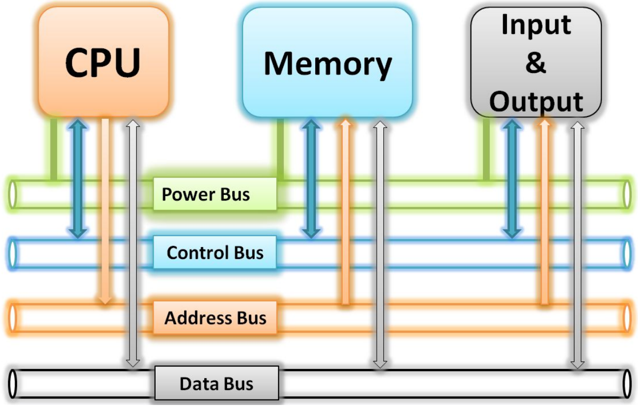
Yrkes Akademin
Vi hjälper dig att lyckas!

Programming

Introduction

Computer System

- What is a computer system?
  - An electronic device designed to retrieve, process and store **data** using an **instruction set.**
- Basic System Architecture: three components connected to a bus system
  - Processor
    - The computing and controlling part of a computer system which processes data
  - Memory (RAM)
    - Is used by the processor to retrieve and store data
    - Is organized in bytes and every byte can be addressed
  - I/O Devices
    - Are used by the processor to communicate with the external world



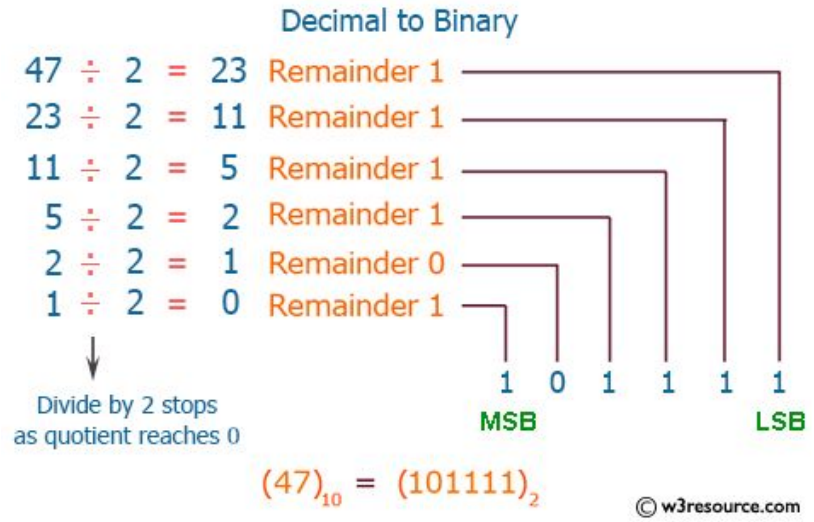
- Instruction Set (typically 100 to 1000 micro instructions)
  - A set of primitive arithmetic, logical and controlling micro instructions executable by a processor.
  - E.g. Add two numbers, jump to an address, load/store data from/to memory, and etc.

Numbering Systems

- What is data? Binary numerical values.
- Base 10 numbering system (**decimal**)
  - There are **10** digits starting from **0** and ending to **9**.
  - All numbers are created by combining the **10** digits.
  - E.g. $123_{10} = 1\times10^2 + 2\times10^1 + 3\times10^0 = 123$
- Base 2 numbering system (**binary**)
  - There are only **2** digits; **0** and **1**. All numbers are created by combining the **2** digits.
  - E.g. **0b**1011 = $1011_2 = 1\times2^3 + 0\times2^2 + 1\times2^1 + 1\times2^0 = 8 + 0 + 2 + 1 = 11_{10}$
- Base 8 numbering system (**octal**)
  - There are **8** digits; 0, 1, 2, 3, 4, 5, 6, 7. All numbers are created by combining the **8** digits.
  - E.g. **0**$56_8 = 5\times8^1 + 6\times8^0 = 40 + 6 = 46_{10}$

Numbering Systems

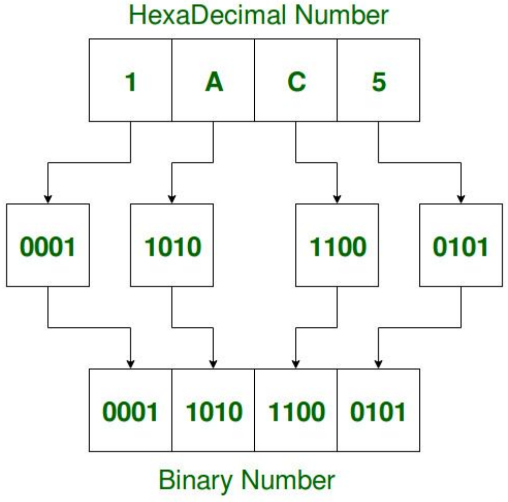
- Base 16 numbering system (**hexadecimal**)
  - There are **16** digits; 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.
  - $A_{16} = 10_{10}$, $B_{16} = 11_{10}$, $C_{16} = 12_{10}$, $D_{16} = 13_{10}$, $E_{16} = 14_{10}$ and $F_{16} = 15_{10}$
  - E.g. **0x**$F6_{16} = 15 \times 16^1 + 6 \times 16^0 = 240 + 6 = 246_{10}$
- Convert from **decimal** to **binary** and vice versa
  - Decimal to binary
    - Successive division by 2 of the decimal number until the result of the division gets 0. The remainders from bottom to top are the digits of the binary number.
    - **MSB** means the **M**ost **S**ignificant **B**it
    - **LSB** means the **L**east **S**ignificant **B**it



  - Binary to Decimal: Sum of binary digits times 2 power of their position.
    - E.g. **0b**$101010 = 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 1^2 + 1 \times 2^1 + 0 \times 2^0 = 32 + 0 + 8 + 0 + 2 + 0 = 42$

Numbering Systems

- Convert from **hexadecimal** to **binary** and vice versa
  - Hexadecimal to binary
    - Convert each hex digit to its equivalent 4 binary digits
    - Discard any leading zeros at the left of the binary number.
    - E.g. **0x**1AC5 = 0001 1010 1100 0101 = **0b**1101011000101
  - Binary to hexadecimal
    - Every 4 binary digits (start from right) to its equivalent hex digit
    - E.g. **0b**1101011000101 = 0001 1010 1100 0101 = **0x**1AC5



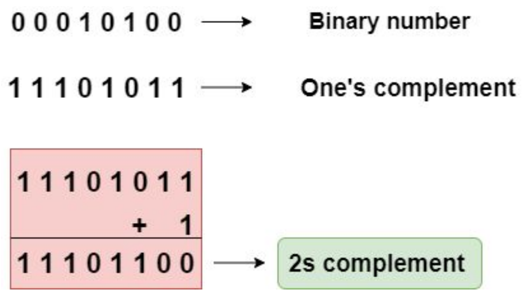
- Binary arithmetic operations are like what we do for decimal numbers. For example
  - Addition - probably we have some carry out numbers
  - Subtraction - probably we need to borrow
  - **Overflowing and borrowing an imaginary digit are possible**

```
    2 1
  0 0 2 2
  1 1 0 0 1 1
-   1 1 1 0 0
-------------
    1 0 1 1 1
```

```
 51
-28
----
 23
```

```
  0111
  00111     7
+ 10101    21
-----     --
  11100  = 28
```

Numbering Systems



- ❖ Binary negative numbers: 2’s complement
  - ➢ A placeholder is required. E.g. one byte ≡ 8 bits
  - ➢ **MSB** is the sign bit. **0** means positive and **1** means negative
  - ➢ Invert the digits and then add 1 to the inverted value.

    The result is the negative value of the the positive number
- ❖ Boolean Algebra

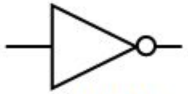
TTL Voltage Level	CMOS Voltage Level	Voltage Level	Binary Value	Boolean Value
0.0v - 0.5v	0.0v - 0.8v	L (Low)	0	F (False)
2.0v - 5.0v	2.0v - 3.3v	H (High)	1	T (True)

- TTL and CMOS voltage levels depend on the used technologies and input/output voltages are different

- ➢ In boolean algebra NOT, AND, OR, XOR and parentheses ( ) operators are defined

Boolean Algebra and Symbols

- **NOT** operator ($\overline{A}$, !A, ~A)
- **OR** operator (A + B, A || B, A | B)
- **AND** operator (A ● B, A && B, A & B)
- **XOR** operator (A ⊕ B, !=, ^)
- Parentheses **( )**
  - Are used to prioritize expressions
- **NAND** which is NOT AND
  - For example $(\overline{A \bullet B})$
- **NOR** which is NOT OR
  - For example $(\overline{A + B})$
- **XNOR** which is NOT XOR
  - For example $(\overline{A \oplus B})$



NOT

Input	Output
I	F
0	1
1	0

AND

Inputs		Output
A	B	F
0	0	0
1	0	0
0	1	0
1	1	1

NAND

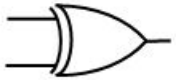
Inputs		Output
A	B	F
0	0	1
1	0	1
0	1	1
1	1	0

OR

Inputs		Output
A	B	F
0	0	0
1	0	1
0	1	1
1	1	1

NOR

Inputs		Output
A	B	F
0	0	1
1	0	0
0	1	0
1	1	0



Inputs		Output
A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE OR

EXCLUSIVE NOR

Inputs		Output
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1

Operators

- Logical operators
  - Zero is interpreted as FALSE (0) and any number except zero is interpreted as TRUE (1)
  - There are NOT(**!**), AND(**&&**) and OR(**||**)
  - De Morgan's laws
    - $\overline{(A \bullet B)} = \overline{A} + \overline{B}$ and $\overline{(A + B)} = \overline{A} \bullet \overline{B}$

A	B	$\overline{A}$	$\overline{B}$	$A+B$	$A \cdot B$	$\overline{A+B}$	$\overline{A} \cdot \overline{B}$	$\overline{A \cdot B}$	$\overline{A} + \overline{B}$
0	0	1	1	0	0	1	1	1	1
0	1	1	1	1	0	0	0	1	1
1	0	0	0	1	0	0	0	1	1
1	1	0	0	1	1	0	0	0	0

De Morgan's laws

  - Some examples
    - !0 is true, !12 is false, 12 && 0 is false, 0 || 12 is true, 0 != 12 is true and 0 == 12 is false
    - !(0 && 12) = (!0 || !12) = (1 || 0) = 1 (true) and !(0 || 12) = (!0 && !12) = (1 && 0) = 0 (false)

- Relational Operators
  - The defined relational operators are **==**, **!=**, **>**, **>=**, **<** and **<=**
    - E.g. (7 == 12) is false, (7 != 12) is true, (7 > 12) is false, (7 >= 12) is false, (7 < 12) is true and (7 <= 12) is true

Operators

- Bitwise operators
  - There are NOT(**~**), AND(**&**), OR(**|**), XOR(**^**), Shift to left(**<<**) and Shift to right(**>>**)
  - Are used to perform bit-level operations
  - A placeholder is required.
  - Some examples
    - 5 & 3 = 1, 5 | 3 = 7 and 5 ^ 3 = 6
    - 5 >> 1 = 2 and 5 << 1 = 10
    - ~0 = 0b11111111 = 255 = 0xFF

	Decimal value	Binary value
Input 1	5	00000101
Input 2	3	00000011
Bitwise And	1	00000001
Bitwise Or	7	00000111
Bitwise XOr	6	00000110

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise left shift
>>	Bitwise right shift

- Bit manipulation
  - The memory is organized by bytes. Not possible to access data in bit level directly
  - Using bitwise operators and masks we can manipulate data in the bit level
  - E.g. to reset the second bit of 10101010 we can **AND** it with 11111101 => 10101000

Bit manipulation

- Setting the ***n*th** bit of a number (N bits)
  - Facts: (D | 0) = D and (D | 1) = 1; D is a bit (0 or 1)
  - Result = (number | (1 << n)); N > n ≥ 0
- Resetting(clearing) the ***n*th** bit of a number (N bits)
  - Facts: (D & 1) = D and (D & 0) = 0; D is a bit (0 or 1)
  - Result = (number & (~(1 << n))); N > n ≥ 0
- Toggling the ***n*th** bit of a number (N bits)
  - Facts: (D ^ 1) = ~D and (D ^ 0) = D; D is a bit (0 or 1)
  - Result = (number ^ (1 << n)); N > n ≥ 0
- Getting the value of the ***n*th** bit of a number (N bits)
  - bit_value = (number >> n) & 1; N > n ≥ 0 and bit_value is 0 or 1
- Changing the value of the ***n*th** bit of a number (N bits)
  - Result = (number & (~(1 << n))) | (D << n); N > n ≥ 0 and **D** is 0 or 1

Example: Toggle the third bit of 53

```
   00110101
^
   00000100 = (1 << 2)
-------------------
   00110001
```