Introduction to Computers and Programming

Prof. I. K. Lundqvist

Reading: B pp. 47-71

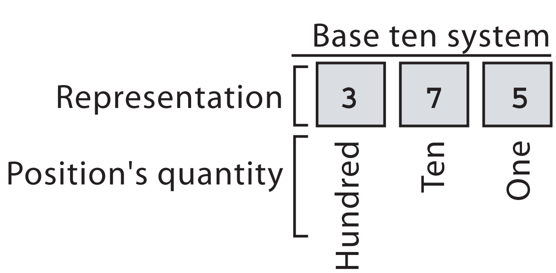
Lecture 5
Sept 12 2003

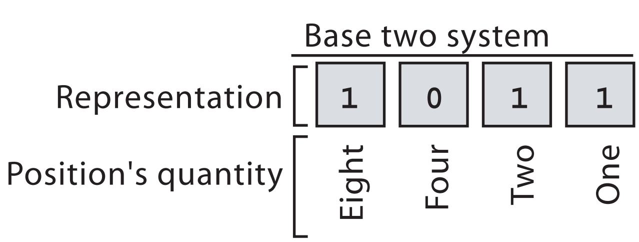
Numeric Values

- Storing the value of 25_{10} using ASCII:

  00110010 00110101

- Binary notation: $00000000\ 000011001_2$

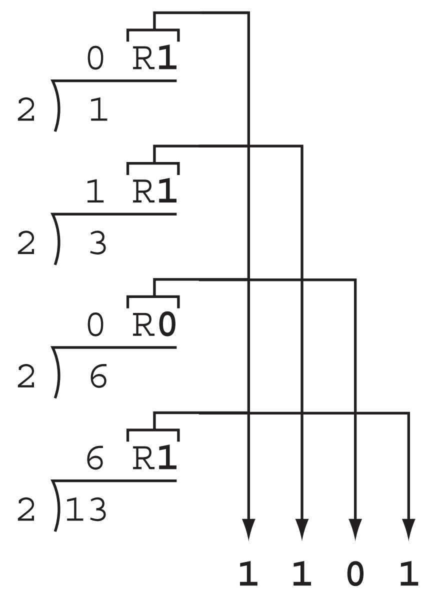




1. Binary place	2^5	2^4	2^3	2^2	2^1	2^0	
2. Position's quantity	32	16	8	4	2	1	
3. Example binary pattern	1	0	1	1	0	1	
4. Total (2. x 3.)	32 +	0 +	8 +	4 +	0 +	1	= 45

Finding Binary Representation of Large Values

1. Divide the value by 2 and record the remainder
2. As long as the quotient obtained is not 0, continue to divide the newest quotient by 2 and record the remainder
3. Now that a quotient of 0 has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded



The Binary System

- Decimal: Position represents a power of 10
  - $5382_{10} = 5x10^3 + 3x10^2 + 8x10^1 + 2x10^0$
- Binary: Position represents a power of 2
  - $1011_2 = 1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$
    $= 8 + 0 + 2 + 1 = 11_{10}$
- Binary addition

$$\begin{array}{r}0\\+0\\\hline 0\end{array}\qquad\begin{array}{r}1\\+0\\\hline 1\end{array}\qquad\begin{array}{r}0\\+1\\\hline 1\end{array}\qquad\begin{array}{r}1\\+1\\\hline 10\end{array}$$

- A byte

bit number	7	6	5	4	3	2	1	0
	0	1	0	1	0	0	1	1
bit value	128	64	32	16	8	4	2	1

Representing Negative Numbers

- Using one byte, any suggestions for representing negative numbers?
  - E.g., $00010011_2 = 19_{10}$. How to represent -19_{10} in binary?
- Reserve 1 bit (#7) for sign, 7 bits for number (sign magnitude)
  - **0**$0000001_2 = 1_{10}$ **1**$0000001_2 = -1_{10}$
  - **0**$0010011_2 = 19_{10}$ **1**$0010011_2 = -19_{10}$

Representing Negative Numbers

- One's complement – invert each bit
  - $00010011_2 = 19_{10}$ $11101100_2 = -19_{10}$
  - 0_{10}: 00000000_2 and 11111111_2
- Two's complement – invert each bit and add 1
  - $00010011_2 = 19_{10}$ $11101101_2 = -19_{10}$
  - Try to negate 0:
    - $0_{10} = 00000000_2$
    - invert: $00000000_2 \rightarrow 11111111_2$
    - add 1: $11111111_2 + 00000001_2 = 00000000_2$

Two's Complement Notation Systems

Bit Pattern	Value Represented
011	3
010	2
001	1
000	0
111	-1
110	-2
101	-3
100	-4

Bit Pattern	Value Represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Addition

Problem in base ten	Problem in two's complement	Answer in base ten
3 + 2	0011 + 0010 = 0101	5
-3 + -2	1101 + 1110 = 1011	-5
7 + -5	0111 + 1011 = 0010	2

- Example: we are using 8 bit two's complement and have the problem 97-81.

97	0	1	1	0	0	0	0	1
-81	+1	0	1	0	1	1	1	1
16	0	0	0	1	0	0	0	0

Summary: One's/Two's Complement

- Note that in *sign magnitude* and in *one's complement* there are two possible representations of 0, and we can represent every integer n with $-(2^k - 1) \leq n \leq 2^k - 1$ with a k-bit field.
- With *two's complement* there is only one representation of 0, and we can represent the integers n with $-2^k \leq n \leq 2^k - 1$.
- The most significant advantage of two's complement is the fact that subtraction can be performed with addition circuitry.

The Problem of Overflow

- Overflow: when a value to be represented falls outside the range of values that can be represented.
- An overflow is indicated if the addition of two positive values results in the pattern for a negative value or vice versa.
- Remember: small values can accumulate to produce large numbers.

Fractions in Binary

- Each position is assigned a quantity of twice the size of the one to its right.
- 101.101 = ?

1. Binary place	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}	
2. Position's quantity	4	2	1		1/2	1/4	1/8	
3. Example binary pattern	1	0	1	.	1	0	1	
4. Total (2 x 3)	4 +	0 +	1 +		1/2 +	0 +	1/8	= 5 5/8

```
  10.001
+ 100.000
  111.001
```

Floating Point representation

- To represent a floating point number the number is divided into two parts: the integer and the fraction
  - 3.1415 has integer 3 and fraction 0.1415
- Converting FP to binary:
  - Convert integer part to binary
  - Convert fraction part to binary
  - Put a decimal point between the two numbers

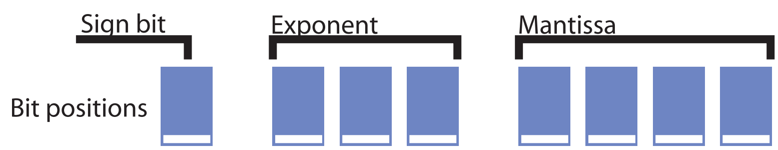
Floating Point representation

- Assume 12 bits to represent the integer part and 4 bits to represent the fraction:

  71.3425 = +1000111.0101

Normalization

- To represent +1000111.0101
  - Store sign, all bits, and the position of decimal point
- Instead we use Normalization
  - Move the decimal point so that there is only one 1 to the left of the decimal point.
  - To indicate the original value of the number, multiply by 2^e where e is the number of bits that the decimal point moved, positive for left movement, negative for right movement
  - Store:
    - The sign
    - The exponent
    - The mantissa



Excess (or bias) Notation

- Alternative to two's complement used to store the exponent for floating point numbers.

Bit Pattern	Value Represented
111	3
110	2
101	1
100	0
011	-1
010	-2
001	-3
000	-4

With 3 bits we have excess 4 ($=2^{3-1}$)

Excess (or bias) Notation

Bit Pattern	Value Represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	-1
0110	-2
0101	-3
0100	-4
0011	-5
0010	-6
0001	-7
0000	-8

With 4 bits we have excess 8 ($=2^{4-1}$)

With N bits we have excess 2^{N-1}

We add the magic number, 2^{N-1}, to the integer, change the result to binary, and add 0's (on the left) to make up N bits.

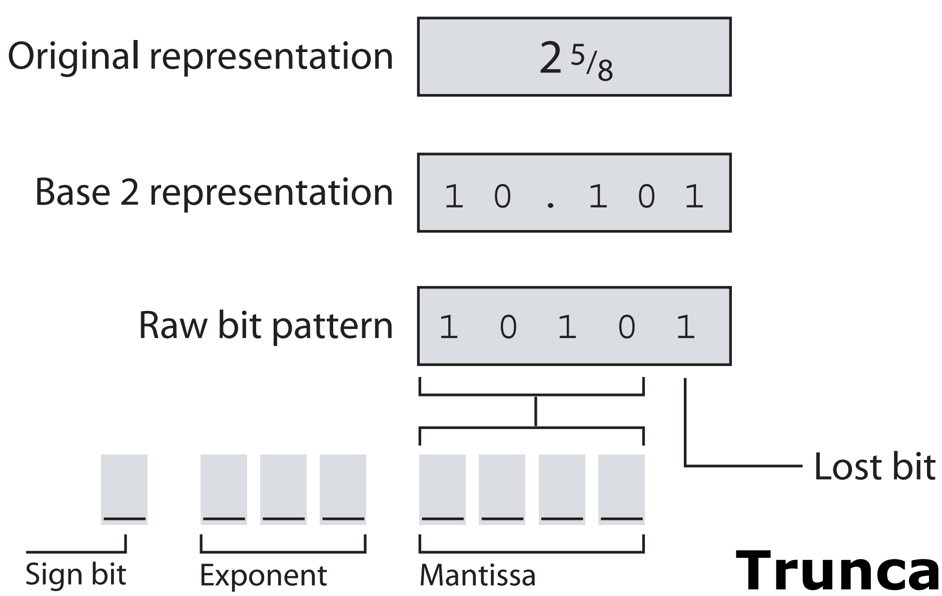
FP Representation

- The standard IEEE representation for FP uses 32 bits for single-precision format:
  - 1 bit sign
  - 8 bits exponent (excess 127 = 2^{8-1}-1)
  - 23 bits mantissa
- Representation. Store the:
  - Sign as 0 (positive), 1 (negative)
  - Exponent (power of 2) as excess127
  - Mantissa as an unsigned integer

Example: FP Representation

- Consider a 16-bit representation with
  - 1 bit sign
  - 5 bits exponent (excess 16 = 2^{5-1})
  - 10 bits mantissa
- 0 10011 1101100000
  - The sign 0 represents positive
  - The exponent 10011 = 19_{10} represents 3
  - The mantissa is .1101100000 (1 to left of . is not stored, it is understood)

Coding the value 2 5/8



Truncation Errors