

# Computing Overview

---

**Sumaiya Tasnim**  
**Lecturer(Provisional)**  
**Department of CSE**  
**Varendra University**



---

# CSE-425

## Lecture-2

Reference: First chapter of textbook

# Lecture Objective

---

- To understand the software and hardware of a microcontroller-based system, one must first master the basic concepts underlying computer architecture.
  - This lecture has been planned as a review of some previous computer architecture and digital logic design terms so that the students can refresh their mind on these topics.

# Contents

---

- Numbering & Coding System
  - Digital Primer
- Semiconductor Memory
  - CPU Architecture

# Numbering & Coding System

---

- Human beings use base 10 (decimal) number system
  - There are 10 distinct symbols, 0, 1, 2, ..., 9
- Computers use base 2 (binary) system
  - There are only 0 and 1
  - These two binary digits are commonly referred to as bits

---

To a computer, everything is a number

Numbers, letters, punctuation marks, sounds, pictures even the computer's own instructions are numbers to a computer

Consider the following sentence:

*Here are some words*

This sentence is nothing but a string of zeros and ones to a computer!!

---

H 0100 1000

e 0110 0101

r 0111 0010

e 0110 0101

0010 0000

a 0110 0001

r 0111 0010

e 0110 0101

0010 0000

s 0111 0010

o 0110 1111

m 0110 0010

e 0110 0101

0010 0000

w 0110 0001

o 0111 0010

r 0110 0101

d 0010 0000


s 0111 0011

0010 1110

# Converting from Decimal to Binary

- Divide the decimal number by 2 repeatedly
- Keep track of the remainders
- Continue this process until the quotient becomes zero
- Write the remainders in reverse order to obtain the binary number

**Ex. Convert  $25_{10}$  to binary**

	<b>Quotient</b>	<b>Remainder</b>	
<b><math>25/2 =</math></b>	<b>12</b>	<b>1</b>	<b>LSB (least significant bit)</b>
<b><math>12/2 =</math></b>	<b>6</b>	<b>0</b>	
<b><math>6/2 =</math></b>	<b>3</b>	<b>0</b>	
<b><math>3/2 =</math></b>	<b>1</b>	<b>1</b>	
<b><math>1/2 =</math></b>	<b>0</b>	<b>1</b>	<b>MSB (most significant bit)</b>

**Therefore  $25_{10} = 11001_2$**



# Converting from Decimal to Binary

---

- Use the concept of weight to convert a decimal number to a binary directly

**Ex. Convert  $39_{10}$  to binary**

$$32 + 0 + 0 + 4 + 2 + 1 = 39$$

**Therefore,  $39_{10} = 100111_2$**

# Converting from Binary to Decimal

---

- Know the weight of each bit in a binary number
- Add them together to get its decimal equivalent

**Ex. Convert  $11001_2$  to decimal**

<b>Weight:</b>	<b><math>2^4</math></b>	<b><math>2^3</math></b>	<b><math>2^2</math></b>	<b><math>2^1</math></b>	<b><math>2^0</math></b>
<b>Digits:</b>	<b>1</b>	<b>1</b>	<b>0</b>	<b>0</b>	<b>1</b>
<b>Sum:</b>	<b>16 +</b>	<b>8 +</b>	<b>0 +</b>	<b>0 +</b>	<b>1 = <math>25_{10}</math></b>

# Hexadecimal System

---

- Base 16, the hexadecimal system, is used as a convenient representation of binary numbers
  - For example: It is much easier to represent a string of 0s and 1s such as 100010010110 as its hexadecimal equivalent of 896H

Decimal	Binary	Octal	Hexadecimal
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

# Conversion between Binary and Hexadecimal

---

- To represent a binary number as its equivalent hexadecimal number, start from the right and group 4 bits at a time, replacing each 4-bit binary number with its hex equivalent

**Ex. Represent binary 100111110101 in hex**

	<b>1001</b>	<b>1111</b>	<b>0101</b>
<b>=</b>	<b>9</b>	<b>F</b>	<b>5</b>

- To convert from hex to binary, each hex digit is replaced with its 4-bit binary equivalent

**Ex. Convert hex 29B to binary**

	<b>2</b>	<b>9</b>	<b>B</b>
<b>=</b>	<b>0010</b>	<b>1001</b>	<b>1011</b>

# Converting from Decimal to Hex

- Convert to binary first and then convert to hex
- Or, convert directly from decimal to hex by repeated division, keeping track of the remainders

**Ex. Convert  $45_{10}$  to hex**

32   16   8   4   2   1

1   0   1   1   0   1    $32 + 8 + 4 + 1 = 45$

$45_{10} = 0010\ 1101_2 = 2D_{16}$

**Ex. Convert  $629_{10}$  to hex**

512   256   128   64   32   16   8   4   2   1

1   0   0   1   1   1   0   1   0   1

$629_{10} = 512+64+32+16+4+1 = 0010\ 0111\ 0101_2 = 275_{16}$

# Converting from Hex to Decimal

---

- Convert from hex to binary and then to decimal
- Or, convert directly from hex to decimal by summing the weight of all digits

**Ex.  $6B2_{16} = 0110\ 1011\ 0010_2$**

<u>1024</u>	<u>512</u>	<u>256</u>	<u>128</u>	<u>64</u>	<u>32</u>	<u>16</u>	<u>8</u>	<u>4</u>	<u>2</u>	<u>1</u>
1	1	0	1	0	1	1	0	0	1	0
<b><math>1024 + 512 + 128 + 32 + 16 + 2 = 1714_{10}</math></b>										

# Addition of Hex Numbers

---

- Adding the digits together from the least significant digits
  - If the result is less than 16, write that digit as the sum for that position
  - If it is greater than 16, subtract 16 from it to get the digit and carry 1 to the next digit

**Ex. Perform hex addition: 23D9 + 94BE**

<b>23D9</b>	<b>LSD: 9 + 14 = 23</b>	<b>23 - 16 = 7 w/ carry</b>
<b>+ 94BE</b>	<b>1 + 13 + 11 = 25</b>	<b>25 - 16 = 9 w/ carry</b>
<b>B897</b>	<b>1 + 3 + 4 = 8</b>	
	<b>MSD: 2 + 9 = B</b>	



# Subtraction of Hex Numbers

---

- If the second digit is greater than the first, borrow 16 from the preceding digit

**Ex. Perform hex subtraction: 59F – 2B8**

**59F**  
**– 2B8**  
**2E7**

**LSD: 15 – 8 = 7**

**9 + 16 – 11 = 14 = E<sub>16</sub>**

**5 – 1 – 2 = 2**

# ASCII Code

- The ASCII (pronounced “ask-E”) code assigns binary patterns for
  - Numbers 0 to 9
  - All the letters of English alphabet, uppercase and lowercase
  - Many control codes and punctuation marks
- The ASCII system uses 7 bits to represent each code

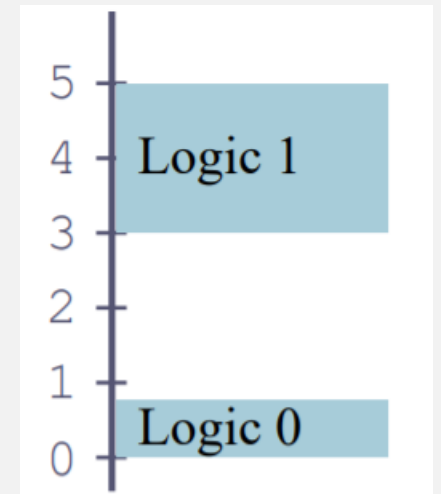
Selected ASCII codes		<i>Hex</i>	<i>Symbol</i>	<i>Hex</i>	<i>Symbol</i>
		41	A	61	a
		42	B	62	b
		43	C	63	c
		44	D	64	d
		...	...	...	...
		59	Y	79	y
		5A	Z	7A	z

# Digital Primer

---

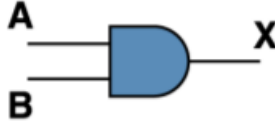
## Binary Logic:

- Two voltage levels can be represented as the two digits 0 and 1
- Signals in digital electronics have two distinct voltage levels with built-in tolerances for variations in the voltage
- A valid digital signal should be within either of the two shaded areas

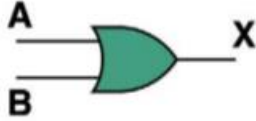


# Logic Gates

- **AND gate:**

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \cdot B$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	0	0	1	0	1	0	0	1	1	1
A	B	X															
0	0	0															
0	1	0															
1	0	0															
1	1	1															

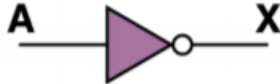
- **OR gate:**

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A + B$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	1
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	1															

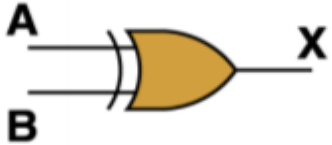
# Logic Gates (cont')

- Tri-state buffer

- Inverter:

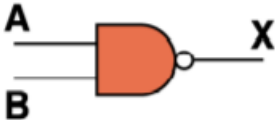
Boolean Expression	Logic Diagram Symbol	Truth Table						
$X = A'$		<table><tr><th>A</th><th>X</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	A	X	0	1	1	0
A	X							
0	1							
1	0							

- XOR gate:

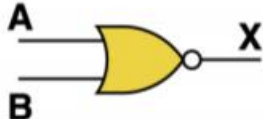
Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = A \oplus B$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	0	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	0															
0	1	1															
1	0	1															
1	1	0															

# Logic Gates (cont')

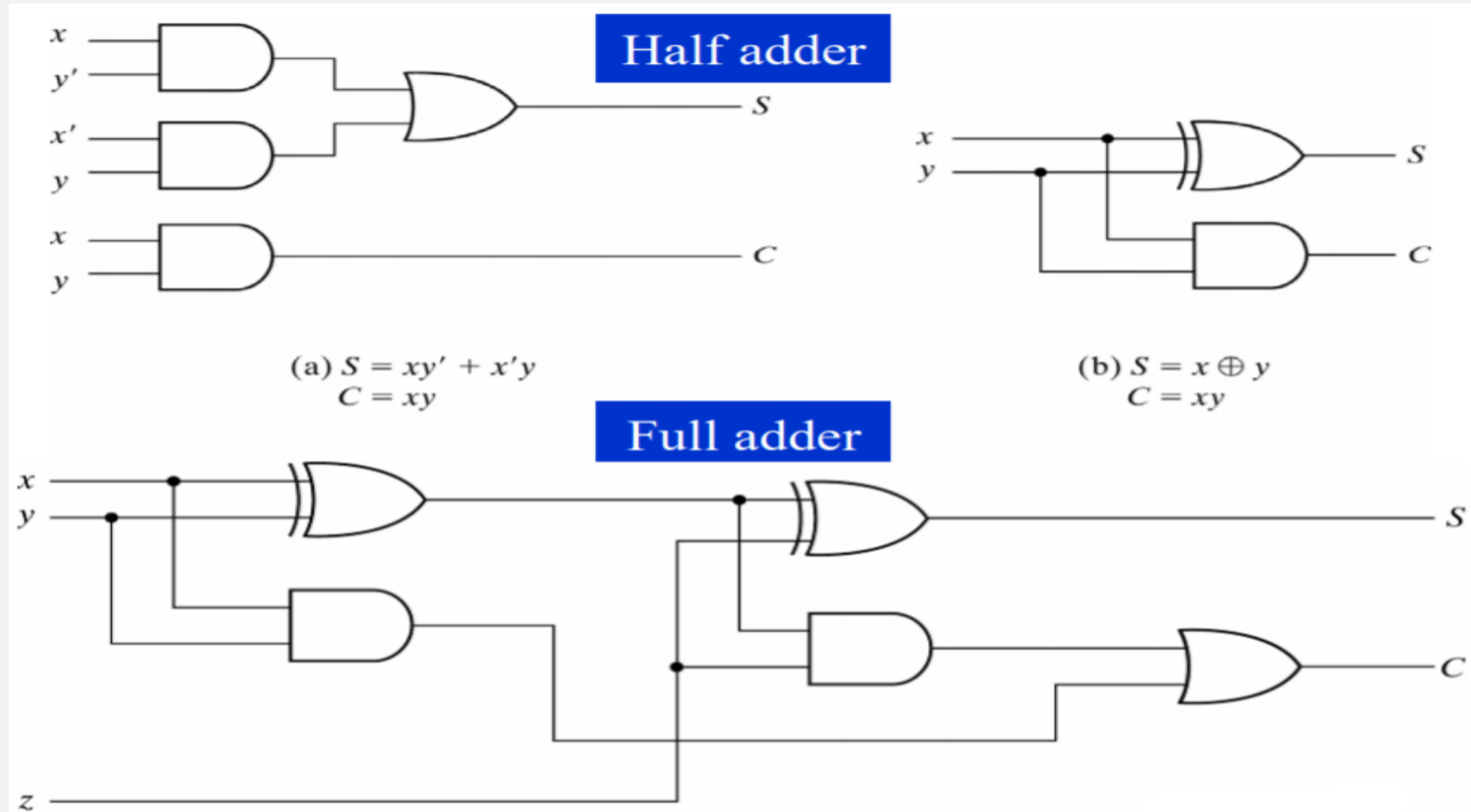
- NAND gate:

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = (A \cdot B)'$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	1	0	1	1	1	0	1	1	1	0
A	B	X															
0	0	1															
0	1	1															
1	0	1															
1	1	0															

- NOR gate:

Boolean Expression	Logic Diagram Symbol	Truth Table															
$X = (A + B)'$		<table><tr><th>A</th><th>B</th><th>X</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	A	B	X	0	0	1	0	1	0	1	0	0	1	1	0
A	B	X															
0	0	1															
0	1	0															
1	0	0															
1	1	0															

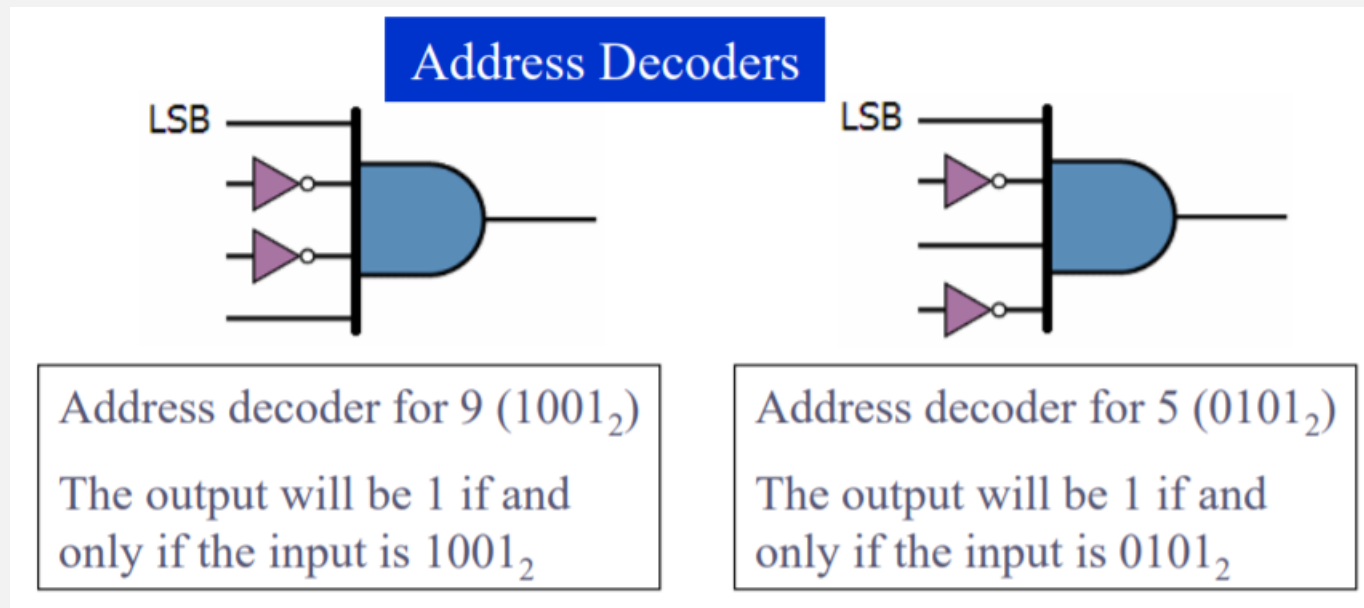
# Logic Design Using Gates



# Logic Design Using Gates(cont')

- **Decoders:**

- Decoders are widely used for address decoding in computer design

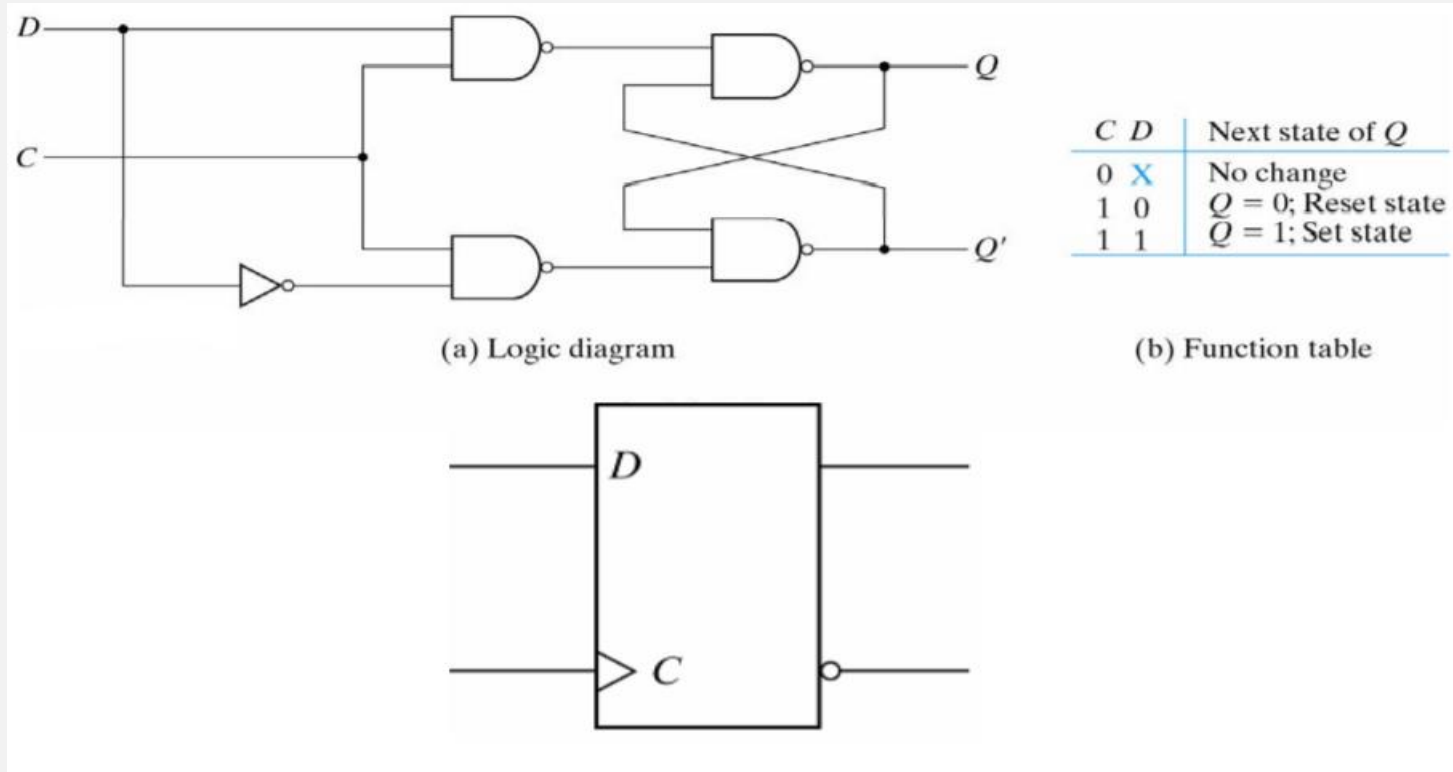




# Logic Design Using Gates(cont')

- **Flip-flops:**

- Flip-flops are frequently used to store data



# Semiconductor Memory

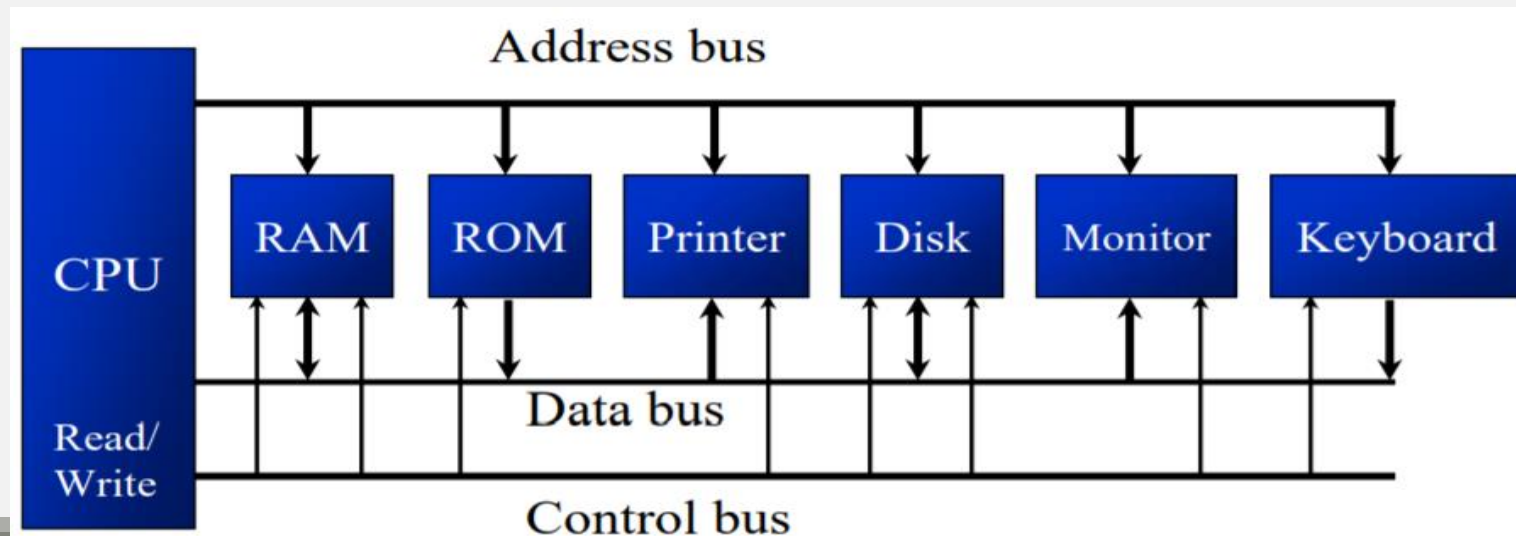
---

## Important Terminology:

- The unit of data size:
  - Bit : a binary digit that can have the value 0 or 1
  - Byte : 8 bits
  - Nibble : half of a byte, or 4 bits
  - Word : two bytes, or 16 bits
- The terms used to describe amounts of memory in IBM PCs and compatibles:
  - Kilobyte (K):  $2^{10}$  bytes
  - Megabyte (M) :  $2^{20}$  bytes, over 1 million
  - Gigabyte (G) :  $2^{30}$  bytes, over 1 billion
  - Terabyte (T) :  $2^{40}$  bytes, over 1 trillion

# Internal Organization of Computers

- The CPU is connected to memory and I/O through strips of wire called a bus
- Carries information from place to place:
  - Address bus
  - Data bus
  - Control bus



# Internal Organization of Computers (cont')

---

- Address bus:
  - For a device (memory or I/O) to be recognized by the CPU, it must be assigned an address
  - The address assigned to a given device must be unique
  - The CPU puts the address on the address bus, and the decoding circuitry finds the device
- Data bus:
  - The CPU either gets data from the device or sends data to it
- Control bus:
  - Provides read or write signals to the device to indicate if the CPU is asking for information or sending it information

# More about Data Bus

---

- The more data buses available, the better the CPU
  - Think of data buses as highway lanes
- More data buses mean a more expensive CPU and computer
  - The average size of data buses in CPUs varies between 8 and 64
- Data buses are bidirectional
  - To receive or send data
- The processing power of a computer is related to the size of its buses

# More about Address Bus

---

- The more address buses available, the larger the number of devices that can be addressed
- The number of locations with which a CPU can communicate is always equal to,  $2^x$  where x is the address lines, regardless of the size of the data bus
  - ex. a CPU with 24 address lines and 16 data lines can provide a total of  $2^{24}$  or 16M bytes of addressable memory
  - Each location can have a maximum of 1 byte of data, since all general-purpose CPUs are byte addressable
- The address bus is unidirectional

# CPU's Relation to RAM and ROM

---

- For the CPU to process information, the data must be stored in RAM or ROM, which are referred to as primary memory
- ROM provides information that is fixed and permanent
  - Tables or initialization program
- RAM stores information that is not permanent and can change with time
  - Various versions of OS and application packages
- CPU gets information to be processed
  - first from RAM (or ROM)
  - if it is not there, then seeks it from a mass storage device, called secondary memory, and transfers the information to RAM

# Semiconductor Memories:

---

- ROM
  - PROM
  - EPROM
  - EEPROM
  - Flash memory
  - EPROM
  - Mask ROM
- RAM
  - SRAM
  - NV-RAM
  - DRAM
- Characteristics:
  - Memory Capacity
  - Memory Organization
  - Speed



# CPU Architecture

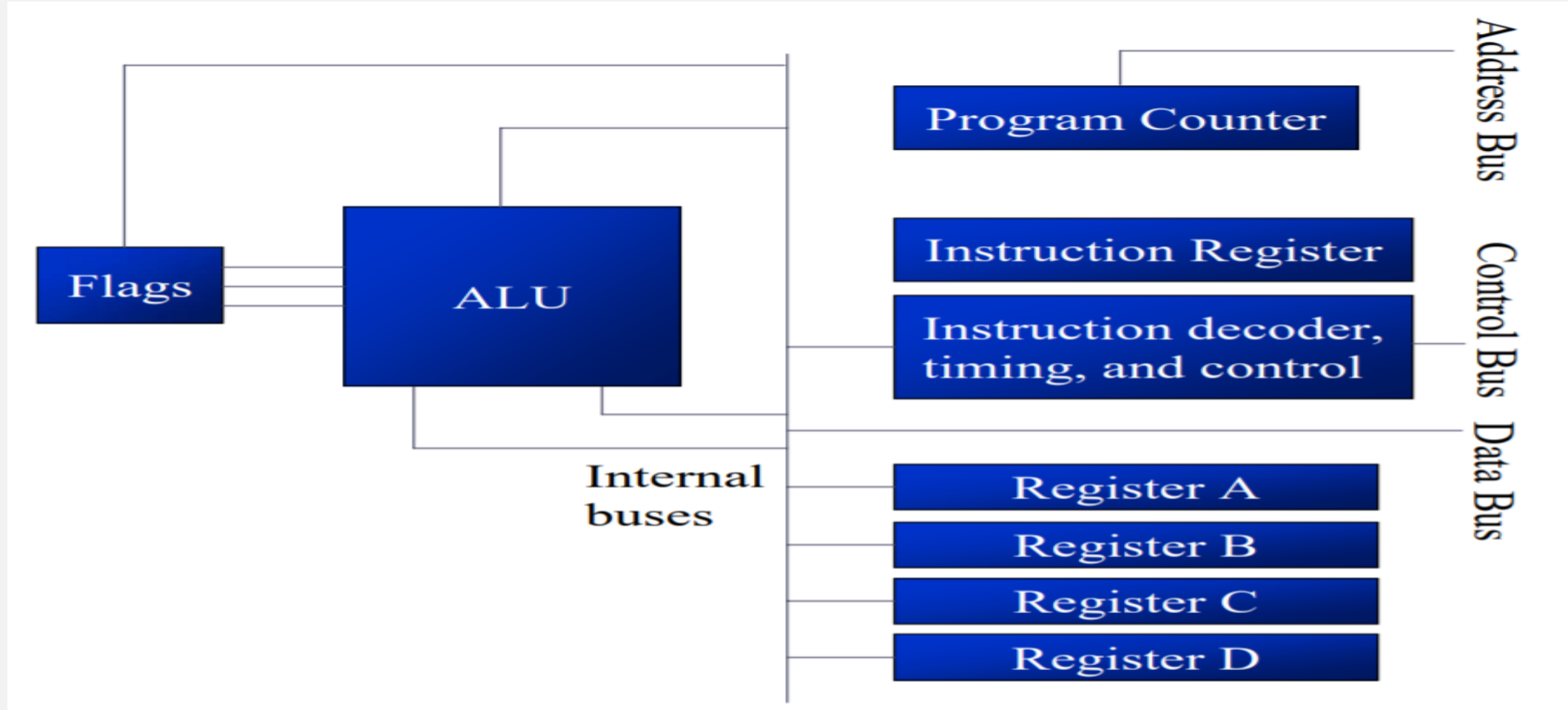


Figure: Internal Block Diagram of a CPU

# CPU Architecture

---

## Registers:

- The CPU uses registers to store information temporarily
  - Values to be processed
  - Address of value to be fetched from memory
- In general, the more and bigger the registers, the better the CPU
  - Registers can be 8, 16, 32 or 64-bit
  - The disadvantage of more and bigger registers is the increased cost of such a CPU

# CPU Architecture

---

## **ALU (arithmetic/logic unit):**

- Performs arithmetic functions such as add, subtract, multiply, and divide, and logic functions such as AND, OR, and NOT

## **Program counter:**

- Points to the address of the next instruction to be executed
  - As each instruction is executed, the program counter is incremented to point to the address of the next instruction to be executed

## **Instruction decoder:**

- Interprets the instruction fetched into the CPU
  - A CPU capable of understanding more instructions requires more transistors to design

## An example:

**Ex. A CPU has registers A, B, C, and D and it has an 8-bit data bus and a 16-bit address bus. The CPU can access memory from addresses 0000 to FFFFH**

**Assume that the code for the CPU to move a value to register A is B0H and the code for adding a value to register A is 04H**

**The action to be performed by the CPU is to put 21H into register A, and then add to register A values 42H and 12H**

Explaining Internal Working of Computers through an example:

<b><i>Action</i></b>	<b><i>Code</i></b>	<b><i>Data</i></b>
<b>Move value 21H into reg. A</b>	<b>B0H</b>	<b>21H</b>
<b>Add value 42H to reg. A</b>	<b>04H</b>	<b>42H</b>
<b>Add value 12H to reg. A</b>	<b>04H</b>	<b>12H</b>

<b><i>Mem. addr.</i></b>	<b><i>Contents of memory address</i></b>
<b>1400</b>	<b>(B0) code for moving a value to register A</b>
<b>1401</b>	<b>(21) value to be moved</b>
<b>1402</b>	<b>(04) code for adding a value to register A</b>
<b>1403</b>	<b>(42) value to be added</b>
<b>1404</b>	<b>(04) code for adding a value to register A</b>
<b>1405</b>	<b>(12) value to be added</b>
<b>1406</b>	<b>(F4) code for halt</b>

Explaining Internal Working of Computers through an example:

**The actions performed by CPU are as follows:**

- 1. The program counter is set to the value 1400H, indicating the address of the first instruction code to be executed**
- 2.**
  - **The CPU puts 1400H on address bus and sends it out**
    - **The memory circuitry finds the location**
  - **The CPU activates the READ signal, indicating to memory that it wants the byte at location 1400H**
    - **This causes the contents of memory location 1400H, which is B0, to be put on the data bus and brought into the CPU**



Explaining Internal Working of Computers through an example:

**3.**

- **The CPU decodes the instruction B0**
- **The CPU commands its controller circuitry to bring into register A of the CPU the byte in the next memory location**
  - **The value 21H goes into register A**
- **The program counter points to the address of the next instruction to be executed, which is 1402H**
  - **Address 1402 is sent out on the address bus to fetch the next instruction**

## Explaining Internal Working of Computers through an example:

**4.**

- **From memory location 1402H it fetches code 04H**
- **After decoding, the CPU knows that it must add to the contents of register A the byte sitting at the next address (1403)**
- **After the CPU brings the value (42H), it provides the contents of register A along with this value to the ALU to perform the addition**
  - **It then takes the result of the addition from the ALU's output and puts it in register A**
  - **The program counter becomes 1404, the address of the next instruction**



Explaining Internal Working of Computers through an example:

**5.**

- **Address 1404H is put on the address bus and the code is fetched into the CPU, decoded, and executed**
  - **This code is again adding a value to register A**
  - **The program counter is updated to 1406H**

**6.**

- **The contents of address 1406 are fetched in and executed**
- **This HALT instruction tells the CPU to stop incrementing the program counter and asking for the next instruction**

Ex. A CPU has registers A, B, C, and D and it has an 8-bit data bus and a 16-bit address bus. The CPU can access memory from addresses 0000 to FFFFH

Assume that the code for the CPU to move a value to register A is B0H and the code for adding a value to register A is 04H

The action to be performed by the CPU is to put 21H into register A, and then add to register A values 42H and 12H. The actions performed by CPU are as follows:

1. The program counter is set to the value 1400H, indicating the address of the first instruction code to be executed
2.
  - The CPU puts 1400H on address bus and sends it out
    - The memory circuitry finds the location
  - The CPU activates the READ signal, indicating to memory that it wants the byte at location 1400H
    - This causes the contents of memory location 1400H, which is B0, to be put on the data bus and brought into the CPU

Action	Code	Data 5.
Move value 21H into reg. A	B0H	21H
Add value 42H to reg. A	04H	42H
Add value 12H to reg. A	04H	12H

Mem. addr.	Contents of memory address	
1400	(B0) code for moving a value to register A	6.
1401	(21) value to be moved	
1402	(04) code for adding a value to register A	
1403	(42) value to be added	
1404	(04) code for adding a value to register A	
1405	(12) value to be added	
1406	(F4) code for halt	

3.

- The CPU decodes the instruction B0
- The CPU commands its controller circuitry to bring into register A of the CPU the byte in the next memory location
  - The value 21H goes into register A
- The program counter points to the address of the next instruction to be executed, which is 1402H
  - Address 1402 is sent out on the address bus to fetch the next instruction

➤ Address 1404H is put on the address bus and the code is fetched into the CPU, decoded, and executed

- This code is again adding a value to register A
- The program counter is updated to 1406H

➤ The contents of address 1406 are fetched in and executed

➤ This HALT instruction tells the CPU to stop incrementing the program counter and asking for the next instruction

4.

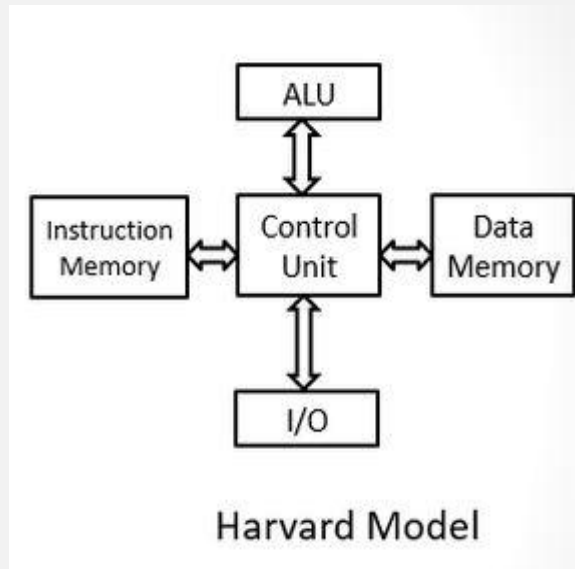
- From memory location 1402H it fetches code 04H
- After decoding, the CPU knows that it must add to the contents of register A the byte sitting at the next address (1403)
  - After the CPU brings the value (42H), it provides the contents of register A along with this value to the ALU to perform the addition
    - It then takes the result of the addition from the ALU's output and puts it in register A
    - The program counter becomes 1404, the address of the next instruction

**Figure:** Explaining Internal Working of Computers through an example

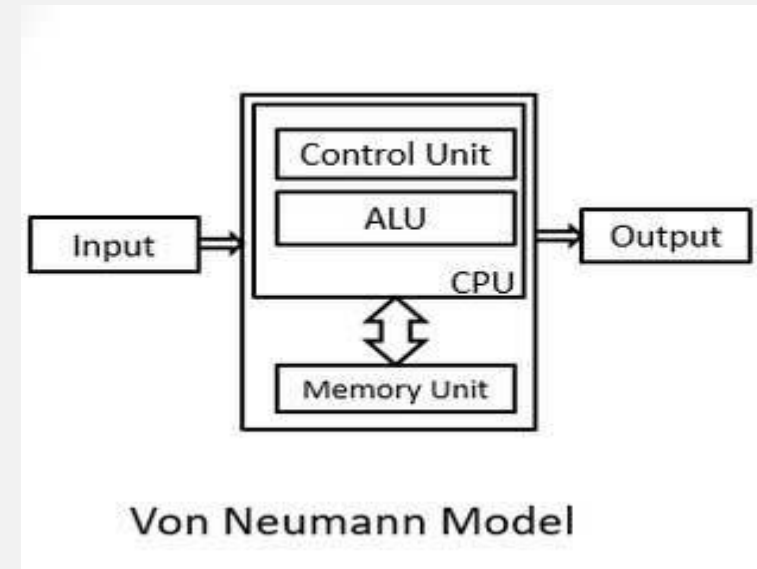


# Harvard and Von Neumann Architectures

---



Developed at Harvard University

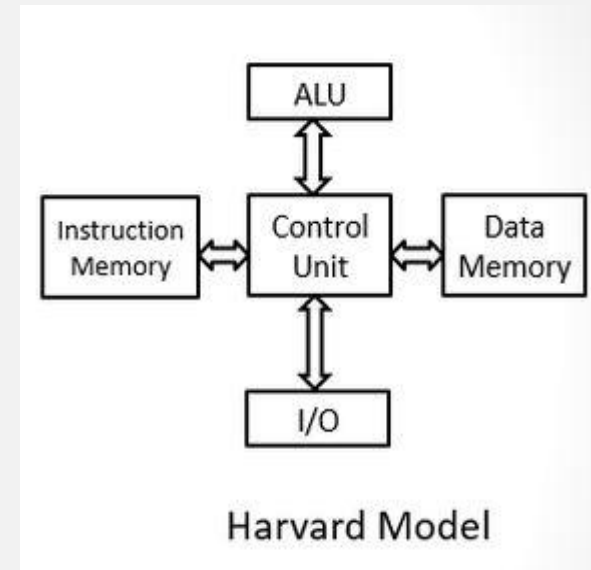


Developed at Princeton University

# Harvard Architecture

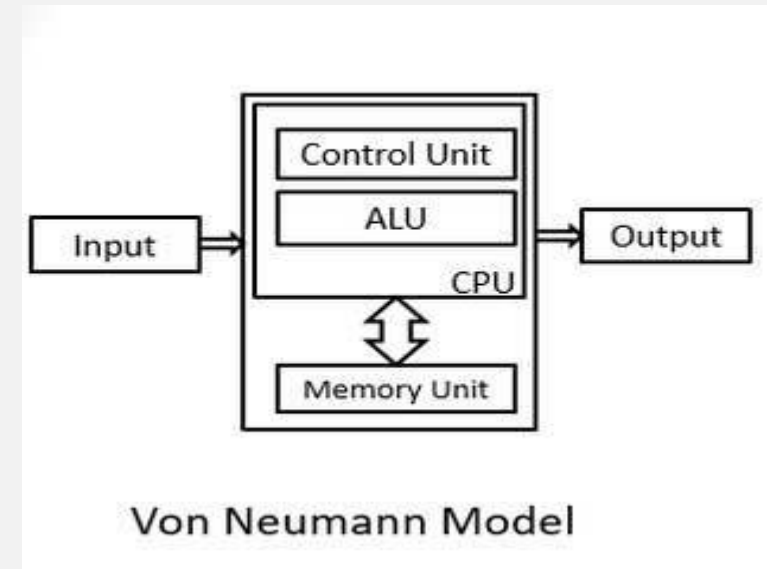
---

- Uses separate buses for accessing the code and data and it needs 4 set of buses to implement this.
- This architecture is faster but complex and expensive.
- The system needs comparatively much more wire traces on motherboard than Von Neumann architecture.
- Some microcontrollers use this architecture internally.



# Von Neumann Architecture

- Uses same bus for accessing both the code and data.
- For Von Neumann computers, the process of accessing code or data could cause them to get in each other's way thus slowing the process of CPU, as each have to wait for the other to finish fetching.
- The system needs comparatively less wire traces on motherboard in this architecture
- If a microcontroller needs external memory for code and data space, it uses Von Neumann architecture.



---

Have a Wonderful Journey with **Microcontroller** !