3 iWarp FORTRAN Extensions

The iWarp FORTRAN compiler supports the following iWarp-specific extensions to the FORTRAN programming language:

- communication support (send and receive primitives)
- condition code checking support (iWarp condition codes)
- assembly code inlining
- pragma support for inlining

Communication Support (SEND and RECEIVE)

The iWarp FORTRAN compiler recognizes send and receive pseudo-functions for sending and receiving data from either of two logical gates. The logical gates are named GATE0 and GATE1 by f77/gates.h, which you must include in your program when using send or receive. There are send and receive pseudo-functions corresponding to each data type as shown in the following list:

receive:

INTEGER	_RECEIVEI(*channel*)
REAL	_RECEIVEF(*channel*)
DOUBLE PRECISION	_RECEIVED(*channel*)

send:

CALL	_SENDI(*channel, int-data*)
CALL	_SENDF(*channel, real-data*)
CALL	_SENDD(*channel, double-data*)

channel is either GATE0 or GATE1
int-data is an integer expression
real-data is a real expression
double-data is a double precision expression

NOTE

Integer and real communication both use a single word (32 bits) to transfer data between cells. Double-precision communication uses two words (64 bits).

You must write programs so that data sent using _SENDI is received using _RECEIVEI, data sent using _SENDF is received using _RECEIVEF, and data sent using _SENDD is received using _RECEIVED. The compiler does not check for conflicting data types in send and receive pseudo-functions.

Condition Code Checking Support (COND and CONDCALL)

You can check iWarp condition codes and use them for the following:

- creating a boolean value (.FALSE. or .TRUE.)
- conditional branching
- conditionally calling a function

The iWarp FORTRAN compiler provides two pseudo-functions to support this functionality: _COND and _CONDCALL.

_COND

The syntax of the _COND pseudo-function is as follows:

LOGICAL _COND(*condition-code*)

condition-code is a symbol defined by either the file ‘‘f77/iwarp_cc.h” or the file ‘‘f77/iwarp_math_cc.h”, depending on the symbol.

The _COND pseudo-function tests a single iWarp condition and can be used for any of the purposes previously listed. The value of _COND is .TRUE. if the condition tested is true and .FALSE. if the condition is false.

_CONDCALL

The syntax of the _CONDCALL pseudo-function is as follows:

CALL _CONDCALL (*function-name, condition-code*)

function-name is the name of a SUBROUTINE which is specified in an EXTERNAL statement.

condition-code is a symbol defined by either include ‘‘f77/iwarp_cc.h” or include ‘‘f77/iwarp_math_cc.h”, depending on the symbol.

The _CONDCALL pseudo-function places a call to the named function if the condition tested is true.

You can also use the negative of the condition code to reverse the meaning of the condition. So the following syntax would force the function to be called if the condition is not true:

CALL _CONDCALL (*function-name, - condition-code*)

Unsafe Condition Codes

You should not use _COND or _CONDCALL to check any unsafe condition codes. Unsafe conditions occur synchronous to the code being executed. This includes all conditions generated by arithmetic instructions.

Appendix B contains complete lists of the condition codes that are associated with safe (asynchronous) and unsafe (synchronous) conditions. Only the asynchronous condition codes are safe for use in _COND and _CONDCALL constructs. The safe condition codes are defined in ‘‘f77/iwarp_cc.h”. The unsafe condition codes are defined in ‘‘f77/iwarp_math_cc.h”.

Assembly Code Inlining

The assembly code inlining (asm-inline) facility allows you to define assembly code templates that the compiler can insert into the code it generates. Asm-inlines may have input parameters

and an optional return value. An asm-inline is generally expanded in the code rather than as a subroutine, thus eliminating the overhead associated with the subroutine call. Asm-inline parameters are matched as one of the following:

- constants
- values in registers (modifiable or non-modifiable)
- register numbers (control space or general register file)

To use inline assembly code in FORTRAN, do the following:

1. Write the assembly inline templates and the functions that ‘‘wrap” the assembly inline templates in C.
2. Compile the functions and templates using the C compiler -K option.
3. Compile the resulting .K file with the FORTRAN compiler.

There is a library of asm-inline templates available for use. The template definitions are found in the asm subdirectory of the default include area for C (reached by <asm/*.h>).

NOTE

You may need control space register name definitions to invoke certain asm-inline routines. You can include these definitions by specifying include ‘‘f77/regnums.h”.

Pragma Support for Inlining

iWarp FORTRAN has been extended to include a pragma directive to allow inline assembly code to be passed to the optimizer. The syntax of the pragma directive for assembly language inlining is as follows:

```
C$PRAGMA inline routine
```

routine is the name of a routine.

The PRAGMA inline directive specifies that calls to *routine* be expanded inline when possible instead of implemented as an actual routine call. The pragma directive must start in column one in the source code.

NOTE

Actual inlining is done by the PCO optimizer at compile time, so pragma directives are ignored unless you use the -O option to invoke the PCO optimizer when compiling the program.

You can also use the pragma directive to prevent inlining which may otherwise occur. The syntax for preventing inlining is as follows:

```
C$PRAGMA noinline routine
```

routine is the name of a routine.

Both pragma inline and pragma noinline apply to the entire file you are compiling, regardless of the position of the directive in the file. Specifying a routine name which is not defined in the file in a pragma directive will not result in an error.

Refer to Chapter 4, *Enhancing Program Performance,* for discussions of inter-file and intra-file inlining.