IEEE Standard 754 and You: What the GEM Computer User Needs to Know About IEEE Floating-Point Arithmetic

Lee A. Roberts
SSC Laboratory

May 28, 1992

Abstract:

GEM computer users, for better or for worse, will interact with the *IEEE Standard for Binary Floating-Point Arithmetic* (IEEE 754). All of today's popular RISC/UNIX architectures support the IEEE 754 standard—both in data format and exception handling. However, details of the IEEE 754 exception handling vary among the RISC/UNIX architectures. GEM code development efforts on these popular RISC/UNIX systems can be greatly enhanced with proper understanding of IEEE 754 exception handling. Suggestions for successful use of the IEEE 754 standard are presented for each of today's popular RISC/UNIX architectures.

IEEE Standard 754 and You

What the GEM Computer User Needs to Know About IEEE Floating-Point Arithmetic

Lee A. Roberts
Physics Research Division
Superconducting Super Collider Laboratory
Dallas, TX 75237

May 28, 1992

Abstract

GEM computer users, for better or for worse, will interact with the *IEEE Standard for Binary Floating-Point Arithmetic* (IEEE 754). All of today's popular RISC/UNIX architectures support the IEEE 754 standard—both in data format and exception handling. However, details of the IEEE 754 exception handling vary among the RISC/UNIX architectures. GEM code development efforts on these popular RISC/UNIX systems can be greatly enhanced with proper understanding of IEEE 754 exception handling. Suggestions for successful use of the IEEE 754 standard are presented for each of today's popular RISC/UNIX architectures.

1 Introduction

GEM computer users, some of whom are unfamiliar with today's modern RISC/UNIX environments, must successfully develop and use simulation and analysis codes on these RISC/UNIX platforms. An often-presented argument for these RISC/UNIX systems is that, in addition to their price/performance advantages, they conform to "standards" covering numerous aspects of the hardware and software operating environment. However, some of these standards specify a flexible environment which must be appropriately configured to satisfy the requirements of a given task. We must beware the empty refrain, **"It conforms to standards!"**, and learn to work within the standards environment.

The *IEEE Standard for Binary Floating-Point Arithmetic* (IEEE 754)[1] is one of the many standards within which we as GEM physicists must learn to work. GEM physicists who are moving to the RISC/UNIX environment quickly realize that the floating-point representations provided by IEEE 754 are different from the implementations on VAX or IBM System/370 architectures. (IEEE 754 representations match VAX F- and G-floating representations in exponent and fraction bit counts, but different interpretation of the exponent field results in narrower (by one) and offset (by plus two) exponent ranges for IEEE 754 values.) GEM physicists must become acquainted with the requirements of IEEE 754 regarding floating-point exception handling and the specific implementations on today's popular RISC/UNIX platforms. Proper configuration of the floating-point exception-handling environment will enhance the code development environment and thereby assist the GEM physicist in avoiding "inexplicable" program failures.

One of the complexities of the IEEE 754 standard for binary floating-point arithmetic is that it provides **choices**. Choices are complicated things; correct selections often require a detailed understanding of the options. Of course, IEEE 754 defines default behaviors, with a goal to

> minimize for users the complications arising from exceptional conditions. The arithmetic system is intended to continue to function on a computation as long as possible, handling unusual situations with reasonable default responses, including setting appropriate flags.[1]

In a code development environment, however, the GEM physicist may not want an incorrect computation to continue "as long as possible," but would rather have immediate notification of the incorrect floating-point result. IEEE 754 provides appropriate choices for this environment as well—but these are not, in my view, the default IEEE choices.

This paper will concentrate upon proper non-default IEEE 754 settings for the GEM software development environment on each of today's popular RISC/UNIX workstations. Examples for Silicon Graphics, Sun, Digital, Hewlett-Packard, and IBM workstations will be presented. A "general" GEM floating-point software development environment interface will be presented which consolidates the workstation-specific information into one convenient FORTRAN and C programming interface.

2 Features of IEEE Standard 754

Many features of the *IEEE Standard for Binary Floating-Point Arithmetic* (IEEE 754) are discussed in the *Numerical Computation Guide*[2] written by Sun Microsystems, Inc. Please review this manual for a detailed discussion of IEEE floating-point computation issues.

IEEE Standard 754 specifies:[2]

- ◇ Hardware storage formats for IEEE double precision (64 bits) and IEEE single precision (32 bits).
- ◇ Accuracy requirements on basic floating-point operations: add, subtract, multiply, divide, square root, remainder and compare. These operations may not suffer more than one rounding error.
- ◇ Accuracy requirements for conversions between formats.
- ◇ Five IEEE floating-point exceptions (invalid, division by zero, overflow, underflow, inexact), and when these exceptions should be indicated to the user.
- ◇ Four rounding modes (round toward the nearest representable value, round toward zero, round toward $+\infty$, round toward $-\infty$).
- ◇ Rounding precision (that is, if a computer delivers results only in extended precision format, you should be able to specify that the result is nonetheless rounded to single precision format, then stored with the trailing zeros).

As further explanation of the IEEE floating-point exceptions, the IBM *AIX XL FORTRAN Compiler/6000 User's Guide Version 2.2*[3] states

> The IEEE standard for floating-point arithmetic specifies that five types of exceptions be signaled when detected. These are Invalid Operation, Division by Zero, Overflow, Underflow and Inexact. By default, the signaling of an exception involves setting a status flag and continuing. Optionally, an exception should generate a trap and invoke a handler routine specified by a user.[3]

[1] IEEE Standard 754[1], p. 18

[2] *Numerical Computation Guide*[2], Sun Microsystems, Inc., pp. 3–4

[3] *AIX XL FORTRAN Compiler/6000 User's Guide Version 2.2*[3], International Business Machines Corp., p. 37

The following discussion will concentrate upon the user control and handling of the five IEEE floating-point exceptions. The IEEE standard provides a variety of choices in addition to the default results. Correct choices are important to efficient and successful development of GEM physics simulation and analysis codes.

2.1 Default IEEE Results

Sun's *Numerical Computation Guide* defines *default results* as follows:[4]

> The term *default result* refers to the value that is delivered as the result of a floating-point operation that causes an exception. IEEE Standard 754-1985 defines default results for the five exceptions. The intent of the Standard, in specifying these defaults, is to allow you to continue computing, with some sensible result, but also with a definite indication that an exception occurred.
>
> In fact, the IEEE Standard defines two sets of default results: those returned when the exception is *not* trapped by a signal handler (the usual case), and those returned when the exception *is* trapped by a signal handler. However, UNIX signal handlers have no provision for specifying the latter results.

The following table[5] presents the default IEEE result (when not trapped by a signal handler) from various floating-point calculations.

Calculation	*Default Result*	*IEEE Exception*
big * big	+Inf	overflow
big * (−)big	−Inf	overflow
number/0.0	+Inf (for *number* > 0.0)	division
number/0.0	−Inf (for *number* < 0.0)	division
0.0/0.0	`NaN`	invalid
small/big	*subnormal number*	underflow
2.0/3.0	2/3, *rounded to destination precision*	inexact

Table 1: Default results for IEEE 754 floating-point exceptions.

IEEE Standard 754 chooses gradual underflow as the preferred method for dealing with underflowed results. Gradual underflow uses subnormal values to extend the lower range of representable floating-point numbers. Gradual underflow means that errors due to underflow are no worse than usual roundoff error. Abrupt underflow, where the underflowed result is replaced by zero, is also available on most modern RISC/UNIX systems. However, hardware implementation of underflow varies among floating-point units; in some cases, software may be used to (partially) implement either underflow scheme.

On most of today's popular RISC/UNIX workstations, the default IEEE behavior is provided as the default programming environment. On these machines, floating-point exceptions produce default results and hardware exceptions are raised, but no interrupts or traps are generated. Physics calculations are continued with "numbers" such as $\pm\infty$ and `NaN` (Not-a-Number) propagating through

[4] *Numerical Computation Guide*[2], Sun Microsystems, Inc., p. 63
[5] *Numerical Computation Guide*[2], Sun Microsystems, Inc., p. 2

many levels of calculation. Only the DECstation selects non-default IEEE behavior as the standard programming environment.

2.2 Suggested IEEE Results for GEM Code Development Environment

Many GEM physicists are familiar and comfortable with the floating-point exception-handling characteristics of VAX/VMS FORTRAN. In the non-IEEE VAX/VMS environment, floating-point exceptions corresponding to the IEEE overflow, division by zero and invalid exceptions are trapped and cause program termination. Such an environment, although non-default, may be selected within the IEEE 754 standard. In fact, Digital provides such an environment as the default DEC FORTRAN environment on the DECstation. Similar environments are available on other RISC/UNIX systems, often via compiler or loader option switches.

My suggested environment for GEM software development enables the following actions upon IEEE floating-point exceptions. These actions make the RISC/UNIX floating-point behavior similar to the VAX/VMS FORTRAN behavior. Development of GEM physics software will be greatly simplified by signaling floating-point exceptions such that they can be caught and examined by an interactive debugger.

IEEE Exception	*Action*
invalid	trap & abort
division by zero	trap & abort
overflow	trap & abort
underflow	IEEE default
inexact	IEEE default

Table 2: Suggested IEEE 754 Results for GEM Software Development Environment.

3 Implementation of GEM Floating-Point Software Development Environment

GEM software developers must clearly understand the difference between the following actions:

- Indication of a floating-point exception according to IEEE 754.
- Generation of a program interrupt in response to a floating-point exception.

All systems which support IEEE 754 floating-point must detect and indicate floating-point exceptions. On each of today's popular RISC/UNIX systems, floating-point exceptions are indicated by raising an exception bit in the system's *floating-point control and status register* (`fpcsr`).

Generation of a program interrupt in response to a floating-point exception is an optional action according to the IEEE 754 standard. (IEEE 754 requires the capability to generate interrupts; it does not require generation of the interrupts.) On the SPARC[2], MIPS[4], PA-RISC[5] and POWER[6] architectures, the `fpcsr` contains a mask of *trap enable* bits which must be raised to cause hardware interrupts and generation of a floating-point exception signal (`SIGFPE`). On the POWER architecture, an additional floating-point exception enable bit in the *machine state register* (`MSR(FE)`) must be raised to enable floating-point exceptions and force the process to execute in serial (rather than pipelined) mode. Due to the inefficiencies imposed by serial execution on a pipelined processor,

IBM supports (and recommends) software checking of floating-point exceptions and generation of trap signals (SIGTRAP) as a more efficient mechanism for program interrupts. Whether hardware- or software-generated, these floating-point exception signals may be caught and handled using the standard UNIX signal-handling techniques—either with an interactive debugger or by a user-written or system-default exception handler.

The following subsections discuss implementation of a GEM floating-point software development environment on each of the popular RISC/UNIX workstations. In each case, the critical, machine-dependent step is the enabling of the floating-point interrupts—whether by hardware or software. Without the floating-point interrupts and the generation of SIGFPE or SIGTRAP signals, the standard UNIX signal-handling software is helpless.

3.1 Silicon Graphics

Silicon Graphics provides a simple, easy-to-use interface to control IEEE floating-point exception handling. The same interface is supported for FORTRAN and C (and Pascal) programs. Two steps are required to enable this interface, as follows:

- ◇ Load library `libfpe.a` when loading a binary executable by including the library specification "`-lfpe`" on the compiler driver (`f77`,`cc`) command line.
- ◇ Set environment variable `TRAP_FPE` to be "`DIVZERO=ABORT;OVERFL=ABORT;INVALID=ABORT`".

These actions fully establish the GEM software development environment for IEEE 754 floating-point exceptions suggested previously. No FORTRAN or C programming changes to GEM software are necessary.

Silicon Graphics provides a rich IEEE floating-point exception-handling environment which may be easily controlled by the `TRAP_FPE` environment variable. The value suggested above is only the bare minimum to establish a VAX/VMS-like environment where floating-point exceptions cause program interrupts. Please consult the `fsigfpe(3F)` and `sigfpe(3)` manual pages[7] for a full description.

With the above setting for `TRAP_FPE`, floating-point exceptions signal `SIGFPE` and are caught by the various system debuggers, including `dbx(1)`, `edge(1)` and `cvd(1)` (CodeVision Debugger). This will enable the GEM software developer to immediately trace computational errors due to floating-point exceptions.

If `TRAP_FPE` is not defined, program execution remains the default Silicon Graphics environment, which corresponds to the IEEE 754 default.

For the advanced programmer, the `fsigfpe(3F)` and `sigfpe(3)` manual pages present FORTRAN and C programming interfaces to `handle_sigfpes`, the floating-point exception handler package. Direct access to the floating-point control and status register (`fpcsr`) is provided by the `fpc(3C)` interface, for the do-it-yourself programming fanatic.

3.2 Sun

Sun provides a simple, easy-to-use interface to create an IEEE floating-point exception-handling environment suitable for GEM code development and testing. The same interface is supported for FORTRAN and C programs. Two steps are required to enable this interface, as follows:

- ◇ Load the main program (either FORTRAN or C) with the compiler option "`-fnonstd`" on the compiler driver (`f77`,`cc`) command line.
- ◇ When `dbx(1)` is invoked for debugging, enter the command "`catch FPE`" to cause `dbx` to listen for floating-point exceptions.

One side-effect of the "`-fnonstd`" compiler option is the activation of abrupt underflow. This underflow behavior does not correspond to the suggested GEM software development environment, but may be viewed as a minor exception. This exception may be corrected via software modifications. The following software modification may be used to correct this side-effect, if deemed necessary:

- ◇ In FORTRAN programs, insert the following as the first executable statement:
  `call gradual_underflow()`
- ◇ In C programs, insert the following as the first executable statement:
  `gradual_underflow_();`

These actions establish the GEM software development environment for IEEE 754 floating-point exceptions suggested previously. No FORTRAN or C programming changes to GEM software are necessary to implement an adequate software development environment; only minor FORTRAN or C programming changes to GEM software are necessary to implement the full environment.

For the advanced programmer, `f77_ieee_environment(3F)`[8] and `ieee_handler(3M)`[9] manual pages present FORTRAN and C programming interfaces to `ieee_handler`, the floating-point exception handler package. Please consult the `f77_ieee_environment(3F)` and `ieee_handler(3M)` manual pages for a full description.

3.3 Digital

Digital's default environment for DEC FORTRAN[10] has been configured to match, as closely as possible, the standard VAX/VMS environment. As such, the default compilation options ("`-fpe0`") provide IEEE floating-point exception trapping for division-by-zero, overflow and invalid exceptions. In addition, abrupt underflow is enabled via the system-default trap handler. This underflow behavior does not correspond to the suggested GEM software development environment. When debugging in this environment,

- ◇ With `dbx(1)`, `SIGFPE` is caught by default and underflow exceptions will cause interrupts. Use the command "`ignore FPE`" to allow the system-default trap handler to perform its actions upon `SIGFPE`, which include generation of `SIGTRAP` for IEEE divide-by-zero, overflow and invalid. Be aware that `dbx(1)` will stop inside the trap handler, but the stack trace will show the exception location.
- ◇ With the FUSE debugger (`fuse(1)`), `SIGFPE` is ignored by default and underflow exceptions do not cause interrupts. `SIGTRAP` signals generated by the system-default trap handler are caught and provide debugging locations for floating-point exceptions. Be aware that the FUSE debugger will stop inside the trap handler, but the stack trace will show the exception location.

Full compliance with the suggested GEM software development environment may be achieved via program modifications. DEC provides functions `for_get_fpe(3f)` which provide user-control of the trap enable bits in the MIPS floating-point control and status register. The following changes to DEC FORTRAN programs may be added to implement gradual underflow:

- ◇ Add the following lines at the beginning of the FORTRAN program:

```
include '/usr/include/for_fpe_flags.f'
integer for_get_fpe, for_set_fpe
ifpe = for_set_fpe(iand(for_get_fpe(),not(FPE_M_TRAP_UND)))
```

Gradual underflow is the IEEE default, therefore no SIGFPE signals are generated for IEEE underflow. Debuggers (dbx(1),fuse(1)) may catch SIGFPE for IEEE divide-by-zero, overflow and invalid exceptions without difficulty, providing a direct indication of the floating-point exception, when using the full GEM software development environment. Advanced programmers should consult the for_get_fpe(3f) manual page for more details.

DEC FORTRAN provides a selection of "-fpe" options which provide different levels of IEEE floating-point exception handling and reporting. Compiler option "-fpe3" corresponds to the default IEEE 754 behavior. Users should note the following conditions:

- ◇ The "-fpe" options are *compiler* options rather than *loader* options.
- ◇ IEEE floating-point exception-handling characteristics of a DEC FORTRAN executable are determined by the "-fpe" compilation option of the main program. Only the main program need be recompiled to change the floating-point exception-handling characteristics.

DEC C provides default IEEE 754 floating-point exception handling, which does not correspond to the suggested GEM software development environment. No options to the compiler driver (c89) are provided to change the floating-point exception-handling environment. The suggested GEM software development environment may be created with the following programming changes:

- ◇ Include the following C source lines in the program's initialization code.

```
#include <mips/fpu.h>
union fpc_csr fpcsr;
fpcsr.fc_word = get_fpc_csr();
fpcsr.fc_struct.en_invalid = 1;
fpcsr.fc_struct.en_divide0 = 1;
fpcsr.fc_struct.en_overflow = 1;
fpcsr.fc_word = set_fpc_csr(fpcsr.fc_word);
```

Advanced programmers should consult the fpc(3)[11] manual pages for additional details.

These actions establish the GEM software development environment for IEEE 754 floating-point exceptions suggested previously. No FORTRAN programming changes to GEM software are necessary to implement an adequate software development environment; relatively small FORTRAN or C programming changes to GEM software are necessary to implement the full environment.

3.4 Hewlett-Packard

Hewlett-Packard provides an interface to IEEE floating-point exception handling that requires the GEM software developer to slightly modify the FORTRAN source code to obtain a suitable debugging environment. The Hewlett-Packard FORTRAN/9000[12] debugging option "+T" provides a partial solution; when applied to the FORTRAN main program, it causes the application to trap IEEE division-by-zero, invalid, overflow and *underflow* exceptions. GEM physics software applications will generate many underflow exceptions; such exceptions should be appropriately masked by gradual (preferred) or abrupt (acceptable) underflow conventions and should not cause program abort.

The suggested GEM software development environment may be created with a slight modification to the FORTRAN main program. Use the following two steps to implement the suggested environment under HP FORTRAN/9000:

- ◇ Add the following FORTRAN statement as the first executable statement in the main program. This statement will lower the trap enable bit for underflow in the floating-point status register.

```
ON REAL UNDERFLOW IGNORE
```

◇ Compile the FORTRAN main program with the "+T" option to the FORTRAN/9000 compiler driver (`fort77`,`f77`).

Use of the "`ON REAL UNDERFLOW IGNORE`" statement without the "+T" compiler option causes a compilation warning and lowers the underflow trap enable bit at run time, but otherwise has no effect on program execution.

Upon first glance, Hewlett-Packard provides neither a C compiler option nor a C programming interface to change the floating-point control and status register trap enable bits. However, the *HP Pascal/HP-UX Programmer's Guide*[14] documents the `HPENBLTRAP` subprogram available in `libcl.a`, which provides the HP Pascal and HP FORTRAN run-time libraries. `HPENBLTRAP` is an MPE XL compatibility subprogram and its use in C programs is documented only in reference to packed-decimal arithmetic and in a software status bulletin. `HPENBLTRAP` is, however, used by the HP FORTRAN "+T" trap handler. The GEM floating-point software development environment may be established using `HPENBLTRAP` within a C program using the following steps:

◇ Add the following lines to the program's initialization code:
```
int oldmask;
HPENBLTRAP(0x00070000,&oldmask);
```

◇ Load library `libcl.a` when loading a binary executable by including the library specification "`-lcl`" on the compiler driver (`cc`,`c89`) command line.

Advanced programmers may wish to study the documentation on "Traps" in the *HP Pascal/HP-UX Programming Guide* for more details.

Appendix A presents assembly-language interface subprograms which have been adapted from examples provided by a Hewlett-Packard software engineer.[15] This C programming interface to the floating-point status register may be used as follows to implement the GEM floating-point software development environment under HP C:

◇ Include the following lines in the program's initialization code, using the header file defined in Appendix A.
```
#include "float_traps.h"
enable_fp_traps();
```

◇ Load the program, including the `hp_fpcsr.o` object resulting from assembly of `hp_fpcsr.s` presented in Appendix A.

◇ Default floating-point trap action will be to core dump, or if within the interactive debugger, to break with a floating-point exception. A user-written signal-handler may be provided, if desired.

Advanced programmers may wish to carefully study Appendix A for details on this HP C programming interface.

3.5 IBM

Like the SPARC, MIPS and PA-RISC architectures, the IBM POWER[6] architecture provides a floating-point status and control register, including trap enable bits for floating-point exceptions. However, support for floating-point exception handling differs substantially from the other architectures due to the POWER architecture's heavily-pipelined implementation. Floating-point exception handling on the RISC System/6000 is explained as follows.

For trap-enabled exceptions, two methods are available to transfer program execution from the application to the appropriate trap handler when an exception occurs: software polling and hardware interrupt. Software polling has its advantage in performance. Software can select when to poll for a possible enabled exception. For example, if the Divide by Zero exception is the only trap-enabled exception, the compiler can place the polling *branch and link on exception* instruction after each floating-point divide instruction. This method can be used if the kind of exception handling can be determined at the time a program is compiled.

In order to provide a precise hardware interrupt (precise in that the address of the excepting instruction is saved in a register accessible by the trap handler) on a floating point exception on such a heavily pipelined implementation, the entire processor is put in a mode of executing only one instruction at a time. All instructions must complete before the next instruction is dispatched, including fixed-point instructions. This method allows traps to be enabled or disabled at run time.[6]

Under AIX 3.2, IBM provides high-level programming interfaces for both software polling and hardware interrupt mechanisms for floating-point exceptions. (Previous versions of AIX provided either software polling or no high-level language support.)

3.5.1 Software Polling

AIX XL FORTRAN Compiler/6000 2.2 and *AIX XL C Compiler/6000 1.2* support software polling of floating-point interrupts via the "`-q fltrap`" compiler option. This causes the compiler to generate code which will generate a trap signal (`SIGTRAP`) to flag the occurrence of any enabled floating-point exception. All of the code which might generate floating-point exceptions, not just the main program, must be compiled with the "`-q fltrap`" compiler option to allow proper traceback of floating-point exceptions for debugging purposes. According to IBM, "The code required to detect floating-point exceptions will reduce the performance of programs compiled with the **FLTTRAP** option."[7]

Implementation of the suggested GEM software development environment on the IBM RISC System/6000 using software polling requires the following steps:

- Add the following lines to the FORTRAN main program.

```
include '/usr/include/fpdc.h'
include '/usr/include/fexcp.h'
fpstat(fpve) = .true.
fpstat(fpoe) = .true.
fpstat(fpze) = .true.
call fpsets(fpstat)
call signal(sigtrap,xl_trce)
```

- Add the following block data subprogram.

```
block data
include '/usr/include/fpdc.h'
include '/usr/include/fpdt.h'
end
```

[6] *RISC System/6000 Floating-Point Unit, IBM RISC System/6000 Technology*[16], pp. 41–42
[7] *AIX XL FORTRAN Compiler/6000 User's Guide Version 2.2*[3], p. 37

- ◇ Compile **all** code which could produce floating-point interrupts with the "`-q fltrap`" compiler option.

Similarly, C programs may implement the suggested GEM software development environment using software polling with the following steps:

- ◇ Add the following lines to the program's initialization code.
  ```
  #include <fptrap.h>
  fp_enable( TRP_INVALID | TRP_DIV_BY_ZERO | TRP_OVERFLOW );
  ```
- ◇ Compile **all** code which could produce floating-point interrupts with the "`-q fltrap`" compiler option.

Advanced programmers should consult the IBM RISC System/6000 manuals[17] for `fp_any_enable`, etc., for more details.

The above steps successfully create the suggested GEM software development environment on the IBM RISC System/6000. Users must be aware of the following problems:

- ◇ Use of the `fpsets` or `fp_enable` subroutines without the "`-q fltrap`" compiler option causes incorrect results. IEEE floating-point exceptions do **not** cause interrupts and the values returned are neither reasonable nor the IEEE default values.
- ◇ **All** of the code which might generate floating-point exceptions, not just the main program, **must** be compiled with the "`-q fltrap`" compiler option to allow proper traceback of floating-point exceptions for debugging purposes.

Software polling of floating-point exceptions requires a substantial amount of effort by the programmer. Not only are software modifications to GEM main programs required, but large amounts of code—including many subroutine libraries—need to be compiled with the "`-q fltrap`" option. Furthermore, setting the IEEE trap enable bits with `fpsets` or `fp_enable` produces incorrect results when not using "`-q fltrap`," requiring separate main program versions for software development and production.

3.5.2 Hardware Interrupts

Under AIX 3.2, generation of hardware interrupts in response to floating-point exceptions is supported on the IBM RISC System/6000. The `fp_trap` subroutine may be used to change the floating-point exception enable bit in the machine state register (`MSR(FE)`) to enable floating-point exceptions and force the process to execute in serial (rather than pipelined) mode. Use of the hardware interrupts will allow easy access to floating-point exceptions and generation of `SIGFPE` signals throughout the executable, but will impose a substantial performance penalty.

No FORTRAN-callable interface for the hardware interrupt mechanism is provided by IBM. However, the following steps may be used to implement a FORTRAN interface:

- ◇ Add the following line as the first FORTRAN executable statement:
  ```
  CALL FP_ENABLE_TRAP
  ```
- ◇ Create and compile the following C interface subroutine for use as a FORTRAN-callable interface.
  ```
  #include <fptrap.h>
  void fp_enable_trap_()
  {
  ```

```
int oflag;
fp_enable( TRP_INVALID | TRP_DIV_BY_ZERO | TRP_OVERFLOW );
oflag = fp_trap( FP_TRAP_SYNC );
}
```

- Load the FORTRAN program, including the `fp_enable_trap_.o` object on the compiler driver (`f77,xlf`) command line.

Similarly, C programs may implement the suggested GEM software development environment using hardware interrupts with the following steps:

- Add the following lines to the program's initialization code.

```
#include <fptrap.h>
int oflag;
fp_enable( TRP_INVALID | TRP_DIV_BY_ZERO | TRP_OVERFLOW );
oflag = fp_trap( FP_TRAP_SYNC );
```

Advanced programmers should consult the IBM RISC System/6000 manual pages for `fp_any_enable`, `fp_trap`, etc., for more details.

The above steps successfully create the suggested GEM software development environment on the IBM RISC System/6000. Hardware interrupts eliminate the requirement to recompile all code using special compiler options, but require the sacrifice of pipelined execution. According to IBM, "System performance with the `MSR(FE)` bit set to 1 can be significantly degraded."[8]

4 Summary

Successful development of GEM simulation and analysis codes will depend upon efficient use of today's popular RISC/UNIX workstations. GEM physicists need to understand the *IEEE Standard for Binary Floating-Point Arithmetic* (IEEE 754) to better enjoy its benefits (and to avoid its consequences for the unwary).

A GEM software development environment for IEEE floating-point exception handling has been defined and can be implemented on each of today's popular RISC/UNIX workstations with varying ease or difficulty. For ease in code development and debugging, the suggested GEM software development environment handles floating-point exceptions in a VAX/VMS-like manner. IEEE 754 divide-by-zero, overflow and invalid exceptions are configured to cause program abort with core dump and may be efficiently caught and examined by an interactive debugger. Use of IEEE gradual underflow is recommended, but abrupt underflow is acceptable for many debugging purposes on machines which select this option by default.

Implementation details of the GEM software development environment for IEEE floating-point exception handing are provided for today's popular RISC/UNIX workstations. In the easiest case (Silicon Graphics), no FORTRAN source modifications are necessary and the IEEE floating-point exception-handling environment can be changed from default IEEE to GEM software development by simply changing an environment variable. On Sun, Digital and Hewlett-Packard systems, full implementation of the suggested GEM software development environment requires slight FORTRAN modifications and requires a recompile/relink of the main program to change from default IEEE to GEM software development environments. On the IBM RISC System/6000, both software polling and hardware interrupt mechanisms require FORTRAN code modifications; software polling requires

[8] *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference—General Information Manual*[6], International Business Machines Corporation

the recompilation of all source to implement the GEM software development environment; hardware interrupts sacrifice pipelined program execution and imply a significant speed reduction.

A common FORTRAN and C programming interface for the GEM floating-point software development environment using hardware interrupts may be constructed for all of today's popular RISC/UNIX workstations. With this general programming interface, GEM computer users need only add a single line to the FORTRAN or C main program to implement the GEM floating-point software development environment. However, it is strongly recommended that GEM computer users understand the implementation of this general interface on the RISC/UNIX workstation of interest. Use of this general interface implies a heavy performance penalty on IBM RISC System/6000 workstations. Other vendors are developing superscalar and/or superpipelined RISC implementations as well; control methods and efficiency on these future processors will be determined when these machines are generally available. Implementation details for this general programming interface are presented in Appendix B.

Features of the general programming interface presented in Appendix B are minimal. Users may select between default IEEE floating-point and GEM floating-point software development environment behaviors at run-time; no complex options or controls are provided, unlike some of the vendor-written interfaces. Only basic trap-and-abort exception handling is provided; some of the vendor-specific trap-handlers provide traceback information. Features of the general programming interface include:

- ◇ Common programming interface across today's popular RISC/UNIX workstations. Interface is available for both FORTRAN and C programs. Invoke this interface using the following procedures:
  - FORTRAN programmers should add the following line as the first executable statement: `CALL GEMFPE`
  - C programmers should add the following line as the first executable statement: `GEMfpe();`
- ◇ Load programs including the `GEMfpe.o` object and any other required objects or libraries, such as the `hp_fpcsr.o` object under HP-UX.
- ◇ Select between default IEEE 754 floating-point and GEM floating-point software development environments with the `GEMfpe` environment variable. Run-time environment selection is performed as follows:
  - `GEMfpe` undefined $\Longrightarrow$ default IEEE 754 floating-point environment
  - `GEMfpe` defined $\Longrightarrow$ GEM floating-point software development environment

GEM computer users are invited to test both the vendor-supplied IEEE floating-point interfaces and the common `GEMfpe` interface provided in Appendix B. Comments, suggestions and enhancements are welcome!

A Appendix A

Hewlett-Packard provides an MPE XL compatibility subprogram in the Pascal and FORTRAN run-time libraries which may be used as a C programming interface to the PA-RISC floating-point status register. This appendix presents a collection of subprograms which provide a more general C programming interface to the PA-RISC floating-point status register. This set of assembly-language subprograms has been adapted from examples provided by a Hewlett-Packard software engineer.[15] The C programming interface is defined via the `float_traps.h` header file reproduced below.

```
/*
 * float_traps.h prototype
 */

#ifndef __FLOAT_TRAPS_H__
#define __FLOAT_TRAPS_H__

/*
 * codes for kinds of floating-point traps.
 */

#define ASSIST_TRAP        0x0e
#define FP_INEXACT_RESULT  0x01
#define FP_UNDERFLOW       0x02
#define FP_OVERFLOW        0x04
#define FP_DIVIDE_BY_ZERO  0x08
#define FP_INVALID_OP      0x10

#ifdef __STDC__

extern void enable_fp_traps(void);
extern void disable_fp_traps(void);
extern void set_fp_traps(unsigned int);
extern int  get_fp_status(void);
extern int  get_fp_traps(void);

#else

extern void enable_fp_traps();
extern void disable_fp_traps();
extern void set_fp_traps();
extern int  get_fp_traps();
extern int  get_fp_status();

#endif  /* __STDC__ */

#endif  /* __FLOAT_TRAPS_H__ */
```

Assembly-language subprograms from `hp_fpcsr.s` for manipulation of the PA-RISC floating-point status register are reproduced on the next page.

```
;
; These routines make up an interface for manipulating the FP status
; register on an hp9000 s8xx machine.
;

;
; get_fp_status returns the contents of the FP status register as is
;
        .SPACE $TEXT$
        .SUBSPA $CODE$

get_fp_status
        .PROC
        .CALLINFO
        .ENTRY
        ldo     4(%sp),%sp
        fstws   %fr0,0(%sp)
        ldw     -0(%sp),%ret0
        bv      %r0(%rp)
        ldo     -4(%sp),%sp
        .PROCEND

;
; set_fp_status blindly sets the FP status register from arg0.  It's a
; dangerous routine, and is not exported.
;
set_fp_status
        .PROC
        .CALLINFO
        .ENTRY
        ldo     4(%sp),%sp
        stw     %arg0,0(%sp)
        fldws   0(%sp),%fr0
        bv      %r0(%rp)
        ldo     -4(%sp),%sp
        .PROCEND

;
; get_fp_traps returns the trap enable bits from the FP status register
; in the lowest 5 bits of ret0
;
get_fp_traps
        .PROC
        .CALLINFO
        .ENTRY
        stw     %rp,-20(%sp)
        bl      get_fp_status,2
        nop
```

```
        ldw     -20(%sp),%rp
        bv      %r0(%rp)
        dep     0,26,27,%ret0           ; zap out top 27 bits
        .PROCEND

;
; set_fp_traps sets the trap enable bits in the FP status register from the
; 5 lowest bits in arg0.
;
set_fp_traps
        .PROC
        .CALLINFO
        .ENTRY
        b       set_fp_status
        dep     0,26,27,%arg0           ; zap out top 27 bits
        .PROCEND

;
; enable_fp_traps calls set_fp_traps with a reasonable default value to
; enable the "common" FP traps (invalid operation, divide-by-zero and overflow).
;
enable_fp_traps
        .PROC
        .CALLINFO
        .ENTRY
        b       set_fp_status
        ldi     0x1c,%arg0              ; VZOui
        .PROCEND

;
; disable_fp_traps calls set_fp_traps with a zero mask to disable all
; floating-point traps.
;
disable_fp_traps
        .PROC
        .CALLINFO
        .ENTRY
        b       set_fp_status
        copy    %r0,%arg0               ; vzoui
        .PROCEND

        .EXPORT get_fp_status,ENTRY,PRIV_LEV=3
        .EXPORT get_fp_traps,ENTRY,PRIV_LEV=3
        .EXPORT set_fp_traps,ENTRY,PRIV_LEV=3
        .EXPORT enable_fp_traps,ENTRY,PRIV_LEV=3
        .EXPORT disable_fp_traps,ENTRY,PRIV_LEV=3

        .END
```

B Appendix B

The following C subprogram implements a common interface for the GEM floating-point software development environment. Use an ANSI C compiler when compiling this subprogram.

```
/*

   GEMfpe  --  A general implementation of the GEM floating-point
               software development environment within IEEE 754.

   The GEM floating-point software development environment is characterized
   by the following actions on IEEE floating-point exceptions:

               invalid              trap & abort
               divide by zero       trap & abort
               overflow             trap & abort
               underflow            IEEE default (gradual)
               inexact              IEEE default

   GEMfpe provides both the default IEEE floating-point environment and
   the GEM floating-point software development environment.  Selection
   is based upon the existence of the environment variable GEMfpe.

               GEMfpe undefined ==> IEEE default environment
               GEMfpe defined   ==> GEM floating-point environment

   Operating systems supported include:

               IRIX 4.0.1
               ULTRIX 4.2A
               HP-UX 8.07
               AIX 3.2
               SunOS 4.1.1

   FORTRAN interface:

               CALL GEMFPE

   C interface:

               GEMfpe();


   Author:     Lee A. Roberts
   Location:   Superconducting Super Collider Laboratory
   Date:       May 27, 1992

*/
```

```
#include <stdlib.h>
#if   defined (__sgi)
#include <sys/fpu.h>
#include <signal.h>
#elif defined (__ultrix)
#include <mips/fpu.h>
#include <signal.h>
#elif defined (__hpux)
#include "float_traps.h"
#include <signal.h>
#elif defined (_AIX)
#include <fptrap.h>
#include <signal.h>
#elif defined (sun)
#include <math.h>
#include <floatingpoint.h>
#endif

void GEMfpe()
{
   if (getenv("GEMfpe") == NULL) {
#if   defined (__sgi) || defined (__ultrix)
      union fpc_csr fpcsr;
      fpcsr.fc_word = get_fpc_csr();
      fpcsr.fc_struct.en_invalid   = 0;
      fpcsr.fc_struct.en_divide0   = 0;
      fpcsr.fc_struct.en_overflow  = 0;
      fpcsr.fc_struct.en_underflow = 0;
      fpcsr.fc_struct.en_inexact   = 0;
      fpcsr.fc_word = set_fpc_csr(fpcsr.fc_word);
      signal(SIGFPE,SIG_IGN);
#elif defined (__hpux)
      disable_fp_traps();
      signal(SIGFPE,SIG_IGN);
#elif defined (_AIX)
      int oflag;
      fp_disable_all();
      oflag = fp_trap(FP_TRAP_OFF);
      signal(SIGFPE,SIG_IGN);
#elif defined (sun)
      int i;
      i = ieee_handler("clear","all",SIGFPE_IGNORE);
#endif
   } else {
#if   defined (__sgi) || defined (__ultrix)
      union fpc_csr fpcsr;
      fpcsr.fc_word = get_fpc_csr();
      fpcsr.fc_struct.en_invalid   = 1;
```

```
      fpcsr.fc_struct.en_divide0   = 1;
      fpcsr.fc_struct.en_overflow  = 1;
      fpcsr.fc_struct.en_underflow = 0;
      fpcsr.fc_struct.en_inexact   = 0;
      fpcsr.fc_word = set_fpc_csr(fpcsr.fc_word);
      signal(SIGFPE,SIG_DFL);
#elif defined (__hpux)
      enable_fp_traps();
      signal(SIGFPE,SIG_DFL);
#elif defined (_AIX)
      int oflag;
      fp_disable_all();
      fp_enable(TRP_INVALID | TRP_DIV_BY_ZERO | TRP_OVERFLOW);
      oflag = fp_trap(FP_TRAP_SYNC);
      signal(SIGFPE,SIG_DFL);
#elif defined (sun)
      int i;
      i = ieee_handler("clear","all",SIGFPE_DEFAULT);
      i = ieee_handler("set","common",SIGFPE_ABORT);
#endif
   }
}

void gemfpe_()
{
   GEMfpe();
}
```

References

[1] *IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1985* (IEEE 754), published by the Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017, 1985.

[2] *Numerical Computation Guide*, Sun Microsystems, Inc., Mountain View, CA. Part No. 800-3555-10, Revision A of 16 March 1990.

[3] *AIX XL FORTRAN Compiler/6000 User's Guide Version 2.2*, International Business Machines Corporation, Armonk, NY, September 1991.

[4] *Assembly Language Programmer's Guide Version 1.0*, Silicon Graphics, Inc., Mountain View, CA. Document 007-0730-010, 1987.

[5] *HP Precision Architecture and Instruction Set*, Hewlett-Packard, USA. HP Part No. 09740-90014, April 1989.

[6] *IBM RISC System/6000 POWERstation and POWERserver Hardware Technical Reference—General Information Manual*, International Business Machines Corporation, Austin, TX. 1992.

[7] IRIX 4.0.1 Operating System, Silicon Graphics, Inc., Mountain View, CA. 1991.

[8] *Sun FORTRAN Reference Manual*, Sun Microsystems, Inc., Mountain View, CA. Part No. 800-3418-10, Revision A of 16 March 1990.

[9] *SunOS Reference Manual*, Sun Microsystems, Inc., Mountain View, CA. Part No. 800-3827-10, Revision A of 27 March 1990.

[10] *DEC Fortran for ULTRIX RISC Systems User Manual*, Digital Equipment Corporation, Maynard, MA. Order Number AA-PE1CA-TE, March 1991.

[11] *ULTRIX Reference Pages*, Digital Equipment Corporation, Maynard, MA. Order Number AA-LY16B-TE, 1990.

[12] *FORTRAN/9000 Reference*, Hewlett-Packard Company, USA. HP Part No. B2408-90003, June 1991.

[13] *HP-UX Operating System 8.07*, Hewlett-Packard Company, USA. November 1991.

[14] *HP Pascal/HP-UX Programming Guide*, Hewlett-Packard Company, USA. HP Part No. 92431-9006, January 1991.

[15] Private communication. Stuart Jarriel, Hewlett-Packard, Austin, TX.

[16] *IBM RISC System/6000 Technology*, International Business Machines Corporation, Austin, TX. 1990.

[17] *AIX Technical Reference: Base Operating System and Extensions*, International Business Machines Corporation, Austin, TX. January 1992.