

1. IEEE 754: Binary Floating Point Standard

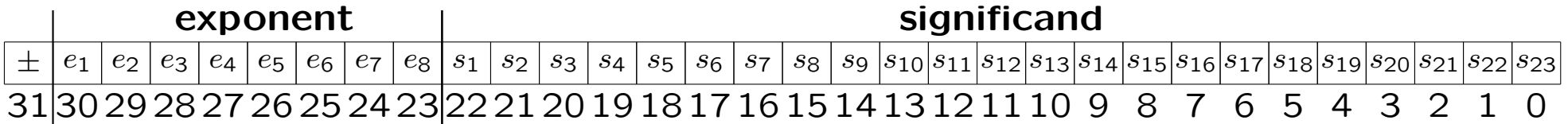
requires:

1. Consistent representation of fp
#s \forall arch
2. Correctly rounded fp ops:
 - (a) *round to nearest*
 - (b) round down
 - (c) round up
 - (d) round towards zero
3. Consistent treatment of exceptional situations

Exceptional Situations:

- $1.0/0.0 = \infty$
 - $-1.0/0.0 = -\infty$
 - $0.0/0.0 = 0.0 \times \infty = \text{NaN}$
 - Gradual underflow
 - Lose bits off of significand when exp is exceeded
 - Many arch handle in software or flush to zero
- Many archs handle some exceptions (NaN/underflow) in software
- ⇒ Major performance drop
- FPU signals exceptions in status bits

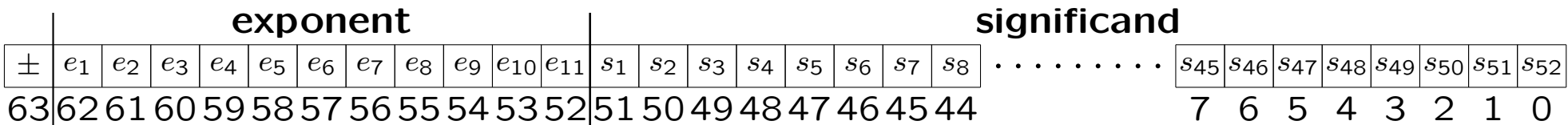
2. Single Precision IEEE Floating Point Format



(e ₁ e ₂ e ₃ ...e ₈)	Numerical value represented is
(00000000) = 0	$\pm(0.s_1s_2s_3 \dots s_{23} \times 2^{-126})$
(00000001) = 1	$\pm(1.s_1s_2s_3 \dots s_{23} \times 2^{-126})$
(00000010) = 2	$\pm(1.s_1s_2s_3 \dots s_{23} \times 2^{-125})$
⋮	⋮
(01111111) = 127	$\pm(1.s_1s_2s_3 \dots s_{23} \times 2^0)$
⋮	⋮
(11111101) = 253	$\pm(1.s_1s_2s_3 \dots s_{23} \times 2^{126})$
(11111110) = 254	$\pm(1.s_1s_2s_3 \dots s_{23} \times 2^{127})$
(11111111) = 255	$\pm\infty$ if S=0, else NaN

- 1 sign bit
- 8 exponent bits:
 - $E = 0, 255$ flag
 - $exp = E - 127$
- 24 bits of sig. prec.:
 - leading 1 not stored
 - $E = 0$, gradual underflow

3. Double Precision IEEE Floating Point Format



$(e_1 e_2 e_3 \dots e_{11})$	Numerical value represented is
(00000000000) = 0	$\pm(0.s_1 s_2 s_3 \dots s_{52} \times 2^{-1022})$
(00000000001) = 1	$\pm(1.s_1 s_2 s_3 \dots s_{52} \times 2^{-1022})$
(00000000010) = 2	$\pm(1.s_1 s_2 s_3 \dots s_{52} \times 2^{-1021})$
⋮	⋮
(01111111111) = 1023	$\pm(1.s_1 s_2 s_3 \dots s_{52} \times 2^0)$
⋮	⋮
(11111111100) = 2044	$\pm(1.s_1 s_2 s_3 \dots s_{52} \times 2^{1022})$
(11111111110) = 2046	$\pm(1.s_1 s_2 s_3 \dots s_{52} \times 2^{1023})$
(11111111111) = 2047	$\pm\infty$ if S=0, else NaN

- 1 sign bit
- 11 exponent bits:
 - $E = 0, 2047$ flag
 - $exp = E - 1023$
- 53 bits of sig. prec.:
 - leading 1 not stored
 - $E = 0$, gradual underflow

4. Intel 80 bit x87 Registers

- 1 sign bit
- 15 exp bits
- 63 bits for significand (no assumed leading bit)

5. Floating Point Arithmetic Steps

addition

1. Right shift smaller op so both have same exponent
2. Add significands
3. Renormalize and possibly round
→ Need guard bit for significand

multiplication

1. multiply significands
2. Add exponents
3. Renormalize and possibly round
→ Need guard op for significand

addition

1. $(1.11)_2 \times 2^2 + (1.1)_2 \times 2^1$
2. $= (1.11 + 0.1)_2 \times 2^2 = 10.01 \times 2^2$
3. $= 1.001 \times 2^3$

multiply

1. $(1.11)_2 \times 2^2 * (1.1)_2 \times 2^1$
2. $= (1.11 * 1.1)_2 \times 2^{2+1} = 10.101 \times 2^3$
3. $= 1.0101 \times 2^4$

7. Challenges for floating point

Not all numbers can be stored:

- * Real #s continuous and infinite
- * fp #s discrete and finite
- 1. Can be too large → *overflow*
- 2. Can be too small → *underflow* or *subnormal/denormalized*
- 3. Can have too many significant digits → *rounded*
- 4. Can have no representation in fp storage → *rounded*
 - Infinite decimal in binary storage, eg. $\frac{1}{10}$

subnormals:

For number smaller than exponentiation:

1. Choose denormalized exp flag ($E = 0$)
 2. Shift significand right:
 - Drop digits off of right end (*possible loss of precision*)
 - Insert zeros in left end of significand
- Some systems just underflow to zero immediately
- Some handle denorms in software

8. Affects of rounding

Problems with fp flow from inability to store all numbers:

- Mult may result in overflow/underflow
 - non-fp \neq only due to too many sig digits
 - Addition of two fp numbers may not be an fp number
 - $1 \times 2^{10} + 1 \times 2^{-120} = 1 \times 2^{10}$
 - Rounding non-fp \neq s least significant bits may be “junk”
 - Subtractive Cancellation:
 - Two numbers of \approx size are subtracted, leaving you with only the “junk” bits
- \Rightarrow * more accurate than \pm
 \rightarrow Strassen's $O(N^{2.807})$ matmul
less acc than gemm $O(N^3)$

- FP arithmetic no longer commutative

$$- x + y + z \neq z + y + x$$

- $x = 3.141592653589793$
- $y = 3.141592653585682$
- true diff = 4.111×10^{-12}
- single diff = 0.0
 - \rightarrow Both numbers round to same
- correctly rounded double diff = $4.110933815582030e - 12$
- Now sub $4.11e - 12$ from this, and all we have is junk

9. Determining Correctness with Reordering

$$\approx \frac{|A_g - A_t|}{|maxval| \times \epsilon \times 2 \times N_{flops}}$$

epsilon

Smallest number such that

- $1.0 + \epsilon > 1.0$
- Single (p=24): $\epsilon = 2^{-23} \approx 1.2 \times 10^{-7}$
- Double (p=53): $\epsilon = 2^{-52} \approx 2.2 \times 10^{-16}$
- Intel x87 (p=64): $\epsilon = 2^{-63} \approx 1.1 \times 10^{-19}$

determining ϵ :

```
TYPE eps;  
TYPE half=0.5;  
volatile TYPE maxval, f1=0.5;  
  
do  
{  
    eps = f1;  
    f1 *= half;  
    maxval = 1.0 + f1;  
}  
while (maxval != 1.0);
```