Pro Fortran

Linux

Fortran User Guide

absoft
development tools and languages

Pro Fortran

Linux

Fortran User Guide

absoft
development tools and languages
2781 Bond Street
Rochester Hills, MI 48309
U.S.A.
Tel (248) 853-0095
Fax (248) 853-0108
support@absoft.com

8.0073102

Fortran User Guide

Contents

CHAPTER 1

Introduction

INTRODUCTION TO ABSOFT PRO FORTRAN

Absoft specializes in the development of Fortran compilers and related tools. Full implementations of Fortran 77 and Fortran 90/95 are available for Macintosh, Windows, and Linux platforms. Absoft will continue to focus on Fortran in the future, but the popularity of C/C++ in the Unix environment has required many of today's Fortran programmers who are moving code to their desktop, to link Fortran code with C libraries. Absoft compilers support most popular inter-language calling conventions implemented on Linux systems, providing compatibility with existing libraries and object files, simplifying porting efforts.

This User Guide explains the operation of Absoft Fortran 90/95, Absoft FORTRAN 77, and the Fx™ debugger on the Linux operating system for the x86 and PowerPC families of processors. In the event you have licensed only one of these compilers, please refer only to the appropriate section(s) and disregard the others. All compilers operate in a similar manner, share a common tool set, and are link compatible. A brief summary of each compiler appears below.

Absoft Fortran 90/95

A complete, optimizing ANSI Fortran 90/95 implementation with extensions. Absoft Fortran 90/95 is the result of a five-year joint development effort with Cray Research. It utilizes a version of the CF90 front-end and is source compatible with several Cray F90 releases. It provides full support for interfacing with FORTRAN 77 and C Programming Language libraries.

Absoft FORTRAN 77

Refined over 16 years, with emphasis on porting legacy code from workstations. Absoft Fortran 77 is full ANSI 77 incorporating MIL-STD-1753, Cray-style POINTERs, plus most extensions from VAX FORTRAN as well as many from IBM, Sun, HP, and Cray. Absoft Fortran 77 supports legacy extensions that are not part of the Fortran 90/95 standard. See the chapter on **Porting Code** in this manual for further information. Fortran 77 is fully link compatible with Fortran 90/95 and C/C++ so existing, extended FORTRAN 77 routines can be easily compiled and linked with new Fortran 90/95 or C/C++ code.

CONVENTIONS USED IN THIS MANUAL

There are a few typographic and syntactic conventions used throughout this manual for clarity.

- `[]` square brackets indicate that a syntactic item is optional.
- ... indicates a repetition of a syntactic element.
- Term definitions are underlined.
- **-option** font indicates a compiler option.
- *Italics* are used for emphasis and book titles.
- Unless otherwise indicated, all numbers are in decimal form.
- FORTRAN examples appear in the following form:

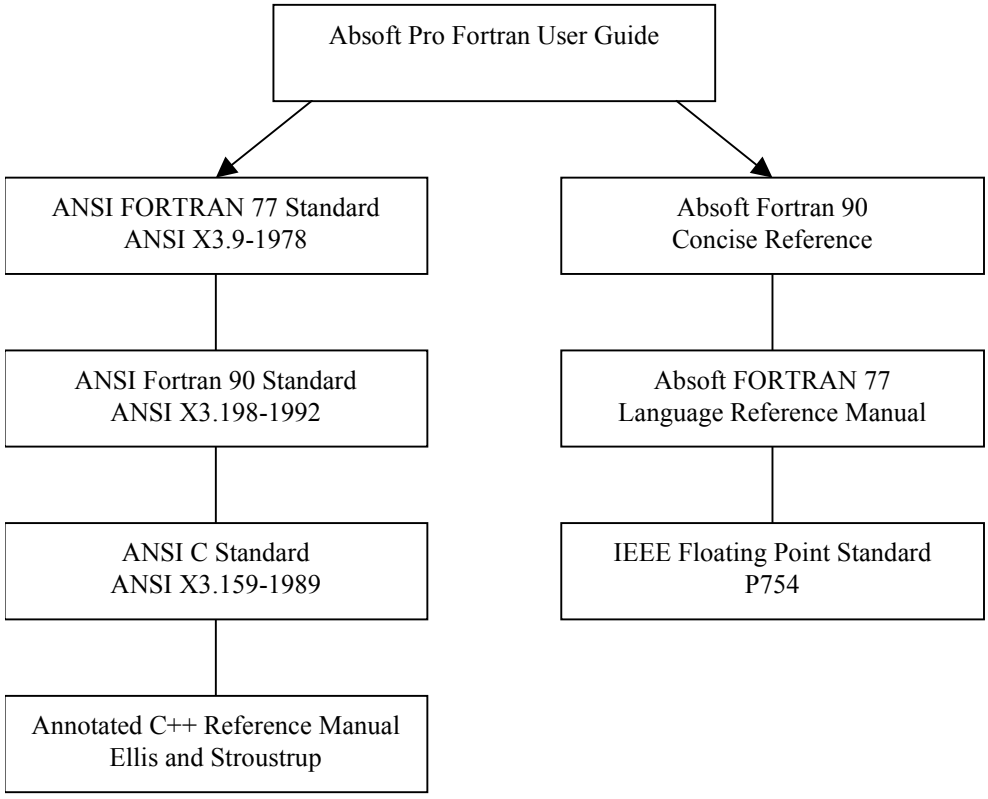
```
PROGRAM SAMPLE
WRITE (9,*) "Hello World!"
END
```

ROAD MAPS

Although this manual contains all the information needed to build programs with Absoft Pro Fortran on Linux, there are a number of other manuals that describe Fortran 90/95 and FORTRAN 77 in further detail. The *road map* in this chapter will guide you to these manuals for introductory or advanced reference. The bibliography in appendices lists further information about each manual.

Fortran Road Maps

The Absoft implementation of Fortran 90/95 is detailed in the online manual, *Fortran 90 Concise Reference*, in the Documentation directory of the Pro Fortran CDROM. FORTRAN 77 is detailed in the online manual, *FORTRAN 77 Language Reference Manual*, also in the Documentation directory of the Pro Fortran CDROM. A discussion of floating point precision is at the end of the chapter **Porting Code**. Figures 1-1 shows additional manuals that can be used for referencing the FORTRAN language and internal math operations.



FORTRAN 77 language road map
Figure 1-1

YEAR 2000 PROBLEM

All versions of Absoft Pro Fortran products for Macintosh, Power Macintosh, Windows 95/98, Windows NT, Linux, and UNIX will operate correctly across the date transition to the year 2000. Neither the compilers nor the runtime libraries have ever used 2-digit years in their internal operation. This means the version of Absoft Pro Fortran that you already have will continue to operate correctly. No patches or version updates are required.

The only caveat may be for those porting code from VAX/VMS systems. Since the early 1980s, Absoft Pro Fortran products have included software libraries designed to facilitate porting code from the VAX/VMS environment. Included in these VAX compatibility libraries are two subroutines that emulate the VAX/VMS DATE and IDATE subroutines. These subroutines return the year using a two-digit format. If you use DATE or IDATE in a program that stores or compares dates, you may need to recode portions of your application. Below are listed some of the alternatives supplied with Pro Fortran:

Fortran 90/95 `DATE_AND_TIME` Subroutine

This subroutine is part of the Fortran 90/95 language and returns integer data from the date and real time clock. Refer to the *Fortran 90 Concise Reference* for further information.

Unix Compatibility Library

There are a number of subroutines in the Unix Compatibility Library that return the date and time in both `INTEGER` and `CHARACTER` format. Refer to the manual **Absoft Compatibility Libraries** for information on their format and use.

CHAPTER 2

Using the Compilers

This chapter describes how to use the Absoft Fortran 90/95 and FORTRAN 77 compilers to create executable files on the Linux operating system for the Intel, AMD, and PowerPC families of processors. Beginning with an overview of the compilers, this chapter explains how to compile a small number of Fortran source files into an executable application. File name conventions and process control options are described first. The final sections of this chapter describe the compiler options in detail.

COMPILING PROGRAMS

The Fortran 90/95 and the FORTRAN 77 compilers are invoked from the Linux command line in the same manner:

```
f95 [options] files…

f77 [options] files…
```

FILE NAME CONVENTIONS

Compilation is controlled by the two compiler drivers: `f77` and `f95`. These drivers take a collection of files and, by default, produce an executable output file. Acceptable inputs to `f95` are:

File Type	Default form
Free format Fortran 90/95 source files	*file*`.f90 or` *file*`.f95`
Free format Fortran 90/95 preprocessor files	*file*`.F90 or` *file*`.F95`
Fixed format Fortran 90/95 source files	*file*`.f`
Fixed format Fortran 90/95 preprocessor files	*file*`.F`
C language source files	*file*`.c`
Assembly language source files	*file*`.s`
Relocatable object files	*file*`.o`

Acceptable inputs to `f77` are:

File Type	Default form
FORTRAN 77 source files	*file*`.f or` *file*`.for`
FORTRAN 77 preprocessor files	*file*`.F or` *file*`.FOR`
C language source files	*file*`.c`
Assembly language source files	*file*`.s`
Relocatable object files	*file*`.o`

File names that do not have one of these default forms are passed to the linker. It is assumed that the C compiler (*cc*), assembler (*as*), and linker (ld) are installed on the system and use standard command line syntax.

Output file names take the form:

File Type	Default form
Assembly language source files	*file*.s
Relocatable object files	*file*.o
Precompiled module file	*file*.mod
Executable object files	a.out

COMPILER PROCESS CONTROL

By default the f77 and f95 compiler drivers construct and execute the necessary commands to produce an executable application. This process requires compilation, assembly and linkage. As each of these processes finishes, all files that were created by the preceding stage are deleted. In some cases it may be desirable to save these intermediate files. Options controlling this are described here. These switches, in conjunction with the input file names, can also be used to stop the compilation process at any stage.

Generate Assembly Language (-S)

Specifying the **–S** option will cause the compilers to generate assembly language output in a form suitable for the system assembler. The file created will have the suffix ".s". For example, compiling test.f with the **–S** option will create test.s. If any C source files are given as arguments to f77 or f95, this option will be passed to the C compiler. If no other compiler process control options are specified and there are no relocatable object files specified on the command line, the compilation process will halt after all Fortran 90/95, FORTRAN 77, and any C source code files have been compiled to assembly language source.

Generate Relocatable Object (-c)

Specifying the **–c** option will cause the compilers to generate relocatable object files. In the Linux environment, this option indicates that all source files (Fortran 90/95, FORTRAN 77, C, and assembly) should be processed to relocatable object files. If no linker options are present (see below), then the compilation process stops after all object files have been created. If any C source files are given as arguments to f77 or f95, this option will be passed to the C compiler.

Passing Options To The Linker

For ease of use within the Linux environment, many of the options that are available to the system linker are also available to the `f77` and `f95` compiler drivers. Specifying any of these options indicates that all files specified on the command line should be processed through the linkage phase. Unless the **–S** or **–c** options are specified, all intermediate files (relocatable objects and/or assembly source) will be deleted. See the system documentation on *ld* for more information regarding these options. In brief, the options are as follows:

Executable File Name (-o *name*)

Use of the **–o *name*** option will cause the linker to produce an executable file called ***name***. The default is to produce an executable file called `a.out`.

Library Specification (-l)

Specifying the **–l***name* option will cause the linker to search the library file `lib`*name*`.a`.

Library Path Specification (-L)

The **–L***path* option will cause the linker to search the specified directory named in *path* for library files given with succeeding **–l** options.

Undefine A Symbol (-u)

Specifying the **–u***symbol_name* option will enter *symbol_name* as an undefined symbol to the linker.

Linker Options (-X)

Use the **–X*option*** switch to pass an option directly to the linker. The FORTRAN 77 or Fortran 90/95 driver will pass ***option*** to the linker. If you want to pass an options which takes an argument, use the **–X** option twice.

Generate Debugging Information (-g)

Specifying the **–g** option will cause the compilers to include symbol and line information appropriate for debugging a compiled program with Fx, the Absoft debugger.

The Absoft Fortran 90/95 and FORTRAN 77 compilers have the capability to output special symbol information for use with the Fx debugger from Absoft. This information allows Fx to display the contents of adjustable arrays, arrays with more than four dimensions, arrays with lower bounds other than 1, and arrays with dimensions greater than 32767.

Enable Exception Traceback (-et)

The **–et** option causes the compilers to include symbol and line information, exception handling initialization, and library to code to perform execution tracebacks. The traceback includes file name and line number of the program units in the call tree to the point of the exception. There is no program execution time overhead when enabling this option, but all files that are incorporated in the executable must be compiled with this option for the diagnostic output to be meaningful.

g77 Compatibility (-g77)

Use the **–g77** option to enable compiler switches that produce g77 compatible object code. These options are:

f90	f77
-YEST_NAMES=LCS	-f
-s	-s
-YCFRL=1	-N90
-B108	-B108

This option is useful when linking against libraries built with g77.

FPU CONTROL OPTIONS

These options provide control over several aspects of the operation of the *Floating-Point Unit* of the processor including rounding mode, exception handling, control word state, and FPU stack integrity.

FPU Rounding Mode

The FPU rounding method is controlled with the **–round** option:

–round=*mode*

where ***mode*** is one of:

NEAREST
DOWN
UP
TOZERO

This option implicitly enables the **Preserve FPU control word (-B24)** floating-point control word option (see below).

FPU Exception Handling

When a floating-point exception is produced, the default action of an application is to supply an IEEE P754 defined value and continue. For undefined or illegal operations (such as divide by zero or square root of a negative number) this value will usually be either Infinity (INF) or Not A Number (NaN) depending on the floating-point operation.

Checking any of the exception boxes will cause the program to stop and produce a core dump, rather than continue, if the exception is encountered. If the program is being debugged, it will stop in the debugger at the statement line that caused the exception. The syntax for using this option on the command line is:

-trap=*exception*[,*exception*,…]

where ***exception*** is one or more of:

INVALID
DIVBYZERO
OVERFLOW
UNDERFLOW
INEXACT
ALL

This option implicitly enables the **Preserve FPU control word (-B24)** floating-point control word option (see below).

X86 PROCESSOR SPECIFIC OPTIONS

The options described in this section are specific to the x86 family of processors.

CPU Specific Optimizations (-cpu:*type*)

Use the **–cpu:*type*** option to generate instructions specific to a particular processor. The recognized ***type*** arguments are:

486	non-Pentium Intel processor
p5	Pentium
p6	Pentium Pro, Pentium II, and Pentium III
p7	Pentium 4
athlon	AMD Athlon and Duron
host	automatically establishes ***type*** based on the processor in the machine that the program is compiled with. If the CPU type cannot be determined, **p5** is assumed.

No Register Variables (+B41)

If optimization is enabled, some `REAL*4` variables may be assigned to registers and maintained there in double precision. This extra precision may cause problems in some numerically sensitive programs. To overcome this, add the **+B41** option *after* the optimization option to prevent the assignment of floating point variables to registers.

Don't change FPU control word (-B23)

If your code requires that the compiler not generate any instructions to manipulate the FPU control word on Intel and AMD processors, select the **-B23** option. Refer to the **-B24** option (**Preserve FPU control word**) described next for details.

Preserve FPU control word (-B24)

In order to insure correct floating point to integer conversions on Intel and AMD hardware, the compiler generates instructions to set the rounding mode to truncate before generating code for these conversions. It does not, however, preserve any pre-existing state of the FPU control word. If your code requires preserving the state of the FPU control word (perhaps after setting it to some application specific state), use the **-B24** option to direct the compiler to save and restore its state around floating point to integer conversions. Note that this is an Intel and AMD specific option.

Verify FPU Stack (-B111)

The **–B111** option directs the compiler to generate special code to verify the contents of the Intel or AMD FPU after each subroutine and function reference. This option is useful for tracking down mistyped functions and functions that are incorrectly referenced in subroutine `CALL` statements.

POWERPC PROCESSOR SPECIFIC OPTIONS

The options described in this section are specific to the PowerPC family of processors.

Don’t generate FMA instructions (+B51)

Use of the **+B51** option will cause the compiler not to use floating-point multiply-add type of instructions. Since there is no rounding performed between the multiplication and addition during the execution of these instructions, numeric results will vary depending on where they are used.

Use long branches (-B18)

The program counter relative branch instructions of the PowerPC microprocessor are limited to signed 16-bit offsets. By default, the Absoft compilers issue these single instruction branches within program units. In some instances of large source files, the range of this branch instruction may be exceeded, resulting in an error diagnostic being issued by the assembler.

The **-B18** compiler option should be used to overcome the limitation. This option should only be used when the assembler issues diagnostics as it will cause more code to be generated than is usually necessary.

ABSOFT FORTRAN 90/95 OPTIONS

The compiler options detailed in this section give you a great deal of control over the compilation and execution of Fortran 90/95 programs. The options fall into five general categories: Compiler Control, Optimizations, Compatibility, Modules paths and file, and Miscellaneous.

Each option is listed with the corresponding option letter(s) and a description. Options that take arguments may optionally have a space to separate the option from its argument. The only exceptions are the `B` and `N` options; they cannot have a space between the option and its argument (e.g. `-N33`).

Many of these options are also discussed in the *Absoft Fortran 90 Concise Reference* on your CD-ROM. Refer to Chapter 12, **A Fortran 90 Implementation**.

Compiler control

These options control various aspects of the compilation process such as warning level, verbosity, code generation, where module files can be found, and the definition of compiler directive variables. The generation of debugging information, for the symbolic source-level debugger, Fx, is also controlled by compiler control options.

Show progress (-v)

Enabling the **-v** option will cause the `f95` command, described above, to display the commands it is sending to the compiler, assembler, and linker.

Output Version number (-V)

The **-V** option will cause the `f95` compiler to display its version number. This option may be used with or without other arguments.

Suppress warnings (-w)

Suppresses the listing of warning messages. For example, unreachable code will generate a warning message. Error diagnostics will still be displayed on standard error.

Warn of non-standard usage (-en)

Use of the **-en** option will cause the compiler to issue a warning whenever the source code contains an extension to the Fortran 90/95 standard. This option is useful for developing code that must be portable to other environments.

Warning level (-m*nn*)

Use the **-m*nn*** option to suppress messages by message level, where ***nn*** is a message level. Diagnostics issued at the various levels are:

0	errors, warnings, cautions, notes, comments
1	errors, warnings, cautions, notes
2	errors, warnings, cautions
3	errors, warnings
4	errors

The default level is **-m3**; the compiler will issue error and warning diagnostics, but not cautions, notes, and comments. See also the **-M*nn*** option.

Suppress Warning number(s) (-M*nn*)

Use the **-M*nn*** option to suppress messages by message number, where ***nn*** is a message number. This option is useful if the source code generates a large number of messages with the same message number, but you still want to see other messages. See also the **-m*nn*** option.

Stop on error (-ea)

The **-ea** option will cause the `f95` compiler to abort the compilation process on the first error that it encounters.

Allow greater than 100 errors (-dq)

Normally, the Absoft Fortran 90/95 compiler will stop if more than 100 errors are encountered. This many errors usually indicate a problem with the source file itself or the inability to locate an `INCLUDE` file. If you want the compiler to continue in this circumstance, select the **-dq** option.

Default Recursion (-eR)

If you select the **-eR** option, all `FUNCTION`s and `SUBROUTINE`s are given the `RECURSIVE` attribute. Normally, if the compiler detects a recursive invocation of a procedure not explicitly given the `RECURSIVE` attribute, a diagnostic message will be issued. The **-eR** option disables this.

Append Underscore To Names (-B108)

Use of the **–B108** option directs the compiler to append an underscore to `SUBROUTINE` and `FUNCTION` definitions and references. This option can be used to avoid name conflicts with the system libraries or other Fortran environments.

Disable Stack Alignment (-B112)

Use the **–B112** option to prevent the compiler from aligning the stack to a optimal sixteen-byte boundary at the start of a main program. Use of this option is likely to cause a compiled program to run slower.

Procedure Trace (-B80)

Specifying the **-B80** option will cause the compiler to generate code to write the name of the currently executing procedure to standard out. This option is useful for tracing program execution and quickly isolating execution problems.

Assume Pointer Aliases Exist (-B19)

The **-B19** option is selected when more than one symbolic name is used to reference a variable's memory location. This can occur when pointers are used, when variables in COMMON are also passed as arguments, or when two dummy arguments are the same actual argument.

Note: Standard FORTRAN should not require this option, but the use of extensions may dictate its use. Performance loss should be expected when this option is selected.

Generate Debugging Information (-g)

Specifying the **–g** option will cause the compilers to include symbol and line information appropriate for debugging a compiled program with Fx, the Absoft debugger.

The Absoft Fortran 90/95 and FORTRAN 77 compilers have the capability to output special symbol information for use with the Fx debugger from Absoft. This information allows Fx to display the contents of adjustable arrays, arrays with more than four dimensions, arrays with lower bounds other than 1, and arrays with dimensions greater than 32767.

Generate Profiler Information (-P)

Specifying the **–P** option will place information for profiling execution into a compiled program. For information on using the Linux profiler, see the Linux manual page for *gprof*.

Optimizations

These options control compile time optimizations to generate an application with code that executes more quickly. Absoft Fortran 90/95 is a globally optimizing compiler, so various optimizers can be turned on which affect single statements, groups of statements or entire programs. There are pros and cons when choosing optimizations; the application will execute much faster after compilation but the compilation speed itself will be slow. Some of the optimizations described below will benefit almost any Fortran code, while others should only be applied to specific situations.

Basic Optimizations (-O1)

The **–O1** option will cause most code to run faster and enables optimizations that do not rearrange your program. The optimizations include common subexpression elimination, constant propagation, and branch straightening. This option is generally usable with debugging options. **–cpu:host** is implied with this option.

Normal Optimizations (-O2)

The **–O2** option enables normal optimizers that can substantially rearrange the code generated for a program. The optimizations include strength reduction, loop invariant removal, code hoisting, and loop closure. This option is not usable with debugging options. **–cpu:host** is implied with this option.

Advanced Optimizations (-O3)

The **–O3** option enables advanced optimizers that can significantly rearrange and modify the code generated for a program. The optimizations include loop permutation (loop reordering), loop tiling (improved cache performance), loop skewing, loop reversal, unimodular transformations, forward substitution, and expression simplification. This option is not usable with debugging options. **–cpu:host** is implied with this option.

Compatibility

These options allow Absoft Fortran 90/95 to accept older or variant extensions of Fortran source code from other computers such as mainframes. Many of these can be used for increased compatibility with Fortran compilers on various mainframe computers.

Source Formats

For compatibility with other Fortran environments and to provide more flexibility, the compiler can be directed to accept source code that has been written in a number of different formats. The two basic formats are free-form and fixed-form.

Free-Form (-f free)

The **-f free** option instructs the compiler to accept source code written in the format for the Fortran 90/95 Free Source Form. This is the default for file names with an extension of "`.f90`" or "`.f95`".

Fixed-Form (-f fixed)

The **-f fixed** option instructs the compiler to accept source code written in the format for the Fortran 90/95 Fixed Source Form that is the same as the standard FORTRAN 77 source form.

Alternate Fixed form (-f alt_fixed)

The **-f alt_fixed** option instructs the compiler to accept source code written in following form:

If a tab appears in columns 1 through 5, then the compiler examines the next character. If the next character is not a letter (a-z, or A-Z) then it is considered a continuation character and normal rules apply. If it is a zero, a blank, another tab, or a letter, the line is not a continuation line.

Fixed line length (-W *nn*)

Use the **-W** option to set the line length of source statements accepted by the compiler in Fixed-Form source format. The default value of *nn* is 72. The other legal values for *nn* are 80 and 132 — any other value produces an error diagnostic.

Escape Sequences in Strings (-YCSLASH=1)

If the **-YCSLASH=1** option is turned on, the compiler will transform the following escape sequences marked with a '\' embedded in character constants:

```
\a                Audible Alarm (BEL, ASCII 07)
\b                Backspace (BS, ASCII 8)
\f                Form Feed (FF, ASCII 12)
\n                Newline (LF, ASCII 10)
\r                Carriage Return (CR, ASCII 13)
\t                Horizontal Tab (HT, ASCII 09)
\v                Vertical Tab (VT, ASCII 11)
\xh[h]            Hexadecimal, up to 2 digits
\o[o[o]]          Octal number, up to 3 digits
\\                Backslash
```

No Dot for Percent (-YNDFP=1)

This option instructs the compiler to disallow the use of a '.' (period) as a structure field component dereference operator. The default is to allow both '%' (percent), which is the Fortran 90/95 standard, and a period which is typically used with DEC style RECORD declarations. The use of a period may cause certain Fortran 90/95 conforming programs to be misinterpreted (a period is used to delineate user defined operators and some intrinsic operators). The default is **-YNDFP=0**. This switch implements Fortran 90/95 standard parsing for structure component referencing.

MS Fortran 77 Directives (-YMS7D)

The **-YMS7D** option causes the compiler to recognize Microsoft Fortran 77 style directives in the form of $directive where the dollar-sign character is in column one of the source file. directive must be from the set of supported MS directives.

DVF/CVF Compatible CHARACTER arguments (-YVF_CHAR)

The **–YVF_CHAR** option causes the compiler to pass and expect `CHARACTER` arguments in a manner compatible with Digital/Compaq Visual Fortran. The length of the argument (as a value) immediately follows the argument itself as opposed to the more common method of passing the length(s) at the end of the argument list. This is primarily a Windows™ compatibility option

Integer Sizes (-i2 and -i8)

Without an explicit length declaration, `INTEGER` data types default to thirty-two bits or four bytes (`KIND=4`). The **–i2** option can be used to change this default length to sixteen bits or two bytes (`KIND=2`). The **–i8** option can be used to change the default `INTEGER` size to 64 bits or 8 bytes (`KIND=8`). However, an explicit length specification in a type declaration statement always overrides the default data length.

Demote Double Precision to Real (-dp)

The **-dp** option will cause variables declared in a DOUBLE PRECISION statement and constants specified with the D exponent to be converted to the default real kind. Similarly, variables declared in a DOUBLE COMPLEX statement and complex constants specified with D exponents will be converted to the complex kind in which each part has the default real kind.

Promote REAL to REAL(KIND=8) (-N113)

Without an explicit length declaration, single precision REAL and COMPLEX data types default to thirty-two bits or four bytes (KIND=4) and sixty-four bits or eight bytes (KIND=4), respectively. The **-N113** option is used to promote these to their double precision equivalents (KIND=8). This option does not affect variables which appear in type statements with explicit sizes (such as REAL (KIND=4) or COMPLEX (KIND=4)).

One trip DO loops (-ej)

Fortran 90/95 requires that a DO loop not be executed if the iteration count, as established from the DO parameter list, is zero. The **-ej** option will cause all DO loops to be executed at least once, regardless of the initial value of the iteration count.

Static storage (-s)

The **-s** option is used to allocate local variables statically, even if SAVE was not specified as an attribute. In this way, they will retain their definition status on repeated references to the procedure that declared them. Two types of variables are not allocated to static storage: variables allocated in an ALLOCATE statement and local variables in recursive procedures.

Disable compiler directive (-x*directive*)

The **-x** option is used to disable compiler directive in the source file. ***directive*** may be any of the following:

ATTRIBUTES
FIXED
FIXEDFORMLINESIZE
FREE[FORM]
NAME
NOFREEFORM
PACK[ON]
PACKOFF
STATIC

See the section **Absoft Fortran 90/95 Compiler Directives** for more information on using compiler directives in your source code.

Max Internal Handle (-T *nn*)

This option is used to change the number of handles used internally by the compiler. Under most conditions, the default value of 100000 handles is sufficient to compile even extremely large programs. However, under certain circumstances, this value may be exceeded and the compiler will issue a diagnostic indicating that the value should be increased.

The default value can be increased by powers of ten by specifying the **-T** *nn*, where *nn* is a positive integer constant. When this option is specified, the number of handles will be 100000×10^{nn} bytes.

Temporary string size (-t *nn*)

In certain cases the compiler is unable to determine the amount of temporary string space that string operations will require. The compiler will assume that the operation in question will require 1024 bytes of temporary string space. This default value can be increased by powers of ten by specifying the **-t** *nn*, where *nn* is a positive integer constant. When this option is specified, the default temporary string size will be 1024×10^{nn} bytes.

Module File Path(s) (-p *path*)

The Absoft Fortran 90/95 compiler will automatically search the current directory for precompiled module files. If module files are maintained in other directories, the **–p *path*** option can be used to specify additional paths to be searched. If ***path*** specifies a directory name only, all module files in the directory will be searched. If ***path*** specifies a filename, only the specified file will be searched.

Check Array Boundaries (-Rb)

When the **–Rb** compiler option is turned on, code will be generated to check that array indexes are within the bounds of an array. Assumed size arrays whose last dimension is * cannot be checked. In addition, file names and source code line numbers will be displayed with all run time error messages.

Check Array Conformance (-Rc)

The **–Rc** compiler option is used to check array conformance. When array shapes are not known at compile time and where they must conform, runtime checks are created to insure that two arrays have the same shape.

Check Substrings (-Rs)

When the **–Rs** compiler option is turned on, code will be generated to check that character substring expressions do not specify a character index outside of the scope of the character variable or character array element.

Check Pointers (-Rp)

Use **–Rp** compiler option is used to generate additional program code to insure that Fortran 90 style POIINTER references are not null.

Character Argument Parameters (-YCFRL={0|1})

Use the **–YCFRL=1** option to force the compiler to pass `CHARACTER` arguments in a manner that is compatible with `g77` and `f2c` protocols. Use the **–YCFRL=0** option (the default) to pass `CHARACTER` arguments in a manner that is compatible with Absoft Compilers on other platforms. **Note:** this option should be used consistently on all files that will be linked together into the final application.

External Symbol Character Case (-YEXT_NAMES={ASIS | UCS | LCS})

The **-YEXT_NAMES** option is used to specify how the external name of globally visible symbols, such as `FUNCTION` and `SUBROUTINE` names, are emitted. By default, names are emitted entirely in upper case (-**YEXT_NAMES=UCS**). Set this option to **LCS** to emit names entirely in lower case. Set this option to **ASIS** to force external names to be emitted exactly as they appear in the source program. This option controls how external names will appear to other object files.

External Symbol Prefix (-YEXT_PFX=*string*)

The **-YEXT_PFX** option can be used to prepend a user specified ***string*** to the external representation of external procedure names.

External Symbol Suffix (-YEXT_SFX=*string*)

The **-YEXT_SFX** option can be used to append a user specified ***string*** to the external representation of external procedure names.

COMMON Block Name Character Case (-YCOM_NAMES={UCS | LCS})

The **-YCOM_NAMES** option is used to specify how the external names `COMMON` blocks are emitted. The default (-**YEXT_NAMES=UCS**) is to emit `COMMON` block names entirely in upper case. Set this option to **LCS** to emit names entirely in lower case.

COMMON Block Name Prefix (-YCOM_PFX=*string*)

The **-YEXT_PFX** option can be used to prepend a user specified ***string*** to the external representation of `COMMON` block names.

COMMON Block Name Suffix (-YCOM_SFX=*string*)

The **-YEXT_SFX** option can be used to append a user specified ***string*** to the external representation of `COMMON` block names.

Cache Control (-YDEALLOC= {MINE | ALL | CACHE})

This option is used to control the underlying runtime memory management associated with the Fortran 95 `ALLOCATE` and `DEALLOCATE` statements. By default the runtime caches memory which has been deallocated (**CACHE**). Specifying **MINE** will cause all user allocated memory to be returned via a call to `free(2)` when a call to `DEALLOCATE` is executed. Specifying **ALL** will cause all user allocated memory to be returned via a call to `free(2)` and return any compiler allocated memory that has been cached. The tradeoff is minimizing memory use (**ALL/MIME**) versus speed of execution (**CACHE**).

Pointers Equivalent to Integers (-YPEI={0|1})

This option controls whether or not the compiler will allow or accept a CRI style pointer to be equivalent to an integer argument. By default the Absoft Fortran 90/95 compiler allows this. Even with this relaxed error checking the compiler will correctly choose the right interface for the following example:

```
interface generic
  subroutine specific1(i)
      integer i
  end subroutine specific1
  subroutine specific2(p)
      integer i
      pointer (p,i)
  end subroutine specific2
end interface
call generic(i)
call generic(loc(i))
end
```

Regardless of the switch setting, this example will compile and the executable generated will be equivalent to:

```
call specific1(i)
call specific2(loc(i))
```

Absoft Fortran 90/95 Compiler Directives

Compiler directives are lines inserted into source code that specify actions to be performed by the compiler. They are not Fortran 90/95 statements. If you specify a compiler directive while running on a system that does not support that particular directive, the compiler ignores the directive and continues with compilation.

A *compiler directive line* begins with the characters CDIR$ or !DIR$. How you specify compiler directives depends on the source form you are using.

If you are using fixed source form, indicate a compiler directive line by placing the characters CDIR$ or !DIR$ in columns 1 through 5. If the compiler encounters a nonblank character in column 6, the line is assumed to be a compiler directive continuation line. Columns 7 and beyond can contain one or more compiler directives. If you are using the default 72 column width, characters beyond column 72 are ignored. If you have specified 80 column lines, characters beyond column 80 are ignored.

If you are using free source form, indicate a compiler directive line by placing the characters !DIR$ followed by a space, and then one or more compiler directives. If the position following the !DIR$ contains a character other than a blank, tab, or newline character, the line is assumed to be a compiler directive continuation line.

If you want to specify more than one compiler directive on a line, separate each directive with a comma.

NAME Directive

The NAME directive allows you to specify a case-sensitive external name in a Fortran program. You can use this directive, for example, when writing calls to C routines. The case-sensitive external name is specified on the NAME directive, in the following format:

```
!DIR$ NAME (fortran="external" [,fortran="external"]...)
```

where: *fortran* is the name used for the object throughout the Fortran program whenever the external name is referenced.

external is the external name.

FREE[FORM] Directive

The FREE or FREEFORM directive specifies that the source code in the program unit is written in the free source form. The FREE directive may appear anywhere within your source code. The format of the FREE directive is:

```
!DIR$ FREE
```

You can change source form within an INCLUDE file. After the INCLUDE file has been processed, the source form reverts back to the source form that was being used prior to processing the INCLUDE file.

FIXED Directive

The FIXED directive specifies that the source code in the program unit is written in the fixed source form. The FIXED directive may appear anywhere within your source code. The format of the FIXED directive is:

```
!DIR$ FIXED
```

You can change source form within an INCLUDE file. After the INCLUDE file has been processed, the source form reverts back to the source form that was being used prior to processing the INCLUDE file.

NOFREEFROM Directive

The NOFREEFORM directive is the same as the FIXED directive (see above) and specifies that the source code in the program unit is written in the fixed source form.

FIXEDFORMLINESIZE Directive

The FIXEDFORMLINESIZE directive specifies the line length for fixed-form source code. The format of the FIXEDFORMLINESIZE directive is:

```
!DIR$ FIXEDFORMLINESIZE:{72|80|132}
```

ATTRIBUTES Directive

The ATTTRIBUTES directive can be used to apply special attributes to simplify passing variables between Fortran 90/95 and other languages. The format of the ATTTRIBUTES directive is:

```
!DIR$ ATTTRIBUTES attr-list::sym-list
```

where: *attr-list* is a comma separated list of attributes from the following set.

```
ALIAS
C
REFERENCE
STDCALL
VALUE
```

sym-list is a comma separated list of symbols.

The ALIAS attribute takes the form of

```
ALIAS:external
```

where: *external* is the is the external name of the procedure.

PACK[ON] Directive

The PACK or PACKON directive specifies that sequenced structure fields be aligned on byte even byte or word (four-byte) boundaries. The default is 1 (byte). The format for this compiler directive is:

```
!DIR$ PACK [= {1|2|4}]
```

The packing directives affect the current program unit being compiled (if there is one), or the next program unit (when there is no current program unit). The packing directive is reset to the default (PACKOFF) after the end of each program unit. A packing directive affects only derived-types found below the directive in the source code.

PACKOFF Directive

The PACKOFF directive returns structure field alignment to the default for the machine architecture which is alignmed on the most efficient boundary for the data type. The format for this compiler directive is:

```
!DIR$ PACKOFF
```

STACK Directive

The STACK directive causes the default storage allocation to be the stack in the program unit that contains the directive. This directive overrides the **-s** command line option in specific program units of a compilation unit. The format for this compiler directive is:

```
!DIR$ STACK
```

ABSOFT FORTRAN 77 OPTIONS

The compiler options detailed in this section give you a great deal of control over the compilation and execution of FORTRAN programs. The options fall into three general categories: Compiler Control, Optimizations, and Compatibility.

Each option is listed with the corresponding option letter(s) and a short description. Options that take arguments (e.g. `-h 4` or `-o file`) must have a space to separate the option from the argument. The only exceptions are the `B` and `N` options; they do not have a space between the option and the argument (e.g. `-N33`).

Compiler control

These options control various aspects of the compilation process such warnings, verbosity, and definition of compiler directive variables. The generation of debugging information, for the symbolic source-level debugger, Fx, is also controlled by compiler control options.

Show progress (-v)

Enabling the **-v** option will cause the `f77` compiler driver, described earlier in this chapter, to display the commands it is sending to the compiler, assembler, and linker.

Quiet Compilation (-q)

The Absoft Fortran 77 compiler normally displays information to standard output as it compiles an application. Enabling the **-q** option will suppress any messages printed to standard output. Error and warning messages will still be printed to standard diagnostic, however.

Suppress warnings (-w)

Suppresses the listing of warning messages. For example, unreachable code and a missing label on a `FORMAT` statement generate warning messages. Compile time diagnostic messages are divided into two categories: errors and warnings. Error messages indicate that the compiler was unable to generate an output file. Warning messages indicate that some syntactic element was not appropriate, but the compiler was able to produce an output file.

Suppress alignment warnings (-A)

The compiler normally will issue a warning message if a variable is aligned on a boundary that does not match its size. Misaligned storage locations slow down memory access and can cause difficulty if you attempt to port the program to other computers. Use the **-A** option to suppress the listing of this type of warning only.

Warn of non-ANSI usage (-N32)

Use of the **-N32** option will cause the compiler to issue a warning whenever the source code contains an extension to the ANSI FORTRAN 77 standard (American National Standard Programming Language FORTRAN, X3.9-1978). This option is useful for developing code which must be portable to other environments.

Check Syntax Only (-N52)

The **–N52** option runs only the front end of the compiler. No object or executable files are created.

Append Underscore To Names (-B108)

Use of the **–B108** option directs the compiler to append an underscore to `SUBROUTINE` and `FUNCTION` definitions and references. This option can be used to avoid name conflicts with the system libraries or other Fortran environments.

Character Argument Parameters (-N90)

Use the **–N90** option to force the compiler to pass `CHARACTER` arguments in a manner that is compatible with `g77` and `f2c` protocols. The default is to pass `CHARACTER` arguments in a manner that is compatible with Absoft Compilers on other platforms.

Disable Stack Alignment (-B112)

Use the **–B112** option to prevent the compiler from aligning the stack to an optimal sixteen-byte boundary at the start of a main program. Use of this option is likely to cause a compiled program to run slower.

BLOCK DATA Code Section (-N116)

`BLOCK DATA` subprograms generate data initialization records only in the object file. Since they do not generate code references, they cannot effectively be used in libraries; the linker will not include them, as they resolve no references. Use the **–N116** option to force the compiler to generate empty, stub routines that will allow the linker to use the associated data initialization records.

Procedure Trace (-B80)

Specifying the **-B80** option will cause the compiler to generate code to write the name of the currently executing procedure to standard out. This option is useful for tracing program execution and quickly isolating execution problems.

Assume Pointer Aliases Exist (-B19)

The **-B19** option is selected when more than one symbolic name is used to reference a variable's memory location. This can occur when pointers are used, when variables in COMMON are also passed as arguments, or when two dummy arguments are the same actual argument.

Note: Standard FORTRAN should not require this option, but the use of extensions may dictate its use. Performance loss should be expected when this option is selected.

Check array boundaries (-C)

When the **-C** compiler option is turned on, code will be generated to check that array indexes are within the bounds of an array. Exceptions: arrays whose last dimension is * and dummy arguments whose last dimension is 1 cannot be checked. In addition, file names and source code line numbers will be displayed with all run time error messages.

Generate Debugging Information (-g)

Specifying the **–g** option will cause the compilers to include symbol and line information appropriate for debugging a compiled program with Fx, the Absoft debugger.

The Absoft Fortran 90/95 and FORTRAN 77 compilers have the capability to output special symbol information for use with the Fx debugger from Absoft. This information

allows Fx to display the contents of adjustable arrays, arrays with more than four dimensions, arrays with lower bounds other than 1, and arrays with dimensions greater than 32767.

Info for unused structures (-N111)

Normally, the compiler does not place information in the debugger symbol tables for structures which are only declared, but never have storage associated with them. This keeps the symbol tables to a manageable size when include files are used to make structure declarations. The **-N111** option can be used to force the compiler to place information in the debugger symbol tables for all structures whether they have associated storage or not. This option is only enabled when the **-g** option has been selected.

Generate Profiler Information (-P)

Specifying the **–P** option will place information for profiling execution into a compiled program. For information on using the Linux profiler, see the Linux manual page for *gprof*.

Conditional compilation (-x)

Statements containing an X or a D in column one are treated as comments by the compiler unless the **-x** compiler option is selected. This option allows a restricted form of conditional compilation designed primarily as a means for easily removing debugging code from the final program. When the **-x** option is selected, a blank character replaces any occurrence of an X or a D in column one. The only source formats for which conditional compilation is valid are standard FORTRAN 77, VAX Tab-Format, and wide format. The compiler also incorporates a complete set of statements for conditional compilation which are described in the **Conditional Compilation Statements** section of the **FORTRAN 77 Program** chapter in the *FORTRAN 77 Language Reference Manual*.

Max Internal Handle (-T nn)

This option is used to change the number of handles used internally by the compiler. Under most conditions, the default value of 20000 handles is sufficient to compile even extremely large programs. However, under certain circumstances, this value may be exceeded and the compiler will issue a diagnostic indicating that the value should be increased.

Define Compiler Directive (-D*name*[=*value*])

The **-D** option is used to define conditional compilation variables from the command line. `value` can only be an integer constant. If `value` is not present, the variable is given the value of 1. Conditional compilation is described in the **Conditional Compilation Statements** section in the **FORTRAN 77 Program** chapter of the *FORTRAN 77 Language Reference Manual*.

Set Include Paths (-I)

Use this command to select additional directory paths to be searched for include and header files. The **-I** option is used to supply a comma separated list of directory paths which are prepended to file names used with the Fortran `INCLUDE` statement or the C/C++ `#include` directive.

```
-Ipath[,path…]
```

The paths are prepended in the order presented with the **-I** option when the include file is not first found in the local directory and when it is not itself an absolute path (a full file specification).

Optimizations

Absoft Fortran 77 is a globally optimizing compiler, so various optimizers can be turned on which affect single statements, groups of statements or entire programs. There are pros and cons when choosing optimizations; the application will execute much faster after compilation but the compilation speed itself will be slow. Some of the optimizations described below will benefit almost any FORTRAN code, while others should only be applied to specific situations.

You may want to ignore optimizations during program development or for compilations of FORTRAN source code ported to the Linux to save time. When a FORTRAN program is executing correctly and has been debugged, turn on optimizations for improved run-time performance. In general, all optimizations should be selected carefully.

Basic Optimizations (-O1)

The **–O1** option will cause most code to run faster and enables optimizations that do not rearrange your program. The optimizations include common subexpression elimination, constant propagation, and branch straightening. This option is generally usable with debugging options. **–cpu:host** is implied with this option.

Advanced Optimizations (-O2)

The **–O2** option enables advanced optimizers that can substantially rearrange the code generated for a program. The optimizations include strength reduction, loop invariant removal, code hoisting, and loop closure. This option is not usable with debugging options. **–cpu:host** is implied with this option.

DATA treated as constants (-N5)

The **-N5** compiler option enables the optimizer to propagate as constants those variables initialized in DATA statements that are not redefined. For example:

Original code:

```
SUBROUTINE DIVIDE(A)
INTEGER A(50),B,C

DATA B,C/100,10/

DO I=1,50
     A(I) = A(I)/B-I*C
END DO
RETURN
END
```

Becomes:

```
SUBROUTINE DIVIDE(A)
INTEGER A(50),B,C

DATA B,C/100,10/

DO I=1,50
     A(I) = A(I)/100-I*10
END DO
RETURN
END
```

This option is automatically turned on with the **-O** option for basic optimizations.

Function decomposition (-N18)

The **-N18** compiler option causes intrinsic functions to be decomposed in line wherever possible. For example:

Original code:

```
I = MOD (J,K)
```

Becomes:

```
I = (J-((J/K)*K))
```

This option is automatically turned on with the **-O** option for basic optimizations.

Evaluate Constant Functions (-N41)

Use the **–N41** compiler option to direct the compiler to evaluate FORTRAN 77 intrinsic functions whose arguments are constant expressions. This option is automatically turned on with the **-O** option for basic optimizations.

Loop unrolling (-U and -h *nn* and -H *nn*)

The Absoft Fortran 77 compiler has the ability to automatically unroll some of the loops in your source code. Loops may be unrolled by any power of two. Generally it is beneficial to unroll loops which execute a large number of iterations, while the benefit is small for loops which iterate only a few times. Due to this, only innermost loops are considered for unrolling. The **-h** *nn* option will cause the compiler to unroll your innermost loops *nn* times, where *nn* is any power of two. The **-H** *nn* option will cause the compiler to consider loops containing *nn* or fewer statements for unrolling. When the **-O** option is used, the default is to only consider loops of a single line and unroll them four times. Using the **-U** option is equivalent to using **-h 2 -H 10**, causing innermost loops of ten or fewer lines to be unrolled twice. Loop unrolling will provide a speed increase in most cases, but will make your application larger and it will require more memory to compile. Consider the following example:

Original code:

```
SUBROUTINE SUB(A,N,X)
INTEGER A(100)

DO i=1,N
    A(i) = X*A(i)
END DO
RETURN
END
```

Becomes:

```
SUBROUTINE SUB(A,N,X)
INTEGER A(100)

DO i=1,MOD(N,4)
    A(i) = X*A(i)
END DO
DO i=4,N-(MOD(N,4)),4
    A(i)   = X*A(i)
    A(i+1) = X*A(i+1)
    A(i+2) = X*A(i+2)
    A(i+3) = X*A(i+3)
END DO
RETURN
END
```

At least three comparisons and three branch instructions are saved each time the second loop is executed. Note that if your code contains extended range DO loops, unrolling loops will invalidate your program.

Optimize Address Expressions (-N86)

The **–N86** option forces the compiler to remove indexed address expressions from within loops. For the X86, this often has the desirable effect of reducing instruction stalls for floating point access. However, because the index must still be calculated, additional integer operations must be performed. If the application needs to be as fast as possible, try running once with this option and once without.

Compatibility

These options allow Absoft Fortran 77 to accept older or variant extensions of FORTRAN 77 source code from other computers such as mainframes. Many of these can be used for increased compatibility with FORTRAN 77 compilers on various mainframe computers.

Folding to lower case (-f)

The **-f** option will force all symbolic names to be folded to lower case. By default, the compiler considers upper and lowercase characters to be unique, an extension to FORTRAN 77. If you do not require case sensitivity for your compilations or specifically require that the compiler not distinguish between case, as in FORTRAN 77, use this option. This option should be used for compatibility with VAX and other FORTRAN environments.

Folding to upper case (-N109)

By default, the compiler considers upper and lowercase characters to be unique, an extension to FORTRAN 77. If you do not require case sensitivity for your compilations or specifically require that the compiler not distinguish between case, as in FORTRAN 77, including the **-N109** option on the compiler invocation command line will force all symbolic names to be folded to upper case.

Static storage (-s)

In FORTRAN 66, all storage was static. If you called a subroutine, defined local variables, and returned, the variables would retain their values the next time you called the subroutine. FORTRAN 77 establishes both static and dynamic storage. Storage local to an external procedure is dynamic and will become undefined with the execution of a `RETURN` statement. The `SAVE` statement is normally used to prevent this, but the **-s** compiler option will force all program storage to be treated as static and initialized to zero. The **-N1** compiler option causes the definition of variables initialized in `DATA` statements to be maintained after the execution of a `RETURN` or `END` statement. This option should be used for compatibility with VAX and other FORTRAN environments.

Use record lengths in I/O (-N3)

If the **-N3** compiler option is used, record length information will be included for sequential, unformatted files as if the "`BLOCK=-1`" specifier were implicitly included in all appropriate `OPEN` statements. See the **Input/Output and Format Specifications** chapter of the *FORTRAN 77 Language Reference Manual* for more information about the `BLOCK=-1` specifier. This option should be used for compatibility with VAX and other FORTRAN environments.

RECL Defines 32-bit words (-N51)

If the **-N51** compiler option is used, the "`RECL`" specifier will be interpreted as the number of 32 bit words in a record for `UNFORMATTED`, `DIRECT` access files. Without this option, `RECL` defines the number of bytes in a record. This option should be used for compatibility with VAX and other FORTRAN environments.

One-trip DO loops (-d)

FORTRAN 66 did not specify the execution path if the iteration count of a DO loop, as established from the DO parameter list, was zero. Many processors would execute this loop once, testing the iteration count at the bottom of the loop. FORTRAN 77 requires that such a DO loop not be executed. The **-d** option will cause all DO loops to be executed at least once, regardless of the initial value of the iteration count.

Integer Sizes (-i2 and -i8)

Without an explicit length declaration, INTEGER and LOGICAL data types default to thirty-two bits (four bytes). The **–i2** option can be used to change this default length to sixteen bits (two bytes) for both INTEGER and LOGICAL. The **–i8** option can be used to change the default INTEGER size to 64 bits (8 bytes). However, an explicit length specification in a type declaration statement always overrides the default data length.

Zero extend INTEGER*1 (-N102)

Normally, INTEGER*1 variables are sign extended when they are loaded from memory, providing for integers which range from -128 to 127. In order to provide compatibility with other implementations of FORTRAN, the **-N102** option can be used to direct the compiler to zero extend these variables when they are loaded, making them essentially unsigned entities with a range of 0-255.

Set Big-Endian (-N26)

Use this option to force the compiler to consider the byte ordering of all unformatted files to be big-endian by default . The CONVERT specifier in the OPEN statement may be used to override this setting for individual files.

Set Little-Endian (-N27)

Use this option to force the compiler to consider the byte ordering of all unformatted files to be little-endian by default . The CONVERT specifier in the OPEN statement may be used to override this setting for individual files.

Set COMMON block name (-N22)

The **-N22** option is used to change the scheme the compiler employs for generating global names for COMMON blocks. The default is to prepend the characters "_C" to the COMMON block name. This option cause the compiler to append a single underscore (_) instead.

Evaluate left-to-right (-N20)

When two or more operators of equal precedence appear consecutively in an arithmetic expression, the **-N20** forces the compiler to evaluate the operators from left to right

(except for the exponentiation operators), regardless of whether it is the most efficient method or not.

Double precision transcendentals (-N2)

The **-N2** option causes the compiler to use double precision or double complex transcendental intrinsic functions, overriding single precision and complex type specifications. This provides an extra bit of precision in some cases and is compatible with some C environments.

Maintain Floating Point Precision (-e)

When the **-e** option is enabled, all assignments to program variables result in both a store to memory and potentially a load from memory. For all data types maintained in processor registers, if the precision of the data type in memory is less than the precision of the data type in a register, this has the effect of rounding (floating point data types) or truncating (integral data types). Normally, Absoft FORTRAN 77 uses sophisticated methods to avoid storing or loading data unless it is actually required; the net effect being much faster executables from reduced memory traffic and small executables from less code generation. However, for a small number of programs, the additional precision supported by the machine registers actually causes the program to produce unexpected results. This is usually the fault of poor numerical practices or bad algorithms in otherwise valid code. Dues to the drastic effects on execution speed, we recommend not using this option unless it is proved absolutely necessary. This option will also force the rounding of some internally created expression intermediates and delay the evaluation of most constant expressions until runtime.

Sign extend BYTE() & WORD() (-N7)

The **-N7** compiler option causes the compiler to extend the sign of a value returned from the intrinsic functions `BYTE` and `WORD`. This option has no effect on the memory assignment statements described in the **Expressions and Assignments** chapter of the *FORTRAN 77 Language Reference Manual*.

DATA variables are static (-N1)

The **-N1** compiler option causes all variables initialized with `DATA` statements to be stored as static variables.

Promote REAL and COMPLEX (-N113)

Without an explicit length declaration, single precision `REAL` and `COMPLEX` data types default to thirty-two bits (four bytes) and sixty-four bits (eight bytes), respectively. The **-N113** option is used to promote these to their double precision equivalents: `DOUBLE PRECISION` and `DOUBLE COMPLEX`. This option does not affect variables which appear in type statements with explicit sizes (such as `REAL*4` or `COMPLEX*8`).

Escape sequences in strings (-K)

If the **-K** option is turned on, the compiler will transform certain escape sequences marked with a '\' embedded in character constants. For example '\n' will be transformed into a newline character for your system. Refer to the **FORTRAN 77 Program** chapter *FORTRAN 77 Language Reference Manual* for more information on the escape sequences that are supported.

Allows CASE without DEFAULT (-N4)

By default, a run-time error is reported if a `CASE DEFAULT` statement is not present in a block `CASE` structure when a match is not found. The **-N4** causes control of execution to be transferred to the statement following the `END SELECT` statement when a `CASE DEFAULT` statement is not present and no match is found, preventing such a run-time error.

Allows UNIT= without FMT= (-N16)

If the **-N16** compiler option is used, the format specifier `FMT=` may be omitted in an I/O statement when the unit specifier `UNIT=` is present.

Pack STRUCTURE elements (-N33)

Normally, the fields in a `STRUCTURE` are aligned based on the standard for C. This may cause spaces to be left between structure elements and space to be added to the end of a structure. The **-N33** option will cause structure fields to be "packed" on two byte boundaries. You can also use the conditional compilation directives `$PACK` and `$PACKOFF` to control the packing of individual structures (see the section **Conditional Compilation Directives** in the **FORTRAN 77 Program** chapter of the *FORTRAN 77 Language Reference Manual*). The use of this option may cause misaligned storage locations.

Align STRUCTURE fields to one byte boundaries (-N56)

The **–N56** option will force structure fields to be completely "packed". No padding will occur between fields. The use of this option may cause misaligned storage locations.

Align STRUCTURE fields to two byte boundaries (-N57)

The **–N57** option will force structure fields to be aligned to 2-byte boundaries. Fields beginning at odd addresses will be aligned to the next even address. The use of this option may cause misaligned storage locations for fields greater than 2 bytes in length..

Align STRUCTURE fields to four byte boundaries (-N58)

The **–N58** option will force structure fields to be aligned to 4-byte boundaries. Fields not beginning at a modulo 4 address will be aligned to the next 4-byte boundary. The use of this option may cause misaligned storage locations for fields greater than 4 bytes in length..

Align STRUCTURE fields to eight byte boundaries (-N59)

The **–N59** option will force structure fields to be aligned to 8-byte boundaries. Fields not beginning at a modulo 8 address will be aligned to the next 8-byte boundary.

Align COMMON variables (-N34)

If a COMMON block is defined in a manner that causes a misaligned storage location, the **-N34** option can be used to insert space to eliminate the misalignment. This option may invalidate your code if the same `COMMON` block is defined differently in different program units.

Temporary string size (-t *nn*)

In certain cases the compiler is unable to determine the amount of temporary string space that string operations will require. This undetermined length occurs when the REPEAT function is used or when a `CHARACTER*(*)` variable is declared in a subroutine or function. In these cases, the compiler will assume that the operation in question will require 1024 bytes of temporary string space. This default value can be changed by specifying the **-t** *nn*, where *nn* is a positive integer constant. When this option is specified, the default temporary string size will be *nn* bytes.

Warnings for Undeclared Variables (-N114)

If the `IMPLICIT NONE` statement appears in a program unit, the compiler will issue an error diagnostic whenever it encounters an undeclared variable. If you specify the –N114 option, the compiler will issue a warning diagnostic.

Pad Source Lines (-N115)

Use the **–N115** option to pad source lines to column 72 with spaces (or 132 with the **–W** option). By default, the compiler considers only the characters actually present in the source file. This option is useful when porting certain legacy programs that depend on the compiler reading source records as card images.

Source Formats

For compatibility with other FORTRAN environments and to provide more flexibility, the compiler can be directed to accept source code that has been written in a variety of different formats. The default setting is to accept only ANSI standard FORTRAN source code format. See the **FORTRAN 77 Program** chapter of the *FORTRAN 77 Language Reference Manual* for more information on alternative source code formats.

Fortran 90/95 Free-Form (-8)

Use of the **-8** option instructs the compiler to accept source code written in the format for the FORTRAN 90/95 Free Source Form.

IBM VS Free Form (-N112)

Use of the **-N112** option causes the compiler to accept source code in the form specified by IBM VS Free Form.

VAX Tab-Format (-V)

Use of the **-V** option causes the compiler to accept source code in the form specified by VAX Tab Format.

Wide format (-W)

Use of the **-W** option causes the compiler to accept statements that extend beyond column 72 up to column 132.

CHAPTER 3

Porting Code

This chapter describes issues involved in porting legacy FORTRAN 77 code from other platforms. One of the major design goals for Absoft Pro Fortran is to permit easy porting of source code from mainframe computers such as VAX and IBM, and from workstations such as Sun. The result is the rich set of statements and intrinsic functions accepted by the Absoft Fortran 77 compiler.

The Absoft Fortran 77 compiler is recommended for porting most legacy codes because of the number extensions and features it supports. Consequently, FORTRAN 77 options and language features will be described in this chapter. However, in most cases, the Fortran 90/95 compiler has equivalent options and can also be used. Refer to the **Using the Compilers** chapter for information on Fortran 90/95 compile time options.

The last section of this chapter describes Linux specific issues about porting code.

As a general rule when porting code, use the following two compiler options:

-f Fold all symbols to lower case.

-s Force all program storage to be treated as static and initialized to zero.

Ported programs that have incorrect runs or invalid results are usually caused by the differences between Linux and other environments such as floating point math precision or stack-size issues. See the section **Other Porting Issues** later in this chapter for special considerations when porting code to Linux. In addition, you may want to use this option:

-C Check array boundaries and generate better runtime errors. Using this option makes programs slightly larger and they will execute slower.

-B111 Validate FPU stack after procedure calls.

If you want to use the Absoft debugger, Fx, add the **-g** option to generate debugging information.

PORTING CODE FROM VAX

Absoft Fortran 77 automatically supports most of the VAX FORTRAN language extensions. Below are a list of key VAX FORTRAN extensions that are supported and a list of those that are not supported. For a complete list of VAX extensions, refer to Appendix H. Using various options, the compiler can also accept VAX Tab-Format source lines and/or 132-column lines. Otherwise, only ANSI FORTRAN 77 fixed format lines are accepted.

Key Supported VAX FORTRAN Extensions

- NAMELIST—the NAMELIST terminator may be either "$" or "&"
- STRUCTURE, RECORD, UNION, MAP, %FILL statements
- DO WHILE loops
- INCLUDE statement
- ENCODE, DECODE, ACCEPT, TYPE, and most OPEN I/O specifiers
- Hollerith and hexadecimal constant formats
- "!" comments

Key Unsupported VAX FORTRAN Extensions

- Quad-precision floating point math
- Absoft Pro Fortran uses IEEE floating point representation
- I/O statements DELETE, DEFINE FILE, and REWRITE
- Data dictionaries

Compile Time Options and Issues

Absoft Fortran 77 can be made even more compatible with VAX FORTRAN by using a group of compiler options collectively referred to as the "VAX compatibility options", listed below:

-f Fold all symbols to lower case.
-s Force all program storage to be treated as static and initialized to zero.
-N3 Include record length information for SEQUENTIAL, UNFORMATTED files.
-N51 Interpret the RECL specifier as the number of 32-bit words in a record.

VAX-compatible time, date, and random number routines are available by linking with the library file libV77.a in the /opt/absoft/lib directory. The routine names may be referenced as all upper case, all upper case with an underscore appended (**-B108**), or all lower case with an underscore appended. The routine names are:

DATE subroutine	returns current date as CHARACTER*9
IDATE subroutine	returns current date as 3 INTEGER*4
TIME subroutine	returns current time as CHARACTER*8
SECNDS subroutine	returns seconds since midnight
RAN function	returns random number

The following list of VAX FORTRAN "qualifiers" shows the equivalent Absoft Fortran 77 options or procedures:

/ANALYSIS_DATA	no equivalent
/CHECK BOUNDS	**-C** to check array boundaries
/CHECK NONE	do not use the **-C** option
/CHECK OVERFLOW	no equivalent

/CHECK UNDERFLOW	no equivalent
/CONTINUATIONS	Absoft Fortran 77 automatically accepts an unlimited number of continuation lines
/CROSS_REFERENCE	no equivalent
/DEBUG	**-g** to generate debugging information
/D_LINES	**-x** to compile lines with a "D" or "X" in column 1
/DIAGNOSTICS	append > filename to the f77 command line to create a file containing compiler warning and error messages.
/DML	no equivalent
/EXTEND_SOURCE	**-W** to permit source lines up to column 132 instead of 72
/F77	do not use the **-d** option
/NOF77	**-d** for FORTRAN 66 compatible DO loops
/G_FLOATING	see the section **Numeric Precision** later in this chapter
/I4	do not use the **-i** option
/NOI4	**-i** for interpreting INTEGER and LOGICAL as INTEGER*2 and LOGICAL*2
/LIBRARY	no equivalent
/LIST	a symbol table dump may be generated with the **-D** option
/MACHINE_CODE	**-S** to generate an assembly source file that *can* be assembled
/OBJECT	no equivalent—you can use the cp command to copy an object file to another name
/OPTIMIZE	**-O** to use basic optimizations
/PARALLEL	no equivalent
/SHOW	no equivalent
/STANDARD	**-N32** to generate warnings for non-ANSI FORTRAN 77 usage
/WARNINGS DECLARATIONS	the IMPLICIT NONE statement may be used to generate warnings for untyped data items
/WARNINGS NONE	**-w** to suppress compiler warnings

The tab size on Linux may be different than the VAX. You can set the tab size for the compiler with the environment variable TABSIZE. For more information about tab size, see the **Tab Character Size** section later in this chapter.

Runtime Issues

If the program is having problems with I/O, make sure you are using the **-N3** and **-N51** options described in detail in sections **Use record lengths in I/O** and **RECL Defines 32-bit words** in the chapter **Using the Compilers**.

PORTING CODE FROM IBM VS FORTRAN

Absoft Fortran 77 automatically supports most of the IBM VS FORTRAN language extensions. Below is a list of key VS FORTRAN extensions that are supported and not

supported. Using a compiler option, Absoft Fortran 77 can also accept VS FORTRAN Free-Form source lines which use 80 columns, otherwise, only ANSI FORTRAN 77 fixed format lines are accepted.

Key Supported VS FORTRAN Extensions

- "*" comments in column 1
- Can mix CHARACTER and non-CHARACTER data types in COMMON blocks
- The NAMELIST terminator may be an ampersand "&"
- Hollerith constants

Key Unsupported VS FORTRAN Extensions

- Quad-precision floating point math
- Absoft Fortran 77 uses IEEE floating point representation (more accurate)
- Debug statements
- I/O statements DELETE, REWRITE, and WAIT
- INCLUDE statement syntax is different

Compile-time Options and Issues

Absoft Fortran 77 can be made even more compatible with VS FORTRAN by using these compiler options:

-f Fold all symbols to lower case
-s Force all program storage to be treated as static and initialized to zero
-N3 Include record length information for SEQUENTIAL, UNFORMATTED files

Run-time Issues

If the program is having problems with unformatted I/O, make sure you are using the **-N3** option described in detail in the chapter **Using the Compilers**.

PORTING CODE FROM MICROSOFT FORTRAN (PC VERSION)

Absoft Fortran 77 automatically supports many of the Microsoft FORTRAN language extensions. Below is a list of key Microsoft FORTRAN extensions that are supported and not supported. Absoft Fortran 77 does not have the code size restrictions found in the segmented Microsoft FORTRAN models.

Key Supported Microsoft FORTRAN Extensions

- The NAMELIST terminator may be an ampersand "&"
- The Free-Form Source Code is very similar to VS FORTRAN (**-V** option)
- Unlimited number of continuation lines
- AUTOMATIC statement

- STRUCTURE, RECORD, UNION, MAP statements
- SELECT CASE statements
- DO WHILE loops
- INCLUDE statement
- Conditional compilation statements

Key Unsupported Microsoft FORTRAN Extensions

- Metacommands
- MS-DOS specific intrinsic functions
- INTERFACE TO statement
- OPEN statement displays standard file dialog when using FILE=""

Compile-time Options and Issues

Absoft Fortran 77 can be made even more compatible with Microsoft FORTRAN by using these compiler options:

-f Fold all symbols to lower case
-s Force all program storage to be treated as static and initialized to zero
-N3 Include record length information for SEQUENTIAL, UNFORMATTED files

If you use the Microsoft FORTRAN I/O specifier FORM='BINARY' to read and write binary sequential files with no internal structure, do *not* use the **-N3** option which includes record length within sequential, unformatted files.

The following list of Microsoft FORTRAN metacommands shows the equivalent Absoft Fortran 77 options or procedures:

$DEBUG	**-C** to check array boundaries and other run-time checks
$DECLARE	the IMPLICIT NONE statement may be used to generate warnings for untyped data items
$DO66	**-d** for FORTRAN 66 compatible DO loops
$FLOATCALLS	all floating point is calculated inline or with a threaded math library in Absoft Fortran 77
$FREEFORM	**-V** for IBM VS FORTRAN Free-Form source code
$INCLUDE	use the INCLUDE statement
$LARGE	not necessary — Absoft Fortran 77 does not have the data size restrictions found in the segmented Microsoft FORTRAN models
$LINESIZE	not applicable
$LIST	no equivalent
$LOOPOPT	**-U** for loop unrolling optimization; **-R** for loop invariant removal
$MESSAGE	no equivalent
$PACK	use $PACKON and $PACKOFF
$PAGE	not applicable
$PAGESIZE	not applicable

`$STORAGE:2`	**-i** for interpreting `INTEGER` and `LOGICAL` as `INTEGER*2` and `LOGICAL*2`
`$STORAGE:4`	do not use the **-i** option
`$STRICT`	**-N32** to generate warnings for non-ANSI FORTRAN 77 usage
`$SUBTITLE`	not applicable
`$TITLE`	not applicable
`$TRUNCATE`	no equivalent

PORTING CODE FROM SUN WORKSTATIONS

Absoft Fortran 77 automatically supports most of the Sun FORTRAN language extensions. Below is a list of key Sun FORTRAN extensions that are supported and not supported. The Sun FORTRAN compiler appends an underscore to all external names to prevent collisions with the C library. Absoft Fortran 77, by default, does not append an underscore to maintain compatibility with Linux functions and other development languages. The **-B108** option may be used to append underscores to routine names.

Key Supported Sun FORTRAN Extensions

- `NAMELIST`; the `NAMELIST` terminator may be either "`$`" or "`&`"
- `STRUCTURE`, `RECORD`, `POINTER`, `UNION`, `MAP`, `%FILL` statements
- `DO WHILE` loops
- `INCLUDE` statement
- `ENCODE`, `DECODE`, `ACCEPT`, `TYPE`, and most `OPEN` I/O specifiers
- Hollerith and hexadecimal constant formats
- "`!`" comments in column 1

Key Unsupported Sun FORTRAN Extensions

- Quad-precision floating point math

PORTING CODE FROM INTEL 386/486/PENTIUM COMPUTERS

Absoft Pro Fortran is available for the Intel Pentium systems including Windows 95, Windows 98, and Windows/NT. It has the same optimizations and language extensions as Absoft Pro Fortran for Linux with PowerPC. The compilers are 100% source compatible.

PORTING CODE FROM MACINTOSH SYSTEMS

Language Systems Fortran

Absoft Fortran 77 and Language Systems Fortran share many extensions implemented in other compilers. In addition, Absoft Fortran 77 automatically supports most of the Language Systems Fortran specific language extensions. Below is a list of key Language Systems extensions that are supported and a list of those that are not supported.

Key Supported Language Systems Fortran Extensions

- `STRING` declaration statement
- `POINTER` declaration statement
- `LEAVE` control statement
- `GLOBAL`, `CGLOBAL`, and `PBLOBAL` statements
- `CEXTERNAL` and `PEXTERNAL` statements
- `INT1`, `INT2`, `INT4`, and `JSIZEOF` intrinsic functions

Key Unsupported Language Systems Fortran Extensions

- variables in `FORMAT` statements
- Language Systems Fortran compiler directives

Other Absoft Macintosh Compilers

Over the past 15 years, Absoft has offered several different compilers for a number of Macintosh environments. This section outlines some of the differences between these products.

MacFortran	This 68000 compiler supported ANSI FORTRAN 77 and compiled programs directly from the Finder without using MPW. Although it lacked optimizations and support for many of the extensions in Absoft Pro Fortran for Macintosh with PowerPC, it compiled very fast and was easy to use.
MacFortran/020	This 68000 compiler was the same as MacFortran but it could also produce faster code for 68020 and 68030 systems that incorporated a floating point unit.
MacFortran II	This 68000 compiler is very similar to Absoft Pro Fortran for Macintosh with PowerPC. It supports many of the same optimizations and extensions, but is designed for 68000 based Macintoshes.

DISTRIBUTION ISSUES

If you plan to distribute executable programs generated with Absoft Fortran 77, you must obtain a copy of the Absoft "Redistribution License Agreement", complete it, and return it to Absoft. There is no charge for this license or the redistribution of programs created with Absoft Pro Fortran. To obtain the Absoft "Redistribution License Agreement", visit the Absoft Corporation web site at http://www.absoft.com, or write to:

Absoft Corporation
2781 Bond Street
Rochester Hills, MI 48309

OTHER PORTING ISSUES

Not all porting and compatibility issues can be solved automatically by Absoft Pro Fortran or by using various option combinations. There are six issues that must be addressed on a program-by-program basis for the Linux computer:

Memory Management	Tab Character Size
Naming Conventions	Numeric Precision
File and Path Names	Floating Point Math Control

Memory Management

Local variables and temporary values are stored in the ESP stack frame. All other storage is allocated statically in the data and/or bss sections.

Dynamic Storage

Storage for variables local to a function or a subroutine is allocated in the stack frame. As a result, local variables are undefined when execution of a function or subroutine begins and become undefined again when execution terminates. This can cause difficulties in two areas.

First, problems may arise when porting Fortran applications from environments that statically allocate all memory; the application may except variables to retain their definition status across procedure references. However, it produces applications that make more effective use of memory and provides the ability to call functions and subroutines recursively. The next section describes how to declare static storage space.

Second, the Linux stack is limited to 8 MB and large arrays allocated in the stack frame may overflow the stack. You can increase the stack size with the `ulimit` command (`ulimit` is a bash command - the csh equivalent to `ulimit -s` is `limit stack`) to raise the stack size limit:

```
# ulimit –s
8192
# ulimit -s 32768
# ulimit -s
```

```
32768
```

The Linux stack limit is defined by the following around line 293 in `sched.h`:

```
#define _STK_LIM        (8*1024*1024)
```

Static Storage

There are three ways to define static storage in Fortran. The first two allow static variables to be defined selectively and are either placing them in `COMMON` blocks or using the `SAVE` statement. The third method, using the **`-s`** compiler option, forces *all* program storage to be treated as static. Static memory is allocated out of the data and/or bss sections and remains defined for as long as the application runs. In addition, all static storage will be initialized to zero when the application begins execution.

Naming Conventions

Global names in Fortran include all procedure names and `COMMON` block names, both of which are significant to 31 characters. All global names in Absoft FORTRAN 77 are case sensitive unless one of the compiler character case options has been selected. All global names in Absoft Fortran 90/95 are upper case unless one of the compiler character case options has been selected. All other symbols are manipulated as addresses or offsets from local labels and are invisible to the linker.

Procedure Names

Names of functions and subroutines in Fortran programs will appear in the assembly language source output or object file records exactly as they are stated in the Fortran source code. This is identical to how the C Programming Language represents symbolic names on Linux.

If a FORTRAN 77 subroutine is defined as:

```
SUBROUTINE SUB(…)
.
.
.
RETURN
END
```

It will be defined in assembly language as:

```
      .globl sub
sub:

      ret
```

COMMON Block Names

The convention in Absoft Pro Fortran us to precede the name given in the COMMON statement with the characters "_C". BLANK common uses the characters _blank.

For example, the COMMON block declaration:

```
COMMON /the_block/ a, b, c
```

Eill produce the following assembler directive:

```
.comm _Cthe_block, 0x0000000c
```

File and Path Names

When the compiler encounters the Fortran INCLUDE statement, it takes the CHARACTER constant immediately following as a file name, searches for the file, and, if the file is found, copies its contents into the source file. If an absolute or relative path name is specified, the compiler will search only that path. If only a file name is given, the compiler will first look for the file in the current directory. It will then search any directory defined by the environment variable F77INCLUDES. Additional search paths may be specified with the –I compiler option.

Tab Character Size

The compiler assumes a standard tab size of eight spaces. This is the default for most editors. When the compiler encounters a tab character (ASCII 9) during compilation, it is replaced with the appropriate number of spaces for alignment to the next tab stop. By setting the environment variable TABSIZE, the tab size used by the compiler can be changed. The following command line for the Bourne shell will set the tab size for the compiler to four spaces:

```
TABSIZE=4
export TABSIZE
```

Runtime Environment

A number of the aspects of the runtime environment can be controlled with the ABSOFT_RT_FLAGS environment variable. This variable can be a combination of any of the following switches (the leading minus sign is required for each switch and multiple switches must be separated by one or more spaces):

-defaultcarriage

> Causes the units preconnected to standard output to interpret carriage control characters as if they had been connected with ACTION='PRINT'.

-fileprompt

Causes the library to prompt the user for a filename when it implicitly opens a file as the result of I/O to an unconnected unit number. By default, the library creates a filename based on the unit number.

-vaxnames

Causes the library to use 'vax style' names (FORnnn.DAT) when creating a filename as the result of I/O to an unconnected unit number.

-unixnames

Causes the library to use 'unix style' names (fort.nnn) when creating a filename as the result of I/O to an unconnected unit number.

-bigendian

Causes the library to interpret all unformatted files using big endian byte ordering.

-littleendian

Causes the library to interpret all unformatted files using little endian byte ordering.

-noleadzero

Causes the library to surpress the printing of leading zeroes when processing an Fw.d edit descriptor. This only affects the limited number of cases where the ANSI standard makes printing of a leading zero implementation defined.

-reclen32

Causes the library to interpret the value specified for `RECL=` in an `OPEN` statement as 32-bit words instead of bytes.

-f90nlexts

Allows f90 `namelist` reads to accept non-standard syntax for array elements. Without this flag, the following input results in a runtime error:

```
$ONE
A(1)=1,2,3,4
$END
```

When -f90nlexts is set, the values are assigned to the first four elements of A.

-nounit9

Causes UNIT 9 not to be preconnected to standard input and output.

-maceol

Formatted sequential files are in Classic Macintosh format where each record ends with a carriage return,

-doseol

Formatted sequential files are in Windows format where each record ends with a carriage return followed by a line feed.

-unixeol

Formatted sequential files are in Unix format where each record ends with a line feed.

-hex_uppercase

Data written with the Z edit descriptor will use upper case characters for A-F.

Floating Point Math Control

This section describes the basic information needed to control the floating-point unit (FPU) built into Intel. The FPU provides a hardware implementation of the IEEE Standard For Binary Floating Point Arithmetic (ANSI/IEEE Std 754-1985). As a result it allows a large degree of program control over operating modes. There are two aspects of FPU operation that can affect the performance of a FORTRAN program:

Rounding direction

Exception handling

A single subroutine is provided with the compiler that is used to retrieve the current state of the floating-point unit or establish new control conditions:

```
CALL fpcontrol(cmd,arg)
```

where: *cmd* is an `INTEGER` variable that is set to 0 to retrieve the state of the floating point unit and 1 to set it to a new state.

arg is an `INTEGER` variable that receives the current state of the floating point unit if *cmd* is 0 and contains the new state if *cmd* is 1.

Rounding Direction

The first aspect of FPU operation that may affect a FORTRAN program is rounding direction. This refers to the way floating-point values are rounded after completion of a floating-point operation such as addition or multiplication. The four possibilities as defined in the fenv.inc include file are:

`FE_TONEAREST`	round to nearest
`FE_TOWARDZERO`	round toward zero
`FE_UPWARD`	round toward +infinity
`FE_DOWNWARD`	round toward -infinity

Exception Handling

The second aspect of FPU operation that affects FORTRAN programs is the action taken when the FPU detects an error condition. These error conditions are called exceptions, and when one occurs the default action of the FPU is to supply an error value (either Infinity or NaN) and continue program execution. Alternatively, the FPU can be instructed to generate a floating point exception and a run time error when an exception takes place. This is known as *enabling the exception*. The five exceptions that can occur in a FORTRAN program are:

`FE_INEXACT`	inexact operation
`FE_DIVBYZERO`	divide-by-zero
`FE_UNDERFLOW`	underflow
`FE_OVERFLOW`	overflow
`FE_INVALID`	invalid argument

FSPLIT - SOURCE CODE SPLITTING UTILITY

When you need to manage large files, work on small portions of Fortran code, or port code from other environments, you may want to split large, cumbersome source files into one procedure per file. This can be done using the Fsplit tool. The command syntax for the tool is shown below.

```
Fsplit [option…] [file…]
```

Fsplit splits FORTRAN source files into separate files with one procedure per file. The following command line will generate individual files for each procedure:

```
Fsplit largefile.f
```

A procedure includes block data, function, main, program, and subroutine program declarations. The procedure, proc, is put into file *proc*`.f` with the following exceptions:

- An unnamed main program is placed in `MAIN.f`.
- An unnamed block data subprogram is placed in a file named `blockdataNNN.f`, where `NNN` is a unique integer value for that file. An existing block data file with the same name will not be overwritten.

- Newly created procedures (non-block data) will replace files of the same name.
- File names are truncated to 14 characters.

Output files are placed into the directory in which the `fsplit` command was executed. The tab size is pulled from the environment variable `TABSIZE` if it exists, otherwise, a tab size of 8 is used. Options for the command are:

-v Verbose progress of `fsplit` is displayed on standard diagnostic.

-V Source files are in VAX FORTRAN Tab-Format.

-I Source files are in IBM VS FORTRAN Free-Form.

-8 Source files are in Fortran 90/95 Free Source Form.

-W Source files are in wide format.

CHAPTER 4

Interfacing With Other Languages

This chapter discusses interfacing Absoft Pro Fortran with the C Programming Language and assembly language, debugging programs, and profiling executables. Although Fortran programs can call C functions easily with just a `CALL` statement, the sections below should be read carefully to understand the differences between argument and data types.

INTERFACING WITH C

Absoft Pro Fortran is designed to be fully compatible with the implementation of the standard C Programming Language provided on Linux. The linker can be used to freely link C modules with Fortran main programs and vice versa. However, some precautions must be taken to ensure proper interfacing. Data types in arguments and results must be equivalent. The case of global symbols C is significant. The symbolic names of external procedure must match in case.

Fortran Data Types in C

Declarations for Fortran data types and the equivalent declarations in C are as follows:

Fortran	C
`LOGICAL*1 l`	`unsigned char l;`
`LOGICAL*2 m`	`unsigned short m;`
`LOGICAL*4 n`	`unsigned long n;`
`CHARACTER*n c`	`char c[n];`
`INTEGER*1 i` or `BYTE i`	`char i;`
`INTEGER*2 j`	`short j;`
`INTEGER*4 k`	`int k;` `long k;`
`REAL*4 a`	`float a;`
`REAL*8 d`	`double d;`

`COMPLEX*8 c`	`struct complx {`
	`  float x;`
	`  float y;`
	`};`
	`struct complx c;`

`COMPLEX*16 d`	`struct dcomp {`
	`  double x;`
	`  double y;`
	`};`
	`struct dcomp d;`

The storage allocated by the C language declarations will be identical to the storage allocated by the corresponding Fortran declaration.

There are additional precautions when passing Fortran strings to C routines. See the section **Passing Strings to C** later in this chapter for more information.

Required Compiler Options

FORTRAN 77 code should be compiled with the following options:

-f	fold symbols to lower case
-s	use static storage
-B108	append trailing underscores to global names
-N90	use g77 `CHARACTER` argument protocols

Fortran 90 code should be compiled with the following options:

-YEXT_NAMES=LCS	fold symbols to lower case
-s	use static storage
-B108	append trailing underscores to global names
-YCFRL=1	use g77 `CHARACTER` argument protocols

C code does not have to be compiled with any special options for the C compiler.

Rules for Linking

When linking Fortran and C programs, the `f77` or `f90` compiler driver should be used so that the appropriate Fortran and C libraries are included in the final application. The following command will compile the file `f1.f` with the FORTRAN 77 compiler and the file `c1.c` with the C compiler. It will then link the two resulting object files along with `o1.o` and the appropriate libraries to generate an executable application named exec:

```
f77 –o exec f1.f c1.c o1.o
```

If object files or libraries that have been built with `g77` are used, the `g77` runtime library should be specified as either: `-lf2c` or `-lg2c` depending on your version of Linux. Further, current information can be obtained in the technical support section at the Absoft web site: www.absoft.com.

Passing Parameters Between C and Fortran

The Absoft Pro Fortran compilers use the same calling conventions as the C programming language. Therefore, a Fortran routine may be called from C without being declared in the C program and vice versa, if the routine returns all results in parameters. Otherwise, the function must be typed compatibly in both program units. In addition, care must be taken to pass compatible parameter types between the languages. Refer to the table earlier in this chapter.

Reference parameters

By default, all Fortran arguments to routines are passed by reference, which means pointers to the data are passed, not the actual data. Therefore, when calling a Fortran procedure from C, pointers to arguments must be passed rather than values. Both integer and floating point values may be passed by reference. Consider the following example:

```
SUBROUTINE SUB(a_dummy,i_dummy)
REAL*4 a_dummy
INTEGER*4 i_dummy

WRITE (*,*) 'The arguments are ',a_dummy, ' and ', i_dummy
RETURN
END
```

The above subroutine is called from Fortran using the CALL statement:

```
a_actual = 3.3
i_actual = 9
CALL SUB(a_actual, i_actual)
END
```

However, to call the subroutine from C, the function reference must explicitly pass pointers to the actual parameters as follows:

```
int main()
{
      float a_actual;
      int i_actual;
      void SUB();

      a_actual = 3.3;
      i_actual = 9;
      SUB(&a_actual,&i_actual);
      return 0;
}
```

Note that the values of the actual parameters may then be changed in the Fortran subroutine with an assignment statement or an I/O statement.

When calling a C function from Fortran with a reference parameter, the C parameters are declared as pointers to the data type and the Fortran parameters are passed normally:

```
PROGRAM convert_to_radians
WRITE (*,*) 'Enter degrees:'
READ (*,*) c
CALL C_RAD(c)
WRITE (*,*) 'Equal to ',c,' radians'
END
```

```
void C_RAD(c)
float *c;
{
      float deg_to_rad = 3.14159/180.0;
      *c = *c * deg_to_rad;
}
```

Value parameters

Absoft Pro Fortran provides the intrinsic function %VAL() for passing value parameters. Function interfaces may also be used to specify which arguments to pass by value. Although it is generally pointless to pass a value directly to a Fortran procedure, these functions may be used to pass a value to a C function. The following is an example of passing a 4-byte integer:

```
WRITE (*,*) 'Enter an integer:'
READ (*,*) i
CALL C_FUN(VAL(i))
```

```
END
```

```
void C_FUN(i)
int i;
{
      printf ("%d is ",i);
      if (i % 2 == 0)
        printf ("even.\n");
      else
        printf ("odd.\n");
}
```

The value of `i` will be passed directly to `C_FUN`, and will be left unaltered upon return. Value parameters can be passed from C to Fortran with use of the `VALUE` statement. The arguments that are passed by value are simply declared as `VALUE`.

```
void C_FUN()
{
void FORTRAN_SUB();
int i;

      FORTRAN_SUB(i);
}
```

```
SUBROUTINE FORTRAN_SUB(i)
VALUE i
...
END
```

Note that C will pass all floating-point data as double precision by default, and that the only Fortran data type that cannot be passed by value is `CHARACTER`.

Array Parameters

One-dimensional arrays can be passed freely back and forth as both language implementations pass arrays by reference. However, since C and Fortran use different row/column ordering, multi-dimensional arrays cannot be easily passed and indexed between the languages.

```
INTEGER ia(10)

CALL C_FUN(ia)
WRITE (*,*) ia

END
```

```
void C_FUN(i)
int i[];
{
int j;
      for(i=0; j<10; j++)
            i[j]=j;
}
```

Function Results

In order to obtain function results in Fortran from C language functions and vice versa, the functions must be typed equivalently in both languages: either `INTEGER`, `REAL`, or `DOUBLE PRECSION`. All other data types must be returned in reference parameters. The following are examples of the passing of function results between Fortran and C. The names are case-sensitive, so trying to call `cmax`, for example, will result in an error at link time.

A call to C from Fortran

```
PROGRAM callc
INTEGER*4 CMAX, A, B

WRITE (*,*) 'Enter two numbers:'
READ (*,*) A, B
WRITE (*,*) 'The largest of', A, ' and', B, ' is ', CMAX(A,B)
END
```

```
int CMAX (x,y)
int *x,*y;
{
      return( (*x >= *y) ? *x : *y );
}
```

A call to Fortran from C

```
main()
{
float QT_TO_LITERS(), qt;
```

```
      printf ("Enter number of quarts:\n");
      scanf ("%f",&qt);
      printf("%f quarts = %f liters.\n", qt, QT_TO_LITERS(&qt));
}
```

```
REAL*4 FUNCTION QT_TO_LITERS(q)
REAL*4 q;

QT_TO_LITERS = q * 0.9461;
END
```

Passing Strings to C

Fortran strings are a sequence of characters padded with blanks out to their full fixed length, while strings in C are a sequence of characters terminated by a null character. Therefore, when passing Fortran strings to C routines, you should terminate them with a null character. The following Fortran expression will properly pass the Fortran string `string` to the C routine `CPRINT`:

```
PROGRAM cstringcall
character*255 string
string = 'Moscow on the Hudson'
CALL CPRINT(TRIM(string)//CHAR(0))
END
```

```
void CPRINT (anystring)
char *anystring;
{
      printf ("%s\n",anystring);
}
```

This example will neatly output "`Moscow on the Hudson`". If the `TRIM` function were not used, the same string would be printed, but followed by 235 blanks. If the `CHAR(0)` function was omitted, C would print characters until a null character was encountered, whenever that might be.

You can also take advantage of the string length arguments that Fortran passes. After the end of the formal argument list, Fortran passes (and expects) the length of each CHARACTER argument as a 32-bit integer value parameter. For example:

```
SUBROUTINE FPRINT(string)
character*(*) string
print *, string
END
```

```
#include <string.h>

int main()
{
char string[] = {"Moscow on the Hudson"};
void FPRINT(char *, int);

      FPRINT(string, strlen(string));
      return 0;
}
```

Calling Fortran math routines

All of the Fortran intrinsic math functions which return values recognized by the C Programming Language can be called directly from C as long as the Fortran run time library, libf77math.a, is linked to the application.

Taking the intrinsic function names in lower case and adding two underscores to the beginning forms the names of the functions that can be called.

The following example calls the Fortran intrinsic function SIN directly from C:

```
main()
{
float sin_of_a, a, __sin();

      a = 3.1415926/6;
      sin_of_a = __sin(a);
}
```

Naming Conventions

Global names in FORTRAN include procedure names and COMMON block names, both of which are significant to 31 characters. All global names are case sensitive, meaning the compiler recognizes the difference between upper and lower case characters. Use of the **-f** option will fold global names to lower case, while the **-N109** option will fold global names to upper case. All other symbols in FORTRAN are manipulated as addresses or offsets from local labels and are invisible to the linker.

Procedure Names

Names of functions and subroutines in FORTRAN programs will appear in the assembly language source output or object file records as they were typed in the source code with a period prefix character attached. Symbolic names in the C language are case sensitive, distinguishing between upper and lower case characters. To make FORTRAN code compatible with C, avoid using the **-f** or **-N109** options when compiling the FORTRAN source code.

Accessing COMMON blocks from C

COMMON block names are global symbols formed in Absoft Pro Fortran by prepending the characters "_C" to the name of the COMMON block. The elements of the COMMON block can be accessed from C by declaring an external structure using this name. For example,

```
COMMON /comm/ a,b,c
```

can be accessed with the C declaration:

```
extern struct {
     float a;
     float b;
     float c;
} _CCOMM;
```

Declaring C Structures in Absoft Pro Fortran

If there are equivalent data types in FORTRAN for all elements of a C structure, a RECORD can be declared in FORTRAN to match the structure in C:

C	FORTRAN
`struct str {`	`STRUCTURE /str/`
`char c;`	`  CHARACTER c`
`long l;`	`  INTEGER*4 l`
`float f;`	`  REAL*4 f`
`double d;`	`  REAL*8 d`
`};`	`END STRUCTURE`
`struct str my_struct;`	`RECORD /str/ my_struct`

By default, the alignment of the C structure should be identical to the FORTRAN RECORD. Refer to the **Specification and DATA Statements** chapter of the *FORTRAN 77 Language Reference Manual* for more information on the FORTRAN RECORD type.

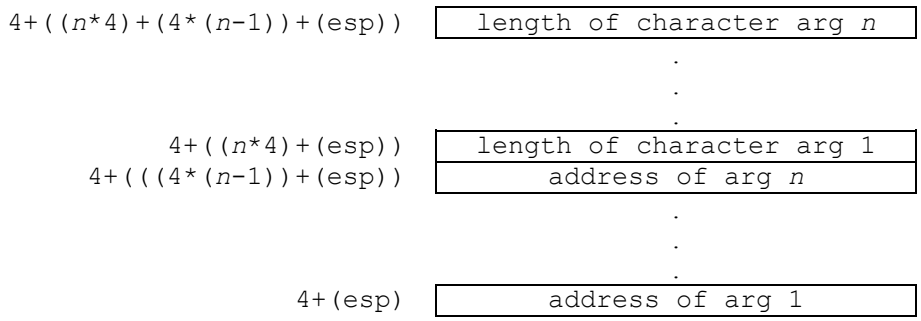
INTERFACING WITH ASSEMBLY LANGUAGE

This section discusses how arguments and results are passed on the stack and in registers.

The Fortran Stack Frame

The addresses of arguments to a Fortran procedure are passed in a right to left order on the ESP the stack. The lengths of character arguments are passed as 32 bit integers above these addresses. On entry to a Fortran procedure, the stack frame is defined as follows:

Subroutine declaration: `SUBROUTINE sub(arg 1, ... ,arg n)`



argument position = `4 + ((m-1) * 4)`
length position = `4 + (n * 4 + 4 + (m-1) * 4)`

where: *m* = argument number
n = total arguments

The Fortran Stack Frame

Value arguments for all data types are passed in the stack frame beginning at the argument position described above and extending as far as they need to. Value arguments that are less than four bytes in length are extended to four bytes before they are passed. The stack is always aligned to a sixteen byte boundary.

Space for `CHARACTER` and derived type function results is passed as if it were an extra argument at the beginning of the argument list. For example, the following two calls are equivalent in respect to how arguments are passed to the external function or subroutine:

```
CHARACTER*10 funct, arg, result
EXTERNAL sub

result = funct(argument)
CALL sub(result,argument)
```

Function Results

Absoft Pro Fortran returns all numeric and logical function registers. Floating point results are returned in `st(0)` or `st(0)` and `st(1)`. Integer and logical results are returned in `EAX`. `POINTER` results are also returned in `EAX`.

`CHARACTER` and derived type results cannot be returned in registers. Since space for the result is passed in as the first argument, no result need be returned. `RECORD` results are returned in the same fashion except that `EAX` is set to point the returned structure.

DEBUGGING

Debugging a Fortran program is accomplished with the Absoft source-level debugger, Fx™. This is a multi-language, windowed debugger designed especially for Linux based computers. The operation of the debugger is detailed in the document, **Fx Debugger User Guide**. The following paragraphs describe the compiler options and resources necessary to prepare a program for debugging.

Compiler Options

The **-g** compiler option directs the compiler to add symbol and line number information to the object file. This option should be enabled for each source file that you will want to have source code displayed while debugging. It is not required for files that you are not interested in.

It is recommended that all optimization options be disabled while debugging. This is because the optimizers can greatly distort the appearance and order of execution of the individual statements in your program. Code can be removed or added (for loop unrolling), variables may be removed or allocated to registers (making it impossible to examine or modify them), and statements may be executed out of order.

PROFILING

The Linux operating system includes the libraries and tools necessary to obtain procedure level profiles of your application. You simply create an instrumented version of your application (see **Compiler Options** below) and then execute it. The file `gmon.out` will automatically be created. Use *gprof* to display and analyze the results.

Compiler Options

The **-P** compiler option directs the compiler to add the symbol information to the object file necessary to profile an application. Enabling this option will allow the application to report the number of times a particular subroutine is called or a function is referenced.

All other options that you would normally use should be enabled, including optimization.

Appendix A

Absoft Compiler Option Guide

This appendix summarizes general options for Absoft Pro Fortran compilers and specific options for the Absoft Fortran 90/95 and FORTRAN 77 compilers. Refer to the chapter, **Using the Compilers** for detailed descriptions of the options

ABSOFT PRO FORTRAN COMPILER OPTIONS

Option	*Effect*
-c	suppresses creation of an executable file — leaves compiled files in object code format.
-g	generates symbol information for Fx™.
-L*path*	library file search path specification.
-l*name*	library file specification.
-O	enables a group of basic optimizations which will cause most code to run faster without the expense of application size or memory usage.
-o *name*	directs the compiler to produce an executable file called *name* where *name* is a Windows file name.
-P	instrument executable for profiling.
-S	generates an assembly language output file.
-s	allocate local variables statically.
-u	undefine a symbol to the linker.
-v	directs the compiler to print status information as the compilation process proceeds.
-w	suppresses listing of all compile-time warning messages.
-X*option*	linker option.

FPU CONTROL OPTIONS

-round=*mode* set the FPU rounding method.

-trap=*exception* enable FPU exceptions.

-B23 do not modify the FPU control register.

-B24 preserve the FPU control register.

-B111 issue instructions to insure the integrity of the FPU stack.

X86 PROCESSOR SPECIFIC OPTIONS

-cpu:*type* Processor specific optimization.

-B23 do not modify the FPU control register.

+B41 disable floating point register allocation.

-B24 preserve the FPU control register.

-B111 issue instructions to insure the integrity of the FPU stack.

POWERPC PROCESSOR SPECIFIC OPTIONS

-B18 generates long branches.

+B51 disables fused floating point multiply/add instruction generation.

FORTRAN 90/95 CONTROL OPTIONS

-B19 assume pointer aliases exist.

-B80 causes the compiler to generate code to write the name of the currently executing procedure to standard out.

-B108 append trailing underscores to procedure names.

-B112 disable stack alignment.

-dq Allow more than 100 error diagnostics.

-ea Causes the `f95` compiler to abort the compilation process on the first error that it encounters.

-en Causes the compiler to issue a warning whenever the source code contains an extension to the Fortran 90/95 standard.

-eR Directs the compiler to place information in the debugger symbol tables for all structures whether or not they have associated storage

-g Generates symbol information for Fx™.

-M*nn* Suppresses messages by message number.

-m*nn* Suppresses messages by message level.

-P Instrument executable for profiling.

-V Causes the `f95` compiler to display its version number.

-v Directs the compiler to print status information as the compilation process proceeds

-w Suppresses listing of all compile-time warning messages.

FORTRAN 90/95 OPTIMIZATION OPTIONS

-O1 enables level optimization.

-O2 enables block level optimization.

FORTRAN 90/95 SOURCE FORMAT OPTIONS

-f*form* sets the form of the source file to **free**, **fixed**, or **alt_fixed**.

-W*n* sets the line length of source statements accepted by the compiler in Fixed-Form source format.

FORTRAN 90/95 COMPATIBILITY OPTIONS

-dp causes variables declared in a `DOUBLE PRECISION` statement and constants specified with the `D` exponent to be converted to the default real kind.

-ej causes all `DO` loops to be executed at least once, regardless of the initial value of the iteration count.

-i*n* set default integer size to ***n*** (4 or 8) bytes.

-N113 set default real size to 8 bytes (`KIND=8`).

-p path specify module search path

-s allocate local variables statically

-Rb generate code to check array boundaries.

-Rc generate code to validate substring indexes.

-Rp generate code to check for null pointers.

-Rs generate code check array conformance.

-t*n* this option increases the default temporary string size to $1024x10^n$ bytes.

-x*directive* disable compiler directive in the source file.

-YCFRL forces the compiler to pass `g77/f2c` compatible `CHARACTER` arguments.

-YCOM_NAMES specify `COMMON` block names externally in upper or lower case.

-YCOM_PFX specify `COMMON` block external name prefix.

-YCOM_SFX specify `COMMON` block external name suffix.

-YCSLASH directs the compiler to transform certain escape sequences marked with a '\' embedded in character constants.

-YEXT_NAMES Specify procedure names externally in upper, lower, or mixed case.

-YEXT_PFX Specify procedure external name prefix.

-YEXT_SFX Specify procedure external name suffix.

-YMS7D Recognize Microsoft style compiler directives beginning with a '$' in column 1.

-YNDFP disallow the use of a '.' as a structure field separator.

-YPEI pointers are Equivalent to Integers allows a Cray-style pointer to be manipulated as an integer.

FORTRAN 77 CONTROL OPTIONS

-A suppress alignment warnings.

-B19 used when more than one symbolic name is used to reference a variable's memory location. This can occur when pointers are used, when variables in COMMON are passed as arguments, or when two dummy arguments are the same actual argument.

-B80 causes the compiler to generate code to write the name of the currently executing procedure to standard out.

-B108 append trailing underscores to procedure names.

-B112 disable stack alignment.

-C generates code to check that array indexes are within array bounds - file names and source code line numbers will be displayed with all run time error messages

-D used to define conditional compilation variables from the command line **(-D *name*[=value])** — if *value* is not present, the variable is assigned the value of 1

-g generates symbol information for Fx™.

-I*path* specify path to search for INCLUDE files.

-N32 directs the compiler to issue a warning whenever the source code contains an extension to the ANSI FORTRAN 77 standard

-N52 check syntax only.

-N90 forces the compiler to pass g77/f2c compatible CHARACTER arguments.

-N111 directs the compiler to place information in the debugger symbol tables for all structures whether or not they have associated storage

-N116 `BLOCK DATA` code section.

-P instrument executable for profiling.

-q suppress non-diagnostic output.

-T*nn* used to change the number of handles used internally by the compiler.

-t*nn* modifies the default temporary string size to *nn* bytes from the default of 1024 bytes

-v directs the compiler to print status information as the compilation process proceeds

-w suppresses listing of all compile-time warning messages

-x replaces any occurrence of `X` or `D` in column one with a blank character: allows a restricted form of conditional compilation

FORTRAN 77 OPTIMIZATION OPTIONS

-H*nn* set loop unrolling limit.

-h*nn* set loop unrolling factor.

-N5 treat DATA as constants.

-N18 inline function decomposition.

-N41 evaluate constant intrinsic functions.

-N86 enable address expression opimization.

-O1 enables level optimization.

-O2 enables block level optimization.

FORTRAN 77 SOURCE FORMAT OPTIONS

-8 directs the compiler to accept source code written in Fortran 90/95 Free Source Form

-N112 directs the compiler to accept source code written in IBM VS Free Form

-V directs the compiler to accept VAX Tab-Format source code

-W directs the compiler to accept statements which extend beyond column 72 up to column 132

FORTRAN 77 COMPATIBILITY OPTIONS

-B108 causes the compiler to define `SUBROUTINE` and `FUNCTION` names with a trailing underscore

-d causes all `DO` loops to be executed at least once, regardless of the initial value of the iteration count (FORTRAN 66 convention)

-f folds all symbolic names to lower case

-i*n* changes the default storage length of `INTEGER` from 4 bytes to ***n*** (2 or 8).

-K directs the compiler to transform certain escape sequences marked with a '\' embedded in character constants

-N1 causes all variables initialized with `DATA` statements to be stored as static variables.

-N2 uses only double precision or double complex transcendental intrinsics

-N3 includes record length information for sequential unformatted files

-N4 suppresses any run-time `CASE DEFAULT` error messages

-N7 extends the sign of a value returned from `BYTE,` and `WORD` intrinsic functions

-N16 `[FMT=]` format specifier may be omitted in an I/O statement when `[UNIT=]` unit specifier is present

-N20 directs the compiler to always evaluate operators of equal precedence from left to right (except for exponential operators)

-N22 don't mangle `COMMON` block names with leading "_c"

-N26 force the compiler to consider the byte ordering of all unformatted files to be big-endian by default

-N27 force the compiler to consider the byte ordering of all unformatted files to be little-endian by default

-N33 causes structure fields to be "packed" — allocated with no space between them

-N34 automatically align `COMMON` block variables

-N51 if a file is opened as `DIRECT` access `UNFORMATTED,` causes the value set with `RECL` to be interpreted as the number of 32 bit words in a record instead of the number of bytes

-N102 directs the compiler to zero extend `INTEGER*1` variables to unsigned entities with a range of 0-255 when loaded from memory

-N109 folds all symbolic names to UPPER CASE

-N113 changes `REAL` and `COMPLEX` data types without explicit length declara-tion to `DOUBLE PRECISION` and `DOUBLE COMPLEX`

-N114 issue a warning diagnostic, rather than an error, for undeclared variables in the presence of an `IMPLICIT NONE` declaration

-N115 Pad source lines to column 72 (or 132 with **–W** option)

-s forces all program storage to be treated as static: see **-N1** also

Appendix B

ASCII Table

ASCII codes 0 through 31 are control codes that may or may not have meaning on Linux. They are listed for historical reasons and may aid when porting code from other systems. Codes 128 through 255 are extensions to the 7-bit ASCII standard and the symbol displayed depends on the font being used; the symbols shown below are from the Times New Roman font. The Dec, Oct, and Hex columns refer to the decimal, octal, and hexadecimal numerical representations.

Character	Dec	Oct	Hex	Description
NULL	0	000	00	null
SOH	1	001	01	start of heading
STX	2	002	02	start of text
ETX	3	003	03	end of text
ECT	4	004	04	end of trans
ENQ	5	005	05	enquiry
ACK	6	006	06	acknowledge
BEL	7	007	07	bell code
BS	8	010	08	back space
HT	9	011	09	horizontal tab
LF	10	012	0A	line feed
VT	11	013	0B	vertical tab
FF	12	014	0C	form feed
CR	13	015	0D	carriage return
SO	14	016	0E	shift out
SI	15	017	0F	shift in
DLE	16	020	10	data link escape
DC1	17	021	11	device control 1
DC2	18	022	12	device control 2
DC3	19	023	13	device control 3
DC4	20	024	14	device control 4
NAK	21	025	15	negative ack
SYN	22	026	16	synch idle
ETB	23	027	17	end of trans blk
CAN	24	030	18	cancel
EM	25	031	19	end of medium
SS	26	032	1A	special sequence
ESC	27	033	1B	escape
FS	28	034	1C	file separator
GS	29	035	1D	group separator
RS	30	036	1E	record separator
US	31	037	1F	unit separator

Character	Dec	Oct	Hex	Description
	32	040	20	space
!	33	041	21	exclamation
"	34	042	22	quotation mark
#	35	043	23	number sign
$	36	044	24	dollar sign
%	37	045	25	percent sign
&	38	046	26	ampersand
'	39	047	27	apostrophe
(	40	050	28	opening paren
)	41	051	29	closing paren
*	42	052	2A	asterisk
+	43	053	2B	plus
,	44	054	2C	comma
-	45	055	2D	minus
.	46	056	2E	period
/	47	057	2F	slash
0	48	060	30	zero
1	49	061	31	one
2	50	062	32	two
3	51	063	33	three
4	52	064	34	four
5	53	065	35	five
6	54	066	36	six
7	55	067	37	seven
8	56	070	38	eight
9	57	071	39	nine
:	58	072	3A	colon
;	59	073	3B	semicolon
<	60	074	3C	less than
=	61	075	3D	equal
>	62	076	3E	greater than
?	63	077	3F	question mark

Character	Dec	Oct	Hex	Description
@	64	100	40	commercial at
A	65	101	41	upper case letter
B	66	102	42	upper case letter
C	67	103	43	upper case letter
D	68	104	44	upper case letter
E	69	105	45	upper case letter
F	70	106	46	upper case letter
G	71	107	47	upper case letter
H	72	110	48	upper case letter
I	73	111	49	upper case letter
J	74	112	4A	upper case letter
K	75	113	4B	upper case letter
L	76	114	4C	upper case letter
M	77	115	4D	upper case letter
N	78	116	4E	upper case letter
O	79	117	4F	upper case letter
P	80	120	50	upper case letter
Q	81	121	51	upper case letter
R	82	122	52	upper case letter
S	83	123	53	upper case letter
T	84	124	54	upper case letter
U	85	125	55	upper case letter
V	86	126	56	upper case letter
W	87	127	57	upper case letter
X	88	130	58	upper case letter
Y	89	131	59	upper case letter
Z	90	132	5A	upper case letter
[	91	133	5B	opening bracket
\	92	134	5C	back slash
]	93	135	5D	closing bracket
^	94	136	5E	circumflex
_	95	137	5F	underscore
`	96	140	60	grave accent
a	97	141	61	lower case letter
b	98	142	62	lower case letter
c	99	143	63	lower case letter
d	100	144	64	lower case letter
e	101	145	65	lower case letter
f	102	146	66	lower case letter
g	103	147	67	lower case letter
h	104	140	68	lower case letter
i	105	151	69	lower case letter
j	106	152	6A	lower case letter
k	107	153	6B	lower case letter
l	108	154	6C	lower case letter
m	109	155	6D	lower case letter
n	110	156	6E	lower case letter
o	111	157	6F	lower case letter
p	112	160	70	lower case letter
q	113	161	71	lower case letter
r	114	162	72	lower case letter
s	115	163	73	lower case letter
t	116	164	74	lower case letter
u	117	165	75	lower case letter
v	118	166	76	lower case letter
w	119	167	77	lower case letter
x	120	170	78	lower case letter
y	121	171	79	lower case letter
z	122	172	7A	lower case letter
{	123	173	7B	opening brace
|	124	174	7C	vertical bar
}	125	175	7D	closing brace

Character	Dec	Oct	Hex	
~	126	176	7E	tilde
	127	177	7F	delete
□	128	200	80	
□	129	201	81	
‚	130	202	82	
ƒ	131	203	83	
„	132	204	84	
…	133	205	85	
†	134	206	86	
‡	135	207	87	
ˆ	136	210	88	
‰	137	211	89	
Š	138	212	8A	
‹	139	213	8B	
Œ	140	214	8C	
□	141	215	8D	
□	142	216	8E	
□	143	217	8F	
□	144	220	90	
‘	145	221	91	
’	146	222	92	
“	147	223	93	
”	148	224	94	
•	149	225	95	
–	150	226	96	
—	151	227	97	
˜	152	230	98	
™	153	231	99	
š	154	232	9A	
›	155	233	9B	
œ	156	234	9C	
□	157	235	9D	
□	158	236	9E	
Ÿ	159	237	9F	
	160	240	A0	
¡	161	241	A1	
¢	162	242	A2	
£	163	243	A3	
¤	164	244	A4	
¥	165	245	A5	
¦	166	246	A6	
§	167	247	A7	
¨	168	250	A8	
©	169	251	A9	
ª	170	252	AA	
«	171	253	AB	
¬	172	254	AC	
-	173	255	AD	
®	174	256	AE	
¯	175	257	AF	
°	176	260	B0	
±	177	261	B1	
²	178	262	B2	
³	179	263	B3	
´	180	264	B4	
µ	181	265	B5	
¶	182	266	B6	
·	183	267	B7	
¸	184	270	B8	
¹	185	271	B9	
º	186	272	BA	
»	187	273	BB	

Character	Dec	Oct	Hex
¼	188	274	BC
½	189	275	BD
¾	190	276	BE
¿	191	277	BF
À	192	300	C0
Á	193	301	C1
Â	194	302	C2
Ã	195	303	C3
Ä	196	304	C4
Å	197	305	C5
Æ	198	306	C6
Ç	199	307	C7
È	200	310	C8
É	201	311	C9
Ê	202	312	CA
Ë	203	313	CB
Ì	204	314	CC
Í	205	315	CD
Î	206	316	CE
Ï	207	317	CF
Ð	208	320	D0
Ñ	209	321	D1
Ò	210	322	D2
Ó	211	323	D3
Ô	212	324	D4
Õ	213	325	D5
Ö	214	326	D6
×	215	327	D7
Ø	216	330	D8
Ù	217	331	D9
Ú	218	332	DA
Û	219	333	DB
Ü	220	334	DC
Ý	221	335	DD

Character	Dec	Oct	Hex
Þ	222	336	DE
ß	223	337	DF
à	224	340	E0
á	225	341	E1
â	226	342	E2
ã	227	343	E3
ä	228	344	E4
å	229	345	E5
æ	230	346	E6
ç	231	347	E7
è	232	350	E8
é	233	351	E9
ê	234	352	EA
ë	235	353	EB
ì	236	354	EC
í	237	355	ED
î	238	356	EE
ï	239	357	EF
ð	240	360	F0
ñ	241	361	F1
ò	242	362	F2
ó	243	363	F3
ô	244	364	F4
õ	245	365	F5
ö	246	366	F6
÷	247	367	F7
ø	248	370	F8
ù	249	371	F9
ú	250	372	FA
û	251	373	FB
ü	252	374	FC
ý	253	375	FD
þ	254	376	FE
ÿ	255	377	FF

Appendix C

Bibliography

FORTRAN 90/95

These books and manuals are useful references for the Fortran 90/95 programming language and the floating point math format used by Absoft Pro Fortran on Linux.

Michael Metcalf and John Reid, *FORTRAN 90/95 explained*, Oxford University Press (1996)

Walter S. Brainerd, Charles H. Goldberg, and Jeanne C. Adams, *Programmer's Guide to Fortran90*, Unicomp, Inc (1994)

Jeanne C. Adams, Walter S. Brainerd, Jeanne T. Martin, and Brian T. Smith, *Fortran Top 90*, Unicomp, Inc (1994)

James F. Kerrigan, *Fortran 90*, O'Reilly & Associates, Inc (1993)

American National Standard Programming Language Fortran 90, X3.198-1991, ANSI, 1430 Broadway, New York, N.Y. 10018

COMPUTER, *A Proposed Standard for Binary Floating-Point Arithmetic*, Draft 8.0 of IEEE Task P754, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720 (1981)

FORTRAN 77

These books and manuals are useful references for the FORTRAN language and the floating point math format used by Absoft Pro Fortran on Linux.

Page, Didday, and Alpert, *FORTRAN 77 for Humans*, West Publishing Company (1983)

Kruger, Anton, *Efficient FORTRAN Programming*, John Wiley & Sons, Inc. (1990)

Loren P. Meissner and Elliot I. Organick, *FORTRAN 77*, Addison-Wesley Publishing Company (1980)

Harry Katzan, Jr., *FORTRAN 77*, Van Nostrand Reinhold Company (1978)

J.N.P. Hume and R.C. Holt, *Programming FORTRAN 77*, Reston Publishing Company, Inc. (1979)

Harice L. Seeds, *FORTRAN IV*, John Wiley & Sons (1975)

Jehosua Friedmann, Philip Greenberg, and Alan M. Hoffberg, *FORTRAN IV, A Self-Teaching Guide*, John Wiley & Sons, Inc. (1975)

James S. Coan, *Basic FORTRAN*, Hayden Book Company (1980)

American National Standard Programming Language FORTRAN, X3.9-1978, ANSI, 1430 Broadway, New York, N.Y. 10018

COMPUTER, *A Proposed Standard for Binary Floating-Point Arithmetic*, Draft 8.0 of IEEE Task P754, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720 (1981)

M. Abramowitz and I.E. Stegun, *Handbook of Mathematical Functions*, U.S. Department of Commerce, National Bureau of Standards (1972)

Appendix D

Technical Support

The Absoft Technical Support Group will provide technical assistance to all registered users of current products. They will *not* answer general questions about operating systems, operating system interfaces, graphical user interfaces, or teach programming. For further help on these subjects, please consult this manual and any of the books and manuals listed in the bibliography.

Before contacting Technical Support, please study this manual and the language reference manuals to be sure your problem is not covered here. Specifically, refer to the chapter **Using the Compilers** in this manual. To help Technical Support provide a quick and accurate solution to your problem, please include the following information in any correspondence or have it available when calling.

Product Information:

Name of product, version number, and serial number
Version number of the operating system

System Configuration:

Hardware configuration (hard drive, memory, etc.)
System software release (i.e. 4.0, 3.5, etc)
Any software or hardware modifications to your system

Problem Description:

What happens?
When does it occur?
Provide a small (20 line) step-by-step example if possible.

Contacting Technical Support:

Address: Absoft Corporation
Attn: Technical Support
2781 Bond Street
Rochester Hills, MI 48309

telephone:	(248) 853-0095	9am - 3pm EST
FAX	(248) 853-0108	24 Hours
email	support@absoft.com	24 Hours
World Wide Web	http://www.absoft.com	

Index