COMP1001

Computer Systems

Dr. Vasilios Kelefouras

Email: v.kelefouras@plymouth.ac.uk

Website:
https://www.plymouth.ac.uk/staff/vasilios-kelefouras

Outline

- Positional Numbering Systems
- Converting Positional Numbering Systems
- Basic Binary Arithmetic Operations
- Signed Integer Representation
- Floating Point Representation
- Character Codes

Basics (1)

- The bit is the most basic unit of information in a computer
  - Switching activity 0 or 1
- A Byte is a group of 8 bits
  - A byte is the smallest possible addressable unit of computer storage
  - The term, "addressable," means that a particular byte can be retrieved according to its location in memory
- A word is a contiguous group of bytes, e.g., an integer uses 4 bytes
- Word sizes of 4 or 8 bytes are most common

Basics (2)

Kilo- (K) = 1 thousand = $\mathbf{10^3}$ and 2^{10}

Mega- (M) = 1 million = $\mathbf{10^6}$ and 2^{20}

Giga- (G) = 1 billion = $\mathbf{10^9}$ and 2^{30}

Tera- (T) = 1 trillion = $\mathbf{10^{12}}$ and 2^{40}

Peta- (P) = 1 quadrillion = $\mathbf{10^{15}}$ and 2^{50}

Exa- (E) = 1 quintillion = $\mathbf{10^{18}}$ and 2^{60}

Zetta- (Z) = 1 sextillion = $\mathbf{10^{21}}$ and 2^{70}

Yotta- (Y) = 1 septillion = $\mathbf{10^{24}}$ and 2^{80}

Normally, powers of 2 are used for measuring capacity

Milli- (m) = 1 thousandth = $\mathbf{10^{-3}}$

Micro- (µ) = 1 millionth = $\mathbf{10^{-6}}$

Nano- (n) = 1 billionth = $\mathbf{10^{-9}}$

Pico- (p) = 1 trillionth = $\mathbf{10^{-12}}$

Basics (3)

- Hertz = clock cycles per second (frequency)
  - 1MHz = 1,000,000Hz
  - Processor speeds are measured in MHz or GHz
- Byte = a unit of storage
  - 1KB = 2^{10} = 1024 Bytes
  - 1MB = 2^{20} = 1,048,576 Bytes
  - 1GB = 2^{30} = 1,099,511,627,776 Bytes
- Main memory (RAM) is measured in GB
- Disk storage is measured in TB (2^{40})

Think Pair Share activity

- How many milliseconds (ms) are in 1 second?
- How many nanoseconds (ns) are in 1 millisecond?
- How many kilobytes (KB) are in 1 gigabyte (GB)?
- How many bytes are in 20 megabytes?

POSITIONAL NUMBERING SYSTEMS

- Positional numbering systems are systems in which the placement of a digit in connection to its intrinsic value determines its actual meaning in a numeral string
- The organization of any computer depends considerably on how it represents numbers, characters, and control information
  - **There are several positional numbering systems such as Decimal, Binary, Octal, Hexadecimal etc**
- The positioning system is provided as a subscript, e.g., 14_{10}, 10101_{2}, 82_{16}
- Our decimal system is the base-10 system. It uses powers of 10 for each position in a number
- The binary system is also called the base-2 system
- The hexadecimal system is the base-16 system
- The Mayan and other Mesoamerican cultures used a number system based in a base-20 system

Decimal System

- **<u>Decimal system</u>**: Our well known and used system.
  - **It uses 10 different digits: 0,1,2,3,4,5,6,7,8,9**
  - Our decimal system is the base-10 system. It uses powers of 10 for each position in a number
  - For example, the decimal number 947 in powers of 10 is
    947 =

$$=9\times100 + 4\times10 + 7\times1 =$$

$$=9\times10^2 + 4\times10^1 + 7\times10^0$$

  - $70216=7\times10000+0\times1000+2\times100+1\times10+6\times1=$

$$=7\times10^4+0\times10^3+2\times10^2+1\times10^1+6\times10^0$$

- The decimal number 3812.46 in powers of 10 is ($3\times10^3 + 8\times10^2 +1\times10^1+2\times10^0+ 4\times10^{-1}+ 6\times10^{-2}$)

Binary System (1)

- A binary number is a number expressed in the base-2 numeral system or binary numeral system, which uses only two symbols: typically 0 (zero) and 1 (one)
- The base is 2
- **2 different digits are used: 0,1**
- For example, $101_2 = 1\mathbf{x}2^2 + 0\mathbf{x}2^1 + 1\mathbf{x}2^0$

  $= 1\mathbf{x}4 + 0\mathbf{x}2 + 1\mathbf{x}1$

  $= 5_{10}$
- The binary number 11001 in powers of 2 is: $1x2^4 + 1x2^3 + 0x2^2 + 0x2^1 + 1x2^0 = 16 + 8 + 0 + 0 + 1 = 25_{10}$
- $1011.101_2 =$

  $= 1\mathbf{x}2^3 + 0\mathbf{x}2^2 + 1\mathbf{x}2^1 + 1\mathbf{x}2^0 + 1\mathbf{x}2^{-1} + 0\mathbf{x}2^{-2} + 1\mathbf{x}2^{-3} =$

  $= 1\mathbf{x}8 + 0\mathbf{x}4 + 1\mathbf{x}2 + 1\mathbf{x}1 + 1\mathbf{x}0.5 + 0\mathbf{x}0.25 + 1\mathbf{x}0.125$

  $= 11.625_{10}$

Binary System (2)

2^n representation	2^{10}	2^4	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}
number	1024	16	8	4	2	1	0.5	0.25	0.125

Convert the following binary number 1101.101 to decimal

$$1101.101_2 = 1x2^3 + 1x2^2 + 0x2^1 + 1x2^0 + 1x2^{-1} + 0x2^{-2} + 1x2^{-3}$$
$$= 8 + 4 + 0 + 1 + 0.5 + 0 + 0.125 = 13.625_{10}$$

Octal system

- The base is 8
- **8 different digits are used only: 0,1,2,3,4,5,6,7**
- For example: $436_8 = 4\mathbf{x}8^2 + 3\mathbf{x}8^1 + 6\mathbf{x}8^0$

  $= 4\mathbf{x}64 + 3\mathbf{x}8 + 6\mathbf{x}1$

  $= 286_{10}$

Convert the following octal number 205.24_8, to decimal:

$205.24_8 = 2x8^2 + 0\mathbf{x}8^1 + 5\mathbf{x}8^0 + 2x8^{-1} + 4x8^{-2}$

$= 2x64 + 0 + 5 + 2x0.125 + 4x0.015625$

$= \mathbf{133.3125_{10}}$

Hexadecimal system

- The base is 16
- **16 different digits are used: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F**

(we do not use numbers with 2 digits like 10,11,12,...), but

A instead of 10, B instead of 11, C instead of 12, etc)

- Example: $3B1_{16} = 3\mathbf{x}16^2 + 11\mathbf{x}16^1 + 1\mathbf{x}16^0$

$= 3\mathbf{x}256+11\mathbf{x}16+1 =$

$= 768 + 176 + 1 =$

$= 945_{10}$

Convert the following hexadecimal number $20C.2_{16}$ to decimal

$\mathbf{20C.2_{16}} = 2x16^2 + 0x16^1 + 12x16^0 + 2x16^{-1} =$

$= 2x256 + 0 + 12x1 + 2x0.0625 =$

$= 512 + 12 + 0.125 =$

$= \mathbf{524.125_{10}}$

Positional Numbering Systems - General case

- Base: r
- **Uses r different digits: 0,1,2,3,..r-1**

- $N\mathbf{r} = A_{n-1}\ A_{n-2} \ldots A_1\ A_0\ A_{-1}\ A_{-2} \ldots A_{-(m-1)} A_{-m}$
  $N_r = A_{n-1} \times r^{n-1} + A_{n-2} \times r^{n-2} + \ldots + A_1 \times r^1 + A_0 \times r^0 + A_{-1} \times r^{-1} + A_{-2} \times r^{-2} + \ldots + A_{-(m-1)} \times r^{-(m-1)} + A_{-m} \times r^{-m}$

For example if $234.03_5 = ?_{10}$ *then* $n=3$, $m=2$ *and* $r=5$

- The left most digit (A**n-1**) is called Most Significant Bit-(MSB) while the right most (A**-m**) Least Significant Bit-(LSB)

Converting Positional Numbering Systems

From Decimal to Binary

The easiest method of converting integers from decimal to some other base uses division

$$37 \,/\, \underbrace{2}_{\text{Base}} = \underbrace{18}_{\text{Quotient}} \text{ remainder } \underbrace{1}_{\text{Remainder}}$$

Procedure:

a. Divide the decimal by 2

b. Write down the quotient and the remainder

c. Divide quotient by 2

d. Write down the quotient and the remainder

e. Repeat the process (c)-(d) until the quotient becomes zero

f. Write down the binary number from bottom (MSB) to top (LSB)

From Decimal to Binary, an example

$83_{10} = ?_2$

$83 \div 2 = 41$ remainder **1** **(LSB)**

$41 \div 2 = 20$ remainder **1**

$20 \div 2 = 10$ remainder **0**

$10 \div 2 = 5$ remainder **0**

$5 \div 2 = 2$ remainder **1**

$2 \div 2 = 1$ remainder **0**

$1 \div 2 = 0$ remainder **1** **(MSB)**

$83_{10} = 1010011_2$

If the decimal number is lower than 1, e.g., 0.25, or contains a fractional part the procedure applied is different

Our result, is the remainders in reverse order (reading from bottom to top)

Convert From Decimal To Another Base

- We follow the same procedure as in the previous slide (from decimal to binary) but instead of using 2-base we use the r-base.

- **Convert the decimal 524_{10} to hexadecimal**

524:16= 32 remainder 12 ($12_{10}=C_{16}$)

32:16=2 remainder 0

2:16=0 remainder 2

Thus, **$524_{10}= 20C_{16}$**

$66_{10} = ?_2$

$128_{10}= ?_{16}$

Convert from binary to octal

Procedure:

1. ***To convert a binary number of octal***, *we group all the 1's and 0's in the binary number in* ***sets of three***, *starting from the far right*
2. *Start from the right to make your groups*
3. *Add zeros to the left of the last digit if you don't have enough digits to make a set of three*
4. *Write down the decimal representation of every group*

Binary			Octal
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

$101011_2 = ¿_8$

$= 101 \quad 011$

$= 5 \quad 3$

$101011_2 = 53_8$

$10011011_2 = ¿_8$

10 011 011

010 011 011

2 3 3

$10011011_2 = 233_8$

Convert from Octal to binary

- Converting from octal to binary is as easy as converting from binary to octal. Simply look up each octal digit to obtain the equivalent group of three binary digits

$317.2_8 = ?_2$

$= 011\ 001\ 111.\ 010$

$= 11001111.01_2$

Binary			Octal
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7

Convert Binary to Hexadecimal

Procedure:

1. *Cut your string of binary numbers **into groups of four,** starting from the right*
2. *Add extra zeros to the front of the first number if it is not four digits*
3. *Convert one 4-digit group at a time. To convert between Binary and Hexadecimal, you simply replace each Hex digit with its 4-bit binary equivalent and vice versa*

Hexadecimal	Binary
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
A	1010
B	1011
C	1100
D	1101
E	1110
F	1111

$\mathbf{10001101011_2 = ;_{16}}$

$= 100 \quad 0110 \quad 1011$

$= 0100 \quad 0110 \quad 1011$

$= 4 \quad 6 \quad B$

$\mathbf{10001101011_2 = 46B_{16}}$

$0100_2=0+4+0+0=4_{16}$

$0110_2=0+4+2+0=6_{16}$

$1011_2=8+0+2+1=11=B_{16}$

Convert Hexadecimal to Binary

- Likewise from Octal to Binary
- Simply look up each hexadecimal digit to obtain the equivalent group of **four binary digits**

$8B.2_{16} = 1000\ 1011.0010_2$

Hexadecimal	Binary	Decimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
A	1010	10
B	1011	11
C	1100	12
D	1101	13
E	1110	14
F	1111	15

- That's why we use hexadecimal in computer systems! Humans can still understand it, and computers can calculate Hex faster than decimal values!

Basic arithmetic operations

- The basic arithmetic operations are applied to all the previous numerical systems. There are:
  - Addition
  - Subtraction
  - Multiplication
  - Division

- For the reminder of this lecture we will focus on the binary system

Binary Addition (1)

Binary addition is like decimal addition:

1. Put the numbers in a vertical column, aligning the decimal points
2. Add each column of digits, starting on the right and working left. If the sum of a column is more than ten, "carry" digits to the next column on the left.

457 +

148

605

(011 Carry)

- In the example above we add 8+7 and write 5 instead of 15. We propagate 10 (which is the base) to the left and we write the remainder
- For every propagation, we add 1 carry to the next addition (left)
- **The same holds when adding different numerical system numbers too**

Binary Addition (2)

Binary addition:

```
   1 0 1 1 0 1 1 1 +
     1 1 1 0 1 1 0
 1 0 0 1 0 1 1 0 1
( 0 1 1 1 1 0 1 1 0 carry)
```

```
   1 1 1 1 1 0 1 1 +
   1 1 1 0 1 0 1 1
 1 1 1 1 0 0 1 1 0
( 1 1 1 1 1 0 1 1   carry)
```

Note that:

0+0= 0 and carry 0

1+0=0+1= 1 and carry 0

1+1= 0 and carry 1, as $1 + 1 = 10_2 = 2_{10}$

1+1+1= 1 and carry 1, as $1 + 1 + 1 = 11_2 = 3_{10}$

Binary Multiplication

- As in the decimal system:

```
      110111 X
        1101
     110111
    000000
   110111    +
  110111______
 1011001011
(0111221100) Carry
```

Note that 1+1+1+1= 0 and carry 2, as 1+1+1+1= $100_2 = 4_{10}$

Signed integer representation

<u>Introduction</u>

- In practice we have to use **negative** binary numbers too. **We need to define signed binary numbers**

- **There are three ways in which signed binary integers may be expressed:**
  1. **Signed magnitude**
  2. **One's complement**
  3. **Two's complement**

Signed Magnitude Representation (1)

- **Allocate the high-order (leftmost) bit to indicate the sign of a number**
  - The high-order bit is the leftmost bit. It is also called the most significant bit
  - 0 is used to indicate a positive number; 1 indicates a negative number
- The remaining bits contain the value of the number
- Note that we also pay attention to the number of bits used to represent signed binary numbers
  - i.e. if using 4 bit numbers, then we use 0001_2 rather than 1_2
- In an 8-bit word, signed magnitude representation places the absolute value of the number in the 7 bits to the right of the sign bit

For example:

+3 is: **0**0000011

- 3 is: **1**0000011

Signed Magnitude Representation (2)

- The "binary addition algorithm" does NOT work with sign-magnitude

$0 \quad 011_2 = 3_{10}$

$1 \quad 100_2 = -4_{10}$

$$\begin{array}{lr} 0 & 011 \\ 1 & +\ 100 \\ \hline 1 & 111 \end{array}$$

this is wrong

Signed Magnitude: intuitive for humans, difficult for computers

- Signed magnitude representation is easy for people to understand, but it requires complicated computer hardware

- Also it allows two different representations for zero: positive zero and negative zero

- As such, computer systems employ **complement systems** for signed number representation

Signed Integer Representation
Complement Systems

- In binary systems, these are:
  - **One's Complement.** To represent negative values, invert all the bits in the binary representation of the number (swapping 0s for 1s and vice versa)
    - 1 becomes 0 and 0 becomes 1
    - To represent positive numbers no change is applied

    For example, using 8-bit one's complement representation

    + 3 is: 00000011

    - 3 is: 11111100

    More examples

    X=11011100, 1C(X)=00100011

    X=1011, 1C(X)=?

    - One's complement still has the disadvantage of having two different representations for zero: positive zero and negative zero
    - In addition positive and negative integers need to be processed separately
    - Two's complement solves this problem
  - **Two's complement**
    - One's Complement add 1

Signed Integer Representation
Two's Complement

Two's complement 2C(X)

- You represent positive numbers, just like the unsigned numbers
- To represent negative values, start with the corresponding positive number, invert all the bits. Then add 1
- For example, using 8-bit two's complement representation:

+ 3 is: 00000011	*1.Start with positive number*
11111100	*2.Invert bits*
+ 1	*3. Add 1*
- 3 is: 11111101	

-3 in 8-bit Two's Complement Representation is 11111101

✓Negative numbers must always start with '1'
✓Both positive and negative numbers must have the same number of bits

Signed Integer Representation
Two's Complement – Example1

- Example: X=01001 ($+9_{10}$), n=5 bits -> Y=2C(X)= 10111 = -9_{10}

Check: X+Y=

```
  01001
 +10111
 ------
 100000=00000
```

The carry is discarded as the result must be 5 bits

- We can always check whether the two's complement result is correct by adding the two numbers. The result has to be zero. **Note that the result of the addition must be of the n bits, where n is the number of bits of the inputs**
- *Find the negative binary number (two's complement) of the following positive number with 7bits 0101101 (45_{10})*

  Answer 1010011

Signed Integer Representation
Two's Complement – Example2

Find the negative binary number (-12_{10})

- Write down the positive $12_{10} = 01100_2$
- Decide the number of the bits. We can use 5 or more. Let say 8 bits
- Find the two's complement as follows

$12_{10} = 00001100_2$,-12_{10} = 11110011+1=**11110100**

-12_{10} =11110100 (negative number)

Wrap up

Given a positive binary number, we find its negative binary number by following the procedure:

1. *We decide the number of bits of the positive number. At least one '0' has to appear on its left.*
2. *We find its two's complement*
3. *If the MSB (the left most) is not '1' then we made a mistake…*

Subtraction (with two's complement)

- We know that A-B=A+(-B)
- So, instead of applying a subtraction we can make an addition with the opposite number, i.e., the two's complement. *The procedure follows:*
  1. *Find –B, i.e., its two's complement*
  2. *Add A with B's two complement*
  3. ***The result has as many bits as the inputs***

Make the subtraction Z=12-5 (use 5 digits)

$12_{10}=01100_2$, $5_{10}=00101_2$, $-5_{10}=11011_2$

Z=01100 + 11011=100111, but we need 5 bits thus $Z=00111_2=7_{10}$

Z=00111

Make the subtraction 9-12 (use 5 digits)

$9_{10}=01001_2$, $12_{10}=01100_2$, $-12_{10}=10100_2$

Z=01001 + 10100=11101 and thus Z=11101 (-3_{10})

Z=11101

Unsigned and Signed Integer Representation

- **Both signed and unsigned numbers are useful**
  - For example, memory addresses are always unsigned
- Using the same number of bits, unsigned integers can express twice as many "positive" values as signed numbers.
  - For example, the range of values that can be represented in **4-bits** is:
    - 0000_2 to 1111_2 (or 0 to 15) as unsigned
    - 0111_2 to 1111_2 (or +7 to -7) as signed magnitude
    - 0111_2 to 1000_2 (or +7 to -8) as two's complement

Example #1

What decimal value does the 8-bit binary number 10011110 have if

a) it is interpreted as an unsigned number?

b) it is on a computer using signed-magnitude representation?

c) it is on a computer using one's complement representation?

d) it is on a computer using two's complement representation?

Answer:

a. **$10011110_2 = 1x2^7 + 1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 = 128+16+8+4+2=158_{10}$**

b. **10011110_2 = 1 (negative) 0011110_2 =- $1x2^4 + 1x2^3 + 1x2^2 + 1x2^1 = -30_{10}$**

c. Find the positive of **10011110_2** ;invert the bits of **10011110_2** and therefore **01100001_2 . 01100001_2 =$1x2^6 + 1x2^5 + 1x2^0$=64+32+1=97_{10}**. Since the original number was negative, the number is **-97_{10}**

d. Find the positive of **10011110_2** ;invert the bits of **10011110_2 and add 1; therefore 01100010_2 . 01100010_2 =$1x2^6 + 1x2^5 + 1x2^1$=64+32+2=98_{10}**. Since the original number was negative, the number is **-98_{10}**

Example #2

What decimal value does the 8-bit binary number 00010001 have if

a) it is interpreted as an unsigned number?

b) it is on a computer using signed-magnitude representation?

c) it is on a computer using one's complement representation?

d) it is on a computer using two's complement representation?

Answer:

a. $00010001_2 = 1x2^4 + 1x2^0 = 16+1=17_{10}$

b. $00010001_2 = 0$ (positive) $0010001_2 = 1x2^4 + 1x2^0 = 16+1=17_{10}$

c. $00010001_2 = 1x2^4 + 1x2^0 = 16+1=17_{10}$

d. $00010001_2 = 1x2^4 + 1x2^0 = 16+1=17_{10}$

Example #3

Perform the following binary subtraction using two's complement representation: Z=8 – 6

Answer:

1. Instead of subtraction we do addition: 8 - 6 =8 + (-6)
2. Choose the number of bits to represent 8 and 6 decimal numbers to binary. **At least one zero has to appear on the left.** Thus, we need 5 bits or more

   $8_{10}=01000_2$

   $6_{10}=00110_2$

3. Find - 6_{10} (two's complement) – 6_{10} = 11010_2
3. Make the addition (***The result has 5 bits***)

$$
\begin{array}{r}
0\,1\,0\,0\,0 \\
1\,1\,0\,1\,0 \\
\hline
1\,\boxed{0\,0\,0\,1\,0}
\end{array}
$$

-> $\mathbf{00010_2=2_{10}}$

Signed Integer Representation Multiplication and Division by 2 (1)

- Binary multiplication and division by 2 is very easy. (as easy as it is to multiply and divide by 10 in the decimal system)
- Simply use an arithmetic shift operation

$$1_2 = 1_{10}$$
$$10_2 = 2_{10}$$
$$100_2 = 4_{10}$$
$$1000_2 = 8_{10}$$
$$10000_2 = 16_{10}$$

- A left arithmetic shift inserts a 0 in for the rightmost bit and shifts everything else left one bit; in effect, it multiplies by 2
- A right arithmetic shift shifts everything one bit to the right, but copies the sign bit; it divides by 2

Signed Integer Representation Multiplication and Division by 2 (2)

- Multiplication by 2
  - Shift left by one place
  - E.g. to calculate 2 * 7
- Division by 2
  - Shift right by one place
  - E.g., to calculate 14/2

Binary	Decimal
0000 0111	+7
0000 1110	+14

- To multiply by 4, we perform a left shift twice
- To divide by 4, we perform a right shift twice

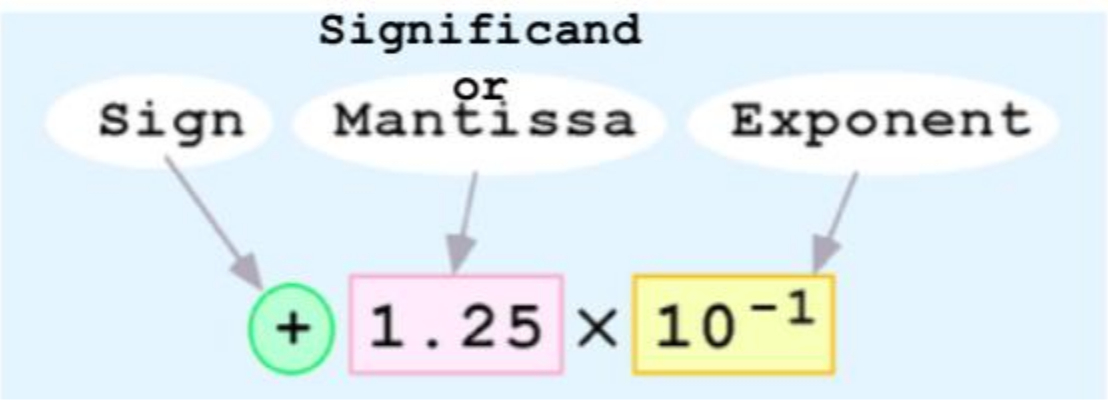
- Using arithmetic shifting, perform the following:
  a) double the value 00010101_2
  b) quadruple the value 00110111_2
  c) divide the value 11001010_2 in half

Floating-Point Representation (1)

- To represent real numbers with fractional values, floating-point representation is used
- Floating-point numbers are often expressed in scientific notation
  - For example: $0.125 = 1.25 \times 10^{-1}$
- Remember that when a number is multiplied by its base, e.g., 10, then we add a zero or we move the ',' by one position to the right
  - $235 \times 10 = 2350$
  - $1.345 \times 10 = 13.45$
  - $110_2 \times 2 = 1100_2$ ($6 \times 2 = 12_{10}$)
  - $101.11_2 \times 2 = 1011.1$ ($5.75 \times 2 = 11.5_{10}$)

Floating-Point Representation (2)

- Computers use a form of scientific notation for floating-point representation
  - Single Precision floating point format 32-bit
  - Double Precision floating point format 64-bit
- Numbers written in scientific notation have three components:



Single precision Floating-Point format (1)

A binary number is represented in FP format as follows:

1. We write the number using only a single non-zero digit before the radix point :

   e.g., $1011010010001 = 1.011010010001 \times 2^{12}$

   $1101.10111 = 1.10110111 \times 2^{3}$

2. Then we transform the number to the following format using 32 bits

$$N = (-1)^{S}\,(1+F)(2^{E-127})$$

Sign-S 1-bit	Exponent-E 8 - bits	Mantissa (Fraction) - F 23 - bits

S: Sign, 0/1 for positives/negatives, respectively

E: Exponent. E-127=exp, where exp is the corresponding exponent

F: Significant or Mantissa. We write the fractional part in 23 bits

E=127+exp in order to avoid using negative numbers. exp=[-127,128] and therefore E=[0,255] – 255 needs 8 bits

Single precision Floating-Point format (2)

Convert the positive number N=1011010010001 in Floating point format

Step1: 1011010010001= 1.011010010001 x 2^{12}

Step2: **$N = (-1)^S (1+F)(2^{E-127})$**

S = 0 (positive number)

E - 127 = 12, and thus E = 139_{10} and E = 10001011_2

F = 01101001000100000000000

Therefore N in FP format is:

0	10001011	01101001000100000000000

Single precision Floating-Point format (3)

Suppose that the 32-bit floating-point representation pattern is the following. Find the binary number

1	10010001	10001110001000000000000

S is 1 and thus the number is negative

E is 10010001 = 145_{10}, and thus the exponent is exp=E-127=145-127=18

F= 10001110001000000000000

$N = (-1)^S (1+F)(2^{E-127})$

N is $(-1)^1 \times 1.10001110001000000000000 \times 2^{18}$ or

N = - 1100011100010000000

Floating-Point Representation (1)

- No matter how many bits we use in a FP representation, the model is finite
  - The real number system is, of course, infinite, so our models can give nothing more than an approximation of a real value
  - e.g., how to represent 33.33333333333333333333?
- At some point, every model breaks down, introducing errors into our calculations
  - By using a greater number of bits in our model, we can reduce these errors, but we can never totally eliminate them

Why is 0.1+0.2 not equal to 0.3 in most programming languages?

- computers use a binary floating point format that cannot accurately represent a number like 0.1_{10}
- 0.1_{10} is already rounded to the nearest number in that format
- 0.1_{10} doesn't exist in the FP representation
- 0.1_{10} is already rounded to the nearest number in that format, which results in a small rounding error
- This means that 0.1_{10} is converted to a binary number that's just very close to 0.1_{10}
- The error is tiny since 0.1_{10} is 0.1000000000000000055511151231257827
- The constants 0.2_{10} and 0.3_{10} are also approximations to their true values
- So, $0.1_{10} + 0.2_{10} == 0.3000000000000000444089209850006_{10}$

Character Codes

- So far, we have learnt how to represent numbers. How about text?
- To represent text characters, we use character codes
  - Essentially, we assign a number for each character we want to represent
- As computers have evolved, character codes have evolved. Larger computer memories and storage devices permit richer character codes
- Some of the character codes are
  1. BCD
  2. ASCII (American Standard Code for Information Interchange) (7 bits)
  3. Extended ASCII (8-bits)
  4. Unicode
  5. and others
- A binary number of n bits gives 2^n different codes
  - For n=2 there are $2^2=4$ different codes, i.e., bit combinations {00, 01, 10, 11}

Binary Coded Decimal (BCD) code

- when numbers, letters or words are represented by a specific group of symbols, it is said that the number, letter or word is being encoded. The group of symbols is called as a code

- **Binary Coded Decimal (BCD) code**
  - In this code each decimal digit is represented by a 4-bit binary number
  - BCD is a way to express each of the decimal digits with a binary code
  - In the BCD, with four bits we can represent sixteen numbers (0000 to 1111)

$256_{10} = 0010\ 0101\ 0110_{BCD}$

And vise versa

$0011\ 1000\ 1001_{BCD} = 389_{10}$

ASCII Code

- The most widely accepted code is called the American Standard Code for Information Interchange (ASCII).
- The ASCII code associates an integer value for each symbol in the character set, such as letters, digits, punctuation marks, special characters, and control characters
- The ASCII table has 128 characters, with values from 0 through 127. Thus, 7 bits are sufficient to represent a character in ASCII

ASCII Code

Dec	Hx	Oct	Char	
0	0	000	NUL	(null)
1	1	001	SOH	(start of heading)
2	2	002	STX	(start of text)
3	3	003	ETX	(end of text)
4	4	004	EOT	(end of transmission)
5	5	005	ENQ	(enquiry)
6	6	006	ACK	(acknowledge)
7	7	007	BEL	(bell)
8	8	010	BS	(backspace)
9	9	011	TAB	(horizontal tab)
10	A	012	LF	(NL line feed, new line)
11	B	013	VT	(vertical tab)
12	C	014	FF	(NP form feed, new page)
13	D	015	CR	(carriage return)
14	E	016	SO	(shift out)
15	F	017	SI	(shift in)
16	10	020	DLE	(data link escape)
17	11	021	DC1	(device control 1)
18	12	022	DC2	(device control 2)
19	13	023	DC3	(device control 3)
20	14	024	DC4	(device control 4)
21	15	025	NAK	(negative acknowledge)
22	16	026	SYN	(synchronous idle)
23	17	027	ETB	(end of trans. block)
24	18	030	CAN	(cancel)
25	19	031	EM	(end of medium)
26	1A	032	SUB	(substitute)
27	1B	033	ESC	(escape)
28	1C	034	FS	(file separator)
29	1D	035	GS	(group separator)
30	1E	036	RS	(record separator)
31	1F	037	US	(unit separator)

Dec	Hx	Oct	Html	Chr
32	20	040		Space
33	21	041	!	!
34	22	042	"	"
35	23	043	#	#
36	24	044	$	$
37	25	045	%	%
38	26	046	&	&
39	27	047	'	'
40	28	050	(	(
41	29	051	)	)
42	2A	052	*	*
43	2B	053	+	+
44	2C	054	,	,
45	2D	055	-	-
46	2E	056	.	.
47	2F	057	/	/
48	30	060	0	0
49	31	061	1	1
50	32	062	2	2
51	33	063	3	3
52	34	064	4	4
53	35	065	5	5
54	36	066	6	6
55	37	067	7	7
56	38	070	8	8
57	39	071	9	9
58	3A	072	:	:
59	3B	073	;	;
60	3C	074	<	<
61	3D	075	=	=
62	3E	076	>	>
63	3F	077	?	?

Dec	Hx	Oct	Html	Chr
64	40	100	@	@
65	41	101	A	A
66	42	102	B	B
67	43	103	C	C
68	44	104	D	D
69	45	105	E	E
70	46	106	F	F
71	47	107	G	G
72	48	110	H	H
73	49	111	I	I
74	4A	112	J	J
75	4B	113	K	K
76	4C	114	L	L
77	4D	115	M	M
78	4E	116	N	N
79	4F	117	O	O
80	50	120	P	P
81	51	121	Q	Q
82	52	122	R	R
83	53	123	S	S
84	54	124	T	T
85	55	125	U	U
86	56	126	V	V
87	57	127	W	W
88	58	130	X	X
89	59	131	Y	Y
90	5A	132	Z	Z
91	5B	133	[	[
92	5C	134	\	\
93	5D	135	]	]
94	5E	136	^	^
95	5F	137	_	_

Dec	Hx	Oct	Html	Chr
96	60	140	`	`
97	61	141	a	a
98	62	142	b	b
99	63	143	c	c
100	64	144	d	d
101	65	145	e	e
102	66	146	f	f
103	67	147	g	g
104	68	150	h	h
105	69	151	i	i
106	6A	152	j	j
107	6B	153	k	k
108	6C	154	l	l
109	6D	155	m	m
110	6E	156	n	n
111	6F	157	o	o
112	70	160	p	p
113	71	161	q	q
114	72	162	r	r
115	73	163	s	s
116	74	164	t	t
117	75	165	u	u
118	76	166	v	v
119	77	167	w	w
120	78	170	x	x
121	79	171	y	y
122	7A	172	z	z
123	7B	173	{	{
124	7C	174			|
125	7D	175	}	}
126	7E	176	~	~
127	7F	177	&#127;	DEL

www.LookupTables.com

Source: www.LookupTables.com

Extended ASCII Characters

- ASCII was designed in the 1960s for teleprinters and telegraphy, and some computing
- The number of printable characters was deliberately kept small, to keep teleprinters and line printers inexpensive
- When computers and peripherals standardized on eight-bit bytes, it became obvious that computers and software could handle text that uses 256-character sets at almost no additional cost in programming, and no additional cost for storage
- An eight-bit character set (using one byte per character) encodes 256 characters, so it can include ASCII plus 128 more characters
- The extra characters represent characters from foreign languages and special symbols for drawing pictures

A set of codes that extends the basic ASCII set. The extended ASCII character set uses 8 bits, which gives it an additional 128 characters

128	Ç	144	É	160	á	176	░	192	└	208	╨	224	α	240	≡
129	ü	145	æ	161	í	177	▒	193	┴	209	╤	225	ß	241	±
130	é	146	Æ	162	ó	178	▓	194	┬	210	╥	226	Γ	242	≥
131	â	147	ô	163	ú	179	│	195	├	211	╙	227	π	243	≤
132	ä	148	ö	164	ñ	180	┤	196	─	212	╘	228	Σ	244	⌠
133	à	149	ò	165	Ñ	181	╡	197	┼	213	╒	229	σ	245	⌡
134	å	150	û	166	ª	182	╢	198	╞	214	╓	230	µ	246	÷
135	ç	151	ù	167	º	183	╖	199	╟	215	╫	231	τ	247	≈
136	ê	152	ÿ	168	¿	184	╕	200	╚	216	╪	232	Φ	248	°
137	ë	153	Ö	169	⌐	185	╣	201	╔	217	┘	233	Θ	249	∙
138	è	154	Ü	170	¬	186	║	202	╩	218	┌	234	Ω	250	·
139	ï	155	¢	171	½	187	╗	203	╦	219	█	235	δ	251	√
140	î	156	£	172	¼	188	╝	204	╠	220	▄	236	∞	252	ⁿ
141	ì	157	¥	173	¡	189	╜	205	═	221	▌	237	φ	253	²
142	Ä	158	₧	174	«	190	╛	206	╬	222	▐	238	ε	254	■
143	Å	159	ƒ	175	»	191	┐	207	╧	223	▀	239	∩	255	

Source: www.LookupTables.com

www.LookupTables.com

UNICODE

- Many of today's systems embrace Unicode that can encode the characters of every language in the world
  - The Java programming language, and some operating systems now use Unicode as their default character code
    - UTF-8 (8-bits: essentially the extended ASCII Table)
    - UTF-16 (16 bits: Most spoken languages in the world, widely used)
    - UTF-32 (32 bits: includes past languages, space inefficient)

Any questions?

Further Reading

- Chapter 9 and chapter 10 in 'Computer Organization and architecture' available at http://home.ustc.edu.cn/~louwenqi/reference_books_tools/Computer%20Organization%20and%20Architecture%2010th%20-%20William%20Stallings.pdf