element for the specified error number. Thus, when an error occurs, the new error item controls error processing.

The ERRSTR service subroutine calling format is as follows:

CALL ERRSTR (*errno* , *darea*)

errno

Default integer scalar. Error number.

darea

16 bytes character scalar. The error item is set.

If the type of *darea* is not character, this result may be undefined.

Example: ERRSTR and ERRSAV

```
CHARACTER(LEN=16) ERR113 !(1)ESTOP,(2)MPRINT,(3)ECOUNT,(4)INF
CALL ERRSAV(113,ERR113)
WRITE(ERR113(1:2),'(2a1)')0,0
CALL ERRSTR(113,ERR113)
OPEN(10,FILE='X.DAT',FORM='FORMATTED')
DO I=1,15
   WRITE(10)I
END DO
CLOSE(10,STATUS='DELETE')
END
```

8.1.2.3 ERRSET Subroutine

ERRSET changes the control table data for the error item with the specified error number. The error count limit, message print limit, trace back map printing switch, and error corrective action may be changed.

The ERRSET service subroutine calling format is as follows:

CALL ERRSET (*errno* , *estop* , *mprint* , *trace* , *uexit* , *r*)

errno

Default integer scalar. Error number.

estop

Default integer scalar. Error count limit.

<=0 : Error count limit is the same as before.

>=256 : It indicates unlimited error counts.

mprint

Default integer scalar. Message print limit.

<0 : Message print limit is zero (that is, a no-print message).

=0 : Message print limit is the same as before.

>=256 : It indicates unlimited message prints.

trace

Default integer scalar. One of the following values indicates whether to print a trace back map:

=0 : Do not change.

=1 : Do not print.

=2 : Print.

uexit

Default integer scalar. Error correction or user-defined correction subroutine indicated by one of the following:

=0 : Do not change.

=1 : Perform standard correction.

=other : Execute the user-defined correction.

r

If errno is not 132, a four-byte integer expression indicates the largest error number to be changed.

If errno is 132, indicates whether a blank character is inserted at the beginning of the correction data with the following values:

=1 : Insert a blank character.

=other : Do not insert a blank character.

Example: ERRSET Service Subroutine

```
EXTERNAL FIXUP
CALL ERRSET(202,50,-1,0,FIXUP,205)
```

In this example, the error items for error numbers 202 to 205 are changed as follows:

- The error count limit is changed to 50.
- A message will not be printed if an error occurs.
- The print control information for the trace back map is not changed.
- The user-defined subroutine FIXUP is called to process errors.

8.1.2.4 ERRTRA Subroutine

The ERRTRA service subroutine generates a trace back map up to the program unit currently executing. See Section "4.2 Executable Program Output" for the map format. Execution continues after this subroutine is called.

The ERRTRA service subroutine calling format is as follows:

CALL ERRTRA

Example: ERRTRA Service Subroutine

```
OPEN(10,FILE='X.DAT')
CALL SUB1
END
SUBROUTINE SUB1
CALL ERRTRA
DO I=1,15
   WRITE(10,*)I
END DO
END
```

8.1.3 Using the Error Monitor

By calling the error monitor, the user can control error processing, specify user-defined error subroutines, and use the error service functions. Related topics explain how to use the error service functions.

8.1.3.1 User-Defined Error Subroutines

The ERRSET subroutine may be used to store the address of a user-defined error subroutine in the error control table. If a user-defined error subroutine has been specified, the error monitor calls the error subroutine as follows:

CALL user (ret , errno [, data1 , ... , datan])

user : Name of the user-defined error subroutine.

ret : four-byte integer scalar variable that stores a return code. The return code must be provided by the user-defined error subroutine.

errno : Number of the detected error.

data1, ..., datan : Scalar variable names or array element name list used by the user-defined subroutine.

The arguments passed to the user-defined error subroutine are as follows:

- Input-output error: The arguments in Section "8.1.4 Input/Output Error Processing" are passed.
- Intrinsic function error: The arguments in Section "8.1.5 Intrinsic Function Error Processing" are passed.
- Intrinsic subroutine error: The arguments in Section "8.1.6 Intrinsic Subroutine Error Processing" are passed.
- Interrupt error: The arguments in Section "8.1.7 Exception Handling Processing" are passed.
- Otherwise, the arguments in Section "8.1.8 Other Error Processing" are passed.

The following restrictions apply to user-defined error subroutines:

- Restrictions on input/output statements.
  If an error occurs, input/output statements cannot be used with the same unit number as the unit with the error. For example, if an error occurs with unit number 10, input/output statements with unit number 10 cannot be used.
- Restrictions on intrinsic functions.
  If an error occurs, the intrinsic function that caused the error cannot be used in the user-defined subroutine. For example, if an argument value is incorrect in SQRT, SQRT is disabled in the user-defined subroutine. This restriction also applies to intrinsic functions referenced implicitly when expressions that include exponentiation are used.
- Restrictions on intrinsic subroutines.
  If an error occurs, the intrinsic subroutine that caused the error cannot be used in the user-defined subroutine.
- Restrictions on return codes.
  If arguments data1...datan of the user-defined error subroutine are not changed, the user-defined subroutine must set the return code to 0. If the passed arguments are changed, the subroutine must set the return code to 1. Only 0 or 1 may be used as return code values. For all other values, the results are unpredictable.
- Restrictions on the argument type.
  For error subroutines written in Fortran, the argument type must match that of the CALL statement.

8.1.3.2 Handling of Incorrect Arguments in Intrinsic Functions

If an intrinsic function argument is incorrect, the argument is assigned to a local variable in the intrinsic function and the error monitor is called. The local variable is passed to the error monitor. In user-defined error subroutines, this local variable must be corrected. Do not correct the argument in the program unit that used the incorrect argument.

8.1.3.3 User Error Processing

The following example shows user error processing. In this example, when the user enters incorrect data (1234567890123), error number 171 is generated. To continue processing, the user corrects the data to 9999 with the user-defined error subroutine FIXUP.

Example: User error processing

Fortran program:

```
      EXTERNAL FIXUP
      CALL ERRSET(171,0,0,0,FIXUP,0)
      READ(5,*)I
      WRITE(6,100)I
100   FORMAT(1H, 5X, 'I=',I4)
      STOP
      END
```

User-defined error correction subroutine:

```
SUBROUTINE FIXUP(IRET,INO,I4CONV)
IF(INO==171)THEN
   I4CONV=9999
   IRET=1
END IF
RETURN
END
```

Input data:

```
1234567890123
```

Result:

```
jwe0171i-e line 3 Integer(kind=1), integer(kind=2) or integer(kind=4) number (1234567890123)is out
of range (unit= 5).
 error occurs at MAIN__   line 3 loc 0000000000400bb5 offset 0000000000000055
 MAIN__       at loc 0000000000400b60 called from o.s.
taken to (user) corrective action, execution continuing.
      I=9999
```

8.1.4 Input/Output Error Processing

If an IOSTAT= specifier, ERR= specifier, END= specifier, or EOR= specifier has been specified, input/output errors are processed based on the specifier without regard to the error count limit. See Section "7.7.1.3 IOSTAT= Specifier", Section "7.7.1.4 Error Branch", Section "7.7.1.5 End-of-File Branch", and Section "7.7.1.6 End-of-Record Branch" for details.

If an error occurs, the file position and input list values being executed at the time of the error are undefined.

The table below lists the standard system and user-defined corrections for errors that occur during input/output statement execution. The following arguments are passed to the user-defined error program:

- Return code
- Error number
- The data listed in the table below

Table 8.2 Error processing for input/output

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
21 to 22	Terminates the program	Not correctable	UNIT	None
28	Terminates the program	Not correctable	UNIT	None
42	Ignores the statement	Not correctable	UNIT	None
55	Ignores the statement	Not correctable	UNIT	None
62	Ignores the statement	Not correctable	UNIT	None
64 to 65	Ignores the statement	Not correctable	UNIT	None
71 to 72	Ignores the statement	Not correctable	UNIT	None
74	Ignores the statement	Not correctable	UNIT	None
80	Ignores the statement	Not correctable	UNIT	REC
81	Ignores the statement	Not correctable	None	None
82	Ignores the statement	Not correctable	UNIT	None
83 to 85	Continues processing without setting a specifier value.	Not correctable	None	None
86	Continues processing without setting a specifier value.	Not correctable	UNIT	None

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
87 to 89	Ignores the statement	Not correctable	UNIT	None
97	Uses DIRECT as the ACCESS=specifier	Not correctable	UNIT	None
102 to 103	Ignores the statement	Not correctable	UNIT	None
104	Ignores a POSITION=specifier and continues processing	Not correctable	UNIT	None
105	Ignores the statement	Not correctable	UNIT	None
109	The maximum value of the type specified by specifier value is set, and processing is continued.	Not correctable	UNIT	None
111 to 115	Ignores the statement	Not correctable	UNIT	None
120	Ignores the statement	Not correctable	UNIT	None
122	Ignores the statement	Not correctable	UNIT	None
131	Ignores subsequent input/output lists	Not correctable	None	None
132	a. For input, ignores data that exceeded the record b. For output, writes the excess data to the next logical record	Not correctable	UNIT	None
133	Ignores subsequent output list items	Not correctable	UNIT	None
134	Ignores subsequent input/output lists	Not correctable	UNIT	None
140	Ignores the conversion and continues processing	Not correctable	UNIT	None
141 to 142	Assigns maximum value after transforming data	Assign the transformed data to FDATA	FDATA	UDATA
143 to 144	Assigns the value 0 after transforming data	Assign the transformed data to FDATA	FDATA	UDATA
145 to 146	Assigns maximum value (Inf or NaN) after transforming data	Assign the transformed data to FDATA	FDATA	UDATA
147	The last bit of mantissa assigns the value	Not correctable	UNIT	None
151	Ignores subsequent input/output lists	Not correctable	UNIT	None
152	Edits according to the edit descriptor	Not correctable	UNIT	None
153	Adds the initial left parentheses	Not correctable	UNIT	None
154	Ignores nests exceeding 30 levels in the format specification	Not correctable	UNIT	None
155	Ignores the edit descriptor and goes by default to the final right parenthesis of the format specification	Not correctable	UNIT	None
156	Ignores the repeat specification and goes by default to the final right parenthesis of the format specification	Not correctable	UNIT	None
157	Ignores the incorrect character and goes by default to the final right parenthesis of the format specification	Assign the character of the format specification to FCODE	FCODE	None

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
158	Adds the right parenthesis and continues format control	Not correctable	UNIT	None
159	Replaces the input data with the character string in the format specification	Not correctable	UNIT	None
162	Ignores the input/output list and continues a format control	Not correctable	UNIT	None
171	a. Uses by default the minimum specifiable value if the one-, two- or four-byte input item is negative b. Uses by default the maximum specifiable value if the one-, two- or four-byte input item is positive	Correct I4CNV	I4CNV	None
172	a. Uses by default the minimum specifiable value if the eight-byte input item is negative b. Uses by default the maximum specifiable value if the eight-byte input item is positive	Correct I8CNV	I8CNV	None
173 to 175	Uses 0 for the incorrect character	Assign the corrected character to IDATA	IDATA	None
176	a. Sets true 0 if the absolute value is less than 2**(-126)*2**(-23) (real), 2**(-1022)*2**(-52) (double-precision real) or 2**(-16382)*2**(-112) (quadruple-precision real) b. Sets 2**127*(1+1.0-2**(-23))(real), 2**1023*(1+1.0-2**(-52)) (double-precision real) or 2**16383*(1+1.0- 2**(-112)) (quadruple-precision real) without changing the sign if the absolute value is greater than 2**127*(1+1.0-2**(-23)) real), 2**1023*(1+1.0-2**(-52)) (double-precision real) or 2**16383*(1+1.0-2**(-112)) (quadruple-precision real)	Correct FCONV	FCONV	None
182 to 183	Executes character editing	Not correctable	UNIT	None
184	Ignores the constant and terminates the statement	Not correctable	UNIT	None
185	Ignores the repeat count or constant and terminates the statement	Not correctable	UNIT	None
186 to 188	Ignores the item name and terminates the statement	Not correctable	UNIT	None
189	Ignores the statement	Not correctable	UNIT	None
190	Ignores the constant that exceeded the element and continues the processing	Not correctable	UNIT	None
1073	Ignores the statement	Not correctable	UNIT	None
1111 to 1113	Ignores the statement	Not correctable	UNIT	None

Error number	Standard correction processing	User-defined error subroutine	Data passed to user-defined subroutine	
			Data1	Data2
1114	Assigns the maximum value for 4 byte integer to the specifier, and continues the processing	Not correctable	UNIT	None
1115	An invalid specifier in an open statement, and continues the processing	Not correctable	UNIT	None
1117 to 1122	Ignores the statement	Not correctable	UNIT	None
1146	Ignores the statement	Not correctable	UNIT	None
1147	Ignores the statement	Not correctable	UNIT	None
1148	Ignores the statement	Not correctable	UNIT	None
1149	Uses NO as the ASYNCHRONOUS= specifier	Not correctable	UNIT	None
1150	Ignores the statement	Not correctable	UNIT	None
1161	Terminates the program	Not correctable	UNIT	None
1162	The multiple records are written or read, and continues the processing	Not correctable	UNIT	None
1163	Assigns the maximum value for 4 byte integer to the variable, and continues the processing	Not correctable	None	None
1181	Ignores the input data and continues the program	Not correctable	UNIT	None
1201	Uses 0 for the incorrect character	Assign the corrected character to IDATA	IDATA	None
1202	Ignores the input data and continues the processing	Not correctable	UNIT	None
1203	Edits with the type of data and continues the processing	Not correctable	UNIT	None

UNIT: For an external file, the unit number (four-byte integer) is specified. For an internal file, -1 is specified.

IDATA: Incorrect character in the input data (one-byte character)

FCODE: Incorrect format code (one-byte character) in the format assigned by the array

REC: Record variable (four-byte integer)

I4CNV: Integer indicating result of format conversion (four-byte integer)

I8CNV: Integer indicating result of format conversion (eight-byte integer)

FCONV: Integer indicating result of format conversion (real)

FDATA: Integer indicating result of IEEE-IBM370 conversion (real)

UDATA: For an input/output statement, the same as UNIT; for a service subroutine, the maximum value of a four-byte integer

8.1.5 Intrinsic Function Error Processing

The table below lists the causes of intrinsic function errors and the standard system correction processing in execution.

When -Ncheck_intrfunc option is specified, the error numbers of 202-255, 259-282, 1355-1370, 1396-1414, 1416-1417, and 1425-1454 are detected and the intrinsic function error processing is performed.

When -Ncheck_intrfunc option is not specified, the floating point exception occurs instead of the error. The floating point exception is detected that -NRtrap option and environment variable FLIB_EXCEPT=u are specified.

See Section "2.2 Compiler Options" for details of -Ncheck_intrfunc option. See Section "3.8 Using Environment Variables for Execution" for detail of environment variable FLIB_EXCEPT=u.

The following arguments are passed to the user-defined error subroutine:

- Return code
- Error number
- The data listed in the table below

Table 8.3 Error processing for intrinsic function

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
202	SIN(X)	|X|>=8.23e+05	NaN	X	None
	COS(X)				
203	DSIN(DX)	|DX|>=3.53d+15	NaN	DX	None
	DCOS(DX)				
204	QSIN(QX)	|QX|>=2**62*PI	NaN	QX	None
	QCOS(QX)				
205	CSIN(CX)	|X1|>=8.23e+05	(NaN,NaN)	X1	None
	CCOS(CX)				
206	CSIN(CX)	X2>=89.415	(SIGN(Inf,SIN(X1)), SIGN(Inf,COS(X1)))	X2	None
		X2<=-89.415	(SIGN(Inf,SIN(X1)), -SIGN(Inf,COS(X1)))		
	CCOS(CX)	X2>=89.415	(SIGN(Inf,COS(X1)), -SIGN(Inf,SIN(X1)))		
		X2<=-89.415	(SIGN(Inf,COS(X1)), SIGN(Inf,SIN(X1)))		
207	CDSIN(CDX)	|DX1|>=3.53d+15	(NaN,NaN)	DX1	None
	CDCOS(CDX)				
208	CDSIN(CDX)	DX2>=710.475	(DSIGN(Inf,DSIN(DX1)), DSIGN(Inf,DCOS(DX1)))	DX2	None
		DX2<=-710.475	(DSIGN(Inf,DSIN(DX1)), -DSIGN(Inf,DCOS(DX1)))		
	CDCOS(CDX)	DX2>=710.475	(DSIGN(Inf,DCOS(DX1)), -DSIGN(Inf,DSIN(DX1)))		
		DX2<=-710.475	(DSIGN(Inf,DCOS(DX1)), DSIGN(Inf,DSIN(DX1)))		
209	CQSIN(CQX)	|QX1|>=2**62*PI	(NaN,NaN)	QX1	None
	CQCOS(CQX)				
210	CQSIN(CQX)	QX2>=11357.125	(QSIGN(Inf,QSIN(QX1)), QSIGN(Inf,QCOS(QX1)))	QX2	None
		QX2<=-11357.125	(QSIGN(Inf,QSIN(QX1)), -QSIGN(Inf,QCOS(QX1)))		
	CQCOS(CQX)	QX2>=11357.125	(QSIGN(Inf,QCOS(QX1)), -QSIGN(Inf,QSIN(QX1)))		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
		QX2<=-11357.125	(QSIGN(Inf,QCOS(QX1)), QSIGN(Inf,QSIN(QX1)))		
211	TAN(X)	|X|>=8.23e+05	NaN	X	None
	COTAN(X)				
212	TAN(X)	|X| is close to singularity (+-PI/2,+-3PI/2,...)	Inf	X	None
	COTAN(X)	|X| is close to singularity (0,+-PI,+-2PI,...)			
213	DTAN(DX)	|DX|>=3.53d+15	NaN	DX	None
	DCOTAN(DX)				
214	DTAN(DX)	|DX| is close to singularity (+-PI/2,+-3PI/2,...)	Inf	DX	None
	DCOTAN(DX)	|DX| is close to singularity (0,+-PI,+-2PI,...)			
215	QTAN(QX)	|QX|>=2**62*PI	NaN	QX	None
	QCOTAN(QX)				
216	QTAN(QX)	|QX| is close to singularity (+-PI/2,+-3PI/2,...)	Inf	QX	None
	QCOTAN(QX)	|QX| is close to singularity (0,+-PI,+-2PI,...)	Inf	QX	None
217	ASIN(X)	|X|>1.0	NaN	X	None
	ACOS(X)				
218	DASIN(DX)	|DX|>1.0	NaN	DX	None
	DACOS(DX)				
219	QASIN(QX)	|QX|>1.0	NaN	QX	None
	QACOS(QX)				
220	ATAN(X1,X2)	X1=0.0 and X2=0.0	NaN	X1	X2
	ATAN2(X1,X2)				
221	DATAN2(DX1,DX2)	DX1=0.0 and DX2=0.0	NaN	DX1	DX2
222	QATAN2(QX1,QX2)	QX1=0.0 and QX2=0.0	NaN	QX1	QX2
223	SINH(X)	X>=89.415	Inf	X	None
		X<=-89.415	-Inf		
	COSH(X)	|X|>=89.415	Inf		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
224	DSINH(DX)	DX>=710.475	Inf	DX	None
		DX<=-710.475	-Inf		
	DCOSH(DX)	|DX|>=710.475	Inf		
225	QSINH(QX)	QX>=11357.125	Inf	QX	None
		QX<=-11357.125	-Inf		
	QCOSH(QX)	|QX|>=11357.125	Inf		
226	SQRT(X)	X<0.0	NaN	X	None
227	DSQRT(DX)	DX<0.0	NaN	DX	None
228	QSQRT(QX)	QX<0.0	NaN	QX	None
229	EXP(X)	X>=88.722	Inf	X	None
230	DEXP(DX)	DX>=709.782	Inf	DX	None
231	QEXP(QX)	QX>=11356.5	Inf	QX	None
232	CEXP(CX)	X1>=88.722	(SIGN(Inf,COS(X1)), SIGN(Inf,SIN(X1)))	X1	None
233	CEXP(CX)	|X2|>=8.23e+05	(NaN,NaN)	X2	None
234	CDEXP(CDX)	DX1>=709.782	(DSIGN(Inf,DCOS(DX1)), DSIGN(Inf,DSIN(DX1)))	DX1	None
235	CDEXP(CDX)	|DX2|>=3.53d+15	(NaN,NaN)	DX2	None
236	CQEXP(CQX)	QX1>=11356.5	(QSIGN(Inf,QCOS(QX1)), QSIGN(Inf,QSIN(QX1)))	QX1	None
237	CQEXP(CQX)	|QX2|>=2**62*PI	(NaN,NaN)	QX2	None
238	EXP2(X)	X>=128.0	Inf	X	None
	2.0**X				
239	DEXP2(DX)	DX>=1024.0	Inf	DX	None
	2.0**DX				
240	QEXP2(QX)	QX>=16384.0	Inf	QX	None
	2.0**QX				
241	EXP10(X)	X>=38.531	Inf	X	None
	10.0**X				
242	DEXP10(DX)	DX>=308.254	Inf	DX	None
	10.0**DX				
243	QEXP10(QX)	QX>=4932.0625	Inf	QX	None
	10.0**QX				
244	ALOG(X)	X=0.0	-Inf	X	None
		X<0.0	NaN		
	ALOG10(X)	X=0.0	-Inf		
		X<0.0	NaN		
	ALOG2(X)	X=0.0	-Inf		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
		X<0.0	NaN		
245	DLOG(DX)	DX=0.0	-Inf	DX	None
		DX<0.0	NaN		
	DLOG10(DX)	DX=0.0	-Inf		
		DX<0.0	NaN		
	DLOG2(DX)	DX=0.0	-Inf		
		DX<0.0	NaN		
246	QLOG(QX)	QX=0.0	-Inf	QX	None
		QX<0.0	NaN		
	QLOG10(QX)	QX=0.0	-Inf		
		QX<0.0	NaN		
	QLOG2(QX)	QX=0.0	-Inf		
		QX<0.0	NaN		
247	CLOG(CX)	CX=(0.0,0.0)	(Inf,NaN)	CX	None
248	CDLOG(CDX)	CDX=(0.0,0.0)	(Inf,NaN)	CDX	None
249	CQLOG(CQX)	CQX=(0.0,0.0)	(Inf,NaN)	CQX	None
250	GAMMA(X)	X<=0.0	NaN	X	None
		X>=35.039860	Inf		
251	DGAMMA(DX)	DX<=0.0	NaN	DX	None
		DX>=171.6243	Inf		
252	QGAMMA(QX)	QX<=0.0	NaN	QX	None
		QX>=1.755q+03	Inf		
253	ALGAMA(X)	X<=0.0	NaN	X	None
		X>=0.403711e+37	Inf		
254	DLGAMA(DX)	DX<=0.0	NaN	DX	None
		DX>=2.55634d+305	Inf		
255	QLGAMA(QX)	QX<=0.0	NaN	QX	None
		QX>=1.048q+4928	Inf		
256	LGT(C1,C2)	An EBCDIC code, not corresponding to an ASCII code was detected in a character string C1 or C2	The default ASCII collation order of incorrect argument, 26(X'1A'), is used.	C3	None
	LGE(C1,C2)				
	LLT(C1,C2)				
	LLE(C1,C2)				
257	IBSET(I,POS)	The value of POS exceeded the range	I	None	
	IBCLR(I,POS)				
	BTEST(I,POS)		.FALSE.		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
259	IX1**IX2	IX1=0 and IX2<0	0	IX1	IX2
260	JX1**JX2	JX1=0 and JX2<0	0	JX1	JX2
261	X**IX	X=0.0 and IX<0	Inf	X	IX
262	X1**X2	X1=0.0 and X2<0.0	Inf	X1	X2
263	X1**X2	X1<0.0 and X2/=0.0	NaN	X1	X2
264	DX**IX	DX=0.0 and IX<0	Inf	DX	IX
265	DX**JX	DX=0.0 and JX<0	Inf	DX	JX
266	DX1**DX2	DX1=0.0 and DX2<0.0	Inf	DX1	DX2
267	DX1**DX2	DX1<0.0 and DX2/=0.0	NaN	DX1	DX2
268	QX**IX	QX=0.0 and IX<0	Inf	QX	IX
269	QX**JX	QX=0.0 and JX<0	Inf	QX	JX
270	QX1**QX2	QX1=0.0 and QX2<0.0	Inf	QX1	QX2
271	QX1**QX2	QX1<0.0 and QX2/=0.0	NaN	QX1	QX2
272	QX1**QX2	|QX1**QX2| >=2**16384	Inf	QX1	QX2
273	CX**IX	CX=(0.0,0.0) and IX<0	(NaN,NaN)	CX	IX
274	CDX**IX	CDX=(0.0,0.0) and IX<0	(NaN,NaN)	CDX	IX
275	CDX**JX	CDX=(0.0,0.0) and JX<0	(NaN,NaN)	CDX	JX
276	CQX**IX	CQX=(0.0,0.0) and IX<0	(NaN,NaN)	CQX	IX
277	CQX**JX	CQX=(0.0,0.0) and JX<0	(NaN,NaN)	CQX	JX
278	QX1/QX2	|QX1/QX2| >=2**16384	Inf	QX1	QX2
279	QX1/QX2	|QX1/QX2| <2**(-16382)	0.0	QX1	QX2
280	QX1/QX2	QX2=0.0	For QX1=0.0 NaN For QX1/=0.0 SIGN(QX1)*Inf	QX1	QX2
281	JX1+JX2	|JX1+JX2| >2**63-1	2**63-1	JX1	JX2

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	JX1-JX2	|JX1-JX2| >2**63-1			
	JX1*JX2	|JX1*JX2| >2**63-1			
	JX1**JX2	|JX1**JX2| >2**63-1			
282	JX1/JX2	JX2=0	For JX1=0 0 For JX1/=0 2**63-1	QX1	QX2
1355	TANQ(X)	|X| is close to singularity (+-1,+-3, ...)	Inf	X	None
	COTANQ(X)	|X| is close to singularity (0,+-2,+-4, ...)			
1356	DTANQ(DX)	|DX| is close to singularity (+-1,+-3, ...)	Inf	DX	None
	DCOTANQ(DX)	|DX| is close to singularity (0,+-2,+-4, ...)			
1357	ASINQ(X)	|X|>1.0	NaN	X	None
	ACOSQ(X)				
1358	DASINQ(DX)	|DX|>1.0	NaN	DX	None
	DACOSQ(DX)				
1359	ATAN2Q(X1,X2)	X1=0.0 and X2=0.0	NaN	X1	X2
1360	DATAN2Q(DX1,DX2)	DX1=0.0 and DX2=0.0	NaN	DX1	DX2
1361	SIND(X)	|X|>=4.72e+07	NaN	X	None
	COSD(X)				
1362	DSIND(DX)	|DX|>=2.03d+17	NaN	DX	None
	DCOSD(DX)				
1363	TAND(X)	|X|>=4.72e+07	NaN	X	None
	COTAND(X)				
1364	DTAND(DX)	|DX|>=2.03d+17	NaN	DX	None
	DCOTAND(DX)				
1365	TAND(X)	|X| is close to singularity (+-90,+-270, ...)	Inf	X	None

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	COTAND(X)	|X| is close to singularity (0,+-180,+-360, ...)			
1366	DTAND(DX)	|DX| is close to singularity (+-90,+-270, ...)	Inf	DX	None
	DCOTAND(DX)	|DX| is close to singularity (0,+-180,+-360, ...)			
1367	ASIND(X)	|X|>1.0	NaN	X	None
	ACOSD(X)				
1368	DASIND(DX)	|DX|>1.0	NaN	DX	None
	DACOSD(DX)				
1369	ATAN2D(X1,X2)	X1=0.0 and X2=0.0	NaN	X1	X2
1370	DATAN2D(DX1,DX2)	DX1=0.0 and DX2=0.0	NaN	DX1	DX2
1371	ISHFTC(I,SHIFT[,SIZE])	SIZE<=0 or SIZE>BIT_SIZE(I)	BIT_SIZE(I) is set to SIZE, and process is continued	None	
		|SHIFT|>SIZE	Value of SIZE is set to SHIFT, and process is continued		
1372	IBITS(I,POS,LEN)	LEN<0 or LEN>BIT_SIZE(I)-POS	0	None	
		POS<0 or POS>=BIT_SIZE(I)-LEN			
1373	ISHFT(I,SHIFT)	|SHIFT| >BIT_SIZE(I)	BIT_SIZE(I) is set to SHIFT, and process is continued	None	
1375	MATMUL(MATRIX_A,MATRIX_B)	The size of last dimension of MATRIX_A and the size of first dimension of MATRIX_B is not same	Terminates the program	None	
1376	SPREAD(SOURCE,DIM,NCOPIES)	DIM<=0 or DIM>(The rank of SOURCE)+1	Terminates the program	None	
1377	ALL(MASK[,DIM])	DIM<=0 or DIM>(The rank of MASK)	Terminates the program	None	
	ANY(MASK[,DIM])				
	COUNT(MASK[,DIM,KIND])				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	SUM(ARRAY,DIM[,MASK])	DIM<=0 or DIM>(The rank of ARRAY)			
	PRODUCT(ARRAY,DIM[,MASK])				
	MAXVAL(ARRAY,DIM[,MASK])				
	MINVAL(ARRAY,DIM[,MASK])				
	UBOUND(ARRAY[,DIM])				
	LBOUND(ARRAY[,DIM,KIND])				
	SIZE(ARRAY[,DIM])				
	MAXLOC(ARRAY,DIM[,MASK,KIND])				
	MINLOC(ARRAY,DIM[,MASK,KIND])				
1378	MAXLOC(ARRAY,DIM[,MASK,KIND]) or MAXLOC(ARRAY[,MASK,KIND])	The shape of ARRAY and MASK is different	Terminates the program	None	
	MINLOC(ARRAY,DIM[,MASK,KIND]) or MINLOC(ARRAY[,MASK,KIND])				
	SUM(ARRAY,DIM[,MASK]) or SUM(ARRAY[,MASK])				
	PRODUCT(ARRAY,DIM[,MASK]) or PRODUCT(ARRAY[,MASK])				
	MAXVAL(ARRAY,DIM[,MASK]) or MAXVAL(ARRAY[,MASK])				
	MINVAL(ARRAY,DIM[,MASK]) or MINVAL(ARRAY[,MASK])				
	PACK(ARRAY,MASK[,VECTOR])				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1381	PACK(ARRAY,MASK[,VECTOR])	The elements number of VECTOR is less than total number of TRUE of MASK	Terminates the program	None	
1382	UNPACK(VECTOR,MASK[,FIELD])	The elements number of VECTOR is less than total number of TRUE of MASK	Terminates the program	None	
1383	UNPACK(VECTOR,MASK[,FIELD])	The shape of MASK and FIELD is not same	Terminates the program	None	
1384	CSHIFT(ARRAY,SHIFT[,DIM]) EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	DIM<=0 or DIM>(The rank of ARRAY)	Terminates the program	None	
1385	CSHIFT(ARRAY,SHIFT[,DIM]) EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	The shape of SHIFT is invalid	Terminates the program	None	
1386	EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	The shape of BOUNDARY is invalid	Terminates the program	None	
1387	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The size of SOURCE is less than the product of every element's values of SHAPE	Terminates the program	None	
1390	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The shape of SHAPE and ORDER is different	Terminates the program	None	
1391	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The value of the elements ORDER must be the combination of the numbers to the size of SHAPE	Terminates the program	None	
1392	REPEAT(STRING,NCOPIES)	NCOPIES<0	Terminates the program	None	
1393	NEAREST(X,S)	S=0.0	X	None	
1394	SIZE(ARRAY[,DIM]) UBOUND(ARRAY[,DIM])	If ARRAY is the assumed-size array DIM<1 or DIM>=(The rank of ARRAY)	Terminates the program	None	
1395	RESHAPE(SOURCE,SHAPE[,PAD][,ORDER])	The type parameter of SOURCE and PAD is different	Terminates the program	None	

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	PACK(ARRAY,MASK[,VECTOR])	The type parameter of ARRAY and VECTOR is different			
	UNPACK(VECTOR,MASK[,FIELD])	The type parameter of VECTOR and FIELD is different			
	MERGE(TSOURCE,FSOURCE,MASK)	The type parameter of TSOURCE and FSOURCE is different			
	EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	The type parameter of ARRAY and BOUNDARY is different			
1396	MODULO(A,P)	P=0	For REAL type A Inf For INTEGER type A HUGE(A)	None	
1397	QX1+QX2	|QX1+QX2| >=2**16384(over flow)	Inf	QX1	QX2
	QX1-QX2	|QX1-QX2| >=2**16384(over flow)			
	QX1*QX2	|QX1*QX2| >=2**16384(over flow)			
1398	QX1*QX2	|QX1*QX2| <2**(-16382) (under flow)	0.0	QX1	QX2
1399	QTANQ(QX)	|QX| is close to singularity (+-1,+-3, ...)	Inf	QX	None
	QCOTANQ(QX)	|QX| is close to singularity (0,+-2,+-4, ...)			
1400	QASINQ(QX)	|QX|>1.0	NaN	QX	None
	QACOSQ(QX)				
1401	QATAN2Q(QX1,QX2)	QX1=0.0 and QX2=0.0	NaN	QX1	QX2
1402	QSIND(QX)	|QX|>=2**62*180	NaN	QX	None
	QCOSD(QX)				
1403	QTAND(QX)	|QX|>=2**63*90	NaN	QX	None
	QCOTAND(QX)				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1404	QTAND(QX)	|QX| is close to singularity (+-90,+-270, ...)	Inf	QX	None
	QCOTAND(QX)	|QX| is close to singularity (0, +-180,+-360, ...)			
1405	QASIND(QX)	|QX|>1.0	NaN	QX	None
	QACOSD(QX)				
1406	QATAN2D(QX1,QX2)	QX1=0.0 and QX2=0.0	NaN	QX1	QX2
1407	CSINQ(CX)	|X1|>=5.24e+05	(NaN,NaN)	X1	None
	CCOSQ(CX)				
1408	CSINQ(CX)	X2>=56.92	(SIGN(Inf,SINQ(X1)), SIGN(Inf,COSQ(X1)))	X2	None
		X2<=-56.92	(SIGN(Inf,SINQ(X1)), -SIGN(Inf,COSQ(X1)))		
	CCOSQ(CX)	X2>=56.92	(SIGN(Inf,COSQ(X1)), -SIGN(Inf,SINQ(X1)))		
		X2<=-56.92	(SIGN(Inf,COSQ(X1)), SIGN(Inf,SINQ(X1)))		
1409	CDSINQ(CDX)	|DX1|>=2.25d+15	(NaN,NaN)	DX1	None
	CDCOSQ(CDX)				
1410	CDSINQ(CDX)	DX2>=452.30	(DSIGN(Inf,DSINQ(DX1)), DSIGN(Inf,DCOSQ(DX1)))	DX2	None
		DX2<=-452.30	(DSIGN(Inf,DSINQ(DX1)), -DSIGN(Inf,DCOSQ(DX1)))		
	CDCOSQ(CDX)	DX2>=452.30	(DSIGN(Inf,DCOSQ(DX1)) ,-DSIGN(Inf,DSINQ(DX1)))		
		DX2<=-452.30	(DSIGN(Inf,DCOSQ(DX1)) ,DSIGN(Inf,DSINQ(DX1)))		
1411	CQSINQ(CQX)	|QX1|>=2**63	(NaN,NaN)	QX1	None
	CQCOSQ(CQX)				
1412	CQSINQ(CQX)	QX2>=7230.125	(QSIGN(Inf,QSINQ(QX1)), QSIGN(Inf,QCOSQ(QX1)))	QX2	None
		QX2<=-7230.125	(QSIGN(Inf,QSINQ(QX1)), -QSIGN(Inf,QCOSQ(QX1)))		
	CQCOSQ(CQX)	QX2>=7230.125	(QSIGN(Inf,QCOSQ(QX1)) ,-QSIGN(Inf,QSINQ(QX1)))		

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
		QX2<=-7230.125	(QSIGN(Inf,QCOSQ(QX1)) ,QSIGN(Inf,QSINQ(QX1)))		
1413	X**JX	X=0.0 and JX<0	Inf	X	JX
1414	CX**JX	CX=(0.0,0.0) and JX<0	(NaN,NaN)	CX	JX
1415	SELECTED_REAL_KIND(P,R)	P and R are not specified	Terminates the program	None	
1416	X1**X2	|X1**X2| >=2**128	Inf	X1	X2
1417	DX1**DX2	|DX1**DX2| >=2**1024	Inf	DX1	DX2
1418	PACK(ARRAY,MASK[,VECTOR])	The elements number of VECTOR must be greater than or equal to the total number of ARRAY if MASK is scalar with the value TRUE	Terminates the program	None	
1419	EOSHIFT(ARRAY,SHIFT[,BOUNDARY][,DIM])	Do not omit BOUNDARY, when ARRAY is a derived type.	Terminates the program	None	
1420	ICHAR(C [,KIND])	The function result exceeds the expressible range that kind type parameter is specified by KIND	The kind type parameter of result uses the specified kind value	None	
	INDEX(STRING,SUBSTRING[,BACK, KIND])				
	LBOUND(ARRAY,[, DIM, KIND])				
	LEN(STRING[, KIND])				
	SCAN(STRING, SET[, BACK, KIND])				
	SHAPE(SOURCE[, KIND])				
	SIZE(ARRAY[,DIM, KIND])				
	UBOUND(ARRAY[, DIM, KIND])				
	VERIFY(STRING, STE[, BACK, KIND])				
1421	IEEE_CLASS(X)	The function used is not supported.	0 or .FALSE.	None	
	IEEE_COPY_SIGN(X,Y)				
	IEEE_IS_FINITE(X)				
	IEEE_IS_NORMAL(X)				
	IEEE_IS_NEGATIVE(X)				

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
	IEEE_LOGB(X)				
	IEEE_NEXT_AFTER(X,Y)				
	IEEE_REM(X,Y)				
	IEEE_RINT(X)				
	IEEE_SCALB(X, I)				
	IEEE_UNORDERD(X,Y)				
	IEEE_VALUE(X,CLASS)				
1422	IEEE_IS_NAN(X)	The function used is not supported.	0 or .FALSE.	None	
	IEEE_VALUE(X,CLASS)				
1425	ACOSH(X)	X<1	NaN	X	None
1426	ACOSH(DX)	DX<1	NaN	DX	None
1427	ACOSH(QX)	QX<1	NaN	QX	None
1428	ATANH(X)	|X|>1	NaN	X	None
		X=1	Inf		
		X=-1	-Inf		
1429	ATANH(DX)	|DX|>1	NaN	DX	None
		DX=1	Inf		
		DX=-1	-Inf		
1430	ATANH(QX)	|QX|>1	NaN	QX	None
		QX=1	Inf		
		QX=-1	-Inf		
1431	TAN(CX)	|X1|>=8.23E+05	(NaN,NaN)	X1	None
1432	TAN(CX)	CX is close to singularity ((+-PI/2,0),(+-3PI/2,0), ...)	For REAL(CX)>=0.0 (Inf,0.0) For REAL(CX)<0.0 (-Inf,0.0)	CX	None
1433	TAN(CDX)	|DX1|>=3.53D+15	(NaN,NaN)	DX1	None
1434	TAN(CDX)	CDX is close to singularity ((+-PI/2,0),(+-3PI/2,0), ...)	For DREAL(CDX)>=0.0 (Inf,0.0) For DREAL(CDX)<0.0 (-Inf,0.0)	CDX	None
1435	TAN(CQX)	|QX1| >=2.0**62*PI	(NaN,NaN)	QX1	None
1436	TAN(CQX)	CQX is close to singularity ((+-PI/2,0),(+-3PI/2,0), ...)	For QREAL(CQX)>=0.0 (Inf,0.0) For QREAL(CQX)<0.0 (-Inf,0.0)	CQX	None
1437	ATAN(CX)	CX=(0.0,+-1.0)	(0.0,+-Inf)	CX	None
1438	ATAN(CDX)	CDX=(0.0,+-1.0)	(0.0,+-Inf)	CDX	None

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1439	ATAN(CQX)	CQX=(0.0,+-1.0)	(0.0,+-Inf)	CQX	None
1440	SINH(CX)	|X2|>=8.23E+05	(NaN,NaN)	X2	None
	COSH(CX)				
1441	SINH(CX)	X1>=89.4150	(SIGN(Inf,COS(X2)), SIGN(Inf,SIN(X2)))	X1	None
		X1<=-89.4150	(-SIGN(Inf,COS(X2)), SIGN(Inf,SIN(X2)))		
	COSH(CX)	X1>=89.4150	(SIGN(Inf,COS(X2)), SIGN(Inf,SIN(X2)))		
		X1<=-89.4150	(SIGN(Inf,COS(X2)), -SIGN(Inf,SIN(X2)))		
1442	SINH(CDX)	|DX2|>=3.53D+15	(NaN,NaN)	DX2	None
	COSH(CDX)				
1443	SINH(CDX)	DX1>=710.475	(DSIGN(Inf,DCOS(DX2)), DSIGN(Inf,DSIN(DX2)))	DX1	None
		DX1<=-710.475	(-DSIGN(Inf,DCOS(DX2)), DSIGN(Inf,DSIN(DX2)))		
	COSH(CDX)	DX1>=710.475	(DSIGN(Inf,DCOS(DX2)), DSIGN(Inf,DSIN(DX2)))		
		DX1<=-710.475	(DSIGN(Inf,DCOS(DX2)), -DSIGN(Inf,DSIN(DX2)))		
1444	SINH(CQX)	|QX2| >=2.0**62*PI	(NaN,NaN)	QX2	None
	COSH(CQX)				
1445	SINH(CQX)	QX1>=11357.125	(QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))	QX1	None
		QX1<=-11357.125	(-QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))		
	COSH(CQX)	QX1>=11357.125	(QSIGN(Inf,QCOS(QX2)), QSIGN(Inf,QSIN(QX2)))		
		QX1<=-11357.125	(QSIGN(Inf,QCOS(QX2)), -QSIGN(Inf,QSIN(QX2)))		
1446	TANH(CX)	|X2|>=8.23E+05	(NaN,NaN)	X2	None
1447	TANH(CX)	CX is close to singularity ((0,+-PI/2),(0,+-3PI/2), ...)	(NaN,+-Inf)	CX	None
1448	TANH(CDX)	|DX2|>=3.53D+15	(NaN,NaN)	DX2	None
1449	TANH(CDX)	CDX is close to singularity ((0,+-PI/2),(0,+-3PI/2), ...)	(NaN,+-Inf)	CDX	None
1450	TANH(CQX)	|QX2| >=2.0**62*PI	(NaN,NaN)	QX2	None

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine	
				data1	data2
1451	TANH(CQX)	CQX is close to singularity ((0,+-PI/2),(0,+-3PI/2), ...)	(NaN,+-Inf)	CQX	None
1452	ATANH(CX)	CX=(+-1.0,0.0)	(+-Inf,0.0)	CX	None
1453	ATANH(CDX)	CDX=(+-1.0,0.0)	(+-Inf,0.0)	CDX	None
1454	ATANH(CQX)	CQX=(+-1.0,0.0)	(+-Inf,0.0)	CQX	None
1455	DSHIFTL(I, J, SHIFT)	SHIFT<0 or SHIFT>BIT_SIZE(I)	0 is set to SHIFT in SHIFT<0 or BIT_SIZE(I) is set to SHIFT in SHIFT>BIT_SIZE(I), and process is continued	None	
	DSHIFTR(I, J, SHIFT)				
	SHIFTA(I, SHIFT)				
	SHIFTL (I, SHIFT)				
	SHIFTR (I, SHIFT)				
	MASKL(I [, KIND])	I<0 or I>BIT_SIZE(I)	0 is set to I in I<0 or BIT_SIZE(I) is set to I in I>BIT_SIZE(I), and process is continued		
	MASKR(I [, KIND])				

- C1, C2, and C3 are one-byte character data. The value 26(X'1A') is assigned to C3.
- IX, IX1, and IX2 are four-byte integer data.
- JX, JX1, and JX2 are eight-byte integer data.
- X, X1, and X2 are real data.
- DX, DX1, and DX2 are double-precision real data.
- QX, QX1, and QX2 are quadruple-precision real data.
- CX is complex data. CX indicates (X1,X2).
- CDX is double-precision complex data. CDX indicates (DX1,DX2).
- CQX is quadruple-precision complex data. CQX indicates (QX1,QX2).
- PI is about 3.141592.

Note:

In many case, the arguments passed to the user-defined error subroutines are same as or part of argument of intrinsic procedure referencing, however it is not same area.

8.1.6 Intrinsic Subroutine Error Processing

The table below lists the causes of intrinsic subroutine errors and the standard system correction processing.

The following arguments are passed to the user-defined error subroutine:

- Return code
- Error number
- The data listed in the table bellow

Table 8.4 Error processing for intrinsic subroutine

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine
1301	MVBITS(FROM,FROMPOS,LEN,TO,TOPOS)	LEN < 0 or LEN > BIT_SIZE(FROM)	TO is set to Zero	None
		FROMPOS < 0 or FROMPOS >= BIT_SIZE(FROM)-LEN	TO is set to Zero	None
		TOPOS < 0 or TOPOS >= BIT_SIZE(FROM)-LEN	TO is set to Zero	None
1302	DATE_AND_TIME([DATE,TIME,ZONE,VALES])	The length of DATE is less than 8.	The value to be returned is truncated from the same length as DATE.	None
		The length of TIME is less than 10.	The value to be returned is truncated from the same length as TIME.	None
		The length of ZONE is less than 5.	The value to be returned is truncated from the same length as ZONE.	None
1303	DATE_AND_TIME([DATE,TIME,ZONE,VALES])	Size of VALES is less seven	The values are returned in VALES, and the rest of the values to be returned are discarded.	None
1304	RANDOM_SEED([SIZE,PUT,GET])	Size of PUT is less SIZE	The seed values are returned in PUT, and therest of the values to be returned are discarded.	None
		Size of GET is less SIZE	The seed values are returned in GET, and therest of the values to be returned are discarded.	None
1305	RANDOM_SEED([SIZE,PUT,GET])	Two or three arguments are specified.	Terminates the program	None
1306	MOVE_ALLOC([FROM,TO])	The length of argument FROM and TO of the character type is different	Terminates the program	None

Error number	Reference format	Cause of error	Standard correction processing	Data passed to user-defined subroutine
1422	IEEE_GET_HALTING_MODE(FLAG,HALTING)	The function used is not supported.	Nothing is done.	None
	IEEE_SET_HALTING_MODE(FLAG,HALTING)			
	IEEE_GET_UNDERFLOW_MODE(GRADUAL)			
	IEEE_SET_UNDERFLOW_MODE(GRADUAL)			
	IEEE_VALUE(X,CLASS)			
1423	IEEE_SET_ROUNDING_MODE(ROUND_VALUE)	The function used is not supported.	Nothing is done.	None

8.1.7 Exception Handling Processing

This system has several program interrupts for error processing. These interrupts are exponent overflow, exponent underflow, division check, and integer overflow. If runtime option -i is specified, no exception handling is taken. See Section "3.3 Runtime Options" and Section "8.2.2 Debugging Programs for Abend" for details.

The table below shows an exception handling.

Table 8.5 Exception handling processing

Error number	Interrupt cause	Standard correction	User-defined correction	Data passed to the user-defined error subroutine	
				Data1	Data2
11	The result of the floating- point operation exceeds |3.40282347e +38|(real) |1.797693134862316d+308| (double-precision real)	Terminates the program	Not correctable	None	
12	The result of the floating- point operation is less than | 1.17549435e-38|(real) | 2.225073858507201d- 308| (double-precision real)	Terminates the program	Not correctable	None	
13	The divisor in floatingpoint division is zero.	Terminates the program	Not correctable	None	
14	The divisor in fixedpoint division is zero, or absolute value of quotient is more than 2**31.	Terminates the program	Not correctable	None	
17	Terminated abnormally due to run time option -t	Terminates the program	Not correctable	None	
18	Terminated abnormally due to run time option -a	Terminates the program	Not correctable	None	
19	Terminated abnormally due to an invalid access.	Terminates the program	Not correctable	None	
20	Detected an error during abnormally termination process	Terminates the program	Not correctable	None	

Error number	Interrupt cause	Standard correction	User-defined correction	Data passed to the user-defined error subroutine	
				Data1	Data2
292	An invalid calculation in floating-point is executed.	Terminates the program	Not correctable	None	

Note:

- When standard correction processing sets a value in the result register, the sign is unchanged. In addition, the maximum values are as follows:
Default real : 3.40282347e+38
Double-precision real : 1.797693134862316d+308

- The above interrupts do not occur for quadruple-precision real. See Section "8.1.5 Intrinsic Function Error Processing" for details.

8.1.8 Other Error Processing

The table below lists the error processing performed when the system detects an error other than an input/output, intrinsic function, or program interrupts error. The following arguments are passed to the user-defined error subroutine:

- Return code
- Error number
- The data listed in the table below

Table 8.6 Other error processing

Error number	Standard correction processing	User-defined correction	Data passed to the user-defined error subroutine
301	Continues processing, assuming that first argument is default value	Not correctable	None
304	CALL PRNSET(IX) Continues processing using IX = 0 for IX < 0, or IX = 15 for IX > 15	Not correctable	IX
306	Terminates the program	Not correctable	None
308	Terminates the program	Not correctable	None
311 to 318	Continues checking arguments	Not correctable	None
320	Continues processing	Not correctable	None
322 to 324	Continues processing	Not correctable	None
329	Terminates the program	Not correctable	None
330	Continues processing	Not correctable	None
1001	Terminates the program	Not correctable	None
1003 to 1008	Terminates the program	Not correctable	None
1031	Terminates the program	Not correctable	None
1032	The multiple number is defined by CPU number and the program is continued.	Not correctable	None
1033 (*1)	Terminates the program	Not correctable	None

Error number	Standard correction processing	User-defined correction	Data passed to the user-defined error subroutine
1034	Continues processing	Not correctable	None
1035	Terminates the program	Not correctable	None
1036	Continues processing	Not correctable	None
1038 to 1041	Terminates the program	Not correctable	None
1042	The multiple number is defined by num and the program is continued.	Not correctable	None
1043	Continues processing	Not correctable	None
1044 to 1046	Terminates the program	Not correctable	None
1047	Continues processing	Not correctable	None
1048	Terminates the program	Not correctable	None
1049	Continues processing	Not correctable	None
1050	Continues processing	Not correctable	None
1071 to 1072	Continues processing	Not correctable	None
1141 to 1143	Continues processing	Not correctable	None
1231	Continues processing	Not correctable	None
1232	Terminates the program	Not correctable	None
1551	Continues processing	Not correctable	None
1561	Continues processing	Not correctable	None
1563 to 1564	Terminates the program	Not correctable	None
1565 to 1566	Continues processing	Not correctable	None
1569	Continues processing	Not correctable	None
1570 to 1571	Continues checking arguments	Not correctable	None
1572 to 1575	Terminates the program	Not correctable	None
1576 to 1577	Continues processing	Not correctable	None
1578	Continues processing	Not correctable	None
1579	Terminates the program	Not correctable	None
1582	Continues processing	Not correctable	None
1652	Continues processing	Not correctable	None

IX: Default integer scalar

*1: The diagnostic message (jwe1033i-i) indicates that the Fortran program has been terminated during execution on multiple processors. If this message is output, the results of the Fortran program may be different from the results of terminating the program while running on a single processor by executing a STOP/ERROR STOP statement or an EXIT service subroutine, or by exceeding the error count limit.

8.2 Debugging Functions

This section describes the check functions for debugging Fortran programs and the debug information output when execution ends abnormally.

8.2.1 Debugging Check Functions

The debugging check functions perform debugging during compilation or execution.

The -Ncheck_global compiler option is available as check functions to perform debugging during compilation.

During compilation, the compiler option -Ncheck_global checks the following:

- The procedure characteristics between external procedure definitions and references
- The procedure characteristics between external procedure definitions and interface body.
- In program units the size of common blocks

During compilation, the compiler option -Ha or -Nquickdbg=argchk checks the following:

- Array size in procedure in explicit reference specification.

During compilation, the compiler option -Hs or -Nquickdbg=subchk checks the following:

- The ranges of the declaration and reference for array element, and substring.

During execution, the compile option -H and -Nquickdbg check the following:

Table 8.7 List of runtime check items

	-H option	-Nquickdbg option
Checking Argument Validity (ARGCHK)		
Checking the Number of Arguments	-Ha	-Nquickdbg=argchk
Checking the Argument Types		
Checking Argument Attributes		-
Checking Function Types		-Nquickdbg=argchk
Checking Argument Sizes		-
Checking Explicit Interface		-Nquickdbg=argchk
Checking Rank of Assumed-Shape Array		-
Checking Subscript and Substring Values (SUBCHK)		
Checking Array and Substring References	-Hs	-Nquickdbg =subchk
Checking Array Section References		-
Checking References for Undefined Data Items (UNDEF)		
Checking for Undefined Data	-Hu	-Nquickdbg =undef -Nquickdbg =undefnan
Checking for an Allocatable Variable		-
Checking for an Optional Argument		-
Shape Conformance Check (SHAPECHK)		
Shape Conformance Check	-He	-
Extended Checking of UNDEF (EXTCHK)		
Checking for Undefined Data of a Module and a Common Block	-Hx	-
Checking for Unassociated Pointer		-

	-H option	-Nquickdbg option
Checking for Stride and Increment Value		-
Checking for DO Variable		-
Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP)		
Overlapping Dummy Arguments	-Ho	-
Checking for undefined Assumed-size Array with INTENT(OUT)		-
Checking for undefined variable of SAVE attribute		-
Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)		
Checking Connection of a File to A Unit	-Hf	-
I/O Recursive Call CHECK		-

- : No support

Details of checked items are given in the following sections:

- Check argument validity. (See Section "8.2.1.1 Checking Argument Validity (ARGCHK)")
- Check subscript and substring values. (See Section "8.2.1.2 Checking Subscript and Substring Values (SUBCHK)")
- Check references for undefined data. (See Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)")
- Check shape conformance. (See Section "8.2.1.4 Shape Conformance Check (SHAPECHK)")
- Extended check references for undefined data. (e.g. the variable declared in a module or in a common block).(See Section "8.2.1.5 Extended Checking of UNDEF (EXTCHK)")
- Check the part of overlap is changed when two dummy arguments are overlapped. (See Section "8.2.1.6 Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP)")
- Check the file is connected with two devices or more. (See Section "8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)")
- Check an input/output statement is executed from other input/output statement. (See Section "8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)")

The compiler option -H and -Nquickdbg check have the characteristics shown following:

Table 8.8 Characteristics of runtime check functions

	-H option	-Nquickdbg option
Effect on execution performance	Large	Small
Checked items	Many	Few
Thread parallelism (*1)	Not possible	Possible

*1: Checks are possible/not possible during OpenMP or automatic parallel execution.

Compared with the check function of the compiler option -H, the check function of the compiler option -Nquickdbg has less effect on execution performance because the items checked are limited. Debugging is also enabled during OpenMP and automatic parallel execution.

The compiler option -Nquickdbg includes the compiler option -Nquickdbg=inf_detail and -Nquickdbg=inf_simple.

If the compiler option -Nquickdbg=inf_detail is enabled, in addition to the message and line number where the error occurred, information is output such as the variable name, procedure name, and element location in order to identify the cause of the error.

If the compiler option -Nquickdbg=inf_simple is enabled, the error and the line number where it occurred are displayed in the diagnostic message. However, if the compiler option -Nnoline is enabled, the value of line number is not guaranteed.

By limiting the information included in diagnostic messages, the compiler option -Nquickdbg=inf_simple has less impact on execution performance than the compiler option -Nquickdbg=inf_detail check function.

See Section "2.2 Compiler Options" for more information about -H, -Nquickdbg, and -Ncheck_global.

8.2.1.1 Checking Argument Validity (ARGCHK)

When the subprogram, a subroutine or function, is called, the compiler option -Ha or -Nquickdbg=argchk checks the validity of the dummy and actual arguments.

However, checks are not performed in the following cases:

- The actual argument is an actual name that has a POINTER attribute (including cases in which the actual argument is a derived type variable with one or more components that have a POINTER attribute).
- An actual argument is an array section or array expression (other than an array variable).
- A dummy procedure name is used to call a subprogram.
- A derived type actual argument with an ultimate component that is an allocatable variable.
- A procedure pointer or type bound procedure is referred.

8.2.1.1.1 Checking the Number of Arguments

The compiler option -Ha or -Nquickdbg=argchk checks whether or not the number of actual arguments matches the number of dummy arguments. The number of actual arguments also includes the alternate return specifier. If the numbers of arguments do not match, a diagnostic message jwe0313i-w is output.

8.2.1.1.2 Checking the Argument Types

The compiler option -Ha or -Nquickdbg=argchk checks whether or not the dummy argument type matches the corresponding actual argument type. If the argument types do not match, a diagnostic message jwe0315i-w is output.

Argument types are not checked in the following cases:

- The dummy argument is a procedure name or an asterisk (*).
- The actual argument is an external procedure name or an alternate return specifier (but argument types are checked if the dummy argument and actual arguments are external procedure names that have types).

If the compiler option -Ha is set, the following checks are also performed

If the actual argument and dummy argument are derived type variables, the function checks whether or not the types of the components match. If the types do not match, a diagnostic message jwe1561i-w is output.

If the argument type is the character type, the function checks whether or not the length of the dummy argument exceeds that of the actual argument.

If the length is exceeded, a diagnostic message jwe0317i-w is output.

8.2.1.1.3 Checking Argument Attributes

The compiler option -Ha checks whether or not the dummy argument attributes matches the corresponding actual argument attributes. A dummy argument may be associated only with an actual argument as shown in "Table 8.9 Correspondence for dummy and actual arguments". If a dummy argument is not associated, the diagnostic message jwe0314i-w is output. If a dummy argument has INTENT(IN), the actual argument must be defined.

If an actual argument is a constant or an expression that is not a variable, diagnostic message jwe0318i-w is output if the corresponding dummy argument is changed in the subprogram.

"Table 8.9 Correspondence for dummy and actual arguments" lists the correspondence between dummy and actual arguments.

Table 8.9 Correspondence for dummy and actual arguments

Dummy argument	Actual argument
Scalar variable	Scalar variable, substring, array element, constant, result of scalar expression
Array excluding character type	Actual argument excluding character type Whole array, array section, array element, substring of array element, result of array expression

Dummy argument	Actual argument
Character type array	Actual argument of character type Scalar variable, substring, constant, result of scalar expression, whole array, array section, array element, substring of array element, result of array expression
Dummy procedure	External procedure, intrinsic procedure, or module procedure
*	Statement label

8.2.1.1.4 Checking Function Types

The compiler option -Ha or -Nquickdbg=argchk checks whether or not the type of function defined in a function subprogram matches the type of the reference source external procedure. If the function types do not match, a diagnostic message jwe0311i-w is generated if the function types do not match.

If a function subprogram is referenced as a subroutine subprogram, or if a subroutine subprogram is referenced as a function subprogram, a diagnostic message jwe0330i-w is output.

If the compiler option -Ha is set, the following checks are also performed.

If the function type is the character type, the function checks whether or not the lengths are the same. If the function lengths are not the same, a diagnostic message jwe0312i-w is output.

8.2.1.1.5 Checking Argument Sizes

For an array dummy argument, the compiler option -Ha checks whether or not the size of array dummy argument exceed the corresponding actual argument. If the upper bound of the final dimension of the dummy array is an asterisk (*), or the upper and lower bounds of the dummy array are 1, the size of the dummy array is assumed to be equal to the corresponding actual argument size. Then this check isn't performed.

If type of assumed-shape array is character type, character lengths are checked to confirm that the length of the dummy argument and the actual argument match. If the length does not match a diagnostic message jwe1571i-w is output.

As listed in "Table 8.10 Array size", the size of the actual argument that corresponds to a dummy argument array is obtained from the actual argument attribute. If the size of the dummy array exceeds the actual argument size, a diagnostic message jwe0316i-w or jwe1577i-w is output.

Table 8.10 Array size

Actual argument attribute	Actual argument size
Array	Whole array size
Array element	Size from the specified array element up to the end of the array
Substring of array element	Size from the beginning of the specified array element substring to the end of the array
Scalar expression of character type excluding array element and substring of array element	Length for scalar expression of character type

8.2.1.1.6 Checking Explicit Interface

If an explicit reference specification is not present in an external procedure reference that requires an explicit reference specification when the compiler option -Ha or -Nquickdbg=argchk is set, a diagnostic message jwe1569i-w is output.

8.2.1.1.7 Checking Rank of Assumed-Shape Array

If a dummy argument is an assumed-shape array, when the compiler option -Ha is set, the rank is compare to actual argument and dummy argument. If the rank is different between actual argument and dummy argument, a diagnostic message jwe1570i-w is output.

8.2.1.2 Checking Subscript and Substring Values (SUBCHK)

If the compiler option -Hs or -Nquickdbg=subchk is set, the validity of the referenced subscript or substring expression is checked against the declared subscript and substring expression.

However, this check is not performed in the following cases:

- The referenced expression has the POINTER attribute (including cases in which a structure component has a POINTER attribute).
- An assumed-shape array is referenced.
- An array section specified by a vector subscript is referenced.
- An array in a masked array assignment is referenced.
- A named constant is referenced.
- The parent string of substring is a scalar constant.
- Data is referenced by a declaration expression.
- A derived type variable with an ultimate component that is an allocatable variable.
- The subscript range of io-implied-do (but checked if the -Hs compiler option is set).

This check is performed during compilation and execution. If an error is detected during compilation, a w level diagnostic message is output. The items checked during execution are explained below.

8.2.1.2.1 Checking Array and Substring References

If the compiler option -Hs or -Nquickdbg=subchk is set, when substring, array element, or array section is referenced, the array reference check determines whether or not those subscript and substring ranges are within the declared ranges. If it isn't within the declared range, a diagnostic message (-Hs, -Nquickdbg=inf_detail:jwe0320i-w, -Nquickdbg=inf_simple:jwe1606i-u) is output.

If the upper and lower bounds of the dummy array are 1, the dummy array is regarded as assumed-size array. If the diagnostic message is output by the compiler option -Nquickdbg=subchk, the maximum number of 8 byte integer is output as the upper bound of the last dimension of assumed-size array.

If the compiler option -Hs is set, the following checks are also performed

If the upper limit of an explicit shape array is a smaller value than the lower limit, a diagnostic message jwe1566i-w is output.

If the compiler option -Hs and -Ha are set simultaneously, this check whether or not an array element or substring is within the corresponding actual argument range when it corresponds to a dummy array in which the upper limit of the last dimension is an asterisk (*) or in which the upper and lower limit is the constant 1. If it is not within the actual argument range, a diagnostic message (jwe0322i-w) is output.

8.2.1.2.2 Checking Array Section References

If the compiler option -Hs is set, the following check is performed.

If an array section with a subscript triplet is referenced, the result of adding the first subscript to the extent of the subscript triplet is checked to determine whether the value exceeds the maximum value of an integer or is less than the minimum value of an integer. In either case, diagnostic message jwe1565i-w is output.

8.2.1.3 Checking References for Undefined Data Items (UNDEF)

If the compiler option -Hu, -Nquickdbg=undef, or -Nquickdbg=undefnan is set, the function checks whether or not referenced data is defined.

However, this check is not performed in the following cases:

- An actual name that has a POINTER attribute is referenced (including cases in which a structure component has a POINTER attribute, but checked if the compiler option -Nquickdbg=undef or -Nquickdbg=undefnan is set).).
- A structure component has an ALLOCATABLE attribute.
- An assumed-shape array is referenced (but checked if the compiler option -Nquickdbg=undef or -Nquickdbg=undefnan is set).
- An array section specified by a vector subscript is referenced.
- Data in a masked array assignment is referenced.
- A derived type variable that has an allocatable array to an ultimate component is referenced.
- A variable of derived type having length type parameter is referenced.

- An array allocatable to an ultimate component by a structure component is referenced (but checked if the compiler option -Nquickdbg=undef or -Nquickdbg=undefnan is set).
- A structure component is referenced (but checked if the compiler option -Hu or -Nquickdbg=undef is set).
- The entities that share a storage sequence have different types (but checked if the compiler option -Hu or -Nquickdbg=undef is set).
- The subscript range of io-implied-do (but checked if the compiler option -Hu is set).

If diagnostic messages jwe0320i-w, jwe0322i-w, jwe1565i-w, and jwe1566i-w are detected by the SUBCHK function checks when the compiler option -Hu and -Hs are set simultaneously, the UNDEF function is disabled.

Undefined data checks are explained below.

8.2.1.3.1 Checking for Undefined Data

If the compiler option -Hu or -Nquickdbg=undef is set, when module declaration part or variable not included in common block is referenced, the values is checked to be defined for that data. When array variable is referenced, check is performed for each array element. If derived type variable is referenced when the compiler option -Hu is set, check is performed for each of the structure variable components.

If values is not defined, a diagnostic message (-Hu,-Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u) is output.

If a function subprogram ends when the compiler option -Hu is set, the values in the function results is checked. If it is not defined in the function results, a diagnostic message (-Hu, -Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u) is output. The line number at that time becomes the END statement of the function subprogram.

If data value has the values below, a diagnostic message (-Hu, -Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u) is output, but, as the Fortran program is correct, no correction is required. To change the data values below, specify the FLIB_UNDEF_VALUE environment variable. See Section "3.8 Using Environment Variables for Execution" for detail.

One-byte integer	: -117
Two-byte integer	: -29813
Four-byte integer	: -1953789045
Eight-byte integer	: -8391460049216894069
Single-precision real	: -5.37508134e-32
Double precision real	: -4.696323204354320d-253
Quadruple precision real	: -9.0818487627532284154072898964213742q-4043
Single-precision complex	: (-5.37508134e-32,-5.37508134e-32)
Double precision complex	: (-4.696323204354320d-253,-4.696323204354320d-253)
Quadruple precision complex	: (-9.0818487627532284154072898964213742q-4043,
	-9081848762753228415407289896421374 2q-4043)
Character	: Z'8B'

8.2.1.3.2 Checking for Undefined Data Using Invalid Operation Exception

If the compiler option -Nquickdbg=undefnan is set, when module declaration part or variable not included in common block is referenced, the value is checked to be defined for that data. When array variable is referenced, check is performed for each array element. If value is not defined, a diagnostic message (-Hu,-Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u) is output.

Output diagnostic message is different according to a variable and a type of function results.

If integer type and character type checks show that values are not defined, a diagnostic message (-Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u) is output.

If data has the values below, a diagnostic message (-Hu, -Nquickdbg=inf_detail:jwe0323i-w, -Nquickdbg=inf_simple:jwe1606i-u) is output, but, as the Fortran program is correct, no correction is required. To change the data values below, specify the environment variable FLIB_UNDEF_VALUE. See Section "3.8 Using Environment Variables for Execution" for detail.

One-byte integer	:	-117
Two-byte integer	:	-29813

Four-byte integer	:	-1953789045
Eight-byte integer	:	-8391460049216894069
Character	:	Z'8B'

The real type and complex type variable have SNaN as the initial value. If values aren't defined, the real type and complex type checks detect an invalid operation exception (jwe0292i-u).

However, the invalid operation exception may not be detected in the following case:

- The real type and complex type variables do not appear in the operands of the command that issues invalid operation exceptions. (The CPU architecture determines which instruction issues invalid operation exception. For example, in an architecture where the reference instruction does not issue exception, undefined variable is detected during referencing when the compiler option -Nquickdbg=undef is set, but undefined variable is not detected during referencing when the compiler option -Nquickdbg=undefnan is set.)
- The invalid operation exception is masked.

In addition undefined data reference, the invalid operation exception is detected when the execution result is SNaN.

8.2.1.3.3 Checking for an Allocatable Variable

If the compiler option -Hu is set, the variable checked to already allocated by ALLOCATE statement when an ALLOCATABLE variable is referenced; if not, diagnostic message jwe0324i-w is generated.

8.2.1.3.4 Checking for an Optional Argument

If the compiler option -Hu is set, the argument checked to be present when an optional argument is referenced; if not, diagnostic message jwe1575i-s is generated.

8.2.1.4 Shape Conformance Check (SHAPECHK)

When an array expression or array assignment statement is executed, the compiler option -He checks shape conformance.

Diagnostic message jwe0329i-w is generated if a mismatch of shape conformance occurs during execution.

8.2.1.5 Extended Checking of UNDEF (EXTCHK)

The -Hx option checks to see if the variable declared in a module or in a common block is defined when the variable is referenced, to see if the pointer is associated when a pointer is referenced, and to see if the incrementation parameter of the DO construct, stride of the FORALL, incrementation parameter of array constructor implied do control and stride of subscript-triplet is not 0.

Diagnostic message jwe0323i-w is generated for the variable declared in a module or in a common block that is not defined. Note that the message is generated if the item is defined with a value listed in Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF) ", even though the Fortran program may be correct.

A Pointer is referenced, the pointer checked to be associated with target; if not, diagnostic message jwe1572i-s is generated. However, this check facility is not performed for a pointer of derived type and a pointer of component.

If the incrementation parameter of the DO construct, stride of the FORALL, incrementation parameter of array constructor implied do control and stride of subscript-triplet is zero, diagnostic message jwe1573i-s is generated.

In the range of a DO construct, if an actual argument is a do-variable, the definition of do-variable is checked. Diagnostic message jwe1574i-s is generated if the do-variable is defined.

The execution of a RETURN or an END statement within a subprogram causes all local variables in that scope unit without the SAVE attribute becomes undefined when the -Hx option is present.

When the variables have a BIND attribute, the undefined value is not checked.

If the -Hx option is specified, the -Hu option is in effect, see Section "8.2.1.3 Checking References for Undefined Data Items (UNDEF)".

Be careful to the following when the -Hx option using:

- All program units that have a definition or initialization for common block object shall be compiled with -Hx option.

  Example: Invalid specification of -Hx

  File: a.f90

```
BLOCK DATA INIT
COMMON /CMN/ J
DATA J/200/
END BLOCK DATA
```

File: b.f90

```
PROGRAM MAIN
COMMON /CMN/ J
PRINT *,J
END PROGRAM
```

Command:

```
$ frtpx a.f90 -c
$ frtpx b.f90 a.o -Hx
```

The -Hx options shall be specified for compiling a.f90.

- All program units that uses the module shall be compiled with -Hx option, if the module is compiled with -Hx option.

  Example: Invalid specification of -Hx

  File: a.f90

```
MODULE MOD
INTEGER :: J
END MODULE
```

  File: b.f90

```
PROGRAM MAIN
USE MOD
J = 200
CALL SUB()
END PROGRAM
```

  File : c.f90

```
SUBROUTINE SUB()
USE MOD
PRINT *,J
END SUBROUTINE
```

  Command:

```
$ frtpx a.f90 -c -Hx
$ frtpx b.f90 -c
$ frtpx c.f90 a.o b.o -Hx
```

  The -Hx options shall be specified for compiling b.f90.

8.2.1.6 Overlapping Dummy Arguments and Extend Undefined CHECK (OVERLAP)

When compiler option -Ho is specified, the following check is done.

- Two dummy arguments are overlapped and the part of overlap is changed.
- When an assumed-size array with INTENT(OUT) attribute is referenced during execution, checks to see if the variable is defined.
- When a variable with SAVE attribute is referenced during execution, checks to see if the variable is defined.

8.2.1.6.1 Overlapping Dummy Arguments

Do not define or undefine the value of the part overlapping executing the procedure when two dummy arguments are overlapping in the same procedure. When compiler option -Ho is specified, the function checks whether the value is defined or undefined while executing.

Diagnostic message jwe1576i-w is generated if a mismatch occurs during execution. The validity is not checked if any of the following conditions is true.

The validity is not checked if:

- The dummy argument is defined in a masked array assignment.
- The dummy argument is defined in FORALL assignment statement.
- The dummy argument as DO variable is defined.
- The dummy argument is defined with array section of vector subscript.
- The dummy argument is defined and the corresponding actual argument has no continuation element.
- The dummy argument is defined in reference of procedure.
- When the dummy argument which has TARGET attribute or POINTER attribute is overlapping the object of the other dummy argument which has neither TARGET attribute nor POINTER attribute, and the dummy argument is defined.
- When the dummy argument of assumed shape array is overlapping the object of the other dummy argument, and the dummy argument is defined.

8.2.1.6.2 Undefined Assumed-size Array with INTENT(OUT)

When compiler option -Ho is specified, the function checks whether assumed-size array with INTENT(OUT) attribute is defined while executing. Diagnostic message jwe0323i-w is generated if a mismatch occurs during execution. If compiler option -Hs also is specified, the function does not check the data if diagnostic message jwe0320i-w, jwe0322i-w, jwe1565i-w, or jwe1566i-w is generated.

The validity is not checked if:

- A pointer variable is referenced.
- The referenced expression is an array section with a vector subscript
- The referenced expression is in a masked array assignment.
- The referenced expression is a derived type variable with an ultimate component that is an allocatable variable.
- The referenced expression is an allocatable variable as an ultimate component by a structure component.

8.2.1.6.3 Undefined Variable of SAVE Attribute

When compiler option -Ho is specified, the function checks whether variable with SAVE attribute is defined while executing. Diagnostic message jwe0323i-w is generated if a mismatch occurs during execution. If compiler option -Hs also is specified, -Hu option does not check the data if diagnostic message jwe0320i-w, jwe0322i-w, jwe1565i-w, or jwe1566i-w is generated.

The validity is not checked if:

- The referenced expression has a structure variable one of whose structure components has the POINTER attribute.
- The referenced expression is an array section with a vector subscript.
- The referenced expression is in a masked array assignment.
- The referenced expression is a derived type variable with an ultimate component that is an allocatable variable.
- The referenced expression is an allocatable variable as an ultimate component by a structure component.

8.2.1.7 Simultaneous OPEN and I/O Recursive Call CHECK (I/O OPEN)

When compiler option -Hf is specified, the following check is done.

A file is connected with two devices or more at the same time in the input/output statement.

An input/output statement is called by the function while executing other input/output statement.

8.2.1.7.1 Checking Connection of a File to A Unit

In the input/output statement, do not connect the one file with two devices or more at the same time. When compiler option -Hf is specified, the function checks whether the one file is connected with two devices or more at the same time while executing. Diagnostic message jwe1231i-w is generated if a mismatch occurs during execution.

The validity is not checked if:

- The file is specified by the file names other than the real name when file system that the file is not local is used.
- As for this file, specification different from the real name is used when the file is defined and the alias is defined by the "alias" command.
- "../" is included in the path of the file.

8.2.1.7.2 I/O Recursive Call CHECK

Do not recurrently execute the input/output statement. When compiler option -Hf is specified, the function checks whether the input/output statement is executed recurrently while executing. Diagnostic message jwe1232i-s is generated if a mismatch occurs during execution.

8.2.2 Debugging Programs for Abend

If an abnormal termination event (hereafter called an abend) occurs during execution of a Fortran program, the system generates information to help the user determine the cause of the abend.

8.2.2.1 Causes of Abend

"Table 8.11 Detected errors and corresponding signal codes" lists the errors that the system detects and the corresponding signal codes. If the -i runtime option is specified, the system does not check for these errors.

If the -NRnotrap compiler option is specified, the system does not check for floating point exceptions (jwe0011i-u, jwe0012i-u, jwe0013i-u or jwe0292i-u). For more information about compiler option -NRnotrap, see Section "2.2 Compiler Options".

Table 8.11 Detected errors and corresponding signal codes

Signal number	Meaning of signal number	Signal code		Meaning of signal code
SIGILL(04) *	Incorrect instruction executed	1	ILL_ILLOPC	Illegal opcode
		2	ILL_ILLOPN	Illegal operand
		3	ILL_ILLADR	Illegal addressing mode
		4	ILL_ILLTRP	Illegal trap
		5	ILL_PRVOPC	Privileged opcode
		6	ILL_PRVREG	Privileged register
		7	ILL_COPROC	Coprocessor error
		8	ILL_BADSTK	Internal stack error
SIGFPE(08)	Arithmetic exception	1	FPE__INTDIV *	Fixed-point division exception
		3	FPE__FLTDIV	Floating-point division exception
		4	FPE__FLTOVF	Floating-point overflow exception
		5	FPE__FLTUND	Floating-point underflow exception (*1)
		7	FPE_FLTINV	Invalid floating-point processing
SIGBUS(10) *	Storage protection exception	1	BUS_ADRALN	Invalid address alignment
		2	BUS_ADRERR	Non-existent physical address
		3	BUS_OBJERR	Object specific hardware error
SIGSEGV(11) *	Segmentation exception	1	SEGV_MAPERR	Address not mapped to object

Signal number	Meaning of signal number	Signal code		Meaning of signal code
		2	SEGV_ACCERR	Invalid permissions
SIGXCPU(30) *	CPU time interrupt	--	--	--

*1: Interrupt detected when the -NRtrap compiler option and the environment variable FLIB_EXCEPT=u (exponent underflow) were specified.

Note:

1. For signal numbers marked with an asterisk, a core file is created when the -i runtime option is specified to disable system error detection.

8.2.2.2 Information Generated when an Abend Occurs

Depending on the error type, information is written to the standard error file as explained in the following sections. Information that cannot be written to the standard error file is sent to the terminal.

8.2.2.2.1 Information Generated for a General Abend

The information generated for a general abend is follows. This information is generated when SIGILL, SIGBUS, or SIGSEGV is detected. A trace back map is printed after this information.

```
jwe0019i-u The program was terminated abnormally with signal number SSSSSSS.
           signal identifier = NNNNNNNNNN, (Detailed information.)
```

SSSSSSS: SIGILL, SIGBUS, or SIGSEGV

NNNNNNNNNN: Signal code for the abend cause

(Detailed information.): Detailed information of Signal code *NNNNNNNNNN*

When it is strong possibility that the cause of SIGBUS or SIGSEGV is satck-overflow, add the following information.

```
The cause of this exception may be stack-overflow
```

8.2.2.2.2 Information of SIGXCPU

The output when SIGXCPU is detected is follows.

```
jwe0017i-u The program was terminated with signal number SIGXCPU.
```

8.2.2.2.3 Information Generated by Run-Time Option -a

The information generated by runtime option -a when SIGIOT is detected is follows. The program is forced to abend without deleting the temporary files being used by the program and a core file is created.

The cause of this exception may be stack-overflow.

```
jwe0018i-u The program was terminated abnormally due to runtime option -a with signal number SIGIOT.
```

8.2.2.2.4 Information when an Abend Occurs Again during Abend Processing

The output when an abend occurs again during abend processing is follows.

```
jwe0020i-u An error was detected during an abnormal termination process.
```

8.2.3 Hook Function

This section describes the hook function which calls a user-defined subroutine by specified location in a Fortran program, or by regular time interval. Program operation can be checked via a user-defined subroutine to output trace information and specific variable values.

8.2.3.1 User-defined Subroutine Format

The user-defined subroutine name is fixed (USER_DEFINED_PROC).

Define a user-defined subroutine in the format shown below.

Format:

```
SUBROUTINE USER_DEFINED_PROC(FLAG, NAME, LINE, THREAD)
INTEGER(KIND=4),INTENT(IN):: FLAG, LINE, THREAD
CHARACTER(LEN=*),INTENT(IN):: NAME
```

Arguments:

FLAG	:	Indicate the calling source for the user-defined subroutine.
= 0	:	Program entry
= 1	:	Program exit
= 2	:	Procedure entry
= 3	:	Procedure exit
= 4	:	Parallel region (OpenMP or automatic parallelization) entry
= 5	:	Parallel region (OpenMP or automatic parallelization) exit
= 6	:	Regular time interval
= 7~99	:	Reserved for system
= 100~	:	Can be used by the user
NAME	:	Indicate the calling source procedure name. NAME can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.
LINE	:	Indicate the calling source line number. LINE can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.
THREAD	:	With thread parallelization using OpenMP or automatic parallelization, indicate the number of the thread that calls the user-defined subroutine. THREAD can be referenced only if the FLAG is 2, 3, 4, 5, or 100 or greater.

8.2.3.2 Notes on the Hook Function

The following notes apply when the hook function is used:

- The procedure name "USER_DEFINED_PROC" must not be used for any other purpose.
- Operation is not guaranteed if the user-defined subroutine is not defined.
- Values must not be set in the user-defined subroutine arguments.
- If the -Nnoline compile option is set, the LINE argument value is not guaranteed.
- In thread parallelized programs, the user-defined subroutine may be called from multiple threads.
- Operation is not guaranteed if the user-defined subroutine is compiled with the -CcI4I8 option set.
- Operation is not guaranteed if the user-defined subroutine is compiled with the -AU option set.
- In procedures that include an ENTRY statement, the procedure name set in the NAME argument may be the entry name of the ENTRY statement.
- The procedure exit becomes the END statement line number.
- The characteristics of a user-defined subroutine procedure cannot be changed.
- The entry and exit of user-defined subroutine except the calling from any location in the program must not be calling source for the user-defined subroutine.

8.2.3.3 Calling a User-defined Subroutine from a Specified Location

When -Nhook_func option is specified when the object program and executable program are created, user-defined subroutine can be called from the following locations:

- Program entry and exit
- Procedure entry and exit

If the -Kopenmp or -Kparallel option is enabled, user-defined subroutine can also be called from the following locations:

- Parallel region (OpenMP or automatic parallelization) entry and exit

The following internal procedure name is passed to NAME argument in OpenMP, automatic parallelization entry and exit.

- OpenMP "_OMP_IdNumber_"
- automatic parallelization "_OMP_IdNumber_"

When the -Nhook_func is set, operation is as follows for Fortran, and C linked programs:

- If user-defined subroutine/function is defined within the Fortran program and the C program respectively, the user-defined subroutine defined in the Fortran program is called from the Fortran program, and the user-defined function defined in the C program is called from the C program.
- If user-defined subroutine/function is defined within only the Fortran program or only the C program, the defined user-defined subroutine/function is called from both the Fortran program and the C program.

An example using the hook function with location specified is shown below.

User-defined subroutine program example:

File: hook.f95

```
WRITE(*,*) 'HELLO'
CALL SUB
END

SUBROUTINE SUB
WRITE(*,*) 'SUB'
END

SUBROUTINE USER_DEFINED_PROC(FLAG,NAME,LINE,THREAD)
INTEGER(KIND=4),INTENT(IN):: FLAG,LINE,THREAD
CHARACTER(LEN=*),INTENT(IN):: NAME
SELECT CASE(FLAG)
  CASE(0)
     WRITE(*,*) 'PROGRAM START'
  CASE(1)
     WRITE(*,*) 'PROGRAM END'
  CASE(2)
     WRITE(*,*) 'PROC START: ',NAME,' LINE: ',LINE
  CASE(3)
     WRITE(*,*) 'PROC END: ',NAME,' LINE: ',LINE
END SELECT
END
```

Executable program creation:

```
$ frt -Nhook_func hook.f95
```

Execution results:

```
$ ./a.out
 PROGRAM START
 HELLO
 PROC START: SUB LINE:  5
 SUB
```

```
 PROC END: SUB LINE:  7
 PROGRAM END
$
```

8.2.3.3.1 Notes on Calling from a Specified Location

The following notes apply when the -Nhook_func option is set:

- If a procedure with few computations is called repeatedly, calling a user-defined subroutine from a procedure entry/exit may affect execution performance.
- If a parallel region with few computations is executed repeatedly, calling a user-defined subroutine from a parallel region entry/exit may affect execution performance.
- User-defined subroutine is not called from procedures called from input-output lists.

8.2.3.4 Calling a User-defined Function at Regular Time Interval

When the -Nhook_time option is set when an executable program is created, the user-defined function is called by regular user time interval.

The calling interval can be specified using the FLIB_HOOK_TIME environment variable. If it is not specified, the user-defined function is called once every minute. See Section "3.8 Using Environment Variables for Execution", for information about the FLIB_HOOK_TIME environment variable.

See the "C User's Guide" or the "C++ User's Guide", for information about the user-defined function.

8.2.3.4.1 Notes on Calling by Regular Time interval

The following notes apply when the -Nhook_time option is set.

- The environment variable FLIB_HOOK_TIME specification must not be changed while program execution is in progress.
- Operation is not guaranteed if a value outside the range 0 to 2147483647 is specified in the environment variable FLIB_HOOK_TIME.
- If the calling interval is short, the user-defined function is called at the regular time intervals but execution performance may be affected.
- The user-defined function must be defined in C or C++ program.

This function calls the user-defined function from asynchronous signals. Therefore, the following restrictions apply to calling the user-defined function at regular time interval:

- Functions excluding async-signal-safe functions cannot be used in the user-defined function.
- When the -Nhook_time option is set, signal processing routines corresponding to SIGVTALRM must not be defined.
- When the -Nhook_time option is set, ALARM(3) service function must not be used.
- The debugger and profiler must not be used for the executable program created with the -Nhook_time option.

8.2.3.5 Calling from Any Location in the Program

A user-defined subroutine can be called from any location in a program by specifying arguments (FLAG:100~, NAME, LINE, THREAD).

Chapter 9 Optimization Functions

This chapter explains functions to use optimization functions effectively, as well as some points to note.

9.1 Overview of Optimization

Optimization aims to generate object modules (instructions and data areas) that allow the program to execute as quickly as possible.

Some optimization functions can significantly increase the size of the object program. Therefore, the amount of virtual storage space can also increase.

Therefore, it can be selected whether to use optimization function by options.

See Section 2.2 Compiler Options for more information about optimizations parameters.

9.1.1 Standard Optimization

When an optimization level between level 1 (-O1 option) and level 3 (-O3 option) is specified, standard optimization is performed.

To determine whether each optimization was applied, specify the -Koptmsg=2 option.

9.1.1.1 Elimination of Common Expression

If two expressions that give equal calculation results (common expressions) are present, the result of the former expression can be used in the latter expression without calculation.

An example of elimination of common expressions is given below.

Example:

[Original source-code]

```
...
A=X*Y+C
...
B=X*Y+D
```

->

[Optimized pseudo-code]

```
T=X*Y
A=T+C
...
B=T+D
```

The X*Y portion of the expression on the right hand side of the assignment is a common expression. The code is changed so that the second calculation is eliminated and the result of the first calculation T is used instead. T is a variable generated by the compiler.

9.1.1.2 Movement of Invariant Expressions

Expressions with values that do not change within a loop (invariant expressions) are moved outside the loop.

An example of the movement of invariant expressions is given below.

Example:

[Original source-code]

```
DO I=1,N
  Y=A(J)*2
  X=X+Y*Z
ENDDO
```

->

[Optimized pseudo-code]

```
Y=A(J)*2
T=Y*Z
DO I=1,N
  X=X+T

ENDDO
```

The entire statement Y=A(J)*2 and the Y*Z portion of the expression are moved outside the loop. T is a variable generated by the compiler.

The objects that are usually moved by this optimization are statements or parts of statements that are always executed within the loop.

However, if the -Kpreex option is specified, invariant expressions in statements that are selectively executed within the loop depending on a decision statement are also moved.

To differentiate this optimization from the normal movement of invariant expressions, it is called advance evaluation of invariant expressions. This optimization can further reduce execution time.

However, side effects may occur due to the movement. For details of the side effects, see Section 9.1.2.3 Advance evaluation of invariant expressions.

9.1.1.3 Reducing the Strength of Operators

The "strength" of operators describes the relative amount of execution time required for operation. A "strong" operator requires a large amount of execution time. Typically, addition and subtraction have the same strength. Multiplication is stronger than addition or subtraction and division is even stronger than multiplication. In addition, type conversion between the integer and float types is stronger than addition or subtraction but weaker than multiplication.

An example of reducing strength is given below.

Example:

[Original source-code]

```
DO I=1,10
  ...
  J=K+I*100
  ...
  ...
ENDDO
```

->

[Optimized pseudo-code]

```
T=100
DO I=1,10
  ...
  J=K+T
  ...
  T=T+100
ENDDO
```

Optimization is performed on the operation I*100 on the derivative variable I, so the operation I*100 within the loop is converted to the addition T=T+100. t is a variable generated by the compiler.

9.1.1.4 Loop Unrolling

Applying loop unrolling, all of the executable statements in a loop are unrolled *n* times and the loop iteration count is divided by *n*. However, if the compile-time option -Ksimd[=*level*] is set and the loop has been SIMD optimized, the executable statement is unrolled 2**n*, and the loop iteration count is divided by 2**n*.

The value of *n* is determined by the compiler, depending on the iteration count and the number of executable statements in the loop.

The object module size increases because the statements in the loop are unrolled many times.

Loop unrolling can be suppressed by specifying the compile-time option -Knounroll.

To output line numbers and expansion counts of optimized DO loops, specify -Koptmsg=2 option.

Statements that include array assignments (array expressions) are converted to DO loops, and will be subject to loop unrolling. However, the messages for -Koptmsg=2 are not output.

9.1.1.5 Loop Blocking

Loop blocking is the optimization to subdivide the access for array by multiplexing the loop in block size. As a result, the localization of data access is improved, and then efficient use of the cache is promoted.

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

Loop blocking is performed in optimization level 2 (-O2 option). It can be suppressed by specifying the -Kloop_noblocking option.

The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

An example of loop blocking is given below.

Example:

[Original source-code]

```
DO K=1,NK
  DO J=1,NJ
    DO I=1,NI
      A(I,J)=A(I,J)+B(I,K)*C(K,J)
```

->

[Optimized pseudo-code]

```
DO KK=1,NK,MK+1
  DO JJ=1,NJ,MJ+1
    DO II=1,NI,MI+1
      DO K=KK,MIN(NK,II+MK)
```

```
    ENDDO
  ENDDO
ENDDO
```

```
        DO J=JJ,MIN(NJ,JJ+MJ)
          DO I=II,MIN(NI,II+MI)
            A(I,J)=A(I,J)+B(I,K)*C(K,J)
          ENDDO
        ENDDO
      ENDDO
    ENDDO
  ENDDO
ENDDO
```

9.1.1.6 Software Pipelining

Software pipelining schedules instructions in a loop to execute as parallel as possible.

Software pipelining can be suppressed by specifying the compile-time option -Knoswp.

Software pipelining performs an instruction scheduling to arrange instructions of a particular iteration in a loop with instructions of the following iterations, and reshapes the loop. Thus software pipelining requires enough loop iteration count. Other optimizations affect the required iteration count because it depends on instructions in the loop. If the original loop is short, software pipelining may be less effective because instruction scheduling is performed moderately to keep required iteration count small.

When software pipelining is applied to a loop whose iteration count is variable, as shown in the following example, the compiler inserts a branch instruction to choose the software-pipelined loop or the original loop in case the iteration count is short. It is decided at run time whether the software-pipelined loop is chosen. The size of the object module will increase.

Example:

[Original source-code]

```
SUBROUTINE SUB(N)
INTEGER(KIND=4) :: N
(The original loop: the iteration count is N.)
RETURN
END
```

[Optimized pseudo-code]

```
SUBROUTINE SUB(N)
INTEGER(KIND=4) :: N
IF (Is N enough?) THEN
  (The software-pipelined loop)
ELSE
  (The original loop: the iteration count is N.)
ENDIF
RETURN
END
```

If the compile-time option -Koptmsg=2 is specified, optimization messages about results of software pipelining are output with the line numbers of loops. When software pipelining is applied to a loop, optimization messages jwd8204o-i and jwd8205o-i are output at once. When software pipelining is not applied to a loop, one of optimization messages from jwd8661o-i to jwd8672o-i is output. Optimization messages about software pipelining are not output when the -Knoswp option is specified, or when a loop disappears because of other optimizations before software pipelining, such as full unrolling.

The optimization message jwd8205o-i shows the minimum iteration count required to choose the software-pipelined loop. Make sure to consider following optimizations applied to the same loop because the target loop of software pipelining is the innermost loop; when loop collapse, loop interchange, thread parallelization, or inline expansion is applied, the iteration count of the target loop of software pipelining is changed as shown in the following table. If this is the case, the value shown in the optimization message jwd8205o-i should be compared with the iteration count in the table.

Optimization applied to the same loop	Message number	The iteration count of the target loop to which software pipelining is applied
Loop collapse	jwd8330o-i	Product of the iteration counts of the collapsed loops.

Optimization applied to the same loop	Message number	The iteration count of the target loop to which software pipelining is applied
Loop interchange	jwd8211o-i	The iteration count of the innermost loop after loop interchange.
Thread parallelization	jwd5001p-i, etc.	Quotient of the iteration count divided by the number of threads.
Inline expansion	jwd8101o-i	The iteration count of the innermost loop after inline expansion.

9.1.1.7 SIMD

9.1.1.7.1 Normal SIMD

The compiler may apply optimizations using SIMD extensions. Using SIMD extensions, multiple operations of the same kind are executed at once.

The optimizations using SIMD extensions are suppressed by specifying compiler option -Knosimd. If compiler option -Koptmsg=2 is specified, messages listing the line numbers where optimized loops are placed are output.

9.1.1.7.2 SIMD Extension for Loop Containing IF Construct

When -Ksimd={2|auto} options is specified, SIMD extension can be applied to the loop that contains IF construct. The conditional execution instructions are only store instruction and move instruction in CPU architecture, and other instructions are executed speculatively (statements in the IF construct are executed even if the condition of the IF construct is false). Therefore when the true rate of condition of the IF construct is high, the performance may be improved.

However, side effects, such as exceptions at execution, may occur in the execution results because instructions are executed speculatively.

9.1.1.7.3 List Vector Conversion

When the IF construct in loop contains many instructions and the true rate of the IF construct is low, the performance may be improved by applying list vector conversion.

The list vector conversion generates the following two loops:

1. The loop to save the value of loop control variable into a new array when the condition of IF construct is true.
2. The loop to execute the statement of the original IF construct, and its loop iteration count is equal to the number of the new array element.

In the case of 2, that loop becomes list access method, and becomes the target of SIMD extension and software pipelining.

Example of list vector conversion is shown in the following. The list vector conversion is applied by specifying the optimization control line. See Section "9.10.4 Optimization Control Specifiers" for information on "!OCL SIMD_LISTV[(ALL | THEN | ELSE)]".

Example:

[Original source-code]

```
DO I=1, N
!OCL SIMD_LISTV
  IF (M(I)) THEN
    A(I)=B(I)+C(I)
  ENDIF
ENDDO
```

[Optimized pseudo-code]

```
J=1
DO I=1,N
  IF (M(I)) THEN
    IDX(J)=I
    J=J+1
```

```
  ENDIF
ENDDO
!OCL NORECURRENCE
DO K=1,J-1
  I=IDX(K)
  A(I)=B(I)+C(I)
ENDDO
```

9.1.1.7.4 SIMD with Redundant Executions for the SIMD Width

This function is validated by the optimization control line "!OCL SIMD_REDUNDANT_VL(*n*)". This is applied to loops whose number of iterations is equal to or less than *n*, which is the argument of the SIMD_REDUNDANT_VL specifier. When the number of iterations of the loops is not a multiple of the SIMD width, SIMD instructions with masks are used to generate loops which use SIMD extensions over the whole iterations. (Refer to the following example and figures.) Since instructions with masks are provided only for moves and stores, other instructions are executed redundantly.

The objective is to benefit by using SIMD extensions for loops whose number of iterations is small, while the following optimizations are suppressed because they require plenty of iterations. Thus this function does not always improve performance.

- Software Pipelining
- Loop Striping
- XFILL

Note that the performance of loops such as the followings could decrease. Use this function for programs you understand well.

- Loops whose number of iterations tends to be a multiple of the SIMD width plus 0, 1, or 2
- Loops which use SIMD extensions to execute eight, which is double of the SIMD width, single-precision floating-point operations simultaneously; the loops can be identified by the optimization information "SIMD (VL: 8)" output by means of the -Qt option or the -Nlst=t option.

The followings are an example and images of this function.

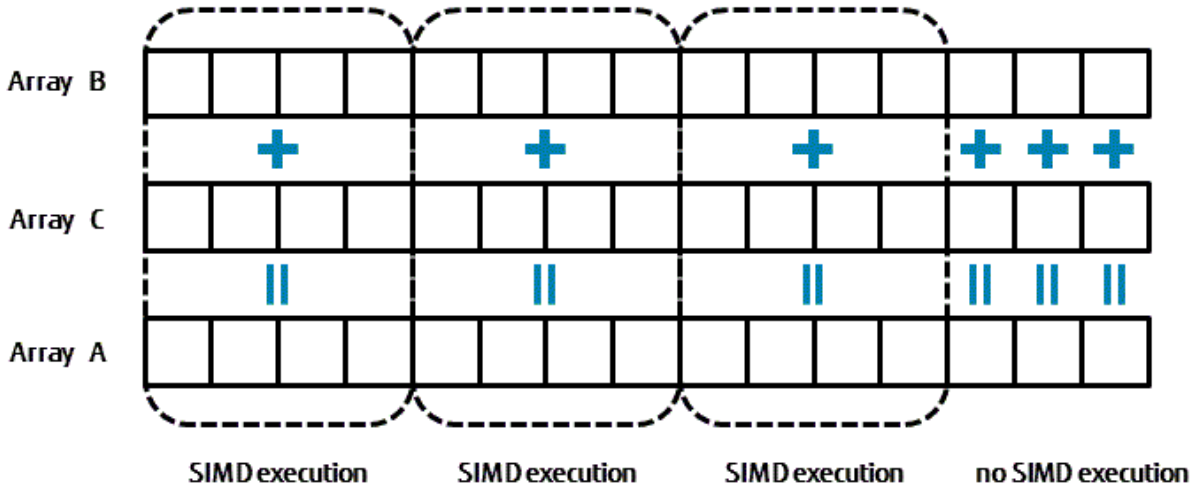
When the function is not applied to the source program in the example, SIMD instructions are used as shown in "Figure 9.1 The optimization control line is not effective (4-wide SIMD with M=15)". The remaining iterations, which correspond the remainder after division of the number of iterations by the SIMD width, do not use SIMD instructions.

When the function is applied, SIMD instructions are used as shown in "Figure 9.2 The optimization control line is effective (4-wide SIMD with M=15 and the SIMD_REDUNDANT_VL(15) specifier)". The remaining iterations are executed by using SIMD instructions with masks.

Example: The optimization control line (OCL) is specified

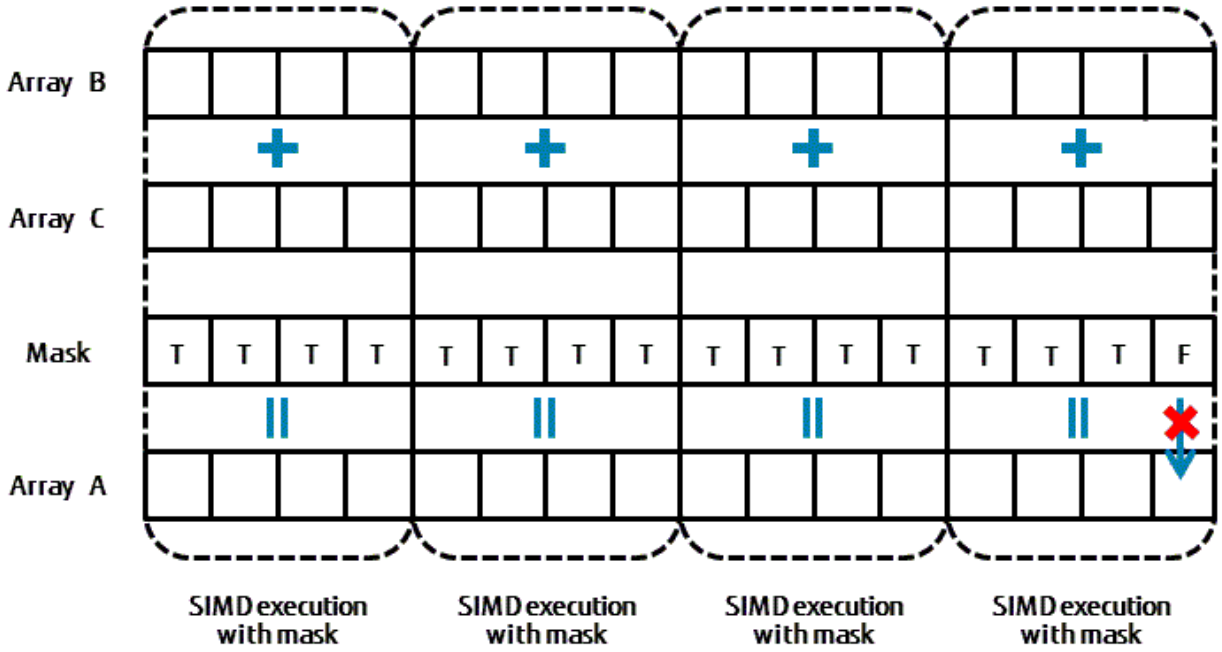
```
!OCL SIMD_REDUNDANT_VL(n)  ! n is a value between 2 and 255
  DO I=1,M
    A(I)=B(I)+C(I)
  END DO
```

Figure 9.1 The optimization control line is not effective (4-wide SIMD with M=15)



[Explanation]
When the loop control variable I is from 13 to 15, SIMD instructions are not used.

Figure 9.2 The optimization control line is effective (4-wide SIMD with M=15 and the SIMD_REDUNDANT_VL(15) specifier)



[Explanation]
The whole iterations are executed by using SIMD instructions with masks.

9.1.1.8 Loop Unswitching

Loop unswitching is an optimization to modify loop by putting out the IF construct out of the loop and creating a loop in each "TRUE" and "FALSE" blocks of the IF construct in order to promote some optimizations such as instruction scheduling if the condition of the IF construct is invariant in a loop.

The loop unswitching is enable by the -O3 option, and disabled by the -O2 option or less.

The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

[Optimized pseudo-code]

```
SUBROUTINE SUB(A,B,C,N)
REAL*8 A(N)
DO I=1,N
  IF (B .EQ. C) THEN
    A(I) = 0.0D0
  ELSE
    A(I) = 1.0D0
  ENDIF
ENDDO
END SUBROUTINE
```

->

```
SUBROUTINE SUB(A,B,C,N)
REAL*8 A(N)
IF (B .EQ. C) THEN
  DO I=1,N
     A(I) = 0.0D0
  ENDDO
ELSE
  DO I=1,N
     A(I) = 1.0D0
  ENDDO
ENDIF
END SUBROUTINE
```

9.1.1.9 Loop Fission

Loop fission is the optimization to split a loop body into multiple small loops. This can improve locality of reference, both of the data being accessed in the loop and the code in the small loop's body. As a result, other optimizations are promoted.

This optimization is performed on loops which contain no jump from the inside to the outside and no jump from the outside to the inside.

Loop fission is performed in optimization level 2 (-O2 option). It can be suppressed by specifying the -Kloop_nofission option.

The message to show the optimized loop can be output by specifying the -Koptmsg=2 option.

An example of loop blocking is given below.

Example:

[Original source-code]

```
DO I=1,N
   E(I) = A1(I)*B1(I) + A2(I)*B2(I) + A3(I)*B3(I) + A4(I)*B4(I) + A5(I)*B5(I)
   F(I) = C1(I)*D1(I) + C2(I)*D2(I) + C3(I)*D3(I) + C4(I)*D4(I) + C5(I)*D5(I)
ENDDO
```

[Optimized pseudo-code]

```
DO I=1,N
   E(I) = A1(I)*B1(I) + A2(I)*B2(I) + A3(I)*B3(I) + A4(I)*B4(I) + A5(I)*B5(I)
ENDDO

DO I=1,N
   F(I) = C1(I)*D1(I) + C2(I)*D2(I) + C3(I)*D3(I) + C4(I)*D4(I) + C5(I)*D5(I)
ENDDO
```

9.1.2 Extended Optimization

When one of the extended function options is specified regardless of optimization, extended optimization is performed in addition to the standard optimization.

This section describes the functions comprising extended optimization.

Depending on the extended optimization function, the size of the object modules (instructions and data areas) generated by the compiler may be greatly enlarged or side effects may occur in the execution results.

9.1.2.1 Inline Expansion

Applying inline expansion, user-defined procedures are inlined to referenced points.

The following expression shows how much size of object modules in program units increases when inline expansion is performed on user-defined procedures.

```
(Size of instructions + Size of data sections) * Expansion count
```

The size of instructions is determined by the kind and number of executable statements.

The size of data sections is the total size of variables, arrays, and constants, excluding common blocks, dummy arguments, allocatable variables, and elements with host associations.

Even if a user-defined procedure is inlined to a program unit more than once, the size of data sections increases only once.

Even if a user-defined external or module procedure is inlined to all referenced points, the procedure is not deleted completely; output into the object file. This is because the procedure may be referenced from other program units. Therefore, internal procedures which are inlined to all referenced points are deleted completely; not output into object file.

If compile-time option -Koptmsg=2 is specified, messages are output listing the expanded places and procedures.

9.1.2.2 Optimization by modifying evaluation methods

When the compiler option -Kpreex is specified, the optimizations which move invariant expressions evaluated in advance are applied. As a result, execution may abort because an instruction which should not be executed based on the logic of the program is executed.

However, this will not affect the precision of calculation results.

The effects of moving the execution location may appear upon referencing an intrinsic function or array element, and upon dividing.

The following example shows the effect of moving the execution location:

Example1: Using intrinsic functions

```
DIMENSION A(100)
...

DO I=1,100
   IF(X.GT.0.0) THEN
     A(I)=ALOG(X)
   END IF
ENDDO
...
```

->

```
DIMENSION A(100)
...
T=ALOG(X)
DO I=1,100
   IF(X.GT.0.0)THEN
     A(I)=T
   END IF
ENDDO
...
```

[Description]

In the program on the left, there are no side effects because the function "alog" is called while the argument is greater than 0.0. In the program on the right, optimization (advance evaluation of invariant expressions) moves the function "alog(x)" out of the loop, and it is therefore always executed. Consequently, an error occurs when 'x' is less or equal than 0.0, and error message indicating that the argument of "alog" is incorrect is output.

Example2: Using array elements

```
SUBROUTINE SUB(A,B,N)
DIMENSION A(N),B(N)


DO I=1,N
   IF(J.LE.N) THEN
     A(I)=B(J)*F
   END IF
ENDDO
...
```

->

```
SUBROUTINE SUB(A,B,N)
DIMENSION A(N),B(N)

T=B(J)*F
DO I=1,N
   IF(J.LE.N) THEN
     A(I)=T
   END IF
ENDDO
...
```

[Description]

In the program on the left, the reference to array "b" will not exceed the boundary, because array element "b(j)" is referenced only when "j" is less than "n". In the program on the right, however, optimization (advance evaluation of invariant expressions) has moved the array "b(j)" out of the loop, and it is therefore always referenced regardless the value of "j". Consequently, "b(j)" may be outside of the array boundary. If an array element outside the program boundary is referenced, the program will be aborted with an error message indicating that a memory protection exception (read) has occurred.

Example3: Division

```
SUBROUTINE SUB(A,N,F)
DIMENSION A(N)


DO I=1,N
   IF(F.NE.0.0) THEN
     A(I)=B/F
   ELSE
     A(I)=0.0
   END IF
ENDDO
...
```

->

```
SUBROUTINE SUB(A,N,F)
DIMENSION A(N)

T=B/F
DO I=1,N
   IF(F.NE.0.0) THEN
     A(I)=T
   ELSE
     A(I)=0.0
   END IF
ENDDO
...
```

[Description]

In the program on the left, division is performed only when the variable "f" is not 0.0, so a division exception will not occur. In the program on the right, however, optimization (advance evaluation of invariant expressions) has moved the division "b/f" out of the loop, and therefore it is always executed regardless the value of "f". This means that the program may be aborted with a division exception error, if the division is performed while the value of "f" is 0.0.

Specifying -NRtrap option at the same time may cause exceptions that would not normally occur.

9.1.2.3 Advance evaluation of invariant expressions

If the compiler option -Keval is specified, the optimization is performed that modifies how operations are evaluated for object programs, which can cause side effects in the calculation results. Operation exceptions, such as exponential overflow, may also occur. Changing division to multiplication is performed on division that uses floating point type variables.

The following example shows the effects of changing division to multiplication:

Example: Changing division-to-Multiplication operator

```
DIMENSION A(100),B(100)
...


DO I=1,100
   A(I)=B(I)/C
ENDDO
...
```

->

```
DIMENSION A(100),B(100)
...

T=1.0/C
DO I=1,100
A(I)=B(I)*T
ENDDO
...
```

[Description]

The variable "c" is not changed in the loop. With optimization (changing division to multiplication), the reciprocal of variable "c" is calculated outside the loop, and the division within the loop is changed into a multiplication of the reciprocal. In other words, a single division operation is calculated from a division and a multiplication. If the result of "t=1.0/c" overflows the precision of the type of variable "c", the lower digit is rounded off, introducing a different result to the one obtained when optimization is not performed. Further, depending on the value of the variable "c", an exponential overflow exception may occur when calculating the reciprocal outside the loop.

9.1.2.4 Loop Striping

Loop striping superimposes an iteration of a loop on next several iterations of the loop. (The number of superimposed loop is called 'striped length'.) It can reduce overheads to iterate the loop, and it can promote software pipelining.

Example:

```
DO I=1,N
  A(I) = B(I) + C(I)
ENDDO
```

If striped length is equal to 4, the -Kstriping=4 option is specified, instructions within a loop are expanded as shown in the following.

Example:

```
DO I=1,N,4
   TMP_B1 = B(I)
   TMP_B2 = B(I+1)
   TMP_B3 = B(I+2)
   TMP_B4 = B(I+3)
   TMP_C1 = C(I)
   TMP_C2 = C(I+1)
   TMP_C3 = C(I+2)
   TMP_C4 = C(I+3)
   TMP_A1 = TMP_B1 + TMP_C1
   TMP_A2 = TMP_B2 + TMP_C2
   TMP_A3 = TMP_B3 + TMP_C3
   TMP_A4 = TMP_B4 + TMP_C4
   A(I) = TMP_A1
   A(I+1) = TMP_A2
   A(I+2) = TMP_A3
   A(I+3) = TMP_A4
ENDDO
```

Loop striping unrolls statements in a loop as with loop unrolling, so the sizes of object modules increase. It also increases compilation time and memory requirement.

Note that performance may deteriorate by applying this optimization because the number of used registers increases.

9.1.2.5 XFILL

XFILL instructions speed up following associated store instructions by securing the cache line.

It have similar characteristics to PREFETCH instructions in that they both secure the cache line for speed up, but differ in that the cache line is used for "writing".

XFILL instructions are generated for stored array data which is in an innermost loop.

However, the compiler does not generate XFILL instructions for array data which is referred, accessed non-sequentially, or stored in IF construct.

When XFILL instructions are generated, prefetch instructions to the second level cache are not generated.

It is prohibited to apply the following optimizations because the loop is modified that data are always stored to the cache lines secured by XFILL instructions. It becomes performance can also decrease.

- Loop unrolling
- Loop striping

Performance may also be reduced under the following conditions:

- When there few iterations
- When the loop iteration count is smaller than the number of elements to be stored in the cache lines specified with -KXFILL=*N*.

However when the loop is modified as shown in Section 9.1.1.7 SIMD, even when XFILL optimization effect, message or optimization information to which the above-mentioned optimizations are executed can be output.

The optimizations to generate XFILL instructions are suppressed by specifying compiler option -KNOXFILL.

Do not specify the -KXFILL option if performance is likely to decrease as a result.

It is usually desirable to control XFILL for each loop.

It is therefore recommended to specify the optimization control line "!OCL XFILL" rather than specifying the -K[NO]XFILL option which effects on the entire program.

9.1.2.6 UXSIMD

UXSIMD(Unloop-eXamination SIMD) is optimization that not only loops but also widely applies SIMD extensions.

Do not depending on loop structure, UXSIMD function directs to choose operation and perform SIMD optimization.

The -Kuxsimd option with the -Ksimd and -KHPC_ACE options enables UXSIMD.

If -Knouxsimd option is set, UXSIMD optimization is not performed.

If -Koptmsg=2 option is specified, the message to show optimized sentence is output.

9.1.2.7 Loop Fission before and after IF constructs in loops(LFI)

"LFI" is optimization that divides the loop into two or more loops when the loop can be divided before and after IF constructs in the loop. The performance might decrease by this optimization.

The improvement of the execution performance is expected in the following cases.

- IF constructs exists in the loop. AND,
- A lot of executable statements exist in the loop and IF constructs.

There is a possibility that the execution performance decreases in the following cases.

- IF constructs exists in the loop and there are few sentences in the loop. OR,
- IF constructs exists in the loop and there are a lot of IF constructs in the loop.

"LFI" perform the optimization when the -Kloop_fission and -Kloop_fission_if option together. If -Kloop_nofission is set, "LFT" optimization is not performed. If -Koptmsg=2 option is specified, the message to show optimized sentence is output.

An example of LFI is given below.

Example:LFI

[Original source-code]

```
REAL,DIMENSION(100) :: A,B,C
INTEGER :: I
DO I=1,100
  A(I) = 1+A(I)
  ...

  IF (COND > 50) THEN
    B(I) = B(I) / 2
    ...
  END IF

  C(I) = C(I)*2
   ...
END DO
```

->

[Optimized pseudo-code]

```
REAL,DIMENSION(100) :: A,B,C
INTEGER :: I1,I2,I3
DO I1=1,100
  A(I1) = 1+A(I1)
  ...
END DO
DO I2=1,100
  IF (COND > 50) THEN
    B(I2) = B(I2) / 2
    ...
  END IF
END DO
DO I3=1,100
  C(I3) = C(I3)*2
  ...
END DO
```

9.1.2.8 Loop Versioning

The loop versioning generates two loops to which optimization is applied and is not applied. The object program created by the compiler judges the data dependency of array at execution time and selects either loop.

The optimization, such as SIMD, software pipelining, or automatic parallelization, is promoted by loop versioning.

The size of the object module and compile time may increase because the loop versioning generates two loops.

The execution performance may decrease because the processing for judgement is overhead.

The loop versioning is applied only to the innermost loop which contains a single array whose data dependency is unknown. The loop versioning may decrease the overhead because of the unknown dependency.

Moreover, the message to show where optimized by the loop versioning can be output by specifying the -Koptmsg=2 option.

The following example shows the loop versioning.

Example:

```
DO I=1,N
  A(I) = A(I+M) + B(I)
ENDDO
```

[Optimized pseudo-code]

```
IF ((1 .GT. N+M) .OR. (N .LT. 1+M)) THEN
!OCL NORECURRENCE
  DO I=1,N                  ! Optimization
    A(I) = A(I+M) + B(I)    !
  ENDDO                     !
ELSE
  DO I=1,N
    A(I) = A(I+M) + B(I)
  ENDDO
ENDIF
```

When the -Kloop_versioning option is specified, the compiler generates two loops with IF-constructs for judging the values of variable "N" and "M" so that the data dependency of the array "A" can be analyzed at execution time.

If the array "A" judges no data dependency, SIMD may be promoted for the loop.

9.1.2.9 CLONE Optimization

CLONE optimization generates conditional branches of variables for loops in order to promote other optimizations, such as full unrolling. CLONE optimization is applied by specifying the optimization control line. The calculation result is not guaranteed if the optimization control line CLONE is used incorrectly. See Section "9.10.4 Optimization Control Specifiers" and "9.18 Incorrectly Specified Optimization Indicators" for more information about CLONE.

Example:

```
!OCL CLONE(N==10)
DO I=1,N
  A(I) = I
ENDDO
```

[Optimized pseudo-code]

```
IF (N==10) THEN
  DO I=1,10
    A(I) = I
  ENDDO
ELSE
  DO I=1,N
    A(I) = I
  ENDDO
ENDIF
```

9.1.2.10 Link Time Optimization

The following describes link time optimization.

9.1.2.10.1 Overview of Link Time Optimization

If the program consists of two or more source files, link time optimization is performed between the files at linking. Link time optimization can be controlled with the -Klto option. To apply link time optimization, -Klto option should be specified at both compilation and linking. Note that link time optimization is not applied when not frtpx command but ld command is used at linking.

Example:

- When the compilation and the linking are specified at a time:

```
$ frtpx -Kfast -Klto sample1.f90 sample2.f90 sample3.f90
```

- When the compilation and the linking are specified at different times:

```
$ frtpx -Kfast -Klto -c sample1.f90
$ frtpx -Kfast -Klto -c sample2.f90
$ frtpx -Kfast -Klto -c sample3.f90
$ frtpx -Klto sample1.o sample2.o sample3.o
```

Link time optimization is performed for the object files which are compiled with the -Klto option. And, only files with an ".o" extension are subject to link time optimization. Therefore, the library such as .so or .a is excluded from link time optimization.

The size of the compiled object files with the -Klto option is increased by the additional information for optimization. However, the additional information for optimization is not included in the executable.

When the -Klto option is specified, the compiler outputs an intermediate file to the temporary directory by both of the compilation and the linking. This file is deleted at the end of compilation. The default temporary directory is /tmp. The temporary directory can be changed by setting the environment variable TMPDIR.

Moreover, link time optimization can be applied to mixed language programming. But, link time optimization is effective only to the language which the compilation command used at linking supports. See "Chapter 11 Mixed Language Programming" for the compilation command used at mixed language programming.

9.1.2.10.2 Notes on Link Time Optimization

The following notes apply when the link time optimization is used

- Link time optimization is a function which optimizes the program again before linking by using the information added to the object files. Therefore, the memory or time which is needed to generate the executable may increase significantly.
- The following functions may be affected when link time optimization is applied.
  - Information collection function at compilation

    The optimization which is actually applied may be different from the following information at compilation:

    - Compilation information which is output by the compiler option -Qp(-Nlst=p) or -Qt(-Nlst=t) (Optimization information)
    - Optimization message which is displayed by the compiler option -Koptmsg.

    See Section "4.1 Compilation Output" for compilation output of Fortran program.
  - Program checks

    The runtime check to the following Fortran program may not be executed:

    - Validity of arguments and result of procedure checks (-Nquickdbg=argchk, -Ha)
    - Subscript range checks (-Nquickdbg=subchk, -Hs)
    - Undefined data reference checks (-Nquickdbg=undef, -Nquickdbg=undefnan, -Hu)

    See Section "8.2.1 Debugging Check Functions" for debugging function of Fortran program.
  - Profiler

    The following information which the profiler outputs may not be guaranteed:

    - Call graph information.

      The function names in call graph information may include the function name which is generated internally by link time optimization.
    - Source code information.

      The each line cost may not be displayed correctly.

See the "Profiler User's Guide" for the profiler.

- Runtime information output function

  The line number may not match the line number of the source program in information which runtime information output function outputs.

  See the "Runtime Information Output Function" for runtime information.

- Trace back map

  The line number of the statement where error occurred may not be correct.

  See Section "4.2.2 Trace Back Map" for trace back map.

9.2 Notes of wrong erroneous program

When optimization is performed on the following erroneous programs, execution results may differ with the optimization level. If results differ, check the following and correct the programs.

- Variables containing undefined values are referenced
- Variables containing values with unacceptable formats are referenced
- Array elements are referenced using index values that exceed array declarations
- There are violations of Fortran syntax

9.3 Effects of Environmental Change on Optimization

As optimization functions operate based on the information from the source program that is converted in the compiler, differences in the version of the compiler, the application of patches, and changes in the system environment may yield different results, even if the same source program and compiler options are being used.

This may cause the following symptoms:

- Information in compile messages and compile log files regarding optimization and automatic parallelization show different compilation results to previous ones.
- The results have some difference within the margin of error.

To know the difference of optimization results, compare compile messages or compile log file to before.

9.4 Effects of Changing the Execution Order

The locations or numbers of the interruption on debugging may differ, because the execution order may be changed by removing a common expression from a loop or moving an invariant expression.

Note that the interruption caused by calculating constant values will occur only at runtime, because the calculations are not performed at compilation.

9.5 Branching Target of Assigned GO TO Statements

The label-lists of assigned GO TO statements must be accurate.

Assigned GO TO statements must not branch to a label that is not in the label-list.

Optimization assumes that each branch is directed to one of the labels in the label-list. Therefore, if there is branching to a label that is not in the list, unexpected results may occur.

If the label-list is omitted, the compiler analyzes the program and creates all possible statement labels as branch targets. Therefore, explicitly defining branching labels will result in better optimization.

9.6 Effect of dummy arguments

Note that using incorrect syntax for dummy arguments will cause unexpected results.

Associate with an actual argument

Each dummy argument in a procedure is optimized as a separate variable. Therefore, updating a dummy variable may also update an actual argument and introduce an incorrect result. Even if the variables have different actual arguments, the result may be incorrect if the memory is shared. According to the syntax, it is prohibited to redefine a value for a dummy argument both directly and indirectly in a subprogram.

The following example shows the effects of associating a dummy argument with an actual argument.

Example: Effect of connection with the same actual argument

Main program

```
PROGRAM MAIN
EQUIVALENCE(Q,R)
 :
CALL SUB(X,X)
CALL SUB(Q,R)
 :
END PROGRAM
```

Subroutine subprogram

```
SUBROUTINE SUB(Y,Z)
DIMENSION C(100)
 :
DO I=1,100
 :
 Y=A/I          <-- This statement is equivalent to Z=A/I.
 :
 C(I)=Z+10.0
 :
END DO
 :
END SUBROUTINE
```

[Description]

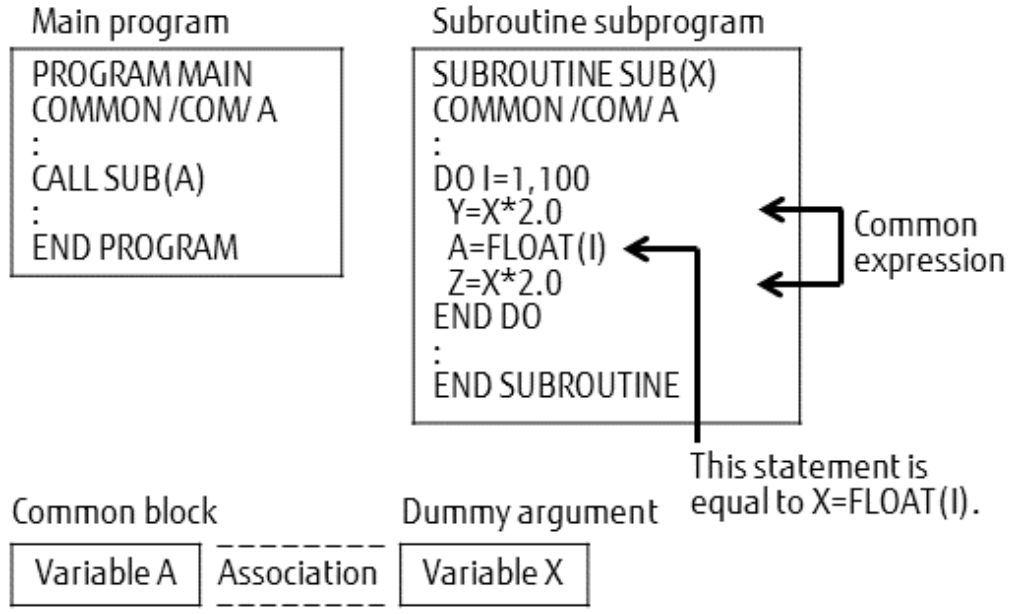
Dummy arguments "Y" and "Z" are optimized as different variables. Therefore, the optimization assumes that the value of "Z" in the DO loop is unchanged, and moves "Z+10.0" to outside the loop (moving an invariant expression). As a result, the results of the "Y=A/I" operation are not seen in the results of the "Z+10.0" operation. Further, even if the value of "Z" does change in the DO loop (Z+10.0 is not moved), the results of the "Y=A/I" operation are not seen in the results of the Z+10.0 operation if the variables "Y" and "Z" are allocated to a register in the DO loop. This is because different registers are allocated to variable "Y" and variable "Z".

Linkage with data in a common block

Dummy arguments and variables in a common block are optimized under the assumption that they are not associated (that is, the user has not made grammar errors). For this reason, associating dummy arguments with variables in a common block will introduce an error.

The following example shows the effects of associating a dummy argument with data in a common block.

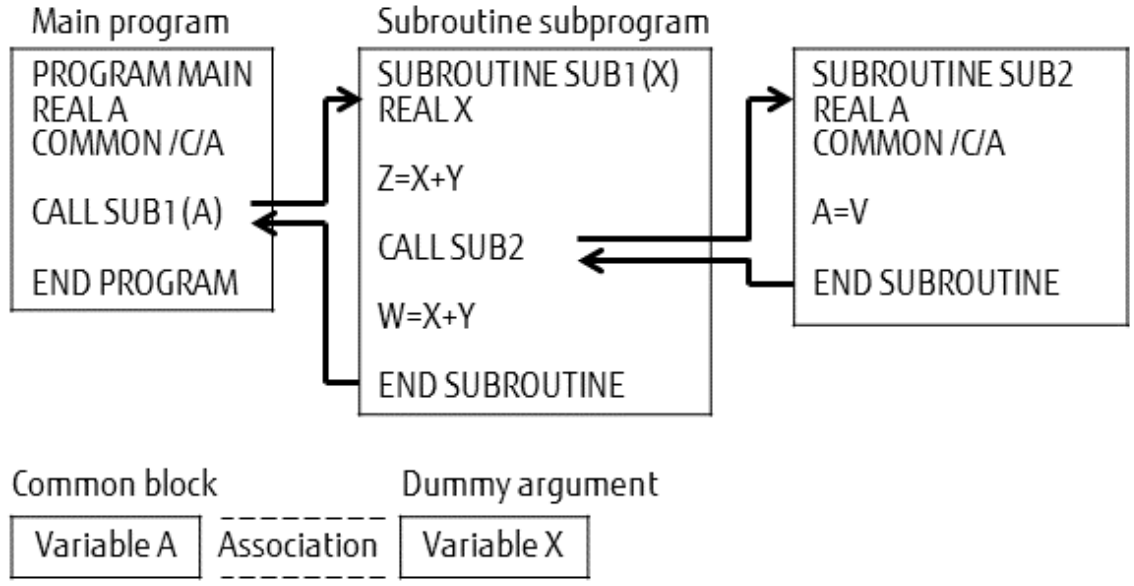
Example 1: Effect of linking data in a common block



[Description]

The dummy-data "X" and the variable "A" in the common block are optimized as independent variables. For this reason, the optimization assumes that the "X*2.0" in the DO loop is a common expression, and replaces "Z=X*2.0" with Z=Y" (removing common expression). As a result, the value "Z" will not include the result of the expression "A=FLOAT(I)". The syntax prohibits assigning a value by associating both a dummy argument and a common block with one instance at the same time. Only the dummy arguments or the common block can be associated.

Example 2: Effect of associating a dummy argument with data in a common block



[Description]

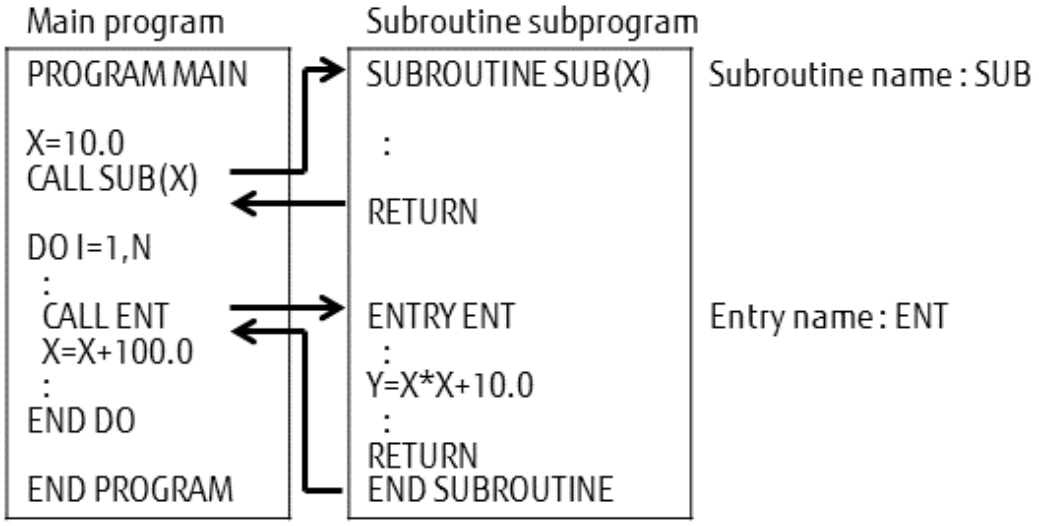
The value of the instance "A" in the common block is changed by "SUB2" called by "SUB1". At the same time, dummy argument "X" associated with common block object "A" is also changed. However, the compiler cannot identify the changes and the expression "X +Y" before and after calling "SUB2" is subject to optimization (removing common expression). As a result, the "W=X+Y" after calling "SUB2" is replaced with "W=Z", and the "W" will no longer include the change in "X" made by "SUB2".

Association using an ENTRY statement

Suppose a subprogram that uses the ENTRY statement is called using an entry name, function name, or subroutine name, and then the dummy arguments and the corresponding actual arguments are associated. Referencing the variable may introduce a problem if the subroutine is called using a different entry name, function name, or subroutine name, without instantiating the dummy argument as an actual argument. This is based on the assumption that the address of the actual argument is passed to the subprogram.

The following example shows the effect of the association using an entry name.

Example: Effect of association using a subentry



Explanation:

To be corrected, the call must be changed to

```
CALL ENT(X)
```

and the entry must be changed to

```
ENTRY ENT(X)
```

9.7 Restrictions on Inline Expansion

As there are various restrictions, such as correspondence between actual and dummy variables, user-defined procedures selected for inline expansion are not always inline expanded. The procedures that can be inline expanded are external procedures, internal procedures, and module procedures.

The following describes the restrictions on inline expansion.

9.7.1 Restrictions on Called User-Defined Procedures

a. Excessive number of instructions

   If the total number of instructions for a user-defined procedure exceeds the allowed maximum, the user-defined procedure is not inline expanded.
   The value of the allowed maximum can be changed with the parameter of the compile-time option -x. The default maximum value is determined by the compiler.

b. If the size of an array exceeds the allowed limit

   The user-defined procedure will not be inline expanded when the size of the array exceeds the maximum, excluding the common block of the user-defined procedure, dummy argument, and assigned variable (that is, the arrays that are allocated as static and local within the user-defined procedure).

The value of the allowed maximum can be changed using the parameter of the compile-time option -x. The default maximum value is infinite.

c. Including a statement that restricts inline expansion

Procedures that include the following statements will not be inline expanded:

- ALLOCATE statement
- DEALLOCATE statement
- ASSIGN statement
- GO TO statement
- NAMELIST statement
- Variable group input-output statement
- SAVE statement
  When there is a SAVE statement that activates a variable, excluding the entirety of common blocks or elements of a common block (that is, variables that are local to the user-defined procedure), the user-defined procedure is not inline expanded.
- RETURN statement
  If a user-defined procedure contains RETURN statement that returns an integer expression, it is not inline expanded.
- NULLIFY statement
- CONTAINS statement

d. When a procedure is subject to restrictions on inline expansion, the following procedures will not be inline expanded:

- Procedures with a VOLATILE attribute
- Functions that return an array, polymorphic, or polymorphic component
- Procedures whose result has a VOLATILE attribute
- Module procedures with a host module that has a PRIVATE variable.
- Module procedures with a host module that has an allocatable variable.
- Procedures using associated allocatable variable
- Module procedures defined by an ENTRY statement
- Procedures that has a procedure language binding specifier
- Procedures that have an ASYNCHRONOUS attribute
- Function result has a deferred length type parameter.
- Function result has a character type and the FUNCTION statement or ENTRY statement has a procedure language binding specifier.

e. When changing a value of a variable or an array that has an initial value.

- A user-defined procedure is not inline expanded if a variable of the user-defined procedure has an initial value defined by a DATA statement or type declaration statement, and the value may be changed directly or indirectly.
- A module procedure is not inline expanded if a variable or array has an initial value defined by a DATA statement or type declaration statement in its host module, and the value may be changed directly or indirectly.

f. An ENTRY statement is included

A subprogram that includes the following ENTRY statements is not inline expanded.

- The dummy parameters, defined with the function name or subroutine name, are different from ones defined on the entry name.

g. The -Knoalias=s option is specified

A procedure is not inline expanded if the procedure of either of the following conditions:

1. The calling and called procedures appear in different program units. And,
   the following either appears in the called procedure:
   - Derived type variable with pointer attribute.
   - Derived type variable contains a pointer component.
2. The calling and called procedures appear in different program units. And,
   the called procedure appears in module program unit. And,
   the following either appears in declaration part of the called module procedure:
   - Derived type variable with pointer attribute.
   - Derived type variable contains a pointer component.
3. The calling and called procedures appear in same program units. And,
   the dummy argument contained in the called procedure has a variable with a pointer attribute in the derived type variable.

9.7.2 Restrictions on the Relationship between Calling and Called Program Units

a. When the actual and dummy arguments are mismatched in the following ways:
   - The number of actual and dummy arguments differs.
   - An actual argument and a dummy argument differ in attribute, type, or kind.

b. When the relationship between actual and dummy arguments is one of the following:
   - A dummy argument is a dummy procedure name, and it corresponds to an actual argument which is an intrinsic function or a procedure pointer
   - A dummy argument is * and the actual argument it corresponds to is a statement label
   - There is an array description (array expression) in the actual argument
   - A dummy argument has the OPTIONAL attribute
   - A dummy argument has the POINTER attribute
   - A dummy argument has the ALLOCATABLE attribute
   - A dummy argument has the VOLATILE attribute
   - A dummy argument is an assumed-shape array
   - A dummy argument has the VALUE attribute
   - A dummy argument has the ASYNCHRONOUS attribute
   - A dummy argument is polymorphic
   - The component of derived type for a dummy argument is polymorphic
   - A dummy argument is of parameterized derived type having assumed length type parameter

c. When there is an error in the way user-defined external procedures are referenced
   - When an external function subprogram is referenced as an external subroutine subprogram
   - When an external subroutine subprogram is referenced as an external function subprogram
   - When an external function subprogram type is referenced with a different type

d. The user-defined external procedure and module procedure corresponds to the following criteria:
   - A procedure specified as a local name by specifying a tentative name.

e. A reference to an elemental procedure.

9.7.3 Restrictions on User-Defined External Procedure Names

When there is a user-defined external procedure with the same name, a program error occurs and inline expansion is not applied:

- There is an external subroutine subprogram with the same name
- There is an external function subprogram with the same name
- There is a block data program unit with the same name
- There are two or more block data program units without names

When the compiler option -Koptmsg=2 is specified, a message is output indicating whether inline expansion is applied.

If inline expansion is not applied, a message is output indicating the reason.

9.7.4 Restrictions on the option

When -H option is specified, inline expansion is not applied.

9.8 Effects of Cray Pointer Variables

A pointer-based variable may be considered to share the memory of the variable with the address returned by the intrinsic function LOC. However, if the pointer contains the address of a common block object, a structure member, or a binding object, it is assumed that the address is shared with other data that belongs to the same common block object, structure, or binding object type.

In addition, assigning an address to a pointer-based variable without using the intrinsic function LOC may introduce an unexpected result.

Extra attention is required when a pointer-based variable is passed as a dummy argument, because the corresponding pointer is not considered to be sharing memory with the following variables:

- Other dummy arguments
- If other dummy arguments are pointer-based variables, the corresponding pointers
- Variables belonging to common blocks

9.9 Effects of Compiler Option -Kcommonpad[=N]

If the compiler option -Kcommonpad is specified during a separate compilation, the compiler option -Kcommonpad must also be specified for source files that include common blocks or using module with the same names. (Refer to example 1)

When compiling with the compiler option -Kcommonpad=*N* specified for multiple files, the value for N must be the same.

In addition, the program may not run properly if the compiler option -Kcommonpad is specified when the element of a common block with the same name is changed and used. (Refer to example 2)

If a source file which includes a common block or declaration part of module is compiled with the -Kcommonpad[=*N*] option, any other source file which includes the same common block or using module must be compiled with the -Kcommonpad[=*N*] option, and value of N must be identical when the source files are compiled separately. (Refer to example 1.)

If common blocks with the same name have different elements, the execution result is unpredictable when the -Kcommonpad[=*N*] option is specified. (Refer to example 2.)

Example1:

a.f

```
COMMON/CMN/A,B
INTEGER,DIMENSION(10) ::A=1,B=2
...
```

b.f

```
COMMON/CMN/A,B
INTEGER,DIMENSION(10) ::A,B
```

```
PRINT *,B
...
```

Both a.f and b.f include the common block named cmn, so the compile-time option -Kcommonpad must be specified for both a.f and b.f.

Example2:

```
SUBROUTINE SUB_A
COMMON/CMN/A,B
REAL(4),DIMENSION(10) ::A=1.0,B=2.0
...
END SUBROUTINE SUB_A
SUBROUTINE SUB_B
COMMON /CMN/X
COMPLEX(4),DIMENSION(10) :: X
PRINT *,X
...
END SUBROUTINE SUB_B
```

As the element of the common block cmn is different in subroutines sub_a and sub_b, operation will be incorrect if the compile-time option -Kcommonpad is specified.

9.10 Using Optimization control line (OCL)

By providing relevant optimization information in the source program, the effectiveness of optimization can be increased. Specifying OCLs in the source program can improve optimizations.

9.10.1 Notation rules for optimization control lines

Optimization control lines consist of lines beginning with the five characters "!OCL " (the fifth character is a space). Lines that begin with '!OCL' are usually treated as comment lines, but when the compiler option -Kocl is specified, they are used as optimization control lines and the operands become effective.

The notation rules are as follows:

```
!OCL i [,i] ....
i: Optimization control specifier
However, one or more blank spaces are necessary between "!OCL" and "i".
```

9.10.2 Description position of the optimization control line

The position where the optimization control line can be inserted depends on the type of optimization indicator.

Optimization control lines are specified in program units: DO loop units, statement units, or array assignment statement units. Program units, DO loop units, statement units, and array assignment statement units are defined as follows:

- Program unit
  The top of each program unit.
- DO Loop unit

  Immediately in front of DO statements, however, blank lines, comment lines, and other optimization control lines (that can be specified within DO-loop unit) can be specified between optimization control lines and the DO statement.
- Statement units
  Immediately in front of statements.
- Array assignment statement units
  Immediately in front of array assignment statements.

9.10.3 Wild Card Specification

In the operand of the following optimization control specifiers, a wild card may be specified for a variable name or a procedure name:

- NORECURRENCE
- NOVREC

See 12.2.3.2.5 Wild Card Specification for more information about Wild Card.

9.10.4 Optimization Control Specifiers

Table 9.1 lists optimization control specifiers that can be included in optimization control lines.

Table 9.1 List of optimization control specifiers that can be included in optimization control lines

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
ARRAY_FUSION[,*opt*] END_ARRAY_FUSION	Fusion of the array assignment statement within the range is promoted. The specified optimization specifier designates to all the array assignment statement within the range.	-	-	-	*
ARRAY_MERGE (*array1*,*array2*[,*array3*]...) or ARRAY_MERGE (*base_array*:*array1*[,*array2*]...)	It specifies to merge the array or the array of few dimensions to the array of many dimensions. *base_array*, *array1*, *array2*, ... are names of arrays	*	-	-	-
ARRAY_SUBSCRIPT (*array1*[,*array2*]...)	It specifies to perform dimension shift of array. *array1*, *array2* ... are names of arrays.	*	-	-	-
CACHE_SECTOR_SIZE (*l1_n1*, *l1_n2*, *l2_n1*, *l2_n2*) END_CACHE_SECTOR_SIZE	It specifies to direct the maximum number of ways of first level cache and second level cache in sectors 0 and 1 of the cache.	*	-	*	-
CACHE_SECTOR_SIZE (*l2_n1*, *l2_n2*) END_CACHE_SECTOR_SIZE	It specifies to direct the maximum number of ways of second level cache in sectors 0 and 1 of the cache.	*	-	*	-
CACHE_SUBSECTOR_ASSIGN (*array1* [,*array2*...]) END_CACHE_SUBSECTOR	It directs to specify an array stored in sector 1 of the cache.	*	-	*	-
CLONE(*var*==*n1*[,*n2*]...)	It directs to generate conditional branches along to the arguments and generate loop copies and put them for the conditional blocks assuming that the *var* is invariant in the loops. The conditional expressions are equals-to expressions which operands are *var* and *n1*[,*n2*]... of specified arguments. Variable *var* is a name of integer variable.	-	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
	Decimal from -9223372036854775808 to 9223372036854775807 or integer named constants can be specified as constant value (*n1*[,*n2*]...).				
EVAL	Instructs the system to perform optimization that changes the expression evaluation sequence.	*	*	-	*
NOEVAL	Cancels the EVAL.	*	*	-	*
FISSION_POINT[(*n*)]	It specifies to instruct to apply loop fission optimization.	-	-	*	-
FLTLD	Cancels the NOFLTLD.	*	*	-	*
NOFLTLD	Instructs the system to perform optimization by using non-faulting load instruction which causes no signal.	*	*	-	*
FP_CONTRACT	Instructs the system to use M&A instruction.	*	*	-	*
NOFP_CONTRACT	Cancels the FP_CONTRACT.	*	*	-	*
FP_RELAXED	The FP_RELAXED specifier instructs to apply optimizations which use reciprocal approximation instructions.	*	*	-	*
NOFP_RELAXED	The NOFP_RELAXED specifier instructs to suppress optimizations which use reciprocal approximation instructions.	*	*	-	*
LOOP_BLOCKING(*n*)	Designates blocking. Decimal from 2 to 10000 can be specified as block size *n*.	*	*	-	*
LOOP_NOBLOCKING	Cancels the LOOP_BLOCKING.	*	*	-	*
LOOP_INTERCHANGE(*var1*,*var2*[,*var3*]...)	Designates that the sequence of a nested DO loop is changed according to the specified order. *var1*, *var2*, *var3*... are induction variables.	-	*	-	-
LOOP_NOINTERCHANGE	The LOOP_NOINTERCHANGE specifier instructs to suppress loop interchange optimizations.	-	*	-	-
LOOP_NOFISSION	The LOOP_NOFISSION suppress loop fission optimization.	*	*	-	*
LOOP_NOFUSION	This directs that fusion of the specified loop and the adjacent loops are not performed.	*	*	-	-
LOOP_PART_SIMD	Performs optimization which divides loops and partially uses SIMD extensions.	*	*	-	*
LOOP_NOPART_SIMD	Suppresses optimization which divides loops and partially uses SIMD extensions.	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
LOOP_VERSIONING	Designates to perform optimization by the loop versioning.	*	*	-	-
LOOP_NOVERSIONING	Cancels the LOOP_VERSIONING.	*	*	-	-
MFUNC[(*level*)]	Designates to change the function to a multi-operation function. Decimal 1, 2 or 3 can be specified as *level*.	*	*	-	*
NOMFUNC	Designates canceling to change the function to a multi-operation function.	*	*	-	*
NF	Optimization using the Non-Faulting mode becomes effective.	*	*	-	*
NONF	Cancels the NF.	*	*	-	*
NOALIAS	Designates that pointer variables and other variable do not share memory area.	*	*	-	*
NOARRAYPAD(*array1*)	Designates canceling to execute padding with array element. *array1* is the name of array.	*	-	-	-
NOVREC[(*array1*[,*array2*]...)]	Designates that arrays of recurrence operation do not exist in loop which can use SIMD instructions. *array1*, *array2*, ... are names of arrays.	*	*	-	*
NORECURRENCE[(*array1*[,*array2*]...)]	Designates that array section references do not extend into loops of plural iterations.	*	*	-	*
PREEX	Instructs the system to perform optimization by evaluating invariant first.	*	*	-	*
NOPREEX	Cancels the PREEX.	*	*	-	*
PREFETCH	Instructs the system to perform the automatic PREFETCH function of the compiler.	*	*	-	-
NOPREFETCH	Cancels the PREFETCH.	*	*	-	-
PREFETCH_CACHE_LEVEL (*c-level*)	Designates cache-level to prefetch of data. 1, 2 or all can be specified as *c-level*.	*	*	-	-
PREFETCH_INFER	Instructs the system to be prefetch by distance is guessed from existing information while be omitted of address calculation for prefetch.	*	*	-	-
PREFETCH_NOINFER	Cancels the PREFETCH_INFER.	*	*	-	-
PREFETCH_ITERATION(*n*)	Instructs the system to be target of a prefetch instruction would be data that is referred to *n* iterations of a loop. Decimal from 1 to 10000 can be specified as *n*.	*	*	-	-
PREFETCH_ITERATION_L2(*n*)	Instructs the system to be target of a prefetch instruction would be data that is defined or is referred to *n* iterations of a	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
	loop. However, this target of prefetch instruction while is prefetch only second cache. Decimal from 1 to 10000 can be specified as *n*.				
PREFETCH_READ (*name*[,level={1|2}] [,strong={0|1}])	Instructs the system to generate the prefetch instruction to the referred data. *name* is the name of array element. level specifies the level of cache to prefetch, and strong specifies to use strong prefetch or not.	-	-	*	-
PREFETCH_WRITE (*name*[,level={1|2}] [,strong={0|1}])	Instructs the system to generate the prefetch instruction to the defined data. *name* is the name of array element. level specifies the level of cache to prefetch, and strong specifies to use strong prefetch or not.	-	-	*	-
PREFETCH_SEQUENTIAL [(AUTO|SOFT)]	This directs that prefetch instructions are generated for array data that is accessed sequentially.	*	*	-	*
PREFETCH_STRONG	This directs that the prefetch instruction for the first level cache is to be the strong prefetch.	*	*	-	*
PREFETCH_NOSTRONG	This directs that the prefetch instruction for the first level cache will not be strong prefetch.	*	*	-	*
PREFETCH_STRONG_L2	This directs that the prefetch instructions for the second level cache are to be strong prefetch.	*	*	-	*
PREFETCH_NOSTRONG_L2	This directs that the prefetch instructions for the second level cache are not to be strong prefetch.	*	*	-	*
SHORTLOOP(*n*)	The SHORTLOOP specifier instructs to apply some optimizations assuming its iteration is few as the specified N.	*	*	-	*
NOSHORTLOOP	The SHORTLOOP specifier instructs to apply some optimizations without assuming its iteration is few.	*	*	-	*
SIMD[(ALIGNED | UNALIGNED)]	The SIMD specifier instructs to use SIMD instructions.	*	*	-	*
NOSIMD	The NOSIMD specifier instructs not to use SIMD instructions.	*	*	-	*
SIMD_LISTV [(ALL | THEN | ELSE)]	This directs that the list vector conversion is performed to the execution statements in IF construct.	-	-	*	-
SIMD_REDUNDANT_VL(*n*)	This directs that SIMD with redundant executions for the SIMD width is applied	*	*	-	*

Optimization control specifier	Outline of meaning	Specifiable optimization control line			
		P	D	S	A
	to loops whose number of iterations is equal to or less than *n*. Decimal from 2 to 255 can be specified as *n*.				
SIMD_NOREDUNDANT_VL	This directs that SIMD with redundant executions for the SIMD width is not applied.	*	*	-	*
STRIPING[(*n*)]	This directs that the striping optimization is performed. The striped length (number of expansions) can be specified for *n*. *n* indicates the stripe-length (number of expansions) which should be from 2 to 100.	*	*	-	*
NOSTRIPING	This directs that the loop striping optimization is suppressed.	*	*	-	*
SWP	Designates software pipelining.	*	*	-	*
NOSWP	Cancels the SWP.	*	*	-	*
UNROLL[(*n*)]	Designates unrolling a corresponding DO loop. Decimal from 2 to 100 can be specified as the times of unrolling *n*.	-	*	-	-
UNROLL('full')	Designates full unrolling.	-	*	-	-
NOUNROLL	Cancels the UNROLL.	-	*	-	-
UNSWITCHING	The UNSWITCHING specifier instructs loop unswitching to IF construct.	-	-	*	-
UXSIMD[(ALIGNED | UNALIGNED)]	The UXSIMD directs to use SIMD extensions by UXSIMD optimization.	*	*	-	*
NOUXSIMD	Cancels the UXSIMD.	*	*	-	*
XFILL[(*n*)]	It is directed to generate XFILL instructions. Decimal from 1 to 100 can be specified as the cache lines *n*.	-	*	-	*
NOXFILL	It is directed not to generate XFILL instructions.	-	*	-	*

P: Program unit

D: DO loop unit

S: Statement unit

A: Array assignment statement unit

*: The optimization control specifier can be specified in the optimization control lines.

-: The optimization control specifier cannot be specified in the optimization control lines.

1. !OCL ARRAY_FUSION[,*opt*]
   !OCL END_ARRAY_FUSION

   When ARRAY_FUSION specifier is put at the beginning of the target range and END ARRAY_FUSION specifier is put at the end of the range, array fusion is promoted in the range.

   Not all array assignment statements within the range are fused even if the optimization control lines are used. Further, *opt* may be specified in the optimization control specifier, and this will be effective in all array assignment statements within the range.
   The following shows an example:

   Example:

```
  REAL,DIMENSION(10) :: A,B,C
  ...
!OCL ARRAY_FUSION       *
  A=B+ ...              *Useful range
  C=C+ ...              *
!OCL END_ARRAY_FUSION   *
  ...
```

2. !OCL ARRAY_MERGE(*array1*,*array2*[,*array3*]...)

   The ARRAY_MERGE specifier instructs to merge arrays. The arrays to be merged are explicit shape arrays.

   See Section 9.15.2 The Restrictions of Merging Local Array Variables for information on restrictions.

   This function will not be applied within specification expressions nor to the constant expressions of declaratives.

   The following shows an example:

   Example1:

```
!OCL ARRAY_MERGE(A,B,C)
  INTEGER,DIMENSION(101,102,103)::A,B,C
  CALL SUB(A)

  PRINT *,B(101,102,103)
  A=C
  A(101,102,103)=C(1,2,3)
```

   When the -Kocl option is specified, the program above is considered as the following program.

```
INTEGER,DIMENSION(3,101,102,103)::TMP
 ! A, in 1 element of the 1st dimension, merge : TMP (1,101,102,103)
 ! B, in 2 element of the 1st dimension, merge : TMP (2,101,102,103)
 ! C, in 3 element of the 1st dimension, merge : TMP (3,101,102,103)
CALL SUB(TMP(1,:,:,:)) ! An array name is changed to the array
                       ! element which has shape.
   ...
PRINT *,TMP(2,101,102,103)
TMP(1,:,:,:) = TMP(3,:,:,:)
TMP(1,101,102,103)=TMP(3,1,2,3)
```

   Note:

   TMP is a variable generated by the compiler.

   Example2:

```
!OCL ARRAY_MERGE(A,B)
  PROGRAM MAIN
  INTEGER,DIMENSION(100,101,102) :: A,B
   ...
  CALL SUB(A)
  END
  SUBROUTINE SUB(C)
  INTEGER,DIMENSION(100) :: C
```

```
  ...
 END
```

When this feature is in effect, the arguments for the procedure SUB specified in MAIN may not be associated with the parameter C(100) in SUB, because a new element is appended in a dimension of A.

3. !OCL ARRAY_MERGE(*base_array*:*array1*[,*array2*]...)

This merges arrays to the base_array. This will affect intrinsic functions, for which the keyword arguments DIM and MASK can be specified.

The arrays are merged into the final dimension of the base variable. The target arrays are explicit shape arrays with bounds that are constant expressions (local arrays and common block objects).

See Section 9.15.2 The Restrictions of Merging Local Array Variables for information on restrictions (except for common-block-object items). This function will not be applied within specification expressions or the constant expressions of declaratives.

The following shows an example:

Example1:

```
!OCL ARRAY_MERGE(A:B,C)
  PROGRAM MAIN
  INTEGER,DIMENSION(10,11,12,2) :: A
  INTEGER,DIMENSION(10,11,12) :: B,C
   ...
  A=1
  B=1
  C(1,2,3)=1
```

The following shows how arrays are merged:

```
PROGRAM MAIN
INTEGER,DIMENSION(10,11,12,4) :: A
A=1
A(:,:,:,3)=1
A(1,2,3,4)=1
```

Example2:

```
!OCL ARRAY_MERGE(A:B,C)
  INTEGER,DIMENSION(1,2,3,4) :: A,X
  INTEGER,DIMENSION(1,2,3) :: B,C
   ...
  X=A   ! An error occurs during compilation.
  A=RESHAPE((/(i,i=1,24)/),(/1,2,3,4/))
   !The calculation result is not guaranteed because of expression
   ! shape difference from the right side to the left.
  PRINT *,UBOUND(A,4)
   ! 6 is output.
```

4. !OCL ARRAY_SUBSCRIPT(*array1*[,*array2*]...)

The ARRAY_SUBSCRIPT specifier instructs to perform dimension shift of arrays. The target array variables are allocatable arrays and explicit shape arrays with bounds that are constant expressions (local arrays, common block objects and dummy arguments).

See Section 9.14.2 Restrictions of the Target Variable of the Dimension Shift of the Array Declaration for information on restrictions (except the restriction for combination substance). This function will not be applied within specification expressions or the constant expressions of declaratives.

The following shows an example:

Example1:

```
!OCL ARRAY_SUBSCRIPT(A)
  PROGRAM MAIN
  INTEGER,DIMENSION(101,102,103,5)::A
  CALL SUB(A)
```

```
  ...
PRINT *,A(1,2,3,4)
```

If this type of program is compiled with the -Kocl option specified, the compiler will treat it as one of the following types of program:

```
PROGRAM MAIN
INTEGER,DIMENSION(5,101,102,103)::A
CALL SUB(A)
...
PRINT *,A(4,1,2,3)
```

Example2:

```
!OCL ARRAY_SUBSCRIPT(A)
  PROGRAM MAIN
  INTEGER,DIMENSION(100,101,102,3) :: A
    ...
  CALL SUB(A)
  END
  SUBROUTINE SUB(B)
  INTEGER,DIMENSION(100) :: B
    ...
  END
```

In this example, the argument A(3,100,101,102) for SUB specified in MAIN will be merged with the parameter B(100) defined in SUB. However, the result may be incorrect because the dimension of the array is not identical.

Example3:

```
!OCL ARRAY_SUBSCRIPT(A)
  PROGRAM MAIN
  INTEGER,DIMENSION(1,2,3,4) :: A,B
    ...
  B=A   ! An error occurs during compilation.
  A=RESHAPE((/(i,i=1,24)/),(/1,2,3,4/))
   !The calculation result is not guaranteed because of expression shape
   ! difference from the right side to the left.
  PRINT *,UBOUND(A,4)
   !3 is output.
```

5. !OCL CACHE_SECTOR_SIZE(*l1_n1*, *l1_n2*, *l2_n1*, *l2_n2*)

   !OCL END_CACHE_SECTOR_SIZE

   !OCL CACHE_SECTOR_SIZE(*l2_n1*, *l2_n2*)

   !OCL END_CACHE_SECTOR_SIZE

   !OCL CACHE_SUBSECTOR_ASSIGN(*array1* [,*array2*...])

   !OCL END_CACHE_SUBSECTOR

   See Section 9.17.2.1 Software control with optimization control rows for details.

6. !OCL CLONE(*var*==*n1*[,*n2*]...)

   The CLONE specifier instructs to generate conditional branches along to the arguments and generate loop copies and put them for the conditional blocks assuming that the *var* is invariant in the loops. The order of the generated branches is the order of the corresponding arguments. This specifier promotes other optimizations, such as full unrolling. The execution result is not guaranteed in case that the value of variable *var* changes in the loop. For more details, refer to "9.18 Incorrectly Specified Optimization Indicators".

   The size of the object module and compile time may increase because this specifier makes copies of loops. This specifier is effective only when the -O3 option is set.

   Variable *var* is a name of integer variable whose kind value is 1, 2, 4, or 8.

You cannot specify following integer variable.

- Structure component
- Array, Array elements, or Substrings
- Pointer variables
- Allocatable variables
- Threadprivate variables

And, you cannot specify integer variable with following attributes.

- ALLOCATABLE
- CHANGEENTRY
- CONTIGUOUS
- DIMENSION
- EXTERNAL
- INTRINSIC
- PARAMETER
- POINTER

Decimal from -9223372036854775808 to 9223372036854775807 or integer named constants can be specified as constant values (*n1*[,*n2*]...). Do not specify any kind type parameter with an underscore '_' regardless of values.

You can use comma to enumerate constant values and colon to specify a range. The following two specifications are equivalent.

```
!OCL CLONE(var==1,3:5,7)
```

```
!OCL CLONE(var==1,3,4,5,7)
```

You can specify up to 20 values at once. When more than 20 values are specified, excess values are ignored. The following specification is limited in the number of values because it specifies 30 values.

```
!OCL CLONE(var==11:40)
```

The above specification is treated as the same as the following specification because excess values are ignored.

```
!OCL CLONE(var==11:30)
```

The following shows examples:

Example 1:

```
SUBROUTINE SUB(A,N)
...
!OCL CLONE(N==10,20)
  DO I=1,N
      A(I) = I
  ENDDO
...
END SUBROUTINE
```

->

```
SUBROUTINE SUB(A,N)
...
IF (N==10) THEN
  DO I=1,10
      A(I) = I
  ENDDO
ELSE IF (N==20) THEN
  DO I=1,20
      A(I) = I
  ENDDO
ELSE
  DO I=1,N
      A(I) = I
  ENDDO
ENDIF
...
END SUBROUTINE
```

Loop is copied with IF constructs in the order of the specification.

Example 2:

```
SUBROUTINE SUB(A,N,M)
...
!OCL CLONE(N==10)
!OCL CLONE(M==20)
  DO I=1,N
      A(I) = M
  ENDDO
...
END SUBROUTINE
```

->

```
SUBROUTINE SUB(A,N,M)
...
IF (N==10) THEN
  DO I=1,10
      A(I) = M
  ENDDO
ELSE IF (M==20) THEN
  DO I=1,N
      A(I) = 20
  ENDDO
ELSE
  DO I=1,N
      A(I) = M
  ENDDO
ENDIF
...
END SUBROUTINE
```

Loop is copied with IF constructs in the order of the multiple specifications.

Example 3:

```
SUBROUTINE SUB(A,N,M)
...
!OCL CLONE(M==10)
DO J=1,M
!OCL CLONE(N==20)
  DO I=1,N
      A(I,J) = 0
  ENDDO
ENDDO
...
END SUBROUTINE
```

->

```
SUBROUTINE SUB(A,N,M)
...
IF(M==10) THEN
  DO J=1,10
    IF (N==20) THEN
      DO I=1,20
        A(I,J) = 0
      ENDDO
    ELSE
      DO I=1,N
        A(I,J) = 0
      ENDDO
    ENDIF
  ENDDO
ELSE
  DO J=1,M
    IF (N==20) THEN
      DO I=1,20
        A(I,J) = 0
      ENDDO
    ELSE
      DO I=1,N
        A(I,J) = 0
      ENDDO
    ENDIF
  ENDDO
ENDIF
...
END SUBROUTINE
```

CLONE specifiers for nested loops generate nested IF constructs.

7. !OCL EVAL

The EVAL specifier instructs to perform the optimization that changes the operation evaluation type.
The following shows an example:

Example:

```
!OCL EVAL
  DO I=1,N
    A(I)=A(I)+B(I)+C(I)+D(I)
  ENDDO
```

->

```
DO I=1,N
  A(I)=(A(I)+B(I))+(C(I)+ D(I))
ENDDO
```

Optimization that changes the operation evaluation type in the target loop is performed.

8. !OCL NOEVAL

   The NOEVAL specifier instructs to suppress the optimization that changes the operation evaluation type.

   The following shows an example:

   Example:

```
!OCL NOEVAL
  DO I=1,N
    A(I) = A(I) + B(I) + C(I) + D(I)
  ENDDO
```

   Optimization that changes the operation evaluation type in the target loop is not performed.

9. !OCL FISSION_POINT[(*n*)]

   The FISSION_POINT(*n*) specifier instructs to apply loop fission optimization.

   The multiloop which is (*n*-1)th-parent from innermost loop is distributed.

   - It is possible to specify "!OCL FISSION_POINT(*n*)" up to 30 lines in a loop, and these OCLs must have same *n* value.

   An example is shown in the following.

   Example:

```
  DO I=2,N
    DO J=2,N
      A(I,J) = A(I-1,J-1)
!OCL FISSION_POINT(1)
      A(I,J) = A(I,J) + A(I-1,J)
    ENDDO
  ENDDO
```

->

```
  DO I=2,N
    DO J=2,N
      A(I,J) = A(I-1,J-1)
    ENDDO
    DO J=2,N
      A(I,J) = A(I,J) + A(I-1,J)
    ENDDO
  ENDDO
```

10. !OCL FLTLD

    The FLTLD specifier instructs that optimization using non-faulting load instructions (load where exceptions do not occur) is not performed. This specifier is effective when the -KHPC_ACE option is set.

    Example:

```
!OCL FLTLD
  DO I=1,N
    IF (M(I)) THEN
    A(I) = B(K)
    ENDIF
  ENDDO
```

    Optimization using non-faulting load instructions within the target loop is not performed.

11. !OCL NOFLTLD

    The NOFLTLD specifier instructs that optimization using non-faulting load instructions (load where exceptions do not occur) is performed. This specifier is effective when the -KHPC_ACE option is set.

    Example:

```
!OCL NOFLTLD
  DO I=1,N
    IF (M(I)) THEN
```

```
    A(I) = B(K)
    ENDIF
  ENDDO
```

Optimization using non-faulting load instructions within the target loop is performed.

12. !OCL FP_CONTRACT

The FP_CONTRACT specifier instructs that Floating-Point Multiply-Add/Subtract instructions are to be used. Note that use of these instructions may introduce errors on the same level as rounding errors.

Example:

```
!OCL FP_CONTRACT
  DO I=1,N
    A(I) = A(I) + B(I) * C(I)
  ENDDO
```

13. !OCL NOFP_CONTRACT

The NOFP_CONTRACT specifier instructs that Floating-Point Multiply-Add/Subtract floating point operation instructions are not to be used.

Example:

```
!OCL NOFP_CONTRACT
  DO I=1,N
    A(I) = A(I) + B(I) * C(I)
  ENDDO
```

14. !OCL FP_RELAXED

The FP_RELAXED specifier instructs to perform high-speed reciprocal approximation operations for division or SQRT functions. This applies to single-precision and double-precision floating point calculations. This optimization may introduce rounding errors, compared to when the standard division or SQRT instructions are used.

Example:

```
!OCL FP_RELAXED
  DO I=1,N
    A(I) = SQRT(B(I) / C(I))
  ENDDO
```

15. !OCL NOFP_RELAXED

The NOFP_RELAXED specifier instructs to perform operations using division and SQRT without using reciprocal approximation instructions, for single-precision and double-precision floating point calculations.
An example is shown in the following.

Example:

```
!OCL NOFP_RELAXED
  DO I=1,N
    A(I) = SQRT(B(I) / C(I))
  ENDDO
```

16. !OCL LOOP_BLOCKING(*n*)

The LOOP_BLOCKING specifier instructs to perform the blocking optimization. Specify the block size when blocking in *n*.

The following shows an example:

Example:

```
!OCL LOOP_BLOCKING(80)
  DO J=1,N
    DO I=1,N
      A(I,J) = A(I,J) + B(J,I)
```

->

```
DO JJ=1,N,81
  DO II=1,N,81
    DO J=JJ,MIN(N,JJ+80)
      DO I=II,MIN(N,II+80)
        A(I,J) = A(I,J) + B(J,I)
      ENDDO
```

```
    ENDDO
  ENDDO
```

```
    ENDDO
  ENDDO
ENDDO
```

Loop blocking is performed with a block size of 80.

17. !OCL LOOP_NOBLOCKING

    The LOOP_NOBLOCKING specifier instructs that loop blocking is suppressed.

    The following shows an example:

    Example:

```
!OCL LOOP_NOBLOCKING
  DO J=1,N
    DO I=1,N
       A(I, J) = A(I, J) + B(I, J)
    ENDDO
  ENDDO
```

18. !OCL LOOP_INTERCHANGE(*var1*,*var2*[,*var3*]...)

    The LOOP_NOINTERCHANGE specifier instructs not to apply loop interchange. This allows DO loops to be interchanged in a specified order. However, loop interchange is not performed when interchanging is not possible because it may introduce different result.

    Example:

```
!OCL LOOP_INTERCHANGE(I,J)
  DO I=1,M
    DO J=1,N
      A(I,J) = A(I,J) + B(J,I)
    ENDDO
  ENDDO
```

->

```
DO J=1,N
  DO I=1,M
    A(I,J) = A(I,J) + B(J,I)
  ENDDO
ENDDO
```

    When LOOP_INTERCHANGE is not specified, parallelization occurs using DO-variable I, but with LOOP_INTERCHANGE, loops are interchanged and parallelization occurs using DO-variable J. When the variable N is sufficiently larger than the variable M, the parallelization becomes more effective.

19. !OCL LOOP_NOINTERCHANGE

    The LOOP_NOINTERCHANGE specifier instructs that nested DO loops interchange is not performed.

    Example:

```
!OCL LOOP_NOINTERCHANGE
  DO I=1,M
     DO J=1,N
        A(I,J) = A(I,J) + B(J,I)
     ENDDO
  ENDDO
```

20. !OCL LOOP_NOFISSION

    The LOOP_NOFISSION specifier instructs to suppress loop fission optimization.

    An example is shown in the following.

    Example:

```
!OCL LOOP_NOFISSION
  DO I=2,N
     DO J=2,N
        A(I,J) = A(I-1,J-1)
        A(I,J) = A(I,J) + A(I-1,J)
```

```
    ENDDO
  ENDDO
```

21. !OCL LOOP_NOFUSION

    The LOOP_NOFUSION specifier instructs that fusion of the specified loop and the adjacent loops are not performed.
    An example is shown in the following.

    Example:

```
!OCL LOOP_NOFUSION
  DO I=1,N
      statement
  ENDDO
  DO J=1,N
      statement
  ENDDO
  DO K=1,N
      statement
  ENDDO
```

    Fusion of the loop I and J are not performed. Fusion of the loop J and K are performed.

22. !OCL LOOP_PART_SIMD

    The LOOP_PART_SIMD specifier instructs to perform optimization which divides loops and partially uses SIMD extensions.

    An example is shown in the following.

    Example:

```
!OCL LOOP_PART_SIMD
  DO I=2,N
    A(I) = A(I)-B(I)+LOG(C(I))  ! SIMD extensions can be applied to
                                ! this statement
    D(I) = D(I-1)+A(I)          ! SIMD extensions cannot be applied to
                                ! this statement
  ENDDO
```

    [Optimized pseudo-code]

```
!OCL LOOP_PART_SIMD
  DO I=2,N
    A(I) = A(I)-B(I)+LOG(C(I))  ! SIMD execution
  ENDDO
  DO II=2,N
    D(II) = D(II-1)+A(II)       ! no SIMD execution
  ENDDO
```

    The loop is divided and SIMD extensions are applied to the part of the divided loop.

23. !OCL LOOP_NOPART_SIMD

    The LOOP_NOPART_SIMD specifier instructs to suppress optimization which divides loops and partially uses SIMD extensions.

    An example is shown in the following.

    Example:

```
!OCL LOOP_NOPART_SIMD
  DO I=2,N
    A(I) = A(I)-B(I)+LOG(C(I))
    D(I) = D(I-1)+A(I)
  ENDDO
```

    Neither loop dividing nor using SIMD extensions is applied to the loop.

24. !OCL LOOP_VERSIONING

    The LOOP_VERSIONING specifier instructs to perform optimization by the loop versioning.

The loop versioning is applied to only an innermost loop, and also the loop contains only one array with unknown data dependency.

The following shows an example:

Example:

```
  DO I=1,N
!OCL LOOP_VERSIONING
    DO J=1,N
      A(J) = A(J+M) + B(J,I)
    ENDDO
  ENDDO
```

The data dependency of array "A" is unknown at compile time because values of variable "N" and "M" are unknown at compile time.

When LOOP_VERSIONING is specified, the compiler generates two loops with IF-constructs for judging the values of variable "N" and "M" so that the data dependency of the array "A" can be analyzed at execution time.

If the array "A" judges no data dependency, SIMD may be promoted for the loop.

25. !OCL LOOP_NOVERSIONING

The LOOP_NOVERSIONING specifier instructs that loop versioning is suppressed.

The LOOP_NOVERSIONING specifier is effective only for the innermost loop.

The following shows an example:

Example:

```
  DO I=1,N
!OCL LOOP_NOVERSIONING
    DO J=1,N
      A(J) = A(J+M) + B(J,I)
    ENDDO
  ENDDO
```

26. !OCL MFUNC[(*level*)]

The MFUNC specifier instructs to perform optimization by converting intrinsic functions within DO loops, array assignment statement intrinsic functions, and operations within loops to multi-operation functions. Specify 1, 2, or 3 for level. Refer to the compiler option -Kmfunc for more information on level.
An example is shown in the following.

Example: When specified for loop

```
  REAL(KIND=8),DIMENSION(10) :: A,B,C,D,E,F,G
!OCL MFUNC
  DO I=1,10
    A(I) = LOG(B(I))
    E(I) = LOG(F(I))
  ENDDO
  D = LOG(C) + LOG(G)
  ...
```

LOG(B(I)) and LOG(F(I)) are converted into multi-operation functions.

27. !OCL NOMFUNC

The NOMFUNC specifier instructs that optimization using multi-operation functions is not performed.
An example is shown in the following.

Example:

```
  REAL(KIND=8),DIMENSION(10) :: A,B,C,D,E,F,G
!OCL NOMFUNC
  DO I=1,10
    A(I) = LOG(B(I))
    E(I) = LOG(F(I))
```

```
  ENDDO
  D = LOG(C) + LOG(G)
  ...
```

LOG(B(I)) and LOG(F(I)) are not converted into multi-operation functions.

28. !OCL NF

The NF specifier instructs that optimization using the Non-Faulting mode is performed. This specifier is effective when the -KHPC_ACE2 option is set.
An example is shown in the following.

Example:

```
!OCL NF
  DO I=1,N
    IF (M(I).NE.0) THEN
      A(I) = B(K)
    ENDIF
  ENDDO
```

29. !OCL NONF

The NONF specifier instructs that optimization using the Non-Faulting mode is not performed. This specifier is effective when the -KHPC_ACE2 option is set.
An example is shown in the following.

Example:

```
!OCL NONF
  DO I=1,N
    IF (M(I).NE.0) THEN
      A(I) = B(K)
    ENDIF
  ENDDO
```

30. !OCL NOALIAS

The NOALIAS specifier informs the compiler that no pointer variables share an address.
The compiler can identify that the pointer variable will not share memory area with other pointers during the compilation, although the content of the pointer is determined only at runtime. This promotes optimizations on the pointer variable. Note that this specifier may not promote the optimization when the association status of the pointer variable is changed in the loop.
An example is shown in the following.

Example:

```
  REAL, DIMENSION(100), TARGET :: X
  REAL, DIMENSION(:), POINTER :: A,B
  A=>X(1:10)
  B=>X(11:20)
!OCL NOALIAS
  DO I=1,10
    B(I) = A(I) + 1.0
  ENDDO
  END
```

Note:

If there are two or more pointer variables referring the same memory area in a loop with the NOALIAS specifier, the execution result may be incorrect.

31. !OCL NOARRAYPAD(*array1*)

The NOARRAYPAD specifier instructs that padding is not performed for the specified array elements. This is valid when the compiler option

-Karraypad_const[=*N*] or -Karraypad_expr=*N* is set.
An example is shown in the following.

Example:

```
!OCL NOARRAYPAD(A)
  PROGRAM MAIN
  INTEGER,DIMENSION(100,100) :: A
  COMMON /X/A
  PRINT *,UBOUND(A)
  PRINT *,SIZE(A,1),SIZE(A,2)
  CALL SUB
  END PROGRAM
!OCL NOARRAYPAD(A)
  SUBROUTINE SUB
  INTEGER,DIMENSION(100,100) :: A
  COMMON /X/A
  PRINT *,UBOUND(A)
  PRINT *,SIZE(A,1),SIZE(A,2)
  END SUBROUTINE
```

Padding will not be performed to the array "A".

32. !OCL NOVREC[(*array1*[,*array2*]...)]

The NOVREC specifier is used to prescribe that arrays of recurrence operation do not exist in loop. The arrays in loop are used SIMD instructions. But, the arrays in loop is not used SIMD instructions by kind of operation or loop structure.

The pointer array cannot be specified in *array*.

Example1:

```
  REAL A(20),B(20)
!OCL NOVREC
  DO I=2,10
    A(I) = A(I+N) + 1
    B(I) = B(I+M) + 2
  ENDDO
```

The NOVREC specifier is used to prescribe that all arrays in loop is not recurrence operation. The operation of arrays A and B uses SIMD instructions.

Example2:

```
  REAL A(20),B(20)
!OCL NOVREC(A)
  DO I=2,10
    A(I) = A(I+N) + 1
    B(I) = B(I) + 2
  ENDDO
```

The NOVREC specifier is used to prescribe that array A whose data dependency is unknown is not recurrence operation. The operation of arrays A and B uses SIMD instructions.

Refer to Section 9.18.2 Example of NOVREC specifier for invalid usages of NOVREC specifier

33. !OCL NORECURRENCE[(*array1*[,*array2*]...)]

The NORECURRENCE specifier instructs that each element of the arrays which are specified by this OCL is defined and referenced within the limits of iteration.

This enables some optimizations to be performed for the DO loop because the order of definitions and references become certain.

For example, the following optimizations are applied:

- Loop Slicing(Automatic Parallelization)
- SIMD

- Software Pipelining

Here, '*array*' is an array, for which loop slicing can be performed. Wildcards can be specified in '*array*'. When no array name is specified, this specifier is applied to all of the arrays within the target range. See 12.2.3.2.5 Wild Card Specification for information on specifying wildcards.

In "Example: Code without NORECURRENCE specifier", the compiler cannot determine whether loop slicing can be performed because the index of array-A is another array element L(I). For this reason, loop slicing is not performed for this DO loop.

Example: Code without NORECURRENCE specifier

```
REAL,DIMENSION(10000) :: A,B
INTEGER,DIMENSION(10000) :: L
DO I=1,10000
   A(L(I)) = A(L(I)) + B(I)
ENDDO
END
```

If it is certain that loop slicing will not affect array-A, specify NORECURRENCE as in "Example: Usage of NORECURRENCE specifier" so that DO loop is sliced.

Example: Usage of NORECURRENCE specifier

```
      REAL,DIMENSION(10000) :: A,B
      INTEGER,DIMENSION(10000) :: L
     !OCL NORECURRENCE(A)
p     DO I=1,10000
p       A(L(I)) = A(L(I)) + B(I)
p     ENDDO
      END
```

Note:

Usage of this specifier is the programmer's responsibility.

When an element of the arrays which are specified by the OCL is defined or referenced in plural iterations, undesirable optimizations may be applied. And the results will be unpredictable.

34. !OCL PREEX

The PREEX specifier instructs that optimization via advance evaluation of invariant expressions is performed.
An example is shown in the following.

Example:

```
!OCL PREEX
  DO I=1,N
    IF (M(I)) THEN
      A(I) = A(I) / B(K)
    ENDIF
  ENDDO
```

->

```
  T = 1 / B(K)
  DO I=1,N
    IF (M(I)) THEN
      A(I) = A(I) * T
    ENDIF
  ENDDO
```

Advance evaluation of invariant expressions is performed.

35. !OCL NOPREEX

The NOPREEX specifier instructs to suppress optimizations which evaluate invariant expressions in advance.
An example is shown in the following.

Example:

```
!OCL NOPREEX
  DO I=1,N
    IF (M(I)) THEN
       A(I) = A(I) / B(K)
    ENDIF
  ENDDO
```

Advance evaluation of invariant expressions is not performed.

36. !OCL PREFETCH
    !OCL NOPREFETCH
    !OCL PREFETCH_CACHE_LEVEL(*c-level*)
    !OCL PREFETCH_INFER
    !OCL PREFETCH_NOINFER
    !OCL PREFETCH_ITERATION(*n*)
    !OCL PREFETCH_ITERATION_L2(*n*)

    This instructs to perform the automatic prefetch function of the compiler. The compiler automatically determines the most appropriate position to insert and generates the prefetch instructions.

37. !OCL PREFETCH_READ (*name*[,level={1|2}] [,strong={0|1}])

    The PREFETCH_READ specifier instructs to generate prefetch instructions for referenced data.

    In name, specify the data (array element) referenced in the program. Expressions can also be specified in subscripts. level={1|2} directs which cache level is to have data prefetched. level=1 is default.

    Specify whether to strong prefetch by specifying strong={0|1}. If strong=0 is specified, strong prefetch is not performed. If strong=1 is specified, strong prefetch is performed. The strong=0 is default.

    For data that is referenced and defined, use PREFETCH_WRITE.

38. !OCL PREFETCH_WRITE (*name*[,level={1|2}] [,strong={0|1}])

    The PREFETCH_WRITE specifier instructs to generate prefetch instructions for defined data.

    In name, specify the data (array element) defined in the program. Expressions can also be specified in subscripts. level={1|2} directs which cache level is to have data prefetched. The level=1 is default.

    Specify whether to strong prefetch by specifying strong={0|1}. If strong=0 is specified, strong prefetch is not performed. If strong=1 is specified, strong prefetch is performed. strong=0 is default.

    The following shows an example:

    Example1: To generate prefetch instruction for A(INDX(I+16))

```
  DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)))
    T = T + A(INDX(I))
  ENDDO
```

    Example2: To generate prefetch instruction to the second level cache for A(INDX(I+16))

```
  DO I = 1,N
!OCL PREFETCH_READ(A(INDX(I+16)),level=2)
    T = T + A(INDX(I))
  ENDDO
```

    Moreover, when the automatic PREFETCH function of the compiler and it wants to control it exclusively because of the cache use efficiency, it is possible to achieve it by combining OCL as follows.

    Example3: To generate prefetch instruction using the strong prefetch for A(INDX(I+16))

```
  DO I = 1,N
 !OCL PREFETCH_READ(A(INDX(I+16)),strong=1)
  T = T + A(INDX(I))
  ENDDO
```

    Example4: To generate prefetch instruction for INDX(I+32) and A(INDX(I+16))

```
   DO I = 1,N
!OCL PREFETCH_READ(INDX(I+32))
!OCL PREFETCH_READ(A(INDX(I+16)))
     T = T + A(INDX(I))
   ENDDO
```

When controlling the use of the automatic prefetch provided by the compiler to improve cache utilization, use multiple OCL statements as shown below.

Example5:

```
!OCL NOPREFETCH
  DO I = 1,N,4
!OCL PREFETCH_READ(A(I+16))
!OCL PREFETCH_READ(C(I+16))
!OCL PREFETCH_WRITE(B(I+16))
     B(I) = A(I) + C(I)
     B(I+1) = A(I+1) + C(I+1)
     B(I+2) = A(I+2) + C(I+2)
     B(I+3) = A(I+3) + C(I+3)
   ENDDO
```

39. !OCL PREFETCH_SEQUENTIAL [(AUTO|SOFT)]

The PREFETCH_SEQUENTIAL specifier instructs that prefetch instructions are generated for array data that is accessed sequentially. If neither AUTO nor SOFT is specified, AUTO is set by default.

1. !OCL PREFETCH_SEQUENTIAL(AUTO)

   The PREFETCH_SEQUENTIAL(AUTO) instructs that the compiler automatically selects whether to use hardware-prefetch or to generate prefetch instructions for array data that is accessed sequentially in the loop.

   The following shows an example:

   Example:

   ```
   !OCL PREFETCH_SEQUENTIAL (AUTO)
     DO I=1,N
       A(I) = B(I)
     ENDDO
   ```

   Hardware prefetch is used for the target loop rather than generating prefetch instructions.

2. !OCL PREFETCH_SEQUENTIAL(SOFT)

   The PREFETCH_SEQUENTIAL(SOFT) specifier instructs that the compiler does not use hardware-prefetch, but rather generates prefetch instructions for array data that is accessed sequentially in the loop.

   The following shows an example:

   Example:

   ```
   !OCL PREFETCH_SEQUENTIAL (SOFT)
     DO I=1,N
       A(I) = B(I)
     ENDDO
   ```

   Prefetch instructions are generated for the target loop rather than using hardware-prefetch.

40. !OCL PREFETCH_STRONG

The PREFETCH_STRONG specifier instructs that the prefetch instruction for the first level cache is to be the strong prefetch.

The following shows an example:

Example:

```
!OCL PREFETCH_STRONG
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

Prefetch instruction for the first level cache generated for the target loop will be strong prefetch.

41. !OCL PREFETCH_NOSTRONG

    The PREFETCH_NOSTRONG specifier instructs that the prefetch instruction generated for the first level cache will not be strong prefetch.

    The following shows an example:

    Example:

```
!OCL PREFETCH_NOSTRONG
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

    The prefetch instruction for the first level cache generated for the target loop will not be strong prefetch.

42. !OCL PREFETCH_STRONG_L2

    The PREFETCH_STRONG_L2 specifier instructs that the prefetch instructions generated for the second level cache are to be strong prefetch.

    The following shows an example:

    Example:

```
!OCL PREFETCH_STRONG_L2
  DO I=1,N
    A(I) = B(I)
  ENDDO
```

    The prefetch instruction for the second level cache generated for the target loop will be strong prefetch.

43. !OCL PREFETCH_NOSTRONG_L2

    The PREFETCH_NOSTRONG_L2 specifier instructs that the prefetch instructions generated for the second level cache are not to be strong prefetch.

    The following shows an example:

    Example:

```
!OCL PREFETCH_NOSTRONG_L2
  DO I=1,N
     A(I) = B(I)
  ENDDO
```

    The prefetch instruction for the second level cache generated for the target loop will not be strong prefetch.

44. !OCL SHORTLOOP(*n*)

    The SHORTLOOP specifier instructs to apply optimization for short loops assuming its iteration is few as the specified *n*. See Section -Kshortloop for more information about optimization for short loops. This specifier becomes effective for innermost loop.

    The following shows an example:

    Example:

```
  SUBROUTINE SUB(A,B,C,N)
  REAL(KIND=8),DIMENSION(N) :: A,B,C
  ...
!OCL SHORTLOOP(5)
  DO I=1,N
    A(I) = B(I) *C(I)
  ENDDO
  END SUBROUTINE
```

    Optimization for short loops is performed assuming its iteration is 5 times.

45. !OCL NOSHORTLOOP

    The SHORTLOOP specifier instructs to apply some optimizations without assuming its iteration is few.

The following shows an example:

Example:

```
  SUBROUTINE SUB(A,B,C,N)
  REAL(KIND=8),DIMENSION(N) :: A,B,C
  ...
!OCL NOSHORTLOOP
  DO I=1,N
    A(I) = B(I) *C(I)
  ENDDO
  END SUBROUTINE
```

Some optimizations are performed without assuming its iteration is few.

46. !OCL SIMD[(ALIGNED | UNALIGNED)]

The SIMD specifier instructs to perform SIMD optimization. Depending on the type of operation and the loop structure, however, SIMD optimization may not be performed.

ALIGNED and UNALIGNED are specifiers for controlling SIMD optimization to store floating point data. See Section 9.18.1 Example of SIMD(ALIGNED) specifier for examples of common errors in specifying the SIMD(ALIGNED) specifier.

The following shows an example:

Example:

```
  REAL(KIND=8),DIMENSION(10) :: A,B
!OCL SIMD
  DO I=1,10
    A(I) = A(I) + B(I)
  ENDDO
```

SIMD optimization is performed.

1. !OCL SIMD(ALIGNED)

   This specifier is effective when the -KHPC_ACE option is set.

   Fortran compilers perform distortion of loops to adjust the address boundaries in order to promote SIMD optimization. If SIMD(ALIGNED) has been specified, loops are not modified to adjust address boundaries, and SIMD optimization is performed for store instructions assuming that they are aligned to the following address boundaries:

   - Single-precision floating point stores are aligned to 8-byte boundaries
   - Double-precision floating point stores are aligned to 16-byte boundaries

   See 5.2 Correct Data Boundaries for details about array boundaries the compiler allocates.

   Example:

```
  SUBROUTINE SUB(A,B,N)
  REAL(KIND=8),DIMENSION(N) :: A,B
!OCL SIMD(ALIGNED)
  DO I=1,N
    A(I) = A(I) + B(I)
  ENDDO
  END SUBROUTINE
```

   SIMD optimization is performed. SIMD optimization is also performed on store instructions.

2. !OCL SIMD(UNALIGNED)

   This specifier is effective when the -KHPC_ACE option is set.

   When SIMD(UNALIGNED) has been specified, loops are not modified to adjust address boundaries, and the SIMD optimization is not performed for store instructions with the assumption that store instructions are not aligned to 8 nor 16-byte boundaries.

   The following shows an example:

Example2:

```
  SUBROUTINE SUB(A,B,N)
  REAL(KIND=8),DIMENSION(N) :: A,B
!OCL SIMD(UNALIGNED)
  DO I=1,N
    A(I) = A(I) + B(I)
  ENDDO
  END SUBROUTINE
```

SIMD optimization is performed. SIMD optimization is not performed on store instructions.

47. !OCL NOSIMD

The NOSIMD specifier instructs that SIMD optimization is not performed.

Example:

```
  REAL(KIND=8),DIMENSION(10) :: A,B
!OCL NOSIMD
  DO I=1,10
    A(I) = A(I) + B(I)
  ENDDO
```

SIMD optimization is not performed.

48. !OCL SIMD_LISTV[(ALL | THEN | ELSE)]

The SIMD_LISTV specifier instructs to perform list vector conversion to the execution statements of the specified IF construct. ALL, THEN or ELSE is specifier for controlling the range of list vector conversion optimization. When the -O2 option or higher is set, and the -Ksimd=2 option or the same effect by the SIMD specifier is performed, this specifiers is effective.

See Section "9.1.1.7.3 List Vector Conversion" for the list vector conversion.

1. !OCL SIMD_LISTV(THEN)

   The list vector conversion is performed to the statements in the subsequent block of the specified IF THEN statement.

   The following shows an example:

   Example:

```
!OCL SIMD
  DO I=1,N
!OCL SIMD_LISTV(THEN)
    IF (A(I).EQ.0.0) THEN
      A(I) = A(I)+B(I)
    ELSE
      A(I) = A(I)+C(I)
    ENDIF
  ENDDO
```

2. !OCL SIMD_LISTV(ELSE)

   The list vector conversion is performed to the statements in the subsequent block of the specified ELSE statement.

   The following shows an example:

   Example:

```
!OCL SIMD
  DO I=1,N
!OCL SIMD_LISTV(ELSE)
    IF (A(I).EQ.0.0) THEN
      A(I) = A(I)+B(I)
    ELSE
      A(I) = A(I)+C(I)
    ENDIF
  ENDDO
```

3. !OCL SIMD_LISTV(ALL)

   The list vector conversion is performed to the statements in the blocks of the specified IF THEN statement and ELSE statement.

   The following shows an example:

   Example:

```
!OCL SIMD
  DO I=1,N
!OCL SIMD_LISTV(ALL)
    IF (A(I).EQ.0.0) THEN
      A(I) = A(I)+B(I)
    ELSE
      A(I) = A(I)+C(I)
    ENDIF
  ENDDO
```

49. !OCL SIMD_REDUNDANT_VL(*n*)

    The SIMD_REDUNDANT_VL specifier instructs to validate SIMD with redundant executions for the SIMD width. This is applied to loops whose number of iterations is equal to or less than *n*. When the number of iterations of the loops is not a multiple of the SIMD width, SIMD instructions with masks are used to generate loops which use SIMD extensions over the whole iterations. Decimal from 2 to 255 can be specified as *n*.

    See Section "9.1.1.7.4 SIMD with Redundant Executions for the SIMD Width" for details.

    The following shows an example:

    Example:

```
!OCL SIMD_REDUNDANT_VL(7)
  DO I=1,M
    A(I) = B(I)+C(I)
  END DO
```

50. !OCL SIMD_NOREDUNDANT_VL

    The SIMD_NOREDUNDANT_VL specifier instructs to invalidate SIMD with redundant executions for the SIMD width.

    The following shows an example. This specifier is used to suppress the application to specific loops when the SIMD_REDUNDANT_VL specifier is specified for program units.

    Example:

```
!OCL SIMD_REDUNDANT_VL(7)
SUBROUTINE SUB(A,B,C)
  ...
!OCL SIMD_NOREDUNDANT_VL
  DO I=1,M
    A(I)= B(I)+C(I)
  END DO
  ...
END SUBROUTINE
```

51. !OCL STRIPING[(*n*)]

    The STRIPING specifier instructs that the striping optimization is performed. The striped length (number of expansions) can be specified for *n*.

    A value between 2 and 100 can be specified for *n*. If a value was not specified for *n*, the compiler will automatically determine the value. When the iteration count of a loop in the source program is known, the expansion number determined by the compiler will be used if a number that exceeds the iteration count is specified for *n*.

    Example1:

```
!OCL STRIPING
  DO I=1,N
```

```
    statement
  ENDDO
```

This specifies that loop striping is to be performed on the statement using the striping length determined by the compiler.

Example2:

```
!OCL STRIPING(4)
  DO I=1,N
    statement
  ENDDO
```

This specifies that loop striping is to be performed on the statement using the striping length 4.

Example3:

```
!OCL STRIPING(8)
  DO I=1,4
    statement
  ENDDO
```

The specified stripe length (8) exceeds the loop iteration count (4), so the striped length automatically determined by the compiler will be used.

Example4:

```
!OCL STRIPING(4)
  SUBROUTINE SUB
    ...
    ...
  DO I=1,N
     statement
  ENDDO
  DO I=1,N
     statement
  ENDDO
```

If this is specified at the start of the program unit, all the DO loops in the program are targeted.

52. !OCL NOSTRIPING

The NOSTRIPING specifier instructs that the loop striping optimization is suppressed.

Example:

```
!OCL NOSTRIPING
  DO I=1,N
    statement
  ENDDO
```

Loop striping is not performed.

53. !OCL SWP

The SWP specifier instructs that software pipelining is performed. This specifier (or indicator) is valid only when the -O2 option or above is in effect at compilation.

The following shows an example:

Example:

```
!OCL SWP
  DO I=1,N
    statement
  ENDDO
```

54. !OCL NOSWP

The NOSWP instructs that software pipelining is suppressed.

The following shows an example:

Example:

```
!OCL NOSWP
  DO I=1,N
    statement
  ENDDO
```

55. !OCL UNROLL[(*n*)]

The UNROLL specifier instructs the number of unrolls for loop unrolling at *n*. This is described immediately before a DO construct with a DO variable. *n* is unsigned number from 2 to 100. If a value was not specified for *n*, the compiler will automatically determine the value.

Further, if the iteration count for a loop is known, the known number of iterations is given priority, even if a number is specified in *n* that exceeds the iteration count.

Note that only the DO-loop immediately after the optimization control line is subject to optimization control.

When the optimization control line has not been specified, the compiler will determine the loop unrolling expansion count based on the iteration count, number and type of instructions in the loop, and the data type being used.

The following shows an example:

Example1:

```
SUBROUTINE SUB
...
!OCL UNROLL(8)
  DO I=1,N
    statement
  ENDDO
...
END SUBROUTINE
```

This specifies that the statement is to be unrolled 8 times.

Example2:

```
SUBROUTINE SUB
...
!OCL UNROLL(80)
  DO I=1,50
    statement
  ENDDO
...
END SUBROUTINE
```

The specified expansion count (80) exceeds the loop iteration count (50), so the expansion count will be deemed to be 50.

56. !OCL UNROLL('full')

If 'full' is specified as the loop expansion count, instructions will be expanded as many times as the iteration count in the source program. Optimization is not performed if the iteration count is unknown.

Note that only the DO-loop immediately after the optimization control line is subject to optimization control.

The following shows an example:

Example1:

```
SUBROUTINE SUB
...
!OCL UNROLL('full')
  DO I=1,10
    statement
  ENDDO
```

```
...
END SUBROUTINE
```

This specifies that the statement is to be unrolled 10 times, the same as the loop iteration count.

Example2:

```
SUBROUTINE SUB
...
!OCL UNROLL('full')
  DO K=1,2
!OCL UNROLL('full')
    DO J=1,2
!OCL UNROLL('full')
      DO I=1,2
         statement
      ENDDO
    ENDDO
  ENDDO
...
END SUBROUTINE
```

This specifies that the statement is to be unrolled 2*2*2 times.

57. !OCL NOUNROLL

The NOUNROLL specifier instructs that loop unrolling optimization is suppressed.

Note that only the DO-loop immediately after the optimization control line is subject to optimization control.

The following shows an example:

Example:

```
SUBROUTINE SUB
...
!OCL NOUNROLL
  DO I=1,N
    statement
  ENDDO
...
END SUBROUTINE
```

Loop unrolling is not performed.

58. !OCL UNSWITCHING

The UNSWITCHING specifier instructs loop unswitching to IF construct. This specifier must be described immediately before IF construct which is invariant in a loop. This specifier is not effective in the case of specified other than the above.

The following shows an example:

Example1:

```
  SUBROUTINE SUB(A,B,C,X,N)
  REAL(KIND=8),DIMENSION(N) :: A,B,C
  INTEGER(KIND=8) :: X
  ...
  DO I=1,N
!OCL UNSWITCHING
    IF (X == 0) THEN
      A(I) = B(I)
    ELSE
      A(I) = C(I)
    ENDIF
  ENDDO
  END SUBROUTINE
```

Loop unswitching to IF construct is performed.

Note that IF construct without this specifier is not the target of loop unswitching. The following shows an example:

Example2:

```
  SUBROUTINE SUB(A,B,C,D,X,N)
  REAL(KIND=8),DIMENSION(N) :: A,B,C,D
  INTEGER(KIND=8) :: X
  ...
  DO I=1,N
    IF (X == 0) THEN
      A(I) = B(I)
!OCL UNSWITCHING
    ELSE IF (X == 1) THEN
      A(I) = C(I)
    ELSE
      A(I) = D(I)
    ENDIF
  ENDDO
  END SUBROUTINE
```

Only IF construct with this specifier is the target of loop unswitching.

The memory consumption and the compilation time may increase drastically when loops to which loop unswitching is applied include a lot of execution statements.

59. !OCL UXSIMD[(ALIGNED | UNALIGNED)]

The UXSIMD specifier instructs to SIMD optimization using UXSIMD optimization. However depending of kind of instruction, SIMD optimization is not performed.

ALIGNED and UNALIGNED are specifier for controlling SIMD optimization to store floating point data.

However, this does not direct to perform distortion of loops to adjust the address boundaries in order to promote SIMD optimization.

See "9.1.2.6 UXSIMD" for details about UXSIMD optimization.

The following shows an example:

Example:

```
!OCL UXSIMD
  DO I=1,N,2
    A(I) = B(I) + C(I)
    A(I+1) = B(I+1) + C(I+1)
  ENDDO
```

SIMD optimization using UXSIMD is performed for instruction in the loop.

1. !OCL UXSIMD(ALIGNED)

   This specifier is effective when the -KHPC_ACE option is set.

   If UXSIMD(ALIGNED) has been specified, SIMD optimization is performed for store instructions assuming that they are aligned to the following address boundaries:

   - Single-precision floating point stores are aligned to 8-byte boundaries
   - Double-precision floating point stores are aligned to 16-byte boundaries

   Because SIMD optimization is assumed, the first element of a floating point array is aligned to the following address.

   However arrays passed as arguments, pointer arrays, structure array and common block array have the possibility of not corresponding to these.

   - Single-precision floating point arrays are aligned to 8-byte boundaries
   - Double-precision floating point arrays are aligned to 16-byte boundaries

Example:

```
!OCL UXSIMD(ALIGNED)
  DO I=1,N,2
    A(I) = B(I) + C(I)
    A(I+1) = B(I+1) + C(I+1)
  ENDDO
```

SIMD optimization using UXSIMD is performed for instruction in the loop. SIMD optimization is also performed on store instructions.

2. !OCL UXSIMD(UNALIGNED)

   This specifier is effective when the -KHPC_ACE option is set.

   If UXSIMD(UNALIGNED) has been specified, SIMD optimization is not performed for floating point store instructions.

   Example:

```
!OCL UXSIMD(UNALIGNED)
  DO I=1,N,2
    A(I) = B(I) + C(I)
    A(I+1) = B(I+1) + C(I+1)
  ENDDO
```

   SIMD optimization using UXSIMD is performed for instruction in the loop. SIMD optimization is not performed on store instructions.

60. !OCL NOUXSIMD

The UXSIMD directs not to use SIMD extensions by UXSIMD optimization.

Example:

```
!OCL NOUXSIMD
  DO I=1,N,2
    A(I) = B(I) + C(I)
    A(I+1) = B(I+1) + C(I+1)
  ENDDO
```

UXSIMD optimization is not performed for instruction in the loop.

61. !OCL XFILL[(*n*)]

The XFILL specifier instructs to create an object to store data that will improve the cache utilization. Specify the number of lines to be placed on cache using *n*. A value between 1 and 100 can be specified for *n*. If a value was not specified for *n*, the compiler will automatically determine the value.

For details about XFILL, see "9.1.2.5 XFILL".

The following shows an example:

Example1:

```
SUBROUTINE SUB(A,N)
...
!OCL XFILL
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE
```

This specifies to improve the cache utilization for the data on four lines ahead.

Example2:

```
SUBROUTINE SUB(A,N)
...
```

```
!OCL XFILL(1)
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE
```

This specifies to improve the cache utilization for the data on one line ahead.

62. !OCL NOXFILL

The NOXFILL specifier instructs not to create an object to store data that will improve the cache utilization.

The following shows an example:

Example:

```
SUBROUTINE SUB(A,N)
...
!OCL NOXFILL
  DO I=1,N
    A(I) = 1
  ENDDO
...
END SUBROUTINE
```

This specifies that optimization to improve cache utilization will not be performed.

9.11 Effects of Allocating on Stack

The following shows the effects of allocating stack using the compiler option -Kautoobjstack, -Kauto, -Kthreadsafe, or -Ktemparraystack.

If the maximum stack area size is smaller than the memory required by the program, stack area may not be allocated, and the program may terminate abnormally. In this case, increase the maximum stack area size.

To view and change the maximum, use the limit command on (csh) and the ulimit command on (sh). In addition, when executing as a batch request using Job Operation Software, the maximum size of the corresponding stack area of the batch queue (Per-process stack size limit) must be increased.

The following examples show the effects of specifying each of the options:

Example1: When "-Kautoobjstack" is specified

```
$ cat a.f90
  SUBROUTINE SUB(N)
  REAL(KIND=8),DIMENSION(N) :: A
  A = 1.
  PRINT*, A(N)
  END SUBROUTINE

  CALL SUB(2000000)
  END
$ ulimit -s
8192
$ frtpx a.f90
$ ./a.out
  1.000000000000000
$ frtpx a.f90 -Kautoobjstack
$ ./a.out
  Segmentation Fault(core dumped)
```

Program terminated abnormally due to an insufficient stack area size.

Specify "unlimited" for the "stacksize" and compile again.

```
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ frtpx a.f90 -Kautoobjstack
$ ./a.out
  1.000000000000000
```

The program ended normally because the stack size was increased.

Example2: When "-Kauto" is specified

```
$ cat b.f90
      REAL(KIND=8),DIMENSION(2000000) :: A
      A = 1.
      PRINT*, A(1)
      END
$ ulimit -s
8192
$ frtpx b.f90 ; ./a.out
      1.000000000000000
$ frtpx b.f90 -Kauto ; ./a.out
      Segmentation Fault(core dumped)
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ frtpx b.f90 -Kauto ; ./a.out
      1.000000000000000
```

Example3: -Ktemparraystack" is specified

```
$ cat c.f90
      SUBROUTINE SUB(A, N1, N2, M1, M2)
      REAL(KIND=8),DIMENSION(M2) :: A
      A(N1:N2) = A(M1:M2) + 1.
      PRINT*, A(1)
      END SUBROUTINE
      REAL(KIND=8),DIMENSION(2000000) :: A
      A = 0.

      CALL SUB(A,1,1999999,2,2000000)
      END
$ ulimit -s
8192
$ frtpx c.f90 ; ./a.out
      1.000000000000000
$ frtpx c.f90 -Ktemparraystack ; ./a.out
      Segmentation Fault(core dumped)
$ ulimit -s unlimited
$ ulimit -s
unlimited
$ frtpx c.f90 -Ktemparraystack ; ./a.out
      1.000000000000000
```

9.12 Effects of Applying Optimization to Change Shape of Array

Increasing the maximum array size using the compiler option "-Karraypad_const[=*N*] " or "-Karraypad_expr=*N*" will affect the result of the functions UBOUND and SIZE. Further, if the "fdb" command or "SUBCHK" functionality is used, it will be as if the maximum size has been specified in the source program.

When using the compiler options "-Karraypad_const[=*N*]" and "-Karraypad_expr=*N*", the same options must be specified when compiling all the program units that make up the executable program.

This feature is applied to each individual array, without considering storage association and argument associations that are introduced by the EQUIVALENCE statement and COMMON statement. For this reason, the correct result may not be obtained by using this function so caution should be used.

Example1: -Karraypad_expr=*N*

```
PROGRAM MAIN
REAL,DIMENSION(1024*1024) :: A
CALL SUB(A,1024,1024)
   :
SUBROUTINE SUB(X,M,N)
REAL,DIMENSION(M,N) :: X
   :
```

If this type of program is compiled with the "-Karraypad_expr=1" option, then it will be compiled as one of the following types of programs.

```
PROGRAM MAIN
REAL,DIMENSION(1024*1024+1) :: A
CALL SUB(A,1024,1024)
   :
SUBROUTINE SUB(X,M,N)
REAL,DIMENSION(M+1,N) :: X
   :
```

If this feature is not used, A(1025) in MAIN is associated with X(1,2) in SUB. When it is used, A(1025) is associated with X(1025,1) and will derive a different result.

Example2: -Karraypad_const[=*N*]

```
INTEGER A(3,3),B(3,3)
A(:,1)=B(1,:)
A(:,2)=RESHAPE((/1,2,3/),(/3/))
```

If the "-Karraypad_const=1" option is specified on compile, the array shape on the right and left of the assignment will differ, and the results may be incorrect.

9.13 Effects of generating prefetch on program performance

If the -Kprefetch_sequential, -Kprefetch_stride, -Kprefetch_indirect, or -Kprefetch_conditional option is valid, it may decrease in performance to generate prefetch instructions depending on the cache-efficiency of the loop, the presence or absence of branch instructions, and the complexity of subscripts.

9.14 The Dimension Shift of the Array Declaration

This section explains dimension shift of array declarations.

9.14.1 Effects of the Dimension Shift of Array Declaration

The dimension position of arrays is shifted by specifying the compiler option -Karray_subscript. This will affect intrinsic functions, for which the keyword arguments DIM and MASK can be specified.

When using the compiler options -Karray_subscript, the same options must be specified when compiling all the program units that make up the executable program. The target array variables are allocatable arrays and explicit shape arrays with bounds that are constant expressions (local arrays, common block objects, and dummy arguments). This function will not be applied within specification expressions or the constant expressions of declaratives. This feature is applied to each individual array, without considering argument association. For this reason, the correct result may not be obtained by using this function so caution should be used.

The following shows an example:

Example1:

```
INTEGER,DIMENSION(101,102,103,5) :: A
CALL SUB(A)
  :
PRINT *,A(1,2,3,4)
```

If this type of program is compiled with the "-Karray_subscript" option, then it will be compiled as one of the following types of programs:

```
INTEGER,DIMENSION(5,101,102,103) :: A
CALL SUB(A)
   :
PRINT *,A(4,1,2,3)
```

Example2:

```
PROGRAM MAIN
INTEGER,DIMENSION(100,101,102,3) :: A
   :
CALL SUB(A)
END
SUBROUTINE SUB(B)
INTEGER,DIMENSION(100) :: B
   :
END
```

In this sample, the argument A(3,100,101,102) for SUB specified in MAIN will be merged with the parameter B(100) defined in SUB. However, the result may be incorrect because the dimensions of the arrays are not identical.

Example3:

```
 INTEGER,DIMENSION(1,2,3,4) :: A
      :
 A=RESHAPE((/(i,i=1,24)/),(/1,2,3,4/))

! The right-hand shape of the substitution expression is different
! from the left-hand one, so the execution result may not be correct.
 PRINT *,UBOUND(A,4)
! 3 is output.
```

9.14.2 Restrictions of the Target Variable of the Dimension Shift of the Array Declaration

Arrays that meet any of the following conditions cannot be the target of dimension shift of array declarations:

- Derived type, character type, or polymorphic
- Literal with name
- An array with a SAVE attribute (implicit SAVE attribute in declaration part of module is excluded) or TARGET attribute
- Pointer variable
- An array with an initial value
- Equivalence-object
- Namelist-group-object
- Automatic array
- An array with a PUBLIC attribute declared in the module specification part
- Address holding variable
- Function result

- Declaration in BLOCK

9.15 Merging Array Variables

This section explains merging of array declarations.

9.15.1 Effects of Merging Array Variables

Multiple arrays are merged into one array by specifying the compiler option -Karray_merge.

The target array variables are explicit-shape arrays or automatic object arrays with bounds that are constant expressions, and automatically allocated arrays (Note). This function will not be applied within specification expressions or the constant expressions of declaratives. The compiler automatically adds the element of the first dimension and merges the elements. The merging order is undefined. This may cause a compile error or may introduce an incorrect result, because the array names will be replaced with the merged array names. This feature is applied to each individual array, without considering lower bound of declaration and argument association. For this reason, using this function may cause incorrect results.

(Note)

If the bound expression of automatic object array is the following condition, the automatic object array is not restricted.

1. Bound expression of declaration contain exactly one variable, or
2. Bound expression of declaration contain is constant expression.

The following shows an example:

Example1:

```
INTEGER,DIMENSION(101,102,103)::A,B,C
CALL SUB(A)
 :
PRINT *,B(101,102,103)
A=C
A(101,102,103)=C(1,2,3)
```

If this type of program is compiled with the -Karray_merge option specified, then it will be compiled as one of the following types of programs:

```
INTEGER,DIMENSION(3,101,102,103)::TMP
   !  A, in 1 element of the 1st dimension, merge : TMP (1,101,102,103)
   !  B, in 2 element of the 1st dimension, merge : TMP (2,101,102,103)
   !  C, in 3 element of the 1st dimension, merge : TMP (3,101,102,103)
CALL SUB(TMP(1,:,:,:))
 :
PRINT *,TMP(2,101,102,103)
TMP(1,:,:,:) = TMP(3,:,:,:)
TMP(1,101,102,103)=TMP(3,1,2,3)
```

Note:

TMP is a variable generated by the compiler.

Example2:

```
PROGRAM MAIN
INTEGER,DIMENSION(100,101,102) :: A
 :
CALL SUB(A(1,1,1))
END
SUBROUTINE SUB(C)
INTEGER,DIMENSION(100) :: C
 :
END
```

When this feature is used, the "A", which is an argument of SUB on MAIN, may not be associated with C(100), which is a parameter of SUB, because an element is appended in the first dimension of "A".

The compiler option-Karray_merge_common facilitates to merge the named common block object array.

The target array variables are explicit shape arrays within named common blocks with bounds that are constant expressions. In this case, a common block name is created. It is the following form:

```
_0_ <common block name>
```

All of the common-block-objects within the same named common block must be arrays with the same shape. This function will not be applied within specification expressions or the constant expressions of declaratives. The compiler automatically adds the element of the first dimension and merges the elements. The merging order is undefined. This may cause a compile error or may introduce an incorrect result, because the array name will be replaced with the merged array names. This feature is applied to each individual array, without considering argument association. For this reason, using this function may cause incorrect results.

The following shows an example:

Example1:

```
INTEGER,DIMENSION(101,102,103)::A,B,C
COMMON /COM/ A,B,C
CALL SUB(A)
   :
PRINT *,B(101,102,103)
```

If this type of program is compiled with the -Karray_merge_common option, then it will be compiled as one of the following types of programs:

```
INTEGER,DIMENSION(3,101,102,103)::TMP
COMMON /_0_COM/ TMP
CALL SUB(TMP(1,:,:,:))
PRINT *,TMP(2,101,102,103)
```

Note:

TMP is a variable generated by the compiler.

9.15.2 The Restrictions of Merging Local Array Variables

Arrays that meet any of the following conditions cannot be the target of local array variable merge:

- Derived types or character types
- Literal with name
- An array with a SAVE attribute or TARGET attribute
- Pointer variable
- An array with an initial value
- Equivalence-object
- Namelist-group-object
- Declaration within the module
- Address holding variable
- Function result
- Common block objects
- Dummy arguments
- Specified in THREADPRIVATE statement
- Automatic array

- Declaration in BLOCK

  Note:

  If the bound expression of automatic object array is the following condition, the automatic object array is not restricted.

  1. Bound expression of declaration contains exactly one variable, or
  2. Bound expression of declaration is constant expression.

9.15.3 Restrictions of target variables on common block object array merge

Arrays that meet any of the following conditions cannot be the target of common-block-object array merge:

- Derived types or character types
- An array with a TARGET attribute
- An array with an initial value
- Equivalence-object
- Namelist-group-object
- Automatic array
- An array within program units that include OpenMP statements

9.16 Multi-Operation Function

9.16.1 Calling of Multi-Operation Function

Multi-operation functions are functions that improve performance by using intrinsic functions for calculations and operations on multiple arguments called at one time. By specifying compiler option -Kmfunc or optimization control line MFUNC, the compiler analyzes loops and replaces them with multi-operation functions if possible.

If there are complex loops, and the compiler cannot analyze it, modify the program to call multi-operation function directly.

The following is required:

- The argument n which specifies the size must be an 8-byte integer.
- The external procedure names of the multi-operation functions are the external procedure names for the C language. For this reason, when called from a Fortran program, specifies to change how to process external procedure name by $pragma specifier.
- The argument check may be omitted in the multi-operation function for high-speed. Therefore, the user program may be terminated abnormally when the special value (NaN and Inf, etc.) defined by IEEE 754 is input.

Table 9.2 lists the multi-operation functions that can be called directly from user programs.

Table 9.2 Multi-operation functions that can be called directly from user programs

function/operation	type argument	calling format	content of calculation
ACOS	real(4)	v_acos(x,n,y)	y(i) = acos(x(i))
	real(8)	v_dacos(x,n,y)	y(i) = dacos(x(i))
ASIN	real(4)	v_asin(x,n,y)	y(i) = asin(x(i))
	real(8)	v_dasin(x,n,y)	y(i) = dasin(x(i))
ATAN	real(4)	v_atan(x,n,y)	y(i) = atan(x(i))
	real(8)	v_datan(x,n,y)	y(i) = datan(x(i))
ATAN2	real(4)	v_atan2(x1,x2,n,y)	y(i) = atan2(x1(i),x2(i))
	real(8)	v_datan2(x1,x2,n,y)	y(i)=datan2(x1(i),x2(i))
ERF	real(4)	v_erf(x,n,y)	y(i) = erf(x(i))

function/operation	type argument	calling format	content of calculation
	real(8)	v_derf(x,n,y)	y(i) = derf(x(i))
ERFC	real(4)	v_erfc(x,n,y)	y(i) = erfc(x(i))
	real(8)	v_derfc(x,n,y)	y(i) = derfc(x(i))
EXP	real(4)	v_exp(x,n,y)	y(i) = exp(x(i))
	real(8)	v_dexp(x,n,y)	y(i) = dexp(x(i))
	complex(8)	v_cdexp(x,n,y)	y(i) = cdexp(x(i))
EXP10	real(4)	v_exp10(x,n,y)	y(i) = exp10(x(i))
	real(8)	v_dexp10(x,n,y)	y(i) = dexp10(x(i))
LOG	real(4)	v_alog(x,n,y)	y(i) = alog(x(i))
	real(8)	v_dlog(x,n,y)	y(i) = dlog(x(i))
LOG10	real(4)	v_log10(x,n,y)	y(i) = log10(x(i))
	real(8)	v_dlog10(x,n,y)	y(i) = dlog10(x(i))
SIN	real(4)	v_sin(x,n,y)	y(i) = sin(x(i))
	real(8)	v_dsin(x,n,y)	y(i) = dsin(x(i))
COS	real(4)	v_cos(x,n,y)	y(i) = cos(x(i))
	real(8)	v_dcos(x,n,y)	y(i) = dcos(x(i))
SIN & COS	real(4)	v_scn(x,n,y,z)	y(i) = sin(x(i))
			z(i) = cos(x(i))
	real(8)	v_dscn(x,n,y,z)	y(i) = dsin(x(i))
			z(i) = dcos(x(i))
exponentiation	real(4)	v_arxr(x1,x2,n,y)	y(i) = x1(i)**x2(i)
	real(4)	v_arxr1(x,a,n,y)	y(i) = x(i)**a
	real(8)	v_adxd(x1,x2,n,y)	y(i) = x1(i)**x2(i)
	real(8)	v_adxd1(x,a,n,y)	y(i) = x(i)**a

Note:

When calling multi-operation functions directly from user programs, the memory used for the argument to return the results of the operations must be separate from the memory used for the other arguments. If the areas overlap, the result may be incorrect.

The following shows the example of calling multi-operation functions.

Example1:

```
INTEGER,PARAMETER :: N=1000
REAL(KIND=8) :: A
REAL(KIND=8),DIMENSION(N) :: X,Y,Z
   :
A = 0.2D0
DO I=1,N
  Y(I) = DEXP(X(I))
  Z(I) = X(I)**A
ENDDO
```

Invocation of multi-operation function

```
INTEGER(KIND=8),PARAMETER :: N=1000
REAL(KIND=8) :: A
REAL(KIND=8),DIMENSION(N) :: X,Y,Z
EXTERNAL V_DEXP !$PRAGMA C(V_DEXP)
```

```
EXTERNAL V_ADXD1!$PRAGMA C(V_ADXD1)
   :
A = 0.2D0
CALL V_DEXP(X,N,Y)
CALL V_ADXD1(X,A,N,Z)
```

Example2:

If a function is called within an IF construct, multi-operation functions can be used by storing only the elements that need calculation in arrays.

```
INTEGER,PARAMETER :: N=1000
REAL(KIND=8) :: X,Y,Z
REAL(KIND=8),EXTERNAL :: FUNC
   :
DO I=1,N
   X = ..
   IF (X.GT.A) THEN
      Y = Y + SIN(X)
      Z = Z + COS(X)
   END IF
ENDDO
```

Invocation of multi-operation function

```
INTEGER(KIND=8),PARAMETER :: N=1000
INTEGER(KIND=8) :: J
REAL(KIND=8) :: X,Y,Z
REAL(KIND=8),DIMENSION(N) :: WX,WY,WZ
EXTERNAL V_DSCN     !$PRAGMA C(V_DSCN)
   :
J = 0
DO I=1,N
   X = ..
   IF (X.GT.A) THEN
      J = J + 1
      WX(J) = X
   END IF
ENDDO
CALL V_DSCN(WX,J,WY,WZ)
DO I=1,J
   Y = Y + WY(I)
   Z = Z + WZ(I)
ENDDO
```

9.16.2 Effects of Compiler Option -Kmfunc=3

If the compile option -Kmfunc=3 is specified, the program may terminate abnormally as a side effect of changing loops that include IF constructs into multi-operation functions. The following shows an example of the effects of -Kmfunc=3:

Example: Loops that include IF constructs

```
 SUBROUTINE SUB(A,B)
 DIMENSION A(1000),B(1000)
  .
  .
 DO I=1,2000
    IF (I.LE.1000) THEN
  A(I) = COS(B(I))
    ENDIF
 ENDDO
```

[Description]

In the above example, the value of the subscript of array B never exceeds the declared range because the intrinsic function COS, of which the actual argument is the array element, is called if the condition I.LE.1000 holds.

By applying the multi-operation function optimization, the DO loop references an array element for each iteration of the loop, and uses it as an argument of multi-operation function regardless the result of IF-construct. Access may therefore exceed the array bounds. This will cause an abnormal termination of the program, with a message informing of a memory protection exception (READ).

To prevent this, do not specify the compiler option -Kmfunc=3.

9.17 Software Control of Sector Cache

9.17.1 Using Sector Cache

Sector cache is a mechanism to prevent reusable data on the cache from being driven out by non-reusable data. In particular, sector cache is effective for parallel execution when the shared L2 cache is accessed by multiple CPUs. By using sector cache, reusable data is separated from non-reusable data on the cache. There are two methods to control sector cache with software; environment variables and optimization control lines. Both are methods of specifying the maximum number of ways for each sector, which are detailed below.

Performance will not be improved if the maximum number of ways is specified without consideration for the size of the array that is to be protected from being driven out. The cache utilization will decline, and may cause a reduction in speed. Further, even if software control of the sector cache is not used, cache control with LRU (Least Recently Used) still works. Thus speed may not improve even if this is specified. One effective way to improve the performance of sector caches is to determine the reusable array size and specify the number of ways for sector 1 to store the array, after ensuring L2 cache miss has occurred by confirming the PA information. Note that in order to use sector caches, the compile-time option -Kdalign must be set.

9.17.2 How to Control Sector Cache by Software

There are two methods to control sector caches with software in this processor, as follows:

- Optimization control rows
- Environment variables and optimization control rows

9.17.2.1 Software control with optimization control rows

There are three optimization control specifiers used for software control of sector caches, as follows:

- CACHE_SECTOR_SIZE(*l2_n1*, *l2_n2*) to END_CACHE_SECTOR_SIZE
- CACHE_SECTOR_SIZE(*l1_n1*, *l1_n2*, *l2_n1*, *l2_n2*) to END_CACHE_SECTOR_SIZE
- CACHE_SUBSECTOR_ASSIGN(*array1* [,*array2*...]) to END_CACHE_SUBSECTOR

The CACHE_SECTOR_SIZE specifier directs the maximum number of ways in sectors 0 and 1 of the cache.

When 4 arguments are specified, they direct the number of all ways for caches: *l1_n1*, for sector 0; *l1_n2*, for sector1; *l2_n1*, for sector0 of the L2 cache; *l2_n2*, for sector1 of the L2 cache.

When two arguments are specified, they direct that the maximum number of ways for sector 0 in the L2 cache it will be *l2_n1*, and for sector 1 of the L2 cache it will be *l2_n2*.

To direct in function, the CACHE_SECTOR_SIZE specifier must be used in procedure line. In this case, the CACHE_SECTOR_SIZE specifier is valid in the scope of the function.

To direct a part of function, the range must be indicated by the CACHE_SECTOR_SIZE and END_CACHE_SECTOR_SIZE specifiers in statement lines. Note that the range must be not nested.

However, combined use of procedure line and statement line is accepted.

The arguments *l1_n1, l1_n2, l2_n1,* and *l2_n2* must be as follows;

- $0 \leq$ *l1_n1* $\leq$ the maximum number of ways in the L1 cache
- $0 \leq$ *l1_n2* $\leq$ the maximum number of ways in the L1 cache
- $0 \leq$ *l2_n1* $\leq$ the maximum number of ways in the L2 cache

- 0 ≤ *l2_n2* ≤ the maximum number of ways in the L2 cache

In order to ensure that the data in sector 0 does not drive out the data in sector 1, it is recommended to configure {*l1_n1*+ *l1_n2* = maximum number of ways in L1 cache}, and {*l2_n1*+ *l2_n2* = maximum number of ways in L2 cache}.

The CACHE_SUBSECTOR_ASSIGN specifier directs to specify an array stored in sector 1 of the cache.

However, it becomes effective only when the array of numeric type or logical type is specified.

To direct in function, the CACHE_SUBSECTOR_ASSIGN specifier must be used in procedure line. In this case, the CACHE_SUBSECTOR_ASSIGN specifier is valid in the scope of the function.

To direct a part of function, the range must be indicated by the CACHE_SUBSECTOR_ASSIGN and END_CACHE_SUBSECTOR_ASSIGN specifiers in statement lines.

Note that the range must be not nested.

However, combined use of procedure line and statement line is accepted.

The following shows an example:

Example: Control sector caches by OCLs

```
 SUBROUTINE OCL_SECTOR(A,B,N,N2)
 REAL(KIND=8),DIMENSION(N,N) :: A
 REAL(KIND=8),DIMENSION(N,N,N2) :: B
! Specify the maximum number of ways for each sector in the level 2 cache
! Specify the maximum number of ways in sector 0 = 2,
! the maximum number of ways in sector 1 = L2_MAX-2
! L2_MAX = maximum number of ways in the L2 cache
!OCL CACHE_SECTOR_SIZE (the maximum number of ways in sector 0, the maximum number of ways in
sector 1)
! Optimization control specifier to specify an array stored in sector 1
! Array A is specified for Sector 1
!OCL CACHE_SUBSECTOR_ASSIGN (A)
 DO K=1,N2
    DO J=1,N
       DO I=1,N
          A(I,J)=A(I,J)+B(I,J,K)
       ENDDO
    ENDDO
 ENDDO
! End scope of CACHE_SUBSECTOR_ASSIGN.
!OCL END_CACHE_SUBSECTOR
! End scope of CACHE_SECTOR_SIZE.
! Return the maximum number of ways for each sector to the value
! they were before specifying CACHE_SECTOR_SIZE.
!OCL END_CACHE_SECTOR_SIZE
 END SUBROUTINE
```

9.17.2.2 Software control with environment variables and optimization control rows

The initial values for the maximum number of ways for each sector can also be specified with the following environment variable. The maximum number of ways for the object program at startup can be specified with environment variables.

FLIB_SCCR_CNTL

This environment variable specifies to use sector cache. The values that can be set in the environment variable are shown below, along with their meanings. TRUE (use sector cache) is default.

- TRUE: use sector cache.
- FALSE: do not use sector cache.

FLIB_L2_SECTOR_NWAYS_INIT

This environment variable specifies the initial maximum number of ways for sector caches. The values that can be set in the environment variable are of the same format as that used in the CACHE_SECTOR_SIZE optimization specifier; n1 and n2. This is valid only when

FLIB_SCCR_CNTL is TRUE. 0,0 is default. The values that can be specified are of the same format as the CACHE_SECTOR_SIZE optimization specifier, n1, n2, and in range 0 ≤n1, n2 ≤{maximum number of ways} respectively. In order to ensure that the data in sector 0 does not drive out the data in sector 1, it is recommended to specify n1+n2= maximum number of ways.

The following example specifies 2,10 as the value of environment variable FLIB_L2_SECTOR_NWAYS_INIT before starting the program, which enables the software control to store Array A on the sector 1 with maximum number of ways for sector 1 to be 10.

Example: Ways to control sector caches with software (2)

```
  SUBROUTINE OCL_SECTOR(A,B,N,N2)
  REAL(KIND=8),DIMENSION(N,N) :: A
  REAL(KIND=8),DIMENSION(N,N,N2) :: B
! Optimization control specifier to specify an array stored in sector 1
! Array A is specified for Sector 1
!OCL CACHE_SUBSECTOR_ASSIGN (A)
  DO K=1,N2
    DO J=1,N
      DO I=1,N
        A(I,J)=A(I,J)+B(I,J,K)
      ENDDO
    ENDDO
  ENDDO
! End scope of CACHE_SUBSECTOR_ASSIGN.
!OCL END_CACHE_SUBSECTOR
  END SUBROUTINE
```

9.17.2.3 Behavior when Invalid Value is specified

If a value that exceeds the maximum number of ways is specified for n1 or n2 in the optimization indicator CACHE_SECTOR_SIZE and the environment variable FLIB_L2_SECTOR_NWAYS_INIT, the results will be the same as if the actual maximum number of ways is specified. If a number less than 0 is specified, that optimization control line is ignored. If the specified value is outside the range of two-byte signed integer, operation will be incorrect.

9.18 Incorrectly Specified Optimization Indicators

When optimization control lines are incorrectly specified, the execution result may not be incorrect.

9.18.1 Example of SIMD(ALIGNED) specifier

SIMD optimization can be used when double-precision floating point stores are on 16-byte boundaries. However, if SIMD(ALIGNED) specifier is specified for double-precision floating point stores that are not on 16-byte boundaries, the program will terminate abnormally with SIGSEGV.

See 5.2 Correct Data Boundaries for details about array boundaries the compiler allocates.

The following shows an example:

Example1:

```
 REAL(KIND=8),DIMENSION(10) :: A
 COMMON //N,A
!OCL SIMD(ALIGNED)
 DO I=1,10
    A(I) = ...
 ENDDO
```

Array A may not be on 16-byte boundary because that A is not first element of a COMMON block.

Example2:

```
 REAL(KIND=8),DIMENSION(10) :: A
 COMMON //A
!OCL SIMD(ALIGNED)
 DO I=2,10
```

```
   A(I) = ...
ENDDO
```

While the array A is on 16-byte boundary, the element of array referenced first in this loop is the second element of array A, and that element is not on 16-byte boundary.

9.18.2 Example of NOVREC specifier

When array A is not recurrence data and NOVREC specifier is effective, compiler uses SIMD instructions. When array A is recurrence data and NOVREC specifier is effective, compiler uses SIMD instructions. But the execution result is unpredictable.

Example:

```
!OCL NOVREC
  DO I=1,100
    A(I)=A(I+M)+...
  ENDDO
```

9.18.3 Example of CLONE specifier

In the copied loops under the generated branches, the specified variables are treated as invariants in the loops. Thus, the execution result is not guaranteed in case that the value of the variables changes in the loop.

Example 1:

```
INTEGER,DIMENSION(32)::A
INTEGER,TARGET::N
INTEGER,POINTER::M
M=>N
!OCL CLONE(N==10)
DO I=1,32
    M=5
    A(I) = N
ENDDO
```

Do not specify a variable which is changed in the loop because the execution result is not guaranteed.

Example 2:

```
INTEGER::N
!OCL CLONE(N==10)
DO I=1,32
   CALL SUB(N)
ENDDO
```

Do not specify a variable which may be changed in the loop because the execution result is not guaranteed.

9.19 Attentions for Mixed Language Programming

The following describes the notes when programming with different languages in this processor.

9.19.1 Address parameters received in the C language

If an external C program stores the addresses of the parameters, the result may be incorrect because such operation is unsupported and the optimization is performed without considering it. (Refer to example 1)

In this case, to obtain the correct result, pass the address explicitly using arguments rather than saving the parameter address in the function. (Refer to example 2)

Alternatively, pass the parameter address by specifying "%VAL(LOC(argument))" as the argument of the function caller in the Fortran program in Example 1. (Refer to Example 3)

Example1: Incorrect saving of pointer

```
int fun_(int *flag, int *p)
{
  static int *save;
  if (*flag == 0) {
    save = p; /* Saving address dummy argument. */
    return 0;
  } else {
    return *save;
  }
}
```

```
 INTEGER(KIND=4)::I,J
 INTEGER(KIND=4),EXTERNAL::FUN
 J = 0
 I = FUN(0, J)
 J = 1
 I = FUN(1, 0)
 PRINT *, I, J
 END
```

Example2: Format for not saving pointer

```
int fun_(int *flag, int *p)
{
  if (*flag == 0) {
    return 0;
  } else {
    return *p;
  }
}
```

```
 INTEGER(KIND=4)::I,J
 INTEGER(KIND=4),EXTERNAL::FUN
 J = 0
 I = FUN(0, J)
 J = 1
 I = FUN(1, J)
 PRINT *, I, J
 END
```

Example3: Format for explicit passing of address

```
 INTEGER(KIND=4)::I,J
 INTEGER(KIND=4),EXTERNAL::FUN
 J = 0
 I = FUN(0, %VAL(LOC(J)))
 J = 1
 I = FUN(1, 0)
 PRINT *, I, J
 END
```

9.19.2 Pointers received in the C language

If a pointer address is returned as the function result or COMMON when the pointer is received in the C program, the result may be incorrect because such an operation is unsupported and the optimization is performed without considering it. (Refer to example 4)

In this case, using the intrinsic function LOC to store the address in the pointer will produce a correct result. (Refer to example 5)

Example4: Incorrect return of pointer

```
int *my_addr_(int *p)
{
```

```
  return p;
}
```

```
INTEGER(KIND=4) I,IP
POINTER(PTR,IP)
INTEGER(KIND=8) MY_ADDR
I = 2
PTR = MY_ADDR(I)
IP = 1
PRINT *, I
END
```

Example5: Correct pointer usage

```
INTEGER(KIND=4) I,IP
POINTER(PTR,IP)
I = 2
PTR = LOC(I)
IP = 1
PRINT *, I
END
```

Chapter 10 Fortran Modules

A module reference is a USE statement specifying the module name.

This chapter describes items related to compiling input files that have modules, modules references and intrinsic modules references.

10.1 Compilation of Modules and Module References

Note the following:

The object file of a module is created in the current directory. Its filename is the source filename with the suffix replaced by the suffix .o.

When a module is compiled, an information file is created by the compiler. The name of the file is the module name with the suffix .mod appended. This information file is needed when compiling a program that contains a USE statement for the module. The .mod information file is created according to the following rules:

- If the compiler option -M is specified, it is created in the directory specified in the -M option.
- If the compiler option -M is not specified, it is created in the directory from which frtpx is being executed.

When a program that contains a USE statement is compiled, the corresponding information file is searched for in the following order:

- A directory named in the -M option (if specified).
- The directory from which frtpx is being executed.
- A directory named in the -I option (if specified).

When linking a program containing a module reference, you must specify the object file of the corresponding module.

Example1:

The module and the module reference:

! file name : a.f90

```
MODULE MOD
  INTEGER::VALUE=1
END MODULE

PROGRAM MAIN
  USE MOD
  PRINT *,VALUE
END PROGRAM
```

Compile command :

```
$frtpx a.f90
```

Example2:

The module and the module reference appear in different files:

! file name : a.f90

```
MODULE MOD
  INTEGER::VALUE=1
END MODULE
```

! file name : b.f90

```
PROGRAM MAIN
  USE MOD
  PRINT *,VALUE
END PROGRAM
```

Compile command :

```
$ frtpx a.f90 -c
$ frtpx b.f90 a.o
```

or

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx a.o b.o
```

Example3:

The module and two references to the module reference appear in three different files:

! file name : a.f90

```
MODULE MOD
  INTEGER::VALUE=1
END MODULE
```

! file name : b.f90

```
PROGRAM MAIN
  USE MOD
  VALUE=VALUE+1
  CALL EXTERNAL_SUB
END PROGRAM
```

! file name : c.f90

```
SUBROUTINE EXTERNAL_SUB
  USE MOD
  PRINT *,VALUE
END SUBROUTINE
```

Compile command:

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx c.f90 a.o b.o
```

or

```
$ frtpx a.f90 -c
$ frtpx b.f90 -c
$ frtpx c.f90 -c
$ frtpx a.o b.o c.o
```

Example4:

The module and the module reference appear in different files using the compiler options -I, and -M:

! file name : a.f90

```
MODULE MOD1
  INTEGER::VALUE=1
END MODULE
```

! file name : b.f90

```
MODULE MOD2
  USE MOD1
  INTEGER::VALUE2=2
CONTAINS

SUBROUTINE MOD2_SUB
  VALUE=VALUE2+1
```

```
  END SUBROUTINE
END MODULE
```

Compile command:

```
$ frtpx a.f90 -c
```

Note:

The information file mod1.mod for module mod1 is generated in the current directory.

```
$ frtpx a.f90 -c -Mdirectory
```

Note:

The information file mod1.mod for module mod1 is generated in the directory specified by the -M option argument.

```
$ frtpx b.f90 -c -Idirectory
```

Note:

1. The information file mod1.mod for module mod1 is searched for in the directory specified by the -I option argument or in the current directory.
2. The information file mod2.mod for module mod2 is generated in the current directory.

```
$ frtpx b.f90 -c -Iinput_directory -Moutput_directory
```

Note:

1. The information file mod1.mod for module mod1 is searched for in the directory specified in the -I option argument, the directory specified by the -M option argument, or in the current directory.
2. The information file mod2.mod for module mod2 is generated in the directory specified by the -M option argument.

10.2 Recompiling a Program that Contains a Reference to a Module that Has Changed

In either of the following cases, a program unit that contains a module reference must be recompiled.

- An entity with the PUBLIC attribute in the module specification part is changed
- The characteristics of a module procedure with the PUBLIC attribute are changed

10.3 Restrictions on Modules

When the following compiler option is specified for compile a module, you must specify the same option for compile the other program that including the module reference.

- -AA option
- -AU option
- -Kcommonpad[=N] option
- -Karray_subscript option

When a module is compiled with compiler option -X9 or -X03, the other program including the module reference also has to be compiled with compiler option -X9 or -X03.

When a module is compiled with compiler option -X6, the other program including the module reference also has to be compiled with compiler option -X6.

When a module is compiled with compiler option -X7, the other program including the module reference also has to be compiled with compiler option -X7.

10.4 Intrinsic Modules

The intrinsic modules are provided in the processor. There are standard intrinsic module and non-standard intrinsic modules in the intrinsic modules. The Fortran environment module (ISO_FORTRAN_ENV), IEEE module (IEEE_EXCEPTIONS, IEEE_ARITHMETIC and IEEE_FEATURES), module for interoperable with C program (ISO_C_BINDING) are standard intrinsic modules. The SERVICE_ROUTINES and OMP_LIB modules are the non-standard intrinsic module. See 6.6.6 Service Routines for information about SERVICE_ROUTINES modules.

Be careful to the following when the intrinsic module is referenced.

- When the compiler option -AU is specified, the language entities of intrinsic modules must be lowercase letter.

10.4.1 COMPILER_OPTIONS intrinsic module function

Compiler options information on the function result is the same as compiler options information with compiler option -Q or -Nlst. Consider 1200 characters or more for character strings of the function result on programming.

10.4.2 COMPILER_VERSION intrinsic module function

Version information on the function result is the same as information on the Fortran compiler when compiler option -V is specified. Consider 90 characters or more for character strings of the function result on programming.

Chapter 11 Mixed Language Programming

This chapter describes the specification and notes when linking a Fortran program and a C/C++ program.

The description of C program on this chapter is applied C++ program. In linking with the C++ program, see Section "Notes on Interlanguage Linkage" of the "C++ User's Guide" in addition to this chapter.

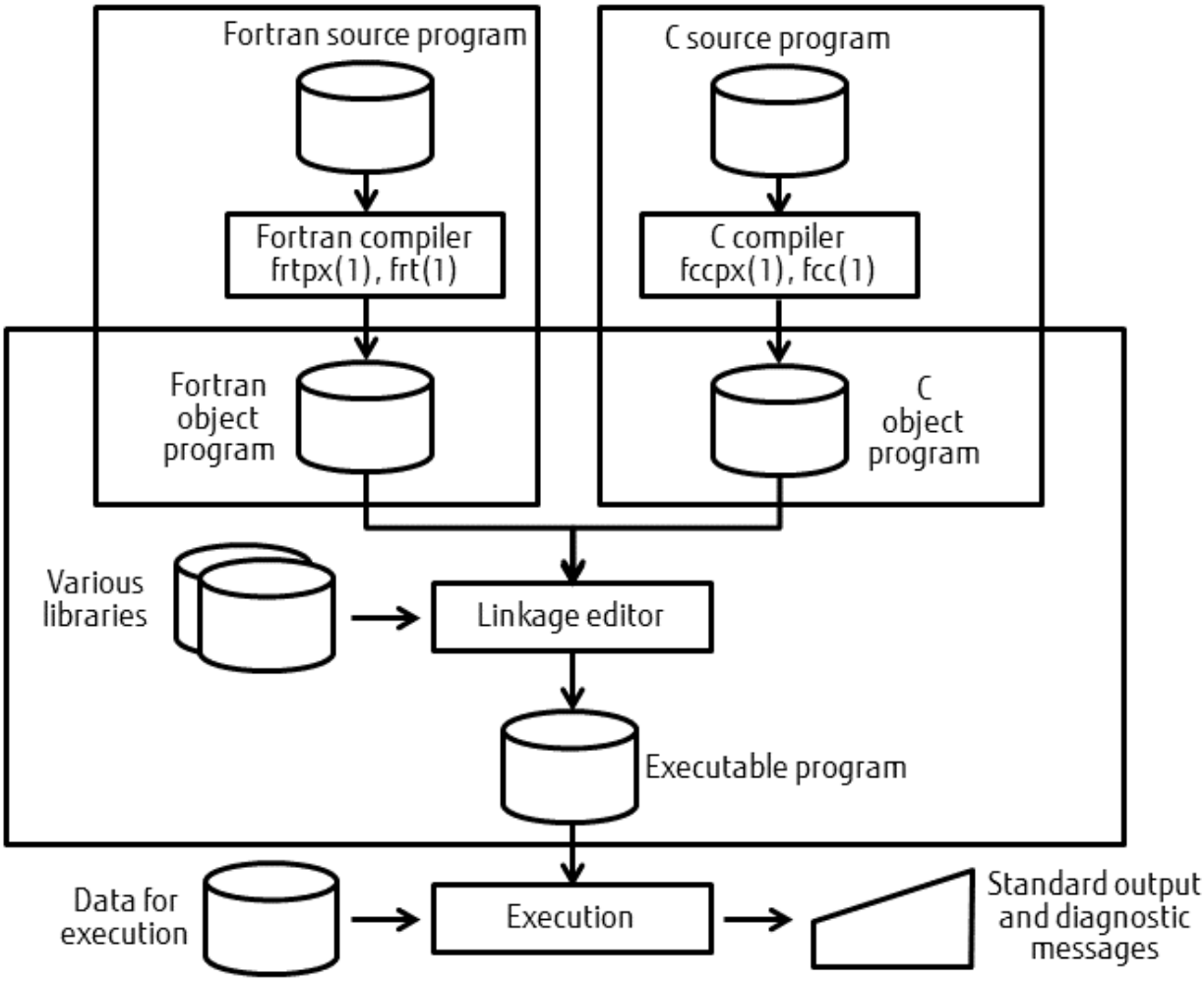
11.1 Summary of Mixed Language Programming

A Fortran program and a C program can be linked statically and executed.

For the C and C++ languages, there are two types of compile commands, one is the cross compile command, and the other is the own compile command.

Programming Language	Cross Compiler	Own Compiler
C	fccpx	fcc
C++	FCCpx	FCC

Summary of compiling and linking a Fortran program and a C program are shown in the following figure.



11.2 Features for Mixed Language Programming

This system has Fortran standard and FUJITSU Fortran extension specifications for mixed language programming.

The following Fortran standard specifications for mixed language programming are offered.

- Intrinsic module ISO_C_BINDING
- BIND statement, BIND attribute specifier, Language binding specifier, Procedure language binding specifier
- VALUE statement, VALUE attribute specifier

The following FUJITSU Fortran extension specifications for mixed language programming are offered.

- VAL intrinsic function, %val(a) specifies that an actual argument
- CHANGEENTRY statement
- $pragma specifier

The following features are supported by compiler option.

- The compiler options -ml and -mldefault to change how to make external procedure. See Section "11.3 Rules for Processing Fortran Procedure Names".
- The compiler option -Az to change how to call the value of character type for the argument. See Section "11.13 Notes of the Compiler Option -Az".

11.2.1 Passing Arguments by Value

By default, actual arguments of external procedure reference are called by address. See Section "11.7 Passing and Getting Arguments". In the following cases, the actual argument is passed by value.

- The intrinsic function VAL(a) or %val(a) for actual argument is specified.
- The VALUE attribute is specified in a procedure interface.

11.2.1.1 Passing Arguments by Value in Procedure Interface

When the VALUE attribute is specified in procedure interface, the argument is passed by value.

See Section "11.2.2 Getting Arguments by Value" for information about the actual argument that can be associated.

Example: the VALUE attribute in procedure interface

```
INTERFACE
   SUBROUTINE CSUB(ARG1,ARG2)  BIND(C)
   USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
   INTEGER(C_INT),VALUE::ARG1
   INTEGER(C_INT)      ::ARG2
   END SUBROUTINE CSUB
END INTERFACE
CALL CSUB(2,3)            ! The first argument is passed by value,
                             !the second argument is called by address.
END
```

11.2.1.2 Passing Arguments by Value in Intrinsic Function

When the compiler option -Nobsfun is specified, the function VAL(a) and %val(a) are intrinsic functions. The intrinsic function VAL(a) and %val(a) specifier can specify only for actual argument of external procedure reference. The actual argument passed by value must be scalar expression of type one-byte integer, two-byte integer, four-byte integer, eight-byte integer, one-byte logical, two-byte logical, four-byte logical, eight-byte logical, single-precision real or double-precision real.

Example: intrinsic function VAL()

```
CALL CSUB(VAL(2),3)       ! first argument passed by value,
                          ! second argument passed by address
END
```

11.2.2 Getting Arguments by Value

By default, dummy arguments of external procedure are got by address. See Section "11.7 Passing and Getting Arguments". The VALUE attribute is specified for a dummy argument of an external procedure, the Fortran program gets the dummy arguments by value. The VALUE attribute can specify by the VALUE statement or attribute specifier in a type declaration statement.

When the dummy argument has the Quadruple-precision complex, the character type of procedure without procedure language binding specifier, the derived type whose size is 16 bytes or less, the derived type who has POINTER attribute in component, the derived type who has ALLOCATABLE attribute in component or OPTIONAL attribute, the address of the variable is always received.

Example: VALUE statement

```
SUBROUTINE CSUB(I,K)
VALUE :: I                ! first argument passed by value,
                          ! second argument passed by address
RRINT *, I,K
END
```

11.2.3 CHANGEENTRY Statement

The CHANGEENTRY statement changes rules for processing Fortran procedure names combine with -ml compiler option. See Section "11.3 Rules for Processing Fortran Procedure Names" for details.

The external procedure name must not have the procedure language binding specifier.

Example: CHANGEENTRY statement

```
CHANGEENTRY :: CSUB
CALL CSUB()
END
```

11.2.4 $pragma Specifier

$pragma specifier has the following form and can be specified like a comment.

```
!$pragma c (prc)
```

prc must be an external procedure name. The $pragma specifier specifies that the procedure *prc* is coded by C program and changes rules for processing procedure names. See Section "11.3 Rules for Processing Fortran Procedure Names".

The external procedure name of $pragma specifier must not have the procedure language binding specifier.

Example: $pragma specifier

```
EXTERNAL CSUB          !$pragma C(CSUB)
   :
CALL CSUB              ! CSUB is treated as a calling procedure in C program.
END
```

11.2.5 Intrinsic Module ISO_C_BINDING

The following specifications are offered by intrinsic module ISO_C_BINDING.

- Named constants for interoperable with C program
- Types for interoperable with C program
- Intrinsic procedures for interoperable with C program

11.2.5.1 Named Constants by Intrinsic Module ISO_C_BINDING

The following named constants are offered by intrinsic module ISO_C_BINDING.

- Named constants for kind type parameter

- Named constants for names of C program character
- Named constants for null pointer

1. The value of named constants by intrinsic module ISO_C_BINDING and C program types correspond in "Table 11.1 Types of C program and, types and kind type parameters of Fortran correspond". If data is passed between Fortran and C programs, type of C program and, type and kind type parameter of Fortran must correspond. If data type is character, the value of type length parameter must be 1.

Table 11.1 Types of C program and, types and kind type parameters of Fortran correspond

Named constants	Value	C program types	Type specifiers of Fortran
C_SHORT	2	short int	INTEGER
C_INT	4	int	INTEGER
C_LONG	8	long int	INTEGER
C_LONG_LONG	8	long long int	INTEGER
C_SIGNED_CHAR	1	signed char / unsigned char	INTEGER
C_SIZE_T	8	size_t	INTEGER
C_INT8_T	1	int8_t	INTEGER
C_INT16_T	2	int16_t	INTEGER
C_INT32_T	4	int32_t	INTEGER
C_INT64_T	8	int64_t	INTEGER
C_INT_LEAST8_T	1	int_least8_t	INTEGER
C_INT_LEAST16_T	2	int_least16_t	INTEGER
C_INT_LEAST32_T	4	int_least32_t	INTEGER
C_INT_LEAST64_T	8	int_least64_t	INTEGER
C_INT_FAST8_T	1	int_fast8_t	INTEGER
C_INT_FAST16_T	8	int_fast16_t	INTEGER
C_INT_FAST32_T	8	int_fast32_t	INTEGER
C_INT_FAST64_T	8	int_fast64_t	INTEGER
C_INTMAX_T	8	intmax_t	INTEGER
C_INTPTR_T	8	intptr_t	INTEGER
C_FLOAT	4	float	REAL
C_DOUBLE	8	double	REAL
C_LONG_DOUBLE	16	long double	REAL
C_FLOAT_COMPLEX	4	float _Complex	COMPLEX
C_DOUBLE_COMPLEX	8	double _Complex	COMPLEX
C_LONG_DOUBLE_COMPLEX	16	long double _Complex	COMPLEX
C_BOOL	1	_Bool	LOGICAL
C_CHAR	1	char	CHARACTER

Example: Using of named constant of kind type parameter by intrinsic module ISO_C_BINDING
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT

```
INTEGER(C_INT)::J,K,L
INTERFACE
 SUBROUTINE SUB(J,K,L) BIND(C)
```

```
   USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
   INTEGER(C_INT)::J,K,L
 END SUBROUTINE
END INTERFACE
J = 10
K = 20
L = 30
CALL SUB(J,K,L)
END
```

C program:

```
#include <stdio.h>
void sub(j,k,l)
int *j,*k,*l;
{
printf ("J:%d K:%d L:%d \n",*j,*k,*l) ;
}
```

2. Named constants of C program character by intrinsic module ISO_C_BINDING, and the meaning are follows.

Named constants	C program characters
C_NULL_CHAR	null character
C_ALERT	alert
C_BACKSPACE	backspace
C_FORM_FEED	form feeds
C_NEW_LINE	new line
C_CARRIAGE_RETURN	carriage return
C_HORIZONTAL_TAB	horizontal tab
C_VERTICAL_TAB	horizontal tab

The type of named constants is character type and length type parameter is 1.

Example: Using of named constant of C program character by intrinsic module ISO_C_BINDING

```
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_NULL_CHAR
CHARACTER(LEN=5)::ST
ST='ABCD'//C_NULL_CHAR ! Append null character
:
END
```

3. Named constants of null pointer by intrinsic module ISO_C_BINDING, the meaning and type are follows.

Named constants	Meaning	Type
C_NULL_PTR	NULL	C_PTR
C_NULL_FUNPTR	null pointer to function	C_FUNPTR

Example: Using of named constant of null pointer by intrinsic module ISO_C_BINDING

```
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_PTR,C_NULL_PTR
INTERFACE
 SUBROUTINE SUB(PTR) BIND(C)
   USE,INTRINSIC::ISO_C_BINDING,ONLY:C_PTR
   TYPE(C_PTR),VALUE::PTR
 END SUBROUTINE
END INTERFACE
TYPE(C_PTR)::PTR
PTR=C_NULL_PTR          ! Set NULL
```

```
CALL SUB(PTR)
END
```

11.2.5.2 Types by Intrinsic Module ISO_C_BINDING

C_PTR type and C_FUNPTR type are offered by intrinsic module ISO_C_BINDING.

C_PTR type is a pointer type of C program. C_FUNPTR type is a function pointer type of C program. C_PTR and C_FUNPTR are derived type with private component.

11.2.5.3 Intrinsic Procedures by Intrinsic Module ISO_C_BINDING

Intrinsic functions C_LOC, C_FUNLOC, C_ASSOCIATED, C_SIZEOF and, Intrinsic subroutines C_F_POINTER, C_F_PROCPOINTER are offered by intrinsic module ISO_C_BINDING.

C_LOC gets pointer (C_PTR type) of C program corresponding to the argument.

C_FUNLOC gets function pointer (C_FUNPTR type) of C program corresponding to the argument.

C_ASSOCIATED compares pointer or function pointer of C program corresponding to the argument.

C_F_POINTER assigns data pointer of C program to pointer of Fortran.

C_F_PROCPOINTER assigns function pointer of C program to procedure pointer of Fortran.

C_SIZEOF gets size of the argument.

See online manual intrinsic(3) for details of these intrinsic procedures.

Example: Using intrinsic procedure by intrinsic module ISO_C_BINDING

```
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_PTR,C_INT,C_LOC
INTERFACE
 SUBROUTINE SUB(PTR) BIND(C)
   USE,INTRINSIC::ISO_C_BINDING,ONLY:C_PTR
   TYPE(C_PTR),VALUE::PTR
 END SUBROUTINE
END INTERFACE
TYPE(C_PTR)::PTR
INTEGER(C_INT),TARGET::TARGET
TARGET=2
PTR=C_LOC(TARGET) ! Set C program pointer of variable TARGET
CALL SUB(PTR)
END
```

C program:

```
#include <stdio.h>
void sub(k)
int *k;
{
printf ("K=%d \n",*k) ;
}
```

11.2.6 BIND Statement, BIND Attribute Specifier, Language Binding Specifier and Procedure Language Binding Specifier

When a language binding specifier is specified, data can interoperate between Fortran and C program.

When a procedure language binding is specified, procedure can interoperate between Fortran and C program.

The language binding specifier and procedure language binding specifier are following syntax.

```
BIND ( C [, NAME = scalar character constant expression] )
```

External name of language entity is decided by language binding specifier or procedure language binding specifier. The external name by the specifier is called a binding label.

If the language binding specifier or procedure language binding specifier is specified, the external name is the follows. However, value of NAME specifier is discarding leading and trailing blanks.

- If NAME specifier is not specified or length of NAME specifier value is zero, the binding label is the same as the name of the entity using lower case letters.
- If NAME specifier value has nonzero length, the binding label has the value. The case of letters in the binding label is significant.

If the language binding specifier of a BIND statement or attribute specifier in type declaration statement is specified, a variable or common block linkage interoperates with an external variable of C program.

If the language binding specifier is specified, the data entity has BIND attribute.

A procedure language binding specifier can specify in SUBROUTINE statement, FUNCTION statement or ENTRY statement, and the procedure interoperates with C program.

Example: Specify language binding specifier and procedure language binding specifier

```
MODULE MOD
  INTEGER,BIND(C,NAME='gvar01')::VAR
  INTEGER,BIND(C              )::GVAR02
END MODULE

USE MOD
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
INTERFACE
 SUBROUTINE SUB() BIND(C,NAME='Sub_proc')
 END SUBROUTINE
END INTERFACE
INTEGER(C_INT)::K1,K2
COMMON /COM/ K1,K2
BIND(C,NAME='gvar03':: /COM/
VAR=1
GVAR02=2
K1=3
K2=4
CALL SUB
END
```

C program

```
#include <stdio.h>
struct {int k1; int k2;} gvar03;
int gvar01;
int gvar02;

void Sub_proc(void)
{
printf ("gvar01=%d \n",gvar01) ;
printf ("gvar02=%d \n",gvar02) ;
printf ("gvar03.k1=%d \n",gvar03.k1) ;
printf ("gvar03.k2=%d \n",gvar03.k2) ;
}
```

11.2.6.1 BIND Statement

A BIND statement specify BIND attribute, and the statement has the following form.

```
Language binding specifier[ :: ] binding-entity-list
```

The binding-entity is a variable name or /common-block-name/. If a variable name is specified, the BIND statement must appear in declaration part of module.

If a NAME specifier specify in language binding specifier, then binding-entity-list must contain exactly one binding-entity.

Example: Interoperates with C program by BIND statement

```
MODULE MOD
  USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
  INTEGER(C_INT)::N
  INTEGER(C_INT)::K1,K2
  COMMON /COM/ K1,K2
  BIND(C):: N,/COM/
END MODULE

USE MOD
INTERFACE
 SUBROUTINE SUB() BIND(C)
 END SUBROUTINE
END INTERFACE
N=1
K1=2
K2=3
CALL SUB
END
```

C program

```
#include <stdio.h>
struct {int k1; int k2;} com;
int n;

void sub(void)
{
printf ("n=%d \n",n) ;
printf ("com.k1=%d \n",com.k1) ;
printf ("com.k2=%d \n",com.k2) ;
}
```

11.2.6.2 BIND Attribute by Attribute Specifier of Type Declaration Statement

A language binding specifier is specified in attribute specifier of type declaration statement, BIND attribute is specified.

A language binding specifier of type declaration statement can specify in declaration part of module. If a NAME specifier specify in language binding specifier, then variable list must contain exactly one variable.

Example: Specify language binding specifier of type declaration statement.

```
MODULE MOD
  INTEGER,BIND(C,NAME='global01')::VAR
  INTEGER,BIND(C                ):: GLOBAL02,GLOBAL03
END MODULE

USE MOD
INTERFACE
 SUBROUTINE SUB() BIND(C)
 END SUBROUTINE
END INTERFACE
VAR=1
GLOBAL02=2
GLOBAL03=3
CALL SUB
END
```

C program

```
#include <stdio.h>
int global01;
```

```
int global02;
int global03;

void sub(void)
{
printf ("global01=%d \n",global01) ;
printf ("global02=%d \n",global02) ;
printf ("global03=%d \n",global03) ;
}
```

11.2.6.3 Procedure Language Binding Specifier

A procedure language binding specifier specify following dummy argument list in SUBROUTINE statement, FUNCTION statement or ENTRY statement. When the procedure language binding is specified and the procedure doesn't have dummy argument list, parentheses enclose the dummy argument are not omissible.

Example: Specify procedure language binding specifier

```
SUBROUTINE SUB(K) BIND(C)
  USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
  INTEGER(C_INT)::K
 PRINT *,'K=',K
 END SUBROUTINE
 FUNCTION IFUN(K) BIND(C,NAME='fun') RESULT(R)
  USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
  INTEGER(C_INT)::R,K
  R=K
 END FUNCTION
```

C program

```
#include <stdio.h>
void sub(int *);
int fun(int *) ;
int  foo(void)
{
int n,k;
k=2 ;
sub(&k) ;
n=fun(&k) ;
printf (" n= %d \n",n) ;
return 0 ;
}
```

11.2.7 VALUE Statement, VALUE Attribute Specifier

A VALUE statement or VALUE attribute specifier of type declaration statement is specified, the dummy argument has VALUE attribute. If program calls procedure of VALUE attribute dummy argument, the procedure must have an explicit interface. When a dummy argument has VALUE attribute, the dummy argument is passed by value without follows:

- A character type of procedure without procedure language binding
- A component has POINTER attribute in derived type
- A component has ALLOCATABLE attribute in derived type
- OPTIONAL attribute

Example: Specify a VALUE attribute and VALUE statement

```
SUBROUTINE SUB(K) BIND(C)
  USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
  INTEGER(C_INT),VALUE::K
  PRINT *,'K=',K
```

```
 END SUBROUTINE
!
 USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
 INTERFACE
  SUBROUTINE FOO(K1,K2) BIND(C)
   USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT
   INTEGER(C_INT),VALUE::K1
   INTEGER(C_INT)      ::K2
   VALUE::K2
  END SUBROUTINE
 END INTERFACE
 INTEGER(C_INT)::N1,N2
 N1=1
 N2=2
 CALL FOO(N1,N2)
END
```

C program

```
void sub(int);
int  foo(int k1,int k2)
{
sub(k1) ;
sub(k2) ;
return 0 ;
}
```

11.2.8 Interoperability of Derive type and C language structure type

In case Fortran derived type is interoperable with structure type of C language, the following conditions should be met:

- Fortran derived type must have BIND Attribute.
- Fortran derived type must not be sequence type.
- Fortran derived type must not have Type parameter.
- Fortran derived type must not have EXTENDS attribute.
- Fortran derived type must not have type bound procedure part.
- Each component of Fortran derived type must be interoperable type as well as type parameter.
- Each component of Fortran derived type must not be pointer or allocatable.
- Fortran derived type must have same number of components as that of C language structure type.
- Each component of Fortran derived type must have component type which supports structure type of C language, interoperable type and type parameter.

11.3 Rules for Processing Fortran Procedure Names

In the object, the Fortran procedure name processed from source name. If a Fortran procedure name is called from C program, the name must conform to the Fortran name rules.

The following table lists the rules for processing Fortran procedure names.

Table 11.2 Rules for processing Fortran procedure names

Procedure name		Processing name
Main program name		MAIN__
External procedure name		external-procedure-name_
Block data program unit	The BLOCK DATA program with a name	block-data-program-unit-name_

Procedure name		Processing name
	The BLOCK DATA program without a name	_BLKDT__
Startup routine		main

Notes:

1. All procedure names are uniformly lowercase if compiler option -AU is not specified.
2. The startup routine is the routine to activate the Fortran system.

If compiler option -AU is specified, the processing names are same spell in source program.

The processing name of the procedure with procedure language binding specifier is the binding label.

Rules for Processing Fortran Procedure Names can be changed by compiler option -mldefault.

When the compiler option -mldefault is specified, the procedures with procedure language binding specifier, the procedures specified in CHANGEENTRY statement and the external procedures name specified in $pragma specifier are not effective.

The following table lists the rules for processing Fortran external procedure names when the -mldefault option is specified.

Table 11.3 Rules for processing Fortran external procedure names when the -mldefault option is specified

Compiler option -mldefault	Processing name
-mldefault=cdecl	external-procedure-name
-mldefault=frt	external-procedure-name_
Not specified	

Compiler option -ml changes rules for processing Fortran external procedure names of external procedure name statement. The following table lists rules for processing Fortran external procedure names.

Table 11.4 Rules for processing Fortran external procedure names specified in CHANGEENTRY statement

Compiler option -ml	Processing name
-ml=cdecl	external-procedure-name
-ml=frt	external-procedure-name_
Not specified	in accordance with specifying -mldefault option

The following table lists rules for processing Fortran external procedure names specified in $pragma specifier.

Table 11.5 Rules for processing Fortran external procedure names in $pragma specifier

$pragma specifier	Processing name
c (*prc*)	external-procedure-name

11.4 Correspondence of type with C

If data is passed between Fortran and other programs, data attributes must correspond. See Section "5.1 Internal Data Representation" for Fortran data types.

When data is passed between Fortran and C programs, it is necessary to make the type correspond.

See Section "Table 11.1 Types of C program and, types and kind type parameters of Fortran correspond" for the type that can be shared when the BIND attribute is specified.

The following table lists data type that Fortran and C when BIND attribute is not specified correspond.

Table 11.6 Data type that A and B correspond

Data Type	Fortran	C
One-byte logical	LOGICAL(1) L1	unsigned char L1 ;
Two-byte logical	LOGICAL(2) L2	short int L2 ;

Data Type	Fortran	C
Four-byte logical	LOGICAL(4) L4	int L4 ;
Eight-byte logical	LOGICAL(8) L8	long long int L8 ;
One-byte integer	INTEGER(1) I1	signed char I1 ;
Two-byte integer	INTEGER(2) I2	short int I2 ;
Four-byte integer	INTEGER(4) I4	int I4 ;
Eight-byte integer	INTEGER(8) I8	long long int I8 ;
Single -precision real	REAL(4) R4	float R4 ;
Double-precision real	REAL(8) R8	double R8 ;
Quadruple-precision real	REAL(16) R16	long double R16 ;
Single -precision complex	COMPLEX(4) C8	struct { float r,i;} C8 ;
Double-precision complex	COMPLEX(8) C16	struct { double r,i;} C16 ;
Quadruple-precision complex	COMPLEX(16) C32	struct { long double r,i; } C32 ;
Character	CHARACTER(10) S	char S [10] ;
Derived type	TYPE TAG INTEGER I4 REAL(8) R8 END TYPE TAG TYPE(TAG)::D	struct tag { int i4; double r8; } d;

11.5 Calling Procedures

If a Fortran procedure name is called from a C program or C function name is called from a Fortran program, the processing name must confirm. See Section "11.3 Rules for Processing Fortran Procedure Names" for processing Fortran procedure names.

The example of the calling procedure is shown as follows.

Example 1: Calling C function from Fortran program with the procedure language binding specifier.

Fortran program:

```
INTERFACE
  SUBROUTINE SUB() BIND(C)
  END SUBROUTINE
END INTERFACE
CALL  SUB( )
END
```

C program:

```
#include <stdio.h>
void sub( )
{
printf ("%s\n","** sub **");
}
```

The external name of Fortran calling procedure SUB is the binding label sub. Therefore, function name of C is sub.

Example 2: Calling Fortran procedure with the procedure language binding specifier from C program

C program:

```
void sub( );
int c_( )
{
sub( );
return 0;
}
```

Fortran program:

```
SUBROUTINE SUB() BIND(C)
PRINT *,'** SUB **'
END
```

The external name of Fortran subroutine SUB is the binding label sub. Therefore, subroutine SUB is able to interoperable.

Example 3: Calling C function from Fortran program without the procedure language binding specifier

Fortran program:

```
EXTERNAL  SUB
CALL SUB( )
END
```

C program:

```
#include <stdio.h>
void sub_( )
{
printf ("%s\n","** sub **");
}
```

In Fortran program, calling SUB are processed "sub_", so C function name must be "sub_".

Example 4: Calling Fortran procedure without the procedure language binding specifier from C program

C program:

```
void sub( );
int c_( )
{
sub( );
return 0;
}
```

Fortran program:

```
SUBROUTINE SUB( )
CHANGEENTRY :: SUB
PRINT *,'** SUB **'
END
```

In Fortran program, specify procedure name in CHANGEENTRY statement and specify compiler option -ml=cdecl to confirm processing procedure names.

11.5.1 Passing Control First to a C Program

If the program to which control is first passed is in C, the function name must be MAIN__. Do not use main. However, if the compiler option -mlcmain=main is specified, main can be used for the function name.

Example 1: Passing control first to a C program:

C program: a.c

```
void sub_( );
int MAIN__( )
{
```

```
sub_( );
return 0;
}
```

Fortran program: b.f95

```
SUBROUTINE SUB( )
PRINT *,'SUB'
END
```

Compile, link and execute:

```
$ fccpx -c a.c
$ frtpx a.o b.f95
$ ./a.out
SUB
$
```

Example 2: Passing control first to a C program (function name is main)

C program: a.c

```
void sub_( );
int main( )
{
sub_( );
return 0;
}
```

Fortran program: b.f95

```
SUBROUTINE SUB( )
PRINT *,'SUB'
END
```

Compile, link and execute:

```
$ fccpx -c a.c
$ frtpx -mlcmain=main a.o b.f95
$ ./a.out
SUB
$
```

11.5.2 Calling Standard C Libraries

When a standard C library is called from the Fortran program, there are two methods.

- The method of the specification of the procedure language binding specifier
- The method of no specification of the procedure language binding specifier

11.5.2.1 Method of the Specification of the Procedure Language Binding Specifier

A standard library of C can be called from the Fortran program.

Example: Calling a standard C library with the procedure language binding specifier.

```
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_NULL_CHAR
INTERFACE
  FUNCTION STRCMP(STR1,STR2) BIND(C)
   USE,INTRINSIC::ISO_C_BINDING,ONLY:C_INT,C_CHAR
   INTEGER(C_INT)::STRCMP
   CHARACTER(KIND=C_CHAR),DIMENSION(*)::STR1,STR2
  END FUNCTION
END INTERFACE
```

```
CHARACTER(LEN=5)::ST1,ST2
ST1='ABCD'//C_NULL_CHAR
ST2='ABCD'//C_NULL_CHAR
IF (STRCMP(ST1,ST2)==0)PRINT *,"same string"
END
```

In this example, the actual argument of the character type scalar corresponds to the dummy argument of the character type array.

Compile and execute:

```
$ frtpx a.f95
$ ./a.out
same string
$
```

11.5.2.2 Method of no Specification of the Procedure Language Binding Specifier

To calling standard C libraries from Fortran program directory, you must do following:

- Specify the standard C library name in CHANGEENTRY statement
- Specify compiler option -ml=cdecl

See Section "11.3 Rules for Processing Fortran Procedure Names", and be careful to passing argument. See Section "11.7 Passing and Getting Arguments" for details.

Example: Calling a standard C library without the procedure language binding specifier.

Fortran program:a.f95

```
INTEGER :: STRCMP
CHANGEENTRY :: STRCMP
CHARACTER(LEN=5) :: ST1,ST2
ST1 = 'ABCD\0' ; ST2 = 'ABCD\0'   ! Compiler option -Ae need to be specified.
IF (STRCMP(ST1,ST2)==0) PRINT *," same string "
END
```

Compile and execute:

```
$ frtpx a.f95 -ml=cdecl -Ae
$ ./a.out
  same string
$
```

11.6 Passing and Receiving Function Values

This section describes passing and receiving function values between Fortran and C program without the procedure language binding specifier. See "Table 11.1 Types of C program and, types and kind type parameters of Fortran correspond" for the type that can be shared when the procedure language binding is specified.

11.6.1 Passing Fortran Function Values to C without the Procedure Language Binding Specifier

The following table lists the methods for calling Fortran procedures from C programs for each type of function result that a Fortran program returns.

Fortran function value	How C accepts values	Example of values returned by C
INTEGER(1)	signed char	res = sub_();
INTEGER(2)	short int	
INTEGER(4)	int	

Fortran function value	How C accepts values	Example of values returned by C
INTEGER(8)	long long int	
REAL(4)	float	
REAL(8)	double	
REAL(16)	long double	
LOGICAL(1)	unsigned char	
LOGICAL(2)	short int	
LOGICAL(4)	long int	
LOGICAL(8)	long long int	
COMPLEX(4)	structure	
COMPLEX(8)		
COMPLEX(16)		
CHARACTER	void	sub_(&res, len); (*1)
Derived type	structure	res = sub_();

sub_: Called Fortran procedure name

res: Area for the function value returned. See Section "11.4 Correspondence of type with C" about the corresponding type

len: Character length of the function type. Then, Fortran receives character length as a value of eight byte integer type.

*1) The area of the return value of the procedure without the procedure language binding specifier is used the area of the actual argument. In this case, the address of the return value is put in the first argument and the length of the return value is put in the second argument. And the actual argument is put after the second arguments.

Example 1: Acceptance of an integer function value:

C program:

```
#include <stdio.h>
int ifun_();
int main( )
{
  int i;
  i = ifun_();
  printf("%d\n",i);
  return 0;
}
```

Fortran program:

```
FUNCTION IFUN()
INTEGER IFUN
IFUN = 100
END
```

Example 2: Acceptance of a double-precision real function value:

C program:

```
#include <stdio.h>
double r8fun_() ;
int main( )
{
  double r8;
  r8 = r8fun_() ;
  printf("%f\n",r8);
```

```
  return 0;
}
```

Fortran program:

```
FUNCTION R8FUN()
REAL(8) R8FUN
R8FUN= 12.2D+0
END
```

Example 3: Acceptance of a character function value:

C program:

```
#include <stdio.h>
void cfun_(char *, long long int);
int main( )
{
  char cha [11] ;
  cfun_(cha,10);
  printf("%s\n",cha);
  return 0;
}
```

Fortran program:

```
FUNCTION CFUN()
CHARACTER(LEN=*) CFUN
CFUN = '1234567890'
END
```

To accept the value of character function CFUN, set the address of the return area to the first argument, and set the length of function to the second argument. Then, Fortran receives length of character function CFUN as a value of eight byte integer type.

Example 4: Acceptance of a derived type function value:

C program:

```
#include <stdio.h>
struct tag
{ int i;
  double fr8; };
struct tag dfun_( );
int main( )
{
  struct tag d;
  d = dfun_( );
  printf("%d %f\n",d.i, d.fr8);
  return 0;
}
```

Fortran program:

```
FUNCTION DFUN()
TYPE TAG
  SEQUENCE
  INTEGER(4) I
  REAL(8) FR8
END TYPE
TYPE (TAG) ::DFUN
DFUN%I = 10
DFUN%FR8 = 1.2D+0
END
```

11.6.2 Receiving Function Values from C without the Procedure Language Binding Specifier

The following table lists the methods for calling C functions from Fortran programs without the procedure language binding Specifier for each type of function result that a C program returns.

Specifier for each type of function result that a C program returns.

C function value	How Fortran accepts value	Example of values returned by Fortran
void	Not accepted	CALL SUB()
signed char	INTEGER(1)	RES = SUB()
short int	INTEGER(2)	
int	INTEGER(4) or LOGICAL(4)	
long long int	INTEGER(8)	
float	REAL(4)	
double	REAL(8)	
long double	REAL(16)	
structure	Derived type	

SUB: Calling C function processed in sub_. See Section "11.3 Rules for Processing Fortran Procedure Names".

RES: Area for the function value returned.

Example 1: Acceptance of an integer function value:

Fortran program:

```
INTEGER(4) I,IFUN
I = IFUN()
PRINT *,I
END
```

C program:

```
int ifun_()
{
   return 100;
}
```

Example 2: Acceptance of a derived type function value:

Fortran program:

```
TYPE TAG
  INTEGER(4)
  REAL(8) FR8
END TYPE
TYPE (TAG) ::DFUN,D
D = DFUN()
PRINT *,D%I,D%FR8
END
```

C program:

```
struct tag
{int i;
  double fr8; } ;
struct tag dfun_()
{
 struct tag d ;
```

```
 d.i=100 ;
 d.fr8=1.2 ;
 return(d) ;
}
```

11.7 Passing and Getting Arguments

The passing and receiving arguments between procedures are the address in the default. If arguments are passed and received from Fortran program, see Section "11.2.1 Passing Arguments by Value".

If a character type argument is passed, the argument length as a value of eight byte integer type is also passed. The actual argument that shows the argument length exists only by the number of arguments of the character type be after all the specified arguments.

However, the argument length is not passed when the argument is the procedure with the procedure language binding specifier or VALUE attribute. The following sections explain how arguments pass data without the procedure language binding specifier.

11.7.1 Passing Arguments from Fortran with the Procedure Language Binding Specifier to C

Because Fortran passes actual arguments by address, the address is accepted if dummy arguments are declared in C. Specify VAL( ) intrinsic function or %val( ) specifier to pass the actual argument by value.

See Section "11.2.1 Passing Arguments by Value" for details.

Example: Passing integer and real type arguments:

Fortran main program:

```
PROGRAM FORTRAN_MAIN
WRITE(*,*) '*** START ***'
J=10
K=20
L=30
CALL FUN1(J,K,L)
F=10.0
CALL FUN2(F)
WRITE(*,*) '*** FMAIN END ***'
END
```

C:

```
#include <stdio.h>
void fun1_(j,k,l)
int *j,*k,*l;
{
 printf("J:%d K:%d L:%d\n",*j,*k,*l);
}

void fun2_(f)
float *f;
{
 printf("F:%f\n",*f);
}
```

11.7.2 Passing Arguments from C to Fortran without the Procedure Language Binding Specifier

Because Fortran accepts dummy arguments by address in the default, the address is passed if actual arguments appear in C. Specify VALUE attribute for dummy argument to get the actual argument by value. See Section "11.2.2 Getting Arguments by Value" for details.

Example 1: Passing integer type arguments:

C program (MAIN__):

```
#include <stdio.h>
int MAIN__()
{
  int fun_();
  int i,j,k;
  i=10;
  j=20;
  k=fun_(&i,&j);
  printf(" k:%d\n",k);
  return 0;
}
```

Fortran program:

```
INTEGER FUNCTION FUN(X,Y)
INTEGER(4) X,Y

WRITE(*,*) '**** FUNC1 **** '
FUN =X+Y
END
```

Example 2: Passing character type arguments:

C program (MAIN__):

```
#include <stdio.h>
int MAIN__( )
{
   void fun_(char *, long long int, char *, long long int);
   char res[3],ch;
   ch = 'o';
   res[2] = 0;
   fun_(res,2,&ch,1);
   printf(" res:%s\n",res);
   return 0;
}
```

Fortran:

```
CHARACTER(LEN=*) FUNCTION FUN(CH)
CHARACTER(LEN=*) CH
FUN = CH//'k'
END
```

Note:

Character type function FUN has one character type argument. If this function is called from a C program, four arguments are passed. The first argument is the address of the area for the function value. The second argument is the length of the function value. The third argument is the address of the argument ch. The fourth argument is the length of ch. Then, Fortran receives result length and argument length of character type function FUN as a value of eight byte integer type.

11.8 Passing Using External Variables

- The method of the specification of the BIND attribute.

- The method of no specification of the BIND attribute.

11.8.1 Passing External Variables with BIND Attribute

When COMMON block or module variable with BIND attribute, this COMMON block or variable can associate to the external variable. The external name is the binding label of the variable or COMMON block.

Example 1: Associating the external variable with BIND attribute

```
  MODULE MOD
    INTEGER,BIND(C)::INT_VARIABLE
           ! Associating the external variable int_variable in C program
  END
```

Example 2: Passing the named COMMON block with BIND attribute

Fortran program:

```
INTERFACE
 SUBROUTINE SUB() BIND(C)
 END SUBROUTINE SUB
END INTERFACE
INTEGER I,J
COMMON /EXT/I,J
BIND(C)::/EXT/
I = 1
J = 1
CALL SUB( )
END
```

C program:

```
#include <stdio.h>
struct tag { int i,j; } ext;
void sub( )
{
printf("i=%d j=%d\n",ext.i,ext.j);
}
```

Passed COMMON block EXT with BIND attribute.

11.8.2 Passing External Variables without BIND attribute

Without BIND attribute, the data in Fortran COMMON block can be passed as the external structure type variable in C program.

A COMMON block name of Fortran and an external variable of C program should be the following correspondences.

Specification of COMMON block	External variable name of structure of C
Named COMMON block	COMMON block name_
Unnamed COMMON block	_BLNK__

The COMMON block name is uniformly lowercase if compiler option -AU is not specified. If compiler option -AU is specified, the COMMON block name is same spell in source program.

Example 1: Named common block without BIND attribute

Fortran program:

```
COMMON /EXT/I,J
I=1
J=2
CALL SUB()
END
```

C program:

```
#include <stdio.h>
struct tag {int i,j;} ext_;
void sub_()
{
  printf("i=%d j=%d\n",ext_.i,ext_.j);
}
```

Note:

To reference common block EXT, change the name to lowercase and add an underscore to the name.

Example 2: Blank common block without BIND attribute

Fortran program:

```
COMMON //X,Y
X=1.0
Y=2.0
CALL SUB()
END
```

C program:

```
#include <stdio.h>
struct tab {float x,y;}_BLNK__ ;
void sub_()
{
printf("x=%f y=%f\n",
_BLNK__.x, _BLNK__.y) ;
}
```

11.9 Passing Using Files

If data is passed by file, the file must be closed. To close files in Fortran and C programs, use the following instructions:

Fortran	C
CLOSE statement	close (2)

11.10 Array Storage Sequence

Fortran and C differ in the order in which they store elements in arrays of two or more dimensions.

Therefore, be careful when passing arrays between Fortran and C.

The following shows the different storage sequences in Fortran and C arrays.

(Array in Fortran)	(Array in C)
INTEGER(4) K(2,3)	int k[2][3];
Storage sequence in Fortran	Storage sequence in C
K(1,1)	k[0][0]
K(2,1)	k[0][1]
K(1,2)	k[0][2]
K(2,2)	k[1][0]
K(1,3)	k[1][1]
K(2,3)	k[1][2]

11.11 Passing a Character String to a C Program

The null character "\0" normally is not appended to the constants of Fortran character type. If a function such as strlen (standard C library function) is called, you must do one of the followings:

- Add the intrinsic module C_NULL_CHAR (null character) of ISO_C_BINDING to the end of character strings.
- Append a null character (CHAR(0)) to the end of constants. Then, compiler option -Ae must be specified.
- Specify the compiler option -Az when the constants of character type is specified in the argument.

Example 1: Passing and receiving the character string using BIND attribute and C_NULL_CHAR

Fortran program:

```
USE,INTRINSIC::ISO_C_BINDING,ONLY:C_NULL_CHAR
INTERFACE
  SUBROUTINE SUBA(STR) BIND(C)
   USE,INTRINSIC::ISO_C_BINDING,ONLY:C_NULL_CHAR,C_CHAR
   CHARACTER(KIND=C_CHAR)::STR(*)
  END SUBROUTINE
END INTERFACE
CALL SUBA('12345'//C_NULL_CHAR)  ! Add null character to the end of the character strings
END
```

C program:

```
#include <stdio.h>
#include <string.h>
void suba(char *str)
{
int i,len;
printf("len=%d,[%s]\n",len=strlen(str),str);
for (i=0; i < len; i++)
{
printf("0x%02x ",*str++);
}
putchar('\n');
}
```

Example 2: Passing and receiving the character strings added NULL character not using BIND attribute.

Fortran program:

```
! CALL SUBA('12345')
  CALL SUBA('12345\0') ! Add \0 to the end of the character strings.
  END
```

C program:

```
#include <stdio.h>
#include <string.h>
void suba_(char *str)
{
int i,len;
printf("len=%d,[%s]\n",len=strlen(str),str);
for (i=0; i < len; i++)
{
printf("0x%02x ",*str++);
}
putchar('\n');
}
```

Note:

Compiler option -Ae must be specified for Fortran program.

11.12 Return Value of Executable Program

When the Fortran program is linked with C/C++ language program, the return value of executable program is decided as follows.

- In the following cases, the return value of Fortran is set the return value of executable program. See Section "3.6 Execution Command Return Values".
  - The program to which control is first passed is Fortran program.
- In the following cases, the return value of C/C++ is set the return value of executable program.
  - The program to which control is first passed is C/C++ program.

11.13 Notes of the Compiler Option -Az

If the compiler option -Az is to add \0(Null character) to the end of character strings. However, a part of execution is not guaranteed by the program described by the specification before FORTRAN77.

Example: The compiler option -Az

```
CALL SUB('12')
END
SUBROUTINE SUB(I)
INTEGER(4) I
IF (I.NE.'12') PRINT *,'ok'
END
```

Please change the CALL statement of the above-mentioned program as follows.

```
CALL SUB('12  ')
```

11.14 Restrictions on Linking with C Language Programs

The following restrictions and notes apply to linkage with C programs. And, see Section "9.19 Attentions for Mixed Language Programming" for notes combined with the optimization functions.

- With BIND attribute, the language entities of Fortran and C should be able to be interoperable.
- The binding label with the BIND attribute and the other processing Fortran procedure names must not be same name.

- When C program is called as a function reference, the function must correspond as shown in "11.6 Passing and Receiving Function Values".
- When a Fortran program is called from a C program, the Fortran program must not be a main program.
- If the main program is a C program, the name must be "MAIN__". However, if the compiler option -mlcmain=main is specified, main can be used for the function name.
- When using an interrupt-associated Fortran function, do not use signal(2) or clock(3C).
- When signal(2) or clock(3C) function is used in C language programs, the -i option must be specified at execution time to suppress interrupt processing by Fortran objects and followings are restrictions.
  - Specifying time limit to continue execution by -t option at execution time
  - Service function "ALARM"
  - Service function "SIGNAL"
- Always close nonstandard files before calling the other language.
- A CALL statement with an alternate return is valid only when calling Fortran subroutines.
- A subroutine called from a C program must not contain a RETURN statement with an integer expression or an aster of dummy argument. Change such a subroutine to an integer function.
- Fortran procedure must not invokes setjmp or longjmp in C program.
- If C program invokes setjmp or longjmp, that procedure must not cause any Fortran procedure to be invoked.
- When a Fortran procedure is called as a signal processing procedure, the signal processing procedure can refer only the variable that have VOLATILE attribute.
- Fortran procedures are not allowed to call C language function with a variable number of arguments.
- If Fortran variable associates with C external variable (see Section "11.8 Passing Using External Variables") and the C external variable has an initial value, the boundary of Fortran variable is changed to the boundary of the C external variable then warning message is output by linker.
- The variable of the language entity C_PTR type of the intrinsic module ISO_C_BINDING cannot debug using the debugger.
- If C++ program invokes exception processing (try, catch or throw), that procedure must not cause any Fortran procedure to be invoked.
- See "Notes on Interlanguage Linkage" in the "C++ User's Guide" for how to link C++ program.

Chapter 12 Multiprocessing

This chapter describes how parallel processing is performed for a Fortran program in this processor.

12.1 Overview of Multiprocessing

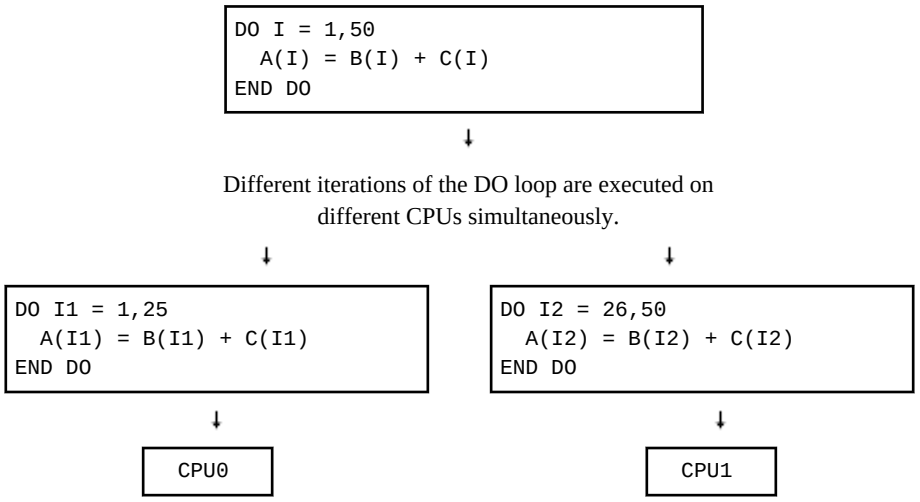
This section outlines the features of parallel processing and parallelization for Fortran in this system.

12.1.1 Parallel processing

Although parallel processing has a broad meaning, the parallel processing mentioned in this section means to perform a single program simultaneously using multiple CPUs working independently. In this section, parallel processing does not mean "multi job", which means running multiple programs at the same time.

The following figure illustrates parallel processing using multiple CPUs simultaneously.

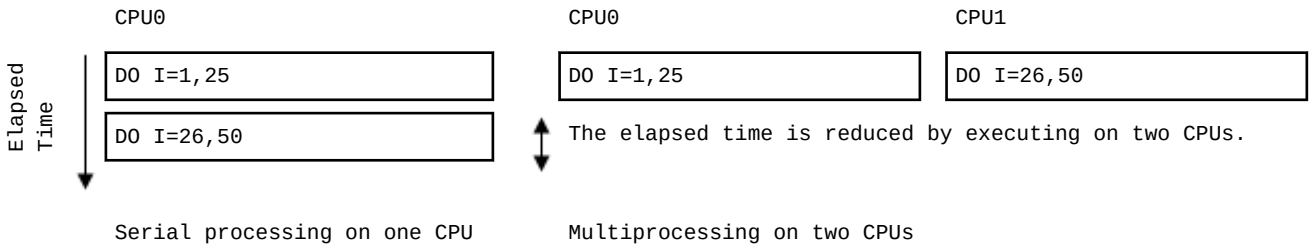
Figure 12.1 Multiprocessing



12.1.2 Effect of Multiprocessing

The aim of parallel processing is to reduce the elapsed execution time of a program using multiple CPUs at the same time. For example, as shown in "Figure 12.1 Multiprocessing", ideally the elapsed time of the program is halved by splitting one DO loop into two and performing them simultaneously ("Figure 12.2 Reducing elapsed time by using multiprocessing").

Figure 12.2 Reducing elapsed time by using multiprocessing



Although the elapsed time of a program can be reduced using parallel processing, the CPU time of the program may not be reduced. This is because, even though a number of CPUs are used in parallel processing, the total CPU time used may be the same as the CPU time used in serial processing, or may be increased because of the overhead to perform parallel processing.

However, in general, the performance of a program is evaluated by the elapsed time rather than the CPU time. If a program first executed serially is then executed in parallel, the total CPU time may be longer because of the overhead required for parallel processing.

12.1.3 Requirements for Effective Multiprocessing

A computer environment that can use multiple CPUs at one time is required in order for parallel processing to reduce elapsed time. Although programs generated by this compiler can run on hardware with a single CPU, if this is the case, a reduced elapsed time cannot be expected. Even if the hardware has multiple CPUs, a reduced elapsed time may not be expected when the CPU time is consumed by other jobs. This is because the opportunity to assign multiple CPUs at the same time to a program performing parallel processing may decrease.

In other words, for maximum benefit, parallel processing must be executed in an environment with multiple CPUs and sufficient capacity to assign multiple CPUs to the program.

Further, to minimize the relative rate of overhead due to parallel processing, the program should have either a large number of loop iterations or a large number of instructions within a loop.

12.1.4 Features of Parallel Processing in this Compiler

This compiler provides automatic parallelization and parallelization based on the OpenMP specifications.

The automatic parallelization is a feature of the compiler to parallelize the program automatically. Automatic parallelization is applied only for statements in DO loops and array operations for which the compiler has recognized that parallelization can be applied, but it does not require additional development.

This compiler also supports optimization control lines, which promote automatic parallelization. Information received from optimization control lines can be used to generate efficient object code for automatic parallelization.

To apply automatic parallelization, it is not necessary to update source code that has been designed for serial processing. For this reason, the source code is easily reusable in another system, as long as it conforms to the standard FORTRAN requirements.See Section "12.2 Automatic Parallelization" for information on automatic parallelization.

The compiler also allows OpenMP parallelization. See Section "12.3 Parallelization by OpenMP Specifications" for information on parallelization in the OpenMP specification.

12.2 Automatic Parallelization

This section explains the automatic parallelization provided by this compiler.

12.2.1 Compilation and Execution

The section explains how to compile and run a program that uses automatic parallelization.

12.2.1.1 Compilation

To activate automatic parallelization, specify the compiler option -Kparallel.

12.2.1.1.1 Compiler Option for Automatic Parallelization

The meaning and format of the automatic parallelization options are described below. See Section "2.2 Compiler Options" for more information on each option.

-Kparallel[,reduction,instance=*N*,array_private,parallel_iteration=*N*,dynamic_iteration,ocl,optmsg=2,
independent=*proc_name*,loop_part_parallel,region_extension]

parallel

Automatic parallelization is performed. It automatically activates the -O2 and -Kregion_extension option. If the -H or -Eg option is set, the -Kparallel option will be ignored.

reduction

This option is effective when it is specified with the -Kparallel option. The reduction optimization (refer to Section "12.2.3.1.8 Loop Reduction") is performed. This optimization may introduce differences in precision. Diagnostic messages output at compilation identify whether or not this optimization was performed.

The reduction optimization is also performed when -Kparallel and -Keval have been specified at the same time.

instance=*N*

This option is effective when it is specified with the -Kparallel option. The compiler generates an object program assuming that the runtime thread number is *N*. Specifying this option allows the compiler to skip calculating the iteration count of DO loops, improving the performance of the compilation.

array_private

This option is effective when it is specified with the -Kparallel option. It privatizes arrays in a loop that can be privatized.

parallel_iteration=*N* 1 <= *N* <= 2147483647

This option is effective when it is specified with the -Kparallel option. It directs that only loops determined at compilation to have an iteration count exceeding *N* will be parallelized.

dynamic_iteration

This option is effective when it is specified with the -Kparallel option. It directs to dynamically select a loop to be parallelized considering the number of threads, when both the inner loop and the outer loop within a nested loop are candidates.

ocl

This option is effective when it is specified with the -Kparallel option. It directs to activate the optimization control lines that are used for the automatic parallelization (see Section "12.2.3.2 Optimization Control Line").

optmsg=2

Messages are output that indicate that optimizations, such as automatic parallelization, SIMD optimization, and loop unrolling, have been performed.

independent=*proc_name*

This option is effective when it is specified with the -Kparallel option. It directs that the procedure specified in the pgm_nm argument always performs as a serial process, even if the procedure reference is specified within a parallelized DO loop. This means that DO-loops with procedure references will be subject to automatic parallelization.

loop_part_parallel

This option is effective when it is specified with the -Kloop_fission option and the -Kparallel option. When a loop contains statements to which parallelization can be applied and statements to which parallelization cannot be applied, the loop is divided into two or more loops and automatic parallelization is applied to one of the loops. This optimization is applied to the innermost loop.

region_extension

Parallelization overhead is reduced by expanding parallelization.

This option is automatically set when the -Kparallel option is specified.

12.2.1.2 Execution Process

When running a program compiled with automatic parallelization, the number of threads can be specified using the environment variable PARALLEL or OMP_NUM_THREADS. The stack area size for each thread can be specified using the environment variable THREAD_STACK_SIZE or OMP_STACKSIZE. It is also possible to specify a synchronization scenario using the environment variable FLIB_SPINWAIT or OMP_WAIT_POLICY.

The other procedures are the same as those used in serial processing.

12.2.1.2.1 Environment Variable PARALLEL

The environment variable PARALLEL specifies the number of threads performed at the same time. See Section "12.2.1.2.2 Number of Threads" for details on determining the number of threads.

12.2.1.2.2 Number of Threads

The priority of the factors that determine the number of threads executed concurrently is as follows.

1. The value specified in the environment variable PARALLEL (Note)
2. The value specified in the environment variable OMP_NUM_THREADS
3. Number of CPUs that can be used with jobs
4. 1 thread

   (Note)

   If the -Kopenmp option has been specified at the linkage phase of the compilation and the environment variable FLIB_FASTOMP is TRUE, the value of the environment variable PARALLEL must be the same as the value of the environment variable OMP_NUM_THREADS. When the value does not match, a message will be output, and smaller value will be used for the number of threads.

The number determined above is compared with the available maximum CPU number, and smaller number is used as the number of threads.

Refer to Section "D.1 Management of the CPUs resource" for information on the number of CPUs that can be used with jobs.

The maximum number of CPUs is determined in the following way.

- For jobs:

  Number of CPUs that can be used with jobs

- Not for job:

  Number of CPUs in the system

See Section "12.3.1.2.2 Environment Variable for OpenMP Specifications" for information on the environment variable OMP_NUM_THREADS.

12.2.1.2.3 Environment variables THREAD_STACK_SIZE and OMP_STACKSIZE

THREAD_STACK_SIZE

Specify the stack size (in Kbytes) for each thread using the environment variable THREAD_STACK_SIZE.

When the environment variable OMP_STACKSIZE is also specified, the greater value of the two is used as the stack area size for each thread. See Section "12.2.1.2.4 Stack Size on Execution" for information on stack area.

OMP_STACKSIZE

The stack area size for each thread can be specified using the environment variable OMP_STACKSIZE in byte, Kbytes, Mbytes, or Gbytes.

When the environment variable THREAD_STACK_SIZE is also specified, the greater value of the two is used as the stack area size. See Section "12.2.1.2.4 Stack Size on Execution" for information on stack area.

12.2.1.2.4 Stack Size on Execution

Local scalar variables within a parallelized DO loop (except the automatic data object when the compiler option -Knoautoobjstack is in effect) are allocated in the stack area for each thread. Allocate sufficient stack area, when there are many variables of this type.

The stack size for each thread is the same as the stack size for the process. However, if the maximum stack size for the process is larger than the value calculated by the following expression:

min(the size of available memory in this system, the size of a virtual memory) / (the number of threads)

the system will determine the stack area size and allocate it as follows:

However, the assumed value does not guarantee that the system works correctly. It is recommended that an appropriate value for maximum process stack size be specified manually. The maximum process stack area size can be specified using the bash built in command "ulimit". Note that the maximum virtual memory size can also be confirmed using ulimit (bash built-in command).

```
S = (M/T)/5
```

S: the stack size for each thread (byte)
M: the smaller size of available memory in this system and a virtual memory (byte)
T: the number of threads

See Section "12.2.1.2.2 Number of Threads" for details. When the stack area size for each thread needs to be specified, use the environment variable THREAD_STACK_SIZE or OMP_STACKSIZE.

12.2.1.2.5 The Process of Waiting Threads

The standby threads in a serial process can be controlled with the environment variable FLIB_SPINWAIT or OMP_WAIT_POLICY.

FLIB_SPINWAIT value:

unlimited	:	Uses spin wait.
0	:	Uses suspending wait.
<n>s	:	Uses spin wait until after <n> seconds, and uses suspending wait.
<n>[ms]	:	Uses spin wait until after <n> milliseconds, and uses suspending wait.

<n> is integer value that must be positive.

The default is unlimited.

Spin waiting is a technique to wait while consuming CPU time. Suspend waiting is a technique to wait without consuming CPU time. Since spin wait is more efficient in parallel execution, "unlimited" is suitable for reducing elapsed time. On the other hand, "0" is suitable for reducing CPU time.

OMP_WAIT_POLICY value:

ACTIVE	:	Uses spin wait.
PASSIVE	:	Uses suspending wait.

The default is "ACTIVE".

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

The settings of environment variable FLIB_SPINWAIT will be valid when environment variable FLIB_SPINWAIT is specified.

12.2.1.2.6 Notes when setting time limits

Operation may not be correct when time limits are set with the limit(1) command. If operation is not correct, use the Fortran exception handling process, the runtime option -Wl, -i to invalidate it. Alternatively, use the runtime option -Wl,-t to set the time limit.

12.2.1.2.7 Notes when using service routines

- ALARM Service Function
  It may not work correctly. It is only for a program using serial processing.
- KILL and SIGNAL Service Functions
  It may cause a deadlock. It is only for a program using serial processing.
- FORK Service Function
  It may not work correctly. It is only for a program using serial processing.
- SYSTEM, SH and CHMOD Service Functions
  Memory usage doubles and performance may deteriorate. Use only with programs using serial processing.

12.2.1.2.8 CPU Binding for Thread

CPU Binding for Thread differs between job execution and non-job execution.

When executing program on non-job.

The thread is not bound to a CPU. But you can control CPU Binding for Thread using environment variable `FLIB_CPU_AFFINITY`.

The environment variable FLIB_CPU_AFFINITY

Threads are bound to CPUs in order of the specified cupid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid shall be separated by comma (',') or space(' ').

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. (0<=*cpuid1*<CPU_SETSIZE)
cpuid2: cpuid of the end of the range. (0<=*cpuid2*<CPU_SETSIZE)
inc: increment (1<=*inc*<CPU_SETSIZE)

In addition, it is necessary to be the following.

```
cpuid1<=cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The cpuid can be used the above-mentioned value. However, cpuid which can actually be used becomes only within the limits of CPU affinity of the process at the time of an execution start.

See CPU_SET(3) about details of CPU_SETSIZE.

Examples of using environment variable FLIB_CPU_AFFINITY are shown below.

Example 1:

```
$ export FLIB_CPU_AFFINITY="0,2,1,3"
```

The thread is bound to CPU in order of 0, 2, 1, and 3.
When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 2:

```
$ export FLIB_CPU_AFFINITY="0-7"
```

The thread is bound to CPU in order of 0, 1, 2, 3, 4, 5, 6 and 7.
When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

Example 3:

```
$ export FLIB_CPU_AFFINITY="0-7:2"
```

The thread is bound to CPU in order of 0, 2, 4 and 6.
When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 4:

```
$ export FLIB_CPU_AFFINITY="0-4:2,1,7"
```

The thread is bound to CPU in order of 0, 2, 4, 1 and 7.
When the number of threads is six or more, it is repeatedly used f from the beginning of the list.

When executing program on job.

When thread is not bound to CPU, you may control CPU Binding with FLIB_CPU_AFFINITY as executing on non-job.

When thread is bound to CPU on job, the environment variable FLIB_CPU_AFFINITY becomes invalid.

For details of CPU Binding, see Section "D.2 CPU Binding".

12.2.1.3 Example of Compilation and Execution

Example 1:

```
$ frtpx -Kparallel,reduction,ocl test1.f
$ ./a.out
```

Example 1 compiles the source program using the reduction optimization while enabling optimization control lines. This program performs using a single thread.

Example 2:

```
$ frtpx -Kparallel -Koptmsg=2 test2.f
Fortran diagnostic messages: program name (main)
   jwd5001p-i "test2.f", line 2: DO loop with DO variable 'i' is parallelized.
$ export PARALLEL=2
$ ./a.out
$ export PARALLEL=4
$ ./a.out
```

Example 2 compiles specifying the -Koptmsg=2 option to confirm that automatic parallelization is performed. By specifying 2 for the environment variable PARALLEL, the program executes with two threads. The next step executes the program with four threads, by specifying 4 for PARALLEL.

12.2.2 Tuning of parallelized programs

A Sampling feature can be used to improve the performance of a parallelized program. The sampling feature helps to identify the statements in a program that are expensive. Designing a program to perform expensive statements simultaneously can improve its performance. On the Job Operation Software environment, use the sampling feature provided by the Programming Workbench to monitor the execution time.

For other environments, use the sampling feature provided by the profiler to monitor the execution time. To collect information about the parallel program using the sampling feature, specify an environment variable on executing the program.

See the "Programming Workbench User's Guide" for information on the programming workbench. See the "Profiler User's Guide" for information on the Profiler.

12.2.3 Details of Automatic Parallelization

This section describes optimization control lines, which are used to implement automatic parallelization. It also describes the points the points to note when using automatic parallelization.

12.2.3.1 Automatic Parallelization

This section describes the automatic parallelization.

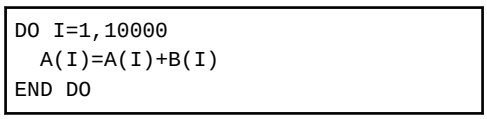
12.2.3.1.1 Targets for Automatic Parallelization

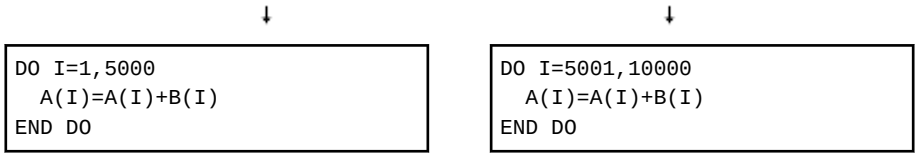
Automatic parallelization is performed for DO loops (including nested DO loops) and array statements (array operation, array assignments) in the Fortran source code.

12.2.3.1.2 What is Loop Slicing?

Automatic parallelization may slice a loop into several pieces. The elapsed execution time is reduced by executing the loop slices in parallel. The following figure illustrates the image of the loop slicing.

Figure 12.3 Loop slicing

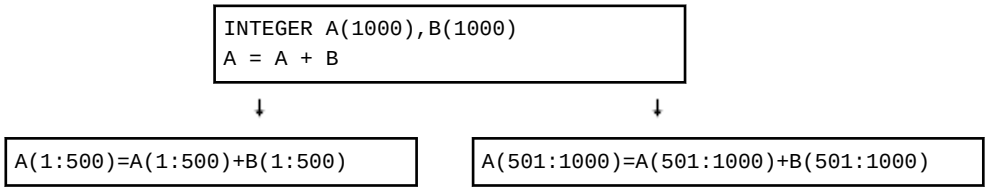




12.2.3.1.3 Array Operations and Automatic Parallelization

In addition to DO loops, automatic parallelization is also performed on array statements (array operation, array assignments.) The following figure illustrates automatic parallelization on array operation.

Figure 12.4 Automatic parallelization of a statement with an array operation



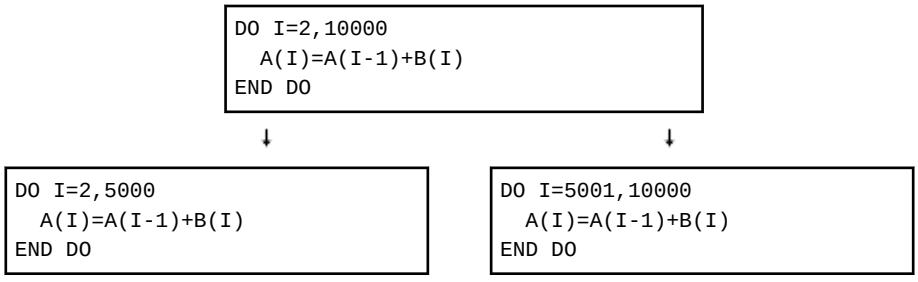
12.2.3.1.4 Automatic Loop Slicing by the Compiler

The system parallelizes a loop if the order of data reference will be the same as with serial execution, and if the system is certain that the result of the parallel loop will be the same as if the loop were processed serially.

"Figure 12.5 A loop that is not a candidate for loop slicing" shows an example where automatic loop slicing is not applied. In this DO loop, the value of the array element A(5000), which is defined when the loop counter I is 5000, is referenced when I is 5001. If loop slicing is performed for this DO loop, however, the element A(5000) may be referenced by the other loop before its value is defined, causing an error.

"Figure 12.5 A loop that is not a candidate for loop slicing" Sample DO loop that is not supported by loop slicing

Figure 12.5 A loop that is not a candidate for loop slicing

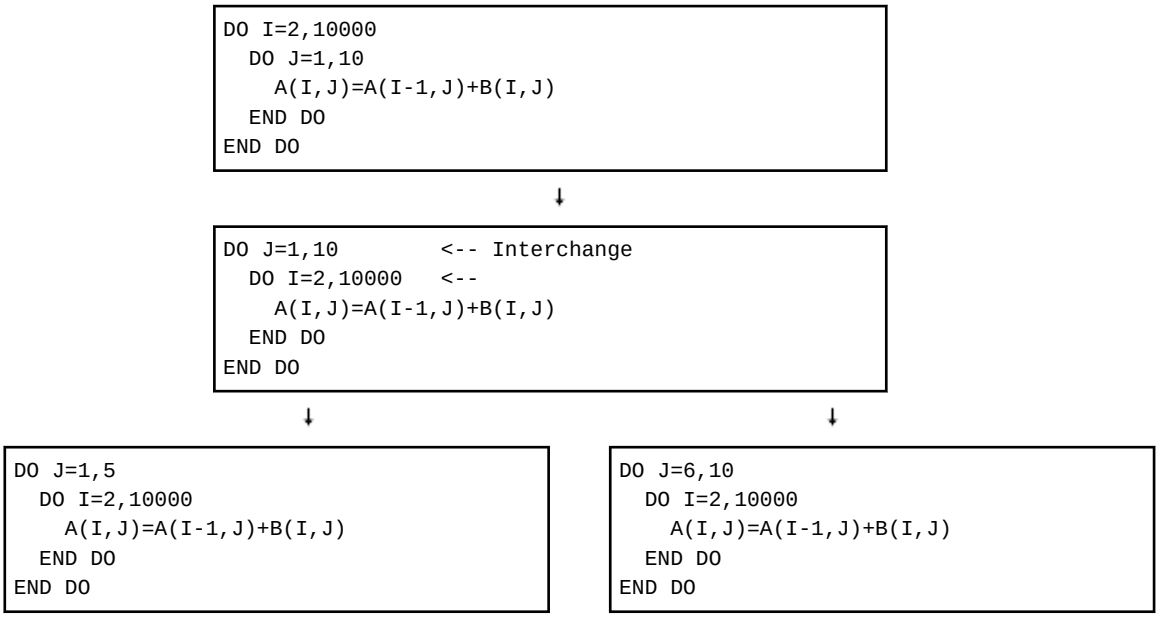


12.2.3.1.5 Loop Interchange and Automatic Loop Slicing

When nested loops are sliced, the system attempts to parallelize the outermost loop if it can. Therefore, the system selects the loop that can be sliced most cheaply and interchanges it with the outermost loop. The purpose of this is to reduce the overhead of multiprocessing and improve the execution performance.

"Figure 12.6 Loop interchange in nested DO loops" illustrates the loop slicing after performing loop interchange on a nested DO loop. Before performing parallelization for the DO loop variable "J", loop interchange is performed so that the parallelization overhead is reduced.

Figure 12.6 Loop interchange in nested DO loops



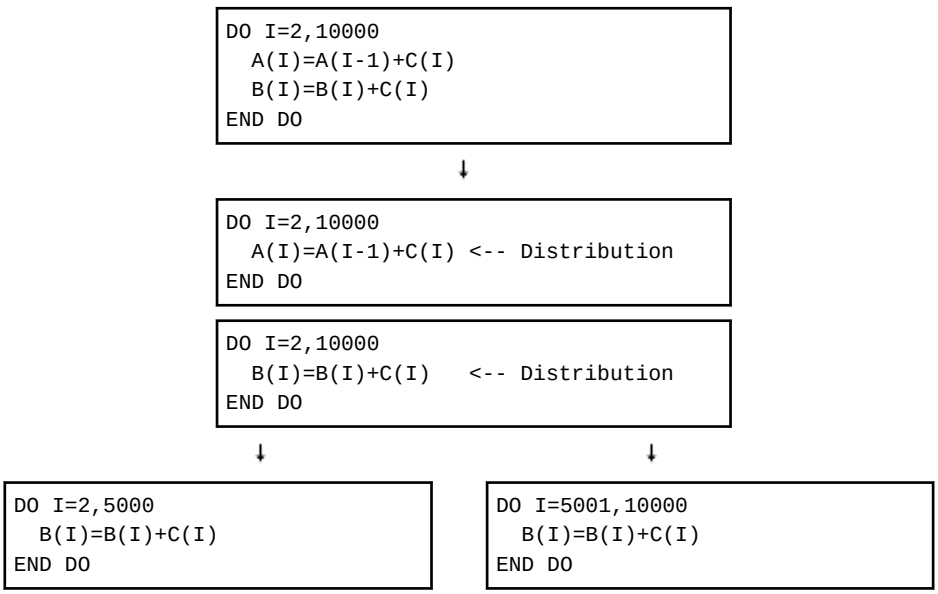
12.2.3.1.6 Loop Distribution and Automatic Loop Slicing

In the DO loop in "Figure 12.7 Separation of DO loop and loop slicing", loop slicing cannot be performed for array A because the definition or order of data references would be different from the order of data references in serial execution. However, loop slicing can be performed for array B, because the order of parallel data references is the same as the order of serial data references. In this case, the statement where array A is defined and the statement where array B is defined are separated into two loops, and loop slicing is performed on the loop where array B is defined.

The partial automatic parallelization by the distribution of loop is performed when the -Kloop_part_parallel option is effective and the compiler assumes that the distribution of loop is effective.

The following figure illustrates separating the statements into DO loops and performing loop slicing.

Figure 12.7 Separation of DO loop and loop slicing

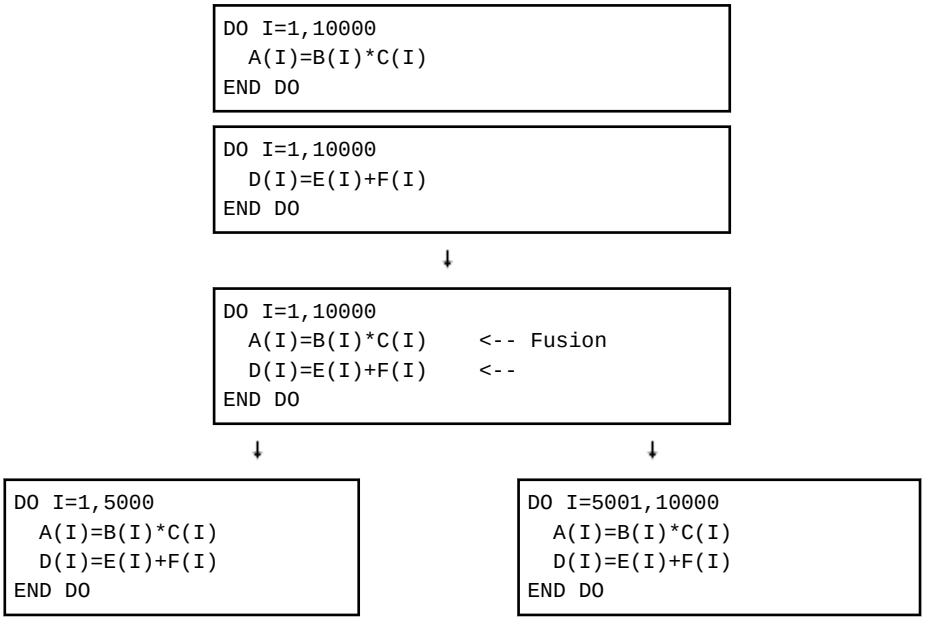


12.2.3.1.7 Loop Fusion and Automatic Loop Slicing

"Figure 12.8 Fusion of DO loops and loop slicing" contains two DO loops with the same iteration count. In this case, combining the loops into one can reduce the overhead of both parallelization and loop control.

The following figure illustrates the loop slicing after loop fusion.

Figure 12.8 Fusion of DO loops and loop slicing



12.2.3.1.8 Loop Reduction

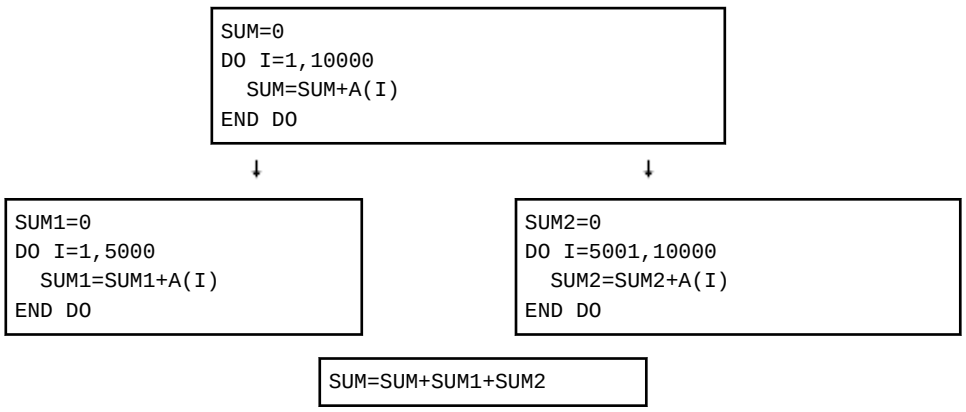
If both the compiler options -Kparallel and -Kreduction are specified at the same time, the automatic parallelization loop reduction will be performed. This performs loop slicing by changing the order of operations such as addition and multiplication if they are interchangeable. However, this may affect precision.

Loop reduction is performed only when the DO loop includes the following operations:

- SUM
  S=S+A(I)
- PRODUCT
  P=P*A(I)
- DOT PRODUCT
  P=P+A(I)*B(I)
- MIN
  X=MIN(X,A(I))
- MAX
  Y=MAX(Y,A(I))
- OR
  N=N.OR. A(I)
- AND
  M=M.AND.A(I)

The following figure shows an example of loop reduction and loop slicing.

Figure 12.9 Loop reduction



12.2.3.1.9 Restrictions on Loop Slicing

Loop slicing does not support the following DO loops:

- If it is unlikely that parallelization will reduce elapsed time.
- If operation types not supported by loop slicing are included
- If references to external procedures, internal procedures, or module procedure are included When the form of the DO-loop is complex
- Input-output statements or intrinsic subroutines are included
- Data definition or order of data references order may differ from serial execution
- If it is unlikely that parallelization will reduce elapsed time

1. It can be forecast that the elapsed time would not be reduced in the case of loop slicing.

   When the iteration counts of a DO loop is small or there are few statements within the DO loop, the parallelization overhead introduced by loop slicing may deteriorate performance. The compiler will not perform loop slicing when performance improvement cannot be expected for the DO loop.

   The following figure shows an example of a DO loop with a small iteration count and few statements.

   Figure 12.10 Small loop iteration count and small number of operations

   ```
   DO I=1,10               ! Not parallelized
     A(I)=A(I)+B(I)        ! because of small loop iteration count and
                           ! small number of operations
   END DO
   ```

2. The loop contains operations on types, which are not supported in loop slicing.

   Loop slicing does not support DO loops or inner DO loop with the following types of operations:

   - Character type
   - Derived type

3. References to external procedures, internal procedures, or module procedure are included.

   Loop slicing does not support DO loops containing a reference to an external procedure, internal procedure, or module procedure. It is also unsupported when an inner DO loop includes the following operation types.

   However, optimization control lines may be used to promote parallelization. See Section "12.2.3.2 Optimization Control Line"for details.

   Inline expanded external procedure and internal procedures are subject to parallelization.

   The following figure shows an example of a DO loop that contains a subroutine call.

   Figure 12.11 DO loop contains a subroutine call

```
DO J=1,10              ! Not parallelized
  DO I=1,10000         ! because of the subroutine call
    A(I,J)=A(I,J)+B(I,J)
    CALL SUB(A)
  END DO
END DO
```

4. DO loop with complicated structure

   The following DO loops cannot be sliced because they are too complicated:

   - There is a branch from inside the loop to outside the DO loop.
   - The structure of the DO loop is complicated.

   The following figure shows an example of a DO loop that has a jump from inside to outside the loop.

   Figure 12.12 Jumping out from inside of the DO loop

```
      DO J=1,10                ! Not parallelized
         DO I=1,10000          ! because of jumping out of the loop
            A(I,J)=A(I,J)+B(I,J)
            IF(A(I,J).LT.0) GO TO 20
         END DO
      END DO
        .
        .
        .
  20 CONTINUE
```

   The following figure shows an example of DO loop with a complicated structure.

   Figure 12.13 Complicated DO loop structure

```
      DO J=1,10000             ! Not parallelized, because
         IF (N >0) THEN        ! of complicated DO loop structure
            ASSIGN 10 TO I
         ELSE IF (N <0) THEN
            ASSIGN 20 TO I
         ELSE
            ASSIGN 30 TO I
         END IF
         GO TO I,(10,20,30)
 10      A(J)=A(J)+0
```

```
20      A(J)=A(J)+1
30      A(J)=A(J)+2
     END DO
```

5. Input-output statements or intrinsic subroutines are included.

   Loop slicing does not support DO loops containing an input-output statement or intrinsic subroutine.

   Intrinsic functions, which are not supported by loop slicing, can be confirmed using diagnostic messages.

6. Data definition or reference order may differ from the serial process

   As shown in "Figure 12.5 A loop that is not a candidate for loop slicing", loop slicing does not support DO loops that change data definition or the order of data references when converted from serial execution.

12.2.3.1.10 Displaying the State of Automatic Parallelization

To confirm whether automatic parallelization is performed, specify compiler options -Kparallel and -Koptmsg=2 at the same time. The compilation diagnostic message will indicate the status.

When automatic parallelization is not performed, the message also outputs the reason.

The following shows an example of diagnosis message:

Example 1: Message when the program in "Figure 12.3 Loop slicing" is compiled

```
jwd5001p-i "test.f", line 2: DO loop with DO variable 'I' is parallelized.
```

Example 2: Message when the program in "Figure 12.5 A loop that is not a candidate for loop slicing" is compiled

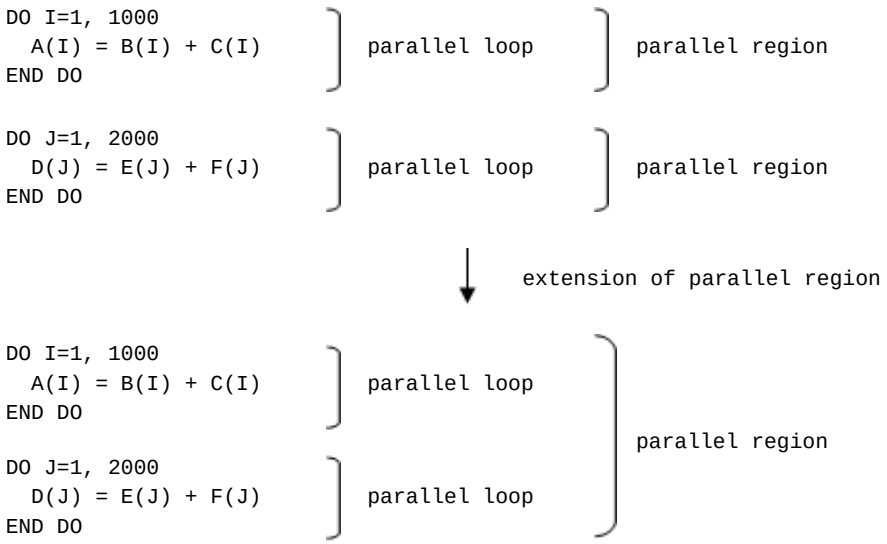
```
jwd5201p-i "test.f", line 2: DO loop is not parallelized: data dependency array  'A' may cause
different results from serial execution for loop.
```

12.2.3.1.11 Extension of Parallel Region

The Parallel region is the range between the creation of multiple threads for automatic parallelization, and the deallocation of each thread when each thread ends. In general, parallelization generates one parallelized loop for one parallel region as shown in "Figure 12.14 Example of extension of parallel region". The cost of generating a loop for parallelization during runtime is considered the parallelization overhead.

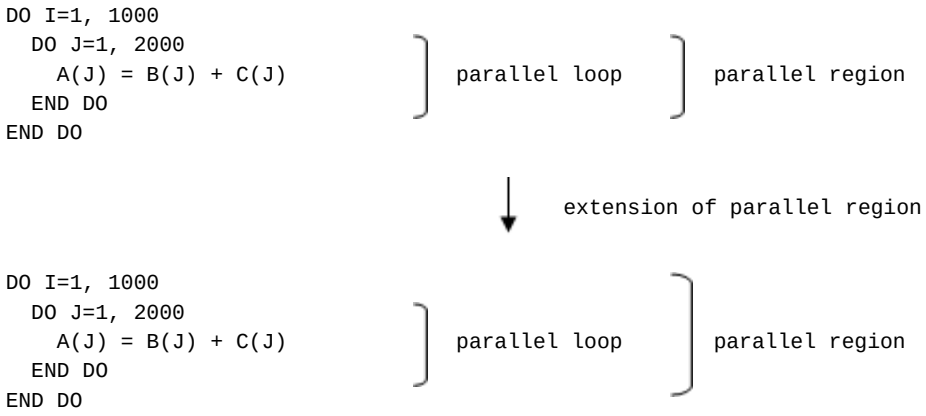
By expanding the parallel region for these two parallelization loops, two parallelized loops will be generated for one parallel region. As a result, parallelization overhead is reduced, because the parallel region is generated only once.

Figure 12.14 Example of extension of parallel region



The parallelization overhead can also be reduced for nested loops as shown in "Figure 12.15 Example of extension of parallel region (nested loop)" by expanding the parallel region. In this example, optimization reduces the number of times the parallel region is generated from 1000 down to 1.

Figure 12.15 Example of extension of parallel region (nested loop)



12.2.3.1.12 Block Distribution and Cyclic Distribution

The methods of loop slicing in the automatic parallelization are block distribution and cyclic distribution.

The block distribution is to allocate the block that the loop iteration count is distributed by the number of threads.

The cyclic distribution is to sequentially allocate the block that the loop iteration count is distributed by the arbitrary size. The default method of the automatic parallelization is block distribution. When PARALLEL_CYCLIC optimization control specifier is specified, the DO loop is cyclically distributed.

Figure 12.16 Block distribution and cyclic distribution

```
SUBROUTINE SUB(A,B)
INTEGER A(20,20),B(20,20)
DO J=1,20                    ! parallelized
  DO I=J,20
    A(I,J) = B(I,J)
  ENDDO
ENDDO
END
```

When block distribution or cyclic distribution is applied to "Figure 12.16 Block distribution and cyclic distribution", the loop iteration count is allocated to each thread as follows:

- Block distribution

  The block that the loop iteration count is divided by number of threads (2) is allocated in each thread as follows:

```
Thread 0:  {1,2,3,4,5,6,7,8,9,10}
Thread 1:  {11,12,13,14,15,16,17,18,19,20}
```

- Cyclic distribution (block size: 2)

  The block that the loop iteration count is divided by two iteration (block size: 2) is sequentially allocated in each thread as follows:

```
Thread 0:  {1,2}, {5,6}, { 9,10}, {13,14}, {17,18}
Thread 1:  {3,4}, {7,8}, {11,12}, {15,16}, {19,20}
```

12.2.3.2 Optimization Control Line

This system provides OCLs (Optimization Control Lines) to control automatic parallelization.

To activate the OCL, specify the compiler options -Kparallel and -Kocl at the same time.

12.2.3.2.1 Notation rules for optimization control lines

The column 1 to 5 of the OCL must be "!OCL ". One or more optimization specifiers for automatic parallelization follows the "!OCL ". The notation rules are detailed in the following sections.

```
!OCL i [,i] ....
```

i: Optimization control specifier for automatic parallelization

12.2.3.2.2 Description position of the optimization control line

The position where the optimization control line can be inserted depends on the type of optimization indicator.

The OCL for automatic parallelization must be specified at the position of total or loop. These are defined as follows:

- Total-position
  At the beginning of each program unit.
- Loop-position
  Immediately in front of DO statements. Other OCLs that can be specified for loop position or comment lines can be specified between the OCL and DO statement.

Example:

```
!OCL SERIAL <------------------ total-position
 SUBROUTINE SUB(B,C,N)
 INTEGER A(N),B(N),C(N)
 DO I=1,N
    A(I)=B(I)+C(I)
 END DO
 PRINT *, FUN(A)
!OCL PARALLEL <---------------- loop-position
 DO I=1,N
    A(I)=B(I)*C(I)
 END DO
 PRINT *, FUN(A)
 END SUBROUTINE
```

12.2.3.2.3 Automatic Parallelization and Optimization Control Specifiers

The Optimization Specifiers for automatic parallelization are ignored if they are specified for a DO loop for which loop slicing is not supported. See Section "12.2.3.1.9 Restrictions on Loop Slicing"for information about DO loops which are not supported by loop slicing.

12.2.3.2.4 Optimization Control Specifiers for Automatic Parallelization

The functionality of OCLs differs depending on the optimization specifier.

The optimization control specifiers for automatic parallelization specify useful information for the compiler to generate an efficient object module. They are listed in the table below.

Table 12.1 List of optimization control specifiers for automatic parallelization

Optimization control specifier	Outline of meaning
ARRAY_PRIVATE	Designates to be privatized arrays.
NOARRAY_PRIVATE	Designates canceling to be privatized arrays.

Optimization control specifier	Outline of meaning
INDEPENDENT [(*e*[,*e*]...)]	Asserts that the procedure (*e*) has no side effects, and that the DO loop that contains a reference to the procedure (*e*) can be sliced.
LOOP_PART_PARALLEL	Designates automatic parallelization that requires dividing loops.
LOOP_NOPART_PARALLEL	Suppresses automatic parallelization that requires dividing loops.
SERIAL PARALLEL	Controls whether the compiler will attempt automatic parallelization or not.
PARALLEL_CYCLIC[(*n*)]	Designates cyclic distribution in block size (*n*).
PARALLEL_STRONG	Designates to parallelize the loop that the loop has small number of loop iterations or the loop has small the number of operations.
REDUCTION	Designates to parallel reduction operations.
NOREDUCTION	Designates canceling to parallel reduction operations.
TEMP [(*var*[,*var*]...)]	Indicate variables (*var*) that are only used in the following DO loop and not outside of the loop
TEMP_PRIVATE(*var*[,*var*]...)	Designates to assign variable (*var*) to local area in thread. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.
FIRST_PRIVATE(*var*[,*var*]...)	Designates to assign variable (*var*) to local area in thread. And, designates to have a possibility of reference of the initial value of variable. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.
LAST_PRIVATE(*var*[,*var*]...)	Designates to assign variable (*var*) to local area in thread. And, the variable is directed to be referred after this DO loop. As a result, the data dependency between threads is solved, and the automatic parallelization is promoted.
var1 op var2 or *var1 op cnst*	Designates relationship between variable and variable or between variable and constant.

The details of the optimization control specifiers are described below.

For clarity, the source code used in this section has; (p) for statements that are parallelized, (m) for statements that are partially parallelized, and (s) for statements that are not parallelized.

ARRAY_PRIVATE

The ARRAY_PRIVATE specifier directs the system to identify arrays in a loop that can be privatized, and promotes parallelization by privatizing them.

The ARRAY_PRIVATE indicator has the following format:

```
!OCL ARRAY_PRIVATE
```

The ARRAY_PRIVATE indicator can be entered in the loop position and total position.

Its effect differs depending on the position.

- In the loop position

  it is effective for the corresponding DO loop.

- In the total position

  It is effective for all DO loops in the corresponding program unit.

In "Figure 12.17 Usage of the ARRAY_PRIVATE specifier", using the ARRAY_PRIVATE specifier, the compiler detects that array A can be privatized and it performs parallelization by privatizing the arrays, even if the compiler option -Karray_private is not specified.

Figure 12.17 Usage of the ARRAY_PRIVATE specifier

```
      SUBROUTINE SUB(N,M)
      INTEGER(KIND=4),DIMENSION(10000):: A
      INTEGER(KIND=4),DIMENSION(10000,100) :: B,C,D
      DATA A/10000*1/, C/1000000*2/, D/1000000*3/
     !OCL ARRAY_PRIVATE
p     DO I=1,N
p        DO J=1,M
p           A(J) = I + D(J,I)
p           B(J,I) = A(J) + C(J,I)
p        ENDDO
p     ENDDO
      PRINT*, A,B
      END
```

NOARRAY_PRIVATE

The NOARRAY_PRIVATE specifier directs the compiler not to privatize arrays even if they can be privatized.

The NOARRAY_PRIVATE indicator has the following format:

```
!OCL NOARRAY_PRIVATE
```

The NOARRAY_PRIVATE indicator can be entered in the loop position and total position. Its effect differs depending on the position.

- At the loop-position

  It is effective for the corresponding DO loop.

- At the total-position

  It is effective for all the DO loops in the corresponding program units.

In "Figure 12.18 Usage of the NOARRAY_PRIVATE specifier", array A is not privatized due to NOARRAY_PRIVATE, although it can be privatized and the compiler option -Karray_private is set.

Figure 12.18 Usage of the NOARRAY_PRIVATE specifier

```
      SUBROUTINE SUB(N,M)
      INTEGER(KIND=4),DIMENSION(10000):: A
      INTEGER(KIND=4),DIMENSION(10000,100) :: B,C,D
      DATA A/10000*1/, C/1000000*2/, D/1000000*3/
     !OCL NOARRAY_PRIVATE
p     DO I=1,N
p       DO J=1,M
p         A(J) = I + D(J,I)
p         B(J,I) = A(J) + C(J,I)
p       ENDDO
p     ENDDO
      PRINT*, A,B
      END
```

INDEPENDENT[(*e*[,*e*]...)]

The INDEPENDENT specifier directs the compiler that performing the procedure call within the DO loop in parallel execution will not change the operation result compared to serial execution. Doing this means that DO-loops with procedure references will be subject to loop slicing.

The INDEPENDENT indicator has the following format:

```
!OCL INDEPENDENT[(e[,e]...)]
```

'*e*' are the procedure names that will not be affected by loop slicing. Wildcards can be specified in '*e*'. When no procedure name is specified, this specifier is applied to all of the procedures within the target range. See Section "12.2.3.2.5 Wild Card Specification" for information on specifying wildcards.

The procedures specified in '*e*' must be compiled with the compiler options -Kthreadsafe and -Kparallel.

The INDEPENDENT indicator can be entered in the loop position and total position.

Its effect differs depending on the position.

- At the loop-position
  It is effective for the corresponding DO loop.
- At the total-position
  It is effective for all the DO loops in the corresponding program units.

Figure 12.19 Example program without INDEPENDENT specifier

```
REAL FUN,A(10000)
DO I=1,10000
   J=I
   A(I)=FUN(J)
END DO
   .
   .
END
FUNCTION FUN(J)
INTEGER J
REAL FUN
FUN=SQRT(REAL(J**2+3*J+6))
RETURN
END
```

In "Figure 12.19 Example program without INDEPENDENT specifier", the procedure FUN is called within the DO loop and the compiler cannot identify whether loop slicing can be performed for this loop.

If performing loop slicing for the DO loop that calls the procedure FUN does not affect the result, using the INDEPENDENT specifier as in "Figure 12.20 Usage of INDEPENDENT specifier" will perform loop slicing for the DO loop.

Figure 12.20 Usage of INDEPENDENT specifier

```
      REAL FUN,A(1000)
     !OCL INDEPENDENT(FUN)
p     DO I=1,1000
p        J=I
p        A(I)=FUN(J)
p     END DO
         .
         .
      END
      RECURSIVE FUNCTION FUN(J)
      INTEGER J
      REAL FUN
      FUN=SQRT(REAL(J**2+3*J+6))
      RETURN
      END
```

Note:

Ensure that INDEPENDENT is only specified for procedures for which loop slicing is supported. The following shows an example of a procedure type that is supported by loop slicing:

- Procedures that have dependency relationship with other procedures

LOOP_PART_PARALLEL

The LOOP_PART_PARALLEL specifier directs the system to perform automatic parallelization that requires dividing loops.

The LOOP_PART_PARALLEL indicator has the following format:

```
!OCL LOOP_PART_PARALLEL
```

The LOOP_PART_PARALLEL indicator can be entered in the loop position and total position.

Its effect differs depending on the position.

- In the loop position
  It is effective for the corresponding DO loop.
- In the total position
  It is effective for all DO loops in the corresponding program unit.

The LOOP_PART_PARALLEL specifier is in effect for the code shown in "Figure 12.21 Usage of the LOOP_PART_PARALLEL specifier", then the loop is divided and automatic parallelization is applied to the part of the divided loop without the -Kloop_part_parallel option.

Figure 12.21 Usage of the LOOP_PART_PARALLEL specifier

```
      REAL(KIND=8)  A(N),B(N),C(N),D(N)
 !OCL LOOP_PART_PARALLEL
m     DO I=2,N
p       A(I) = A(I)-B(I)+LOG(C(I)) ! automatic parallelization can be applied to
                                   ! this statement
s       D(I) = D(I-1)+A(I)         ! automatic parallelization cannot be applied to
                                   ! this statement
p     ENDDO
```

LOOP_NOPART_PARALLEL

The LOOP_NOPART_PARALLEL specifier directs to suppress automatic parallelization that requires dividing loops.

The LOOP_NOPART_PARALLEL indicator has the following format:

```
!OCL LOOP_NOPART_PARALLEL
```

The LOOP_NOPART_PARALLEL indicator can be entered in the loop position and total position.

Its effect differs depending on the position.

- In the loop position
  It is effective for the corresponding DO loop.
- In the total position
  It is effective for all DO loops in the corresponding program unit.

The LOOP_NOPART_PARALLEL specifier is in effect for the code shown in "Figure 12.22 Usage of the LOOP_NOPART_PARALLEL specifier", then the loop is not divided and automatic parallelization is not performed to the loop.

Figure 12.22 Usage of the LOOP_NOPART_PARALLEL specifier

```
      REAL(KIND=8)  A(N),B(N),C(N),D(N)
 !OCL LOOP_NOPART_PARALLEL
```

```
s     DO I=2,N
p       A(I) = A(I)-B(I)+LOG(C(I))
s       D(I) = D(I-1)+A(I)
p     ENDDO
```

SERIAL

The SERIAL indicator is used to prevent loop slicing of DO loops.

This specifier can be used for a DO loop that is faster when executed serially.

The SERIAL indicator has the following format:

```
!OCL SERIAL
```

The SERIAL indicator can be entered in the loop position and total position. Its effect differs depending on the position.

- At the loop-position
  It prevents loop slicing for the corresponding DO loop and its inner DO loops.
- At the total-position
  It prevents loop slicing for all the DO loops in the corresponding program units.

To suppress loop slicing for the sample program in "Figure 12.23 Example program without SERIAL specifier", position the SERIAL specifier as in "Figure 12.24 Usage of the SERIAL specifier".

Figure 12.23 Example program without SERIAL specifier

```
      REAL,DIMENSION(100,100) :: A2,B2,C2,D2,E2,F2,G2,H2
p     DO J=1,100
p        DO I=1,M
p           A2(I,J)=A2(I,J)+B2(I,J)
p           C2(I,J)=C2(I,J)+D2(I,J)
p           E2(I,J)=E2(I,J)+F2(I,J)
p           G2(I,J)=G2(I,J)+H2(I,J)
p        END DO
p     END DO
      END
```

Figure 12.24 Usage of the SERIAL specifier

```
      REAL,DIMENSION(100,100) :: A2,B2,C2,D2,E2,F2,G2,H2
!OCL SERIAL
      DO J=1,100
         DO I=1,M
            A2(I,J)=A2(I,J)+B2(I,J)
            C2(I,J)=C2(I,J)+D2(I,J)
            E2(I,J)=E2(I,J)+F2(I,J)
            G2(I,J)=G2(I,J)+H2(I,J)
         END DO
      END DO
      END
```

PARALLEL

The PARALLEL specifier is used to suppress the effect of the SERIAL specifier and perform loop slicing for specific DO loops.

PARALLEL indicator format:

```
!OCL PARALLEL
```

The PARALLEL indicator can be entered in the loop position and total position. Its effect differs depending on the position.

- At the loop-position
  It performs loop slicing for the corresponding DO loop and its inner DO loop.
- At the total-position
  It pertains to all the DO loops in the corresponding program unit.

To perform loop slicing for loop 2 in "Figure 12.25 Example program without PARALLEL specifier", specify the SERIAL and PARALLEL specifiers as in "Figure 12.26 Example program with combination of PARALLEL and SERIAL specifiers" so that loop slicing is performed only for the DO loop presented in 2.

Figure 12.25 Example program without PARALLEL specifier

```
       SUBROUTINE SUB(A1,A2,A3,B1,B2,B3,M1,M2,M3,N1,N2,N3)
       INTEGER :: M1,M2,M3,N1,N2,N3
       REAL,DIMENSION(M1,N1) :: A1,B1
       REAL,DIMENSION(M2,N2) :: A2,B2
       REAL,DIMENSION(M3,N3) :: A3,B3

p      DO J=1,N1                  ! loop 1
p        DO I=1,100
p          A1(I,J) = A1(I,J) + B1(I,J)
p        ENDDO
p      ENDDO

p      DO J=1,N2                  ! loop 2
p        DO I=1,100
p          A2(I,J) = A2(I,J) + B2(I,J)
p        ENDDO
p      ENDDO

p      DO J=1,N3                  ! loop 3
p        DO I=1,100
p          A3(I,J) = A3(I,J) + B3(I,J)
p        ENDDO
p      ENDDO

       RETURN
       END
```

Figure 12.26 Example program with combination of PARALLEL and SERIAL specifiers

```
 !OCL SERIAL
       SUBROUTINE SUB(A1,A2,A3,B1,B2,B3,M1,M2,M3,N1,N2,N3)
       INTEGER :: M1,M2,M3,N1,N2,N3
       REAL,DIMENSION(M1,N1) :: A1,B1
       REAL,DIMENSION(M2,N2) :: A2,B2
       REAL,DIMENSION(M3,N3) :: A3,B3

       DO J=1,N1                  ! loop 1
         DO I=1,100
           A1(I,J) = A1(I,J) + B1(I,J)
         ENDDO
       ENDDO
```

```
 !OCL PARALLEL
p       DO J=1,N2                  ! loop 2
p         DO I=1,100
p           A2(I,J) = A2(I,J) + B2(I,J)
p         ENDDO
p       ENDDO

        DO J=1,N3                  ! loop 3
          DO I=1,100
            A3(I,J) = A3(I,J) + B3(I,J)
          ENDDO
        ENDDO

        RETURN
        END
```

PARALLEL_CYCLIC[(*n*)]

The PARALLEL_CYCLIC specifier directs the system to perform cyclic distribution for a DO loop that automatic parallelization can be analyzed by compiler. When this specifier is applied, loop is parallelized without estimating the effects of the parallelization.

See Section "12.2.3.1.12 Block Distribution and Cyclic Distribution" for information on cyclic distribution.

The PARALLEL_CYCLIC indicator has the following format:

```
!OCL PARALLEL_CYCLIC[(n)]
```

n is the block size of cyclic distribution. *n* is number from 1 to 10000. The default is 1.

The PARALLEL_CYCLIC indicator can be entered in the loop position. It is effective for the corresponding DO loop at the loop position.

In "Figure 12.27 Usage of the PARALLE_CYCLIC specifier", DO loop is cyclically distributed with block size of 2.

Figure 12.27 Usage of the PARALLE_CYCLIC specifier

```
          SUBROUTINE SUB(A,B,N)
          INTEGER A(N,N),B(N,N)
   !OCL PARALLEL_CYCLIC(2)
  p       DO J=1,N
  p         DO I=J,N
  p           A(I,J) = B(I,J)
  p         ENDDO
  p       ENDDO
          END
```

PARALLEL_STRONG

The PARALLEL_STRONG specifier directs to perform loop slicing for loops which have not been sliced because the iteration count or instruction numbers are too small.

The PARALLEL_STRONG indicator has the following format:

```
!OCL PARALLEL_STRONG
```

The PARALLEL_STRONG indicator can be entered in the loop position and total position. Its effect differs depending on the position.

- At the loop-position
  It pertains to the corresponding DO loop and its inner DO loops.

- At the total-position
  It pertains to all the DO loops in the corresponding program unit.

The PARALLEL_STRONG specifier forces parallelization for the loop in Figure 12.28 Usage of PARALLEL_STRONG specifier, although the loop is usually excluded from parallelization because the iteration count is considered too small to gain any benefit.

Figure 12.28 Usage of PARALLEL_STRONG specifier

```
      REAL(KIND=4) :: A(10),B(10),C(10)
     !OCL PARALLEL_STRONG
p     DO J=1,10
p        A(J) = A(J) + B(J) + C(J)
p     END DO
      PRINT*, A
      END
```

REDUCTION

The REDUCTION specifier directs the compiler to perform parallelization for a DO loop that includes reduction operations.

The REDUCTION indicator has the following format:

```
!OCL REDUCTION
```

The REDUCTION indicator can be entered in the loop position and total position. Its effect differs depending on the position.

- At the loop-position
  It pertains to the corresponding DO loop and its inner DO loops.
- At the total-position
  It pertains to all the DO loops in the corresponding program unit.

As in "Figure 12.29 Usage of the REDUCTION specifier", the REDUCTION specifier forces parallelization for the DO loop that includes reduction operations, even if the compile option -Kreduction is not specified.

Figure 12.29 Usage of the REDUCTION specifier

```
      REAL S,A(5000)
     !OCL REDUCTION
p     DO I=1,5000
p        S = S + A(I)
p     END DO
      END
```

NOREDUCTION

The NOREDUCTION specifier directs the compiler not to perform parallelization for a DO loop that includes reduction operations.

The NOREDUCTION indicator has the following format:

```
!OCL NOREDUCTION
```

The NOREDUCTION specifier may be specified at the loop-position or the total-position. The effect of NOREDUCTION depend on its position.

- At the loop-position
  It pertains to the corresponding DO loop and its inner DO loops.
- At the total-position
  It pertains to all the DO loops in the corresponding program unit.

As illustrated in "Figure 12.30 Usage of the NOREDUCTION specifier", the NOREDUCTION specifier can suppress parallelization of the specified DO loop although it includes the reduction operation.

Figure 12.30 Usage of the NOREDUCTION specifier

```
      REAL S,A(5000)
     !OCL NOREDUCTION
s     DO I=1,5000
s        S = S + A(I)
s     END DO
      END
```

TEMP[(*var*[,*var*]...)]

The TEMP specifier directs the compiler that the variable used in the DO loop is a temporary variable, which is only valid within the DO loop. This can improve the performance of the parallelized DO loop.

The TEMP indicator has the following format:

```
!OCL TEMP[(var[,var]...)]
```

'*var*' is a temporary variable name used in the DO-loop. Wildcards can be specified in '*var*'. When no variable name is specified, this specifier is applied to all of the variables within the target range. See Section "12.2.3.2.5 Wild Card Specification" for information on specifying wildcards.

The TEMP indicator can be entered in the loop position and total position. Its effect differs depending on the position.

- At the loop-position
  It directs that the specified variables, which are used in the corresponding DO loop, are temporary variables.
- At the total-position
  It directs that the specified variables, which are used in all DO loops in the corresponding program unit, are temporary variables.

In "Figure 12.31 Example program without a TEMP specifier", the compiler considers that "T" may be used in the SUB because "T" is declared in the common block, and the compiler performs loop slicing while maintaining the value of "T" although it is only used within the DO loop.

Figure 12.31 Example program without a TEMP specifier

```
      COMMON T
      REAL,DIMENSION(1000,50) :: A,B,C,D
        .
        .
        .
p       DO J=1,50
p          DO I=1,1000
p             T=A(I,J)+B(I,J)
p             C(I,J)=T+D(I,J)
p          END DO
p       END DO
        .
        .
        .
      CALL SUB
      END
```

If it is known that the value of "T", at the end of the DO loop will not be referenced by the subroutine SUB, use TEMP specifier with T as shown in "Figure 12.32 Usage of TEMP specifier". This will improve the performance because the instructions that maintain the value of T at the end of the loop are no longer needed.

Figure 12.32 Usage of TEMP specifier

```
      COMMON T
      REAL,DIMENSION(1000,50) :: A,B,C,D
        .
        .
        .
     !OCL TEMP(T)
p     DO J=1,50
p        DO I=1,1000
p           T=A(I,J)+B(I,J)
p           C(I,J)=T+D(I,J)
p        END DO
p     END DO
        .
        .
        .
      CALL SUB
      END
```

Note:

If variables which are not used for temporary purposes are specified in the TEMP specifier, the compiler will perform loop slicing incorrectly.

TEMP_PRIVATE(*var*[,*var*]...)

In the target loop of automatic parallelization, the TEMP_PRIVATE specifier directs the system to treat a specified variable *var* as a local variable in each thread.

When the optimization message like the following that means automatic parallelization is prevented is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n : DO loop is not parallelized: reference to 'var' may be different from
serial execution.
```

The TEMP_PRIVATE indicator has the following format:

```
!OCL TEMP_PRIVATE(var[,var]...)
```

Wildcards can be specified in *var*. See Section "12.2.3.2.5 Wild Card Specification" for information on specifying wildcards.

The TEMP_PRIVATE indicator can be entered in the loop position. It is effective for the corresponding DO loop at the loop position.

The variable that can be specified for this specifier is as follows.

- The scalar variable of integer type, real type or logical type.
- The array variable that meets the following conditions:
  - Integer type, real type or logical type. And,
  - The size of array variable has been fixed when compilation.

When this specifier is applied, the following message is output.

```
jwd5013p-i "file", line n : !OCL TEMP_PRIVATE is applied to variable 'var'.
```

See Section "12.2.3.3.4 Notes on Using Optimization Control Line" for the notes when specifying this specifier.

In "Figure 12.33 Example program without TEMP_PRIVATE specifier", this loop is not parallelized, because the order of array A definition and the reference is uncertain in an outside loop.

Figure 12.33 Example program without TEMP_PRIVATE specifier

```
      INTEGER A(100), B(100), C(100,100)
      INTEGER M,N
      DO J=1,100            ! Target loop for automatic parallelization.
        DO I=1,M            ! Because the value of variable M is uncertain,
          A(I) = B(I)       ! the range where array A is defined is uncertain.
        ENDDO
        DO I=1,N            ! Because the value of variable N is uncertain,
          C(I,J) = A(I)     ! the range where array A is referred is uncertain.
        ENDDO
      ENDDO
      PRINT*, C
      END
```

In "Figure 12.34 Usage of TEMP_PRIVATE specifier", data dependency is solved by the TEMP_PRIVATE specifier. As a result, loop slicing is performed to the outside loop.

Figure 12.34 Usage of TEMP_PRIVATE specifier

```
 !OCL TEMP_PRIVATE(A)
p      DO J=1,100
p        DO I=1,M
p          A(I) = B(I)
p        ENDDO
p        DO I=1,N
p          C(I,J) = A(I)
p        ENDDO
p      ENDDO
```

FIRST_PRIVATE(*var*[,*var*]...)

In the target loop of automatic parallelization, the FIRST_PRIVATE specifier directs the system to treat a specified variable *var* as a local variable in each thread. And, the initial value of the variable is directed to be referred at the loop entry of each thread.

When the optimization message like the following that means automatic parallelization is prevented is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n : DO loop is not parallelized: reference to 'var' may be different from
serial execution.
```

The FIRST_PRIVATE indicator has the following format:

```
!OCL FIRST_PRIVATE(var[,var]...)
```

Wildcards can be specified in *var*. See Section "12.2.3.2.5 Wild Card Specification" for information on specifying wildcards.

The FIRST_PRIVATE indicator can be entered in the loop position. It is effective for the corresponding DO loop at the loop position.

The variable that can be specified for this specifier is as follows.

- The scalar variable of integer type, real type or logical type.

- The array variable that meets the following conditions:
  - Integer type, real type or logical type. And,
  - The size of array variable has been fixed when compilation.

When this specifier is applied, the following message is output.

```
jwd5014p-i "file", line n : !OCL FIRST_PRIVATE is applied to variable 'var'.
```

See Section "12.2.3.3.4 Notes on Using Optimization Control Line" for the notes when specifying this specifier.

In "Figure 12.35 Example program without FIRST_PRIVATE specifier", this loop is not parallelized, because the order of array A definition and the reference is uncertain in an outside loop.

Figure 12.35 Example program without FIRST_PRIVATE specifier

```
      INTEGER A(100), B(100,100), C(100,100), D(100)
      INTEGER M,N
      DO J=1,100               ! Target loop for automatic parallelization.
        DO I=1,M               ! Because the value of variable M is uncertain,
          B(I,J) = A(I)        ! the range where array A is referred is uncertain.
        ENDDO
        DO I=N,100             ! Because the value of variable N is uncertain,
          A(I) = B(I,J) + D(I) ! the range where array A is defined is uncertain.
          C(I,J) = A(I)
        ENDDO
      ENDDO
      PRINT*, C
      END
```

In "Figure 12.36 Usage of FIRST_PRIVATE specifier", data dependency is solved by the FIRST_PRIVATE specifier. As a result, loop slicing is performed to the outside loop.

Figure 12.36 Usage of FIRST_PRIVATE specifier

```
 !OCL FIRST_PRIVATE(A)
p      DO J=1,100
p        DO I=1,M
p          B(I,J) = A(I)
p        ENDDO
p        DO I=N,100
p          A(I) = B(I,J) + D(I)
p          C(I,J) = A(I)
p        ENDDO
p      ENDDO
```

LAST_PRIVATE(*var*[,*var*]...)

In the target loop of automatic parallelization, the LAST_PRIVATE specifier directs the system to treat a specified variable *var* as a local variable in each thread. And, the variable is directed to be referred after this DO loop.

When the optimization message like the following that means automatic parallelization is prevented is output, specify this specifier to assign the target data to local area of each thread. As a result, the data dependency is solved, and the automatic parallelization is promoted.

```
jwd5208p-i "file", line n : DO loop is not parallelized: reference to 'var' may be different from
serial execution.
```

The LAST_PRIVATE indicator has the following format:

```
!OCL LAST_PRIVATE(var[,var]...)
```

Wildcards can be specified in *var*. See Section "12.2.3.2.5 Wild Card Specification" for information on specifying wildcards.

The LAST_PRIVATE indicator can be entered in the loop position. It is effective for the corresponding DO loop at the loop position.

The variable that can be specified for this specifier is as follows.

- The scalar variable of integer type, real type or logical type.
- The array variable that meets the following conditions:
  - Integer type, real type or logical type. And,
  - The size of array variable has been fixed when compilation.

When this specifier is applied, the following message is output.

```
jwd5015p-i "file", line n : !OCL LAST_PRIVATE is applied to variable 'var'.
```

See Section "12.2.3.3.4 Notes on Using Optimization Control Line" for the notes when specifying this specifier.

In "Figure 12.37 Example program without LAST_PRIVATE specifier", this loop is not parallelized, because the order of array A definition and the reference is uncertain in an outside loop.

Figure 12.37 Example program without LAST_PRIVATE specifier

```
      INTEGER A(100), B(100,100), C(100,100), D(100)
      INTEGER M,N
      DO J=1,100               ! Target loop for automatic parallelization.
        DO I=1,M
          B(I,J) = I
        ENDDO
        DO I=N,100             ! Because the value of variable N is uncertain,
          A(I) = B(I,J) + D(I) ! the range where array A is defined is uncertain.
          C(I,J) = A(I)
        ENDDO
      ENDDO
      PRINT *, A, C
      END
```

In "Figure 12.38 Usage of LAST_PRIVATE specifier", data dependency is solved by the LAST_PRIVATE specifier. As a result, loop slicing is performed to the outside loop.

Figure 12.38 Usage of LAST_PRIVATE specifier

```
 !OCL LAST_PRIVATE(A)
p      DO J=1,100
p        DO I=1,M
p          B(I,J) = I
p        ENDDO
p        DO I=N,100
p          A(I) = B(I,J) + D(I)
p          C(I,J) = A(I)
p        ENDDO
p      ENDDO
       PRINT*, A, C
```

var1 op var2

var1 op cnst

These specifiers are used to indicate to the compiler the relationship between two variables or between a variable and a constant.

'*var1*' and '*var2*' are variable names. '*cnst*' is an integer constant or a named constant. Specify as follows:

```
!OCL var1 op var2
!OCL var1 op cnst
```

The relational operators that can be specified for op are listed below.

Operator	Explanation
.lt. and <	Less than
.le. and <=	Less than or equal to
.eq. and ==	Equal to
.ne. and !=	Not equal to
.gt. and >	Greater than
.ge. and >=	Greater than or equal to

The '*var1 op var2*' or '*var1 op cnst*' indicator can be entered in the loop position and total position.

Its effect differs depending on the position.

- At the loop-position
  It pertains to the corresponding DO loop and its inner DO loops.
- At the total-position
  It pertains to all the DO loops in the corresponding program unit.

In "Figure 12.39 Usage of the var1 op var2 or var1 op cnst specifier", parallelization is not performed when the OCL is not specified because it is not certain that the definition or order of data references of array A remains unchanged by parallelization, unless the magnitude relation between the array indexes is identified. *op* specifier enables parallelization by directing that the definition or order of data references of array A remain unchanged before and after the parallelization.

Figure 12.39 Usage of the var1 op var2 or var1 op cnst specifier

```
      REAL,DIMENSION(2000) :: A,B
     !OCL M.gt.2000
p     DO I = 1,2000
p        A(I) = A(I+M) + B(I)
p     END DO
      END
```

12.2.3.2.5 Wild Card Specification

In the operand of the following optimization control specifiers, a wild card may be specified for a variable name or a procedure name:

- INDEPENDENT
- TEMP

Wildcard is a combination of alphanumeric characters and a special character called a wildcard. Specifying a wildcard is equivalent to specifying multiple variable names or procedure names that correspond to the criteria specified by the string specified as the wildcard.

The two wildcard characters "*" and "?" have the following meanings:

- "*" matches a string of one or more characters consisting of any alphanumeric characters.

- "?" matches any single alphanumeric character.

Note that multiple wildcards cannot be specified when specifying one wildcard.

```
!OCL TEMP(W*)
```

In this example, "W*" specifies a variable names where the first character is "W" and the length is two or more.

For example, the variable names WORK1, W2, and WORK3 match the wildcard.

```
!OCL INDEPENDENT(SUB?)
```

In this example, "SUB?" specifies a procedure name whose first three characters are "SUB" and the length is four.

For example, the array name SUB1, SUB2, and SUB9 match the wildcard.

12.2.3.3 Notes on Automatic Parallelization

This section provides notes about using Automatic Parallelization.

12.2.3.3.1 Caution when specifying compile-time option -Kparallel,instance=N

When specifying the number of threads for parallel processing with the compiler option -Kparallel, instance=*N*, the *N* must be the same as the number of threads used on runtime. See Section "12.2.1.2.2 Number of Threads" for information about specifying the number of threads.

When an incorrect value is specified on -Kinstance, the program will abort with the following runtime message.

```
jwe1040i-s This program cannot be executed, because the number of
            threads specified by -Kinstance=N option and the actual
            number of threads are not equal.
```

12.2.3.3.2 Nested Parallel Processing

Nested parallel processing is when a procedure is called from a sliced DO loop, and the procedure includes another sliced DO loop. When this occurs, the inner DO loop will be executed serially. -Kparallel, instance=*N* must not be specified on compiling a source program that may cause nested parallel processing.

"Figure 12.40 Multiprocessing of nested loops" illustrates a sliced DO loop executed as a serial process. If a program with such a DO loop is compiled with the -Kparallel, instance=*N* option, the program may not work correctly.

Figure 12.40 Multiprocessing of nested loops

File: a.f

```
    !OCL INDEPENDENT(SUB)
     DO I=1,100               ! Executed in parallel
        J=I
        CALL SUB(J)
     END DO
       :
     END
     RECURSIVE SUBROUTINE SUB(N)
       :
     DO I=1,10000               ! should be executed in serial
        A(I)=1/B(I)**N
     END DO
       :
     END
```

The program may not work correctly, if the source code 'a.f' is compiled with the following sample.

```
frtpx -Kparallel,instance=4 a.f (invalid use)
```

To prevent such errors, specify the optimization control line "!OCL SERIAL" for the procedure call defined in the DO loop for which loop slicing is to be performed.

```
!OCL SERIAL
 RECURSIVE SUBROUTINE SUB(N)
   :
 DO I=1,10000
    A(I)=1/B(I)**N
 END DO
   :
 END
```

12.2.3.3.3 Caution when specifying compile-time option -Kparallel,reduction

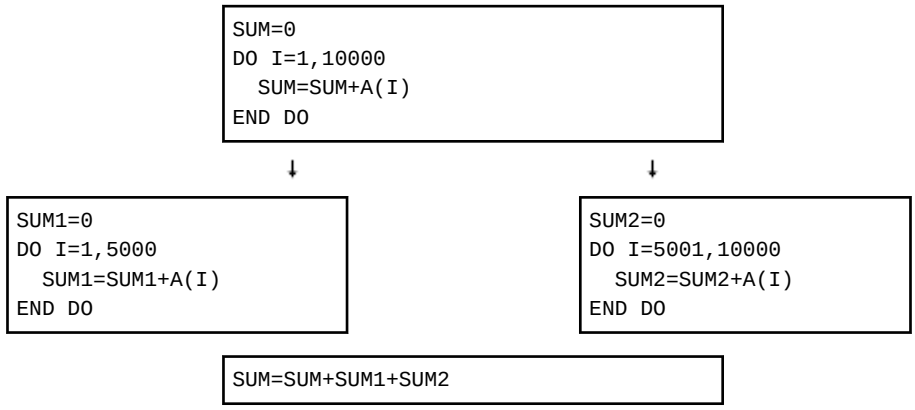
The execution result of a program compiled with -Kparallel,reduction may differ from one executed serially. This is because the sequence of operation in parallel processing may be different from a serial one.

Example:

"Figure 12.41 Side effect of reduction optimization" is a sample used in Section "12.2.3.1.8 Loop Reduction". In serial processing, the array element from A(1) to A(10000) will be sequentially appended to the variable SUM. In parallel processing, the sum of the array element A(1) to A(5000) are stored as SUM1, A(5001) to A(1000) are stored as SUM2. SUM is then derived by adding SUM1 and SUM2.

In this way, the calculation sequence of the array A element in serial processing is different from parallel processing, which will introduce a difference in precision.

Figure 12.41 Side effect of reduction optimization



12.2.3.3.4 Notes on Using Optimization Control Line

Invalid usage of the optimization control lines is shown blow.

The system will perform incorrect loop slicing because of the incorrect optimization control lines.

- TEMP specifier

  The following program specifies TEMP incorrectly for variable T. There will not be a correct value assigned to variable LAST, because the system cannot guarantee a correct value of variable T at the end of the DO loop.

```
!OCL TEMP(T)
 DO I=1, 1000
    T=A(I)+B(I)
    C(I)=T+D(I)
```

```
 END DO
 LAST = T
```

- INDEPENDENT specifier

  The following program specifies INDEPENDENT incorrectly for procedure SUB. The execution result is unpredictable when array A is sliced, because the order of the data references for array A in parallel execution is different from the order of data references in serial execution.

```
!OCL INDEPENDENT(SUB)
 DO I=2, 1000
    A(I)=B(I)+1.0
    CALL SUB(I-1)
 END DO
 ...
 END
 RECURSIVE SUBROUTINE SUB(J)
 COMMON A(1000)
 A(J)=A(J)+1.0
 END
```

- TEMP_PRIVATE specifier

  The following program specifies TEMP_PRIVATE incorrectly for variable A. Because an uncertain value is referred in each thread, the execution result is incorrect.

```
!OCL TEMP_PRIVATE(A)
 DO I=1,1000
    B(I) = A
 ENDDO
```

- LAST_PRIVATE specifier

  The following program specifies LAST_PRIVATE incorrectly for array A. Because the array A may not be defined by the last thread, the execution result is incorrect.

```
!OCL LAST_PRIVATE(A)
 DO J=1,1000
    DO I=J,500
       A(I) = B(J)
    ENDDO
 ENDDO
 PRINT *, A
```

12.2.3.3.5 I/O Statement and Intrinsic Procedure Call on Parallel Processing

When a sliced DO loop calls a procedure that includes an input-output statement, an intrinsic subroutine, or an intrinsic function that is not supported by loop slicing, the program may not work correctly. The performance of the program may deteriorate because of the overhead of parallel processing, or the result of the input/output statement may not be the same as with serial processing.

"Figure 12.42 Multiprocessing I/O statement" shows an example of input-output statements in a procedure called from a sliced DO loop.

Figure 12.42 Multiprocessing I/O statement

File: a.f

```
!OCL INDEPENDENT(SUB)
 DO I=1,100
    J=I
    CALL SUB(J)
 END DO
   :
 END
 RECURSIVE SUBROUTINE SUB(N)
```

```
  :
PRINT *,N
  :
END
```

12.2.4 Linkage with Other Multi-Thread Programs

This section describes the restrictions of linking with object program that perform in multiple threads rather than automatic parallelization.

OpenMP program by this Compiler

OpenMP object modules, which are compiled with the -Kopenmp option, can be linked with object programs that are compiled with the -Kparallel option.

Other Multi-Thread Programs

Linking with object programs that use multiple threads, but which are not compiled with the system, is not supported.

However, the program may be linked with a pthread program when using the environment variable FLIB_PTHREAD.

Environment Variable FLIB_PTHREAD

You can control the linking with a pthread program, using the environment variable FLIB_PTHREAD.

Valid values are as follows. Default value is 0.

Value	Explanation
0 (Default)	The program can be linked with a pthread program which uses the pthread thread as a control thread. However, there are the following restrictions. - The pthread thread must use suspending wait. - The pthread program must not be compiled with -Kopenmp option or -Kparallel option. (*) - The routine which is compiled with -Kopenmp option or -Kparallel option must not be used in the pthread program. (*) - The Fortran routine must not be used in the pthread program. (*)
1	The program can be linked with a pthread program which executes parallel processing. And, the program can be linked with a pthread program which uses the pthread thread as a control thread. However, there are the following restrictions. - A pthread parallel processing, and an OpenMP or automatic parallel processing must be executed in order. (*) An OpenMP or automatic parallel processing must not be executed in a pthread parallel processing. And, a pthread parallel processing must not be executed in OpenMP or automatic parallel processing. - The pthread thread must use suspending wait. - The pthread program must not be compiled with -Kopenmp option or -Kparallel option. (*) - The routine which is compiled with -Kopenmp option or -Kparallel option must not be used in the pthread program. (*) - The Fortran routine must not be used in the pthread program. (*)

Value	Explanation
	- Fujitsu's math libraries must not be used in the pthread program. (The pthread program must not be compiled with -SSL2BLAMP option.) (*) - You must execute OpenMP parallel processing with two or more threads, before creating pthread threads. And, this number of OpenMP threads cannot be changed later. And, when this function is used, the following functions are effective. - FLIB_SPINWAIT=0 - FLIB_CPUBIND=off Operation is not guaranteed when a value which is different in the above-mentioned environment variable is specified.

*) Operation is not guaranteed when the restriction function is used.

12.3 Parallelization by OpenMP Specifications

This section explains the parallelization based on the OpenMP specifications provided by this compiler.

This compiler is designed to compile programs that conform to the OpenMP Application Program Interface Version 3.1 July 2011.

See the "OpenMP Architecture Review Board" website for details about OpenMP Application Program Interface Version 3.1 July 2011.

12.3.1 Compilation and Execution

This section describes how to compile and execute the source program created on OpenMP specifications.

12.3.1.1 Compilation

Specify the compiler option -Kopenmp to compile and link a source program created to OpenMP specifications.

12.3.1.1.1 Compiler Options to Compile OpenMP Programs

This section explains the compiler options for OpenMP Programs.

-Kauto

Local variables, other than those with the SAVE attribute or initial values, are treated as automatic variables and allocated to the stack area. However, the local variables of the main program are not allocated to the stack. See Section "9.11 Effects of Allocating on Stack" for points to note about the size of the stack area.

This option is automatically set when the -Kopenmp option is specified.

-Knoauto

Local variables, other than those with the SAVE attribute or initial values, are not treated as automatic variables.

If this option is to be specified at the same time as the -Kopenmp option, it must be specified after the -Kopenmp option. Specifying this option with -Knothreadsafe option reduces the stack area size of the procedure being compiled.

This option can be specified for the main program or a subprogram called outside of the dynamic parallel region, and must be with OpenMP directives other than the following:

- PARALLEL constructs that do not include private variable declarations
- THREADPRIVATE directing statement

If this is specified for a procedure for which it cannot be specified, the program may not work correctly.

This option is effective only when it is specified with the -Knothreadsafe option. This is because, as with -Kauto, the -Kthreadsafe option set by -Kopenmp allocates local variables to the stack.

-Kopenmp

-Kopenmp will activate the directives that conform to the OpenMP specifications. The -Kthreadsafe, -Kauto, and -D_OPENMP=201107 options are also activated when -Kopenmp is in effect. However, the local variable of the main program will not be allocated to the stack even if the -Kauto option is set. The -Knoauto and -Knothreadsafe options can be valid only when they are specified after the -Kopenmp option. Caution should be taken when the -Knoauto and -Knothreadsafe options are specified at the same time as the -Kopenmp option (Refer to -Knoauto and -Knothreadsafe options). Even when only object programs are specified as file names (that is, when only linking is to be performed), this option must be specified if an object program that was compiled using the -Kopenmp option is included.

```
$ frtpx a.f90 -Kopenmp -c
$ frtpx a.o -Kopenmp
```

-Knoopenmp

When the -Knoopenmp option is specified, the source program is compiled with the directives of OpenMP specifications treated as comment lines.

When both -Kopenmp and -Knoopenmp are not specified, the compiler assumes that -Knoopenmp is specified.

-Kopenmp_ordered_reduction

The -Kopenmp_ordered_reduction option specifies to fix the operation order of the reduction operations same as in the numerical order of the threads at the end of the region for which the reduction clause was specified. Note that execution performance may decrease compared to when the -Kopenmp_ordered_reduction is invalidated. This option is valid only when the -Kopenmp option is in effect.

-Kopenmp_noordered_reduction

The -Kopenmp_noordered_reduction option specifies not to fix the operation order of the reduction operation at the end of the region for which the reduction clause was specified.

-Kopenmp_tls

The -Kopenmp_tls option directs that threadprivate variables are allocated to the Thread-Local Storage. This option is effective only when the -Kopenmp option is set.

If the threadprivate variable is passed between Fortran and C/C++ programs, it is necessary to specify the -Kopenmp_tls option.

If the -Kopenmp_tls option is used, all the source programs comprising the executable program must be compiled with the -Kopenmp_tls option.

-Kopenmp_notls

The -Kopenmp_notls option directs that threadprivate variables are allocated to the heap.

-Kthreadsafe

A thread-safe object program is generated when the -Kthreadsafe option is specified. See Section "9.11 Effects of Allocating on Stack" for points to note about the size of the stack area.

This option is automatically set when the -Kopenmp option is specified.

-Knothreadsafe

Object programs that are not thread-safe are created when the -Knothreadsafe option is specified.

If this option is to be specified at the same time as the -Kopenmp option, is must be specified after the -Kopenmp option. When this option is specified the instructions generated are not those that guarantee the process to be thread-safe.

This option can be specified for the main program or a subprogram called outside of the dynamic parallel region, and must be with OpenMP directives other than the following:

- PARALLEL constructs that do not include private variable declarations
- THREADPRIVATE directives

If this is specified for a procedure for which it cannot be specified, the program may not work correctly.

-Nprivatealloc

For a list item with the ALLOCATABLE attribute in a private clause of OpenMP: if the list item is "currently allocated", the new list item will have an initial state of "not currently allocated" the program does not conform to the OpenMP 3.1 standard. This option is valid only when the -Kopenmp option is in effect.

-Nnoprivatealloc

When an allocatable array is described in the OpenMP private section and the status is "currently allocated", a new list item will have an initial state of "currently allocated".

12.3.1.1.2 Displaying Optimization Information of OpenMP Program

This section describes the display of optimization information of an OpenMP program, when the compiler option -Qp or -Nlst=p is specified. See Section "4.1.2 Compilation Information" for information about the compiler option -Qp and -Nlst=p.

When the compiler option -Qp or -Nlst=p is specified, the optimization information for the instructions generated by the OpenMP directives are marked as follows:

- "p" is marked for statements that can be executed in parallel processing. It is not marked for the statements executed redundantly. Statements directly enclosed in a DO or SECTIONS statement will meet this condition. Within the WORKSHARE directive, only the parallelized statements are marked as "p".
- "s" is marked for statements that will be executed in only one thread. Statements directly enclosed in MASTER, SINGLE, CRITICAL, or ORDERED directive will meet this condition. For the instructions just after the ATOMIC directive, "s" will be marked only when whole instructions are executed within a single thread.
- "m" is marked for statements which include both instructions that will be executed in parallel and in a single thread. A statement immediately after an ATOMIC directive within a WORKSHARE directive may meet this condition, for example.

For nested parallelization, only the parallelization information of the innermost parallel region, which includes the target instructions, is displayed. Whether the parallel region is executed in parallel or serially does not affect the display.

12.3.1.2 Execution Process

The procedure to run the OpenMP program is the same as executing it as a serial process.

12.3.1.2.1 Environment Variable at Execution

THREAD_STACK_SIZE

A user can specify the stack area size for each thread using the environment variable THREAD_STACK_SIZE in Kbytes. When the environment variable OMP_STACKSIZE is specified, the greater value is used as the stack area size for each thread. See Section "12.3.1.2.3 Notes during Execution" for details.

OMP_STACKSIZE

A user can specify the stack area size for each thread using the environment variable OMP_STACKSIZE in Kbytes, Mbytes, or Gbytes. When the environment variable THREAD_STACK_SIZE is specified, the greater value is used as the stack area size. See Section "12.3.1.2.3 Notes during Execution" for details.

OMP_PROC_BIND

A user can control the CPU Binding for Thread using the environment variable OMP_PROC_BIND.

Valid values and their meanings are as follows:

TRUE	:	The thread is bound to a CPU.
FALSE	:	The thread is not bound to a CPU.

The environment variable FLIB_CPUBIND or FLIB_CPU_AFFINITY is given to priority more than OMP_PROC_BIND.

However, OMP_PROC_BIND is effective in the following case.

- FLIB_CPUBIND=off is specified, and FLIB_CPU_AFFINITY is not specified.

When TRUE is specified, threads are bound to CPUs in order of cpuid list. However, cpuid which can actually be used becomes only within the limits of CPU affinity of the process at the time of an execution start.

See Section "D.2.1 FLIB_CPUBIND" for details.

See Section "12.3.1.2.3 Notes during Execution" for details.

FLIB_SPINWAIT and OMP_WAIT_POLICY

The standby threads during the serial execution can be controlled with the environment variable FLIB_SPINWAIT or OMP_WAIT_POLICY.

FLIB_SPINWAIT value:

unlimited	:	Uses spin wait.
0	:	Uses suspending wait.
<n>s	:	Uses spin wait until after <n> seconds, and uses suspending wait.
<n>[ms]		Uses spin wait until after <n> milliseconds, and uses suspending wait.

<n> is integer value that must be positive.

The default is unlimited.

Spin waiting is a technique to wait while consuming CPU. Suspend waiting is a technique to wait without consuming CPU. To give priority to elapsed time, specify unlimited, as spin waiting requires less overhead for parallel process. Select 0 to give priority to the CPU time.

OMP_WAIT_POLICY value:

ACTIVE	:	Uses spin wait.
PASSIVE	:	Uses suspending wait.

The default is "ACTIVE".

Select ACTIVE to give priority to the elapsed time. Select PASSIVE to give priority to the CPU time.

The settings of environment variable FLIB_SPINWAIT will be valid when environment variable FLIB_SPINWAIT is specified.

FLIB_FASTOMP

A user can control the use of the high speed runtime library using the environment variable FLIB_FASTOMP. Valid values and their meanings are as follows: The default is "TRUE".

TRUE	:	High speed runtime library is used.
FALSE	:	High speed runtime library is not used.

The high-speed runtime library achieves its speed by restricting part of the OpenMP specification. The features of the high-speed runtime library are as follows.

- This assumes that the program does not have nested parallelization, and one CPU is dedicated to one thread.
- The hardware barrier can be used in Job Operation Software environment.

The following restrictions are applied to use the high-speed runtime library:

- The number of threads of the OMP_SET_NUM_THREADS subroutine, or the number of threads specified in the NUM_THREADS clause in the PARALLEL directive, must be the same as the number of threads determined in the runtime environment.

See Section "12.3.1.2.3 Notes during Execution" for the number of threads determined in the runtime environment.

The following restrictions are applied for controlling the execution of the OpenMP environment variable and the service procedure when using the high-speed runtime library.

- Regardless of the configuration defined in the environment variable OMP_NESTED and the configuration of OMP_SET_NESTED subroutine, nested parallel regions are always serialized.

- The number of threads cannot exceed the number of available CPUs, regardless of the configuration defined in the environment variable OMP_DYNAMIC and the configuration of OMP_SET_DYNAMIC subroutine.

See Section "12.3.1.2.3 Notes during Execution" for information on the number of available CPUs.

12.3.1.2.2 Environment Variable for OpenMP Specifications

The following environment variables are supported by the OpenMP specifications. See the "OpenMP specifications" for details.

OMP_SCHEDULE

This specifies the schedule type and the chunk size that should be performed for the DO and PARALLEL DO directives with schedule type RUNTIME.

OMP_NUM_THREADS

This specifies the number of threads used during execution.

OMP_DYNAMIC

This specifies to enable and disable dynamic thread adjustment.

OMP_PROC_BIND

This specifies whether threads are bound to CPUs.

OMP_NESTED

This enables or disables nested parallelism.

OMP_STACKSIZE

This specifies the stack size for each thread.

OMP_WAIT_POLICY

This specifies the behavior of the waiting thread.

OMP_MAX_ACTIVE_LEVELS

This specifies the maximum number of nested active parallel regions.

OMP_THREAD_LIMIT

This specifies the maximum number of threads executed on an Open MP program.

12.3.1.2.3 Notes during Execution

The following points should be noted when using parallelization features based on the OpenMP specifications on this compiler.

Allocating Variables at Runtime

The programs on this system that use parallelization features based on OpenMP specifications allocate local variables (other than the automatic data object while compiler option -Knoautoobjstack is in effect) and private variables on the stack, except for the main program. When a procedure requires a large area for local and private variables, a sufficiently large stack must be allocated.

The maximum process stack area can be specified using a command such as "ulimit" (bash built-in command).

The stack area size for each thread is the same as that for the process. When the maximum stack size for the process is larger than min(the size of available memory in this system, the size of a virtual memory) / (the number of threads)

the system will determine the stack area size as follows and allocate it. However, the assumed value does not guarantee that the system works correctly. It is recommended that an appropriate value for maximum process stack size be specified manually. Note that the maximum virtual memory size can also be confirmed using ulimit (bash built-in command).

```
S = (M/T)/5
```

S: the stack size for each thread (byte)
M: a size with a smaller mounting memory and a smaller virtual memory (byte)
T: the number of threads created in the first parallel region

The compiler calculates the stack area size without considering the feature that can dynamically change the thread numbers to align the stack area size of each thread. For this reason, the formula presented above uses the thread number assuming "FLIB_FASTOMP=TRUE". When the thread number is to be changed dynamically, specify the maximum thread numbers using OMP_NUM_THREADS in advance.

When the stack area size for each thread needs to be specified, use environment variable THREAD_STACK_SIZE or OMP_STACKSIZE.

See Section "12.3.1.2.1 Environment Variable at Execution" for information about environment variables THREAD_STACK_SIZE and OMP_STACKSIZE.

Maximum number of CPUs

The maximum number of CPUs is as follows:

- For jobs:
  Number of CPUs that can be used with jobs
  See Section "D.1 Management of the CPUs resource" for details.
- Not for job:
  Number of CPUs in the system

Number of Threads

The number of threads is common for both OpenMP and automatic parallelization when using the high-speed runtime library, otherwise the number will be determined independently. Use of the high-speed runtime library is determined by the environment variable FLIB_FASTOMP (See Section "12.3.1.2.1 Environment Variable at Execution").

- When using the high-speed runtime library (i.e., when FLIB_FASTOMP is TRUE),
  the thread number will be determined with the following priority:

  1. The value specified in the environment variable OMP_NUM_THREADS (Note)
  2. The value specified in the environment variable PARALLEL
  3. Number of CPUs that can be used with jobs
  4. 1 thread

  (Note)

  When -Kparallel is specified during the linkage phase of compilation, the value must be the same as the environment variable PARALLEL if specified. When the value does not match, the following message will be output, and the smaller value will be used.

  ```
  jwe1042i-w The value specified in the environment variable PARALLEL and
             OMP_NUM_THREADS are different when the environment variable
             FLIB_FASTOMP is TRUE. The value num is used as the value
             specified in the environment variable.
  ```

  If the number of threads determined in this priority exceeds the maximum number of CPUs, the number of threads will be aligned to the maximum number of CPUs.

  The value specified in the NUM_THREADS clause section of the PARALLEL directive or defined in the OMP_SET_NUM_THREADS service routine must be the same as the number of threads determined here. If a different value is specified, the program will terminate with the following message.

  ```
  jwe1041i-s The number of thread cannot be changed when the environment
             variable FLIB_FASTOMP is TRUE.
  ```

- When the high-speed runtime library is not used (i.e. when FLIB_FASTOMP is FALSE),
  the number of threads for OpenMP will be determined using the following priority. The number of threads for automatic parallelization follows the description in Section "12.2 Automatic Parallelization".

  1. The value specified in the NUM_THREADS clause in the PARALLEL statement
  2. The value specified in the OMP_SET_NUM_THREADS service routine.
  3. The value specified in the environment variable OMP_NUM_THREADS

4. The value specified in the environment variable PARALLEL
5. Number of CPUs that can be used with jobs
6. 1 thread

When the dynamic thread adjustment feature is enable, the smaller value of the number of threads determined by the above priority, and the CPU number will be used as the number of threads.

When the dynamic thread adjustment feature is disable, the thread number determined by the above priority will be used, even if it exceeds the maximum number of CPUs. When more than one thread is assigned for a CPU, the thread that is to be executed in parallel will be executed in a time sharing manner. In this case, the overhead caused by synchronizing the threads may sacrifice the performance of the program. It is recommended to conserve resources by setting one thread per CPU, unless there is a specific reason to do otherwise. See Section "D.1 Management of the CPUs resource" for information on the number of CPUs that can be used with jobs.

Notes when setting time limits

The program may not work correctly when time limits are set using the command such as ulimit(1). In such a case, use the Fortran exception handling process, execution time option -Wl, -i to invalidate it.

Alternatively, use the execution time option -Wl,-t to set the time limit.

Notes when using service routines

- ALARM Service Function

  It may not work correctly. It is only for a program using serial processing.

- KILL and SIGNAL Services Functions

  It may cause a deadlock. It is only for a program using serial processing.

- FORK Service Function

  It may not work correctly. It is only for a program using serial processing.

- SYSTEM, SH and CHMOD Service Functions

  Memory usage doubles and performance may deteriorate. Use only with programs using serial processing.

CPU Binding for Thread

CPU Binding for Thread differs between job execution and non-job execution.

When executing program on non-job.

The thread is not bound to a CPU. But you can control CPU Binding for Thread using environment variable `FLIB_CPU_AFFINITY`.

The environment variable FLIB_CPU_AFFINITY

Threads are bound to CPUs in order of the specified cupid list.

When the number of specified CPUs is exceeded, it is repeatedly used from the beginning of the list.

The cpuid shall be separated by comma (',') or space(' ').

The cpuid list can have the next form that has range with increment.

```
cpuid1[-cpuid2[:inc]]
```

cpuid1: cpuid of the beginning of the range. (0<=*cpuid1*<CPU_SETSIZE)
cpuid2: cpuid of the end of the range. (0<=*cpuid2*<CPU_SETSIZE)
inc: increment (1<=*inc*<CPU_SETSIZE)

In addition, it is necessary to be the following.

```
cpuid1<=cpuid2
```

It becomes equivalent to the case where all CPUs for every increment value *inc* in the range from *cpuid1* to *cpuid2* are specified.

The cpuid can be used the above-mentioned value. However, cpuid which can actually be used becomes only within the limits of CPU affinity of the process at the time of an execution start.

See `CPU_SET(3)` about details of `CPU_SETSIZE`.

Examples of using environment variable `FLIB_CPU_AFFINITY` are shown below.

Example 1:

```
$ export FLIB_CPU_AFFINITY="0,2,1,3"
```

The thread is bound to CPU in order of 0, 2, 1, and 3.
When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 2:

```
$ export FLIB_CPU_AFFINITY="0-7"
```

The thread is bound to CPU in order of 0, 1, 2, 3, 4, 5, 6 and 7.
When the number of threads is nine or more, it is repeatedly used from the beginning of the list.

Example 3:

```
$ export FLIB_CPU_AFFINITY="0-7:2"
```

The thread is bound to CPU in order of 0, 2, 4 and 6.
When the number of threads is five or more, it is repeatedly used from the beginning of the list.

Example 4:

```
$ export FLIB_CPU_AFFINITY="0-4:2,1,7"
```

The thread is bound to CPU in order of 0, 2, 4, 1 and 7.
When the number of threads is six or more, it is repeatedly used f from the beginning of the list.

When executing program on job.

When thread is not bound to CPU, you may control CPU Binding with FLIB_CPU_AFFINITY as executing on non-job.

When thread is bound to CPU on job, the environment variable FLIB_CPU_AFFINITY becomes invalid.

For details of CPU Binding, see Section "D.2 CPU Binding".

12.3.1.2.4 Output from Multiple Threads

If errors are detected from multiple threads in a program while it is running, diagnostic message and trace back maps are output for each thread. For this reason, a piece of information per thread is output in a parallel region.

12.3.2 Implementation-Dependent Specifications

The implementation-dependent specification in the OpenMP is implemented in the following way in this compiler. Refer to the OpenMP specifications for details about these items.

Memory Model

When multiple threads attempt to update a variable that exceeds 4 bytes or is stored across a 4-byte boundary, the value of the variable may be left undefined without storing any of the values, unless lock control is explicitly performed.

If read and write to the variable occurs from multiple threads at the same time, and this exceeds 4 bytes or crosses a 4-byte boundary, the read value may be undefined rather than the written value, unless lock control is explicitly performed.

Internal Control Variables

The initial values for each of the internal control variables are as follows:

- nthreads-var: 1.

- dyn-var: TRUE.
- run-sched-var: STATIC without chunk size.
- def-sched-var: STATIC without chunk size.
- bind-var:FALSE.
- stacksize-var: the stack area size for threads in Section "12.3.1.2.3 Notes during Execution".
- wait-policy-var: ACTIVE.
- thread-limit-var: 2147483647.
- max-active-levels-var: 2147483647.

Dynamic Thread Adjustment Features

This compiler provides dynamic thread adjustment for parallelization performed based on OpenMP specifications. See Section "12.3.1.2.3 Notes during Execution" for information about the effect of this feature.

The dynamic thread adjustment feature is set by default.

Loop Statement

An eight byte integer type is used to calculate the iteration count of a collapsed loop.

When AUTO is set for the internal control variable run-sched-var, the effect of the SCHEDULE(RUNTIME) construct is set to SCHEDULE(STATIC).

SECTIONS Construct

Assignment to a thread of a structured block in a SECTIONS construct is performed in the same way as a dynamic schedule. When a thread reaches the SECTIONS region or finishes performing one of the structured blocks in the region, the thread starts performing the next structured block in this sequence.

SINGLE Construct

SINGLE region is executed by the thread that reaches the region first.

Task scheduling points

In untied TASK regions, task scheduling points are located at the same positions as in the case of tied TASK regions: only in TASK, TASKWAIT, explicit or implicit BARRIER constructs, and at the completion point of the task.

ATOMIC Construct

Two ATOMIC regions will be executed independently (not exclusively), when a variable type to be updated or a type parameter is different. When the type and type parameter match, it may be executed exclusively even if the address is different.

- When updating a logical type, complex number type, 1 byte integer type, 2 byte integer type, or quadruple precision (16 byte) real type variable
- When updating array element and index expression is not the same
- The target expression has explicit or implicit type conversion

OMP_SET_NUM_THREADS Routine

Calling the OMP_SET_NUM_THREADS routine has no effect when the argument value is equal to or less than 0. A value that exceeds the number of threads supported by the system must not be specified.

OMP_SET_SCHEDULE Routine

There are no schedule types that are implementation-dependent.

OMP_SET_MAX_ACTIVE_LEVELS Routine

A call to the OMP_SET_MAX_ACTIVE_LEVELS routine will be ignored when it is performed from an explicit PARALLEL region. It will also be ignored when the argument is an integer that less than 0.

OMP_GET_MAX_ACTIVE_LEVELS Routine

The OMP_GET_MAX_ACTIVE_LEVELS routine can be called from anywhere in the program and it returns the value of the internal control variable max-active-levels-var.

Environment Variable OMP_SCHEDULE

When the schedule type specified for OMP_SCHEDULE is invalid, the schedule type is ignored, and the default schedule (STATIC without chunk size) is used.

When the schedule type specified for OMP_SCHEDULE is STATIC, DYNAMIC, or GUIDED, and the chunk size is not a positive number, the chunk size will be as follows:

STATIC	:	no chunk size
DYNAMIC, or GUIDED	:	1

Environment Variable OMP_NUM_THREADS

When a value equal to or less than 0 is specified for the list of OMP_NUM_THREADS, it works in the same ways as when 1 is specified. A value that exceeds the number of threads supported by the system must not be specified.

Environment Variable OMP_PROC_BIND

When a value other than TRUE or FALSE is specified for OMP_PROC_BIND, the value is ignored, and the default value (FALSE) is used.

Environment Variable OMP_DYNAMIC

When a value other than TRUE or FALSE is specified for OMP_DYNAMIC, the value is ignored and the default value (TRUE) is used.

Environment Variable OMP_NESTED

When a value other than TRUE or FALSE is specified for OMP_NESTED, it is ignored and the default value (FALSE) is used.

Environment Variable OMP_STACKSIZE

When the value specified for OMP_STACKSIZE does not meet the defined format, it is ignored and the default value (Refer to "12.3.1.2.3 Notes during Execution") is used.

Environment Variable OMP_WAIT_POLICY

ACTIVE performs spin wait. PASSIVE performs suspend wait.

Environment Variable OMP_MAX_ACTIVE_LEVELS

When the value specified for OMP_MAX_ACTIVE_LEVELS is an integer that is less than 0, it is ignored and the default value (2147483647) is used.

Environment Variable OMP_THREAD_LIMIT

When the value specified for OMP_THREAD_LIMIT is not a positive integer, it is ignored and the default value (2147483647) is used.

THREADPRIVATE directing statement

The value, allocation state, and association state of the thread private variables will be maintained across the consecutive PARALLEL regions, only when the following conditions are satisfied. These conditions are not affected by the configuration of the dynamic thread adjustment feature and nested parallelization.

- The consecutive PARALLEL region is not nested within another explicit PARALLEL region.
- The same number of threads is used to execute both PARALLEL regions.

If these conditions are not satisfied, the value, allocation status, and association status of the threadprivate variable at the entry point of the subsequent PARALLEL region will have an undefined status. However, the allocation status will be not allocated at the entry point of the subsequent PARALLEL region, when the following is true:

- The allocation status is not allocated just before entering the subsequent PARALLEL region.
- The allocation status is not allocated by any thread at the exit of the all preceding PARALLEL regions that are already executed.

SHARED Clause

When passing a shared variable to a non-intrinsic procedure, the values of the shared variables are stored in a temporary storage area, and the values are stored in the actual argument area after the procedure call. This can occur when the following conditions are satisfied:

a. The actual argument is one of the following:
   - A shared variable
   - A subobject of a shared variable
   - An object associated to a shared variable
   - An object associated to a subobject of shared variable

b. In addition the actual argument is one of the following:
   - An array section
   - An array section with a vector subscript
   - An assumed-shape array
   - A pointer array

c. The associated dummy argument for this actual argument is an explicit-shape array or an assumed-size array.

Runtime Library Definitions

This compiler provides an include file omp_lib.h and a module omp_lib for the parallelization feature based on OpenMP specifications. This does not include an interface block with generic name.

The following notes must be considered when using the module omp_lib and omp_lib.h:

- When the compiler option -AU is specified, module name omp_lib and each library routine name must be specified using lower case alphabetic characters.
- When the compiler option -CcdII8, -CcI4I8, -CcdLL8, -CcL4L8, -Ccd4d8, or -Cca4a8 is specified, the corresponding type of the library routine argument and the result will be converted. On execution, the corresponding runtime option -Lb or -Li must be specified.
- Although the diagnostic message jwd2004i-i will be displayed if the named constant declared in omp_lib.h was not used, it does not affect the result.

12.3.3 Clarifying OPENMP Specifications and restrictions

This section explains the interpretation of the OpenMP Specifications and some restrictions in this system.

12.3.3.1 Specifying Label with ASSIGN Statement and Assigned GO TO Statement

An ASSIGN statement cannot reference a label outside of the structured block that the ASSIGN statement itself belongs to. An ASSIGN statement cannot reference a label in a PARALLEL region from outside of the PARALLEL region.

An assigned GO TO statement must not jump into, or jump out of the structured block range.

12.3.3.2 Additional Functions and Operators in ATOMIC Directive and REDUCTION Clause

The following intrinsic functions and operations can be specified in an ATOMIC directive and REDUCTION clause.

Intrinsic function	:	AND, OR
Operator	:	.XOR., .EOR.

12.3.3.3 Notes on Using THREADPRIVATE

When using the THREADPRIVATE directive, the same common block name must be used for all program units and procedures in the program specified on THREADPRIVATE.

The common block, specified with THREADPRIVATE, cannot have the size extended.

12.3.3.4 Inline Expansion

The following procedures are not expanded inline.

- User defined procedures which include OpenMP directives.

12.3.3.5 Internal Procedure Calling from Parallel Region

All the parent procedure variables, which are referenced in the internal procedure called from a parallel region, are regarded as SHARED, even if they are privatized in the parallel region.

Example:

```
   ...
  I=1          THIS "I" IS SHARED.
!$OMP PARALLEL PRIVATE(I)
  I=2                   *
  PRINT *,I             *"I" IS PRIVARE IN THIS RANGE.
  CALL C_PROC()         *
!$OMP END PARALLEL
  CONTAINS
  SUBROUTINE C_PROC()
   ...                 *
  PRINT *,I            *"I" is SHARED IN THIS RANGE
   ...                 *
  END SUBROUTINE
   ...
```

12.3.3.6 Polymorphic Variable

Polymorphic variable cannot be specified in THREADPRIVATE directive, and unlimited polymorphic variable cannot be specified as following clauses:

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- COPYPRIVATE
- COPYIN

12.3.3.7 BIND attribute

If the -Kopenmp_tls option is invalid, a variable or common block declared with BIND attribute cannot be specified in THREADPRIVATE directive.

12.3.3.8 OPTIONAL attribute

When a variable declared with OPTIONAL attribute is specified in PRIVATE, FIRSTPRIVATE, LASTPRIVATE, COPYPRIVATE or COPYIN clause, the variable cannot be specified as actual argument of PRESENT intrinsic function in the scope.

12.3.3.9 ASSOCIATE NAME

An associate name of SELECT TYPE or ASSOCIATE construct cannot be specified in OpenMP directive.

12.3.3.10 Selector

The variable of selector in an ASSOCIATE construct or the variable of selector with the associate name in a SELECT TYPE construct cannot be specified as following clauses:

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- COPYPRIVATE
- COPYIN
- REDUCTION

12.3.3.11 Derived type variable with length type parameter

The derived type variable with length type parameter cannot be specified as following clauses:

- PRIVATE
- FIRSTPRIVATE
- LASTPRIVATE
- COPYPRIVATE
- COPYIN

12.3.4 Notes on Creating Program

12.3.4.1 Implementing PARALLEL Region and Explicit TASK Region

The structured block within a PARALLEL construct or a TASK construct is compiled as an internal procedure.

The internal procedures generated from a PARALLEL construct will be named "_OMP_IdNumber_".

The internal procedures generated from a TASK construct will be named "_TSK_IdNumber_".

12.3.4.2 Automatic Parallelization for OpenMP Programs

This compiler allows the specification of the compiler options -Kopenmp and -Kparallel at the same time. When they are specified together, the DO loop automatic parallelization performed is restricted as follows:

- Automatic parallelization is not performed for DO loops within the OpenMP constructs.
- Automatic parallelization is not performed for DO loops that statically include OpenMP directives.
- When there is a DO loop which is parallelized by the OpenMP DO directive, automatic parallelization is not performed for the following DO loops:
  - DO loops parallelized by the OpenMP DO directive
  - DO loops within a DO loop parallelized by the OpenMP DO directive

12.3.5 Linkage with Other Multi-Thread Programs

This section describes the restrictions to link with multi-threaded programs other than OpenMP.

Programs Automatically Parallelized Using this Compiler

Object modules compiled with this compiler using the -Kparallel option can be linked with other modules compiled with this compiler using the -Kparallel option.

Other multi-thread programs

The object module created by this system cannot be linked with multi-threaded object modules created by another multithread system.

However, the program may be linked with a pthread program when using the environment variable FLIB_PTHREAD.

Environment Variable FLIB_PTHREAD

You can control the linking with a pthread program, using the environment variable FLIB_PTHREAD.

Valid values are as follows. Default value is 0.

Value	Explanation
0 (Default)	The program can be linked with a pthread program which uses the pthread thread as a control thread. However, there are the following restrictions. - The pthread thread must use suspending wait. - The pthread program must not be compiled with -Kopenmp option or -Kparallel option. (*) - The routine which is compiled with -Kopenmp option or -Kparallel option must not be used in the pthread program. (*) - The Fortran routine must not be used in the pthread program. (*)
1	The program can be linked with a pthread program which executes parallel processing. And, the program can be linked with a pthread program which uses the pthread thread as a control thread. However, there are the following restrictions. - A pthread parallel processing, and an OpenMP or automatic parallel processing must be executed in order. (*) An OpenMP or automatic parallel processing must not be executed in a pthread parallel processing. And, a pthread parallel processing must not be executed in OpenMP or automatic parallel processing. - The pthread thread must use suspending wait. - The pthread program must not be compiled with -Kopenmp option or -Kparallel option. (*) - The routine which is compiled with -Kopenmp option or -Kparallel option must not be used in the pthread program. (*) - The Fortran routine must not be used in the pthread program. (*) - Fujitsu's math libraries must not be used in the pthread program. (The pthread program must not be compiled with -SSL2BLAMP option.) (*) - You must execute OpenMP parallel processing with two or more threads, before creating pthread threads. And, this number of OpenMP threads cannot be changed later. And, when this function is used, the following functions are effective. - FLIB_SPINWAIT=0 - FLIB_CPUBIND=off Operation is not guaranteed when a value which is different in the above-mentioned environment variable is specified.

*) Operation is not guaranteed when the restriction function is used.

12.3.6 Debug OpenMP Programs

Where an error occurs, a program terminates abnormally, or a program does not provide an expected result, the source code must be analyzed and corrected.

This section describes the debug features to analyze the problems. Note that this section describes the debugging only for the OpenMP programs. Also refer to "Chapter 8 Debugging".

12.3.6.1 Checking Features for Debugging

This section describes the restriction of debugging features.

- The compiler option -H, which provides the checking feature for debugging, is not effective within the parallel region (including dynamic parallelization).

12.4 I/O Buffer Parallel Transfer

The I/O buffer parallel transfer is a feature that provides parallel threading to improve the performance of data transfer between the buffer area used in the array area and the input-output statement in a Fortran program.

12.4.1 Compilation and Execution

This section describes the procedures and criteria to compile and execute a program using I/O buffer parallel transfer.

12.4.1.1 Compilation

To perform I/O buffer parallel transfer, specify the compiler option -Kparallel or -Kopenmp.

The compile time option -Kparallel is an option to perform automatic parallelization. See Section "12.2 Automatic Parallelization" for details.

The compiler option -Kopenmp is an option to perform parallelization based on the OpenMP specifications. See Section "12.3 Parallelization by OpenMP Specifications" for details.

12.4.1.2 Execution Process

To use I/O buffer parallel transfer, the following configuration is required:

- - Specify MP for the environment variable FLIB_IOBUFCPY.
- - Specify the number of threads using the environment variable PARALLEL or OMP_NUM_THREADS.

  See Section "12.2.1.2.2 Number of Threads" for information on the number of threads.

12.4.1.3 Conditions to Perform I/O Buffer Parallel Transfer

The following conditions must be met to perform I/O buffer parallel transfer:

- Target I/O statements are unformatted I/O statements
- Target I/O statements are outside the parallel region or outside the automatically parallelized area
- When the environment variable FLIB_IOBUFCPY_SIZE is not specified, the amount of the data transferred and the I/O buffer value must be more than "NumberOfThread + 30" Kbyte.
- When size is specified on the environment variable FLIB_IOBUFCPY_SIZE, the amount of the data transferred and the I/O buffer value must be more than "size" Kbyte.

For sequential access I/O statements, specify the I/O buffer size using the runtime option -g or the environment variable fuxxbf. For direct access I/O statements, specify using the recl specifier of the open statement and the runtime option -d.

See "3.3 Runtime Options" for more information on runtime option -g.

See Section "3.8 Using Environment Variables for Execution"for information on the environment variables fuxxbf and FLIB_IOBUFCPY_SIZE.

12.4.1.4 Compilation execution example

```
The used approximate stack size for each thread is displayed per Byte.
$ cat test.f
  character ch((30+4)*1024)
  ch='X'
```

```
  open(10,form="unformatted")
  write(10) ch
  close(10)
  end
$ frtpx -Kparallel test.f
$ export FLIB_IOBUFCPY="MP"
$ export PARALLEL=4
$ ./a.out -Wl,-g34
```

This specifies MP for the environment variable FLIB_IOBUFCPY to enable I/O buffer parallel transfer. It also sets 4 for the environment variable PARALLEL to make the number of threads 4. The runtime option -g34 sets the I/O buffer size to 34 Kbytes (Number of thread 4 + 30 Kbytes).

12.4.2 Notes on I/O Buffer Parallel Transfer

When the I/O buffer size or the transferred data size is small, the performance may deteriorate even if I/O buffer parallel transfer is performed.

Appendix A Limits and Restrictions

The Fortran compiler restricts the size and complexity of source programs. Some restrictions apply to one Fortran statement and others apply to a combination of several statements.

A.1 Nesting Level of Functions, Array Sections, Array Elements, and Substring References

In a combination of function, array section, array element, and substring references, nesting must not exceed 255 levels. Depending on the compiler, nesting may be subject to further restrictions.

Example: Nesting level of functions and array element references

```
INTEGER IA(10)
EXTERNAL FUN,IFUN
   :
X=FUN(IFUN(IA(1))) ! The nesting level is 3
```

A.2 DO, CASE, IF, SELECT TYPE, BLOCK, CRITICAL, and ASSOCIATE Constructs

In a combination of DO, CASE, IF, SELECT TYPE, BLOCK, CRITICAL, and ASSOCIATE constructs, the total nesting level must not exceed 50.

Example: Nesting levels of DO and IF constructs

```
 DO I=1,10
    A(I)=A(I)+1    ! The nesting level is 1
    IF (I.LE.5)THEN
       CALL S! The nesting level is 2
       DO J=1,5
          B(J)=A(J)! The nesting level is 3
       END DO
    CALL T! The nesting level is 2
    END IF
 END DO
```

A.3 Implied DO-Loops

The nesting level of implied DO-loops in input/output or DATA statements must not exceed 25.

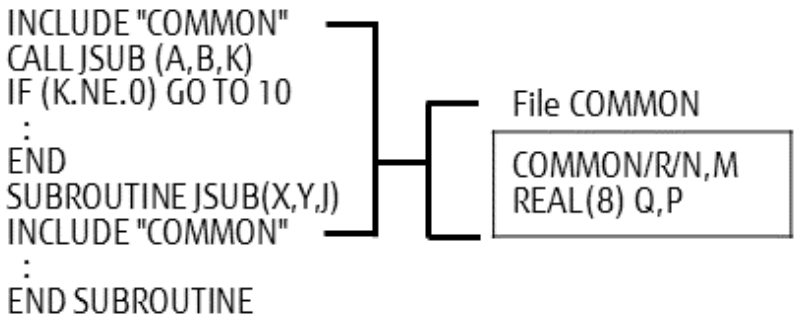
Example: Nesting level of implied DO-loops

```
PRINT *, ((A(I,J),I=1,2),J=1,3) ! The nesting level is 2
```

A.4 INCLUDE Line

The nesting level of INCLUDE lines must not exceed 16.

Example: Nesting level of INCLUDE lines



In this case, the nesting level of INCLUDE lines is 1.

A.5 Array Declarations

For array declarations, there are two restrictions.

A.5.1 Restriction for All Array Declarations

The compiler calculates T for each array declaration to reduce the number of calculations needed for array sections or array element addresses. The absolute value of T obtained by following formula must not exceed the limitation value, and the absolute value must not exceed the limitation value during calculation:

$$T = l_1 * s + \sum_{i=2}^{n} \{ l_i * (\prod_{m=2}^{i} d_{m-1} * s) \}$$

```
n : Array dimension number
s : Array element length
l : Lower bound of each dimension
d : Size of each dimension
T : Value calculated for the array declaration
```

The limitation value is 9223372036854775807.

Example: Restrictions on array declarations

```
PARAMETER (IL=10000000,IU=10000999)
INTEGER(8) A(IL:IU,IL:IU,IL:IU,IL:IU,IL:IU)
```

Because T (the value calculated for array declarations) for array A is 80080080080080000000, this array declaration must not be specified.

A.5.2 Restriction for Array Declaration of Zero-Sized Array

In an array declaration of zero-sized array that has bounds that are nonconstant specification expressions, the absolute value of T obtained by following formula must not exceed the limitation value, and the absolute value must not exceed the limitation value during calculation:

$$T = s * (\prod_{i=1}^{n} u_i - l_i + 1)$$

```
n : Array dimension number
s : Array element length
```

```
l : Lower bound of each dimension
u : Upper bound of each dimension
T : Value calculated for the array declaration
```

The limitation value is 9223372036854775807.

A.6 Array Section

In the following subscript triplet,

```
subscript1 : subscript2 : stride
```

overflow must not occur during calculation.

```
(subscript2 - subscript1 + stride) / stride
```

A.7 Distance between a Branch Instruction and the Branch Destination Instruction

The immediate field size of the branch destination address in a branch instruction is restricted. For this reason, when the branch instruction is separated from the branch destination instruction over a distance that exceeds a certain fixed value, the assembler may not be able to assemble the program. In this case, the assembler displays the following messages:

```
/var/tmp/asmAAAa006jR.s: Assembler messages:
/var/tmp/asmAAAa006jR.s:10: Error: relocation overflow
```

The likelihood for this symptom becomes high when the program is assembled with the -H and -Kthreadsafe options specified simultaneously for a loop with several thousand or more executable statements or when there are numerous array expressions in a huge loop.

If this symptom occurs, correct the program (e.g., perform loop splitting) or -O0 option is in effect. It is necessary to note it for the execution performance because -O0 option suppresses the optimization.

Appendix B Printing Control Command

The printing control command, fot(1) and fotpx(1) convert files output by Fortran programs into standard I/O files for printing on a line printer. You can use fot(1) and fotpx(1) to map from Fortran output to the printer. That is, they map files or pipes that have Fortran carriage-control conventions into a form acceptable to conventional printers.

The first character of a formatted record provides the following control information.

Character	Action
Blank	Advance to the next line and print the line.
0	Advance two lines and print the line.
1	Advance to the first line of the next page and print the line.
+	Do not advance to the next line before printing the line (over-print the current line).

If a formatted record is empty, the first character of the line is regarded as the control character blank.

If the first character of a formatted record is neither a blank, 0, 1, nor +, a diagnostic message is output, and the first character is regarded as the control character blank.

B.1 Command Format

The format of the fot(1) command and fotpx(1) command is as follows:

Command	Command argument
fot	[_-help] [_*file1*] [_*file2*]
fotpx	

_: At least one blank is required.

-help: Usage of fot(1) and fotpx(1) is output.

file1: The *file1* is an input file. The default input file is the standard input file.

file2: The *file2* is an output file. The default output file is the standard output file.

Examples are as follows:

1. File names in the command line are omitted (redirection)

```
$ fot <infile >outfile
```

2. File names in the command line are omitted (pipe)

```
$ command1 | fot | command2
```

3. File names in the command line are specified

```
$ fot infile outfile
```

4. Usage of fot(1) command is output

```
$ fot -help
```

B.2 Example of fot Command and fotpx Command

An example of fot(1) command is as follows:

```
$ cat infile
_aaaa
```

```
1bbbb
cccc
0dddd
_eeee
+ff
$ fot infile | lpr
fot: invalid 1 line carriage control conventions in infile
```

_: Blank

Printing result:

```
aaaa Page 1
bbbb Page 2
cccc

dddd
ffee
```

B.3 Return Values of fot Command and fotpx Command

The following table lists the return values set by the fot(1) command and fotpx(1) command.

Return value	Status
0	Normal termination
1	fot(1) command or fotpx(1) command error

B.4 Diagnostic Messages of fot and fotpx

The fot command and fotpx command output diagnostic messages if the invalid file name is specified or the first character of the line of the input file is invalid.

Messages issued by fot command and fotpx command are in the following format:

```
fot: Message-text
```

```
fotpx: Message-text
```

Appendix C Endian Convert Command

The endian convert commands fcvendian(1) and fcvendianpx(1) swap all bytes in type values within input file, and output the swapped values to the output file.

C.1 Command Format

The format of the fcvendian(1) command and fcvendianpx(1) command is as follows:

Command	Command argument
fcvendian	*_file1_file2_type*
fcvendianpx	

_: At least one blank is required.

file1: An input file with values to swap.

file2: An output file to output the swapped values.

type: A type of values to swap (The value which can be specified is 2, 4, 8, or 16).

If no command arguments are specified, usage of fcvendian(1) and fcvendianpx(1) is output.

Examples are as follows:

Example1: To byteswap a file "in.data" with 8 byte values.

```
$ fcvendian in.data out.data 8
```

Example2: To output the usage of the fcvendian command.

```
$ fcvendian
```

C.2 Return Value of fcvendian(1) and fcvendianpx(1)

The following table lists the return values set by the fcvendian(1) command and the fcvendianpx(1) command.

Return value	Status
0	Normal termination
1	fcvendian(1) or fcvendianpx(1) command error

C.3 Diagnostic Messages of fcvendian(1) and fcvendianpx(1)

If fcvendian(1) is not successful completion, the following diagnostics message is output to standard error file.

```
Usage: fcvendian infile outfile type
```

```
fcvendian: Invalid operand
```

```
fcvendian: Input file open error
```

```
fcvendian: Output file open error
```

```
fcvendian: Memory allocate error
```

If fcvendianpx(1) is not successful completion, the following diagnostics message is output to standard error file.

```
Usage: fcvendianpx infile outfile type
```

```
fcvendianpx: Invalid operand
```

```
fcvendianpx: Input file open error
```

```
fcvendianpx: Output file open error
```

```
fcvendianpx: Memory allocate error
```

C.4 Note

If the file which the type of values is mixed in is swapped, the byteswap may be not successful completion.

Appendix D Job Operation Software

In this appendix, the compilation and execution processes for using high-speed facility on Job Operation Software of Technical Computing Suit. See the "Job Operation Software End-user's Guide" for details on Job Operation Software.

D.1 Management of the CPUs resource

The CPUs resource that can be used is strictly managed on the job of Job Operation Software.

Number of CPUs that can be used on the job

This is the number of CPUs that can be used for a process on the job. See Section "D.1.1 Number of CPUs that can be used on the Job" for details.

Limits of Number of CPUs

The number of CPUs that can be used on the job is defined as the limit of the number of CPUs.

The number of threads when the program is executed is decided in consideration of the limit values of the number of CPUs. See Section "12.2.1.2.2 Number of Threads" for details of the number of the threads for the automatic parallelization. See "Number of Threads" in Section "12.3.1.2.3 Notes during Execution" for details of the number of threads for OpenMP.

D.1.1 Number of CPUs that can be used on the Job

The limit of the number of CPUs for one process is decided in consideration of CPUs resource and the number of processes in the virtual node on the job. This value is defined as the number of CPUs that can be used on the job.

Details are shown as follows.

"The number of CPUs that can be used on job" = cpunum_on_node / procnum_on_node

cpunum_on_node	The number of CPUs that can be used on one virtual node. 1 to the number of all CPUs on one virtual node.
procnum_on_node	The number of processes on one virtual node. For thread parallel processing jobs, it is 1. For hybrid (process and thread) parallel processing jobs, it is 1 to the number of all CPUs on one virtual node.

It is rounded down when it cannot be divided.

See the "Job Operation Software End-user's Guide" for details on job.

When MPI program is executed with VCOORD_FILE that is set the number of CPUs to a process, the value is the number of CPUs that can be used on the job. See the "MPI User's Guide" for details on MPI.

D.1.2 FLIB_USE_ALLCPU

"The number of CPUs that can be used on the job" can be controlled using the environment variable FLIB_USE_ALLCPU. The value that can be set and the meaning are as follows. The default value is "FALSE".

TRUE	The number of CPUs that can be used on the job is cpunum_on_node. This means all CPUs that can be used on the virtual node is used regardless of the number of processes. Therefore, when the number of processes on the virtual node is two or more, these CPUs resource is shared among the processes. When the frequency calculated at the same time in each process is low etc. , the improvement of the execution performance can be expected. In general use, the improvement of the execution performance cannot be expected. You should use the following functions. - -Kopenmp is used at compile time. - FLIB_FASTOMP=FALSE is defined at run time.

	- FLIB_SPINWAIT=0 is defined at run time. When the above-mentioned is not used, the performance might be greatly downed like the dead lock. See "Chapter 12 Multiprocessing" for details of -Kopenmp,FLIB_FASTOMP and FLIB_SPINWAIT. When procnum_on_node is 1, "TRUE" and "FALSE" have the same effect. When procnum_on_node is not 1, the on-chip hardware barrier and the sector-cache cannot be used. See Section "D.1.1 Number of CPUs that can be used on the Job" for details about cpunum_on_node and procnum_on_node. See Section "D.3 On-chip Hardware Barrier" for details the on-chip hardware barrier, and Section "D.4 Sector-cache" for details about the sector-cache.
FALSE	The number of CPUs that can be used is Section "D.1.1 Number of CPUs that can be used on the Job".

When MPI program is executed with VCOORD_FILE that is set the number of CPUs to a process, the value is ignored. See the "MPI User's Guide" for details on MPI.

D.1.3 FLIB_USE_CPURESOURCE

The environment variable FLIB_USE_CPURESOURCE controls the CPU resource manager on the job.

The allowed values their meanings are as follows. The default value is "TRUE".

TRUE	The CPU resources on the job are managed.
FALSE	The CPU resources on the job are not managed. Therefore the number of CPUs is not managed, and you cannot use the following functions. - D.2 CPU Binding - D.3 On-chip Hardware Barrier - D.4 Sector-cache

D.2 CPU Binding

The thread is bound to one CPU when the program using the automatic parallelization or OpenMP is executed on the job. CPUs used can be strictly managed as described in Section "D.1 Management of the CPUs resource". Therefore the improvement of the performance can be expected by CPU Binding.

See "Chapter 12 Multiprocessing for details of Multiprocessing.

D.2.1 FLIB_CPUBIND

The environment variable FLIB_CPUBIND controls the CPU Binding for thread. The following value can be set to FLIB_CPUBIND. The default value is "chip_pack".

chip_pack	All threads are bound to same CPU Chip as much as possible.
chip_unpack	Each thread is bound to a different CPU Chip as much as possible.
off	Thread is not bound to CPU. If you control CPU Binding by yourself, this must be set.

When only one CPU Chip can be used, "chip_pack" and "chip_unpack" have the same effect.

When only one CPU on a CPU Chip can be used, "chip_pack" and "chip_unpack" have the same effect.

D.3 On-chip Hardware Barrier

Since SPARC64VIIIfx, the SPARC64 processor has the on-chip hardware barrier. On the job, it can be used as the thread barrier. The on-chip hardware barrier is a hardware mechanism which facilitates high speed synchronization among threads in a CPU Chip, and improves the execution performance.

D.3.1 Compilation

When generating the program that uses the on-chip hardware barrier as the thread barrier, the special compiler option is unnecessary. It is automatically used when executing in the environment that can be used. The software barrier is used when executing in the environment that cannot be used.

The program generated using compiler option neither -Kparallel nor -Kopenmp doesn't use the thread barrier. For details about Execution, see Section "D.3.2 Execution.

D.3.2 Execution

The on-chip hardware barrier can be used only on the job. The on-chip hardware barrier is a hardware mechanism for synchronization among threads in a CPU Chip.

See the "Job Operation Software End-user's Guide" for detail of the job.

D.3.3 FLIB_CNTL_BARRIER_ERR

When the on-chip hardware barrier cannot be used, the following messages are output, and the processing is continued using the software barrier.

```
jwe1050i-w The hardware barrier couldn't be used and continues
           processing using the software barrier.
```

This diagnostic message (jwe1050i-w) error can be controlled using environment variable FLIB_CNTL_BARRIER_ERR.

The value that can be set and the meaning are as follows. Default is "TRUE".

TRUE	The diagnostic message (jwe1050i-w) error is detected. When the on-chip hardware barrier cannot be used, this messages are output, and the processing is continued using the software barrier.
FALSE	The diagnostic message (jwe1050i-w) error is not detected. When the on-chip hardware barrier cannot be used, the processing is continued using the software barrier.

D.3.4 FLIB_NOHARDBARRIER

Using the on-chip hardware barrier can be controlled using environment variable FLIB_NOHARDBARRIER. If the environment variable FLIB_NOHARDBARRIER exists, the on-chip hardware barrier cannot be used. When environment variable FLIB_NOHARDBARRIER is set (A set value is not necessary.), the software barrier is always used. Even if the on-chip hardware barrier can be used, it is not used. The diagnostic message (jwe1050i-w) error always occurs when environment variable FLIB_NOHARDBARRIER is set.

See Section "D.3.3 FLIB_CNTL_BARRIER_ERR" for details.

D.3.5 Notes

- When the on-chip hardware barrier is used by the program made using compile option -Kopenmp, the environment variable FLIB_FASTOMP shall not be FALSE. See Section "12.3 Parallelization by OpenMP Specifications" for the meaning and directions of the environment variable FLIB_FASTOMP.
- When the on-chip hardware barrier is used, CPU Binding is needed. Therefore, when "off" is set for environment variable FLIB_CPUBIND, the on-chip hardware barrier cannot be used. See Section "D.2 CPU Binding" for details of CPU Binding.
- When the number of threads is 1, the on-chip hardware barrier cannot be used. See Section "12.2.1.2.2 Number of Threads" or Section "12.3.1.2.3 Notes during Execution" for the number of threads.

- When the number of CPUs to a process is three or less, a on-chip hardware barrier may not be able to use. The number of CPUs to a process should be set to four or more.
- When the job is a Swap Target Job and a Node-sharing Job, and the number of CPUs to a process is three or less, an on-chip hardware barrier cannot be used. The number of CPUs to a process should be set to four or more. See the "Job Operation Software End-user's Guide" for details on job.

D.3.6 Usage example on the job

Example 1: A program that uses the on-chip hardware barrier is executed on a thread parallel processing job.

```
$ cat job.sh
#!/bin/sh
./a.out
$ pjsub -L "node=1" job.sh
```

Example 2: A program that uses the on-chip hardware barrier is executed on a hybrid (process and thread) parallel processing job (One job is executed in one dimension: The number of nodes is set to the number of processes.).

```
$ cat job.sh
#!/bin/sh
#PJM -L "node=12"
#PJM --mpi "shape=12"
#PJM --mpi "rank-map-bynode"
mpiexec -n 12 a.out
$ pjsub job.sh
```

See the "Job Operation Software End-user's Guide" for detail of the job.

D.4 Sector-cache

Since SPARC64VIIIfx, the SPARC64 processor has Sector-cache. On the job, the Sector-cache can be controlled.

See Section "9.17 Software Control of Sector Cache" about how to control the Sector-cache. Here, it explains only notes in execution.

D.4.1 Notes in execution

When only one process is executed on a NUMA node, the control of the Sector-cache is possible.

See Section "D.1 Management of the CPUs resource" and the "Job Operation Software End-user's Guide" for details.

When it is not possible to execute by one process alone on a NUMA node and the Sector-cache control function is used, the diagnostic message jwe1047i-w is output as follows, the function is invalidated and processing is continued.

```
jwe1047i-w A sector cache couldn't be used.
```

On the following cases, Runtime Library cannot determine whether one process can reserve a NUMA node or not. Therefore behavior of sector cache is indeterminate. Performance of program may be reduced, or program may be ended with diagnostic-message jwe1048i-u.

- Process is generated by functions other than Fujitsu MPI.

  Example:

  - FORTRAN FORK/SYSTEM/SH service function is used.
  - C fork/system function is used.

- Threads are generated by functions other than Fujitsu Automatic Parallelization/OpenMP.

  Example:

  - Threads are controlled by pthread functions.

To invalidate the Sector-cache control function unconditionally regardless of the execution environment, "FALSE" is set to environment variable FLIB_SCCR_CNTL.

See Section "9.17.2.2 Software control with environment variables and optimization control rows" for details.

D.5 Large-Page Function

The large-page function enlarges the page size which is the unit of the memory management of hardware. If the page size is enlarged, TLBmiss reduces and the program runs faster.

The large-page function is enable for the executable program which is created specifying the -Klargepage option when linking. The stack segment, data segment, heap segment and thread stack segment are enlarged for the executable program.

If the page size is enlarged, the used memory size may increase. If it increases the memory used by a command or a daemon, it may affect the execution environment. The memory can decrease when a command or a daemon is created with specify the -Knolargepage option, and the effect of the execution environment can be decreased.

See "lpgparm(1)" for information of default size of large-page and how to change the large-page size.

Appendix E Preprocessor

This appendix explains the C language preprocessor and the Fortran preprocessor.

When a suffix of the source file is .F, .FOR, .F77, .F90, .F95 or .F03, or when the -Cpp option is specified, the preprocessor is executed. In this case, the -Ccpp or -Cfpp option can be specified to select the C language preprocessor or the Fortran preprocessor. The Fortran preprocessor is selected by default.

For the details of the preprocessor options, see 2.2 Compiler Options.

E.1 C language preprocessor

The C language preprocessor preprocesses in conformance with C language specification.

E.2 Fortran preprocessor

The Fortran preprocessor gives more priority to the Fortran language specification than to the C language specification, and it preprocesses considering Fortran constructs, such as comment and continuation line indicator. Be careful to the followings:

- The Fortran constructs, such as continuation line indicator, comment, fixed and free source form and etc, are considered.
- The Fortran comment construct is also a target of macro substitution. However, the message is not output even if the error is in Fortran comment construct.
- The Hollerith constant and H edit descriptor are not considered as character contexts, and are the targets of macro substitution.
- The B, O, X, Z, b, o, x, and z which represent binary, octal and hexadecimal constants are not affected by preprocessing directives, and are not the targets of macro substitution.
- Because the line number may be changed by the preprocessing of a Fortran source programs, be careful when this option is used. In this case, specify -P option and refer to the produced preprocessed output file.
- When the line exceeds limited column by the preprocessing directives and when -P option is specified, the control lines, such as "#line !fpp start" and "#line !fpp end", may be produced in the produced preprocessed output file. When the preprocessed output file is specified as input file to the compiler, do not change those control lines because they are used by the compiler.

The following appendixes explain the features of the Fortran preprocessor with examples.

E.2.1 Continuation line of Fortran

The continuation line of Fortran is considered.

- In fixed source form, column 6 indicates a continuation line. It is analyzed as the last character of the previous line continues to the column 7 of the continuation line.

```
Example: aaa is substituted to bbb.
   #define aaa bbb
   !23456789
           aa
           *a=1
```

- In free source form, the last character & and the first character & are analyzed as a character to indicate a continuation line. It is analyzed as the character before & continues to the character after &.

```
Example: aaa is substituted to bbb.
   #define aaa bbb
   aa&
     &a=1
```

E.2.2 Comment line of Fortran

The comment line of Fortran is considered.

- The comments of C language and Fortran can be specified. The comment of C language form is substituted to blanks.

```
Example: The comments of C language and Fortran are effective.
/* comment of C language
    comment of C language */
  ! comment of Fortran
```

E.2.3 Comment of Fortran and comment of C language

- When the beginning of C language comment "/*" is specified in the Fortran comment, the beginning of the C language comment is not effective.

```
Example: The beginning of C language form comment is not
           effective.
  ! /* comment of Fortran
  Not comment of C language */
```

- When the comment of Fortran is specified in the C language comment, the comment of Fortran is not effective.

```
Example: Comment of Fortran is not effective and assignment
           statement is effective.
    /* ! */ kkk=1
```

E.2.4 Macro substitution in Fortran comment construct

The macro substitution in Fortran comment construct is considered.

- The macros in Fortran comment construct are the targets of substitution. It is the specification that considers that the OpenMP prefix and the optimization control line and etc begin with comment character.

```
Example: aaa in Fortran comment construct is substituted to bbb.
   #define aaa bbb
   !ocl noarraypad(aaa)
           integer::aaa(100)
           aaa=10

   ! aaa=20
   !$omp parallel shared(aaa)
       :
```

- A message is not output even if the macro substitution error is detected in the Fortran comment construct.

```
Example: A message is not output even if the macro substitution
           error is detected in comment.
   #define a(i) b(2,i)
   a(5,10)=1 ! message is output.
   ! a(5,10)=1 message is not output.
```

E.2.5 Treatment of the characters specified after column 72 in fixed source form

In fixed source form, the characters specified after column 72 are ignored except comment and preprocessing directives. However, when -Fwide option is specified, columns 73 through 255 are effective.

```
Example: In fixed source form, the character C is specified at column 73,
           ABC is considered as AB, and ABC is not substituted to D.
           Specify -Fwide option in order to make columns 73 through 255
           effective.
#define ABC D
!         1           2           3           4           5          6           7
!234567890123456789012345678901234567890123456789012345678901234567890123456789012
```

```
                                                                      ABC
      * = 100
      print *,D
      end
```

Appendix F Notes on Migration from FX10 System to FX100 System

This appendix provides notes on migrating from FX10 system (Generation Number:09 or later) to FX100 system.

For migrating from FX10 system (Generation Number:08 or earlier), refer to "Appendix G Compatibility Information (FX10 System)" also.

F.1 Error check in OpenMP standard at compilation time

This note corresponds to the migration from FX10 system (Generation Number:11 or earlier).

Refer to "G.1.1 Error check in OpenMP standard at compilation time" about this note.

F.2 Error check in Fortran 2003 standard at compilation time

This note corresponds to the migration from FX10 system (Generation Number:11 or earlier).

Refer to "G.1.2 Error check in Fortran 2003 standard at compilation time" about this note.

F.3 Enhancement intrinsic procedures and intrinsic module

This note corresponds to the migration from FX10 system (Generation Number:11 or earlier).

Refer to "H.1.3 Enhancement intrinsic procedures and intrinsic module" about this note.

F.4 The option which the compiler option -Kuxsimd needs is changed

a. Changes

The option which the compiler option -Kuxsimd needs is changed.

[Previous version]

The compiler option -Kuxsimd needed only the compiler option -Ksimd.

[This version]

The compiler option -Kuxsimd needs the compiler option -KHPC_ACE in addition to the compiler option -Ksimd.

b. Influence

When the compiler option -KHPC_ACE is not specified, the compiler option -Kuxsimd is invalidated and the following message is displayed.

```
frtpx: -Kuxsimd is invalidated because -KHPC_ACE is not specified.
```

c. Coping

Specify the compiler option -KHPC_ACE.

F.5 The function for canceling the program compilation that is forecasted to take a long time

This note corresponds to the migration from FX10 system (Generation Number:10 or earlier).

Refer to "G.2.1 The function for canceling the program compilation that is forecasted to take a long time" about this note.

F.6 The compiler option -K{ordered_omp_reduction|noordered_omp_reduction} is changed to -K{openmp_ordered_reduction|openmp_noordered_reduction}

This note corresponds to the migration from FX10 system (Generation Number:10 or earlier).

Refer to "G.2.2 The compiler option -K{ordered_omp_reduction|noordered_omp_reduction} is changed to -K{openmp_ordered_reduction|openmp_noordered_reduction}" about this note.

F.7 Abolition of compilation options and the optimization control specifiers

1. -K{FLTLD|NOFLTLD} options and the optimization control specifiers {FLTLD|NOFLTLD}

   [Note] The following description is changed only when the compiler option -KHPC_ACE2 is set.

   a. Changes

      The -K{FLTLD|NOFLTLD} options are abolished.

      The optimization control specifiers {FLTLD|NOFLTLD} are abolished.

      [Previous version]

      The -K{FLTLD|NOFLTLD} options were effective.

      The optimization control specifiers {FLTLD|NOFLTLD} were effective.

      [This version]

      The -K{FLTLD|NOFLTLD} options are invalid.

      The optimization control specifiers {FLTLD|NOFLTLD} are invalid.

   b. Influence

      The -K{FLTLD|NOFLTLD} options become invalid. If these options are specified, the following message is output.

      ```
      frtpx: -K[NO]FLTLD is invalidated because -KHPC_ACE is not specified.
      ```

      And, the optimization control specifiers {FLTLD|NOFLTLD} become invalid. If these specifiers are specified, the following message is output.

      ```
      jwd1202i-i  "filename", line number: Invalid optimization control specifier in optimization
      control line.
      ```

      When the -K{FLTLD|NOFLTLD} options and the optimization control specifiers {FLTLD|NOFLTLD} become invalid, a normal load instruction is used.

   c. Coping

      Specify the -K{nf|nonf} options or the optimization control specifiers {NF|NONF}.

      Although objects are compiled with the -KNOFLTLD option or the optimization control specifier NOFLTLD, it is possible to execute the objects on SPARC64 XIfx by using non-faulting-load instruction.

2. -K*cpu* option

   a. Changes

      The -K*cpu* option is abolished.

      [Previous version]

      The -K*cpu* option was effective.

[This version]

The -K*cpu* option is invalid.

b. Influence

The -K*cpu* option becomes invalid. If this option is specified, the following warning is output.

```
warning: Invalid suboption cpu specified for -K.
```

c. Coping

It is not necessary to action.

3. -K{prefetch_double_line_L2|prefetch_nodouble_line_L2} options and the optimization control specifiers {PREFETCH_DOUBLE_LINE_L2|PREFETCH_NODOUBLE_LINE_L2}

a. Changes

The -K{prefetch_double_line_L2|prefetch_nodouble_line_L2} options are abolished.

The optimization control specifiers {PREFETCH_DOUBLE_LINE_L2|PREFETCH_NODOUBLE_LINE_L2} are abolished.

[Previous version]

The -K{prefetch_double_line_L2|prefetch_nodouble_line_L2} options were effective.

The optimization control specifiers {PREFETCH_DOUBLE_LINE_L2|PREFETCH_NODOUBLE_LINE_L2} were effective.

[This version]

The -K{prefetch_double_line_L2|prefetch_nodouble_line_L2} options are invalid.

The optimization control specifiers {PREFETCH_DOUBLE_LINE_L2|PREFETCH_NODOUBLE_LINE_L2} are invalid.

b. Influence

The -K{prefetch_double_line_L2|prefetch_nodouble_line_L2} options become invalid. If these options are specified, the following warning is output.

```
warning: Invalid suboption prefetch_[no]double_line_L2 specified for -K.
```

And, the optimization control specifiers {PREFETCH_DOUBLE_LINE_L2|PREFETCH_NODOUBLE_LINE_L2} become invalid. If these specifiers are specified, the following warning is output.

```
jwd1202i-i  "filename", line number: Invalid optimization control specifier in optimization
control line.
```

c. Coping

It is not necessary to action.

4. -K{simd_region_constant|nosimd_region_constant} options

a. Changes

This note corresponds to the migration from FX10 system (Generation Number:07 or later).

The -K{simd_region_constant|nosimd_region_constant} options are abolished.

[Previous version]

The -K{simd_region_constant|nosimd_region_constant} options were effective.

[This version]

The -K{simd_region_constant|nosimd_region_constant} options are invalid.

b. Influence

The -K{simd_region_constant|nosimd_region_constant} options become invalid. If these options are specified, the following warning is output.

```
warning: Invalid suboption [no]simd_region_constant specified for -K.
```

c. Coping

It is not necessary to action.

F.8 The -Komitfp option is induced by -Kfast, and maintenance of the trace back information

This note corresponds to the migration from FX10 system (Generation Number:09 or earlier).

Refer to "G.3.1 The -Komitfp option is induced by -Kfast, and maintenance of the trace back information" about this note.

F.9 Change of value of macro along with the compiler version up

This note corresponds to the migration from FX10 system (Generation Number:09 or earlier).

Refer to "G.3.2 Change of value of macro along with the compiler version up" about this note.

F.10 Change of the name of temporary object file and the directory to store

This note corresponds to the migration from FX10 system (Generation Number:09 or earlier).

Refer to "G.3.3 Change of the name of temporary object file and the directory to store" about this note.

F.11 Change the default value of the -Ksimd[=level] option

a. Changes

The default value of the -Ksimd[=*level*] option is changed.

[Previous version]

1. If a value was not specified for *level* of the -Ksimd[=*level*] option, -Ksimd=1 was set.
2. When the -O2 option or higher was set, the -Ksimd=1 option was induced. (*1)
3. If a value was not specified for *level* of the -Ksimd[=*level*] option, the -Q or -Nlst option output -Ksimd=1.

[This version]

1. If a value is not specified for *level* of the -Ksimd[=*level*] option, -Ksimd=auto is set.
2. When the -O2 option or higher is set, the -Ksimd=auto option is induced. (*1)
3. If a value is not specified for *level* of the -Ksimd[=*level*] option, the -Q or -Nlst option output -Ksimd=auto.

*1) When the -Kfast option is specified, the -O3 option is set. When the -O[*n*] option is not specified, the -O2 option is set.

b. Influence

SIMD extensions are promoted for loops that contain IF constructs in the following cases:

- The *level* value of the -Ksimd[=*level*] option is not specified. And,
- The -O2 option or higher is set.

c. Coping

To keep the SIMD condition the same as the previous version, specify the -Ksimd=1 option.

F.12 Specification change of the optimization control specifiers SIMD and UXSIMD

[Note] The following description is changed only when the compiler option -KHPC_ACE2 is set.

a. Changes

Specifications of the SIMD and UXSIMD specifiers are changed.

Hardware restrictions of data alignment for SIMD store instructions on FX100 system are relaxed.

Therefore, SIMD store instructions are usable without the ALIGNED or UNALIGNED specifier which is following the SIMD and UXSIMD specifiers.

[Previous version]

Previous specifications of the SIMD and UXSIMD specifiers were as follows.

```
!OCL SIMD[(ALIGNED | UNALIGNED)]
```

```
!OCL UXSIMD[(ALIGNED | UNALIGNED)]
```

[This version]

Specifications of the SIMD and UXSIMD specifiers are as follows.

```
!OCL SIMD
```

```
!OCL UXSIMD
```

b. Influence

The compiler ignores ALIGNED and UNALIGNED specifiers following the SIMD and UXSIMD specifiers.

c. Coping

It is not necessary to action.

Appendix G Compatibility Information (FX10 System)

This appendix provides compatibility information as notes on migrating.

G.1 Migrating to V2.0L20(Generation Number:12)

G.1.1 Error check in OpenMP standard at compilation time

a. Changes

The error messages are output at compilation time for incorrect program in OpenMP standard.

- Diagnostic message jwd1838i-s is output at compilation time under the following conditions:
  1. -Kopenmp option is specified. And,
  2. A common block name appears in SAVE statement. And,
  3. The common block appears in THREADPRIVATE directive. And,
  4. The common block does not appear in COMMON statement.
- Diagnostic message jwd1837i-s is output at compilation time under the following conditions:
  1. -Kopenmp option is specified. And,
  2. A common block is declared in COMMON statement in a module. And,
  3. The common block appears in THREADPRIVATE directive. And,
  4. The module name appears in USE statement in another program unit. And,
  5. In that program unit, the common block appears in COPYIN or COPYPRIVATE clause. And,
  6. In that program unit, the common block name does not appear in COMMON statement.
- Diagnostic message jwd2038i-s is output at compilation time under the following conditions:
  1. -Kopenmp option is specified. And,
  2. An automatic object is declared in a subprogram. And,
  3. The variable in specification expression of the automatic object appears in THREADPRIVATE directive.

[Previous version]

Diagnostic messages jwd1838i-s, jwd1837i-s, or jwd2038i-s was not output at compilation time and object was created.

[This version]

Diagnostic messages jwd1838i-s, jwd1837i-s, or jwd2038i-s is output at compilation time.

b. Influence

Diagnostic messages jwd1838i-s, jwd1837i-s, or jwd2038i-s is output at compilation time and object is not created.

c. Coping

Modify program as following:

- Make common blocks in COPYIN clause, COPYPRIVATE clause, or THREADPRIVATE directive, specified in COMMON statement in same scoping unit. Or,
- Do not reference local variable in specification expression.

G.1.2 Error check in Fortran 2003 standard at compilation time

a. Changes

The error messages are output at compilation time for incorrect program in the Fortran 2003 standard.

- Diagnostic message jwd1034i-s is output at compilation time under the following conditions:
  1. Procedure declaration statement appears. And,
  2. Declaration type specifier in interface of the procedure declaration statement. And,
  3. Letter or digit appears after the declaration type specifier.
- Diagnostic message jwd1118i-s is output at compilation time under the following conditions:
  1. Procedure declaration statement appears. And,
  2. CHARACTER intrinsic type specifier appears in interface of the procedure declaration statement. And,
  3. KIND= appears after the type specifier. And,
  4. LEN= does not appear after the type specifier. And,
  5. Right parenthesis does not appear after the type specifier.
- Diagnostic message jwd1176i-s or jwd1343i-s is output at compilation time under the following conditions:
  1. Assumed size array is declared. And,
  2. Lower bound is omitted and :* appears in last dimension of the assumed size array.
- Diagnostic message jwd1954i-s is output at compilation time under the following conditions:
  1. SELECT TYPE construct or ASSOCIATE construct appears. And,
  2. Brach appears from out of the scope to the inside the scope. And,
  3. Diagnostic message jwd1065i-w is output.
- Diagnostic message jwd1272-s is output at compilation time under the following conditions:
  1. Interface block appears in module program. And,
  2. USE statement specified module of 1 appears in program unit. And,
  3. Interface appears in program unit of 2. And,
  4. Generic specifier appears in the interface of 3. And,
  5. PROCEDURE statement appears in interface body of 3. And,
  6. The procedure of the interface body of 1 appears two times or more in interface body of 3. And,
  7. The generic specifier does not refer.

[Previous version]

Diagnostic message jwd1118i-s, jwd1176i-s, jwd1272i-s, jwd1343i-s, or jwd1954i-s was not output at compilation time and object was created.

[This version]

Diagnostic message jwd1118i-s, jwd1176i-s, jwd1272i-s, jwd1343i-s, or jwd1954i-s is output at compilation time.

b. Influence

Diagnostic message jwd1118i-s, jwd1176i-s, jwd1272i-s, jwd1343i-s, or jwd1954i-s is output at compilation time and object is not created.

c. Coping

Modify program as following:

- Fix the procedure declaration statement.
- Specify lower bound value or modify :* to * in last dimension of assumed size array.
- Do not specify branch to SELECT TYPE construct or ASSOCIATE construct from outside.
- Do not specify same procedure name two times or more in interface of generic specifier.

G.2 Migrating to V2.0L10(Generation Number:11)

G.2.1 The function for canceling the program compilation that is forecasted to take a long time

a. Changes

The compilation is canceled when the compiler forecasts that it takes a long time (24 hours or more as a guide) to compile the program.

[Previous version]

The compilation was not canceled when it took a long time to compile.

[This version]

The compilation is canceled after outputting the following message when the compiler forecasts that it takes a long time to compile.

```
jwd8695i-u The compilation was canceled because it was forecasted to take a long time.
(name:proc-name)
```

proc-name: Procedure name

b. Influence

It may not be able to compile the program which was able to compile previously.

c. Coping

Specify the -Nnocancel_overtime_compilation option at compilation.

G.2.2 The compiler option -K{ordered_omp_reduction| noordered_omp_reduction} is changed to -K{openmp_ordered_reduction|openmp_noordered_reduction}

a. Changes

The compiler option -K{ ordered_omp_reduction | noordered_omp_reduction } is changed to -K{ openmp_ordered_reduction | openmp_noordered_reduction }.

The option which is output by the compiler option -Q or -Nlst is changed from -K{ ordered_omp_reduction | noordered_omp_reduction } to -K{ openmp_ordered_reduction | openmp_noordered_reduction }.

[Previous version]

The -Q or -Nlst option output -K{ ordered_omp_reduction | noordered_omp_reduction }.

[This version]

The -Q or -Nlst option outputs -K{ openmp_ordered_reduction | openmp_noordered_reduction }.

b. Influence

When -K{ ordered_omp_reduction | noordered_omp_reduction } option is specified, the following warning message is output and -K{ openmp_ordered_reduction | openmp_noordered_reduction } option is effective.

```
warning: obsolete option -Kordered_omp_reduction changed -Kopenmp_ordered_reduction.
```

c. Coping

It is not necessary to action. It is recommended to specify the -K{ openmp_ordered_reduction | openmp_noordered_reduction } option.

G.3 Migrating to V2.0L10(Generation Number:10)

G.3.1 The -Komitfp option is induced by -Kfast, and maintenance of the trace back information

a. Changes

The -Komitfp option is induced by -Kfast.

When the -Komitfp option is set, the trace back information is not kept.

[Previous version]

The -Komitfp option was not prepared.

[This version]

The -Komitfp option is added.

The -Komitfp option is induced by -Kfast.

When the -Komitfp option is set, the trace back information is not kept.

b. Influence

When the -Komitfp option is set, the trace back information is not kept.

c. Coping

Specify the -Knoomitfp option to maintain the trace back information, and re-compile.

G.3.2 Change of value of macro along with the compiler version up

a. Changes

The value of macro __frt_version is changed along with the compiler version up.

[Previous version]

The value of the macro __frt_version was 600.

[This version]

The value of the macro __frt_version is 700.

b. Influence

When the value of macro __frt_version is used in the program, the behavior of the program is different from former.

c. Coping

Correct the program to correspond to the value after the change.

G.3.3 Change of the name of temporary object file and the directory to store

a. Changes

The name of temporary object file and the directory to store is changed.

[Previous version]

The name of temporary object file was "input-filename.o" and the directory to store was current directory.

[This version]

The name of temporary object file is unique filename and the directory to store is the temporary directory.

The temporary directory is a directory specified by the environment variable TMPDIR.

When the environment variable TMPDIR is not specified, /tmp is used instead of TMPDIR.

b. Influence

- The capacity of the temporary directory increases temporarily: the temporary object file is removed automatically.
- When two or more input files were specified, their object files had been left in current directory, but now they are not left.

c. Coping

When the capacity of a temporary directory is insufficient, delete unnecessary files.

Specify the compiler option -c to leave object file in current directory.

G.4 Migrating to V1.0L30(Generation Number:09)

G.4.1 Change of values of macro and named constant along with the support of OpenMP API version 3.1 specifications

a. Changes

The values of the following macro and named constant are changed along with the support of the OpenMP API version 3.1 specifications.

- The value of macro _OPENMP is changed.
- The values of named constant openmp_version defined by include file omp_lib.h and module omp_lib are changed.

[Previous version]

- When compile option -Kopenmp was specified, -D_OPENMP=200805 was effective.
- The value of named constant openmp_version was 200805.

[This version]

- When compile option -Kopenmp is specified, -D_OPENMP=201107 is effective.
- The value of named constant openmp_version is 201107.

b. Influence

When the value of macro _OPENMP or named constant openmp_version is used in the program, the behavior of the program is different from former.

c. Coping

Correct the program to correspond to the value after the change.

G.4.2 Enhancement Fortran 2008 intrinsic procedures and intrinsic modules

a. Changes

1. The procedure name IS_CONTIGUOUS without the EXTERNAL attribute become intrinsic procedure.
2. When intrinsic module ISO_FORTRAN_ENV is referred, COMPILER_OPTIONS and COMPILER_VERSION become intrinsic module procedures by ISO_FORTRAN_ENV.
   When intrinsic module ISO_C_BINDING is referred, C_SIZEOF becomes intrinsic module procedure by ISO_C_BINDING.
3. When intrinsic module ISO_FORTRAN_ENV is referred, the following names become named constants defined by ISO_FORTRAN_ENV.
   - CHARACTER_KINDS
   - INT8, INT16, INT32, INT64
   - REAL32, REAL64, REAL128
   - INTEGER_KINDS

- LOGICAL_KINDS
- REAL_KINDS

[Previous version]

1. The procedure name IS_CONTIGUOUS was not intrinsic procedure name.
2. Neither COMPILER_OPTIONS, COMPILER_VERSION nor C_SIZEOF were procedures of the intrinsic module.
3. The following names were not defined by the intrinsic module.
   - CHARACTER_KINDS
   - INT8, INT16, INT32, INT64
   - REAL32, REAL64, REAL128
   - INTEGER_KINDS
   - LOGICAL_KINDS
   - REAL_KINDS

[This version]

1. The procedure name IS_CONTIGUOUS become intrinsic procedure.
2. COMPILER_OPTIONS, COMPILER_VERSION, and C_SIZEOF become procedures of intrinsic module ISO_FORTRAN_ENV or ISO_C_BINDING.
3. The following names become named constants of intrinsic module ISO_C_FORTRAN_ENV.
   - CHARACTER_KINDS
   - INT8,INT16,INT32,INT64
   - REAL32,REAL64,REAL128
   - INTEGER_KINDS
   - LOGICAL_KINDS
   - REAL_KINDS

b. Influence

1. Diagnostic message is output at compilation time when characteristics of procedure (function, subroutine, type of function result, number of argument or type of argument) and IS_CONTIGUOUS are different.
   Program is abnormally ended or result is an incorrect at execution time when all of the characteristics for the procedure are corresponding.
2. The diagnostic message is output at compilation time when the procedure characteristics (function, subroutine, type of the function result, and number of arguments) of COMPILER_OPTIONS, COMPILER_VERSION, or C_SIZEOF are not corresponding, and referring to intrinsic module ISO_FORTRAN_ENV or ISO_C_BINDING.
   Program is abnormally ended or result is an incorrect at execution time when all of the characteristics for the procedure are corresponding.
3. The diagnostic message is output at compilation time when the same user definition name exists as named constants of the additional, and referring to intrinsic module ISO_FORTRAN_ENV.

c. Coping

1. Correct the source program to specify the EXTERNAL attribute for user function IS_CONTIGUOUS.
2. The use association names only specify by ONLY option of intrinsic module ISO_FORTRAN_ENV or ISO_C_BINDING, and must not refer COMPILER_OPTIONS, COMPILER_VERSION, or C_SIZEOF.
3. The use association names only specify by ONLY option of intrinsic module ISO_FORTRAN_ENV, and must not refer the following names.
   - CHARACTER_KINDS

- INT8, INT16, INT32, INT64
- REAL32, REAL64, REAL128
- INTEGER_KINDS
- LOGICAL_KINDS
- REAL_KINDS

G.4.3 Bound remapping list and pointer target of two rank or more in pointer assignment statement

a. Changes

It corresponds to the change in the Fortran 2008 standard.

Diagnostic message jwd2348i-s is output at compilation time under the following conditions:

- A bound remapping list is declared in pointer assignment statement. And,
- The rank of pointer target is two or more. And,
- The pointer target is not simply CONTIGUOUS.

[Previous version]

Diagnostic message jwd2348i-s was not output at compilation time and object was created.

[This version]

Diagnostic message jwd2348i-s is output at compilation time.

b. Influence

Diagnostic message jwd2348i-s is output at compilation time and object is not created.

c. Coping

Modify pointer target with simply CONTIGUOUS.

G.4.4 Non pure final subroutine referred from pure procedure

a. Changes

It corresponds to the change in the Fortran 2003 standard.

Diagnostic message jwd2597i-s is output at compilation time under the following conditions:

- There is a non pure final subroutine referred from pure procedure. And,
- The final subroutine is called by local variable.

[Previous version]

Diagnostic message jwd2597i-s was not output at compilation time and object was created.

[This version]

Diagnostic message jwd2597i-s is output at compilation time.

b. Influence

Diagnostic message jwd2597i-s is output at compilation time and object is not created.

c. Coping

Fix final subroutine to pure.

G.4.5 Array argument that is not CONTIGUOUS in intrinsic module function C_LOC

a. Changes

It corresponds to the change in the Fortran 2008 standard.

Diagnostic message jwd2351i-s is output at compilation time under the following condition:

- An array without CONTIGUOUS appears in intrinsic module function C_LOC.

[Previous version]

Diagnostic message jwd2351i-s was not output at compilation time and object was created.

[This version]

Diagnostic message jwd2351i-s is output at compilation time.

b. Influence

Diagnostic message jwd2351i-s is output at compilation time and object is not created.

c. Coping

Fix to specify CONTIGUOUS array in intrinsic module function C_LOC.

G.4.6 Runtime message (jwe1007i-s) change

a. Changes

The runtime message (jwe1007i-s) is changed.

[Previous version]

The value specified for a type parameter must be equal to the value specified in the declaration of the allocatable variable.

[This version]

The value of type parameter of the allocate object must not differ from the value of corresponding type parameter of source expression or type specification.

b. Influence

The changed message is output.

When errmsg-variable is specified for ERRMSG, and the error of jwe1007i-s occurs in the ALLOCATE statement, the message content of the variable is changed.

c. Coping

It is not necessary to action.

G.5 Migrating to V1.0L20(Generation Number:07)

G.5.1 Error message output when specifying parallelization options are omitted at linking

1. -Kparallel option

   a. Changes

   The linking process when -Kparallel option is not specified for the automatically parallelized object at linking is changed.

   [Previous version]

   An executable file using serial processing was created without the error message.

[This version]

The following error message is output, and the linking process is suspended.

```
command: fatal: -Kparallel option is not specified at linking of object files to which -
Kparallel applied at compiling. The linking process is suspended.
```

command: { frt | frtpx }

b. Influence

An executable file is not created if -Kparallel option is not specified for the automatically parallelized object at linking.

c. Coping

Specify -Kparallel option at linking if that error message is output.

d. Note

The target files for checking whether specifying -Kparallel option is omitted are object files (*.o) and libraries (*.a, *.so) input for compile command. Libraries specified by -l option are not target.

2. -Kopenmp option

a. Changes

The linking process when -Kopenmp option is not specified for the object parallelized by OpenMP at linking is changed.

[Previous version]

An executable file using serial processing was created without the error message.

[This version]

The following error message is output, and the linking process is suspended.

```
command: fatal: -Kopenmp option is not specified at linking of object files to which -
Kopenmp applied at compiling. The linking process is suspended.
```

command: { frt | frtpx }

b. Influence

An executable file is not created if -Kopenmp option is not specified for the parallelized object by OpenMP at linking.

c. Coping

Specify -Kopenmp option at linking if that error message is output.

d. Note

The target files for checking whether specifying -Kopenmp option is omitted are object files (*.o) and libraries (*.a, *.so) input for compile command. Libraries specified by -l option are not target.

G.6 Migrating to V1.0L20(Generation Number:06)

G.6.1 Message output for unrecognized compiler option

a. Changes

Warning message is output when unrecognized compiler option is specified.

[Previous version]

No warning message was output when unrecognized compiler option was specified.

[This version]

Warning message is output when unrecognized compiler option is specified.

b. Influence

Warning message is output when unrecognized compiler option is specified.

c. Coping

Specify recognizable compiler option. Use the -W option to specify linker option as follows:

```
-Wl,linker-option (*1)
```

(*1) "l" is lowercase L.

G.7 Migrating to V1.0L20

G.7.1 Runtime message (jwe0220i-e) change

a. Changes

The runtime message (jwe0220i-e) is changed.

[Previous version]

In ATAN2(x1,x2), x1.eq.0.0 .and. x2.eq.0.0.

[This version]

In ATAN(x1,x2) or ATAN2(x1,x2), x1.eq.0.0 .and. x2.eq.0.0.

b. Influence

The changed message is output.

c. Coping

It is not necessary to action.

G.7.2 Changes to the specification when SOURCE= specifier appears and an allocatable component is "allocated" in ALLOCATE statement

a. Changes

It corresponds to the change in the Fortran standard.

If SOURCE= specifier appears and the allocation status of allocatable component is "allocated" in ALLOCATE statement, the allocation status for corresponding allocatable component is changed "allocated" from "unallocated".

[Previous version]

Any allocatable components had an allocation status of "unallocated" in ALLOCATE statement.

[This version]

If SOURCE= specifier appears and the allocation status of allocatable component is "allocated" in ALLOCATE statement, the allocation status for corresponding allocatable component is "allocated".

Any allocatable components have an allocation status of "unallocated" unless the conditions.

b. Influence

If SOURCE= specifier appears and the allocation status of allocatable component is" allocated" in ALLOCATE statement, the allocation status for corresponding allocatable component is "allocated".

Therefore, if the component is allocated after the execution, the runtime message jwe1001i-s is output at execution time.

It is because of becoming a reallocation status.

c. Coping

If SOURCE= specifier appears and the allocation status of allocatable component is" allocated" in ALLOCATE statement, the allocation status for corresponding allocatable component is "allocated". Therefore, if the component is allocated after the execution, modify either of the following program correction.

- Change the allocation status for corresponding allocatable component to "unallocated" from "allocated" before the execution of SOURCE= specifier appeared allocatable statement, or
- Change the allocation status for corresponding allocatable component to "unallocated" from "allocated" after the execution of SOURCE= specifier appeared allocatable statement.

G.7.3 Support Fortran 2008 intrinsic procedures

a. Changes

The following procedure names without the EXTERNAL attribute become intrinsic procedure.

```
ACOSH,ASINH,ATANH,BGE,BGT,BLE,BLT,DSHIFTL,DSHIFTR,HYPOT,LEADZ,MASKL,MASKR,MERGE_BITS,POPCNT,POP
PAR,SHIFTA,SHIFTL,SHIFTR,STORAGE_SIZE,TRAILZ
```

[Previous version]

The procedure names became user defined external procedure.

[This version]

The procedure names become intrinsic procedure.

b. Influence

Diagnostic message is output at compilation time when characteristics (function, subroutine, type of function result, number of argument or type of argument) of procedure are different.

Program ends terminate abnormally or result is an incorrect at execution time when all of the characteristics for the procedure are corresponding.

c. Coping

Specify an EXTERNAL attribute in source program.

G.7.4 Change when specifying the -O option before specifying the -g option

a. Changes

The -O option is effective when specifying the option which makes the -O option effective before specifying the -g option.

[Previous version]

The -g option set the -O0 option.

To make the -O option effect, the -O option had to be specified after the -g option.

Example 1: -g and -O0 options were effective

```
$ frtpx -O3 -g a.f90
```

Example 2: -g and -O3 options were effective

```
$ frtpx -g -O3 a.f90
```

[This version]

When the -g option is specified, and the option which makes the -O1 or higher option effective is not specified, the -O0 option is effective.

Example 1: -g and -O0 options are effective

```
$ frtpx -g a.f90
```

Example 2: -g and -O3 options are effective

```
$ frtpx -O3 -g a.f90
```

```
$ frtpx -g -O3 a.f90
```

b. Influence

When -O1 or higher option is specified before the -g option is specified, the optimization becomes effective, compilation time may increase, and the size of object program may increase.

c. Coping

Specify the -O0 option to suppress the optimization.

Appendix H Compatibility Information (FX100 System)

This appendix provides compatibility information as notes on migrating.

H.1 Migrating to V2.0L20(Generation Number:03)

H.1.1 Error check in OpenMP standard at compilation time

Refer to "G.1.1 Error check in OpenMP standard at compilation time" about this note.

H.1.2 Error check in Fortran 2003 standard at compilation time

Refer to "G.1.2 Error check in Fortran 2003 standard at compilation time" about this note.

H.1.3 Enhancement intrinsic procedures and intrinsic module

a. Changes

1. The following procedure names without EXTERNAL attribute become intrinsic procedures:
   - ATOMIC_DEFINE
   - ATOMIC_REF
   - CO_MAX
   - CO_MIN
   - CO_SUM
   - IMAGE_INDEX
   - LCOBOUND
   - NUM_IMAGES
   - THIS_IMAGE
   - UCOBOUND
2. When intrinsic module ISO_FORTRAN_ENV is referred, the following names become named constants defined by ISO_FORTRAN_ENV:
   - ATOMIC_INT_KIND
   - ATOMIC_LOGICAL_KIND
   - STAT_STOPPED_IMAGE
   - STAT_LOCKED
   - STAT_LOCKED_OTHER_IMAGE
   - STAT_UNLOCKED
3. When intrinsic module ISO_FORTRAN_ENV is referred, a LOCK_TYPE become derived type name defined by ISO_FORTRAN_ENV.

[Previous version]

1. The following procedure names without EXTERNAL attribute were not intrinsic procedures:
   - ATOMIC_DEFINE
   - ATOMIC_REF
   - CO_MAX

- CO_MIN
- CO_SUM
- IMAGE_INDEX
- LCOBOUND
- NUM_IMAGES
- THIS_IMAGE
- UCOBOUND

2. The following names were not defined by the intrinsic module ISO_FORTRAN_ENV:
   - ATOMIC_INT_KIND
   - ATOMIC_LOGICAL_KIND
   - STAT_STOPPED_IMAGE
   - STAT_LOCKED
   - STAT_LOCKED_OTHER_IMAGE
   - STAT_UNLOCKED
   - LOCK_TYPE

[This version]

1. The following procedure names without EXTERNAL attribute become intrinsic procedures.
   - ATOMIC_DEFINE
   - ATOMIC_REF
   - CO_MAX
   - CO_MIN
   - CO_SUM
   - IMAGE_INDEX
   - LCOBOUND
   - NUM_IMAGES
   - THIS_IMAGE
   - UCOBOUND
2. When intrinsic module ISO_FORTRAN_ENV is referred, the following names become named constants defined by ISO_FORTRAN_ENV.
   - ATOMIC_INT_KIND
   - ATOMIC_LOGICAL_KIND
   - STAT_STOPPED_IMAGE
   - STAT_LOCKED
   - STAT_LOCKED_OTHER_IMAGE
   - STAT_UNLOCKED
3. When intrinsic module ISO_FORTRAN_ENV is referred, a LOCK_TYPE become derived type name defined by ISO_FORTRAN_ENV.

b. Influence

1. Diagnostic message is output at compilation time when characteristics of procedures (function, subroutine, type of function result, number of argument, or type of argument) are different.
   Program is abnormally ended or result becomes incorrect at execution time when all of the characteristics of the procedures are same.
2. The diagnostic message is output at compilation time when the same user-defined name exists as named constants of the additional, and referring to intrinsic module ISO_FORTRAN_ENV.

c. Coping

1. Correct the source program to specify the EXTERNAL attribute for user function which has same name as above functions.
2. Use ONLY option for USE statement of ISO_FORTRAN_ENV to make above named constants not to be use associated.

H.1.4 Change of the compilation message when the compiler option -K{FLTLD|NOFLTLD} is specified

a. Changes

Compilation message is changed when the compiler option -K{FLTLD|NOFLTLD} is specified.

[Previous version]

The following warning message was output when the compiler option -K{FLTLD|NOFLTLD} was specified.

```
frtpx: warning: Invalid suboption [NO]FLTLD specified for -K.
```

[This version]

The following message is output when the compiler option -KHPC_ACE2 is valid and the compiler option -K{FLTLD|NOFLTLD} is specified.

```
frtpx: -K[NO]FLTLD is invalidated because -KHPC_ACE is not specified.
```

b. Influence

The message of [This version] is output when the compiler option -KHPC_ACE2 is valid and the compiler option -K{FLTLD|NOFLTLD} is specified.

c. Coping

It is not necessary to action.

H.2 Migrating to V2.0L10(Generation Number:02)

H.2.1 The function for canceling the program compilation that is forecasted to take a long time

Refer to "G.2.1 The function for canceling the program compilation that is forecasted to take a long time" about this note.

H.2.2 The compiler option -K{ordered_omp_reduction|noordered_omp_reduction} is changed to -K{openmp_ordered_reduction|openmp_noordered_reduction}

Refer to "G.2.2 The compiler option -K{ordered_omp_reduction|noordered_omp_reduction} is changed to -K{openmp_ordered_reduction|openmp_noordered_reduction}" about this note.

Index

[G]

[H]

[I]

[J]

[K]

[L]

[M]