IEEE standard: basics I

- IEEE 754 during 80s, now standard everywhere
- Two IEEE standards:

  754: specify $\beta = 2, p = 24$ for single, $\beta = 2, p = 53$ for double

  854 ($\beta = 2$ or 10, does not specify how floating-point numbers are encoded into bits)
- Why IEEE 854 allows $\beta = 2$ or 10 but not other numbers? Some reasons:
  1. 10 is the base we use
  2. smaller β causes smaller relative error

IEEE standard: basics II

- Why smaller β causes smaller relative error? Consider

$$\beta = 16, p = 1 \text{ versus } \beta = 2, p = 4$$

- Both use 4 bits for significand, but their ϵ values are different

$$\epsilon = \frac{16}{2} 16^{-1} = 1/2, \quad \epsilon = \frac{2}{2} 2^{-4} = 1/16$$

We can see that ϵ of $\beta = 2, p = 4$ is smaller

IEEE standard: basics III

- However, IBM/370 uses $\beta = 16$. Why? Two possible reasons:
- First, assume

$$4 \text{ bytes} = 32 \text{ bits}$$

are allocated for a number. Let

$$\beta = 16, p = 6.$$

significand:

$$4 \times 6 = 24 \text{ bits},$$

exponents:

IEEE standard: basics IV

$$32 - 24 - 1 = 7 \text{ bits (1 bit for sign)}$$

range of exponent:

$$16^{-2^6} \text{ to } 16^{2^6} = 2^{2^8}$$

If instead $\beta = 2$ is used and range of exponents is the same, then

$$9 \text{ bits } (-2^8 \text{ to } 2^8 = 2^9) \text{ for exponents}$$

and

$$32 - 9 - 1 = 22 \text{ for significand}$$

Same exponents, less significand for $\beta = 2$ (24 vs. 22)

IEEE standard: basics V

- Second reason: cost of shifting.

  If $\beta = 16$, less frequently to adjust exponents when adding or subtracting two numbers

  For modern computers, this saving is not important

IEEE standard: significands and exponents I

- Single precision: $\beta = 2, p = 24$ (23 bits as normalized), exponent 8, 1 bit for sign. Thus

$$32 = 23 + 8 + 1$$

- An example: $176.625 = 1.0101100101 \times 2^7$

0	10000110	01011001010000000000000

1 of $1.\cdots$ is not stored (normalized)

- Biased exponent (described later in detail)

IEEE standard: significands and exponents II

$10000110 = 128 + 4 + 2 = 134, 134 - 127 = 7$

Note that exponent may be negative, but here we don't use a sign bit for exponents

- Use rounding even

Binary	rounded	reason
10.00011	10.00	($<$ 1/2, down)
10.00110	10.01	($>$ 1/2, up)
10.11100	11.00	(1/2, up)
10.10100	10.10	(1/2, down)

IEEE standard: significands and exponents III

This example is from http://www.cs.cmu.edu/afs/cs/academic/class/15213-s12/www/lectures/04-float-4up.pdf

- A summary

IEEE	Fortran	C	Bits	Exp.	Mantissa
Single	REAL*4	float	32	8	24
Single-extended			44	≤ 11	32
Double	REAL*8	double	64	11	53
Double-extended	REAL*10	long double	$\geq$80	≥ 15	$\geq$64

IEEE standard: significands and exponents IV

- Extended precision: we will give some brief discussion later
- In the above table, for single

$$32 = 8 + (24 - 1) + 1 = 8 + 24$$

but for single-extended

$$44 \neq 11 + 32$$

- Why $44 \neq 11 + 32$?

IEEE standard: significands and exponents V

Hardware implementation of extended precision normal don't use a hidden bit

(Remember we normalized each number so 1 is not stored)

- Minimal normalized positive number

$$1 \times 2^{-126} \approx 1.17 \times 10^{-38}$$

$e_{\min} = -126$

IEEE standard: significands and exponents VI

- 8 bits for exponent: 0 to 255. IEEE uses a biased approach for exponents

$$(0 \text{ to } 255) - 127 = -127 \text{ to } 128$$

- Then, -127 for 0 and denormalized numbers (discussed later), -126 to 127 for exponents, 128 for special quantity
- Thus

$$e_{\min} = -126 \text{ and } e_{\max} = 127$$

IEEE standard: significands and exponents VII

- Why not

$$e_{\min} = -127 \text{ and } e_{\max} = 126$$

reasons: $1/2^{e_{\min}}$ not overflow, $1/2^{e_{\max}}$ underflow, but less serious