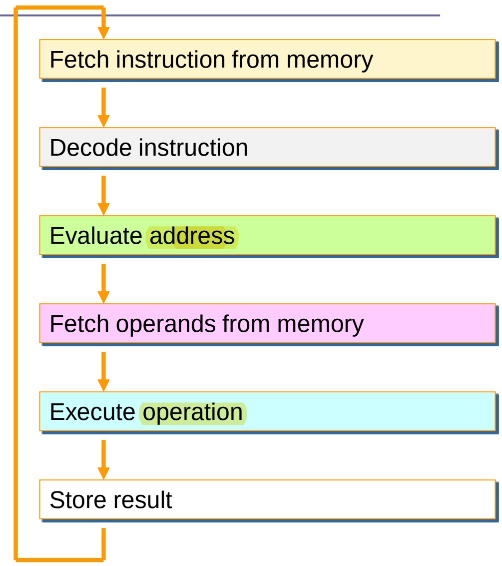
Instruction Processing

- The instruction is the fundamental unit of work.
- Specifies two things:
  - **Opcode:** operation to be performed
  - **Operands:** data/locations to be used for operation



Introduction to Main Digital Component

Introduction to Digital Logic Basics

- Hardware consists of a few simple building blocks
  - These are called *logic gates*
    - AND, OR, NOT, …
    - NAND, NOR, XOR, …
- Logic gates are built using transistors
- Transistors are the fundamental devices
    - Pentium consists of 3 million transistors
    - Compaq Alpha consists of 9 million transistors
    - Now chips can be built with more than 100 million transistors

Data Representation/Binary numbers

- Almost all modern computers are ***digital computers***, which means that they can recognize only two distinct electronic states of electrical charge. For simplicity, these states are identified as **0** and **1**, or equivalently, **false** and **true**, or **off** and **on**. Since 0 and 1 are the most compact means of representing two states, data is represented as sequences of 0's and 1's. Sequences of 0's and 1's are binary numbers.

Integer Number

- The number system that we are used to is a decimal number system because it is base 10. For example:
- 54318 = $5x10^4 + 4x10^3 + 3x10^2 + 1x10^1 + 8x10^0$
- = 5x10000 + 4x1000 + 3x100 + 1x10 + 8x1
- = 50000 + 4000 + 300 + 10 + 8
- = 54318

Binary Number

- To convert from decimal to binary, start with the binary number and keep dividing by 2, writing the remainder (of any) after each division. Keep doing this until reach one. The result, then, is the remainders, starting from the bottom. Here's an example:
- 132-------0
- 66--------0
- 33--------1
- 16--------0
- 8---------0
- 4---------0
- 2---------0
- 1

- Starting from the 1 at the bottom, the binary equivalent of 132 is 10000100.

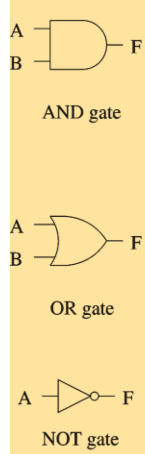
Binary Number

- The binary number system works like the decimal number system, but it is a base 2 system. To convert binary to decimal, use the same method used above but use base 2.

- 11010101 = $1x2^7 + 1x2^6 + 0x2^5 + 1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 1x2^0$
  = 1x128 + 1x64 + 0x32 + 1x16 + 0x8 + 1x4 + 0x2 + 1x1
  = 128 + 64 + 0 + 16 + 0 + 4 + 0 +1
  = 213

Basic Concepts

- **<u>Logical operations (Logic Gates)</u>**
- Simple gates
  - AND
  - OR
  - NOT
- Functionality can be expressed by a truth table
  - A truth table lists output for each possible input combination



Logic symbol

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1

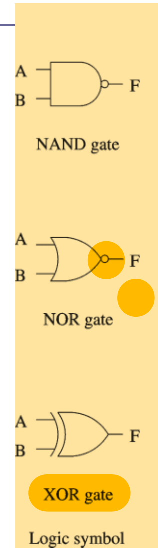
A	F
0	1
1	0

Truth table

18

Basic Concepts

- Additional useful gates
  - NAND
  - NOR
  - XOR
- NAND = AND + NOT
- NOR = OR + NOT



A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0

Truth table

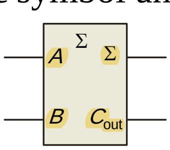
19

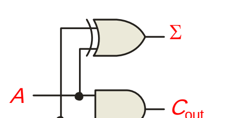
Half Adder

- **Arithmetic Operations: Binary Addition**
- Basic rules of binary addition are performed by a **half adder**, which has two binary inputs (A and B) and two binary outputs (Carry out and Sum).
- The inputs and outputs can be summarized on a truth table.

Inputs		Outputs	
A	B	C_{out}	Σ
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- The logic symbol and equivalent circuit are:





Computer Organization

Full-Adder

- By contrast, a **full adder** has three binary inputs (A, B, and Carry in) and two binary outputs (Carry out and Sum).
- The truth table summarizes the operation.

Inputs			Outputs	
A	B	C_{in}	C_{out}	Σ
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

- A full-adder can be constructed from two half adders as shown: