Appendix D: Implementation Specifics

This appendix describes implementation-specific details of the iWarp FORTRAN compiler.

Data types

Table D-1 shows the types, sizes, and ranges of the data types in iWarp FORTRAN:

Table D-1: Data Type Implementation

Data Type	**Size**	**Range**	
		min	**max**
CHARACTER	byte	-128	+127
CHARACTER*n	n bytes	-128	+127
COMPLEX	2 words	-3.402827E+38	+3.402827E+38
COMPLEX*8	2 words	-3.402827E+38	+3.402827E+38
COMPLEX*16	2 doubles	-1.7976931348623157E+308	+1.7976931348623157E+308
DOUBLE PRECISION	double	-1.7976931348623157E+308	+1.7976931348623157E+308
INTEGER	word	-2,147,483,648	+2,147,483,647
INTEGER*1	byte	-128	+127
INTEGER*2	half-word	-32,768	+32,767
INTEGER*4	word	-2,147,483,648	+2,147,483,647
LOGICAL	word	-2,147,483,648	+2,147,483,647
LOGICAL*1	byte	-128	+127
LOGICAL*2	half-word	-32,768	+32,767
LOGICAL*4	word	-2,147,483,648	+2,147,483,647
REAL	word	-3.402827E+38	+3.402827E+38
REAL*4	word	-3.402827E+38	+3.402827E+38
REAL*8	double	-1.7976931348623157E+308	+1.7976931348623157E+308

NOTE

Ranges for complex numbers apply to each part, real and imaginary.

Type sizes are:

- byte 8 bits
- half-word 16 bits
- word 32 bits
- double 64 bits

Implementation of LOGICAL Type

In iWarp FORTRAN, .TRUE. is represented by 1, and .FALSE. is represented by 0. A LOGICAL variable is represented by an integer of the same size. For example, a LOGICAL*1 would be represented by an INTEGER*1. In conditional tests, even numbers are considered true and odd numbers are considered false. Logical operators are implemented efficiently by bit-wise operators (.AND., .OR. and .NOT.).

When using VMS extensions, it is possible to assign any integer value into a LOGICAL variable. Since .TRUE. is 1 and .FALSE. is 0, tests for equality to .TRUE. and .FALSE. may both be false if other values are assigned to LOGICAL variables.

Since VMS FORTRAN represents .TRUE. as -1, iWarp FORTRAN's use of 1 for .TRUE. can cause VMS FORTRAN programs to fail. Similarly, programs written for the iWarp system which expect .TRUE. to be 1 would fail when run on a VAX. Such programs can be written, but in practice should not exist.

Programs should use .EQV. rather than .EQ. when comparing logicals for equality, and .NEQV. rather than .NEQ. when comparing for inequality of logicals. Similarly, a logical used as a logical should only be assigned .TRUE., .FALSE., or the result of a logical operation.

Persistence of Local Variables

The iWarp FORTRAN compiler and its optimizers implement local variables (variables not declared using SAVE) as automatic variables, causing them to be non-persistent. You can ovverride this default behavior with the -static option to the FORTRAN compiler. However, using the -static option prevents the optimizers from doing many important optimizations, so the option should be avoided unless absolutely necessary.

The option is supplied to allow compilation of programs which do not conform to the FORTRAN 77 standard. Such programs assume a persistence of objects which is not guaranteed by the FORTRAN 77 standard, but was sometimes used by programs written to the FORTRAN 66 standard.

If the -static option is needed, it is worthwhile to update the programs to conform to FORTRAN 77 by selectively adding SAVE to variable declarations that need to be persistent. Once updated to conform to FORTRAN 77, such programs would be more portable, no longer requiring the -static option, and would optimize much better than when the -static option is used.

Temporary Compiler Files

The compiler produces temporary files that are generally created and destroyed in the `/usr/tmp` directory without ever being seen. If you do not want `/usr/tmp` to be used for temporary files, use the environment variable `TMPDIR` to override the default use of `/usr/tmp` for temporary files while compiling. TMPDIR must point to a writable and existing directory.