We have mentioned things like overflow, underflow. What are other exceptional situations?

Motivation: usually when exceptional condition like 1/0 happens, you may want to know.

IEEE requires vendors to provide a way to get status flags.

IEEE defines five exceptions: overflow, underflow, division by zero, invalid operation, inexact.

overflow: larger than the maximal floating-point number.
Underflow: smaller than the smallest floating-point number

- Invalid:
  \[ \infty + (-\infty), 0 \times \infty, 0/0, \infty/\infty, \]
  \[ x \text{ REM } 0, \infty \text{ REM } y, \sqrt{x}, x < 0, \text{ any comparison involves a NaN} \]

- Invalid returns NaN; NaN may not be from invalid operations

- Inexact: the result is not exact
  \[ \beta = 10, p = 3, 3.5 \times 4.2 = 14.7 \text{ exact}, \]
  \[ 3.5 \times 4.3 = 15.05 \Rightarrow 15.0 \text{ not exact} \]
Exception, Flags, Trap handlers III

inexact exception is raised so often, usually we ignore it

<table>
<thead>
<tr>
<th>Exception</th>
<th>when trap disabled</th>
<th>argument to handler</th>
</tr>
</thead>
<tbody>
<tr>
<td>overflow</td>
<td>$\pm \infty$ or $\pm 1.1 \cdots 1 \times 2^{e_{\text{max}}}$</td>
<td>$\text{round}(x2^{-\alpha})$</td>
</tr>
<tr>
<td>underflow</td>
<td>$0, \pm 2^{e_{\text{min}}}$, or denormalized</td>
<td>$\text{round}(x2^{\alpha})$</td>
</tr>
<tr>
<td>division by zero</td>
<td>$\infty$</td>
<td>operands</td>
</tr>
<tr>
<td>invalid</td>
<td>NaN</td>
<td>operands</td>
</tr>
<tr>
<td>inexact</td>
<td>round($x$)</td>
<td>round($x$)</td>
</tr>
</tbody>
</table>

- Trap handler: special subroutines to handle exceptions
You can design your own trap handlers

- In the above table, “when trap disabled” means results of operations if trap handlers not used
- $\alpha = 192$ for single, $\alpha = 1536$ for double
  - reason: you cannot really store $x$
- Examples of using trap handlers described later
Compiler Options I

- Compiler may provide a way so the program stops if an exception occurs
- Easy for debugging
- Example: SUN’s C compiler
  An outdated machine, but concepts are similar
- For the compiler gcc, it doesn’t have the functionality to explicitly detect exceptions
- `-ftrap=t`
  t: `%
all, %none, common, [no%]invalid, [no%]overflow, [no%]underflow, [no%]division, [no%]inexact.`
common: invalid, division by zero, and overflow.
The default is \(-f\text{trap}=%\text{none}\).
Example: \(-f\text{trap}=%\text{all},\text{no}\%\text{inexact}\) means to set all traps, except inexact.
If you compile one routine with \(-f\text{trap}=t\), compile all routines of the program with the same \(-f\text{trap}=t\) option
otherwise, you can get unexpected results.
Example: on the screen you will see
Note: IEEE floating-point exception flags raised:
Inexact; Underflow;
See the Numerical Computation Guide, ieee_flags

- gcc:
  -fno-trapping-math: default -ftrapping-math
  Setting this option may allow faster code if one relies on “non-stop” IEEE arithmetic

- -ftrapv
  Generates traps for signed overflow on addition, subtraction, multiplication
Example:

```c
    do {
        ....
    } while {not x >= 100;}
```

If \( x = \text{NaN} \), an infinite loop

Any comparison involving NaN is wrong. Thus \( x \geq 100 \) returns FALSE

Then \( \text{not } x \geq 100 \) is always true and the loop just keeps running
A trap handler can be installed to abort it

Example:

Calculate $x_1 \times \cdots \times x_n$ may overflow in the middle (the total may be fine!):

```c
for (i = 1; i <= n; i++)
    p = p * x[i];
```

$x_1 \times \cdots \times x_r, r \leq n$ overflow but $x_1 \times \cdots \times x_n$ may be in the range

$e^{\sum \log(x_i)} \Rightarrow$ a solution but less accurate and costs more
A possible solution

```c
for (i = 1; i <= n; i++) {
    if (p * x[i] overflow) {
        p = p * pow(10,-a);
        count = count + 1 ;
    }
    p = p * x[i] ;
}
p = p * pow(10, a*count) ;
```