Computer Arithmetic

Floating Point

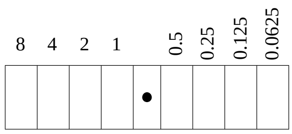
Floating Point

- We need a way to represent
  - numbers with fractions, e.g., 3.1416
  - very small numbers, e.g., .000000001
  - very large numbers, e.g., 3.15576×10^9
- Representation:
  - sign, exponent, significand: $(-1)^{sign} \times$ significand $\times\ 2^{exponent}$
  - more bits for significand gives more accuracy
  - more bits for exponent increases range

Definitions

- A normalised number has no leading zeros
  - e.g. 0.000000001 is 1.0×10^{-9}

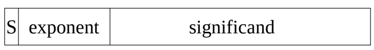
Binary Point



e.g. $3.625_{ten} = 0011.1010_{two}$

IEEE 754 Floating Point Standard

- Single precision: 8 bits exponent, 23 bits significand
  - 32 bits, C **`float`**
  - range: 2.0×10^{-38} to 2.0×10^{38}



- Double precision: 11 bits exponent, 52 bits significand
  - 64 bits: C **`double`**
  - range: 2.0×10^{-308} to 2.0×10^{308}

Pentium / PPC

- Internally, these architectures use an 80 bit floating point representation
  - defined by IEEE 754 as double-extended
  - 15 exponent bits
  - 64 significand bits
- CPU converts to double / float when reqd.
- 80-bit format poorly supported by programming languages

Sorting

- In an ideal world, the sort operation could use existing (integer) hardware.
- Board exercise: In what order do we check the fields of a floating point number when sorting?
  - -1.256×10^{-2}
  - 0.234×10^{-3}
  - 0.187×10^{1}

IEEE 754 Floating Point Standard

- Exponent is “biased” to make sorting easier
  - all 0s is smallest exponent; all 1s is largest
  - bias of 127 for single precision and 1023 for double precision
  - summary: $(-1)^{\text{sign}} \times (1+\text{significand}) \times 2^{\text{exponent} - \text{bias}}$
- Leading “1” bit of significand is implicit
- Board Exercise: encode -0.75 in single precision

IEEE 754 Floating Point Standard

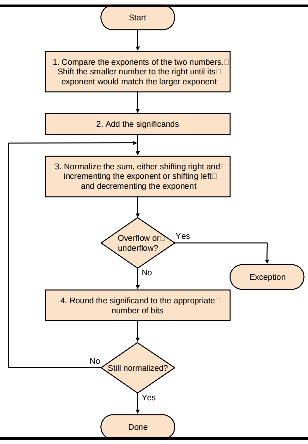
- Example:
  - decimal point: $-.75 = -3/4 = -3/2^2$
  - binary point: $-0.11 \times 2^0 = -1.1 \times 2^{-1}$
  - floating point exponent = 126 = 01111110
  - IEEE single precision: `10111111010000000000000000000000`

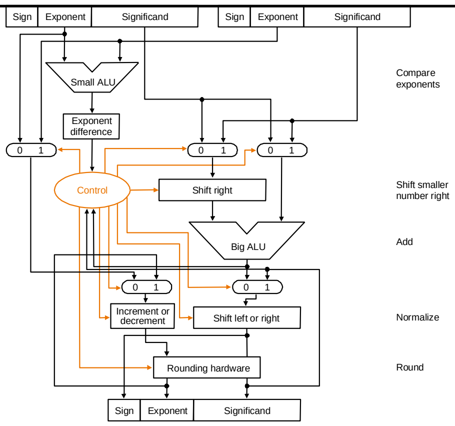
Floating Point Complexities

- Mathematical operations are somewhat more complicated
- In addition to overflow we can have “underflow”
- Accuracy can be a big problem
  - IEEE 754 keeps two extra bits, guard and round
  - four rounding modes
  - positive divided by zero yields “infinity”
  - zero divided by zero yields “not a number”
  - other complexities
- Implementing the standard can be tricky
- Not using the standard can be even worse
  - see text for description of 80x86 and Pentium bug!

Floating Point Addition: Board Exercise

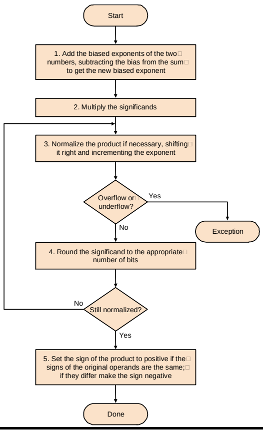
- $9.999 \times 10^1 + 1.610 \times 10^{-1}$
- Assume that we can only store
  - 4 decimal digits of significand
  - 2 decimal digits of exponent





Floating point Multiplication

- Multiply mantissas and add exponents
- Steps:
  - Add exponents
  - Multiply mantissas
  - Normalise result
  - Round results
  - Fix sign of product



Floating Point Accuracy

- Floating point numbers are normally approximations for the numbers they represent
  - infinite numbers between 0 and 1, but only 53 bits in double precision to represent them
- IEEE-754 keeps two extra bits on the right during intermediate calculations called guard and round respectively
- Example: add $2.56_{ten} * 10^0$ to $2.34_{ten} * 10^2$ assuming three significant digits
  - with guard and round digits
  - without guard and round digits

Sticky Bit

- IEEE 754 allows for a third bit in addition to guard and round.
- in school we always rounded 0.5 up
  - error accumulates with each round
  - 0.35 + 0.4 → 0.4 + 0.4 → 0.8
  - solution: use sticky bit, round down ~half the time
  - 0.35 + 0.4 → 0.7

Sticky bit example

- $5.01 \times 10^{-1} + 2.34 \times 10^2$
  - Three significant digits

Floating Point Accuracy

- Four Rounding modes :
  - round to nearest (default)
  - round towards plus infinity
  - round towards minus infinity
  - round towards 0

Examples

	G	R	S	Plus infinity	Minus infinity	Truncate	Nearest even
+1001	1	1	0	+1010	+1001	+1001	+1010
+1001	0	1	1	+1010	+1001	+1001	+1001
+1001	1	0	0	+1010	+1001	+1001	+1010
+1000	1	0	0	+1001	+1000	+1000	+1000
+1001	0	0	0	+1001	+1001	+1001	+1001
-1001	1	1	0	-1001	-1010	-1001	-1010
-1001	0	1	1	-1001	-1010	-1001	-1001
-1001	1	0	0	-1001	-1010	-1001	-1010
-1000	1	0	0	-1000	-1001	-1000	-1000
-1001	0	0	0	-1001	-1001	-1001	-1001

Special Symbols

- Special symbols used by IEEE-754
  - $+\infty$ or $-\infty$ (largest exponent with 0 mantissa) (result of divide by zero)
  - NaN (Not a number) (largest exponent with non 0 mantissa) (0/0 or $\infty - \infty$)
  - Unnormalised Numbers (no explicit 1 in MSB) (0 exponent non-zero mantissa)
  - Zero (zero Mantissa and zero exponent)

Summary

Single Precision		**Double Precision**		**represents**
Exponent	**Significand**	**Exponent**	**Significand**	
0	0	0	0	0
0	Non-zero	0	Non-zero	+/- denormalised number
1-254	Anything	1-2046	Anything	+/- floating point number
255	0	2047	0	+/- infinity
255	Non-zero	2047	Non-zero	NaN (not a number)