Number System

Integer Representation

- Size of Data and Value:

SIZE	BINARY	DEC	HEXA
8	0000 0000 1111 1111	0 - 255	00 - FF
12	0000 0000 0000 1111 1111 1111	0 - 4095	000 - FFF
16	0000 0000 0000 0000 1111 1111 1111 1111	0 – $(2^{16} – 1)$	0000- FFFF
20	0000 0000 0000 0000 0000 1111 1111 1111 1111 1111	0 – $(2^{20} -1)$	00000 - FFFFF
32	0000 …………………………..0000 1111 …………………………..1111	0 – $(2^{32} -1)$	00000000 – FFFFFFFF

Integer Representation

- Only have 0 & 1 to represent everything
- numbers stored in binary
  - e.g. 41=00101001
- No minus sign (negative nos.)
- No period (for real nos.)
- Negative Numbers
  - Sign-Magnitude
  - Two's compliment

Sign-Magnitude

- Left most bit is sign bit (MSB)
  - 0 means positive
  - 1 means negative
- +18 = 00010010
- -18 = 10010010
- Problems
  - Need to consider both sign and magnitude in arithmetic
  - Two representations of zero (+0 and -0)

Two's Compliment

- +3 = 00000011
- +2 = 00000010
- +1 = 00000001
- +0 = 00000000
- -1 = 11111111
- -2 = 11111110
- -3 = 11111101

Benefits

- One representation of zero
- Arithmetic works easily
- Negating is fairly easy
  - 3 = 00000011
  - Boolean complement gives 11111100
  - Add 1 to LSB 11111101

Negation Special Case 1

- 0 = 00000000
- Bitwise not 11111111
- Add 1 to LSB +1
- Result 1 00000000
- Overflow is ignored, so:
- - 0 = 0 √

Negation Special Case 2

- -128 = 10000000
- bitwise not 01111111
- Add 1 to LSB +1
- Result 10000000
- So:
- -(-128) = -128 X
- Monitor MSB (sign bit)
- It should change during negation

Range of Numbers

- 8 bit 2s compliment
  - +127 = 01111111 = 2^7 -1
  - -128 = 10000000 = -2^7
- 16 bit 2s compliment
  - +32767 = 011111111 11111111 = 2^{15} - 1
  - -32768 = 100000000 00000000 = -2^{15}

Real Numbers

- Numbers with fractions
- Could be done in pure binary
  - 1001.1010 = $2^4 + 2^0 + 2^{-1} + 2^{-3}$ =9.625
- Where is the binary point?
- Fixed?
  - Very limited
- Moving?
  - How do you show where it is?

Floating Point

(a) Format

- +/- .significand x $2^{exponent}$
- Misnomer
- Point is actually fixed between sign bit and body of mantissa
- Exponent indicates place value (point position)

Signs for Floating Point

- Mantissa is stored in 2s compliment
- Exponent is in excess or biased notation
  - e.g. Excess (bias) 128 means
  - 8 bit exponent field
  - Pure value range 0-255
  - Subtract 128 to get correct value
  - Range -128 to +127

Normalization

- FP numbers are usually normalized
- i.e. exponent is adjusted so that leading bit (MSB) of mantissa is 1
- Since it is always 1 there is no need to store it
- (c.f. Scientific notation where numbers are normalized to give a single digit before the decimal point
- e.g. 3.123×10^3)

Floating Point Examples

(a) Format

$$
\begin{aligned}
1.1010001 \times 2^{10100} &= 0\ 10010011\ 10100010000000000000000 = 1.638125 \times 2^{20} \\
-1.1010001 \times 2^{10100} &= 1\ 10010011\ 10100010000000000000000 = -1.638125 \times 2^{20} \\
1.1010001 \times 2^{-10100} &= 0\ 01101011\ 10100010000000000000000 = 1.638125 \times 2^{-20} \\
-1.1010001 \times 2^{-10100} &= 1\ 01101011\ 10100010000000000000000 = -1.638125 \times 2^{-20}
\end{aligned}
$$

(b) Examples

FP Ranges

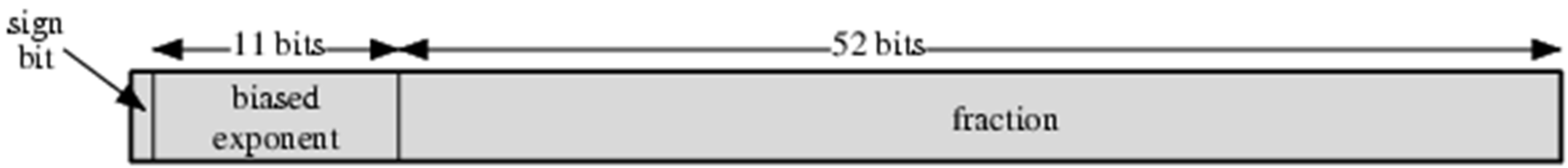
- For a 32 bit number
  - 8 bit exponent
  - +/- $2^{127} \approx 1.5 \times 10^{77}$
- Accuracy
  - The effect of changing lsb of mantissa
  - 23 bit mantissa $2^{-23} \approx 1.2 \times 10^{-7}$
  - About 6 decimal places

IEEE 754

- Standard for floating point storage
- 32 and 64 bit standards
- 8 and 11 bit exponent respectively

IEEE 754 Formats

(a) Single format



(b) Double format

Other Codes

- Excess Code (Excess-128)
- GREY Code
- BCD (Binary Coded Decimal)

Character Representation

- ASCII (American Standard Code for Information Interchange)
- EBCDIC (Extended Binary Coded Decimal Interchange Code)
- UNICODE