Highlights from the Floating Point Workshop

› **13 May 2014**

Fifth International Workshop for Future Challenges in Tracking and Trigger Concepts

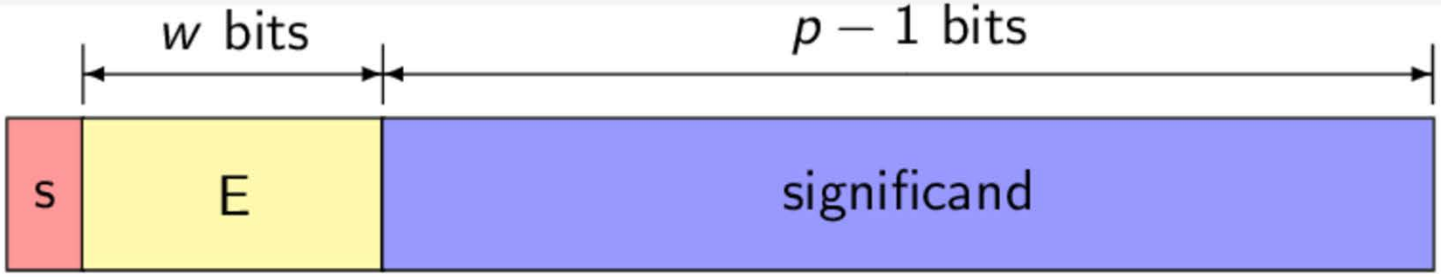
Georgios Bitzes, CERN openlab

CERN openlab

Slides, material

- Numerical workshop at CERN, 5-6 of May 2014
- All slides and material at http://indico.cern.ch/e/flp14

Storage Format of a Binary Floating-Point Number



IEEE Name	Format	Size	w	p	e_{min}	e_{max}
Binary32	Single	32	8	24	-126	+127
Binary64	Double	64	11	53	-1022	+1023
Binary128	Quad	128	15	113	-16382	+16383

Notes:

- $E = e - e_{min} + 1$
- $p - 1$ will be addressed later

Quick example

- **Using decimal base**
  - 152853.5047 has 10 digits of precision
  - Represented with significand 1.528535047
  - Exponent of 5
  - Sign: Positive

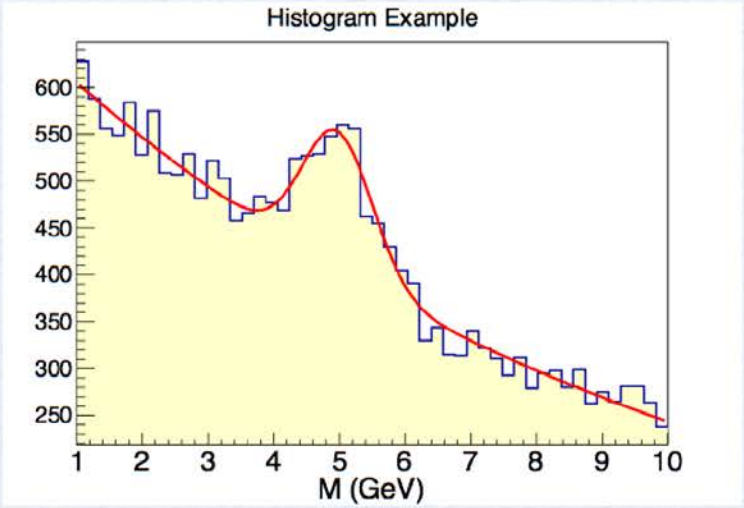
Some Inconvenient Properties of Floating-Point Numbers

Let a, b and c be floating-point numbers. Then

- $a + b$ may not be a floating-point number
  - $a + b$ may not always equal $a \oplus b$
  - Similarly for the operations $-$, $\times$ and $/$
  - Recall that floating-point numbers do not form a field
- $(a \oplus b) \oplus c$ may not be equal to $a \oplus (b \oplus c)$
  - Similarly for the operations $\ominus$, $\otimes$ and $\oslash$
- $a \otimes (b \oplus c)$ may not be equal to $(a \otimes b) \oplus (a \oplus c)$

Scaling

- Importance to try to keep numbers around 1
- Better to apply a linear transformation to the data to have location and scale around 1
  - Non-sense using for observables units not close to 1 (e.g use GeV instead of eV)
  - scale is defined by physical quantities (e.g. detector resolution)
  - use reasonable ranges
    - do not use here a scale from $1.x10^9$ to $10x10^9$ (eV)
    - fit will probably fail if you use that scale



The Patriot bug

In 1991, a Patriot missile failed to intercept a Scud, and 28 people were killed.

- The code worked with time increments of 0.1 s.
- But 0.1 is not representable in binary.
- In the 24-bit format used, the number stored was 0.099999904632568359375
- The error was 0.0000000953.
- After 100 hours = 360,000 seconds, time is wrong by 0.34s.
- In 0.34s, a Scud moves 500m

In single, we don't have that many bits to accumulate garbage in them !

Test : which of the following increments should you use ?

10 5 3 1 0.5 0.25 0.2 0.125 0.1

Challenges of elementary function coding

- Multiple targets
  - numerous architectures (Intel's x86/Xeon-Phi, Kalray's K1, ARM)
  - numerous constraints (throughput, latency, precision, memory consumption)
  - numerous programming styles (e.g. vector versus scalar)
- More function out there than the ones listed in C11 !

libm paradigm poorly addresses this complexity.

Metalibm : generate function code on demand for a given context

Expression Rearrangements

These rearrangements are not value-safe:

- $(a \oplus b) \oplus c \Rightarrow a \oplus (b \oplus c)$
- $a \otimes (b \oplus c) \Rightarrow (a \otimes b) \oplus (a \oplus c)$

To disallow these changes:

gcc Don't use `-ffast-math`

icc Use `-fp-model precise`

- Recall that options such as `-On` are "aggregated" or "composite" options
  - they enable/disable many other options
  - their composition may change with new compiler releases

Disallowing rearrangements may affect performance

The Hardware Floating-Point Environment

The hardware floating-point environment is controled by the FPU control word

- Rounding mode
- Status flags
- Exception mask
- Control of subnormals

If you change anything affecting the default state of the FPU, you must tell the compiler

- Use `#pragma STDC FENV_ACCESS ON`

icc Use `-fp-model strict`

`#pragma STDC FENV_ACCESS ON` is required if flags are accessed

Useful operators that make sense in a processor

- Should a processor include hardware elementary functions ?
  Yes (Paul&Wilson, 1976), No since the transition to RISC
- Should a processor include a divider and square root?
  Yes (Oberman et al, Arith, 1997), No since the transition to FMA (IBM then HP then Intel)
- Should a processor include decimal hardware?
  Yes say IBM, No say Intel
- Should a processor include a multiplier by log(2)?
  No of course.

Useful operators that make sense in an FPGA or ASIC

- Elementary functions ?
  Yes iff your application needs it
- Divider or square root?
  Yes iff your application needs it
- Decimal hardware?
  Yes iff your application needs it
- A multiplier by log(2)?
  Yes iff your application needs it

In FPGAs, useful means: useful to one application.

Agner Fog's instruction tables

- **Detailed measurements of latency, throughput for many architectures**
- **Latency: If I run an instruction, how much need to wait until result is available?**
- **Throughput: Number of instructions of the same kind per clock cycle**
  - Example: throughput of 3 for ADD means we can start 3 additions per clock cycle
  - Says nothing about when result will be available!

Agner Fog's instruction tables (2)

- **Reciprocal throughput: How many cycles per instruction if we don't have dependency chains**
  - Example: reciprocal throughput of 0.33 for ADD means we can start 3 additions per clock cycle
- **Measuring**
  - Latency: Issue long chain of identical instructions, each depending on the result of the previous
  - Throughput: Issue long sequence of independent instructions of the same type
- **Instruction tables available online**
  - http://agner.org/optimize

Agner Fog's instruction tables (3)

- Example: Measuring AVX2 instructions… (from testp/TestScripts/avx2.sh1

```
echo -e "\n\nLatency: $instruct r$r,r$r,r$r"  >> results1/avx2.txt
nasm -f elf64 -o b64.o -Dinstruct=$instruct -Dnumop=3 -Dregsize=$regsize \
-Dcounters=$PMCs -Dtmode=L -Plt.inc TemplateB64.nasm
if [ $? -ne 0 ] ; then exit ; fi
g++ -m64 a64.o b64.o -lpthread
if [ $? -ne 0 ] ; then exit ; fi
./a.out >> results1/avx2.txt
```

- TemplateB64.nasm constructs the snippet to be tested

Agner Fog's instruction tables (3)

```
%ifnmacro testcode
   %macro testcode 0   ; default: run instruction 100 times
      %if numop == 0
         instruct immoperands0
      %elif numop == 1
         instruct reg0 immoperands1
      %elif numop == 2
         instruct reg0, reg1 immoperands1
      %elif numop == 3
         instruct reg0, reg0, reg1 immoperands1
      %else
         %error "unknown numop"
      %endif
      instruct2
   %endmacro
%endif
```

Thanks

georgios.bitzes@cern.ch