Fortran User's Guide

Fortran 77 4.2
Fortran 90 1.2

THE NETWORK IS THE COMPUTER™

SunSoft, Inc.
A Sun Microsystems, Inc. Business
2550 Garcia Avenue
Mountain View, CA 94043 USA
415 960-1300 fax 415 969-9131

Part No.: 802-5663-10
Revision A, December 1996

Contents

Tables

Preface

This guide describes the compile-time environment and command-line options for Sun™ Fortran compilers `f77` (Fortran 77 Release 4.2) and `f90` (Fortran 90 Release 1.2). Run-time error messages and new features of the compilers are listed in appendixes. Additional sources of documentation are listed at the end of this Preface.

See also the companion *Fortran Programmer's Guide* for a discussion of efficient program development and performance methods.

Note – In this guide, "`f77`/`f90`" and "Fortran" to indicate information that is common to *both* the Sun Fortran 77 and Fortran 90 compilers.

Audience

This guide is intended for scientists, engineers, and programmers who have a working knowledge of the Fortran language and wish to learn how to use the Sun Fortran compilers effectively. Familiarity with the Solaris™ operating system or UNIX® in general is also assumed.

Organization of This Guide

This user guide is organized into the following chapters and appendixes:

- **Chapter 1,** "**Introduction**" briefly describes the features of the compilers.
- **Chapter 2,** "**Using Sun Fortran Compilers**" discusses the compiler environments.
- **Chapter 3,** "**Sun Fortran Compiler Options**" gives detailed descriptions of all the command-line options.
- **Appendix A,** "**Runtime Error Messages**" lists error messages issued by the Fortran runtime library and the operating system.
- **Appendix B,** "**Feature Release History**" notes new features of the compilers and behavior changes in recent releases.
- **Appendix C,** "**Fortran 90 Features and Differences**" describes the differences between the Sun `f90` compiler and the Fortran 90 standard.
- **Appendix D,** "**Localization Support**" discusses how the Fortran compilers implement various internationalization requirements.

Multi-Platform Release

The Sun Fortran documentation covers the release of the Fortran compilers on a number of operating systems and hardware platforms:

Fortran 77 4.2 is released for:

- Solaris 2.x operating system on:
  - architectures based on the SPARC® microprocessor
  - x86-based architectures, where x86 refers to the Intel® implementation of one of the following: Intel 80386™, Intel 80486™, Pentium™, or the equivalent
  - PowerPC™ architecture compliant with the Common Hardware Reference Platform (CHRP) and the PowerPC Reference Platform (PReP) specifications

Fortran 90 1.2 is released for:

- Solaris 2.x operating system on SPARC® architectures only.

The Fortran documentation describes the Sun compilers on all the above operating systems and platforms. Anything unique to one or more platforms is identified as "*(SPARC)*", "*(Intel)*", and / or "*(PowerPC)*".

Fortran Compiler Documentation

The following documentation is included with Sun Fortran:

- Manuals
  - Paper ("hard copy")
  - *O*nline versions viewable with Solaris AnswerBook Navigator
  - Online versions in HTML 3.
- Online man pages
- `f77` and `f90` command-line help (`-help`)
- Online `READMEs`

Manuals

The following Sun manuals and guides are provided on-line and hard copy, except as indicated.

- *Fortran 77 Language Reference.* Complete programmer's reference to the Sun Fortran 77 language.
- *Fortran 90 Handbook*. Detailed reference to the ANSI Standard Fortran 90 language. (On-line only).
- *Fortran Library Reference.* Complete programmer's reference to the Sun Fortran 77 and Fortran 90 runtime library.
- *Fortran Programmer's Guide.* Further details about using Sun Fortran compilers effectively, including discussions of I/O, libraries, performance, and parallelization.
- *Workshop: Beyond the Basics.* Guide to program debugging with Sun compilers and debug tools.
- *Incremental Link Editor.* How to use the Solaris runtime incremental link editor (`ild`) effectively.
- *Numerical Computation Guide.* Describes the details of floating-point arithmetic used by Sun compilers on various platforms.
- *WorkShop Installation and Licensing Guide.*. Instructions on software installation.

Much of Sun's compiler and related documentation is available online in either *AnswerBook* or HTML formats, or both. For details on installation, see the Sun Developer Products installation guides. See also the `answerbook`(1) man page and the `f77` or `f90` README.

The following documents are also relevant:

- IEEE and ISO POSIX.1 Standard.
- *American National Standard Programming Language FORTRAN*, ANSI X3.9-1978, April 1978, American National Standards Institute, Inc.
- *American National Standard Programming Language—Fortran—Extended*, ANSI X3.198-1992, 1992, American National Standards Institute, Inc.

Man Pages

On-line manual (`man`) pages provide immediate documentation about a command, function, subroutine, or collection of such things. You can display a man page by running the command:

```
demo% man topic
```

Throughout the Fortran documentation, man page references appear with the topic name and man section number: `f77`(1) is accessed with `man f77`.

For details and useful `man` options, see the `man` command's own man page:

```
demo% man man
Reformatting page.  Wait... done

man(1)                    User Commands                    man(1)

NAME
     man - find and display reference manual pages

SYNOPSIS
     man [ - ] [ -adFlrt ] [ -M path ] [ -T macro-package ]
          [-s section ] name ...
     man [ -M path ] -k keyword ...
     man [ -M path ] -f filename ...

AVAILABILITY
     SUNWdoc

DESCRIPTION
     man displays information from  the  reference  manuals.   It
....etc
```

The following `man` pages are of interest to Fortran user.

`f77`(1) and `f90`(1)	The Fortran compilers command-line options
`asa`(1)	Fortran carriage-control print output post-processor
`dbx`(1)	Command-line interactive debugger
`fpp`(1)	Fortran source code pre-processor
`fsplit`(1)	Pre-processor splits Fortran 77 routines into single files
`ieee_flags`(3M)	Examine, set, or clear floating-point exception bits
`ieee_handler`(3M)	Handle floating-point exceptions
`matherr`(3M)	Math library error handling routine
`ild`(1)	Incremental link editor for object files
`ld`(1)	Link editor for object files

Command-Line Help

You can view very brief descriptions of the `f77` and `f90` command line options by invoking the compiler's `-help` option as shown below:

```
%f77 -help  -or-
 f90 -help
-ansi:         Report non-ANSI extensions.
-arg=local:    Pass by value result
-autopar:      Generate parallelized code
-BX:           Specify dynamic or static binding
-c:            Compile only - produce .o files, suppress linking
-C:            Enable runtime subscript range checking
-cg89:         Generate code for generic SPARC V7 architecture
-cg92:         Generate code for SPARC V8 architecture
-copyargs:     Allow assignment to constant arguments
-dalign:       Assume double-type data is double aligned
-dbl:          Double default size for INTEGER, REAL, etc.
-depend:       Analyze loops for data dependencies
-dn:           Specify static binding
...etc.
```

READMEs

The `READMEs` directory contains files that describe new features, software incompatibilities, bugs, and information that was discovered after the manuals were printed. The location of this directory depends on where your software was installed:

	Standard Installation	**Nonstandard Installation to** */my/dir/*
Solaris 2.x	`/opt/SUNWspro/READMEs/`	*/my/dir*`/SUNWspro/READMEs/`

The files and their contents are:

File	Contents
`feedback`	email template file for sending feedback comments to Sun
`fortran-77` `fortran-90`	`f77/f90` bugs, new features. behavior changes, documentation errata
`math_libraries`	Describes optimized and specialized math libraries available
`profiling_tools`	Information on using the performance profiling tools
`runtime_libraries`	Lists libraries and executables that can be redistributed under the End User License

Conventions in Text

This guide uses the following conventions to display information.

- Code listings and examples appear in boxes:

```
   WRITE( *, * ) 'Hello world'
```

- The `plain Courier font` shows prompts, coding, and generally anything that is computer output.
- In dialogs, the **`boldface Courier font`** shows text you type in:

```
demo% echo hello
hello
demo%
```

- *Italics* indicate general arguments or parameters that you replace with the appropriate input. Italics also indicate emphasis.
- The small clear triangle Δ shows a blank space where that is significant:

```
ΔΔ36.001
```

- Fortran 77 examples appear in tab format; Fortran 90 examples appear in free format. Examples common to both Fortran 77 and Fortran 90 use tab format except where indicated.
- Uppercase characters are generally used to show Fortran keywords and intrinsics (`PRINT`), and lowercase or mixed case for variables (`TbarX`).
- The Sun Fortran compilers are referred to by their command names, either `f77` or `f90`. "`f77/f90`" indicates information that is common to both the Fortran 77 and Fortran 90 compilers.

Introduction 1

This chapter describes the operating environment and features of Sun's Fortran 77 and Fortran 90 compilers, `f77` and `f90`.

Operating Environments

The Fortran compilers integrate with other Sun™ development tools, such as the Sun WorkShop™, C, C++, and Pascal. The compiler and its runtime library are part of the Sun Performance WorkShop™, and can be used to develop threaded applications on multiple processor Solaris 2.x systems.

Release 4.2 of `f77` is available under Solaris 2.x operating environment on SPARC, Intel x86, and PowerPC platforms. Release 1.2 of `f90` is only available under Solaris 2.x on SPARC systems.

Note – Features in this guide identified as being unique to a particular system environment or hardware platform are so indicated. However, most aspects of the compilers on these platforms are the same, including functionality, behavior, and features. The multiprocessor features are available as part of the Sun WorkShop on SPARC with Solaris 2.x, and requires a WorkShop license. See the Fortran README files for details.

1

Standards Conformance

Sun Fortran 77 and Fortran 90 compilers:

- Conform to the ANSI X3.9-1978 (`f77`) and ISO/IEC 1539-1:1991 (`f90`) Fortran standards. NIST (formerly GSA and NBS) validates it at appropriate intervals.
- Conform to the standards FIPS 69-1, BS 6832, and MIL-STD-1753.
- Provide an IEEE standard 754-1985 floating-point package.
- Provide support on SPARC® systems for optimization exploiting features of SPARC V8, including the SuperSPARC™ implementation. These features are defined in the *SPARC Architecture Manual: Version 8*.

Licensing

The Fortran compilers use network licensing, as described in the manual *WorkShop Installation and Licensing Guide*

If you invoke the compiler, and a license is available, the compiler starts. If no license is available, your request for a license is put on a queue, and your compile continues when a license becomes available. A single license can be used for any number of simultaneous compiles by a single user on a single machine.

To run Fortran and the various utilities, several licenses may be required, depending on the package you have purchased.

Fortran parallelization features require a Sun Workshop™ license. See the Fortran README files for details.

Features of Sun Compilers

The Sun Fortran compilers offer the following extended features:

- Global program checking for consistency in definition and use of arguments, commons, parameters, and so on, across routines.
- Performance tuning and code optimization capabilities, including global and peephole scalar optimizations, and optimizations based upon run-time performance timings.

- Automatic and explicit loop parallelization is integrated tightly with the compiled code optimizer.
- For `f77`, many VAX® / VMS® Fortran 5.0 extensions, including:
  - `NAMELIST`
  - `DO WHILE`
  - Structures, records, unions, maps
  - Variable format expressions
- Fortran 77 programs may utilize many of the following VMS extensions, to make them portable over both SPARC and VAX systems:
  - Recursion
  - Pointers
  - Double-precision complex
  - Quadruple-precision real (*SPARC and PowerPC only*)
  - Quadruple-precision complex (*SPARC and PowerPC only*)
- Interoperability between routines written in C, C++, or Pascal and Fortran programs, since these languages have common calling conventions.

Other Fortran Utilities

The following utilities provide assistance in the development of software programs in Fortran:

- **`asa`**

  This utility is a Fortran output filter for printing files that have Fortran carriage-control characters in column one. Use `asa` to transform files formatted with Fortran carriage-control conventions into files formatted according to UNIX line-printer conventions. See `asa`(1).

- **`fsplit`**

  This utility splits one Fortran file of several routines into several files, each with one routine per file. Use `fsplit` on Fortran 77 source files. See `fsplit`(1)

- **`gprof`**

  Profile program run-time performance by procedure. (This utility is available if you do a developer install, rather than an end user install of Solaris 2.x; it is also included if you install the `SUNWbtool` package.)

- **`tcov`**

  Profile program run-time performance by statement.

- **`sbrowser`**

  The SourceBrowser is a Fortran 77 source code and call graph browser that finds occurrences of any symbol in all source files, including header files. It is included with `dbx`.

- **`f90browse`**

  An interactive source code and call graph browser for Fortran 90.

Debugging Utilities

The following debugging utilities are available:

- **`error`**

  A utility to merge compiler error messages with the Fortran source file. (This utility is included if you do a developer install, rather than an end user install of Solaris 2.x; it is also included if you install the `SUNWbtool` package.)

- **`-Xlist`**

  An option to check across routines for consistency of arguments, commons, and so on.

- **Sun WorkShop**

  The WorkShop provides a number of debugging utilities such as dbx and a data visualizer, presented within an easy-to-use graphical framework.

Sun Performance Library™

The Sun Performance Library is a library of optimized subroutines and functions for computational linear algebra and Fourier transforms. It is based on the standard libraries `BLAS1`, `BLAS2`, `BLAS3`, `LINPACK`, `LAPACK`, `FFTPACK`, and `VFFTPACK`.

Each subprogram in the Sun Performance Library performs the same operation and has the same interface as the standard library versions, but is generally much faster and possibly more accurate.

See the `performance_library` README file for details.

1

Using Sun Fortran Compilers 2

This chapter describes how to use the Fortran 77 and Fortran 90 compilers.

The principal use of any compiler is to transform a program written in a procedural language like Fortran into a data file that is executable by the target computer hardware. As part of its job, the compiler may also automatically invoke a system linker to generate the executable file.

The Sun Fortran 77 and Fortran 90 compilers can also be used to:

- Generate a parallelized executable file for multiple processors (`-parallel`)
- Analyze and report on program consistency across source files and subroutines (`-Xlist`)
- Transform source files into:
  - Relocatable binary (`.o`) files, to be linked later into an executable file or static library (`.a`) file
  - A dynamic shared library (`.so`) file (`-G`)
- Link or relink files into an executable file
- Compile an executable file with runtime debugging enabled (`-g`)
- Compile an executable file with runtime statement or procedure level profiling (`-pg`)
- Compile an executable file with runtime parallelized loop profiling (`-Zlp`)
- Check source code for ANSI standards conformance (`-ansi`)

2

A Quick Start

This section provides a quick overview of how to use the Sun Fortran compilers to compile and run Fortran programs. A full reference to command-line options appears in the next chapter.

Note – The command line examples in this chapter show `f77` usages. Except where noted, equivalent usages of `f90` are similarly valid; however, the printed output may be slightly different.

The very basic steps to running a Fortran application involve using an editor to create a Fortran source file with a `.f`, `.for`, `.f90`, `.F`, or `.F90` filename suffix; invoking the compiler to produce an executable; and finally, launching the program into execution by typing the name of the file:

Example: This program displays a message on the screen:

```
demo% cat greetings.f
    PROGRAM GREETINGS
    PRINT *, 'Real programmers write Fortran!'
    END
demo% f77 greetings.f
greetings.f:
  MAIN greetings:
demo% a.out
 Real programmers write Fortran!
demo%
```

In the example, `f77` compiles source file `greetings.f` and compiles the executable program onto the file, `a.out`, by default. To launch our program, the name of the executable file, `a.out`, is typed at the command prompt.

Traditionally, UNIX compilers write executable output to the default file called `a.out`. It can be awkward to have each compilation write to the same file. Moreover, if such a file already exists, it will be overwritten by the next run of the compiler. Instead, use the `-o` compiler option to explicitly specify the name of the executable output file:

```
demo% f77 –o greetings greetings.f
greetings.f:
MAIN greetings:
demo%
```

In the preceeding example, to `-o` option tells the compiler to write the executable code to the file `greetings`. (By convention, executable files usually are given the same name as the main source file, but without an extension.)

Alternatively, the default `a.out` file could be renamed via the `mv` command after each compilation. Either way, run the program by typing the name of the executable file:

```
demo% greetings
 Real programmers write Fortran!
demo%
```

The next sections of this chapter discuss the conventions used by the `f77` and `f90` commands, compiler source line directives, and other issues concerning the use of these compilers. The next chapter describes the command-line syntax and all the options in detail.

2

Invoking the Compiler

The syntax of a *simple* compiler command is as follows:

`f77` [*options*] *sfn* **...**	*invokes the Fortran 77 compiler*
`f90` [*options*] *sfn* **...**	*invokes the Fortran 90 compiler*

Here *sfn* is a Fortran source file name that ends in `.f`, `.F`, `.f90`, `.F90`, or `.for`; *options* is one or more of the compiler options. (Files with names ending in a `.f90` or `.F90` extension are "free-format" Fortran 90 source files recognized only by the `f90` compiler.)

In the example below, `f90` is used to compile two source files to produce an executable file named `growth` with runtime debugging enabled:

```
demo% f90 -g -o growth growth.f fft.f90
```

Compile-Link Sequence

In the previous example, the compiler will automatically generate the loader object files, `growth.o` and `fft.o`, and then invoke the system linker to create the executable program on the file `growth`.

After compilation, the object files, `growth.o` and `fft.o`, will remain. This convention permits easy relinking and recompilation of files.

If the compilation fails, you will receive a message for each error. No `.o` files are generated for those source files with errors, and no executable program is written.

Command-Line File Name Conventions

The suffix extension attached to file names appearing on the command-line determine how the compiler will process the file. File names with a suffix extension other than one of those listed below, or without an extension, are passed to the linker.

Table 2-1 File Name Suffixes That Fortran Compilers Recognize

Suffix	Language	Action
`.f`	Fortran 77 or Fortran 90 fixed-format	Compile Fortran source files, put object files in current directory; default name of object file is that of the source but with `.o` suffix.
`.f90`	Fortran 90 free-format	Same action as `.f` (*`f90` only*)
`.for`	Fortran 77	Same action as `.f`.
`.F`	Fortran 77	Apply the Fortran (or C) preprocessor to the Fortran 77 source file before Fortran compiles it.
`.F90`	Fortran 90	Apply the Fortran (or C) preprocessor to the Fortran 90 free-format source file before Fortran compiles it.
`.r`	Ratfor	Process Ratfor source files before compiling.
`.s`	Assembler	Assemble source files with the assembler.
`.S`	Assembler	Apply the C preprocessor to the assembler source file before assembling it.
`.il`	Inline expansion	Process template files for inline expansion. The compiler will use templates to expand inline calls to selected routines. (Template files are special assembler files; see the `inline`(1) man page.)
`.o`	Object files	Pass object files through to the linker.
`.a`	Libraries	Pass names of libraries to the linker.

Fortran 90 free-format is described in Appendix C of this manual.

2

Source Files

The Fortran compilers will accept multiple source files on the command line. A set of source files compiled together by a single compiler command are often referred to as a *compilation unit*. A single source file may contain any number of procedures (main program, subroutine, function, block data, module, and so on). There are advantages for organizing an application with one procedure per file, as there are for gathering procedures that work together into a single file. Some of these are described in the Sun *Fortran Programmer's Guide.*

Source File Preprocessors

Both `f77` and `f90` support two source file preprocessors, `fpp` and `cpp`. Either can be invoked by the compiler to expand source code "macros" and symbolic definitions prior to compilation. The compilers will use `fpp` by default; the `-xpp=cpp` option changes the default from `fpp` to `cpp`. (See also the discussion of the `-D`*name* option).

`fpp` is a language preprocessor specifically for Fortran syntax. See the `fpp`(1) man page. It is invoked by default by `f77` on files with a `.F` extension, and by `f90` on files with a `.F` or `.F90` extension.

The `cpp` program is the C language preprocessor**.** See `cpp`(1). Use of `fpp` over `cpp` is recommended.

Separate Compiling and Linking

You can compile and link in separate steps. The `-c` option compiles source files and generates `.o` object files, but does not create an executable. Without the `-c` option the compiler will invoke the linker. By splitting the compile and link steps in this manner, a complete recompilation is not needed just to fix one file, as shown in the following example:

Compile one file and link with others in separate steps:

```
demo% f77  -c file1.f                          (Make new object file)
demo% f77  -o prgrm file1.o file2.o file3.o    (Make executable file)
```

Be sure that the link step lists *all* the object files needed to make the complete program. If any object files are missing from this step, the link will fail with undefined external reference errors (missing routines).

Consistent Compiling and Linking

If you do compile and link in separate steps, consistent compiling and linking is critical when using certain compiler options:

`–a`, `–autopar`, `-B`*x*, `–cg92`, `-dy`, `-dn`, `–dalign`, `-dbl`, `-dbl_align_all`, `-explicitpar`, `–f`, `–fast`, `-misalign`, `–p`, `-parallel`, `–pg`, `-r8`, `-xarch=`*a*, `-xcache=`*c*, `-xchip=`*c*, `xprofile=`*p*, `-xtarget=`*t*, `-Zlp`, `-Ztha`

Compile `sbr.f` with `-dbl` and `smain.f` without it:

```
demo% f77 -c -dbl sbr.f
demo% f77 -c smain.f
demo% f77 -dbl sbr.o smain.o    {pass –dbl to the linker}
```

If you *compile* any subprogram with any of these options, be sure to *link* with the same options as well.

Linking Mixed Fortran 90 and Fortran 77 Compilations

As a general rule, if *any* of the object files that make up a program were compiled with `f90`, then the final link step must be done with `f90`. Use `f77` to produce the executable file only if *none* of the `.o` object files were compiled with `f90`.

Unrecognized Command-Line Arguments

Any arguments on the command-line that the compiler does not recognize are interpreted as being possibly linker options, object program file names, or library names.

The basic distinctions are:

- Unrecognized *options* (with a `-`) generate warnings.

- Unrecognized *non-options* (no `-`) generate no warnings. However, they are passed to the linker and if the linker does not recognize them, they generate linker error messages.

For example:

```
demo% f77 -bit move.f            <- -bit is not a recognized f77 option
f77: Warning: Option -bit passed to ld, if ld is invoked, ignored
otherwise
move.f:
 MAIN move:
demo% f77 fast move.f            <- The user meant to type -fast
move.f:
 MAIN move:
ld: fatal: file fast: cannot open file; errno=2
ld: fatal: File processing errors.  No output written to a.out
```

Note that in the first example, `-bit` is not recognized by `f77` and the option is passed on to the linker (`ld`), who tries to interpret it. Because single letter `ld` options may be strung together, the linker sees `-bit` as `-b -i -t`, which are all legitimate `ld` options! This may (or may not) be what the user expects, or intended.

In the second example, the user intended to type the `f77/f90` option `-fast` but neglected the leading dash. The compiler again passes the argument to the linker which, in turn, interprets it as a file name.

These examples indicate that extreme care should be observed when composing compiler command lines!

Modules (Fortran 90)

`f90` automatically creates module files for each `MODULE` declaration encountered in the source files, and searches for modules referenced by a `USE` statement. All the modules appearing in a source file are compiled into a single file with the primary name of the source file and `.M` suffix. For example, the modules on `mysrc.f90` would be compiled to `mysrc.M` by `f90`.

The compiler searches the current directory for module files referenced in `USE` statements. More directories can be added to the search path with the `f90 -M` command-line option. However, `.M` files cannot be specified directly on the command line.

Directives

Use a source code *directive*, a form of Fortran comment, to pass specific information to the compiler regarding special optimization or parallelization choices. Compiler directives are also called *pragmas.*

Only `f77` directives are discussed in this section. For `f90` directives, *see Appendix C.*

Note – Directives are not part of any Fortran standard.

General Directives (`f77`)

The various forms of an `f77` general directive are:

```
C$PRAGMA keyword
C$PRAGMA keyword ( a [ , a ] … )[ , keyword ( a[ , a ]… )],…
C$PRAGMA SUN keyword
```

The variable *keyword* identifies the specific directive; the *a*'s are arguments.

The general directives recognized by `f77` are:

- `C(…)` — The listed arguments are external functions written in C.
- `UNROLL=`*n* — Requests the optimizer to attempt loop unrolling to depth *n.*
- `WEAK (`*name1*`[=`*name2*`])` — Define weak symbol bindings.

A general directive has the following syntax:

- In column one, any of the comment-indicator characters `c`, `C`, `!`, or `*`
- The next seven characters are `$PRAGMA`, no blanks, any uppercase or lowercase
- In *any* column, the `!` comment-indicator character

Observe the following restrictions:

- After the first eight characters, blanks are ignored, and uppercase and lowercase are equivalent, as in Fortran text.
- Because it is a comment, a directive cannot be continued, but you can have many C$PRAGMA lines, one after the other, as needed.
- If a comment satisfies the above syntax, it is expected to contain one or more directives recognized by the compiler; if it does not, a warning is issued.
- Only the C preprocessor, cpp, will expand macro symbol definitions within a directive line; the Fortran preprocessor, fpp, treats all directives as comments, and will not expand macros.

The C Directive (f77)

The C() directive specifies that its arguments are external functions written in the C language. It is equivalent to an EXTERNAL declaration except that unlike ordinary external names, the Fortran compiler will not append an underscore to these argument names. See the Sun *Fortran Programmer's Guide* for more details.

The C() directive for a particular function should appear before the first reference to that function in each subprogram that contains such a reference.

Example - compiling ABC and XYZ for C:

```
EXTERNAL ABC, XYZ
C$PRAGMA C(ABC, XYZ)
```

The UNROLL *Directive (*f77*)*

The UNROLL directive requires that you specify SUN after C$PRAGMA.

The C$PRAGMA SUN UNROLL=*n* directive instructs the compiler to unroll loops *n* times during its optimization pass.

n is a positive integer. The choices are:

- If *n*=1, this directive directs the optimizer *not* to unroll any loops.
- If *n*>1, this directive suggests to the optimizer that it unroll loops *n* times.

If any loops are actually unrolled, the executable file becomes larger. For further information, see the *Fortran Programmer's Guide* chapter on performance and optimization.

Example - unrolling loops two times:

```
C$PRAGMA SUN UNROLL=2
```

The WEAK *Directive (*f77*)*

The WEAK directive defines a symbol to have less precedence than an earlier definition of the same symbol. This pragma is used mainly in sources files for building libraries. The linker does not produce an error message if it is unable to resolve a weak symbol.

```
C$PRAGMA WEAK (name1 [=name2])
```

WEAK (*name1*) defines *name1* to be a weak symbol. The linker does not produce an error message if it does not find a definition for *name1*.

WEAK (*name1=name2*) defines *name1* to be a weak symbol and an alias for *name2*.

If your program calls but does not define *name1*, the linker uses the definition from the library. However, if your program defines its own version of *name1*, then the program's definition is used and the weak global definition of *name1* in the library is not used. If the program directly calls *name2*, the definition from library is used; a duplicate definition of *name2* causes and error. See the Solaris *Linker and Libraries Guide* for more information.

Parallelization Directives (f77)

Parallelization directives explicitly request the compiler attempt to parallelize the DO loop that follows the directive. The syntax differs from general directives. Parallelization directives are only recognized when compilation options -parallel or -explicitpar are used. (f90 parallelization directives are described in Appendix C; detailed information on Fortran parallelization can be found in the *Fortran Programmer's Guide*.)

Parallelization directives have the following syntax:

- The first character must be in column one.
- The first character can be any one of `c`, `C`, `*`, or `!`.
- The next four characters are `$PAR`, no blanks, either upper or lower case.
- Next follows the directive keyword and options, separated by blanks.

The explicit parallelization directive keywords are:

`DOALL, DOSERIAL,` and `DOSERIAL*`

Each parallelization directive has its own set of optional qualifiers that follow the keyword.

Example: Specifying a loop with a shared variable:

```
C$PAR DOALL SHARED(yvalue)
```

See the *Fortran Programmer's Guide* for details about parallelization.

Compiler Usage Notes

The next sections suggest a number of ways to use the Sun Fortran compilers efficiently. A complete compiler options reference follows in the next chapter.

Determining Floating-Point Hardware

Some compiler options are specific to particular hardware options. The utility command `fpversion` tells which floating-point hardware is installed:

```
demo% fpversion
 A SPARC-based CPU is available.
 Mbus module's clock rate appears to be approximately 60.1 MHz.
 Kernel says Mbus module's clock rate is 60.0 MHz.
 Kernel says main memory's clock rate is 50.0 MHz.

 Sun-4 floating-point controller version 0 found.
 A TI TMS390Z50 SuperSPARC chip (FAB 3.x or later) is available.
 A TI TMS390Z55 SuperCache appears to be present.
 FPU's frequency appears to be approximately 58.3 MHz.

 Use "-xtarget=sc2000 -xcache=16/32/4:2048/64/1" code option.
```

It may take a number of seconds before `fpversion` responds while it dynamically calculates apparent hardware clock rates of the CPU and FPU. (The values printed depend on the load on the system at the moment `fpversion` is called.)

See `fpversion`(1) and the *Numerical Computation Guide* for details.

Simplifying Options

You can simplify complicated compiler commands by defining special shell aliases or using the `$FFLAGS` environment variable.

Using Aliases (C Shell)

Example: Define an alias for a command with frequently used options:

```
demo% alias f77fx "f77 -silent -fast -Xlist"
```

Example: Using the alias `f77fx`:

```
demo% f77fx any.f
```

The command `f77fx` is now the same as:

```
f77 -silent -fast -Xlist any.f
```

Using Environment Variables

You can specify options by setting the `FFLAGS` or `OPTIONS` variables.

Either `FFLAGS` or `OPTIONS` can be used explicitly in the command line. When you are using `make` files implicit compilation rules, `FFLAGS` is used automatically by the `make` program.

Example: Set `FFLAGS`: (C Shell)

```
demo% setenv FFLAGS '-silent -fast -Xlist'
```

- Example: Use `FFLAGS` explicitly:

```
demo% f77 $FFLAGS any.f
```

When using `make`, if the `FFLAGS` variable is set as above and the makefile's compilation rules are *implicit*, that is, there is no *explicit* `f77/f90` compile line, then invoking `make` will result in a compilation equivalent to:

```
f77 -silent -fast -Xlist files...
```

`make` is a very powerful program development tool that can easily be used with all Sun compilers. See the `make`(1) man page and the *Program Development* chapter in the *Fortran Programmer's Guide.*

Memory Size

A compilation may need to use a lot of memory. This will depend on the optimization level chosen and the size and complexity of the files being compiled. On SPARC systems, if the optimizer runs out of memory, it tries to recover by retrying the current procedure at a lower level of optimization and resumes subsequent routines at the original level specified in the `-O`*n* option on the command line.

A workstation should have at least 24 megabytes of memory; 32 megabytes are recommended. Memory usage depends on the size of each procedure, the level of optimization, the limits set for virtual memory, the size of the disk swap file, and various other parameters.

Compiling a single source file containing many routines could cause the compiler to run out of memory or swap space.

If the compiler runs out of memory, try reducing the level of optimization, or split multiple-routine source files into files with one routine per file, using `fsplit`(1).

Swap Space Limits

The Solaris 2.x command, `swap -s`, displays available swap space. See `swap`(1M).

Example: Use the `swap` command:

```
demo% swap -s
total: 40236k bytes allocated + 7280k reserved = 47516k used,
1058708k available
```

To determine the actual real memory, use the following command:

```
demo% /usr/sbin/dmesg | grep mem
mem = 655360K (0x28000000)
avail mem = 602476544
```

Increasing Swap Space

Use `mkfile`(1M) and `swap` (1M) to increase the size of the swap space on a workstation. You must become superuser to do this. `mkfile` creates a file of a specific size, and `swap -a` adds the file to the system swap space:

```
demo# mkfile -v 90m /home/swapfile
/home/swapfile 94317840 bytes
demo# /usr/sbin/swap -a  /home/swapfile
```

Control of Virtual Memory

Compiling very large routines (thousands of lines of code in a single procedure) at `-O3` or higher, may require an unreasonable amount of memory. In such cases, performance of the system may degrade. You can control this by limiting the amount of virtual memory available to a single process.

To limit virtual memory:

- In a `sh` shell, use the `ulimit` command. See `sh`(1).

  Example: Limit virtual memory to 16 Mbytes:

```
demo$ ulimit -d 16000
```

- In a `csh` shell, use the `limit` command. See `csh`(1).

Example: Limit virtual memory to 16 Mbytes:

```
demo% limit datasize 16M
```

Each of these command lines causes the optimizer to try to recover at 16 Mbytes of data space.

This limit cannot be greater than the system's total available swap space and, in practice, must be small enough to permit normal use of the system while a large compilation is in progress.

Be sure that no compilation consumes more than half the space.

Example: With 32 Mbytes of swap space, use the following commands:

- In a sh shell:

```
demo$ ulimit -d 1600
```

- In a csh shell:

```
demo% limit datasize 16M
```

The best setting depends on the degree of optimization requested, and the amount of real memory and virtual memory available.

2

Sun Fortran Compiler Options 3

This chapter details the command-line options for the `f77` and `f90` compilers running under Solaris 2.x. Features unique to one or more platforms are identified as "*(SPARC)*", "*(Intel)*", and / or "*(PowerPC)*". Features unique to one or the other compiler are marked "*f77 only*" or "*f90 only*". See the description of this multi-platform release in the *Preface*.

Command Syntax

The general syntax of the compiler command line is:

`f77` [*options*] *list_of_files* [**`-l`***x*]
`f90` [*options*] *list_of_files* [**`-l`***x*]

Items in square brackets indicate optional parameters. The brackets are not part of the command. The *options* are a list of option keywords prefixed by dash (–). Some keyword options take the next item in the list as an argument. The *list_of_files* is a list of source, object, or library file names separated by blanks.

3

Options Syntax

Typical compiler option formats are:

Table 3-1 Options Syntax

Syntax Format	Example
–flag	`–g`
–flagvalue	`–Dnostep`
–flag=value	`–xunroll=4`
–flag value	`–align _comvarx`

The following typographical conventions are also used in this section of the manual when describing the individual options:

Table 3-2 Typographic Notations for Options

Notation	Meaning	Example: Text/Instance
[ ]	Square brackets contain arguments that are optional.	`–O[n]` `-O4`
{ }	Curly brackets contain a set of choices for a required option.	`-d{y|n}` `-dy`
|	The “pipe” or “bar” symbol separates arguments, only *one* of which may be chosen.	`–B{dynamic|static}` `-Bstatic`
:	The colon, like the comma, is sometimes used to separate arguments.	`–R`*dir*`[:`*dir*`]` `-R/local/libs:/U/a`
...	The ellipsis indicates omission in a series.	`–xinline=`*f1*`[,...`*fn*`]` `-xinline=alpha,dos`

Brackets, pipe, and ellipsis are *meta characters* used in the descriptions of the options and are not part of the options themselves.

Some general guidelines for options are:

- `–l`*x* is the option to link with library `lib`*x*`.a`. It is always safer to put `–l`*x* after the list of file names to insure the order libraries are searched.
- In general, processing of the compiler options is from left to right, allowing selective overriding of macro options (options that include other options).
  - The above rule does not apply to linker options.

- The `-I`, `-L`, and `-R` options accumulate, not override.

Source files, object files, and libraries are compiled and linked in the order in which they appear on the command line.

Options Summaries

In this section, the compiler options are grouped by function to provide an easy reference. The details will be found on the pages in the following sections, as indicated.

Commonly Used Options

The Sun Fortran compilers have many features that are selectable by optional command-line parameters. The short list below of commonly used options is a good place to start.

Table 3-3 Commonly Used Options

Action	Option	Details
Debug—global program checking across routines for consistency of arguments, commons, and so on.	`-Xlist`	*page 94*
Debug—produce additional symbol table information for the runtime debugger.	`-g`	*page 57*
Performance—invoke the optimizer to produce faster running programs.	`-O[`*n*`]`	*page 68*
Performance—produce reasonably efficient compilation and run times using a set of predetermined options	`-fast`	*page 50*
Bind as dynamic (or static) any library listed later in the command: `-Bdynamic`, `-Bstatic`	`-B`*x*	*page 42*
Library—Allow or disallow dynamic libraries for the entire executable: `-dy`, `-dn`	`-d`*x*	*page 47*
Compile only—Suppress linking; make a `.o` file for each source file.	`-c`	*page 43*
Output file—Name the executable output file *nm* instead of `a.out`.	`-o` *nm*	*page 70*
Profile—Profile by *procedure* for `gprof`.	`-pg`	*page 73*

Debugging Options

For the following debugging options, the most useful are listed first.

Table 3-4 Debugging Options

Action	Option	Details
Compile for use with the debugger.	`-g`	*page 57*
Global program checking (GPC)—arguments, commons, ...	`-Xlist`	*page 94*
Check for subscripts out of range.	`-C`	*page 42*
Undeclared variables—show a warning message.	`-u`	*page 82*
Version ID—show ID along with name of each compiler pass.	`-V`	*page 82*
Specify VMS extensions.	`-vax=`*v*	*page 83*
Allow debugging by `dbx` without `.o` files.	`-xs`	*page 99*

Floating–Point Options

For the following floating–point options, the most significant are listed first.

Table 3-5 Floating–Point Options

Action	Option	Details
Turn on SPARC nonstandard floating–point *(SPARC)*	`-fns`	*page 50*
Set IEEE rounding mode in effect at startup	`-fround=`*r*	*page 54*
Set IEEE trapping mode in effect at startup	`-ftrap=`*t*	*page 56*
Set floating-point optimization preferences	`-fsimple=`*n*	*page 55*
Se floating-point precision	`-fprecision=`*p*	*page 54*

Library Options

For the following library linking options, the most useful are listed first.

Table 3-6 Library Options

Action	Option	Details
Bind as dynamic or static next library listed on command.	`-B`*x*	*page 42*
Allow or disallow dynamic libraries for executable.	`-d`*x*	*page 47*
Build a dynamic shared library.	`-G`	*page 57*

Table 3-6 Library Options

Action	Option	Details
Search for libraries in this directory first.	`-L`*dir*	*page 61*
Link with library `lib`*x*.	`-l`*x*	*page 62*
Multithread safe libraries, low level threads.	`-mt`	*page 65*
No automatic libraries.	`-nolib`	*page 66*
No inline templates.	`-nolibmil`	*page 67*
No run path in executable.	`-norunpath`	*page 68*
Library—do not make library if relocations remain.	`-ztext`	*page 107*

Licensing Options

The following options are for licensing.

Table 3-7 Licensing Options

Action	Option	Details
Do not queue the license request.	`-noqueue`	*page 67*
Show license server user IDs.	`-xlicinfo`	*page 93*

Performance Options

For the following performance options, those with the greatest significance are listed first.

Table 3-8 Performance Options

Action	Option	Details
Faster execution and compilation using a set of options .	`-fast`	*page 50*
Optimize runtime performance—set optimization level to *n*.	`-O[`*n*`]`	*page 68*
Specify target hardware system.	`-xtarget=`*t*	*page 100*
Collect or use data for a profile to optimize (*SPARC*).	`-xprofile=`*p*	*page 97*
Double load—allow compiler to generate double load / store instructions in compiled code.	`-dalign`	*page 45*
Arithmetic—use simple arithmetic model.	`-fsimple`	*page 55*
Arithmetic—use SPARC non–standard floating point (*SPARC*).	`-fns`	*page 53*
Use inline library	`-libmil`	*page 62*

Table 3-8 Performance Options

Action	Option	Details
Traps—assume no memory–based traps (*SPARC*).	`–xsafe=mem`	*page 99*
Unroll loops—advise optimizer to unroll loops *n* times.	`–unroll=`*n*	*page 82*
Fast math—use special fast math routines (*SPARC*).	`–xlibmopt`	*page 93*
Architecture—specify target instruction set.	`–xarch=`*a*	*page 85*
Chip—specify target processor.	`–xchip=`*c*	*page 89*
Check data dependencies—analyze loops (*SPARC*).	`–depend`	*page 47*
Inline the listed user routines to optimize for speed.	`–inline=`*rlst*	*page 60*
Optimize across all source files on command line	`-xcrossfile`	*page 90*
Do no optimizations that increase code size.	`–xspace`	*page 100*
386—generate code for 80386 (*Intel*).	`–386`	*page 39*
486—generate code for 80486 (*Intel*).	`–486`	*page 39*
Pentium—generate code for pentium (*Intel*).	`–pentium`	*page 73*

Parallelization Options

For the following parallelization options, those with the greatest impact in most situations are listed first.Parallelization options require a WorkShop license. See the Fortran README files for details

Table 3-9 Parallelization Options (*SPARC*)

Action	Option	Details
Parallelize with `–autopar –explicitpar –depend`.	`–parallel`	*page 72*
Parallelize explicitly marked loops.	`-explicitpar`	*page 72*
Parallelize “reduction” loops.	`–reduction`	*page 77*
Parallelize loops automatically.	`-autopar`	*page 64*
Specify the style for MP directives (`cray` or `sun`).	`–mp=`*x*	*page 64*
Prepare loops for profiling parallelization.	`–Zlp`	*page 107*
List which loops are successfully parallelized.	`-loopinfo`	*page 63*
Enable thread performance analysis by `tha`.	`–Ztha`	*page 108*
Stack local variables to optimize for parallelization.	`–stackvar`	*page 79*
Show warnings about parallelization.	`-vpara`	*page 84*
Disable automatic parallelization.	`–noautopar`	*page 66*

Table 3-9 Parallelization Options (*SPARC*)

Action	Option	Details
Disable `-depend`.	`-nodepend`	*page 66*
Disable explicit parallelization.	`-noexplicitpar`	*page 66*
Disable reduction loop analysis.	`-noreduction`	*page 68*

Profiling Options

The following options enable runtime profiling in the compiled program. Depending on the options, profiling is done at either the basic block, procedure, or loop level.

Table 3-10 Profiling Options

Action—Enable Profiling by:	Option	Details
Basic block (using `tcov`, old style).	`-a`	*page 39*
Procedure (using `gprof`).	`-pg`	*page 73*
Procedure (using `prof`).	`-p`	*page 71*
Loops with parallelization (*SPARC*).	`-loopinfo`	*page 63*
Basic block (using `tcov`, new style) (*SPARC*).	`-xprofile=tcov`	*page 97*

Alignment Options

The following options are for specifying data alignment strategies.

Table 3-11 Alignment Options

Action	Option	Details
Align on 8–byte boundaries (*SPARC*).	`-f`	*page 50*
Force 8-byte alignment on all data.	`-dbl_align_all=yes`	*page 46*
Align and use double-word load / store.	`-dalign`	*page 45*
Allow for misaligned data (*SPARC*).	`-misalign`	*page 64*
Specify what VMS alignment features to use	`-vax=`*v*	*page 83*

Backward Compatibility and Legacy Options

The following options are provided for backward compatibility with earlier compiler releases, and certain Fortran legacy capabilities.

Table 3-12 Backward Compatibility Options

Action	Option	Details
Allow assignment to constant arguments.	`-copyargs`	*page 43*
External names—make external names without underscores.	`-ext_names=`*e*	*page 49*
Nonstandard arithmetic—allow nonstandard arithmetic.	`-fnonstd`	*page 52*
Optimize performance for the host system.	`-native`	*page 65*
Output—use old style list–directed output.	`-oldldo`	*page 70*
`DO` loops—use one trip `DO` loops.	`-onetrip`	*page 70*

Obsolescent Options

The following options are no longer functional in the current release of the `f77` and `f90` compilers. Their appearance on a compiler command does not cause an error, and no action is taken; they are ignored.

Table 3-13 Obsolescent Options

Original Intention	Option
Align Common block on page boundaries.	`-align _`*b*`_`
Use faster `malloc`	`-bsdmalloc`
Reduce size of executable file	`-nocx`
Set internal compiler table sizes	`-N[cdlnqsx]`*k*

All Options List

The following table lists all the Fortran 77 and Fortran 90 compiler options, and indicates any platform restrictions. The **SPARC**, **PPC**, and **Intel** columns indicate availability of an option on SPARC, PowerPC, and Intel Solaris 2.x systems, respectively:

77	*indicates the option is only available with* `f77` *on that platform*
90	*indicates the option is only available with* `f90` *on that platform*
77 / 90	*indicates the option is available with both* `f77` *and* `f90` *on that platform*
–	*indicates the option is not available on that platform*

Note that `f90` 1.2 is *only* available on SPARC systems.

Options that are not available for a compiler on a particular platform will still be accepted *silently* by the compiler. That is, the compiler will accept the option on the command-line on that platform without issuing a warning, but the option does nothing.

The options reference section that follows gives full details and examples for each option. (Obsolescent options are not listed; see page 32.)

Table 3-14 Options Index

Option	SPARC	PPC	Intel	Action	Details
`-386`	–	–	77	*Compile for Intel 80386.*	*page 39*
`-486`	–	–	77	*Compile for Intel 80486.*	*page 39*
`-a`	77	77	77	*Profile by basic block using tcov, old style.*	*page 39*
`-ansi`	77 / 90	77	77	*Identify many non–ANSI extensions.*	*page 40*
`-arg=local`	77	77	77	*Preserve actual arguments over ENTRY statements.*	*page 40*
`-autopar`	77	–	–	*Enable automatic loop parallelization.*	*page 41*
`-B`*x*	77 / 90	77	77	*Allow dynamic or require static library linking.*	*page 42*
`-C`	77	77	77	*Check array references for out of range subscripts.*	*page 42*
`-c`	77 / 90	77	77	*Compile only; produce object .o files, but suppress linking.*	*page 43*
`-cg89`	77 / 90	–	–	*Compile for generic SPARC architecture.*	*page 43*
`-cg92`	77 / 90	–	–	*Compile for SPARC V8 architecture.*	*page 43*
`-copyargs`	77	77	77	*Allow assignment to constant arguments.*	*page 43*
`-D`*name*	77 / 90	77	77	*Define symbol name for the preprocessor.*	*page 44*
`-dalign`	77 / 90	–	–	*Force 8-byte alignment and enable double word load/stores.*	*page 45*
`-db`	90	–	–	*Generate a compiler information file (CIF).*	*page 45*

3

Table 3-14 Options Index

Option	SPARC	PPC	Intel	Action	Details
`-dbl`	77	77	77	*Double the default size for REAL, INTEGER, DOUBLE, and COMPLEX.*	*page 45*
`-dbl_align_all`	77	77	77	*Force alignment of all data on 8-byte boundaries.*	*page 46*
`-depend`	77	–	–	*Analyze loops for data dependencies.*	*page 47*
`-dryrun`	77/90	77	77	*Show commands built by driver, but do not compile.*	*page 47*
`-d{y|n}`	77/90	77	77	*Allow or disallow dynamic libraries for the entire executable*	*page 47*
`-e`	77/90	77	77	*Accept extended length input source line.*	*page 47*
`-erroff=`*taglist*	77	77	77	*Suppress warning messages listed by tag name.*	*page 48*
`-errtags`	77	77	77	*Display the message tag with each warning message.*	*page 48*
`-explicitpar`	77/90	–	–	*Parallelize loops explicitly marked by directives.*	*page 48*
`-ext_names=`*e*	77	77	77	*Create external names with or without trailing underscore.*	*page 49*
`-F`	77/90	77	77	*Invoke the source file preprocessor, but do not compile.*	*page 49*
`-f`	77/90	–	–	*Align data on 8–byte boundaries.*	*page 50*
`-fast`	77/90	77	77	*Optimize for speed of execution using a selection of options.*	*page 50*
`-fixed`	90	–	–	*Specify fixed-format source input files.*	*page 52*
`-flags`	77/90	77	77	*Synonym for –help.*	*page 52*
`-fnonstd`	77/90	77	77	*Initialize floating–point hardware to non-standard preferences.*	*page 52*
`-fnonstop`	90	–	–	*Disable floating–point exception trapping.*	*page 53*
`-fns`	77/90	–	–	*Select the SPARC nonstandard floating–point mode.*	*page 53*
`-fprecision=`*p*	–	–	77	*Initialize floating-point precision mode on Intel.*	*page 54*
`-free`	90	–	–	*Specify free-format source input files.*	*page 54*
`-fround=`*r*	77/90	77	77	*Set the IEEE rounding mode in effect at startup.*	*page 54*
`-fsimple`	77	77	77	*Select floating-point optimization preferences.*	*page 55*
`-fstore`	–	–	77	*Force precision of floating–point expressions.*	*page 56*
`-ftrap=`*t*	77/90	77	77	*Set floating–point trapping mode in effect at startup.*	*page 56*
`-G`	77/90	77	77	*Build a dynamic shared library instead of an executable file.*	*page 57*
`-g`	77/90	77	77	*Compile for debugging.*	*page 57*
`-h`*name*	77/90	77	77	*Specify the name of the generated dynamic shared library.*	*page 58*
`-help`	77/90	77	77	*Display a summary list of compiler options.*	*page 58*
`-I`*dir*	77/90	77	77	*Add dir to the INCLUDE file search path.*	*page 58*
`-i2`	77	77	77	*Set the default integer size to two bytes.*	*page 59*

Table 3-14 Options Index

Option	SPARC	PPC	Intel	Action	Details
`-i4`	77 / 90	77	77	*Set the default integer size to four bytes.*	*page 59*
`-inline=`*rlst*	77	77	77	*Inline specified routines.*	*page 60*
`-Kpic`	77	77	77	*Synonym for –pic.*	*page 61*
`-KPIC`	77	77	77	*Synonym for –PIC.*	*page 61*
`-L`*dir*	77 / 90	77	77	*Add dir to list of directories to search for libraries.*	*page 61*
`-l`*x*	77 / 90	77	77	*Add library libx.a to linker's list of search libraries.*	*page 62*
`-libmil`	77	77	77	*Inline selected libm library routines for optimization.*	*page 62*
`-loopinfo`	77	–	–	*Show parallelization results.*	*page 63*
`-M`*dir*	90	–	–	*Add dir to directories searched for Fortran 90 modules.*	*page 63*
`-misalign`	77	77	–	*Allow misaligned data.*	*page 64*
`-mp=`*x*	77	–	–	*Select the style for parallelization directives.*	*page 64*
`-mt`	77 / 90	77	77	*Require multithread-safe libraries.*	*page 65*
`-native`	77 / 90	77	77	*Optimize performance for the host system.*	*page 65*
`-noautopar`	77	–	–	*Disable automatic parallelization.*	*page 66*
`-nodepend`	77	–	–	*Cancel –depend in command line.*	*page 66*
`-noexplicitpar`	77	–	–	*Disable explicit parallelization.*	*page 66*
`-nofstore`	–	–	77	*Disable forcing precision of expression.*	*page 66*
`-nolib`	77 / 90	77	77	*Disable linking with system libraries.*	*page 66*
`-nolibmil`	77	77	77	*Cancel -libmil on command line.*	*page 67*
`-noqueue`	77	77	77	*Disable license queueing.*	*page 67*
`-noreduction`	77 / 90	–	–	*Cancel -reduction on command line.*	*page 68*
`-norunpath`	77 / 90	77	77	*Do not build a runtime shared library search path into the executable.*	*page 68*
`-O[`*n*`]`	77 / 90	77	77	*Specify optimization level.*	*page 68*
`-o` *outfil*	77 / 90	77	77	*Specify the name of the executable file to be written.*	*page 70*
`-oldldo`	77	77	77	*Select "old" list–directed output style.*	*page 70*
`-onetrip`	77 / 90	77	77	*Enable one trip DO loops.*	*page 70*
`-p`	77 / 90	77	77	*Compile for profiling with the prof profiler.*	*page 71*
`-pad[=`*p*`]`	77	–	–	*Insert padding for efficient use of cache.*	*page 71*
`-parallel`	77 / 90	–	–	*Parallelize loops with: -autopar, –explicitpar, -depend*	*page 72*
`-pentium`	–	–	77	*Compile for Intel Pentium.*	*page 73*
`-pg`	77 / 90	77	77	*Compile for profiling with the prof profiler.*	*page 73*

Table 3-14 Options Index

Option	SPARC	PPC	Intel	Action	Details
`-pic`	77 / 90	77	77	*Compile position–independent code for shared library.*	*page 73*
`-PIC`	77 / 90	77	77	*Compile position-independent code, but with 32-bit addresses.*	*page 74*
`-Qoption` *pro op*	77 / 90	77	77	*Pass options to compilation phase pr.*	*page 75*
`-qp`	77 / 90	77	77	*Synonym for -p.*	*page 75*
`-R` *list*	77 / 90	77	77	*Build dynamic library search paths into the executable file.*	*page 75*
`-r8`	77	77	77	*Double default byte size for REAL,INTEGER, DOUBLE and COMPLEX.*	*page 76*
`-reduction`	77 / 90	–	–	*Recognize reduction operations in loops.*	*page 77*
`-S`	77 / 90	77	77	*Compile and only generate assembly code.*	*page 78*
`-s`	77 / 90	77	77	*Strip the symbol table out of the executable file.*	*page 78*
`-sb`	77	77	77	*Produce table information for the WorkShop source code browser.*	*page 78*
`-sbfast`	77	77	77	*Produce only source code browser tables.*	*page 78*
`-silent`	77	77	77	*Suppress compiler messages.*	*page 78*
`-stackvar`	77 / 90	77	77	*Force all local variables to be allocated on the memory stack.*	*page 79*
`-stop_status`	77	77	77	*Permit STOP statement to return an integer status value.*	*page 80*
`-temp=`*dir*	77 / 90	77	77	*Define directory for temporary files.*	*page 81*
`-time`	77 / 90	77	77	*Time each compilation phase.*	*page 81*
`-U`	77	77	77	*Recognize upper and lower case in source files.*	*page 81*
`-u`	77	77	77	*Report undeclared variables.*	*page 82*
`-unroll=`*n*	77	77	77	*Enable unrolling of DO loops where possible.*	*page 82*
`-V`	77 / 90	77	77	*Show name and version of each compiler pass.*	*page 82*
`-v`	77 / 90	77	77	*Verbose mode – show details of each compiler pass.*	*page 83*
`-vax=`*v*	77	77	77	*Specify choice of VMS Fortran extensions enabled.*	*page 83*
`-vpara`	77	–	–	*Show verbose parallelization messages.*	*page 84*
`-w`	77 / 90	77	77	*Suppress warning messages.*	*page 84*
`-xa`	77	77	77	*Synonym for –a.*	*page 84*
`-xarch=`*a*	77	77	77	*Specify target architecture instruction set.*	*page 85*
`-xautopar`	77	–	–	*Synonym for –autopar.*	*page 87*
`-xcache=`*c*	77	–	–	*Define cache properties for the optimizer.*	*page 87*
`-xcg89`	77	–	–	*Synonym for –cg89.*	*page 88*
`-xcg92`	77	–	–	*Synonym for –cg92.*	*page 88*

Table 3-14 Options Index

Option	SPARC	PPC	Intel	Action	Details
`-xchip=`*c*	77	77	77	*Specify target processor for the optimizer.*	*page 89*
`-xcrossfile`	77	–	–	*Enable optimization and inlining across source files.*	*page 90*
`-xdepend`	77	–	–	*Synonym for –depend.*	*page 90*
`-xexplicitpar`	77	–	–	*Synonym for –explicitpar.*	*page 90*
`-xF`	77	77	77	*Allow function–level reordering by WorkShop Analyzer.*	*page 90*
`-xhelp=`*h*	77	77	77	*Show summary help information on options or README file.*	*page 91*
`-xildoff`	77/90	–	–	*Turn off the Incremental Linker.*	*page 91*
`-xildon`	77/90	–	–	*Turn on the Incremental Linker.*	*page 91*
`-xinline=`*rlst*	77	77	77	*Synonym for –inline=f1[,...,fn].*	*page 92*
`-xl`*[d]*	77	77	77	*Enable more VMS Fortran extensions.*	*page 92*
`-xlibmil`	77	77	77	*Synonym for –libmil.*	*page 93*
`-xlibmopt`	77/90	–	–	*Use library of optimized math routines.*	*page 93*
`-xlic_lib=`*libs*	77/90	77	77	*Link with the specified Sun licensed libraries.*	*page 93*
`-xlicinfo`	77/90	77	77	*Show license server information.*	*page 93*
`-Xlist`	77/90	77	77	*Produce listings and do global program checking.*	*page 94*
`-xloopinfo`	77	–	–	*Synonym for –loopinfo.*	*page 95*
`-xnolib`	77/90	77	77	*Synonym for –nolib.*	*page 96*
`-xnolibmil`	77/90	77	77	*Synonym for –nolibmil.*	*page 96*
`-xnolibmopt`	77/90	77	77	*Do not use fast math library.*	*page 96*
`-xO[`*n*`]`	77/90	77	77	*Synonym for –O[n].*	*page 96*
`-xpad=`*p*	77	–	–	*Synonym for -pad.*	*page 96*
`-xparallel`	77/90	–	–	*Synonym for –parallel.*	*page 96*
`-xpg`	77/90	77	77	*Synonym for –pg.*	*page 96*
`-xpp={fpp|cpp}`	77/90	77	77	*Select source file preprocessor.*	*page 96*
`-xprofile=`*p*	77	–	77	*Collect or optimize with runtime profiling data.*	*page 97*
`-xreduction`	77/90	–	–	*Synonym for –reduction.*	*page 77*
`-xregs=`*r*	77	–	–	*Specify register usage.*	*page 98*
`-xs`	77	77	77	*Allow debugging by dbx without object (.o) files .*	*page 99*
`-xsafe=mem`	77	–	–	*Assume no memory–based traps.*	*page 99*
`-xsb`	77	77	77	*Synonym for –sb.*	*page 100*
`-xsbfast`	77	77	77	*Synonym for –sbfast.*	*page 100*

Table 3-14 Options Index

Option	SPARC	PPC	Intel	Action	Details
`-xspace`	77	–	–	*Do not allow optimizations to increase code size.*	*page 100*
`-xtarget=`*t*	77	77	77	*Specify system for optimization.*	*page 100*
`-xtime`	77/90	77	77	*Synonym for –time.*	*page 105*
`-xtypemap=`*spec*	77	77	77	*Specify default data mappings.*	*page 106*
`-xunroll=`*n*	77	77	77	*Synonym for –unroll=n.*	*page 107*
`-xvpara`	77	–	–	*Synonym for –vpara.*	*page 107*
`-Zlp`	77	–	–	*Compile for loop performance profiling by looptool.*	*page 107*
`-ztext`	77	–	–	*Generate only pure libraries with no relocations.*	*page 107*
`-Ztha`	77	–	–	*Compile for performance profiling with Thread Analyzer.*	*page 108*

Options Reference

This section shows all `f77` and `f90` compiler command-line option flags, including various risks, restrictions, caveats, interactions, examples, and other details. Each description indicates platform availability of the option, using the same legend as in the summary list, page 33.

`-386` Compile for Intel 80386.

♦ **SPARC: – PPC: – Intel:77**

Generate code that exploits features available on Intel 80386 compatible processors. The default is `-386`.

`-486` Compile for Intel 80486.

♦ **SPARC: – PPC: – Intel:77**

Generate code that exploits features available on Intel 80486 compatible processors. The default is `-386`. Code compiled with `-486` does run on 80386 hardware, but it may run slightly slower.

`-a` Profile by basic block using `tcov`, old style.

♦ **SPARC:77 PPC:77 Intel:77**

This is the old style of basic block profiling for `tcov`. See `-xprofile=tcov` for information on the new style of profiling and the `tcov`(1) man page for more details. Also see the manual, *Performance Profiling Tools*.

Insert code to count the times each basic block of statements is executed. This invokes a runtime recording mechanism that creates one `.d` file for every `.f` file at normal program termination. The `.d` file accumulates execution data for the corresponding source file. The `tcov`(1) utility can then be run on the source file(s) to generate statistics about the program. The summary output produced by `tcov` is written to *file*`.tcov` for each source file. `-pg` and `gprof` are complementary to `-a` and `tcov`.

If set at compile–time, the `TCOVDIR` environment variable specifies the directory where the `.d` and `.tcov` files are located. If this variable is not set, then the `.d` files remain in the same directory as the `.f` files.

The –xprofile=tcov and the –a options are compatible in a single executable. That is, you can link a program that contains some files which have been compiled with –xprofile=tcov, and others with –a. You cannot compile a single file with both options.

If you compile and link in separate steps, and you compile with -a, then be sure to link with -a. You can mix -a with -O*n*; in some earlier versions -a overrode -O*n*.

For details, see the chapter *Performance Profiling* in the *Fortran Programmer's Guide*.

–ansi Identify many non–ANSI extensions.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Warning messages are issued for any uses of non-standard Fortran 77 or Fortran 90 extensions in the source code.

–arg=local Preserve actual arguments over ENTRY statements.

♦ **SPARC:77 PPC:77 Intel:77**

When you compile a subprogram with alternate entry points with this option, f77 uses *copy restore* to preserve the association of dummy and actual arguments. For example, the following program would require compilation with -arg=local to insure proper execution:

```
      A = SETUP(ALPHA,BETA,GAMMA)
      ZORK = FXGAMMA(GCONST)
      ...
      FUNCTION SETUP(A1,A2,A3)
      ...
      ENTRY FXGAMMA(F)
      FXGAMMA = F*GAMMA
      ...
      RETURN
      END
```

Without this option, there is no guarantee that the correct values of the actual arguments from the SETUP call will be referenced when the routine is entered through FXGAMMA. Code that relies on -arg=local is non-standard.

–autopar Enable automatic loop parallelization.

♦ **SPARC:77 PPC: – Intel: –**

Finds and parallelizes appropriate loops for running in parallel on multiple processors. Analyzes loops for inter–iteration data dependencies and loop restructuring. If the optimization level is not specified `–O3` or higher, it will automatically be raised to `–O3`.

`-g` cancels `-autopar`. Debugging is facilitated by specifying `-g` without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the `dbx` documentation for details.

To improve performance, also specify the `–stackvar` option when using any of the parallelization options, including `–autopar`.

Avoid `-autopar` if the program already contains explicit calls to the `libthread` threads library. See note with `-mt` on page 65.

The `-autopar` option is not appropriate on a single–processor system, and the compiled code will generally run slower.

To run a parallelized program on a multiprocessor system, you must set the `PARALLEL` environment variable prior to execution. The default is 1. However, do not request more processors than are available.

If you use `–autopar` and compile and link in *one* step, the microtasking library and the threads–safe Fortran runtime library will automatically be linked. If you use `–autopar` and compile and link in *separate* steps, then you must also link with `–autopar` to insure linking the appropriate libraries.

Other parallelization options are `–parallel` and `–explicitpar`. While `–autopar` is not supported with the `f90` compiler, both `–parallel` and `–explicitpar` are. Also, the `-reduction` option may be used with `-autopar`.

Refer to the *Fortran: Programmer's Guide* for more information on parallelization.

–B{static|dynamic} Allow dynamic or require static library linking.

♦ **SPARC: 77/90 PPC:77 Intel:77**

No space is allowed between –B and dynamic or static. The default, without -B specified, is -Bdynamic.

- –Bdynamic: Prefer *dynamic* linking (try for shared libraries).
- –Bstatic: Require *static* linking (no shared libraries).

Also note:

- If you specify static, but the linker finds only a dynamic library, then the library is not linked with a warning that the "library was not found."
- If you specify dynamic, but the linker finds only a static version, then that library is linked, with no warning.

You can toggle –Bstatic and –Bdynamic on the command line. That is, you can link some libraries statically and some dynamically by specifying -Bstatic and -Bdynamic any number of times on the command line.

These are loader and linker options. Compiling and linking in separate steps with –B*x* on the compile command will require it in the link step as well.

–C Check array references for out of range subscripts.

♦ **SPARC:77 PPC:77 Intel:77**

Subscripting arrays beyond their declared sizes may result in unexpected results, including segmentation faults. The –C option checks for possible array subscript violations in the source code and during execution.

Specifying -C may make the executable file larger.

If the –C option is used, array subscript violations are treated as an error. If an array subscript range violation is detected in the source code during compilation, it is treated as a compilation error.

If an array subscript violation can only be determined at runtime, the compiler generates range–checking code into the executable program. This may cause an increase in execution time. As a result, it is appropriate to enable full array subscript checking while developing and debugging a program, then recompiling the final production executable without subscript checking.

–c Compile only; produce object `.o` files, but suppress linking.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Suppress linking. Compile a `.o` file for each source file. If only a single source file is being compiled, the –`o` option can be used to specify the name of the `.o` file written.

–cg89 Compile for generic SPARC architecture.

♦ **SPARC: 77/90 PPC: – Intel: –**

This option is a macro for: `–xarch=v7 –xchip=old –xcache=64/32/1` which is equivalent to `-xtarget=ss2` .

–cg92 Compile for SPARC V8 architecture.

♦ **SPARC: 77/90 PPC: – Intel: –**

This option is a macro for:
`–xarch=v8 –xchip=super –xcache=16/32/4:1024/32/1` which is equivalent to `-xtarget=ss1000` .

–copyargs Allow assignment to constant arguments.

♦ **SPARC:77 PPC:77 Intel:77**

Allow a subprogram to change a dummy argument that is a constant. This option is provided only to allow legacy code to compile and execute without a runtime error.

- Without `–copyargs`, if you pass a constant argument to a subroutine, and then within the subroutine try to change that constant, the run aborts.
- With `–copyargs`, if you pass a constant argument to a subroutine, and then within the subroutine change that constant, the run does not necessarily abort.

Code that aborts unless compiled with `–copyargs` is, of course, not FORTRAN standard compliant. Also, such code is often unpredictable.

-D*name*[=*def*] Define symbol *name* for the preprocessor.

♦ **SPARC: 77/90 PPC:77 Intel:77**

-D*name*=*def*

Define *name* to be *def*

-D*name*

Define *name* to be `1`

With `.F` and `.F90` (`f90` only) source files, define *name* to the source-code preprocessor as if:

```
#define name[=def]
```

had appears in the source file. If no =*def* specified, the name *name* is defined as the value 1.

Following are the predefined values:

- The compiler version is predefined (in hex) in `_ _SUNPRO_F77`

  Example: For Fortran 4.2, `_ _SUNPRO_F77=0x42`

- The following values are predefined on appropriate systems:

  `_ _sparc`, `_ _unix`, `_ _sun`, `_ _i386`, `_ _SVR4`, `_ _SunOS_5_3`

  For instance, the value `_ _i386` is defined on systems compatible with the 80386 (including the 80486), and it is not defined on SPARC systems. You can use these values in such preprocessor conditionals as the following.

  ```
  #ifdef _ _sparc
  ```

- From earlier releases, these values (with no underscores) are also predefined, but they may be deleted in a future release:

  `sparc`, `unix`, `sun`, `i386`

The compilers use the `fpp`(1) preprocessor by default. Like the C preprocessor `cpp`(1), `fpp` expands source code macros and enables conditional compilation of code. Unlike `cpp`, `fpp` understand Fortran syntax, and is preferred as a Fortran preprocessor. Use the –`xpp=cpp` flag to force the compiler to specifically use `cpp` rather than `fpp`.

–dalign Force 8-byte alignment and enable double word load / stores.

♦ **SPARC: 77/90 PPC: – Intel: –**

Wherever profitable, the compiler will generate double–word load / store instructions for faster execution of double and quad precision computations.

Using this option automatically triggers the `–f` option, which causes all double–precision and quadruple–precision data types (both real and complex) to be aligned on 8-byte boundaries.

Using both `–dbl` and `–dalign` also causes 64–bit integer data type to be 8–byte aligned.

Caution – `-dalign` may result in non-ANSI standard alignment of data, which could cause problems with variables in `EQUIVALENCE` or `COMMON`.

If you compile one subprogram with `-dalign`, compile all subprograms of the program with `–dalign`.

–db Generate a compiler information file (CIF).

♦ **SPARC:90 PPC: – Intel: –**

The CIF is used by `f90browse`. See the *f90browse* manual.

–dbl Double the default size for `REAL`, `INTEGER`, `DOUBLE`, and `COMPLEX`.

♦ **SPARC:77 PPC:77 Intel:77**

`–dbl` promotes the default byte size for `REAL`, `INTEGER`, `DOUBLE`, and `COMPLEX` variables declared *without an explicit byte size* as follows:

Table 3-15 Default Data Sizes and –`dbl` (Bytes)

Without –dbl option		With –dbl option		
Data Type	**default**	**SPARC**	**Intel**	**PowerPC**
`INTEGER`	4	8	8	8
`REAL`	4	8	8	8
`DOUBLE`	8	16	8	16

This option applies to variables, parameters, constants, and functions.

Also, LOGICAL is treated as INTEGER, COMPLEX as two REALs, and DOUBLE COMPLEX as two DOUBLEs.

Compare –dbl with –r8:

–dbl and –r8 can be expressed in terms of the more general –xtypemap= option:

On SPARC and PowerPC:		
–dbl	*same as:*	–xtypemap=real:64,double:128,integer:64
–r8	*same as:*	–xtypemap=real:64,double:128,integer:mixed
On Intel:		
–dbl	*same as:*	–xtypemap=real:64,double:64,integer:64
–r8	*same as:*	–xtypemap=real:64,double:64,integer:mixed

- For all of the floating point data types, –dbl works the same as –r8; using both -r8 and -dbl produces the same results as using only -dbl.
- For INTEGER and LOGICAL data types, –dbl is different from -r8:
  - –dbl allocates 8 bytes, and does 8–byte arithmetic
  - –r8 allocates 8 bytes, and does only 4–byte arithmetic (“mixed”)

In general, if you compile *one* subprogram with –dbl, then be sure to compile *all* subprograms of that program with –dbl. This is particularly important with programs communicating through files with unformatted I/O — if one program is compiled with –dbl, then the other program must similarly be compiled. Be also aware that this option alters the default data size of function names, including calls to library functions, unless the function name is typed explicitly with a data size.

–dbl_align_all=*y* Force alignment of all data on 8-byte boundaries.

♦ **SPARC:77 PPC:77 Intel:77**

y is either yes or no. If *y* is yes, all variables will be aligned on 8-byte boundaries. Default is –dbl_align_all=no.

–depend Analyze loops for data dependencies.

♦ **SPARC:77 PPC: – Intel: –**

Analyze loops for inter–iteration data dependencies and do loop restructuring. This option will raise the optimization level to `O3` if no optimization level is specified, or if it is specified less than `O3`. Dependence analysis is also included with `-autopar` or `-parallel`. The dependence analysis is done at compile time. (See the *Fortran Programmer's Guide*.)

`-g` cancels `-depend`.

–dryrun Show commands built by driver, but do not compile.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Useful when debugging, this option displays the commands it will run to perform the compilation.

–d{y|n} Allow or disallow *dynamic* libraries for the entire executable

♦ **SPARC: 77/90 PPC:77 Intel:77**

- `–dy`: Yes, allow *dynamically* bound libraries (*allow* shared libraries).
- `–dn`: No, do *not* allow dynamically bound libraries (*no* shared libraries).

The default, if not specified, is `–dy`.

Unlike –B*x*, this option applies to the *whole* executable and need appear only once on the command line.

`–dy|-dn` are loader and linker options. If you compile and link in separate steps with these options, then you need the same option in the link step.

–e Accept extended length input source line.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Accept source lines up to 132 characters long. The compiler pads on the right with trailing blanks to column 132. If you use continuation lines while compiling with `–e`, then do not split character constants across lines, otherwise, unnecessary blanks may be inserted in the constants.

–erroff=*taglist* Suppress warning messages listed by tag name.

♦ **SPARC:77 PPC:77 Intel:77**

Suppress displaying the warning messages specified in the comma-separated list of tag names *taglist*. If *taglist* consists of `%none`, no warnings are suppressed. If *taglist* consists of `%all`, all warnings are suppressed (this is equivalent to the `–w` option.)

–errtags Display the message tag with each warning message.

♦ **SPARC:77 PPC:77 Intel:77**

With this option, the compiler's internal error tag name will appear along with warning messages. The default is not to display the tag.

```
demo% f77 –errtags ink.f
ink.f:
 MAIN:
"ink.f", line 11: Warning: local variable "i" never used
(WDECL_LOCAL_NOTUSED)  <- The warning message's tag name
```

–explicitpar Parallelize loops explicitly marked by directives.

♦ **SPARC: 77/90 PPC: – Intel: –**

This option turns on explicit parallelization. `DO` loops immediately preceded by `DOALL` directives will have threaded, parallel code compiled for them. Parallelization is only appropriate on multiprocessor systems. This option should not be used to compile programs that already do their own multithreading with calls to the `libthread` library.

If the optimization level is not `-O3` or higher, it is raised to `-O3` automatically.

The compiler will generate parallel code even if there are data dependencies in the `DO` loop that would cause the loop to generate incorrect results when run in parallel. With explicit parallelization, it is the user's responsibility to correctly analyze loops for data dependency problems before marking them with parallelization directives.

For details, see the *Parallelization* chapter in the *Fortran Programmer's Guide.*

To improve performance, also specify the –stackvar option when using any of the parallelization options, including –explicitpar.

-g cancels -explicitpar. Debugging is facilitated by specifying -g without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the dbx documentation for details.

If you use –explicitpar and compile and link in *one* step, then linking automatically includes the microtasking library and the threads–safe FORTRAN runtime library. If you use –explicitpar and compile and link in *separate* steps, then you must also *link* with –explicitpar.

–ext_names=*e* Create external names with or without trailing underscore.

♦ **SPARC:77 PPC:77 Intel:77**

e must be either plain or underscore. The default is underscore.

–ext_names=plain: Do not add trailing underscore.

–ext_names=underscore: Add trailing underscore.

An external name is a name of a subroutine, function, block data subprogram, or labeled common. This option affects both the name of the routine's entry point and the name used in calls to it. This option may be used to allow Fortran 77 routines to call and be called by other language routines.

–F Invoke the source file preprocessor, but do not compile.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Apply the fpp preprocessor to .F files (and .F90 files with f90) and write the processed result on a file with the same name but with suffix changed to .f (or .f90), but do not compile.

Example:

```
f77 –F source.F
```

writes to source.f

fpp is the default preprocessor for Fortran. The C preprocessor, cpp, can be selected instead by specifying –xpp=cpp.

-f Align data on 8–byte boundaries.

♦ **SPARC: 77/90 PPC: – Intel: –**

Align all `COMMON` blocks and all double–precision and quadruple–precision local data on 8–byte boundaries. This applies to complex data as well.

Using `–dbl` with `–f` aligns all 64–bit integer data on 8–byte boundaries as well.

A program requiring `-f` may not be standard and may not be portable.

Compiling *any* part of a program with `-f` requires compiling *all* subprograms of that program with `-f`.

By itself, this option does not enable the compiler to generate faster double word fetch/store instructions on double and quad precision data.
See `-dalign`, page 45

-fast Optimize for speed of execution using a selection of options.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Select options that optimize for speed of execution without excessive compilation time. This option provides close–to–the–maximum performance for many applications.

If you compile and link in separate steps, and you compile with `-fast`, then be sure to link with `-fast`.

Note – This option is defined as a particular selection of other options that is subject to change from one release to another, and between compilers. Also, some of the options selected by `–fast` may not be available on some platforms.

`–fast` selects the following options:

- The `-native` hardware target

If the program is intended to run on a different target than the compilation machine, follow the `–fast` with a code–generator option. For example:

```
f77 -fast xtarget=ultra ...
```

- The `–O4` optimization level option
- The `-depend` option (*SPARC only*)
- The `–libmil` option for system–supplied inline expansion templates

  For C functions that depend on exception handling, follow `-fast` by `-nolibmil`: `-fast –nolibmil`. With `–libmil`, exceptions cannot be detected with `errno` or `matherr`(3m).

- The `–fsimple=1` option for a simple floating–point model

  `–fsimple` is unsuitable if strict IEEE 754 standards compliance is required.

- The `–dalign` option to generate double loads and stores (*SPARC only*)

  Using this option may generate non-ANSI standard Fortran data alignment.

- The `–xlibmopt` option (*SPARC only*)
- *For Intel only*: `–nofstore`
- `–fns –ftrap=%none` to turn off all trapping.

The set of options selected by `-fast` differ across platforms:

Table 3-16 `-fast` selections across platforms

Option	SPARC	Intel	PowerPC
`-dalign`	X	—	—
`-depend`	X	—	—
`-fns`	X	X	X
`-fsimple=1`	X	—	—
`-ftrap=%none`	X	X	X
`-libmil`	X (see note)	X	X
`-native`	X	X	X
`-nofstore`	—	X	—
`-O4`	X	X	X
`-xlibmopt`	X	X	—

Note – With `f90`, the `-f` option is substituted for `-libmil` on SPARC.

It is possible to add or subtract from this list by following the `–fast` option with other options, as in:

```
f77 –fast –fsimple=2 –xnolibmopt ...
```

which overrides the `–fsimple=1` option and disables the `–xlibmopt` selected by `–fast`.

`–fixed`

Specify fixed-format source input files.

♦ **SPARC:90 PPC: – Intel: –**

All source files on the command-line will be interpreted as `f77` fixed format regardless of filename extension. Normally, `f90` interprets only `.f` files as fixed format, `.f90` as free format.

`–flags`

Synonym for `–help`.

♦ **SPARC: 77/90 PPC:77 Intel:77**

`–fnonstd`

Initialize floating–point hardware to non-standard preferences.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This option is a synonym for the combination `–fns –ftrap=common` on SPARC, `-ftrap=common` on Intel and PowerPC.

Specifying `–fnonstd` is approximately equivalent to the following two calls at the beginning of a Fortran main program.

```
i=ieee_handler("set", "common", SIGFPE_ABORT)
call nonstandard_arithmetic()
```

The `nonstandard_arithmetic()` routine replaces the obsolete `abrupt_underflow()` routine of earlier releases.

To be effective, the main program must be compiled with this option.

On some floating–point hardware, the `nonstandard_arithmetic()` call causes all underflows to produce zero rather than a possibly subnormal number, as the IEEE standard requires. This may be a performance improvement. See `ieee_functions`(3m).

The `–fnonstd` option allows hardware traps to be enabled for floating–point overflow, division by zero, and invalid operation exceptions. These are converted into SIGFPE signals, and if the program has no SIGFPE handler, it terminates with a dump of memory to a `core` file. See `ieee_handler`(3m).

`–fnonstop` Disable floating–point exception trapping.

♦ **SPARC:90 PPC: – Intel: –**

Default behavior of `f90` is to trap on invalid, overflow, and divide by zero floating-point exceptions. Specifying `–fnonstop` disables exception trapping; execution continues without stopping.

`–fns` Select the SPARC nonstandard floating–point mode.

♦ **SPARC: 77/90 PPC: – Intel: –**

The default is the SPARC standard floating–point mode. (See the *Floating-Point* chapter of the *Fortran Programmer's Guide*.)

Floating-point arithmetic is initialized to nonstandard preferences on program startup:

- Abort on exceptions
- Flush denormalized numbers to zero if it will improve speed

Where *x* does not cause total underflow, *x* is a *denormalized number* if and only if `|x|` is in one of the ranges indicated:

Data Type	Range
`REAL`	`0.0 < |x| < 1.17549435e–38`
`DOUBLE PRECISION`	`0.0 < |x| < 2.22507385072014e–308`

See the *Numerical Computation Guide* for details on denormalized numbers, and the *Fortran Programmer's Guide* chapter *Floating-Point* for more information about this and similar options.

The standard initialization of floating–point preferences is the default:

- IEEE 754 floating–point arithmetic is *nonstop* (do not abort on exception).
- Underflows are gradual.

To be effective, the main program must be compiled with this option.

`-fprecision=`*p* Initialize floating-point precision mode on Intel.

♦ **SPARC: – PPC: – Intel:77**

p is either `single`, `double`, or `extended`.

Initialize the floating-point hardware precision mode to single, double, or extended. Compile the main program with this option. (See the *Floating-Point* chapter of the *Fortran Programmer's Guide*.)

`–free` Specify free-format source input files.

♦ **SPARC:90 PPC: – Intel: –**

All source files on the command-line will be interpreted as `f90` free format regardless of filename extension. Normally, `f90` interprets `.f` files as fixed format, `.f90` as free format.

`–fround=`*r* Set the IEEE rounding mode in effect at startup.

♦ **SPARC: 77/90 PPC:77 Intel:77**

r must be one of: `nearest`, `tozero`, `negative`, `positive`.

The default is `–fround=nearest`.

This option sets the IEEE 754 rounding mode that:

- Can be used by the compiler in evaluating constant expressions.
- Is established at runtime during the program initialization.

The meanings are the same as those for the `ieee_flags` function. (See the *Floating-Point* chapter of the *Fortran Programmer's Guide*.)

To be effective, compile the main program with this option.

-fsimple[=*n*] Select floating-point optimization preferences.

♦ **SPARC:77 PPC:77 Intel:77**

Allow the optimizer to make simplifying assumptions concerning floating–point arithmetic. (See the *Floating-Point* chapter of the *Fortran Programmer's Guide*.)

For consistent results, compile all units of a program with the same –`fsimple` option.

If *n* is present, it must be 0, 1, or 2. The defaults are:

- If there is no `-fsimple`[=*n*] then the compiler uses `-fsimple=0`
- If there is only `-fsimple` then the compiler uses `-fsimple=1`

`-fsimple=0`

Permit no simplifying assumptions. Preserve strict IEEE 754 conformance.

`-fsimple=1`

Allow conservative simplifications. The resulting code does not strictly conform to IEEE 754, but numeric results of most programs are unchanged.

With `-fsimple=1`, the optimizer can assume the following:

- IEEE 754 default rounding/trapping modes do not change after process initialization.
- Computations producing no visible result other than potential floating point exceptions may be deleted.
- Computations with Infinity or NaNs as operands need not propagate NaNs to their results; e.g., `x*0` may be replaced by `0`.
- Computations do not depend on sign of zero.

With `-fsimple=1`, the optimizer is *not* allowed to optimize completely without regard to roundoff or exceptions. In particular, a floating–point computation cannot be replaced by one that produces different results with rounding modes held constant at run time. `-fast` implies `-fsimple=1`.

`-fsimple=2`

Permit aggressive floating point optimizations that may cause many programs to produce different numeric results due to changes in rounding.

For example, in a given loop, permit the optimizer to replace all computations of `x/y` with `x*z`, where `z=1/y`, `x/y` is guaranteed to be evaluated at least once in the loop, and the values of `y` and `z` are known to have constant values during execution of the loop.

Even with `–fsimple=2`, the optimizer still is not permitted to introduce a floating point exception in a program that otherwise produces none.

`–fstore` Force precision of floating–point expressions.

♦ **SPARC: – PPC: – Intel:77**

Use the precision of destination variable to determine the precision of the right-hand-side expression on assignment statements. The default is `–fstore`. (The `–fast` option sets `–nofstore` to disable this option.)

`–ftrap=`*t* Set floating–point trapping mode in effect at startup.

♦ **SPARC: 77/90 PPC:77 Intel:77**

t is a comma–separated list that consists of one or more of the following:

`%all`, `%none`, `common`, [`no%`]`invalid`, [`no%`]`overflow`, [`no%`]`underflow`, [`no%`]`division`, [`no%`]`inexact`.

The default is `–ftrap=%none`. Where the `%` is shown, it is a required character.

This option sets the IEEE 754 trapping modes that are established at program initialization. Processing is left–to–right. The common exceptions, by definition, are invalid, division by zero, and overflow. For example: `–ftrap=overflow`.

Example: `–ftrap=%all,no%inexact` means set all traps, except `inexact`.

The meanings for `–ftrap=`*t* are the same as for `ieee_flags()`, except that:

- `%all` turns on all the trapping modes.
- `%none`, the default, turns off all trapping modes.
- A `no%` prefix turns off that specific trapping mode.

To be effective, compile the main program with this option.

For further information, see the *Floating-Point* chapter in the *Fortran Programmer's Guide*.

-G Build a dynamic shared library instead of an executable file.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Direct the linker to build a *shared dynamic* library. Without –G, the linker builds an executable file. With –G, it builds a dynamic library. Use -o with -G to specify the name of the file to be written. See the *Fortran Programmer's Guide* chapter *Libraries* for details.

-g Compile for debugging.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Produce additional symbol table information for the debugging with dbx(1) and the Sun WorkShop debugging utility.

Although a some debugging is possible without specifying –g, the full capabilities of dbx and debugger are only available to those compilation units compiled with –g.

Some capabilities of other options specified along with –g may be limited. The –g option suppresses the automatic inlining usually obtained with –O4, but it does not suppress –O*n* optimizations.

-g cancels any parallelization option (-autopar, -explicitpar, -parallel) as well as -depend and -reduction. Debugging is facilitated by specifying -g without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the dbx documentation for details.

For *Intel and PowerPC*: –g is ignored when specified with a –O*n* option or –fast.

For *SPARC*: The –g option makes –xildon the default incremental linker option (see "–xildon" on page 91). That is, with –g, the compiler default behavior is to automatically invoke ild in place of ld, unless the –G option is present, or any source file is named on the command line.

-h*nm* Specify the name of the generated dynamic shared library.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This option is passed on to the linker. For details, see the *Linker and Libraries Guide*, and the *Fortran Programmer's Guide* chapter *Libraries*.

The -h*nm* option records the name *nm* to the shared dynamic library being created as the internal name of the library. A space between -h and *nm* is optional. In general, *nm* must be the same as what follows the -o. Use of this option is meaningless without also specifying –G.

Without the -h*nm* option, no internal name is recorded in the library file.

If the library has an internal name, whenever an executable program referencing the library is run the runtime linker will search for a library with the same internal name in any path the linker is searching. With an internal name specified, searching for the library at runtime linking is more flexible. This option can also be used to specify *versions* of shared libraries.

If there is no internal name of a shared library, then the linker uses a specific path for the shared library file instead.

-help Display a summary list of compiler options.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Displays a list of option summaries and indicates how to send feedback comments to Sun. See also –xhelp=*h* on page 91.

-I*dir* Add *dir* to the INCLUDE file search path.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Insert the directory *dir* at the start of the INCLUDE file search path. No space is allowed between -I and *dir*. Invalid directories are ignored with no warning message.

The *include file search path* is the list of directories searched for INCLUDE files: file names appearing on preprocessor #include directives, or Fortran INCLUDE statements.

Example: Search for INCLUDE files in /usr/app/include:

```
demo% f77 -I/usr/app/include growth.F
```

Multiple –I*dir* options may appear on the command line. Each adds to the top of the search path list (first path searched).

The search order for relative path on INCLUDE or #include is:

1. **The directory that contains the source file**
2. **The directories that are named in the** –I*dir* **options**
3. **The directories in the default list**

The default list for –I*dir* depends on the installation directory for the compiler. In a standard install, compiler software packages reside in the /opt directory; however, systems administrators may decide to install packages in other locations. If an environment variable, INSTALL_HOME say, points at the installation path (e.g. /opt, or /some/place), the default search paths for INCLUDE files are:

```
for f77: $INSTALL_HOME/SUNWspro/SC4.2/include/f77  /usr/include
for f90: $INSTALL_HOME/SUNWspro/SC4.2/include/f90  /usr/include
```

–i2 Set the default integer size to two bytes.

♦ **SPARC:77 PPC:77 Intel:77**

Set the default size to 2 bytes for integer and logical constants and variables declared without an explicit size. (INTEGER*n Y still declares Y to be *n* bytes regardless of the –i2.) This option may degrade performance. It is generally recommended to declare specific variables INTEGER*2 rather than use –i2.

–i4 Set the default integer size to four bytes.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Set the default size to 4 bytes for integer and logical constants and variables declared without an explicit size. (INTEGER*n Y still declares Y to be *n* bytes regardless of the –i4.).

Although 4 bytes *is* the default size for `INTEGER` and `LOGICAL`, this option can be used for overriding settings made by options like –`dbl` and –`r8`, which set these defaults to 8:

```
demo% f77 -dbl -i4 *.f
Command line warning: -i4 overrides integer part of -dbl
...
```

`-inline=`*f1[,...fn]* Inline specified routines.

♦ **SPARC:77 PPC:77 Intel:77**

Request that the optimizer inline the user–written routines named in the *f1,...,fn* list. Inlining is an optimization technique whereby the compiler effectively replaces a subprogram reference such as a `CALL` or function call with the actual subprogram code itself. Inlining often provides the optimizer more opportunities to produce efficient code.

The list is a comma–separated list of functions and subroutines.

Example: Inline the routines `xbar`, `zbar`, `vpoint`:

```
demo% f77 -O3 -inline=xbar,zbar,vpoint *.f
```

Following are the restrictions; no warnings are issued:

- Optimization must be `-O3` or greater *(SPARC)*.
- The source for the routine must be in the file being compiled, unless `-xcrossfile` is also specified.
- The compiler determines if actual inlining is profitable and safe.

Note that with `-O4`, `f77` normally tries to inline all appropriate user–written subroutines and functions. Adding `-inline` with `-O4` actually degrades performance by restricting the optimizer's inlining to only those routines in the list.

–Kpic Synonym for –pic.

♦ **SPARC:77 PPC:77 Intel:77**

–KPIC Synonym for –PIC.

♦ **SPARC:77 PPC:77 Intel:77**

–L*dir* Add *dir* to list of directories to search for libraries.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Add *dir* at the *start* of the list of object–library search directories. A space between –L and *dir* is optional. This option is passed to the linker. See also –l*x* on page 62.

While building the executable file, ld(1) searches *dir* for archive libraries (.a files) and shared libraries (.so files). ld searches *dir* before searching the default directories. (See the *Fortran Programmer's Guide* chapter *Libraries* for information on library search order.) For the relative order between LD_LIBRARY_PATH and –L*dir*, see ld(1).

Example: Use -L*dir* to specify a library search directory:

```
demo% f77 -Ldir1 any.f
```

Example: Use -L*dir* again to add more directories:

```
demo% f77 -Ldir1  -Ldir2  any.f
```

Note – Specifying /usr/lib or /usr/ccs/lib with –L*dir* may prevent linking the unbundled libm. These directories are searched by default.

-l*x* Add library lib*x*.a to linker's list of search libraries.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Pass -l*x* to the linker to specify additional libraries for ld to search for unresolved references. ld links with object library lib*x*. If shared library lib*x*.so is available (and –Bstatic or -dn are not specified), ld uses it, otherwise, ld uses static library lib*x*.a. If it uses a shared library, the name is built in to a.out. No space is allowed between -l and *x* character strings.

Example: Link with the library libV77:

```
demo% f77 any.f -lV77
```

Use -l*x* again to link with more libraries.

Example: Link with the libraries liby and libz:

```
demo% f77 any.f -ly -lz
```

See also the *Libraries* chapter in the *Fortran Programmer's Guide* for information on library search paths and search order.

-libmil Inline selected libm library routines for optimization.

♦ **SPARC:77 PPC:77 Intel:77**

There are inline templates for some of the libm library routines. This option selects those inline templates that produce the fastest executables for the floating–point options and platform currently being used. The routines include the following:

d_infinity, d_max_normal, d_max_subnormal, d_min_normal, d_min_subnormal, d_quiet_nan, d_signaling_nan, d_sqrt, ir_finite, ir_fp_class, ir_isinf, ir_isnan, ir_isnormal, ir_issubnormal, ir_iszero, ir_signbit, r_copysign, r_fabs, r_hypot, r_infinity, r_max_normal, r_max_subnormal, r_min_normal, r_min_subnormal, r_quiet_nan, r_signalling_nan, r_sqrt

This list of routines may change with subsequent compiler releases. For more information, see the man pages libm_single(3F) and libm_double(3F)

-loopinfo Show parallelization results.

♦ **SPARC:77 PPC: – Intel: –**

Show which loops parallelized and which did not with the `-parallel`, `–autopar`, or `–explicitpar` options.

`–loopinfo` generates a list of messages on standard error:

```
demo% f77 –o shalow –fast –parallel –loopinfo shalow.f
shalow.f:
 MAIN shalow:
    inital:
    calc1:
...etc
"shalow.f", line 78: not parallelized, call may be unsafe
"shalow.f", line 172: PARALLELIZED
"shalow.f", line 173: not parallelized, not profitable
"shalow.f", line 181: PARALLELIZED, fused
"shalow.f", line 182: not parallelized, not profitable
"shalow.f", line 226: PARALLELIZED, and serial version generated
"shalow.f", line 227: not parallelized, not profitable
...etc
```

Use the `error`(1) utility to merge this list with the source file to produce an annotated source listing with each loop tagged as parallelized or not.

Example: `–loopinfo`, in `sh`, pass standard error to the `error` utility:

```
demo$ f77 –autopar –loopinfo any.f 2>&1 | error options
```

Be aware that `error` rewrites the input source file. For details on `error`, see the `error` man page and the *Fortran Programmer's Guide* chapter on debugging.

–M*dir* Add *dir* to directories searched for Fortran 90 modules.

♦ **SPARC:90 PPC: – Intel: –**

Add *dir* to the list of directories to be searched for module files. No space appears between the `–M` and *dir*.

The directories listed with –M are searched after the current directory. Compiling a source file containing a module generates a .M module file in addition to the .o file. See Appendix C, “Module Files” on page 149 for more information about modules in Fortran 90.

–misalign Allow misaligned data.

♦ **SPARC:77 PPC:77 Intel: –**

The –misalign option permits misaligned data in memory that would otherwise produce an error. Particular uses of COMMON and EQUIVALENCE statements cause data to be misaligned. Using this option may degrade performance; recoding to eliminate the cause of data misalignment is recommended instead.

Example: The following creates misaligned variables.

```
    INTEGER*2   I(4)
    REAL        R1, R2
    EQUIVALENCE (R1, I(1)), (R2, I(2))
    END
```

Two-byte elements of integer array I are equivalenced to the 4-byte reals, contradicting the natural alignment of elements in an INTEGER*2 array . The following error message is issued:

```
"misalign.f", line 4: Error: bad alignment for "r2" forced by
equivalence
```

Compiling with –misalign eliminates the compilation error, but sub-optimal code is generated. If you compile and link in separate steps, compiling with the -misalign option requires the option on the link step as well.

–mp={sun|cray} Select the style for parallelization directives.

♦ **SPARC:77 PPC: – Intel: –**

The default without specifying -mp is sun. Do not combine use in a single application.

`-mp=sun`: Accept only the Sun–style directives: `C$PAR` or `!$PAR` prefix.

`-mp=cray`: Accept only the Cray–style directives: `CMIC$` or `!MIC$` prefix.

See the *Fortran Programmer's Guide* chapter on *Parallelization* for details.

`-mt` Require multithread-safe libraries.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Require linking to multithread–safe libraries. If you do your own low–level thread management (e.g. calls to the `libthread` library), compiling with `–mt` prevents conflicts.

Use `–mt` if you mix C and Fortran, and you manage multithread C coding using the `libthread` primitives. Before you use your own multi–threaded coding, read the Solaris manual, *Multithreaded Programming Guide*.

The equivalent of `–mt` is included automatically with the `-autopar`, `-explicitpar`, or `-parallel` options.

Note the following:

- A function subprogram that does I/O should not itself be referenced as part of an I/O statement. Such *recursive* I/O may cause the program to deadlock with `–mt`.
- In general, do *not* compile your own multi–threaded coding with `-autopar`, `-explicitpar`, or `-parallel`. The compiler's generated calls to the threads library primitives any the programs own calls may conflict, causing unexpected results.
- On a single–processor system, performance may be degraded with the `–mt` option.

`-native` Optimize performance for the host system.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This option is a synonym for `–xtarget=native`. This is one of the options included in the expansion of the `–fast` option .

–noautopar Disable automatic parallelization.

♦ **SPARC:77 PPC: – Intel: –**

Disables automatic parallelization invoked by `-autopar` earlier on the command line.

–nodepend Cancel `–depend` in command line.

♦ **SPARC:77 PPC: – Intel: –**

Cancel any `–depend` appearing earlier on the command line.

–noexplicitpar Disable explicit parallelization.

♦ **SPARC:77 PPC: – Intel: –**

Disables explicit parallelization invoked by `-explicitpar` earlier on the command line.

–nofstore Disable forcing precision of expression.

♦ **SPARC: – PPC: – Intel:77**

Disables forcing precision of expressions in assignment statements invoked by `-fstore` earlier on the command line (*Intel only*). `-nofstore` is invoked if `-fast` is specified.

–nolib Disable linking with system libraries.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Do *not* automatically link with *any* system or language library; that is do *not* pass any default `–l`*x* options on to `ld`. The normal behavior is to link system libraries into the executables automatically, without the user specifying them on the command line.

The `–nolib` option makes it easier to link one of these libraries statically. The system and language libraries are required for final execution. It is your responsibility to link them in manually. This option provides you with complete control.

For example, consider a program linked dynamically with `libF77` that fails on a remote system because has no `libF77`. With this option you can link the library into your program statically.

Link `libF77` statically and link `libc` dynamically with `f77`:

```
demo% f77 -nolib any.f -Bstatic -lF77 -Bdynamic -lm -lc
```

Link `libm` statically and `libc` dynamically with `f90`:

```
demo% f90 -nolib any.f90 -lf90 -Bstatic -lm -Bdynamic -lc
```

`libf90` is always linked statically.

The order for the `-l`*x* options is important. Follow the order shown in the examples.

`-nolibmil`

Cancel `-libmil` on command line.

♦ **SPARC:77 PPC:77 Intel:77**

Use this option *after* the `-fast` option to disable inlining of `libm` math routines:

```
demo% f77 -fast -nolibmil …
```

`-noqueue`

Disable license queueing.

♦ **SPARC:77 PPC:77 Intel:77**

With this option, if no software license is available to run the compiler, it returns without queueing your request and without compiling. A nonzero environment status is returned for testing in `make` files.

–noreduction Cancel `-reduction` on command line.

♦ **SPARC: 77/90 PPC: – Intel: –**

`-reduction` is used with other parallelization options. This option cancels `-reduction`.

–norunpath Do not build a runtime shared library search path into the executable.

♦ **SPARC: 77/90 PPC:77 Intel:77**

The compiler normally builds into an executable a path that tells the runtime linker where to find the shared libraries it will need. The path is installation dependent. The `-norunpath` option prevents that path from being built in to the executable.

This option is helpful when libraries have been installed in some nonstandard location, and you do not wish to make the loader search down those paths when the executable is run at another site. Compare with –R*paths*.

See the *Fortran Programmer's Guide* chapter on *Libraries* for more information.

–O[n] Specify optimization level.

♦ **SPARC: 77/90 PPC:77 Intel:77**

n can be 1, 2, 3, 4, or 5. No space is allowed between –O and *n*.

If `-O`[*n*] is not specified, only a very basic level of optimization limited to local common subexpression elimination and dead code analysis is performed. A program's performance may be significantly improved when compiled with an optimization level than without optimization. Use of –O (which implies –O3) or –`fast` (which implies –O4) is recommended for most programs.

Each –O*n* level includes the optimizations performed at the levels below it. Generally, the higher the level of optimization a program is compiled with, the better runtime performance obtained. However, higher optimization levels may result in increased compilation time and larger executable files.

Debugging with –g does not suppress –O*n*, but –O*n* limits –g in certain ways; this is described on page 57.

The -O3 and -O4 options reduce the utility of debugging such that you cannot display variables from dbx, but you can still use the dbx where command to get a symbolic traceback.

For SPARC: If the optimizer runs out of memory, it attempts to proceed over again at a lower level of optimization, resuming compilation of subsequent routines at the original level.

For details on optimization, see the *Fortran Programmer's Guide* chapters *Performance Profiling,* and *Performance and Optimization*.

–O This is equivalent to –O3.

–O1 Provides a minimum of statement-level optimizations.

Use if higher levels result in excessive compilation time, or exceed available swap space.

–O2 Enables basic block level optimizations.

This level usually gives the smallest code size. (See also -xspace.)

–O3 is preferred over -O2 unless -O3 results in unreasonably long compilation time, exceeds swap space, or generates excessively large executable files.

–O3 Adds loop unrolling and global optimizations at the function level.

Usually –O3 generates larger executable files.

–O4 Adds automatic inlining of routines contained in the same file.

Usually –O4 generates larger executable files due to inlining. (*f77 only; for f90, –O4 is equivalent to –O3*)

The –g option suppresses the –O4 automatic inlining described above. -xcrossfile increases the scope of inlining with -O4.

-O5 Attempt aggressive optimizations. (*f77 only*).

Suitable only for that small fraction of a program that uses the largest fraction of compute time. `-O5`'s optimization algorithms take more compilation time, and may also degrade performance when applied to too large a fraction of the source program.

Optimization at this level is more likely to improve performance if done with profile feedback. See `–xprofile=`*p*.

-o *nm* Specify the name of the executable file to be written.

♦ **SPARC: 77/90 PPC:77 Intel:77**

There must be a blank between `–o` and *nm*. Without this option, the default is to write the executable file to `a.out`. When used with `-c`, `-o` specifies the target `.o` object file; with `-G` it specifies the target `.so` library file.

-oldldo Select "old" list–directed output style.

♦ **SPARC:77 PPC:77 Intel:77**

Omit the blank that starts each record for list–directed output. This is a change from `f77` releases 1.4 and earlier. The default behavior is to provide that blank, since the Fortran Standard requires it. Note also the `FORM='PRINT'` option of `OPEN`. You can compile parts of a program with `-oldldo` and other parts without it.

-onetrip Enable one trip `DO` loops.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Compile `DO` loops such that they are executed at least once. `DO` loops in standard Fortran are not performed at all if the upper limit is smaller than the lower limit, unlike some legacy implementations of Fortran.

-p Compile for profiling with the `prof` profiler.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Prepare object files for profiling, see `prof` (1). If you compile and link in separate steps, and if you compile with the `-p` option, then be sure to link with the `-p` option. `-p` with `prof` is provided mostly for compatibility with older systems. `-pg` profiling with `gprof` is possibly a better alternative. See the *Fortran Programmer's Guide* chapter on *Performance Profiling* for details.

-pad[=*p*] Insert padding for efficient use of cache.

♦ **SPARC:77 PPC: – Intel: –**

This option inserts padding between arrays or character variables if they are static local and not initialized, or in common blocks. The extra padding positions the data to make better use of cache. In either case, the arrays or character variables can not be equivalenced.

For `-pad`[=*p*], if *p* is present, it must be either (or both):

`local`	Put padding between adjacent *local* variables
`common`	Put padding between variables in common blocks

Defaults for `-pad`:

- Without the `-pad`[=*p*] option, the compiler does no padding.
- With `-pad`, but without the =*p*, the compiler does both local and common padding.

The following are equivalent:

- `f77 -pad` *any*`.f`
- `f77 -pad=local,common` *any*`.f`
- `f77 -pad=common,local` *any*`.f`
- `f77 -pad=local -pad=common` *any*`.f`
- `f77 -pad=common -pad=local` *any*`.f`

The `-pad`[=*p*] option applies to items that satisfy the following criteria:

- The items are arrays or character variables
- The items are static local or in common blocks

For a definition of local or static variables, see –`stackvar`, page 79.

Restrictions on `-pad=common`:

- Neither the arrays nor the character strings are equivalenced
- If `–pad=common` is specified for compiling a file that references a common block, it must be specified when compiling all files that reference that common block. The option changes the spacing of variables within the common block. If one program unit is compiled with the option and another is not, references to what should be the same location within the common block might reference different locations.
- If `–pad=common` is specified, the declarations of common block variables in different program units must be the same except for the names of the variables.The amount of padding inserted between variables in a common block depends on the declarations of those variables. If the variables differ in size or rank in different program units, even within the same file, the locations of the variables might not be the same.
- If `–pad=common` is specified, `EQUIVALENCE` declarations involving common block variables are flagged as an error.

`–parallel` Parallelize loops with: `-autopar, –explicitpar, -depend`

♦ **SPARC: 77/90 PPC: – Intel: –**

Parallelize loops chosen automatically by the compiler *and* explicitly specified by user supplied directives. Optimization level is automatically raised to `–O3` if it is lower.

`-g` cancels `-parallel`. Debugging is facilitated by specifying `-g` without any optimization or parallelization options since not all debugging features are available when these options are invoked. See the `dbx` documentation for details.

To improve performance, also specify the `–stackvar` option when using any of the parallelization options, including `–autopar`.

Avoid `-parallel` if you do your own thread management. See the discussion of `–mt` on page 65.

Parallelization options like `–parallel` are intended to produce executables programs to be run on multiprocessor systems. On a single–processor system, parallelization generally degrades performance.

If you compile and link in separate steps, if `-parallel` appears on the compile command it must also appear on the `ld` link command.

See the *Fortran Programmer's Guide* chapter *Parallelization* for further information.

–pentium Compile for Intel Pentium.

♦ **SPARC: – PPC: – Intel:77**

Generate code that exploits features available on Intel Pentium compatible computers. The default on Intel is `–386`. Code compiled with `–pentium` does run on 80386 and 80486 hardware, but it may be slower. Use of the option `-xtarget=pentium` is preferred over `-pentium`.

–pg Compile for profiling with the `gprof` profiler.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Compile self-profiling code in the manner of `–p`, but invoke a runtime recording mechanism that keeps more extensive statistics and produces a `gmon.out` file when the program terminates normally. Generate an execution profile by running `gprof` (1).

Library options must be *after* the `.f` and `.o` files (`–pg` libraries are static).

If you compile and link in separate steps, and you compile with `-pg`, then be sure to link with `-pg`.

–pic Compile position–independent code for shared library.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This kind of code is for dynamic shared libraries. Each reference to a global datum is generated as a dereference of a pointer in the global offset table. Each function call is generated in program–counter–relative addressing mode through a procedure linkage table.

- The size of the global offset table is limited to 8Kb on SPARC, 64Kb on PowerPC. The size of the table is unlimited on Intel.
- Do not mix `–pic` and `–PIC`.

There are two nominal performance costs with `–pic` and `–PIC`:

- A routine compiled with either –pic or –PIC executes a few extra instructions upon entry to set a register to point at the global offset table used for accessing a shared library's global or static variables.
- Each access to a global or static variable involves an extra indirect memory reference through the global offset table. If the compile is done with –PIC, there are two additional instructions per global and static memory reference.

When considering the above costs, remember that the use of -pic and –PIC can significantly reduce system memory requirements, due to the effect of library code sharing. Every page of code in a shared library compiled –pic or –PIC can be shared by every process that uses the library. If a page of code in a shared library contains even a single non-pic (that is, absolute) memory reference, the page becomes nonsharable, and a copy of the page must be created each time a program using the library is executed.

The easiest way to tell whether or not a .o file has been compiled with -pic or –PIC is with the nm command:

```
% nm file.o | grep _GLOBAL_OFFSET_TABLE_
U _GLOBAL_OFFSET_TABLE_
```

A .o file containing position-independent code contains an unresolved external reference to _GLOBAL_OFFSET_TABLE_, as indicated by the letter U.

To determine whether to use –pic or –PIC, use nm to identify the number of distinct global and static variables used or defined in the library. If the size of _GLOBAL_OFFSET_TABLE_ is under 8,192 bytes, you can use -pic. Otherwise, you must use –PIC.

–PIC Compile position-independent code, but with 32-bit addresses.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This option is similar to -pic, but it allows the global offset table to span the range of 32-bit addresses. Use it in those rare cases where there are too many global data objects for -pic. Do not mix –pic and -PIC.

On PowerPC, the size of the global offset table is unlimited. On Intel, –PIC is identical to –pic.

-Qoption *pr ls* Pass options to compilation phase *pr*.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Pass the suboption list *ls* to the compilation phase *pr*. There must be blanks separating `Qoption`, *pr*, and *ls*. The `Q` can be uppercase or lowercase. The list is a comma–delimited list of suboptions, with no blanks within the list. Each suboption must be appropriate for that program phase, and can begin with a minus sign.

pr can be one of the following: `as`, `cg`, `cpp`, `fbe`, `fpp`, `f77pass0`, `f77pass1`, `iropt`, `ld`, or `ratfor`. This option is provided primarily for internal debugging.

Example: Pass the `–s` and `–m` options to the linker `ld`:

```
demo% f77 -Qoption ld -s,-m src.f
```

-qp Synonym for `-p`.

♦ **SPARC: 77/90 PPC:77 Intel:77**

-R *ls* Build dynamic library search paths into the executable file.

♦ **SPARC: 77/90 PPC:77 Intel:77**

With this option, the linker, `ld`(1), stores a list of dynamic library search paths into the executable file.

ls is a colon–separated list of directories for library search paths. The blank between `–R` and *ls* is optional.

Multiple instances of this option are concatenated together, with each list separated by a colon.

The list is used at runtime by the runtime linker, `ld.so`. At runtime, dynamic libraries in the listed paths are scanned to satisfy any unresolved references.

Use this option to let users run shippable executables without a special path option to find needed dynamic libraries.

Building an executable file using -R*paths* adds directory paths to a default path that is always searched last:

Standard Install: `/opt/SUNWspro/lib`

Non-standard Install: *installpath*`/lib`

For more information, see the *Libraries* chapter in the *Fortran Programmer's Guide*, and the Solaris *Linker and Libraries Guide*.

-r8

Double default byte size for `REAL`, `INTEGER`, `DOUBLE` and `COMPLEX`.

♦ **SPARC:77 PPC:77 Intel:77**

`-r8` promotes the default byte size for `REAL`, `INTEGER`, `DOUBLE`, and `COMPLEX` variables declared *without an explicit byte size* as follows:

Table 3-17 Default Data Sizes and –`r8` (Bytes)

	Without –r8 option	With –r8 option		
Data Type	**default**	**SPARC**	**Intel**	**PowerPC**
`INTEGER`	4	8	8	8
`REAL`	4	8	8	8
`DOUBLE`	8	16	8	16

This option applies to variables, parameters, constants, and functions.

Also, `LOGICAL` is treated as `INTEGER`, `COMPLEX` as two `REAL`s, and `DOUBLE COMPLEX` as two `DOUBLE`s.

Compare `-r8` with `-dbl`:

`-dbl` and –`r8` can be expressed in terms of the more general –`xtypemap=` option:

On SPARC and PowerPC:		
`-dbl`	*same as:*	`-xtypemap=real:64,double:128,integer:64`
`-r8`	*same as:*	`-xtypemap=real:64,double:128,integer:mixed`
On Intel:		
`-dbl`	*same as:*	`-xtypemap=real:64,double:64,integer:64`
`-r8`	*same as:*	`-xtypemap=real:64,double:64,integer:mixed`

For all of the floating point data types, `–dbl` works the same as `–r8`; using both `-r8` and `-dbl` produces the same results as using only `-dbl`.

- For `INTEGER` and `LOGICAL` data types, `–dbl` is different from `-r8`:
  - `–dbl` allocates 8 bytes, and does 8–byte arithmetic
  - `–r8` allocates 8 bytes, and does only 4–byte arithmetic (“mixed”)

In general, if you compile *one* subprogram with `–r8`, then be sure to compile *all* subprograms of that program with `–r8`. This also important with programs communicating through unformatted I/O files — if one program is compiled with `–r8`, then the other program must be similarly compiled. Be also aware that this option alters the default data size of function names, including calls to library functions, unless the function name is typed explicitly with a data size.

The impact on runtime performance may be great. With `–r8`, an expression like `float = 15.0d0*float` is evaluated in quadruple precision due to the declaration of the constant.

If you select both `–r8` and `–i2`, the results are unpredictable.

`–reduction`

Recognize reduction operations in loops.

♦ **SPARC: 77/90 PPC: – Intel: –**

Analyze loops for reduction operations during automatic parallelization. There is potential for roundoff error with the reduction.

A loop that transforms the elements of an array into a single scalar value is called a *reduction operation*. For example, summing the elements of a vector is a typical reduction operation. Although these operations violate the criteria for parallelizability, the compiler can recognize them and parallelize them as special cases when `-reduction` is specified. See the *Fortran Programmer’s Guide* chapter *Parallelization* for information on reduction operations recognized by the compilers.

This option applies only with the automatic parallelization options `–autopar` or `–parallel`. It is ignored otherwise. Explicitly parallelized loops are not analyzed for reduction operations.

Example: Automatically parallelize with *reduction*:

```
demo% f77 -parallel -reduction any.f
```

-S Compile and only generate assembly code.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Compile the named programs and leave the assembly–language output on corresponding files suffixed with `.s`. No `.o` file is created.

-s Strip the symbol table out of the executable file.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This option makes the executable file smaller and more difficult to reverse engineer. However, this option inhibits debugging with `dbx` or other tools, and overrides `-g`.

-sb Produce table information for the WorkShop source code browser.

♦ **SPARC:77 PPC:77 Intel:77**

See *WorkShop: Getting Started* for more information.

-sbfast Produce *only* source code browser tables.

♦ **SPARC:77 PPC:77 Intel:77**

Produce *only* table information for the WorkShop source code browser and stop. Do not assemble, link, or make object files.

-silent Suppress compiler messages.

♦ **SPARC:77 PPC:77 Intel:77**

Use this option to suppress non-essential messages from the compiler; error and warning messages are still issued. The default is to show file and entry names as they are reached during the compilation.

–stackvar

Force all local variables to be allocated on the memory stack.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Allocate all the *local* variables and arrays in a routine onto the memory stack, unless otherwise specified. This option makes them *automatic*, rather than *static*, and provides more freedom to the optimizer for parallelizing a CALL in a loop.

Use of –stackvar is recommended with any of the parallelization options.

Variables and arrays are local, unless they are:

- Arguments in a SUBROUTINE or FUNCTION statement (already on stack)
- Global items in a COMMON or SAVE, or STATIC statement
- Items initialized in a type statement or DATA statement, such as:
  REAL X/8.0/ or DATA X/8.0/

Initializing a local variable in a DATA statement after an executable reference to that variable is flagged as an error when -stackvar is used:

```
demo% cat stak.f
        real x
        x =  1.
        t = 0.
        print*, t
        data x/3.0/
        print *,x+t
        end
demo% f77 -o stak -stackvar stak.f
stak.f:
 MAIN:
"stak.f", line 5: Error: attempt to initialize an automati
 variable: x
```

Putting large arrays onto the stack with –stackvar can overflow the stack causing segmentation faults. Increasing the stack size may be required.

There are two stacks:

- The whole program has a *main* stack.
- Each thread of a multi–threaded program has a *thread* stack.

The default stack size is about 8 Megabytes for the main stack and 256 KBytes for each thread stack. The `limit` command (with no parameters) shows the current main stack size. If you get a segmentation fault using `–stackvar`, you might try doubling the main stack size at least once.

Example: Show the current *main* stack size:

```
demo% limit
cputime         unlimited
filesize        unlimited
datasize        523256 kbytes
stacksize       8192 kbytes      <---
coredumpsize    unlimited
descriptors     64
memorysize      unlimited
demo%
```

Example: Set the *main* stack size to 64 Megabytes:

```
demo% limit stacksize 65536
```

Example: Set each *thread* stack size to 8 Megabytes:

```
demo% setenv STACKSIZE 8192
```

For further information of the use of `–stackvar` with parallelization, see the *Parallelization* chapter in the *Fortran Programmer's Guide.* See `csh`(1) for details on the `limit` command.

`–stop_status=`*yn* Permit STOP statement to return an integer status value.

♦ **SPARC:77 PPC:77 Intel:77**

yn is either `yes` or `no`. The default is `no`.

With `–stop_status=yes`, a `STOP` statement may contain an integer constant. That value will be passed to the environment as the program terminates:

```
STOP 123
```

The value must be in the range 0 to 255. Larger values are truncated and a run-time message issued. Note that

```
STOP 'stop string'
```

is still accepted and returns a status value of 0 to the environment, although a compiler warning message will be issued.

The environment status variable is `$status` for the C shell `csh`, and `$?` for the Bourne and Korn shells, `sh` and `ksh`.

`-temp=`*dir* Define directory for temporary files.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Set directory for temporary files used by the compiler to be *dir*. No space is allowed within this option string. Without this option, the files are placed in the `/tmp` directory.

`-time` Time each compilation phase.

♦ **SPARC: 77/90 PPC:77 Intel:77**

The time spent and resources used in each compiler pass is displayed.

`-U` Recognize upper and lower case in source files.

♦ **SPARC:77 PPC:77 Intel:77**

Do not treat uppercase letters as equivalent to lowercase. The default is to treat uppercase as lowercase except within character–string constants. With this option, the compiler treats `Delta`, `DELTA`, and `delta` as different symbols.

Portability and mixing Fortran with other languages may require use of `-U`. These are discussed in the *Fortran Programmer's Guide*. (Note that `f90` does not have this option, always treating upper and lower case as equivalent.)

-u Report undeclared variables.

♦ **SPARC:77 PPC:77 Intel:77**

Make the default type for all variables be *undeclared* rather than using Fortran implicit typing. This option warns of undeclared variables, and does not override any `IMPLICIT` statements or explicit *type* statements.

-unroll=*n* Enable unrolling of DO loops where possible.

♦ **SPARC:77 PPC:77 Intel:77**

n is a positive integer. The choices are:

- *n*=1 inhibits all loop unrolling.
- *n*>1 *suggests* to the optimizer that it attempt to unroll loops *n* times.

Loop unrolling generally improves performance, but will increase the size of the executable file. For more information on this and other compiler optimizations, see the *Performance and Optimization* chapter in the *Fortran Programmer's Guide*. See also the discussion of the `UNROLL` directive on page 16.

-V Show name and version of each compiler pass.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This option prints the name and version of each pass as the compiler executes:

```
demo% f77 -o scalc -fast -autopar -V scalc.f
f77: WorkShop Compilers 4.2 dev 01 May 1996 FORTRAN 77 4.2
f77pass1: WorkShop Compilers 4.2 dev 01 May 1996 FORTRAN 77 4.2
scalc.f:
 MAIN scalc:
    initial:
    calc1:
    calc2:
    calc3:
iropt: WorkShop Compilers 4.2 dev 01 May 1996
cg: WorkShop Compilers 4.2 dev 01 May 1996
ld: Software Generation Utilities (SGU) SunOS/ELF (LK-1.4 (S/I))
```

This information will be helpful when discussing problems with Sun service engineers.

-v Verbose mode – show details of each compiler pass.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Like `-V`, shows the name of each pass as the compiler executes, and details the options and environment variables used by the driver.

-vax=*v* Specify choice of VMS Fortran extensions enabled.

♦ **SPARC:77 PPC:77 Intel:77**

v must be a comma-separated list of at least one suboption. Negatives may be constructed by prefixing each suboption keyword by `no%` (as in `no%logical_name`).

The primary options are `-vax=align` and `-vax=misalign`.

`-vax=align` selects all the suboptions without allowing misaligned data. This is the behavior of the `-xl` option prior to `f77` release 3.0.1.

`-vax=misalign` selects all the suboptions and allows misaligned data. This is the behavior of the `-xl` option with `f77` release 3.0.1 and later.

Table 3-18 lists suboptions that can be individually selected.

Table 3-18 `-vax=` Suboptions

-vax=	**Affect**
`bslash`	Allow backslash (‘\’) in character constants.
`debug`	Allow VMS Fortran ‘`D`’ debugging statements.
`logical_name`	Allow VMS Fortran style logical file names.
`oct_const`	Allow double quote character to signify octal constants.
`param`	Allow non-standard form of `PARAMETER` statement.
`rsize`	Allow unformatted record size in words rather than bytes.
`struct_align`	Align structures as in VMS Fortran.

`%all` and `%none` can also be used to select all or none of these suboptions.

Sub- options accumulate from left to right. For example, to enable all but one feature: `-vax=%all,no%rsize`

See also `-xl` and `-misalign`.

-vpara Show verbose parallelization messages.

♦ **SPARC:77 PPC: – Intel: –**

As the compiler analyzes loops explicitly marked for parallelization with directives, it issues a warning message about certain data dependencies it detects; but the loop will still be parallelized.

Example: `-vpara` for verbose parallelization warnings:

```
demo% f77 -explicitpar -vpara any.f
any.f:
 MAIN any:
"any.f", line 11: Warning: the loop may have parallelization
inhibiting reference
```

-w Suppress warning messages.

♦ **SPARC: 77/90 PPC:77 Intel:77**

This option suppresses most warning messages. However, if one option overrides all or part of an option earlier on the command line, you do get a warning.

Example: `–w` still allows some warnings to get through:

```
demo% f77  –w –fast  –silent –O4  any.f
f77: Warning: –O4 overwrites previously set optimization
level of –O3
demo%
```

For `f90`: Individual levels from 0 to 4 can be specified: `–w0` suppresses the least messages while `–w4` suppresses most warning. `–w` is equivalent to `–w0`.

-xa Synonym for `–a`.

♦ **SPARC:77 PPC:77 Intel:77**

-xarch=*a* Specify target architecture instruction set.

♦ **SPARC:77 PPC:77 Intel:77**

Target architectures specified by keyword *a* are:

Table 3-19 `-xarch` Architecture Keywords

On SPARC:	`generic, v7, v8a, v8, v8plus, v8plusa`
On PowerPC:	`generic, ppc, ppc_nofma`
On Intel:	`generic, 386, pentium_pro`

Although this option can be used alone, it is part of the expansion of the `-xtarget` option; it is provided to allow overriding the `-xarch` value implied by a specific `-xtarget` option.

This option limits the instructions generated to those of the specified architecture, and *allows* the specified set of instructions. It does not guarantee that target-specific instructions are used.

If this option is used with optimization, the appropriate choice can provide good performance of the executable on the specified architecture. An inappropriate choice can result in serious degradation of performance.

For SPARC:

SPARC architectures `v7, v8`, and `v8a` are all binary compatible. `v8plus` and `v8plusa` are binary compatible with each other and forward, but not backward.

For any particular choice, the generated executable may run much more slowly on earlier architectures.

`generic`: Get good performance on most systems.

This is the default. This option uses the best instruction set for good performance on most processors without major performance degradation on any of them. With each new release, the definition of "best" instruction set may be adjusted, if appropriate.

`v7`: Limit the instruction set to V7 architecture.

This option uses the best instruction set for good performance on the V7 architecture, but without the quad–precision floating–point instructions.

This is equivalent to using the best instruction set for good performance on the V8 architecture, but *without*:

– The quad–precision floating–point instructions

– The integer `mul` and `div` instructions

– The `fsmuld` instruction

Examples: SPARCstation 1, SPARCstation 2

`v8a`: Limit the instruction set to the V8a version of the V8 architecture.

By definition, V8a means the V8 architecture, but without:

– The quad–precision floating–point instructions

– The `fsmuld` instruction

This option uses the best instruction set for good performance on the V8a architecture.

Example: Any machine based on the MicroSPARC I chip architecture

`v8`: Limit the instruction set to V8 architecture.

This option uses the best instruction set for good performance on the V8 architecture, but without quad–precision floating–point instructions.

Example: SPARCstation 10

`v8plus`: Limit instructions to the V8plus version of the V9 architecture.

By definition, V8plus means the V9 architecture, except:

– Without the quad–precision floating–point instructions

– Limited to the 32–bit subset defined by the V8plus specification

– Without the VIS instructions

This option uses the best instruction set for good performance on the V8plus chip architecture. In V8plus, a system with the 64–bit registers of V9 runs in 32–bit addressing mode, but the upper 32 bits of the `i`*x* and `l`*x* registers must not affect program results.

Example: Any machine based on the UltraSPARC chip architecture

Use of `–xarch=v8plus` causes the `.o` file to be marked as a V8+ binary. Such binaries will not run on a V7 or V8 machine.

v8plusa: Limit the instruction set to the V8plusa architecture variation.

By definition, V8plusa, means the V8plus architecture, plus:

– The UltraSPARC–specific instructions

– The VIS instructions

This option uses the best instruction set for good performance on the UltraSPARC™ architecture, but limited to the 32–bit subset defined by the V8plus specification.

Example: Any machine based on the UltraSPARC chip architecture

Use of `–xarch=v8plusa` also causes the `.o` file to be marked as a Sun–specific V8plus binary. Such binaries will not run on a V7 or V8 machine.

For PowerPC:

`generic` and `ppc` are equivalent in this release and direct the compiler to produce code for the PowerPC 603 and 604 instruction set.

`ppc_nofma` is the same as `ppc` except that the compiler will not issue the "fused multiply-add" instruction.

For Intel:

`generic` and `386` are equivalent in this release.

`pentium_pro` directs the compiler to issue instructions for the Intel PentiumPro chip.

–xautopar Synonym for `–autopar`.

♦ **SPARC:77 PPC: – Intel: –**

–xcache=*c* Define cache properties for the optimizer.

♦ **SPARC:77 PPC: – Intel: –**

c must be one of the following:

- `generic`
- *s1*`/`*l1*`/`*a1*
- *s1*`/`*l1*`/`*a1*`:`*s2*`/`*l2*`/`*a2*
- *s1*`/`*l1*`/`*a1*`:`*s2*`/`*l2*`/`*a2*`:`*s3*`/`*l3*`/`*a3*

The *si*/*li*/*ai* are defined as follows:

- *si* The size of the data cache at level *i*, in kilobytes
- *li* The line size of the data cache at level *i*, in bytes
- *ai* The associativity of the data cache at level *i*

This option specifies the cache properties that the optimizer can use. It does not guarantee that any particular cache property is used.

Although this option can be used alone, it is part of the expansion of the `-xtarget` option; it is provided to allow overriding an `-xcache` value impled by a specific `-xtarget` option.

Example: `-xcache=16/32/4:1024/32/1` specifies the following:

Table 3-20 `-xcache` Values

Value	Meaning
`generic`	Define the cache properties for good performance on most SPARC processors without any major performance degradation. This is the default.
s1/*l1*/*a1*	Define level 1 cache properties.
s1/*l1*/*a1*:*s2*/*l2*/*a2*	Define levels 1 and 2 cache properties.
s1/*l1*/*a1*:*s2*/*l2*/*a2*:*s3*/*l3*/*a3*	Define levels 1, 2, and 3 cache properties

Level 1 cache has:
16K bytes
32 bytes line size
4–way associativity

Level 2 cache has:
1024K bytes
32 bytes line size
Direct mapping associativity

`-xcg89` Synonym for `-cg89`.

♦ **SPARC:77 PPC: – Intel: –**

`-xcg92` Synonym for `-cg92`.

♦ **SPARC:77 PPC: – Intel: –**

-xchip=*c* Specify target processor for the optimizer.

♦ **SPARC:77 PPC:77 Intel:77**

This option specifies timing properties by specifying the target processor.

Although this option can be used alone, it is part of the expansion of the `-xtarget` option; it is provided to allow overriding a `-xchip` value implied by the a specific `-xtarget` option.

Some effects of `-xchip=`*c* are:

- Iinstruction scheduling
- The way branches are compiled
- The instructions to use in cases where semantically equivalent alternatives are available

Table 3-21 lists the valid `-xchip` values:

Table 3-21 Valid `-xchip` Values

	Value	Optimize for:
SPARC:	`generic`	good performance on most SPARC processors.
	`old`	pre–SuperSPARC™ processors.
	`super`	the SuperSPARC chip.
	`super2`	the SuperSPARC II chip.
	`micro`	the MicroSPARC™ chip.
	`micro2`	the MicroSPARC II chip.
	`hyper`	the HyperSPARC™ chip.
	`hyper2`	the HyperSPARC II chip.
	`powerup`	the Weitek® PowerUp™ chip.
	`ultra`	the UltraSPARC chip.
PowerPC:	`generic`	good performance on most PowerPC processors
	`603`	PowerPC 603
	`604`	PowerPC 604
Intel:	`generic`	good performance on most Intel processors
	`386`	Intel 386

Table 3-21 Valid –xchip Values

Value	Optimize for:
486	Intel 486
pentium	Intel Pentium
pentium_pro	Intel Pentium Pro

–xcrossfile Enable optimization and inlining across source files.

♦ **SPARC:77 PPC: – Intel: –**

Normally, the scope of the compiler's analysis is limited to each separate file on the command line. For example, -O4's automatic inlining is limited to subprograms defined and referenced within the same source file.

With -xcrossfile, the compiler analyzes all the files named on the command line as if they had been concatenated into a single source file.

-xcrossfile is only effective when used with -O4 or -O5.

Cross-file inlining creates a possible source file interdependence that would not normally be there. If any file in a set of files compiled together with -xcrossfile is changed, then all files must be recompiled to insure that the new code is properly inlined. See the discussion of inlining on page 60.

–xdepend Synonym for –depend.

♦ **SPARC:77 PPC: – Intel: –**

–xexplicitpar Synonym for –explicitpar.

♦ **SPARC:77 PPC: – Intel: –**

–xF Allow function–level reordering by WorkShop Analyzer.

♦ **SPARC:77 PPC:77 Intel:77**

Allow the reordering of functions (subprograms) in the core image using the compiler, the Analyzer and the linker. If you compile with the -xF option, then run the Analyzer, you can generate a map file that optimizes the ordering of the functions in memory depending on how they are used together. A

subsequent link to build the executable file can be directed to use that map by using the linker `-Mmapfile` option. It places each function from the executable file into a separate section.

Reordering the subprograms in memory is useful only when the application text page fault time is consuming a large percentage of the application time. Otherwise, reordering may not improve the overall performance of the application. The Analyzer is part of the Sun WorkShop. See *WorkShop: Getting Started* and *Beyond the Basics* for further information on the Analyzer.

`-xhelp=`*h* Show summary help information on options or `README` file.

♦ **SPARC:77 PPC:77 Intel:77**

The *h* is either `readme` or `flags`.

`readme`: Show the online `README` file for this release of the compiler.
`flags`: Show the compiler flags (options).

`-xhelp=flags` is a synonym for `-help`.

`-xildoff` Turn off the Incremental Linker.

♦ **SPARC: 77/90 PPC: – Intel: –**

This forces the use of the standard linker, `ld`.

This option is the default if you do *not* use the `-g` option. It is also the default if you use `-G` or name any source file on the command line.

Override this default by using the `-xildon` option.

`-xildon` Turn on the Incremental Linker.

♦ **SPARC: 77/90 PPC: – Intel: –**

Turn on the Incremental Linker and force the use of `ild` in incremental mode.

This option is the default if you use `-g` and do *not* use `-G`, and do *not* name any source file on the command line.

Override this default by using the `-xildoff` option.

See the *Incremental Link Editor* guide.

-xinline=*f1[,...,fn]* Synonym for `-inline=`*f1[,...,fn]*.

♦ **SPARC:77 PPC:77 Intel:77**

-xl[d] Enable more VMS Fortran extensions.

♦ **SPARC:77 PPC:77 Intel:77**

`-xl`: Enable the compiler to accept more VMS Fortran extensions. This is a macro that is translated to `-vax=misalign`, and provides the language features that are listed later in this description. See the description of `-vax=`, page 83.

Although most VMS features are accepted automatically by `f77` without any special options, you must use the `-xl` option for a few VMS extensions.

In general, you need the `-xl` option if a source statement can be interpreted as either a VMS feature or an `f77` or `f90` feature, and you want the VMS feature. In this case, the `-xl` option forces the compiler to interpret it the VMS way.

The following VMS language features are covered with this option:

- Unformatted record size in words rather than bytes (`-xl`)
- VMS style logical file names (`-xl`)
- Quote (") character introducing octal constants (`-xl`)
- Backslash (\) as ordinary character within character constants (`-xl`)
- Nonstandard form of the `PARAMETER` statement (`-xl`)
- Alignment of structures as in VMS. (`-xl`)
- Debugging lines as comment lines or Fortran statements (`-xld`)

Use the `-xl` to get VMS alignment if your program has some detailed knowledge of how VMS structures are implemented.

`-xld`: Specifying `-xld` option causes debugging comments (`D` or `d` in column one) to be compiled. Without the `-xld` option, they remain comments only. No space is allowed between `-xl` and `d`.

Programs that need to share structures with C programs should use the default and not `-xl`.

You may also be interested in `-lV77` and the VMS library. See the *Fortran Library Reference* for information on the VMS libraries.

Read the chapter on VMS language extensions in the *Fortran 77 Language Reference* for details of the VMS features that you get automatically.

-xlibmil Synonym for `-libmil`.

♦ **SPARC:77 PPC:77 Intel:77**

-xlibmopt Use library of optimized math routines.

♦ **SPARC: 77/90 PPC: – Intel: –**

Use selected math routines optimized for speed. This option usually generates faster code. It may produce slightly different results; if so, they usually differ in the last bit. The order on the command line for this library option is not significant.

-xlic_lib=*libs* Link with the specified Sun licensed libraries.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Specifies a comma-separated list of license-controlled libraries to link with. For example:

```
f77 -o pgx -fast pgx.f -xlic_lib=sunperf
```

As with -l, this option should appear on the command line after all source and object file names.

-xlicinfo Show license server information.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Use this option to return license information about the licensing system—in particular, the name of the license server and the user ID for each of the users who have licenses checked out.

Generally, with this option, no compilation takes place, and a license is not checked out. This option is normally used alone with no other options. However, if a conflicting option is used, then the last one on the command line prevails, and there is a warning.

```
demo% f77 –xlicinfo

   License server :  aardvaark
    sunpro.fortran Version 4.200 expires on : 13-may-1997
    License valid in domain : zoo.furry.COM
   Users of sunpro.fortran Version 4.200 :
   Total Licenses : 10    Licenses Available : 10
Using LM_LICENSE_FILE = ...etc
```

–Xlist[*x*] Produce listings and do global program checking.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Use this option to find potential programming bugs. It invokes an extra compiler pass to check for consistency in subprogram call arguments, common blocks, and parameters, across the global program. The option also generates a line–numbered listing of the source code, including a cross reference table. The error messages issued by the –`Xlist` options are advisory warnings and do not prevent the program from being compiled and linked.

Note – Not all `-Xlist` suboptions are available with release 1.2 of `f90`. Full global program checking will be available in a later `f90` release.

Example: Check across routines for consistency:

```
demo% f77 -Xlist  fil.f
```

The above example writes the following to the output file `fil.lst`:

- A line–numbered source listing (default)
- Error messages (embedded in the listing) for inconsistencies across routines
- A cross reference table of the identifiers (default)

By default, the listings are written to the file `name.lst`, where `name` is taken from the first listed source file on the command line.

A number of sub-options provide further flexibility in the selection of actions. These are specified by suffixes to the main –Xlist option, as shown in the following table:

Table 3-22 –Xlist Sub-options

Option	Feature
–Xlist	Show errors, listing, and cross reference table
–Xlistc	Show call graphs and errors
–XlistE	Show errors
–Xlisterr*[nnn]*	Suppress error *nnn* messages
–Xlistf	Show errors, listing, and cross references, but no object files
–Xlistfln*dir*	Put .fln files in directory *dir*, which must already exist
–Xlisth	Terminate compilation if errors detected
–XlistI	Analyze #include and INCLUDE files as well as source files
–XlistL	Show listing and errors only
–Xlistl*n*	Set page length to *n* lines
–Xlisto *name*	Rename report file to *name*.lst
–Xlists	Suppress unreferenced names from the cross-reference table
–Xlistv*n*	Set checking level to *n* (1,2,3, or 4) – default is 2
–Xlistw*[nnn]*	Set width of output line to *nnn* columns – default is 79
–Xlistwar*[nnn]*	Suppress warning *nnn* messages
–XlistX	Show cross-reference table and errors

Options -Xlistc, -Xlistf, -Xlistfln*dir*, -Xlisth, -Xlists, and -Xlistv*n* are not available with this release (1.2) of f90.

See the *Fortran Programmer's Guide* chapter *Program Analysis and Debugging* for details.

-xloopinfo

Synonym for –loopinfo.

♦ **SPARC:77 PPC: – Intel: –**

-xnolib Synonym for -nolib.

♦ **SPARC: 77/90 PPC:77 Intel:77**

-xnolibmil Synonym for -nolibmil.

♦ **SPARC: 77/90 PPC:77 Intel:77**

-xnolibmopt Do not use fast math library.

♦ **SPARC: 77/90 PPC:77 Intel:77**

Use with -fast to override linking the optimized math library:

```
f77 -fast -xnolibmopt …
```

-xO[*n*] Synonym for -O[*n*].

♦ **SPARC: 77/90 PPC:77 Intel:77**

-xpad Synonym for -pad.

♦ **SPARC:77 PPC: – Intel: –**

-xparallel Synonym for -parallel.

♦ **SPARC: 77/90 PPC: – Intel: –**

-xpg Synonym for -pg.

♦ **SPARC: 77/90 PPC:77 Intel:77**

-xpp={fpp|cpp} Select source file preprocessor.

♦ **SPARC: 77/90 PPC:77 Intel:77**

The default is -xpp=fpp.

The compilers use fpp(1) to preprocess .F or .F90 source files. This preprocessor is appropriate for Fortran. Previous versions used the standard C preprocessor cpp. To select cpp, specify -xpp=cpp.

-xprofile=*p* Collect or optimize with runtime profiling data.

♦ **SPARC:77 PPC: – Intel:77**

p must be one of `collect`[`:`*nm*], `use`[`:`*nm*], or `tcov`. Optimization level must be –`O2` or greater.

Only `-xprofile=tcov` is available on Intel.

`collect`[`:`*nm*] (*SPARC*)

Collect and save execution frequency data for later use by the optimizer with `–xprofile=use`. The compiler generates code to measure statement execution frequency.

The *nm* is the name of the program that is being analyzed. This name is optional. If *nm* is not specified, `a.out` is assumed to be the name of the executable.

At runtime a program compiled with `-xprofile=collect:`*nm* will create the subdirectory *nm*`.profile` to hold the runtime feedback information. Data is written to the file `feedback` in this subdirectory. If you run the program several times, the execution frequency data accumulates in the `feedback` file; that is, output from prior runs is not lost.

`use`[`:`*nm*] (*SPARC*)

Use execution frequency data to optimize strategically.

As with `collect:nm`, the *nm* is optional and may be used to specify the name of the program.

The program is optimized by using the execution frequency data previously generated and saved in the `feedback` files written by a previous execution of the program compiled with `–xprofile=collect`.

The source files and other compiler options must be exactly the same as used for the compilation that created the compiled program that generated the `feedback` file. If compiled with `-xprofile=collect:`*nm*, the same program name *nm* must appear in the optimizing compilation:
`-xprofile=use:`*nm*.

`tcov` (*SPARC, Intel*)

Basic block coverage analysis using “new” style `tcov`.

Code instrumentation is similar to that of `-a`, but `.d` files are no longer generated for each source file. Instead, a single file is generated, whose name is based on the name of the final executable. For example, if `stuff` is the executable file, then `stuff.profile/tcovd` is the data file.

When running `tcov`, you must pass it the `-x` option to make it use the new style of data. If not, `tcov` uses the old `.d` files, if any, by default for data, and produces unexpected output.

Unlike `-a`, the `TCOVDIR` environment variable has no effect at compile–time. However, its value is used at program runtime to identify where to create the profile subdirectory.

See the `tcov`(1) man page, the *Performance Profiling* chapter of the *Fortran Programmer's Guide*, and the Sun *Performance Profiling Tools* manual for more details.

`-xreduction` Synonym for `-reduction`.

♦ **SPARC: 77/90 PPC: – Intel: –**

`-xregs=`*r* Specify register usage.

♦ **SPARC:77 PPC: – Intel: –**

r is a comma–separated list that consists of one or more of the following:

[`no%`]`appl`, [`no%`]`float`.

Where the `%` is shown, it is a required character.

Example: `-xregs=appl,no%float`

`appl`: Allow using the registers `g2`, `g3`, and `g4`.

On SPARC systems, these registers are described as *application* registers. Using these registers can increase performance because fewer load and store instructions are needed. However, such use can conflict with some old library programs written in assembly code.

`no%appl`: Do not use the `appl` registers.

`float`: Allow using the floating–point registers as specified in the SPARC ABI.

You can use these registers even if the program contains no floating–point code.

`no%float`: Do not use the floating–point registers.

With this option, a source program cannot contain any floating–point code.

The default is: `–xregs=appl,float`.

`–xs` Allow debugging by `dbx` without object (`.o`) files .

♦ **SPARC:77 PPC:77 Intel:77**

With `–xs`, if you move executables to another directory, then you can use `dbx` and ignore the object (`.o`) files. Use this option when you cannot keep the `.o` files.

- The compiler passes `-s` to the assembler and then the linker places all symbol tables for `dbx` in the executable file.
- This way of handling symbol tables is the older way. It is sometimes called *no auto–read*.
- The linker links more slowly, and `dbx` initializes more slowly.

Without `–xs`, if you move the executables, you must move both the source files and the object (`.o`) files, or set the path with either the dbx `pathmap` or `use` command.

- This way of handling symbol tables is the newer and default way of loading symbol tables. It is sometimes called *auto–read*.
- The symbol tables are distributed in the `.o` files so that `dbx` loads the symbol table information only if and when it is needed. Hence, the linker links faster, and `dbx` initializes faster.

`–xsafe=mem` Assume no memory–based traps.

♦ **SPARC:77 PPC: – Intel: –**

Using this option allows the compiler to assume no memory–based traps occur. It grants permission to use the speculative load instruction on V9 machines. It is only effective if `-O5` and `-xarch=v8plus` are also specified.

–xsb Synonym for –sb.

♦ **SPARC:77 PPC:77 Intel:77**

–xsbfast Synonym for –sbfast.

♦ **SPARC:77 PPC:77 Intel:77**

–xspace Do not allow optimizations to increase code size.

♦ **SPARC:77 PPC: – Intel: –**

Do no optimizations that increase the code size.

Example: Do not unroll or parallelize loops if it increases code size.

–xtarget=*t* Specify system for optimization.

♦ **SPARC:77 PPC:77 Intel:77**

Specify the target system for the instruction set and optimization.

t must be one of: native, generic, *system–name*.

The –xtarget option permits a quick and easy specification of the –xarch, -xchip, and –xcache combinations that occur on real systems. The only meaning of –xtarget is in its expansion.

The performance of some programs may benefit by providing the compiler with an accurate description of the target computer hardware. When program performance is critical, the proper specification of the target hardware could be very important. This is especially true when running on the newer SPARC processors. However, for most programs and older SPARC processors, the performance gain is negligible and a generic specification is sufficient.

native: Optimize performance for the host system.

The compiler generates code optimized for the host system. It determines the available architecture, chip, and cache properties of the machine on which the compiler is running.

generic: Get the best performance for generic architecture, chip, and cache.

The compiler expands –xtarget=generic to:

`–xarch=generic –xchip=generic –xcache=generic`

This is the default value.

system–name: Get the best performance for the specified system.

Select a system target name from Table 3-23.

This option is a macro. Each specific value for `–xtarget` expands into a specific set of values for the `–xarch`, `–xchip`, and `–xcache` options, as shown in Table 3-23. `fpversion`(1) can be run to determine the target definitions on any system.

For example:

`–xtarget=sun4/15` means `–xarch=v8a –xchip=micro –xcache=2/16/1`

Table 3-23 The `-xtarget` Expansions

-xtarget	**-xarch**	**-xchip**	**-xcache**
sun4/15	v8a	micro	2/16/1
sun4/20	v7	old	64/16/1
sun4/25	v7	old	64/32/1
sun4/30	v8a	micro	2/16/1
sun4/40	v7	old	64/16/1
sun4/50	v7	old	64/32/1
sun4/60	v7	old	64/16/1
sun4/65	v7	old	64/16/1
sun4/75	v7	old	64/32/1
sun4/110	v7	old	2/16/1
sun4/150	v7	old	2/16/1
sun4/260	v7	old	128/16/1
sun4/280	v7	old	128/16/1
sun4/330	v7	old	128/16/1
sun4/370	v7	old	128/16/1

Table 3-23 The `-xtarget` Expansions *(Continued)*

-xtarget	**-xarch**	**-xchip**	**-xcache**
sun4/390	v7	old	128/16/1
sun4/470	v7	old	128/32/1
sun4/490	v7	old	128/32/1
sun4/630	v7	old	64/32/1
sun4/670	v7	old	64/32/1
sun4/690	v7	old	64/32/1
sselc	v7	old	64/32/1
ssipc	v7	old	64/16/1
ssipx	v7	old	64/32/1
sslc	v8a	micro	2/16/1
sslt	v7	old	64/32/1
sslx	v8a	micro	2/16/1
sslx2	v8a	micro2	8/16/1
ssslc	v7	old	64/16/1
ss1	v7	old	64/16/1
ss1plus	v7	old	64/16/1
ss2	v7	old	64/32/1
ss2p	v7	powerup	64/32/1
ss4	v8a	micro2	8/16/1
ss4/85	v8a	micro2	8/16/1
ss4/110	v8a	micro2	8/16/1
ss5	v8a	micro2	8/16/1
ss5/85	v8a	micro2	8/16/1
ss5/110	v8a	micro2	8/16/1
ssvyger	v8a	micro2	8/16/1
ss10	v8	super	16/32/4
ss10/hs11	v8	hyper	256/64/1

Table 3-23 The `-xtarget` Expansions *(Continued)*

-xtarget	**-xarch**	**-xchip**	**-xcache**
ss10/hs12	v8	hyper	256/64/1
ss10/hs14	v8	hyper	256/64/1
ss10/20	v8	super	16/32/4
ss10/hs21	v8	hyper	256/64/1
ss10/hs22	v8	hyper	256/64/1
ss10/30	v8	super	16/32/4
ss10/40	v8	super	16/32/4
ss10/41	v8	super	16/32/4:1024/32/1
ss10/50	v8	super	16/32/4
ss10/51	v8	super	16/32/4:1024/32/1
ss10/61	v8	super	16/32/4:1024/32/1
ss10/71	v8	super2	16/32/4:1024/32/1
ss10/402	v8	super	16/32/4
ss10/412	v8	super	16/32/4:1024/32/1
ss10/512	v8	super	16/32/4:1024/32/1
ss10/514	v8	super	16/32/4:1024/32/1
ss10/612	v8	super	16/32/4:1024/32/1
ss10/712	v8	super2	16/32/4:1024/32/1
ss20	v8	super	16/32/4:1024/32/1
ss20/hs11	v8	hyper	256/64/1
ss20/hs12	v8	hyper	256/64/1
ss20/hs14	v8	hyper	256/64/1
ss20/hs21	v8	hyper	256/64/1
ss20/hs22	v8	hyper	256/64/1
ss20/50	v8	super	16/32/4
ss20/51	v8	super	16/32/4:1024/32/1
ss20/61	v8	super	16/32/4:1024/32/1

Table 3-23 The `-xtarget` Expansions *(Continued)*

-xtarget	**-xarch**	**-xchip**	**-xcache**
ss20/71	v8	super2	16/32/4:1024/32/1
ss20/151	v8	hyper	512/64/1
ss20/152	v8	hyper	512/64/1
ss20/502	v8	super	16/32/4
ss20/512	v8	super	16/32/4:1024/32/1
ss20/514	v8	super	16/32/4:1024/32/1
ss20/612	v8	super	16/32/4:1024/32/1
ss20/712	v8	super	16/32/4:1024/32/1
ss600/41	v8	super	16/32/4:1024/32/1
ss600/51	v8	super	16/32/4:1024/32/1
ss600/61	v8	super	16/32/4:1024/32/1
ss600/120	v7	old	64/32/1
ss600/140	v7	old	64/32/1
ss600/412	v8	super	16/32/4:1024/32/1
ss600/512	v8	super	16/32/4:1024/32/1
ss600/514	v8	super	16/32/4:1024/32/1
ss600/612	v8	super	16/32/4:1024/32/1
ss1000	v8	super	16/32/4:1024/32/1
sc2000	v8	super	16/32/4:2048/64/1
cs6400	v8	super	16/32/4:2048/64/1
solb5	v7	old	128/32/1
solb6	v8	super	16/32/4:1024/32/1
ultra	v8	ultra	16/32/1:512/64/1
ultra2	v8	ultra2	16/32/1:512/64/1
ultra1/140	v8	ultra	16/32/1:512/64/1
ultra1/170	v8	ultra	16/32/1:512/64/1
ultra1/200	v8	ultra	16/32/1:512/64/1

Table 3-23 The `-xtarget` Expansions *(Continued)*

-xtarget	-xarch	-xchip	-xcache
ultra2/1170	v8	ultra	16/32/1:512/64/1
ultra2/1200	v8	ultra	16/32/1:1024/64/1
ultra2/1300	v8	ultra2	16/32/1:2048/64/1
ultra2/2170	v8	ultra	16/32/1:512/64/1
ultra2/2200	v8	ultra	16/32/1:1024/64/1
ultra2/2300	v8	ultra2	16/32/1:2048/64/1
entr2	v8	ultra	16/32/1:512/64/1
entr2/1170	v8	ultra	16/32/1:512/64/1
entr2/2170	v8	ultra	16/32/1:512/64/1
entr2/1200	v8	ultra	16/32/1:512/64/1
entr2/2200	v8	ultra	16/32/1:512/64/1
entr150	v8	ultra	16/32/1:512/64/1
entr3000	v8	ultra	16/32/1:512/64/1
entr4000	v8	ultra	16/32/1:512/64/1
entr5000	v8	ultra	16/32/1:512/64/1
entr6000	v8	ultra	16/32/1:512/64/1

For PowerPC: –`xtarget=` accepts `generic` or `native`.

For Intel: –`xtarget=` accepts:

- `generic` or `native`
- `386` (equivalent to –`386` option) or `486` (equivalent to –`486` option)
- `pentium` (equivalent to –`pentium` option) or `pentium_pro`

`-xtime`

Synonym for –`time`.

♦ **SPARC: 77/90 PPC:77 Intel:77**

-xtypemap=*spec* Specify default data mappings.

♦ **SPARC:77 PPC:77 Intel:77**

This option provides a flexible way to specify the byte sizes for default data types. Compare with `-dbl` and `-r8`.

The syntax of the specification string *spec* is:

type`:`*size*`,` *type*`:`*size*`,`...

The allowable data types are `real`, `double`, `integer`.

The accepted data sizes are `64`, `128`, and `mixed`.

This option applies to all variables declared with default specifications (without explicit byte sizes), as in `REAL XYZ`.

The allowable combinations on each platform are:

Table 3-24 Allowed `-xtypemap=` mappings

SPARC	PowerPC	Intel
`real:64`	`real:64`	`real:64`
`double:64`	`double:64`	`double:64`
`double:128`	`double:128`	–
`integer:64`	`integer:64`	`integer:64`
`integer:mixed`	`integer:mixed`	`integer:mixed`

The mapping `integer:mixed` indicates 8 byte integers but only 4 byte arithmetic.

The `-dbl` and `-r8` options have their `-xtypemap` equivalents:

On SPARC and PowerPC:		
`-dbl`	*same as:*	`-xtypemap=real:64,double:128,integer:64`
`-r8`	*same as:*	`-xtypemap=real:64,double:128,integer:mixed`
On Intel:		
`-dbl`	*same as:*	`-xtypemap=real:64,double:64,integer:64`
`-r8`	*same as:*	`-xtypemap=real:64,double:64,integer:mixed`

There are two additional possibilities on SPARC and PowerPC:

```
-xtypemap=real:64,double:64,integer:mixed
```

```
-xtypemap=real:64,double:64,integer:64
```

which map both default REAL and DOUBLE to 8 bytes.

Note that INTEGER and LOGICAL are treated the same, and COMPLEX is mapped as two REALs. Also, DOUBLE COMPLEX will be treated the way DOUBLE is mapped.

-xunroll=*n* Synonym for -unroll=*n*.

♦ **SPARC:77 PPC:77 Intel:77**

-xvpara Synonym for -vpara.

♦ **SPARC:77 PPC: – Intel: –**

-Zlp Compile for loop performance profiling by looptool.

♦ **SPARC:77 PPC: – Intel: –**

Prepare object files for the loop profiler, looptool. The looptool(1) utility can then be run to generate loop statistics about the program.

If you compile and link in separate steps, and you compile with -Zlp, then be sure to link with -Zlp.

If you compile *one* subprogram with -Zlp, you need not compile *all* the subprograms of that program with -Zlp. However, you receive the loop information only for the files compiled with -Zlp, and no indication that the program includes other files.

Refer to *WorkShop: Beyond the Basics* for more information.

-ztext Generate only pure libraries with no relocations.

♦ **SPARC:77 PPC: – Intel: –**

Do not make the library if relocations remain.

The general purpose of -ztext is verify that a generated library is pure text; instructions are all position–independent code. Therefore, it is generally used with both -G and -pic.

With –ztext, if ld finds an incomplete relocation in the *text* segment, then it does not build the library. If it finds one in the *data* segment, then it generally builds the library anyway; the data segment is writable.

Without –ztext, ld builds the library, relocations or not.

A typical use is to make a library from both source files and object files, where you do not know if the object files were made with –pic.

Example: Make library from both source and object files:

```
demo% f77 –G –pic –ztext –o MyLib –hMyLib a.f b.f x.o y.o
```

An alternate use is to ask if the code is position–independent already: compile without –pic, but ask if it is pure text.

Example: Ask if it is pure text already—even without –pic:

```
demo% f77 –G –ztext –o MyLib –hMyLib a.f b.f x.o y.o
```

If you compile with –ztext and ld does not build the library, then you can recompile without –ztext, and ld will build the library. The failure to build with –ztext means that one or more components of the library cannot be shared; however, maybe some of the other components can be shared. This raises questions of performance that are best left to you, the programmer.

–Ztha

Compile for performance profiling with Thread Analyzer.

♦ **SPARC:77 PPC: – Intel: –**

Prepare object files for Thread Analyzer. This option inserts calls to a profiling library at all procedure entries and exits. Code compiled with –Ztha links with the library libtha.so. The –Ztha option is usable only with a Sun WorkShop license.

If you compile and link in separate steps, and you compile with –Ztha, then link with –Ztha. All subprograms need not be compiled with –Ztha. However, thread statistics will appear only for the files compiled with the option.

Refer to tha (1) or *WorkShop: Beyond the Basics* for details.

Runtime Error Messages A

This appendix describes the error messages generated by the Fortran I/O library, signal handler, and operating system.

Note – Only `f77` is described in this Appendix. Information on `f90` runtime error messages will be added.

Operating System Error Messages

Operating system error messages include system call failures, C library errors, and shell diagnostics. The system call error messages are found in `intro`(2). System calls made through the `Fortran` library do not produce error messages directly. The following system routine in the `Fortran` library calls C library routines which produce an error message:

```
    CALL SYSTEM("rm /")
    END
```

The following message is displayed:

```
rm: / directory
```

A

Signal Handler Error Messages

Before beginning execution of a program, the Fortran library sets up a signal handler (`sigdie`) for signals that can cause termination of the program. `sigdie` prints a message that describes the signal, flushes any pending output, and generates a core image and a traceback.

Presently, the only arithmetic exception that produces an error message is the `INTEGER*2` division with a denominator of zero. All other arithmetic exceptions are ignored.

A signal handler error example follows, where the subroutine `SUB` tries to access parameters that are not passed to it:

```
    CALL SUB()
    END
    SUBROUTINE SUB(I,J,K)
    I=J+K
    RETURN
    END
```

The following error message results:

```
*** Segmentation violation
Illegal instruction (core dumped)
```

I/O Error Messages (f77)

The error messages in this section are generated by the Fortran 77 I/O library. The error numbers are returned in the `IOSTAT` variable if the `ERR` return is taken.

For example, the following program tries to do an unformatted write to a file opened for formatted output:

```
WRITE( 6 ) 1
END
```

and produces error messages like the following:

```
sue: [1003] unformatted io not allowed
logical unit 6, named 'stdout'
lately: writing sequential unformatted external IO
```

The following error messages are generated. These same messages are also documented at the end of the `man` page, `perror`(3f).

If the error number is less than 1000, then it is a *system* error. See `intro` (2).

Table A-1 `f77` Runtime I/O Messages

Error	Message
1000	`error in format` Read the error message output for the location of the error in the format. It can be caused by more than 10 levels of nested parentheses or an extremely long format statement.
1001	`illegal unit number` It is illegal to close logical unit 0. Negative unit numbers are not allowed. The upper limit is 2^{31} - 1.
1002	`formatted io not allowed` The logical unit was opened for unformatted I/O.
1003	`unformatted io not allowed` The logical unit was opened for formatted I/O.
1004	`direct io not allowed` The logical unit was opened for sequential access, or the logical record length was specified as 0.
1005	`sequential io not allowed` The logical unit was opened for direct access I/O.
1006	`can't backspace file` You cannot do a `seek` on the file associated with the logical unit; therefore, you cannot backspace. The file may be a `tty` device or a pipe.
1007	`off beginning of record` You tried to do a left tab to a position before the beginning of an internal input record.

A

Table A-1 `f77` Runtime I/O Messages

Error	Message
1008	`can't stat file` The system cannot return status information about the file. Perhaps the directory is unreadable.
1009	`no * after repeat count` Repeat counts in list-directed I/O must be followed by an * with no blank spaces.
1010	`off end of record` A formatted write tried to go beyond the logical end-of-record. An unformatted read or write also causes this
1011	*<Not used>*
1012	`incomprehensible list input` List input has to be as specified in the declaration.
1013	`out of free space` The library dynamically creates buffers for internal use. You ran out of memory for them; that is, your program is too big.
1014	`unit not connected` The logical unit was not open.
1015	`read unexpected character` Certain format conversions cannot tolerate nonnumeric data.
1016	`illegal logical input field` `logical` data must be `T` or `F`.
1017	`'new' file exists` You tried to open an existing file with `status='new'`.
1018	`can't find 'old' file` You tried to open a nonexistent file with `status='old'`.
1019	`unknown system error` This error should not happen, but..
1020	`requires seek ability` Attempted a `seek` on a file that does not allow it. I/O operation requiring a `seek` are direct access, sequential unformatted I/O, and tabbing left.
1021	`illegal argument` Certain arguments to open and related functions are checked for legitimacy. Often only nondefault forms are checked

Table A-1 `f77` Runtime I/O Messages

Error	Message
1022	`negative repeat count` The repeat count for list-directed input must be a positive integer.
1023	`illegal operation for unit` Attempted an I/O operation that is not possible for the device associated with the logical unit. You get this error if you try to read past end-of-tape, or end-of-file.
1024	*<Not used>*
1025	`incompatible specifiers in open` Attempted to open a file with the `'new'` option and the `access='append'` option, or some other invalid combination.
1026	`illegal input for namelist` A namelist read encountered an invalid data item.
1027	`error in FILEOPT parameter` The `FILEOPT` string in an `OPEN` statement has bad syntax.
1028	`WRITE to readonly file` Attempt to write on a unit that was opened for reading only.
1029	`READ from writeonly file` Attempt to read from a unit that was opened for writing only.

I/O Error Messages (f90)

This is a partial list of runtime I/O messages issued by `f90` 1.2 :

Table A-2 `f90` Runtime I/O Errors

Error	Meaning
1001	Tried to read past end of file
1002	Tried to read an empty file
1003	Tried to read past endfile record
1004	Tried to read past EOF on namelist
1005	Tried to read past internal file EOF

A

Table A-2 `f90` Runtime I/O Errors

Error	Meaning
1006	Read past EOR with ADVANCE='NO'
1010	Invalid unit number
1011	Invalid unit number on OPEN
1021	Unit not opened for direct access
1022	Unit not connected to tape
1023	Unit is not connected
1024	Opening too many named files
1025	File is opened with another structure
1026	File is opened by an auxiliary i/o
1027	ACTION= conflicts with file perms
1029	Error on underlying stdio I/O request
1030	Unknown STATUS parameter on OPEN
1031	Unknown ACCESS parameter on OPEN
1032	Unknown FORM parameter on OPEN
1033	Unknown RECL parameter on OPEN
1034	Unknown BLANK parameter on OPEN
1035	Unknown POSITION parameter on OPEN
1037	RECL must be multiple of 8 for pure
1038	Unknown ACTION specifier on OPEN
1039	Unknown DELIM specifier on OPEN
1040	FILE specifier required on OPEN
1041	FILE specifier invalid on OPEN
1042	RECL specifier required on OPEN
1044	BLANK specifier invalid on OPEN
1045	POSITION specifier invalid on OPEN
1047	ASSIGN by file/unit conflict
1048	Unknown PAD specifier on OPEN

Table A-2 `f90` Runtime I/O Errors

Error	Meaning
1049	DELIM specifier invalid on OPEN
1050	File must exist prior to OPEN
1051	File must not exist prior to OPEN
1052	File is connected to another unit
1053	Unable to position to end of file
1054	Only BLANK can be changed on reopen
1055	File cannot be opened (structure)
1056	File cannot be opened for direct acc.
1058	STATUS=NEW on currently-open-file
1060	Attempt to OPEN standard file wrong
1067	PAD specifier invalid on OPEN
1068	File cannot be opened for unfmtd acc
1069	File cannot be opened for fmtd acc
1070	Unknown STATUS parameter on CLOSE
1071	Invalid STATUS parameter on CLOSE
1072	Increment in implied do is 0.
1079	BACKSPACE requires read permission
1080	Formatted I/O invalid on unformatted
1081	Unformatted I/O invalid on formatted
1082	Direct access I/O invalid on seq.
1083	Sequential I/O invalid on direct acc.
1084	BACKSPACE invalid on direct access
1085	ENDFILE invalid on direct access
1086	REWIND invalid on direct access
1087	Read after write invalid on seq.
1088	Invalid record number (%d)
1090	No read permission

A

Table A-2 `f90` Runtime I/O Errors

Error	Meaning
1091	No write permission
1092	File does not support BACKSPACE
1093	File does not support ENDFILE
1094	File does not support REWIND
1095	WRITE or PRINT invalid after ENDFILE
1096	ENDFILE invalid after ENDFILE
1097	Record number does not exist in file
1100	Record number does not exist in file
1117	Infinite loop in format
1118	Literal invalid in input format
1170	Data type mismatch on READ
1171	Data type mismatch on WRITE
1173	Invalid logical input field
1180	Unknown input on list-directed read
1181	Invalid complex on list-directed read
1182	String too long on list-directed read
1190	Invalid character in numeric input
1191	Overflow converting numeric input
1192	Exponent underflow on numeric input
1193	Exponent overflow on numeric input
1194	Blank numeric input field
1201	Tried to read past end of record
1202	Read/wrote too little data
1205	Unable to request more memory space
1208	An I/O statement was already active
1211	Tried to write a too long record
1212	Tried to write beyond internal file

Table A-2 `f90` Runtime I/O Errors

Error	Meaning
1213	Ptr/alloc array not assoc/alloc'ed
1214	FMT var not allocated or associated
1215	UNIT var not allocated or associated
1216	FMT var or array is zero-sized
1217	Read encountered a malformed record
1220	Internal Fortran library error
1221	Internal error - unknown file struct.
1223	Internal error - unknown data type
1224	Internal error invalid parsed format
1226	Internal error on tape read
1306	First/last character unknown nl read
1307	Unknown input on namelist read
1308	Zero length char in nl for f90
1309	Array section input to f90 nml
1310	Namelist read error
1312	Invalid char passed to namelist rtn.
1313	Namelist variable name too long
1314	Namelist input group name mismatch
1315	Unrecognized namelist variable name
1316	Unable to obtain namelist value
1317	Invalid logical data in namelist read
1318	Invalid complex data in namelist read
1320	Input rec. too long on namelist read
1321	Attempted namelist read beyond array
1322	Namelist not supported for local mem.
1323	Too many namelist elements specified
1324	Unrecognized namelist variable name

Table A-2 `f90` Runtime I/O Errors

Error	Meaning
1325	Data type mismatch on namelist read
1326	Namelist name is larger than recsize
1327	Double complex illegal for f77 mode
1328	Structures illegal for f77 mode
1329	Bad pre-ampersand character in f90
1330	Direct access file invalid for BUFIO
1331	Formatted file invalid for BUFIO
1332	Start address > end address for BUFIO
1334	Invalid argument to SETPOS
1335	Positioning operation not supported
1338	Mixing BUFIO/READ/WRITE on pure file
1339	Mixing auxiliary and Fortran I/O
1340	Invalid DECODE record length
1341	Invalid ENCODE record length
1342	Invalid number of items for BUFIO
1343	Invalid ADVANCE= specifier on rd/wrt
1344	ADVANCE='NO' requ'd with SIZE=
1345	ADVANCE='NO' requ'd with EOR=
1350	Negative tape block number is invalid
1354	Maximum tape block size exceeded
1355	Invalid combination of parameters
1356	Unrecovered tape error on tape read
1360	Tblmgr routine called with bad args.
1361	Tblmgr routine called with bad NTAB
1362	Tblmgr routine called with bad tab. #
1363	Tblmgr routine called with bad incr.
1370	Read or write of nonbyte-data is inv.

Table A-2 `f90` Runtime I/O Errors

Error	Meaning
1371	Data conversion routine not loaded
1372	Can't convert this type with f90
1373	I/O not supported for this KIND
1380	Argument list is not valid

A

Features Release History B

This Appendix lists the new and changed features in this and previous release of `f77` and `f90`:

New Features and Behavior Changes in Fortran 77 (`f77`)

This section lists the new features and behavior changes specific to `f77`.

Features in `f77` 4.2 that are New Since 4.0

`f77` 4.2 includes the following features that are new or changed since the 4.0 release:

- New options:
  - `-dbl_align_all`
  - `-errtags=yes|no` and `-errofs=`*taglist*
  - `-stop_status=no|yes`
  - `-xcrossfile`
  - `-xlic_lib=`*libs*
  - `-xpp=fpp|cpp`
  - `-xtypemap=`*type:spec*,.
- Changed options:
  - Options `-fround`, `-fsimple`, `-ftrap`, `-xprofile=tcov`, `-xspace`, `-xunroll` now available on PowerPC and Intel platforms.

- `-xtarget, -xarch, -xchip` expanded for SPARC Ultra and Intel and PowerPC platforms.
- `-vax=` expanded to enable selection/deselection of individual VAX/VMS Fortran features.
- Default sourcefile preprocessor is `fpp`(1) rather than `cpp`(1).

Features in `f77` 4.0 that are New Since 3.0/3.0.1

`f77` 4.0 includes the following features that are new or changed since 3.0/3.0.1:

- The `DOSERIAL` and `DOSERIAL*` parallel directives have been added, and the `DOALL` directive expanded.
- A directive for unrolling loops has been added.
- The `-I`*dir* option now also affects the `f77 INCLUDE` statement, not only the preprocessor #include directive.
- The Incremental Linker is available. It provides faster linking and speeds up development.
- The `-oldstruct` command-line option has been deleted.
- The following new synonyms have been added: `-xautopar`, `-xdepend`, `-xexplicitpar`, `-xloopinfo`, `-xparallel`, `-xreduction`, and `-xvpara`.
- The `-stackvar` restrictions `EQUIVALANCE`, `NAMELIST`, `STRUCTURE`, and `RECORD` have been removed.
- New options have been added (and some changed):

Table B-1 New Features in `f77` 4.0 Since 3.0/3.0.1

`-arg=local`	Pass by value result.
`-copyargs`	Allow assignment to constant arguments.
`-dbl`	Double the default size for integers, reals, and so forth.
`-ext_names=`*e*	Make external names with or without underscores.
`-fns`	Turn on SPARC non-standard floating-point mode *(SPARC, 2.x)*.
`-fround=`*r*	Set the IEEE rounding mode in effect at startup *(SPARC, 2.x)*.
`-fsimple`[`=`*n*]	Allow levels of simple floating-point model.
`-ftrap=`*t*	Set the IEEE trapping mode in effect at startup *(SPARC, 2.x)*.
`-mp=`*x*	Use either Sun-style or Cray-style MP directives *(SPARC, 2.x)*.
`-O5`	Attempt the highest level of optimization.

Table B-1 New Features in `f77` 4.0 Since 3.0/3.0.1 *(Continued)*

`-pad=`*p*	Pad local variables or common blocks
`-vax=`*v*	Specify a choice of VMS features to use.
`-xarch=`*a*	Limit the set of instructions the compiler may use *(SPARC, 2.x).*
`-xcache=`*c*	Define the cache properties for use by the optimizer *(SPARC, 2.x).*
`-xchip=`*c*	Specify the target processor for use by the optimizer *(SPARC, 2.x).*
`-xhelp=`*h*	Show help information for `README` file or for options (flags).
`-xildoff`	Turn off the Incremental Linker *(SPARC, 2.x).*
`-xildon`	Turn on the Incremental Linker *(SPARC, 2.x).*
`-xprofile=`*p*	Collect data for a profile or use a profile to optimize *(SPARC, 2.x).*
`-xregs=`*r*	Specify the usage of registers for the generated code *(SPARC, 2.x).*
`-xsafe=mem`	Allow compiler to assume no memory-based traps *(SPARC, 2.x).*
`-xspace`	Do no optimizations that increase the code size *(SPARC, 2.x).*
`-xtarget=`*t*	Specify target system for instruction set *(SPARC, 2.x).*
`-ztext`	Do not make the library if relocations remain.

- `DO`-loop code is now implemented differently to allow better optimization and loop parallelization. Legal `DO`-loops behave exactly the same as before; however, illegal `DO`-loops—zero-step, loop variable modified within the loop—may display different behavior.
- Full 64-bit integers have been added. With `-dbl`, integers not declared with a specified size are turned into full 64-bit integers.
- The following `libV77` library routines: `date`, `mvbits`, `ran`, and `secnds`, are now folded into the `libF77` library. That is, you no longer need to compile with the `-lV77` option to get these routines.
- The `OPEN` statement now contains a new keyword specifier, `ACTION=`*act*, where *act* is `READ`, `WRITE`, or `READWRITE`.

Fortran 77 Upward Compatibility

The Fortran 4.2 *source* is compatible with Fortran 3.0/3.0.1 (or earlier), except for minor changes due to operating system changes and bug fixes.

B

Fortran 3.0/3.0.1 to 4.0

Executables (`a.out`), libraries (`.a`), and object files (`.o`) compiled and linked in Fortran 3.0/3.0.1 under Solaris 2.x are compatible with Fortran 4.2 under Solaris 2.x.

BCP: Running Applications from Solaris 1.x in 2.x

You must install the Binary Compatibility Package for the executable to run.

Executables compiled and linked in Solaris 1.x do run in Solaris 2.3 and later, but they do not run as fast as when they are compiled and linked under the appropriate Solaris release.

Libraries (`.a`) and object files (`.o`) compiled and linked in Fortran 2.0.1 under Solaris 1.x are *not* compatible with Fortran 4.2 under Solaris 2.x.

Application Development in Solaris 2.x for 1.x

Under Solaris 2.x, you can make executables and libraries for Solaris 1.x, but it is not recommended. For the compiler to do this correctly, first install the Binary Compatibility Package. Then, to make it all work, you must:

- Use the Solaris 1.x compiler in BCP mode.
- Use the Solaris 1.x linker (`ld`), with `-qpath` set to the path for the 1.x `ld`.
- Link with the Solaris 1.x libraries. If you receive error messages like: `bad magic number`, check the `-L` options and the `LD_LIBRARY_PATH` environment variable.

New Features and Behavior Changes in Fortran 90

This section lists the new features and behavior changes specific to `f90`.

- This release (1.2) incorporates bug fixes against the previous release (1.1).

Fortran 90 Features and Differences C

This appendix shows some of the major features differences between:

- Standard Fortran 90 and Sun Fortran 90
- FORTRAN 77 and Fortran 90

Standards

This Fortran is an enhanced ANSI Standard Fortran development system.

- It conforms to the ANSI X3.198-1992 Fortran standard and the corresponding International Standards Organization ISO/IEC 1539-1:1991 (E) Fortran standard.
- It provides an IEEE standard 754-1985 floating-point package.
- On SPARC systems, it provides support for optimization exploiting features of SPARC V8, including the SuperSPARC™ implementation. These features are defined in the *SPARC Architecture Manual: Version 8*.

Features

Sun Fortran 90 provides the following features.

Tabs in the Source

`f90` allows the tab character in fixed-form source and in free-form source. Standard Fortran does not allow tabs.

The tab character is not converted to a blank, so the visual placement of tabbed statements depends on the utility you use to edit or display text.

Fixed-Form Source

- For a tab in column one:

  If the next character is a nonzero digit,
  then the current line is a *continuation* line;
  otherwise, the current line is an *initial* line.

- A tab cannot precede a statement label.
- A tab after column one is treated by `f90` the same as a blank character, except in literal strings.

Free-Form Source

`f90` treats a tab and a blank character as equivalent, except in literal strings.

Continuation Line Limits

`f90` allows 99 continuation lines (1 initial and 98 continuation lines). Standard Fortran allows 19 for fixed-form and 39 for free-form.

Fixed-Form Source of 96 Characters

In fixed-form source, lines can be 96 characters long. Columns 73 through 96 are ignored. Standard Fortran allows 72-character lines.

Directives

`f90` allows directive lines starting with `CDIR$`, `!DIR$`, `CMIC$`, or `!MIC$`. They look like comments but are not. For full details on directives, see “Directives” on page 140. Standard Fortran has no directives.

Source Form Assumed

The source form assumed by `f90` depends on options, directives, and suffixes.

- Command-line options

Option	Action
`-fixed`	Interpret all source files as Fortran *fixed* form
`-free`	Interpret all source files as Fortran *free* form

If the `-free` or `-fixed` option is used, that overrides the file name suffix.

- File name suffixes

Suffix	Source Form
`.f90`	Fortran *free-form* source files
`.f`	Fortran *fixed-form* source files or ANSI standard FORTRAN 77 source files
`.for`	Same as `.f`.
`.ftn`	Same as `.f`.
other	None—file name is passed to the linker

- Directives

Directive	Action
`!DIR$ FIXED`	Interpret the rest of the source file as Fortran *fixed* form
`!DIR$ FREE`	Interpret the rest of the source file as Fortran *free* form

If either a `FREE` or `FIXED` directive is used, that overrides the option and file name suffix.

Mixing Forms

Some mixing of source forms is allowed.

- In the same `f90` command, some source files can be fixed form, some free.
- In the same file, free form *can* be mixed with fixed form by using directives.

Case

Sun Fortran 90 is case insensitive at this release (1.2). That means that a variable `AbcDeF` is treated as if it were spelled `abcdef`, or `abcdeF`, etc. See *Compatibility with FORTRAN 77* on page 145.

Boolean Type

`f90` supports constants and expressions of Boolean type. There are no Boolean variables or arrays, and there is no Boolean type statement.

Miscellaneous Rules Governing Boolean Type

- *Masking*—A bitwise logical expression has a Boolean result; each of its bits is the result of one or more logical operations on the corresponding bits of the operands.
- For binary arithmetic operators, and for relational operators:
  - If one operand is Boolean, the operation is performed with no conversion.
  - If both operands are Boolean, the operation is performed as if they were integers.
- No user-specified function can generate a Boolean result, although some (nonstandard) intrinsics can.
- Boolean and logical types differ as follows:
  - Variables, arrays, and functions can be of logical type, but they cannot be Boolean type.
  - There is a `LOGICAL` statement, but no `BOOLEAN` statement.
  - A logical variable or constant represents only one value. A Boolean constant can represent as many as 32 values.
  - A logical expression yields one value. A Boolean expression can yield as many as 32 values.
  - Logical entities are invalid in arithmetic, relational, or bitwise logical expressions. Boolean entities are valid in all three.

Alternate Forms of Boolean Constants

`f90` allows a Boolean constant (octal, hexadecimal, or Hollerith) in the following alternate forms (no binary). Variables cannot be declared Boolean. Standard Fortran does not allow these forms.

Octal

*ddddddd*B, where *d* is any octal digit

- You can use the letter B or b.
- There can be 1 to 11 octal digits (0 through 7).
- 11 octal digits represent a full 32-bit word, with the leftmost digit allowed to be 0, 1, 2, or 3.
- Each octal digit specifies three bit values.
- The last (rightmost) digit specifies the content of the rightmost three bit positions (bits 29, 30, and 31).
- If less than 11 digits are present, the value is right-justified—it represents the rightmost bits of a word: bits n through 31. The other bits are 0.
- Blanks are ignored.

Within an I/O format specification, the letter B indicates *binary* digits; elsewhere it indicates *octal* digits.

Hexadecimal

X'*ddd*' or X"*ddd*", where *d* is any hexadecimal digit

- There can be 1 to 8 hexadecimal digits (0 through 9, A-F).
- Any of the letters can be uppercase or lowercase (X, x, A-F, a-f).
- The digits must be enclosed in either apostrophes or quotes.
- Blanks are ignored.
- The hexadecimal digits may be preceded by a + or - sign.
- 8 hexadecimal digits represent a full 32-bit word and the binary equivalents correspond to the contents of each bit position in the 32-bit word.
- If less than 8 digits are present, the value is right-justified—it represents the rightmost bits of a word: bits n through 31. The other bits are 0.

Hollerith

*n*H...	'...'H	"..."H
*n*L...	'...'L	"..."L
*n*R...	'...'R	"..."R

Above, "..." is a string of characters and *n* is the character count.

- A Hollerith constant is type Boolean.
- If any character constant is in a bitwise logical expression, the expression is evaluated as Hollerith.
- A Hollerith constant can have 1 to 4 characters.

Examples: Octal and hexadecimal constants.

Boolean Constant	Internal Octal for 32-bit word
0B	00000000000
77740B	00000077740
X"ABE"	00000005276
X"-340"	37777776300
X'1 2 3'	00000000443
X'FFFFFFFFFFFFFFFF'	37777777777

Examples: Octal and hexadecimal in assignment statements.

```
i = 1357B
j = X"28FF"
k = X'-5A'
```

Use of an octal or hexadecimal constant in an arithmetic expression can produce undefined results and do not generate syntax errors.

Alternate Contexts of Boolean Constants

f90 allows BOZ constants in the places other than DATA statements.

B'*bbb*'	O'*ooo*'	Z'*zzz*'
B"*bbb*"	O"*ooo*"	Z"*zzz*"

If these are assigned to a real variable, no type conversion occurs.

Standard Fortran allows these only in DATA statements.

Abbreviated Size Notation for Numeric Data Types

f90 allows the following nonstandard type declaration forms in declaration statements, function statements, and IMPLICIT statements.

Table C-1 Size Notation for Numeric Data Types

Nonstandard	Declarator	Short Form	Meaning
INTEGER*1	INTEGER(KIND=1)	INTEGER(1)	One-byte signed integers
INTEGER*2	INTEGER(KIND=2)	INTEGER(2)	Two-byte signed integers
INTEGER*4	INTEGER(KIND=4)	INTEGER(4)	Four-byte signed integers
LOGICAL*1	LOGICAL(KIND=1)	LOGICAL(1)	One-byte logicals
LOGICAL*2	LOGICAL(KIND=2)	LOGICAL(2)	Two-byte logicals
LOGICAL*4	LOGICAL(KIND=4)	LOGICAL(4)	Four-byte logicals
REAL*4	REAL(KIND=4)	REAL(4)	IEEE single-precision floating-point (Four-byte)
REAL*8	REAL(KIND=8)	REAL(8)	IEEE double-precision floating-point (Eight-byte)
COMPLEX*8	COMPLEX(KIND=4)	COMPLEX(4)	Single-precision complex (Four-bytes each part)
COMPLEX*16	COMPLEX(KIND=8)	COMPLEX(8)	Double-precision complex (Eight-bytes each part)

The form in column one is nonstandard Fortran 90, though in common use. The kind numbers in column two can vary by vendor.

Note – For release 1.2 of f90, INTEGER with KIND= 1, 2, or 4, are each 4 bytes long and align on 4-byte boundaries.

Cray Pointers

A *Cray pointer* is a variable whose value is the address of another entity, which is called the *pointee*.

`f90` supports Cray pointers. Standard Fortran does not support them.

Syntax

The Cray `POINTER` statement has the following format:

```
POINTER  ( pointer_name, pointee_name [array_spec] ), …
```

Where *pointer_name*, *pointee_name*, and *array_spec* are as follows:

pointer_name	Pointer to the corresponding *pointee_name*. *pointer_name* contains the address of *pointee_name*. Must be: a scalar variable name (but not a structure) Cannot be: a constant, a name of a structure, an array, or a function
pointee_name	Pointee of the corresponding *pointer_name* Must be: a variable name, array declarator, or array name
array_spec	If *array_spec* is present, it must be explicit shape, (constant or nonconstant bounds), or assumed-size.

Example: Declare Cray pointers to two pointees.

```
    POINTER ( p, b ),  ( q, c )
```

The above example declares Cray pointer `p` and its pointee `b`, and Cray pointer `q` and its pointee `c`.

Example: Declare a Cray pointer to an array.

```
     POINTER ( ix, x(n, 0:m) )
```

The above example declares Cray pointer `ix` and its pointee `x`; and declares `x` to be an array of dimensions `n` by `m-1`.

Purpose of Cray Pointers

You can use pointers to access user-managed storage by dynamically associating variables to particular locations in a block of storage.

Cray pointers allow accessing absolute memory locations.

Cray pointers do not provide convenient manipulation of linked lists because (for optimization purposes) it is assumed that no two pointers have the same value.

Cray Pointers and Fortran Pointers

Cray pointers are declared as follows:

```
POINTER ( pointer_name, pointee_name [array_spec] )
```

Fortran pointers are declared as follows:

```
POINTER :: object_name
```

The two kinds of pointers cannot be mixed.

Features of Cray Pointers

- Whenever the pointee is referenced, `f90` uses the current value of the pointer as the address of the pointee.
- The Cray pointer type statement declares both the pointer and the pointee.
- The Cray pointer is of type Cray pointer.
- The value of a Cray pointer occupies one storage unit. Its range of values depends on the size of memory for the machine in use.
- The Cray pointer can appear in a `COMMON` list or as a dummy argument.
- The Cray pointee has no address until the value of the Cray pointer is defined.
- If an array is named as a pointee, it is called a *pointee array.*

  Its array declarator can appear in:
  - A separate type statement
  - A separate `DIMENSION` statement
  - The pointer statement itself

- If the array declarator is in a subprogram, the dimensioning can refer to:
  - Variables in a common block, or
  - Variables that are dummy arguments
- The size of each dimension is evaluated on entrance to the subprogram, not when the pointee is referenced.

Restrictions on Cray Pointers

- If *pointee_name* is of character type, it must be a variable typed `CHARACTER*(*)`.
- If *pointee_name* is an array declarator, it must be explicit shape, (constant or nonconstant bounds), or assumed-size.
- An array of Cray pointers is not allowed.
- A Cray pointer cannot be:
  - Pointed to by another Cray pointer or by a Fortran pointer.
  - A component of a structure.
  - Declared to be any other data type.
- A Cray pointer cannot appear in:
  - A `PARAMETER` statement or in a type declaration statement that includes the `PARAMETER` attribute.
  - A `DATA` statement.

Restrictions on Cray Pointees

- A Cray pointee cannot appear in a `SAVE`, `DATA`, `EQUIVALENCE`, `COMMON`, or `PARAMETER` statement.
- A Cray pointee cannot be a dummy argument.
- A Cray pointee cannot be a function value.
- A Cray pointee cannot be a structure or a structure component.
- A Cray pointee cannot be of a derived type.

Note – Cray pointees can be of type character, but their Cray pointers are different from other Cray pointers. The two kinds cannot be mixed in the same expression.

Usage of Cray Pointers

Cray pointers can be assigned values as follows:

- Set to an absolute address

  Example: `q = 0`

- Assigned to or from integer variables, plus or minus expressions

  Example: `p = q + 100`

- Cray pointers are not integers. You cannot assign them to a real variable.
- The `LOC` function (nonstandard) can be used to define a Cray pointer.

  Example: `p = LOC( x )`

Example: Use Cray pointers as described above.

```
SUBROUTINE  sub ( n )
COMMON pool(100000)
INTEGER blk(128), word64
REAL a(1000), b(n), c(100000-n-1000)
POINTER ( pblk, blk ), (ia, a ), ( ib, b ), &
       ( ic, c ), ( address, word64 )
DATA address / 64 /
pblk = 0
ia = LOC( pool )
ib = ia + 1000
ic = ib + n
...
```

Remarks about the above example:

- `word64` refers to the contents of absolute address 64
- `blk` is an array that occupies the first 128 words of memory
- `a` is an array of length 1000 located in blank common
- `b` follows `a` and is of length `n`
- `c` follows `b`
- `a`, `b`, and `c` are associated with `pool`
- `word64` is the same as `blk(17)` because Cray pointers are byte address and the integer elements of `blk` are each 4 bytes long

Optimization and Cray Pointers

For purposes of optimization, `f90` assumes the storage of a pointee is never overlaid on the storage of another variable—it assumes that a pointee is not associated with another variable.

Such association could occur in either of two ways:

- A Cray pointer has two pointees, or
- Two Cray pointers are given the same value

Note – The programmer responsible for preventing such association.

These kinds of association are sometimes done deliberately, such as for equivalencing arrays, but then results can differ depending on whether optimization is turned on or off.

Example: `b` and `c` have the same pointer.

```
POINTER ( p, b ),  ( p, c )
REAL x, b, c
p = LOC( x )
b = 1.0
c = 2.0
PRINT *, b
...
```

Above, because `b` and `c` have the same pointer, assigning 2.0 to `c` gives the same value to `b`. Therefore `b` prints out as 2.0, even though it was assigned 1.0.

Cray Character Pointers

If a pointee is declared as a character type, its Cray pointer is a Cray character pointer.

Purpose of Cray Character Pointers

A Cray character pointer is a special data type that allows `f90` to maintain character strings by keeping track of the following:

- Byte address of the first character of the string
- Length
- Offset

An assignment to a Cray character pointer alters all three. That is, when you change what it points to, all three change.

Declaration of Cray Character Pointers

For a pointee that has been declared with an assumed length character type, the Cray pointer declaration statement declares the pointer to be a Cray character pointer.

1. **Before the Cray pointer declaration statement, declare the pointee as a character type with an assumed length.**
2. **Declare a Cray pointer to that pointee.**
3. **Assign a value to the Cray character pointer.**

You can use functions `CLOC` or `FCD`, both nonstandard intrinsics.

Example: Declare `Ccp` to be a Cray character pointer and use `CLOC` to make it point to character string `s`.

```
CHARACTER*(*) a
POINTER ( Ccp, a )
CHARACTER*80  :: s = "abcdefgskooterwxyz"
Ccp = CLOC( s )
```

Operations on Cray Character Pointers

You can do the following operations with Cray character pointers:

```
Ccp1 + i
Ccp1 - i
i + Ccp1
Ccp1 = Ccp2
Ccp1 relational_operator Ccp2
```

where `Ccp1` and `Ccp2` are Cray character pointers and `i` is an integer.

Restrictions on Cray Character Pointers and Pointees

All restrictions to Cray pointers also apply to Cray character pointers. In addition, the following apply:

- A Cray character pointee cannot be an array.
- In a relational operation, a Cray character pointer can be mixed with only another Cray character pointer—not with a Cray pointer, not with an integer.
- A relational operation applies only to the character address and the bit offset; the length field is not involved.
- Cray character pointers must not appear in `EQUIVALENCE` statements, or any storage association statements. (The size can vary with the platform.)
- Cray character pointers are not optimized.
- Code containing Cray character pointers is not parallelized.
- A Cray character pointer in a list of an I/O statement is treated as an integer.

Intrinsics

`f90` supports some intrinsic procedures which are extensions beyond the standard.

Table C-2 Nonstandard Intrinsics

Name	Definition	Type Function	Arguments	Arguments	Remark	Notes
`CLOC`	Get Fortran character descriptor (`FCD`)	Cray character pointer	character	`([C=]`*c*`)`		NP, I
`COT`	Cotangent	real	real	`([X=]`*x*`)`		P, E
`DDIM`	Positive difference	double precision	double precision	`([X=]`*x*`,[Y=]`*y*`)`		P, E
`FCD`	Create Cray character pointer in Fortran character descriptor `(FCD)` format	Cray pointer	i: integer or Cray pointer j: integer	`([I=]`*i*`,[J=]`*j*`)`	i: word address of first character j: character length	NP, I
`LEADZ`	Get the number of leading 0 bits	integer	Boolean, integer, real, or pointer	`([I=]`*i*`)`		NP, I
`POPCNT`	Get the number of set bits	integer	Boolean, integer, real, or pointer	`([I=]`*i*`)`		NP, I
`POPPAR`	Calculate bit population parity	integer	Boolean, integer, real, or pointer	`([X=]`*x*`)`		NP, I

Notes on the above table:

Note	Meaning
P	The name can be passed as an argument.
NP	The name cannot be passed as an argument.
E	External code for the intrinsic is called at run time.
I	`f90` generates inline code for the intrinsic procedure.

C

Directives

A compiler *directive* directs the compiler to do some special action. Directives are also called *pragmas.*

A compiler directive is inserted into the source program as one or more lines of text. Each line looks like a comment, but has additional characters that identify it as more than a comment for this compiler. For most other compilers, it is treated as a comment, so there is some code portability.

General Directives

Currently there are only two general directives, FREE and FIXED. These directives tell the compiler to assume free-form source or fixed-form source.

Some other parallel directives are included which are not described in detail because they are *not* guaranteed to be in the next release.

Table C-3 General Directives Guaranteed Only in the Current Release

Directive
TASK, NOTASK
SUPPRESS(*var1*, *var2*, …)
TASKCOMMON(*cb1*, *cb2*, …)

Form of General Directive Lines

General directives have the following syntax.

```
!DIR$ d1, d2, …
```

A general *directive line* is defined as follows.

- A directive line starts with the 5 characters `CDIR$` or `!DIR$`, followed by:
  - A space
  - A directive
- Spaces before, after, or within a directive are ignored.
- Letters of a directive line can be in uppercase, lowercase, or mixed.

The form varies for fixed-form and free-form source as follows.

Fixed-Form Source

- Put `CDIR$` or `!DIR$` in columns 1 through 5.
- Directives are listed in columns 7 and beyond.
- Columns beyond 72 are ignored.
- An *initial* directive line has a blank in column 6.
- A *continuation* directive line has a nonblank in column 6.

Free-Form Source

- Put `!DIR$` followed by a space anywhere in the line.
  The `!DIR$` characters are the first nonblank characters in the line (actually, non-whitespace).
- Directives are listed after the space.
- An *initial* directive line has a blank, tab, or newline in the position immediately after the `!DIR$`.
- A *continuation* directive line has a character other than a blank, tab, or newline in the position immediately after the `!DIR$`.

Thus, `!DIR$` in columns 1 through 5 works for both free-form source and fixed-form source.

FIXED *and* FREE *Directives*

These directives specify the source form of lines following the directive line.

Scope

They apply to the rest of the *file* in which they appear, or until the next FREE or FIXED directive is encountered.

Uses

- They allow you to switch source forms within a source file.
- They allow you to switch source forms for an INCLUDE file. You insert the directive at the start of the INCLUDE file. After the INCLUDE file has been processed, the source form reverts back to the form being used prior to processing the INCLUDE file.

Restrictions

The FREE/FIXED directives:

- Each must appear alone on a compiler directive line (not continued).
- Each can appear anywhere in your source code. Other directives must appear within the program unit they affect.

Example: A FREE directive.

```
!DIR$ FREE
    DO i = 1, n
        a(i) = b(i) * c(i)
    END DO
```

Parallelization Directives

A *parallelization* directive is a special comment that directs the compiler to attempt to parallelize the next DO loop. Currently there is only one parallel directive, DOALL.

The DOALL directive tells the compiler to parallelize the next loop it finds, if possible.

Some other parallel directives are included which are not described in detail because they are *not* guaranteed to be in the next release.

Table C-4 Parallel Directives Guaranteed Only in the Current Release

Directive
CASE, END CASE
PARALLEL, END PARALLEL
DO PARALLEL, END DO
GUARD, END GUARD

Form of Parallelization Directive Lines

Parallel directives have the following syntax.

```
!MIC$ DOALL [general parameters] [scheduling parameter]
```

A *parallelization directive line* is defined as follows.

- A parallel directive starts with the CMIC$ or !MIC$, followed by:
  - A space
  - A directive
  - For some directives, one or more parameters
- Spaces before, after, or within a directive are ignored.
- Letters of a parallelization directive line can be in uppercase, lowercase, or mixed.

The form varies for fixed-form and free-form source as follows.

Fixed

- Put `CMIC$` or `!MIC$` in columns 1 through 5.
- Directives are listed in columns 7 and beyond.
- Columns beyond 72 are ignored.
- An *initial* directive line has a blank in column 6.
- A *continuation* directive line has a nonblank in column 6.

Free

- Put `!MIC$` followed by a space anywhere in the line.
  The `!MIC$` characters are the first nonblank characters in the line (actually, non-whitespace).
- Directives are listed after the space.
- An *initial* directive line has a blank, tab, or newline in the position immediately after the `!MIC$`.
- A *continuation* directive line has a character other than a blank, tab, or newline in the position immediately after the `!MIC$`.

Thus, `!MIC$` in columns 1 through 5 works for both free and fixed.

Example: Directive with continuation lines (`DOALL` directive and parameters.)

```
C$PAR DOALL
!MIC$&   SHARED( a, b, c, n )
!MIC$&   PRIVATE( i )
    DO i = 1, n
        a(i) = b(i) * c(i)
    END DO
```

Example: Same directive and parameters, with *no* continuation lines.

```
C$PAR DOALL  SHARED( a, b, c, n )  PRIVATE( i )
    DO i = 1, n
        a(i) = b(i) * c(i)
    END DO
```

Compatibility with FORTRAN 77

Source

Standard-conforming Fortran 77 source code is compatible with Sun Fortran 90. Use of non-standard extensions, such as VMS Fortran features, are not compatible and may not compile with Sun Fortran 90.

However, this release of Fortran 90 (1.2) treats all source lines as if they were lowercase (except in quoted character strings. Unline `f77`, there is no `-U` option to force `f90` to be sensitive to both upper and lower case. This may present a problem when mixing `f77` and `f90` compiled routines. Since a routine compiled by `f90` will treat `CALL XyZ` the same as `CALL XYz`, and treat them both as if they were `CALL xyz`, care must be taken to rearrange the way these calls are made. A similar situation will exist when trying to define entry points in `f90` compiled routines that are diffentiated by case. The clue to potential problems would be the need to use `-U` with `f77`.

Executables

Libraries compiled and linked in FORTRAN 77 under Solaris 2.x run in the Fortran 4.2 environment.

Libraries

- Libraries (`.a`) and object files (`.o`) compiled and linked in FORTRAN 77 under Solaris 2.x are compatible with Fortran 4.2. You can check the `/usr/4lib` directory on your SunOS 5.*x* system for the `libF77.so.2.0` and `libV77.so.2.0` library files.

Example: f90 main and f77 subroutine.

```
demo% cat m.f90
CHARACTER*74 :: c = 'This is a test.'
   CALL echo1( c )
END
demo$ cat s.f
    SUBROUTINE echo1( a )
    CHARACTER*74 a
    PRINT*, a
    RETURN
    END
demo% f77 -c -silent s.f
demo% f90 m.f90 s.o
demo% a.out
 This is a test.
demo%
```

- The library libF77 is generally compatible with f90.

  Example: f90 main calls a routine from the libF77 library.

```
demo% cat tdtime.f90
        REAL e, dtime, t(2)
        e = dtime( t )
        DO i = 1, 10000
                k = k+1
        END DO
        e = dtime( t )
        PRINT *, 'elapsed:', e, ', user:', t(1), ', sys:', t(2)
        END
demo% f90 tdtime.f90
demo% a.out
 elapsed:6.405999884E-3, user:5.943499971E-3, sys:4.625000001E-4
demo%
```

  See dtime(3f).

I/O

`f77` and `f90` are generally I/O compatible for binary I/O, since `f90` links to the `f77` I/O compatibility library.

Such compatibility includes the following two situations:

- In the same program, you can write some records in `f90`, then read them in `f77`.
- An `f90` program can write a file. Then an `f77` program can read it.

The numbers read back in may or may not equal the numbers written out.

- Unformatted

  The numbers read back in do equal the numbers written out.

- Floating-point formatted

  The numbers read back in can be different from the numbers written out. This is caused by slightly different base conversion routines, or by different conventions for uppercase/lowercase, spaces, plus or minus signs, and so forth.

  Examples: 1.0e12, 1.0E12, 1.0E+12

- List-directed

  The numbers read back in can be different from the numbers written out. This can be caused by various layout conventions with commas, spaces, zeros, repeat factors, and so forth.

  Example: ’0.0’ as compared to ’.0’

  Example: ’ 7’ as compared to ’7’

  Example: ’3, 4, 5’ as compared to ’3 4 5’

  Example: ’3*0’ as compared to ’0 0 0’

  The above results are from: `integer::v(3)=(/0,0,0/); print *,v`

  Example: ’0.333333343’ as compared to ’0.333333’

  The above results are from `PRINT *, 1.0/3.0`

Intrinsics

The Fortran 90 standard supports the following new intrinsic functions that FORTRAN 77 does not have.

If you use one of these names in your program, you must add an `EXTERNAL` statement to make `f90` use your function rather than the intrinsic one.

ADJUSTL	LEN_TRIM	SELECTED_INT_KIND
ADJUSTR	MAXEPONENT	SELECTED_REAL_KIND
ALLOCATED	MINEXPONENT	SET_EXPONENT
ASSOCIATED	NEAREST	SHAPE
BIT_SIZE	PRECISION	SIZE
DIGITS	PRESENT	SPACING
EPSILON	RADIX	TINY
EXPONENT	RANGE	TRANSFER
FRACTION	REPEAT	TRIM
HUGE	RRSPACING	UBOUND
KIND	SCALE	VERIFY
LBOUND	SCAN	

The Fortran 90 standard supports the following new array intrinsic functions.

ALL	MAXLOC	RESHAPE
ANY	MAXVAL	SPREAD
COUNT	MERGE	SUM
CSHIFT	MINLOC	TRANSPOSE
DOT_PRODUCT	MINVAL	UNPACK
EOSHIFT	PACK	
MATMUL	PRODUCT	

Forward Compatibility

The next release of `f90` is intended to be source code compatible with this release.

However, any libraries created with this release of `f90`, are not guaranteed to be compatible with the next release.

Mixing Languages

On Solaris systems, routines written in C can be combined with Fortran programs, since these languages have common calling conventions.

Module Files

Compiling a file containing a Fortran 90 `MODULE` generates a module file (`.M` file) in addition to the `.o` file.

By default, such files are usually sought in the current working directory. The `-M`*dir* option allows you to tell `f90` to seek them in an additional location.

The `.M` files cannot be stored into an archive file. If you have many `.M` files in some directory, and you want to reduce the number of such files (to reduce clutter), you can concatenate them into one large `.M` file.

C

Localization Support D

Support for languages other than English is described in this Appendix.

Native Language Support

This version of Fortran supports the development of applications using languages other than English, including most European languages. As a result, you can switch the development of applications from one native language to another.

This Fortran compiler implements internationalization as follows:

- It recognizes 8-bit characters from European keyboards supported by Sun.
- It is 8-bit clean and allows the printing of your own messages in the native language.
- It allows native language characters in comments, strings, and data.
- It allows you to localize the compile-time error messages files.

Locale

You can enable changing your application from one native language to another by setting the locale. Doing so changes some aspects of displays, such as date and time formats.

For information on this and other native language support features, read Chapter 6, “Native Language Application Support,” of the *System Services Overview* for Solaris software.

Even though some aspects can change if you set the locale, certain aspects cannot change. An internationalized compiler language does not allow input and output in the various international formats. If it does, it does not comply with the language standard appropriate for its language. For example, some languages have standards that specify a period (.) as the decimal unit in the floating-point representation.

Example: No I/O in international formats:

```
    PROGRAM sample
    REAL r
    r = 1.2
    WRITE( 6, 1 ) r
1   FORMAT( 1X F10.5 )
    END
```

Here is the output:

```
    1.20000
```

In the example above, if you reset your system locale to, say, France, and rerun the program, you still receive the same output. The period is not replaced with a comma, the French decimal unit.

Compile-Time Error Messages

The compile-time error messages are on files called source catalogs so you can edit them. You may decide to translate them to a local language such as French or Italian. Usually, a third party does the translating. Then you can make the results available to all local users of `f77/f90`. Each user of `f77/f90` can choose to use these files or not.

Localizing and Installing the Files

Usually a system administrator does the installation. It is generally done only once per language for the whole system, rather than once for each user of `f77/f90`. The results are available to all users of `f77/f90` on the system.

1. **Find the message text files.**
   The file names are:

   - `SUNW_SPRO_SC_f77pass1.msg`
   - `SUNW_SPRO_SC_driver.msg`)
   - `SUNW_SPRO_SC_f90_driver.msg`

2. **Edit the message text files.**

   a. **Make backup copies of the files.**

   b. **In whatever editor you are comfortable with, edit the files.**
   The editor can be `vi`, `emacs`, `textedit`, and so forth.

   Preserve any existing format directives, such as `%f`, `%d`, `%s`, and so forth.

   c. **Save the files.**

3. **Generate the message database catalogs from the message text files.**
   The compiler uses only the formatted message database catalogs. Run the `gencat` program to create the database files.

   a. **Generate the** `SUNW_SPRO_SC_f77pass1.cat` **message database from the** `SUNW_SPRO_SC_f77pass1.msg` **text file using** `gencat`:

```
demo% gencat SUNW_SPRO_SC_f77pass1.cat \
                                SUNW_SPRO_SC_f77pass1.msg
```

   Do this for each changed .msg file.

4. **Make the message database catalobs available to the general user.**
   Either put the catalogs into the standard location or put the path for them into the environment variable `NLSPATH`.

   a. **Define the standard location and name.**
   Put the files into the directory indicated:

   `/opt/SUNWspro/lib/locale/`*lang*`/LC_MESSAGES/`

where *lang* is the directory for the particular (natural) language. For example, the value of *lang* for Italian is `it`.

b. Set up the environment variable.
Put the path for the new files into the `NLSPATH` environment variable. For example, if your files are in `/usr/local/MyMessDir/`, then use the following commands.

In a `sh` shell:

```
demo$ NLSPATH=/usr/local/MyMessDir/%N.cat
demo$ export NLSPATH
```

In a `csh` shell:

```
demo% setenv NLSPATH  /usr/local/MyMessDir/%N.cat
```

The `NLSPATH` variable is standard for the X/Open environment. For more information, read the X/Open documents. See also `gencat`(1) and `catgets`(3C) for more information on message catalogs.

Using the File After Installation

You use the file by setting the environment variable `LC_MESSAGES`. This setup is generally done once for each developer.

Example: Set the environment variable `LC_MESSAGES`, assuming standard install locations, and the messages are localized in Italian:

In a `sh` shell:

```
demo$ LC_MESSAGES=it
demo$ export LC_MESSAGES
```

In a `csh` shell:

```
demo% setenv  LC_MESSAGES it
```

Index

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X