SWINBURNE UNIVERSITY OF TECHNOLOGY

COS10004 Computer Systems

Lecture 5.2 Representing real numbers: Fixed and Floating-point

Dr Chris McCarthy

Number systems: fixed-point numbers

> Integers are much easier to work with than real numbers!

> What's a real number ?

– Numbers that consist of an integer part and fractional part:

– E.g., 3.1, 6.443442, 100.0 etc

> Major representational trade-offs:

- Space efficiency (and precision)
- Computational efficiency

Not all real numbers can be represented!

- Real numbers can be infinitely precise
  - Consider numbers like Pi (3.14159653589793238462 64…..)
  - Or 1/3 = 0.33333333333333333333333….
- Computers only have finite memory
  - The number of bits available will directly impact the value range of values, and the precision
  - How we choose to represent real numbers will greatly impact the bits we have.

Firstly – recall how decimal works

5123.124

Weight	10^3	10^2	10^1	10^0	Decimal Point	10^{-1}	10^{-2}	10^{-3}
Digit	5	1	2	3	.	1	2	4

NO THINK HOW BINARY WORKS

- To represent numbers, the binary system uses base 2. Therefore, the binary system is also known as *base-2 system and represented by two symbols*.
- These symbols are 0 and 1

1011.101

Weight	2^3	2^2	2^1	2^0	Binary Point	2^{-1}	2^{-2}	2^{-3}
Digit	1	0	1	1	.	1	0	1

Fixed-Point Representations

- One approach for representing real number apprxoimations is to dedicate some fixed number of bits for the integer and fractional parts.
- We call this fixed point representation because the binary point (i.e. the split between the two parts) is at a fixed location in the word.

Fixed binary point

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
16	8	4	2	1	0.5	0.25	0.125

Number systems: fixed-point numbers

Fixed binary point

2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
16	8	4	2	1	0.5	0.25	0.125
0	1	0	1	0	1	0	1

- With the above numbering system 01010 101 = 10.625
- Can use all standard arithmetic and also represent in 2's complement form.
- Just have to remain consistent as to which bits are fractional part, i.e. must fix the binary-point.

Fixed Point Example:

Using the fixed<16,7> binary point representation show below, represent the number 25.6640625 (2 marks):

8	7	6	5	4	3	2	1	0 .	-1	-2	-3	-4	-5	-6	-7

1. Don’t panic. Start by converting 25 to binary:
   25 = 16+8+1 = 000011001.
2. Then the “decimal” point .
3. convert 6640625 to binary (starting with 0.5, 0.25, 0.125...);
   0.5+0.125+0.03125+0.781259 converts to .1010101;
4. Concatenate the two numbers: 000011001.1010101

Binary-coded decimal (BCD)

> Each digit of a decimal number represented by a nibble in the data word.

> For example: 7926.34 = $\underbrace{0111}_{7}\ \underbrace{1001}_{9}\ \underbrace{0010}_{2}\ \underbrace{0110}_{6}\ .\ \underbrace{0011}_{3}\ \underbrace{0100}_{4}$

> Conceptually simple means to convert and represent large fractional decimal numbers in binary form on basic CPUs.

> Can make arithmetic algorithms simple.

> Used in basic calculators.

> Overhead is inefficient storage *c.f.* fixed point. (37% wastage)

Binary-coded Decimal (BCD) Example:

Using BCD and the fixed point representation show below, represent the number 29.95 (2 marks):

7	6	5	4	3	2	1	0 .	-1	-2	-3	-4	-5	-6	-7	-8

1. It's a 4-digit number so we will need 4 nib
2. 2 converts to 0010;
3. 9 converts to 1001;
4. Then the "decimal" point .
5. Then 9 (1001);
6. then 5 (0101)

Fixed decimal point, so (for instance) 265.5 would overflow

7	6	5	4	3	2	1	0	-1	-2	-3	-4	-5	-6	-7	-8
0	0	1	0	1	0	0	1	1	0	0	1	0	1	0	1

Floating Point Representation

- Issue with Fixed-point:
  - Precision needs vary with numbers. Fixed point can be wasteful
- Floating point representations:
  - Represent real numbers with variable bit allocations for the fractional component
  - Supports a trade-off between value range and precision
  - BUT – much more complex !
- Follows a similar idea to scientific notation:

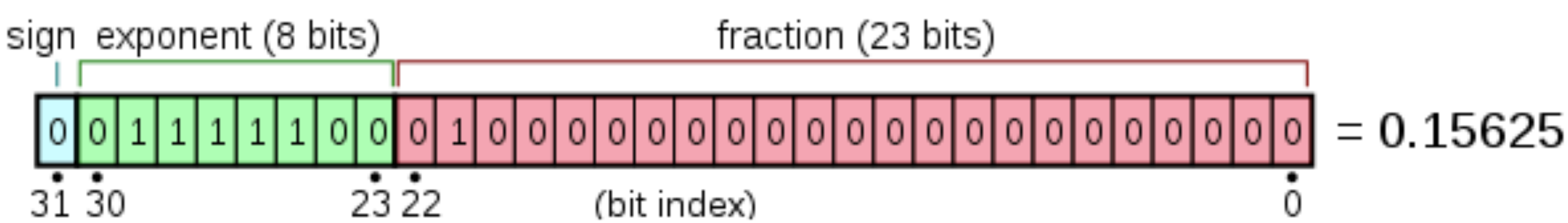
$$1.2345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}^{\overbrace{-4}^{\text{exponent}}} .$$

Floating point

- Floats, doubles.
  - Separate into mantissa and exponent.
- Mantissa:
  - Add / subtract (2's compliment) with adders
  - Multiply / divide with shift registers (+ counter and adder)
- Exponent:
  - Multiply/divide - add / subtract (2's compliment) with adders
  - Add / subtract – If the exponents are the same, just work on the mantissas

IEEE 754 Standard

- IEEE 754: a standardised specification for allocating bits
- Below is for 32 bit floating point, made up of three parts:
  - Sign bit (1 bit)
  - Exponent (8 bits)
  - Fraction/Mantissa (23 bits)



Floating point representation

- Floats, doubles.
  - Separate into mantissa and exponent.
- Mantissa/significand:
  - Add / subtract (2's compliment) with adders
  - Multiply / divide with shift registers (+ counter and adder)
- Exponent:
  - Multiply/divide - add / subtract (2's compliment) with adders
  - Add / subtract – If the exponents are the same, just work on the mantissas

Example:

Using the IEEE 754 floating point standard (shown below), represent the number -273.5 as a 32-bit single precision floating

	31	30 – 23	22 – 0
single precision	S	8-bit exp	23-bit frac

bit 23 = 2^9=256

binary point between 2^0 and $2^{0.5}$

1. This is –ve, so bit 31 will be 1
2. Convert 273.5 to binary: 256+16+1+0.5 = 100010001.100...(trailing 0s)
3. Shift binary point left to right of bit 23 (count the shifts) (=8) (1.00010001100..)
4. Pad with 0s (LSB) and remove bit 23 (always 1) (.00010001100000000000000)
5. Add 2^7-1 to the number of shifts (in binary) the mantissa

```
     8  00001000
+127  01111111
=135  10000111  <-the exponent
```

6. Concatenate sign bit, exponent and mantissa:

Example:

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	1	0	0	0	0	1	1	1	0	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Using the IEEE 754 floating point standard (shown below), represent the number -273.5 as a 32-bit single precision floating

	31	30 … 23	22 … 0
single precision	S	8-bit exp	23-bit frac

bit 23 = 2^9=256

binary point between 2^0 and $2^{0.5}$

1. This is –ve, so bit 31 will be 1
2. Convert 273.5 to binary: 256+16+1+0.5 = 100010001.100...(trailing 0s)
3. Shift binary point left to right of bit 23 (count the shifts) (=8) (1.00010001100..)
4. Pad with 0s (LSB) and remove bit 23 (always 1) (.00010001100000000000000)
5. Add 2^7-1 to the number of shifts (in binary) the mantissa

   8 00001000

   +127 <u>01111111</u>

   =135 10000111 <-the exponent

6. Concatenate sign bit, exponent and mantissa:

SWINBURNE UNIVERSITY OF TECHNOLOGY

Number systems: floating point representation

- Most significant bit of mantissa not included as it is always = 1
- Exponent is in 2’s complement form (positive and negative) and added to +127 (b’0111 1111’)
- The term *significand* has tended to replace mantissa
- There are special patterns to represent:
  +/- infinity; +/- 0 (zero), NaN (not a number)

Floating-point Operations – Where ?

- Floating-point numbers typically handled using dedicated circuits to perform arithmetic operations:
  - Sometimes referred to as the math co-processor or Floating Point Unit
  - One or more FPUs typically resides in the CPU
- Some simpler computers may not offer floating point hardware:
  - May still be emulated using ALU and supporting floating-point library

Summary

- Real numbers pose a specific challenge for representing in binary
- Fixed-point representations offer simplicity, but can be wasteful
- Floating point representations standard in modern computers
  - IEEE 754 standard
  - Allows trade-offs of range and precision
  - Requires dedicated FP arithmetic hardware:
    - FPU – floating point unit