Floating Point Exceptions

Ruud van der Pas

SiliconGraphics

European High Performance Computing Team

Floating Point Exceptions

What is a Floating Point Exception (FPE) ?

An FPE occurs when something "unusual" happens in the system, like:

- **division by zero**
- **overflow**
- **underflow**
- **.....**

If an FPE occurs, the system takes a specified action:

- **Abort**
- **Ignore the situation**
- **....**

Floating Point Exceptions

- **Divide by zero**
  The division by zero exception is signaled on an implemented divide operation if the divisor is zero and the dividend is a finite nonzero number. The result, when no trap occurs, is a correctly signed infinity
- **Underflow**
  One is the creation of a tiny non–zero result between the minimum expressible exponent which, because it is tiny, may cause some other exception later. The other is extraordinary loss of accuracy during the approximation of such tiny numbers by denormalized numbers
- **Overflow**
  The overflow exception is signaled when what would have been the magnitude of the rounded floating–point result, were the exponent range unbounded, is larger than the destination format's largest finite number
- **Invalid**
  One or both of the operands are invalid for an implemented operation. The result (if not trapped) is NaN for fp numbers and not defined for fixed–point numbers.
- **Inexact**
  If the rounded result of an operation is not exact or if it overflows without an overflow trap, then the inexact exception is signaled.
- **Unimplemented operation**
  Normally not user–visible

Floating Point Exceptions

IRIX supports an FPE library:

- **The library functions can be called directly**

  This means that you can write your own trap handler

- **Can be used through the TRAP_FPE environment variable**

  This is a very convenient way to trap errors

- **More information: "man handle_sigfpes"**
- **Make sure to link with the FPE library: –lfpe**

TRAP_FPE environment variable

Possible actions that can be defined:

- Abort
- Count
- Trace
- Substitute a value

Examples:

% setenv TRAP_FPE "all=count,abort(10),trace(10)'

% setenv TRAP_FPE "OVERFL=TRACE(2),ABORT(2);
UNDERFL=COUNT,ZERO;
DIVZERO=TRACE(1),ABORT(1)"

FPEs on the R8000

- The R4400 and R10000 support *precise* exceptions
- The R8000 doesn't have this feature
- Two "modes" can be used:

fpmode precise "command"
slow – but accurate exceptions

fpmode performance "command"
fast – but possibly inaccurate exception handling

default !

FPEs on the R10000

On the R10000, exceptions are

EXPENSIVE

How to detect this:

- Relatively high system time
- Monitor the job with osview or gr_osview
- Use SpeedShop to trace FPEs

FPEs and optimizations

- The compiler can sometimes do more optimizations *if* it is allowed to generate code that may cause exceptions
- A compiler option can be used to control this:

–TENV: X=0..5

X=0 : Enable IEEE–754 exceptions.
X=1 : Disable inexact and underflow IEEE–754 exceptions.
X=2 : Disable all IEEE–754 exceptions except divide–by–zero.
X=3 : Disable all IEEE–754 exceptions.
X=4 : Disable (ignore) memory exceptions.
X=5 : Disable (ignore) all exceptions.

Use this option with great care !

Examples exception handlers

Extracted from the man page (available for Fortran and C)

To get the details: `% man handle_sigfpes`

This example sets up traps of underflow which change the resulting value to zero.

```
#include <fsigfpe.h>
C       call this during program startup
C       to set underflowing values to zero
C
C       underflow to zero

        fsigfpe(FPE_UNDERFL).repls = FPE_ZERO
C
C       only trap on underflow
        call handle_sigfpes(_ON, FPE_EN_UNDERFL ,
     2                      %val(0), FPE_ABORT_ON_ERROR, %val(0));
```


Examples exception handlers/2

This is a more complicated example. The example counts all traps, traces the first five exceptions of each kind, aborts on the first divide by zero, or the 100th overflow. It replaces zero for underflows, max float/double for overflows, max integer for integer overflows, and the default values for divide by zero, invalid operands, and integer overflows.

```
#include <fsigfpe.h>

C   underflow to zero

      fsigfpe(FPE_UNDERFL).repls = FPE_ZERO

C     substitute max real/double precision
C     value on overflow

      fsigfpe(FPE_OVERFL).repls=FPE_MAX

C     trace after 5

      fsigfpe(FPE_UNDERFL).trace=5
      fsigfpe(FPE_OVERFL).trace =5
      fsigfpe(FPE_DIVZERO).trace=5
      fsigfpe(FPE_INVALID).trace=5
      fsigfpe(FPE_INT_OVERFL).trace=5

C     counts at end

      fsigfpe(FPE_UNDERFL).count=2147483647
      fsigfpe(FPE_OVERFL).count =2147483647
      fsigfpe(FPE_DIVZERO).count=2147483647
      fsigfpe(FPE_INVALID).count=2147483647
      fsigfpe(FPE_INT_OVERFL).count=2147483647
```

```
C     abort after 100 underflows or 100 overflows

      fsigfpe(FPE_UNDERFL).abort=100
      fsigfpe(FPE_OVERFL).abort =100

C     abort on first divide by zero or hundredth invalid

      fsigfpe(FPE_DIVZERO).abort=1
      fsigfpe(FPE_INVALID).abort=100


      call handle_sigfpes(FPE_ON,
     x                    FPE_EN_UNDERFL+FPE_EN_OVERFL+
     x                    FPE_EN_DIVZERO+FPE_EN_INVALID,
     x                    %val(0),
     x                    FPE_ABORT_ON_ERROR,
     x                    %val(0))

C     do the real application work here
```


Examples exception handlers/3

How to enable different handlers for various exception types:

```
#include <limits.h> /* to get INT_MAX */
#include <sigfpe.h>

extern user_t my_invalid_handler;

main()
{
    /* counts at end */

    sigfpe_[_UNDERFL].count=INT_MAX;
    sigfpe_[_OVERFL].count =INT_MAX;
    sigfpe_[_DIVZERO].count=INT_MAX;
    sigfpe_[_INVALID].count=INT_MAX;
    sigfpe_[_INT_OVERFL].count=INT_MAX;

    /* enable trapping on overflow, using libfpe's trap handler */

    handle_sigfpes(_ON, _EN_OVERFL, 0, 0, 0);

    /* enable trapping on invalid, using user's own trap handler */

    handle_sigfpes(_ON, _EN_INVALID, my_invalid_handler, 0, 0);

    /* do the real application work here */

    ....

    /* turn off trapping of overflows */

    handle_sigfpes(_OFF, _EN_OVERFL, 0, 0, 0);

    ....

    /* turn off all handling of SIGFPE signals */

    handle_sigfpes(_OFF, 0, 0, 0, 0);

    ....
```