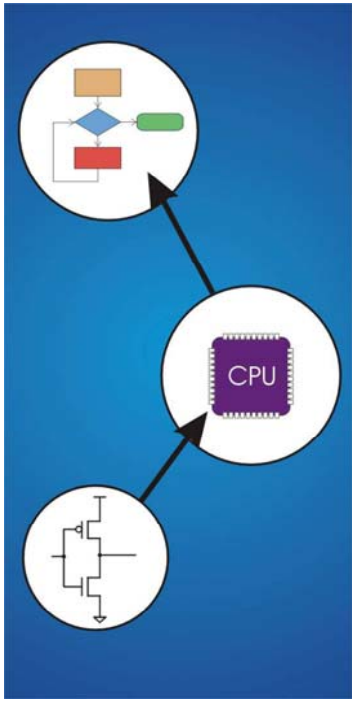

COMPSCI 210
Part III
Floating Point Numbers

Agenda & Reading

Agenda:

- Introduction
- Normalization
- IEEE 754 Floating Point Representation
  - Single Precision (32-bit)
  - Double Precision (64-bit)
- Examples
- Special Numbers
- Conversion

Recommended Reading:

- IEEE Floating Point Numbers
  - http://en.wikipedia.org/wiki/IEEE_floating-point_standard

Animation

- Float.htm

Exercise

- Worksheet 02

Introduction

Given the following numbers:

- $37.25_{10} = 100101.01_2$
- $0.3125_{10} = 0.0101_2$
- $+300000000\ ms^{-1}$
- $+299,792,458\ ms^{-1}$

How can we store the above number in 4 bytes?

People usually represent very large and small numbers in "scientific notation", as a fixed point number times a power of 10.

- Scientific Notation:
  - $3 * 10^8$
  - $2.9979 * 10^8$
  - $3.125 * 10^{-1}$
  - $3.725 * 10^1$
- Floating point numbers are represented in the computer in a similar manner, but using base 2 rather than base 10.
  - 1.0010101×2^5
  - 1.11101×2^2
  - 1.01×2^{-2}

Normalization

Given the following number

- $12.34 = 0.1234 \times 10^2 = 1.234 \times 10^1 = 123.4 \times 10^{-1}$

Scientific numbers are always written with one digit before the point, giving a "normalized" representation.

- After Normalization = 1.234×10^1

Normalization in Base 2

- One digit to the left of the binary point. It must be 1.
- Examples:
  - $100101.01_2 = 1.0010101 \times 2^5$ — Radix point move to left by 5 places
  - $111.101_2 = 1.11101 \times 2^2$ — Radix point move to left by 2 places
  - $0.0101_2 = 1.01 \times 2^{-2}$ — Radix point move to right by 2 places

After normalization, the numbers now have a standard format

Floating Point

Consists of three parts

- Sign bit
- Exponent
- Mantissa/Signficand
  - Note: Normalized binary numbers always start with a 1 (the leftmost bit of the signficand value). We don't need to store the 1. IEEE 754 representation uses this idea. All numbers must be normalized.

IEEE 754 Floating Point Representation

- Single Precision (32 bits) `0 10000100 00101010000000000000000`
  - a 1-bit sign, an 8-bit exponent with a bias of 127, and a 23-bit significand
  - (–1)*sign* *(1.0+*significand*) *2 *(exponent–127)*
- Double Precision (64 bits)
  - a 1-bit sign, an 11-bit exponent with a bias of 1023, and a 52-bit significand
  - (-1)*sign* * (1.0 + *significand*) * 2 (*exponent–1023*)

```
0  10000000100
0010101000000000000000000000000000000000000000000000
```


IEEE 754 Floating Point Representation

Single Precision (float)

```
0  10000100 00101010000000000000000
Sign = 0
Exponent = 10000100 (132-127)
Mantissa = 00101010..0 (1.00101010)
```

- Sign Bit: (1 bit)
  - 0 +ve, 1 –ve
- Exponent Bits: (8 bits)
  - Excess/Biased representation
    - To store positive and negative exponents
      - Subtracting a bias 127 (2^7 -1) from the value
      - Actual exponent is: E = e - 127
    - Example:
      - 00000001 is the representation of -126;
      - 10000000 is the representation of +1
      - An exponent of 5 is therefore stored as 127+5 = (10000100)
      - An exponent of -2 is stored as = 127-2 = 125 = (1111101)
- Mantissa Bits: (23 bits)
  - Mantissa is the set of 0's and 1's to the right of the radix point of the <u>normalized</u> binary number
    - 1.<u>0010101</u> x 2^5 , Mantissa = 00101010…0
    - Numbers stored in normalized form; i.e. 1.xxx…, therefore it is assumed (and therefore "1" is not stored) in the format

IEEE 754 Floating Point Representation

Double Precision (double)

- Sign Bit: (1 bit)
  - 0 +ve, 1 –ve
- Exponent Bits: (11 bits)
  - Excess/Biased representation
    - Subtracting a bias 1023 from the value
      - Note: The exponent is 11 bits, so the bias = 2^{10} -1
    - Example:
      - 00000000001 is the representation of -1022
      - 10000000000 is the representation of +1
- Mantissa Bits: (52 bits)
  - Mantissa is the set of 0's and 1's to the right of the radix point of the <u>normalized</u> binary number
    - Numbers stored in normalized form; i.e. 1.xxx…, therefore it is assumed (and therefore "1" is not stored) in the format

```
0  10000000100 0010101000000000000000000000000000000000000000000000000
Sign = 0
Exponent = 10000000100 (1028-1023)
Mantissa = 00101010..0 (1.00101010)
```


Examples

32-bit IEEE 754 Floating point representation

- 0 01111101 101 0000 0000 0000 0000 0000
  - Sign = 0
  - Exponent = 01111101 = 125 =>125 – 127 = -2
  - Mantissa = 1010…0 = 0.5 + 0.125 = 0.625
  - Answer = $(-1)^0$ * (1.0 + 0.625) * $2^{(-2)}$
  - = (1.625) * $2^{(-2)}$ = 0.40625

64-bit IEEE 754 Floating point representation

- 0 01111111101 101 0000 0000 0000 0000 0000 … 0
  - Sign = 0
  - Exponent = 01111111101 = 1021 =>1021 – 1023 = -2
  - Mantissa = 1010…0 = 0.5 + 0.125 = 0.625
  - Answer = $(-1)^0$ * (1.0 + 0.625) * $2^{(-2)}$
  - = (1.625) * $2^{(-2)}$ = 0.40625

Special Numbers

Special Exponents (Single)

- 00000000
  - Case 1: Represent number in Denormalized format!
    - Exponent is all zeros, the floating-point number is Denormalized
    - = (0.0+*significand*) *2 $^{-126}$
  - Case 2: Represent ZERO with all zeros in exponent and mantissa bits
    - +0, -0
- 11111111
  - NaN
    - An exponent of all ones with any other mantissa is interpreted to mean "not a number". Example: 0 11111111 00000000000000000000001
  - Infinity:
    - An exponent of all ones with a mantissa whose bits are all zero indicates an infinity. The sign of the infinity is indicated by the sign bit.
    - Example: 0 11111111 00000000000000000000000

Table

Range Name		
-NaN	1 11…11 00…01	FF800001
-Infinity	1 11…11 00…00	FF800000
Negative Normalized	1 11..10 11..11	FF7FFFFF
	…	
Negative Normalized	1 00…01 00..00	80800000
Negative Denormalized	1 00..00 11…11	807FFFF
	…	
Negative Denormalized	1 00..00 00…01	80000001
-0	1 00..00 00..00	80000000
+0	0 00..00 00..00	00000000
Positive Denormalized	0 00..00 00…01	00000001
	…	
Positive Denormalized	0 00..00 11…11	007FFFFF
Positive Normalized	0 00..01 00…00	00800000
	…	
Positive Normalized	0 11..10 11…11	7F7FFFFF
+Infinity	0 11..11 00..00	7F800000
+NaN	0 11..11 00..01	7F800001

Conversion:

```
Example:
IEEE754.html (Enter a float value, select
the type (Single/double)
```

Example 1: Decimal -> IEEE Floating Point

- -1.25_{10} ,
  - Binary = -1.01, Normalization => -1.01 = -1.01 * 2^0
  - Answer:
    - Exponent = 127 = 01111111
    - Sign = 1
    - Mantissa = 010…0
  - Answer = 1 01111111 010…0 = BFA00000

Shockwave Movie: float.htm

Example 2: IEEE Floating Point -> Decimal

- 40900000
  - = 0100 0000 1001 0000 … 0000
  - Answer:
    - Sign = 0
    - Exponent = 100 0000 1 = 129 = 129-127
    - Mantissa = 001 0000 … 0000 = 0.125
  - Answer = $(-1)^0$ * (1.0 + 0.125) * 2 $^{(2)}$ = 4.5

Java/C Examples

Print Hexadecimal Strings (Java)

```
Integer.toHexString(Float.floatToIntBits (f))
Long.toHexString(Double.doubleToLongBits (d))
```

Print decimal value (Java)

```
Float.intBitsToFloat(0xbfa00000)
Double.longBitsToDouble(0x4012000000000000L)
```

Print Hexadecimal Strings (C)

```
float num = -1.25;
int temp;
temp = *((int*) &num); //convert the float to the Hex bit patterns
printf("\nThe number is %f = %x", num, temp);
```

Range & Problems

Range:

- Magnitude of numbers that can be represented is in the range:
- Single Precision (Normalized)
  - $2^{-126} * (1.0)$ to $2^{127} * (2 - 2^{-23})$ Why? (see next page)
    - $2^{-126} * (1.0)$: Exponent: 00000001 Mantissa: 000…000
    - $2^{127} * (2 - 2^{-23})$: Exponent: 11111110, Mantissa: 1…111 = 3.40×10^{38}
    - 1.18×10^{-38} to 3.40×10^{38} (approximately)
- Double Precision
  - $2^{-1022}*(1.0)$ to $2^{1023}*(2 - 2^{-52})$ ($2.23*10^{-308}$ to $1.8*10^{308}$)

Problems

- Overflow occurs when the exponent is larger than the allocated space
- Underflow occurs when a negative exponent is too large in absolute value to fit within the bits allocated to store it
- Truncation occurs when there are not enough digits in the mantissa to represent the number:
  - Examples: 1/3, 1/10 etc

```
1/10 = 3DCCCCCC = (1.1001100110011...) x 2^-4 = 0.09999...
1/3 = 3EAAAAAA = (1.010101010101...) x 2^-2 = 0.3333...
```


Extra

0 00..01 00…00

- $= (1.0 + m) * 2^e$
- $= (1.0 + 0) * 2^{1-127}$
- $= 2^{-126}$

0 11..10 11…11

- $= (1.0 + m) * 2^e$
- $= (1.0 + 2^{-1} + 2^{-2} + 2^{-3} + \ldots 2^{-23}) * 2^{254-127}$
- $= (1.0 + 2^{-23} (2^{22} + \ldots + 2^2 + 2 + 1)) * 2^{127}$
- $= (1.0 + 2^{-23} (1 + 2 + 2^2 + \ldots + 2^{22})) * 2^{127}$
- $= (1.0 + 2^{-23} (2^{23} - 1)/(2-1)) \times 2^{127}$
- $= (1.0 + 2^{-23} (2^{23} - 1)) * 2^{127}$
- $= (1.0 + 2^0 - 2^{-23}) * 2^{127}$
- $= (2 - 2^{-23}) * 2^{127}$

Geometric series:
$1 + x + x^2 + \ldots + x^n$
$= x^{n+1} - 1 / (x-1)$