Floating point numbers

Saulius Gražulis

Vilnius, 2021

Vilnius University, Faculty of Mathematics and Informatics
Institute of Informatics

Scientific number notation

$$602\underbrace{00\ldots0}_{\text{21 times}} = 6.02 \times 10^{23}$$

$$\pm d_0.d_1 d_2 \ldots d_{p-1} \times \beta^e = \sum_{i=0}^{p-1} d_i \beta^{-i} \times \beta^e, (0 \leq d_i < \beta)$$

$$\underbrace{602 \times 10^{21}}_{\textit{unnormalised}} = \underbrace{6.02 \times 10^{23}}_{\textit{normalised}} = \underbrace{0.602 \times 10^{24}}_{\textit{unnormalised}}$$

Floating point numbers

$$\pm d_0.d_1 d_2 \ldots d_{p-1} \times \beta^e = \pm \sum_{i=0}^{p-1} d_i \beta^{-i} \times \beta^e, (0 \le d_i < \beta)$$

$$\beta = 2$$

$$0.1_{10} \approx +1.10011001100110011001101_2 \times 2^{-4}$$

- Sign (of the significand)
- Exponent
- Significand (mantissa, fraction)

IEEE 754 Standard

- Binary ($\beta = 2$, (IEEE 1985)) and decimal ($\beta = 10$, (IEEE 2008)) formats
- Single, double, single-extended and double-extended precision (IEEE 1985), half-precision (16 bits), quad precision (128 bits) and octuple precision (256 bits) and longer interchange formats (IEEE 2008)
- Special values: not-a-number(s) (NaN), infinities ($\pm\infty$), signed zeroes (± 0)
- Denormalised numbers
- Roundoff control
- Masked exceptions
- Specifies precision of operations

IEEE 754 Standard encoding

- Significand: represented as *signed magnitude*
- Exponent: uses *biased* representation (excess $2^{n-1} - 1$ for n exponent bits)
- Exponent range: $-(2^{n-1} - 2) - +(2^{n-1} - 1)$
(e.g. $-126 - +127$ for the 8 bit eksponent)
- Fraction (significand): hidden (assumed) bit used for normalised numbers

$$0.1_{10} \approx 1.10011001100110011001101_2 \times 2^{-4}$$

Example: 0.1 in single precision FP:

p: $23 + 1$ bit e: $-126 - 127$ (8 bits); bias = $2^{8-1} - 1 = 128 - 1 = 127$
$f =$ `1.10011001100110011001101`
$e =$ `127 + (-4) = 123`$_{10}$ `= 01111011`$_2$

`0 01111011 10011001100110011001101`

Normal(ised) numbers

$$0.15625_{10} = \underbrace{0.00101_2}_{\text{not normalised}} = \pm 1.01 \times 2^{-3}$$

$$e = 127 + (-3) = 124_{10} = 01111100_2$$

Representation:
Float 32 (float; single precision):

$$\texttt{0 01111100 01 } \underbrace{\texttt{00...0}}_{\texttt{21 zero}}$$

Denormalised numbers

$$1.0_2 \times 2^{-130_{10}}$$

For a single precision floating point,

$$e_{\min} = -126_{10} \Rightarrow \text{ Can not normalise!}$$

Note that:

$$e_{\min} + \text{ bias } = 127_{10} + (-126_{10}) = 1 = \texttt{0000\_0001}_2$$

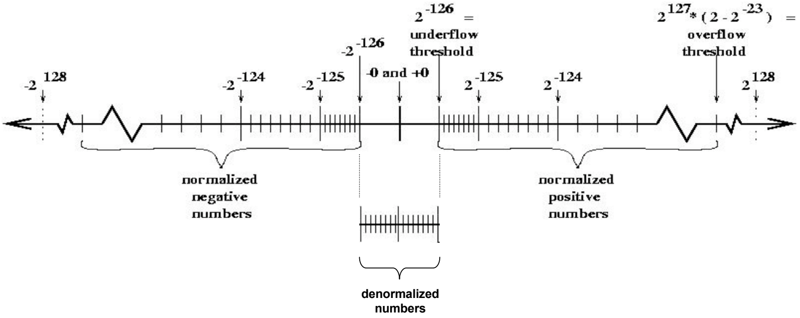
For the 0000_0000 biased exponent, interpretation is changed:

$$\texttt{0 0000 0000 0001}\underbrace{\texttt{00}\ldots\texttt{0}}_{\text{19 zeros}} = +0.0001_2 \times 2^{\mathbf{-126_{10}}} = +0.0625_{10} \times 2^{\mathbf{-126_{10}}}$$

exponent is -126, **not** -127 !

Why bother about denormalised numbers

Gradual underflow



(Engelen 2008)

```
if (a != b) { x = a/(a-b); }
```

Zeroes

$$0\ 0000\,0000\ \underbrace{00\ldots0}_{23\text{ zeros}} = 0$$

$$1\ 0000\,0000\ \underbrace{00\ldots0}_{23\text{ zeros}} = -0$$

$$\frac{1}{0} = \infty$$

$$\frac{1}{-0} = -\infty$$

```
if (a > b) { x = log(a-b); }
```

(Engelen 2008)

Infinities

Still not used the all 1s exponent, `1111_1111`

$$\texttt{0}\ \texttt{1111 1111}\ \underbrace{\texttt{00...0}}_{\text{23 zeros}} = \infty$$

$$\texttt{1}\ \texttt{1111 1111}\ \underbrace{\texttt{00...0}}_{\text{23 zeros}} = -\infty$$

$$\frac{1}{0} = +\infty; \quad \frac{1}{+\infty} = +0$$

$$\frac{1}{-0} = -\infty; \quad \frac{1}{-\infty} = -0$$

Not a number: NaN

$\texttt{0 1111 1111} \quad \underbrace{\mathbf{1}\texttt{1...0}}_{\text{not all 23 positions are zeros}} = \text{qNaN}$

$\texttt{0 1111 1111} \quad \underbrace{\mathbf{0}\texttt{1...0}}_{\text{not all 23 positions are zeros}} = \text{sNaN}$

Operations that produce NaN:

Operation	NaN produced by
$+$	$\infty + (-\infty)$
$\times$	$0 \times \infty$
$/$	$0/0,\ \infty/\infty$
rem	$0 \text{ rem } 0,\ \infty \text{ rem } \infty$
$\sqrt{}$	$\sqrt{x} \quad \forall x < 0$

(Goldberg 1991)

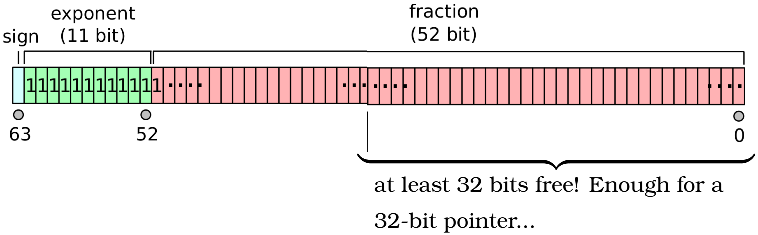
Comparison of NaNs

- Any comparison to NaN returns false $\Rightarrow$ when $x <$ NaN fails, this does not imply $x >=$ NaN
- `!(x < y)` $\not\Leftrightarrow$ `x >= y`
- `(x == y) == FALSE` when `x == NaN`
- Can not sort array of floats with NaNs

(Engelen 2008)

Using NaNs

For single precision FP, NaNs contain 21 “free” bits
For double precision FP, NaNs contain 50 “free” bits



- Dynamic languages (e.g. JavaScript) use “boxed NaN values”
- sNaN is useful for catching uninitialised values
- qNaN can represent unknown/unspecified values

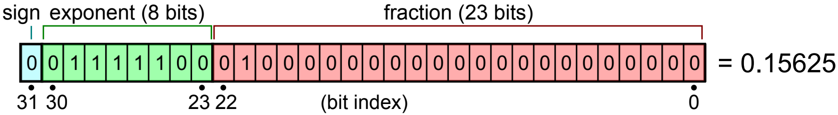
Summary: IEEE 754 special values

Exponent	Fraction	Denotes
$e = e_{\text{min}} - 1$	$f = 0$	± 0
$e = e_{\text{min}} - 1$	$f \neq 0$	$\pm 0.f \times 2^{e_{\text{min}}}$
$e_{\text{min}} \leq e \leq e_{\text{max}}$	$f = \forall n$	$\pm 1.f \times 2^{e}$
$e = e_{\text{max}} + 1$	$f = 0$	$\pm\infty$
$e = e_{\text{max}} + 1$	$f \neq 0,$	NaN

(Goldberg 1991)

Single precision FP

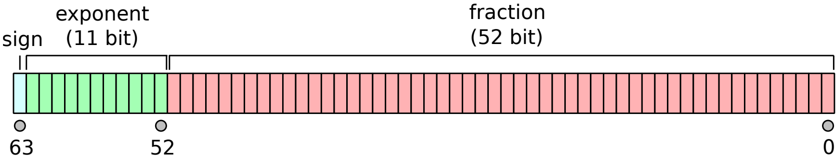
32-bit number



Vectorization: Stannered, CC BY-SA 3.0 via Wikimedia Commons

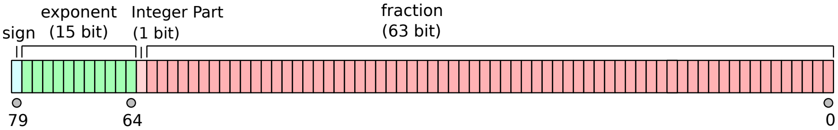
Double precision FP

64-bit number



https://en.wikipedia.org/wiki/File:IEEE_754_Double_Floating_Point_Format.svg

Intel 80 bit extended precision



BillF4, CC BY-SA 3.0, via Wikimedia Commons

- No hidden bit
- Just enough precision to compute x^y
- Intended for intermediate results only

Intel x87 FP registers

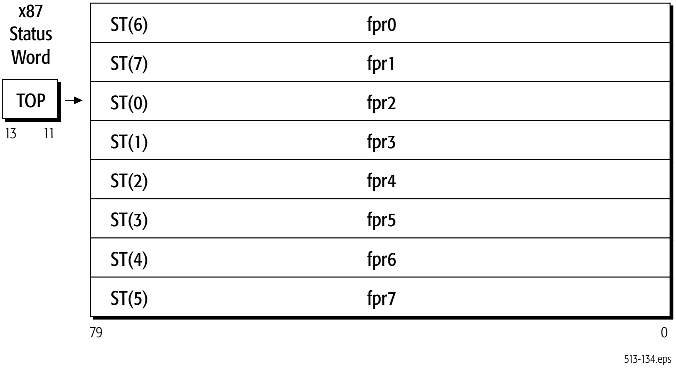


Figure 6-2. x87 Physical and Stack Registers

(AMD 2017)

Floating point status flags

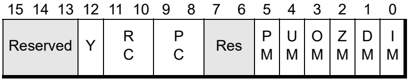
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	C3	TOP			C2	C1	C0	ES	SF	PE	UE	OE	ZE	DE	IE

Bits	Mnemonic	Description
15	B	x87 Floating-Point Unit Busy
14	C3	Condition Code
13:11	TOP	Top of Stack Pointer 000 = FPR0 111 = FPR7
10	C2	Condition Code
9	C1	Condition Code
8	C0	Condition Code
7	ES	Exception Status
6	SF	Stack Fault
	Exception Flags	
5	PE	Precision Exception
4	UE	Underflow Exception
3	OE	Overflow Exception
2	ZE	Zero-Divide Exception
1	DE	Denormalized-Operand Exception
0	IE	Invalid-Operation Exception

Figure 6-3. x87 Status Word Register (FSW)

(AMD 2017)

Floating point control flags



Bits	Mnemonic	Description
12	Y	Infinity Bit (80287 compatibility)
11:10	RC	Rounding Control
9:8	PC	Precision Control
		#MF Exception Masks
5	PM	Precision Exception Mask
4	UM	Underflow Exception Mask
3	OM	Overflow Exception Mask
2	ZM	Zero-Divide Exception Mask
1	DM	Denormalized-Operand Exception Mask
0	IM	Invalid-Operation Exception Mask

Figure 6-4. x87 Control Word Register (FCW)

(AMD 2017)

Floating point properties

- Guaranteed precision of single operations
  Except where stated otherwise, every operation shall be performed as if it first produced an intermediate result correct to infinite precision and with unbounded range, and then rounded that result according to one of the attributes in this clause.

  (IEEE 2019), sect. 4.3

- Each representable number has a single representation
- FP numbers are ordered as signed magnitude integers!
  All of the possible single-precision entities are well ordered in the natural lexicographic ordering of their machine representations interpreted as sign-magnitude binary integers

  (Cody 1981)

Example: FP 16-bit code

```
gcc -c -S \
   -m16 -O3 --omit-frame-pointer \
   -o single-precision.asm single-precision.c
```

```
float parallel( float x, float y )
{
    return x*y/(x + y);
}
```

```
parallel:
.LFB0:
        .cfi_startproc
        flds    4(%esp)
        flds    8(%esp)
        fld     %st(1)
        fmul    %st(1), %st
        fxch    %st(2)
        faddp   %st, %st(1)
        fdivrp  %st, %st(1)
        ret
```

Example: FP 64-bit code

```
gcc -c -S \
    -O3 --omit-frame-pointer \
    -o single-precision.asm single-precision.c
```

```c
float parallel( float x, float y )
{
    return x*y/(x + y);
}
```

```
parallel:
.LFB0:
        .cfi_startproc
        movaps  %xmm0, %xmm2
        addss   %xmm1, %xmm0
        mulss   %xmm1, %xmm2
        divss   %xmm0, %xmm2
        movaps  %xmm2, %xmm0
        ret
```

Floating point alternatives

- Rational arithmetic
- Tapered floating point
- J. Gustafson's Unum number system (Gustafson 2015)
- Logarithmic number systems (Coleman et al. 2008; Ismail et al. 2011)

Take home messages

- Floating point numbers approximate mathematical real numbers
- Usually a normalised representation is used
- Special codes are used for denormalised numbers, infinities, NaNs, ± 0
- Each IEEE 754 floating point (FP) entity has a unique bit representation, and each bit representation represents an FP entity
- Some FP objects (e.g. NaNs) have mathematical properties (e.g. in comparisons) different from those of regular real numbers
- Alternatives and further developments of FP arithmetic are being researched

References

AMD (Dec. 2017). *AMD64 Architecture Programmer's Manual, Volume 1: Application Programming, revision 3.22*. AMD. URL: https://www.amd.com/system/files/TechDocs/24592.pdf.

Cody, W. J. (Mar. 1981). "Analysis of proposals for the floating-point standard". In: *Computer* 14.3, pp. 63–68. DOI: 10.1109/c-m.1981.220379.

Coleman, John N. et al. (2008). "The European Logarithmic Microprocesor". In: *IEEE Transactions on Computers* 57.4, pp. 532–546. DOI: 10.1109/tc.2007.70791.

Engelen, Robert van (2008). *Floating point operations and SIMD extensions*. URL: http://www.cs.fsu.edu/~engelen/courses/HPC-adv-2008/FP.pdf.

Goldberg, David (1991). "What every computer scientist should know about floating-point arithmetic". In: *ACM Comput. Surv.* 23, pp. 5–48. ISSN: 0360-0300. DOI: 10.1145/103162.103163. URL: http://doi.acm.org/10.1145/103162.103163.

Gustafson, John L. (Aug. 2015). *The End of Error Unum Computing by Gustafson, John L*. Vol. 1. CRC Press. ISBN: 978-14-8223-987-4.

IEEE (Oct. 1985). *IEEE standard for binary floating-point arithmetic*. IEEE. DOI: 10.1109/ieeestd.1985.82928.

— (2008). *IEEE standard for floating-point arithmetic*. DOI: 10.1109/ieeestd.2008.4610935.

— (2019). *IEEE standard for floating-point arithmetic*. IEEE. DOI: 10.1109/ieeestd.2019.8766229.

Ismail, R. Che et al. (July 2011). "ROM-less LNS". In: *2011 IEEE 20th Symposium on Computer Arithmetic*. IEEE, pp. 43–51. DOI: 10.1109/arith.2011.15.