Module 1. Computer Fundamentals

Prof. Datta Deshmukh
Assistant Professor
K.J. Somaiya Institute of Technology , Sion

Decimal, Binary, Octal Number System

Decimal number system:

- Base/Radix = 10
- 10 distinct elements = {0,1,2,3,4,5,6,7,8,9}
- Digit position is weighted by power of '10'
- Example: $(235.7)_{10} = (2\times10^{2}) + (3\times10^{1}) + (5\times10^{0}) + (7\times10^{-1})$

Binary number system:

- Base/Radix = 2, only 2 elements {0,1}
- Digit position is weighted by power of '2'
- Example: $(110.01)_{2}=(1\times2^{2}) + (1\times2^{1}) + (0\times2^{0}) + (0\times2^{-1}) + (1\times2^{-2})$

Octal number system:

- Base/Radix = 8, hence total 8 elements in the system {0,1,2,3,4,5,6,7}
- Digit position is weighted by power of '8'
- Example: $(54.36)_{8} = (5\times8^{1}) + (4\times8^{0}) + (3\times8^{-1}) + (6\times8^{-2})$

Hexadecimal, Base-3 system

Hexadecimal number system:

- Base/Radix = 16, hence total 16 elements in the system {0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F}: A to F corresponds to 10 to 15
- Digit position is weighted by power of '16'
- Example: $(A5.7C)_H = (10\times16^1) + (5\times16^0) + (7\times16^{-1}) + (12\times16^{-2})$

Base-3 number system:

- Base/Radix = 3, hence only 3 elements {0,1,2}
- Digit position is weighted by power of '3'
- Example: $(120.02)_3=(1\times3^2) + (2\times3^1) + (0\times3^0) + (0\times3^{-1}) + (2\times3^{-2})$

Decimal to Binary conversion: An example $(57.3125)_{10}$

For integer $(57)_{10}$

$57 \div 2 = 28$	remainder **1**
$28 \div 2 = 14$	remainder **0**
$14 \div 2 = 7$	remainder **0**
$7 \div 2 = 3$	remainder **1**
$3 \div 2 = 1$	remainder **1**
$1 \div 2 = 0$	remainder **1**

For fractional part $(0.3125)_{10}$

$0.3125 \times 2 = 0.625$	integer **0**
$0.625 \times 2 = 1.25$	integer **1**
$0.25 \times 2 = 0.5$	integer **0**
$0.5 \times 2 = 1.0$	integer **1**

Therefore (57.3125) = (111001.0101)

Conversion of Integer Decimal (Base-10) to Base-N system

1. **Divide** the decimal number by 'N'
2. Write the integer **quotient & remainder** on right side
3. Use the **new quotient** & perform **step-1 & 2 repeatedly**, till the quotient becomes '0'
4. Collect all the **remainders in "Bottom-up"** manner
5. Rewrite result: **$(\text{Number})_{10} = (\text{Number})_N$**

Decimal (Base-10) to Binary (Base-2)

For integer $(57)_{10}$

$57 \div 2 = 28$	remainder 1
$28 \div 2 = 14$	remainder 0
$14 \div 2 = 7$	remainder 0
$7 \div 2 = 3$	remainder 1
$3 \div 2 = 1$	remainder 1
$1 \div 2 = 0$	remainder 1

Hence $(57)_{10} = (111001)_2$

Fractional Decimal to Fractional Base-N conversion

1. **Multiply** the fractional part by 'N'
2. Write the Product of this multiplication & has 2 parts:
   - **Integer part** to the left side of decimal point
   - **Fractional part** to the right side of decimal point
3. Use the **new fractional part** & perform **step-1 & 2 repeatedly**, till the product becomes '0'
4. Collect all **integers in "Top-down"** manner
5. Rewrite the result as: $(\text{Number})_{10} = (\text{Number})_N$

Note: For real decimal (containing integer & fractional part), results to be concatenated

Hence, $(57.3125)_{10} = (111001.0101)_2$

$(0.3125)_{10}$ into Binary (Base-2)

For fractional part $(0.3125)_{10}$

$0.3125 \times 2 = 0.625$ integer 0

$0.625 \times 2 = 1.25$ integer 1

$0.25 \times 2 = 0.5$ integer 0

$0.5 \times 2 = 1.0$ integer 1

Hence, $(0.3125)_{10} = (0.0101)_2$

Decimal to Octal conversion: An example $(169.28)_{10}$

For integer $(169)_{10}$

$169 \div 8 = 21$ remainder **1**

$21 \div 8 = 2$ remainder **5**

$2 \div 8 = 0$ remainder **2**

For fractional part $(0.28)_{10}$

$0.28 \times 8 = \mathbf{2}.24$ integer **2**

$0.24 \times 8 = \mathbf{1}.92$ integer **1**

$0.92 \times 8 = \mathbf{7}.36$ integer **7**

$0.36 \times 8 = \mathbf{2}.88$ integer **2**

$0.88 \times 8 = \mathbf{7}.04$ integer **7**

$0.04 \times 8 = \mathbf{0}.32$ integer **0**

Operation is unending,hence stop after 6 to 7 steps

Hence. $(169.28)_{10} = (251.217270)_8$

Decimal to Hexadecimal conversion: Example $(379.02)_{10}$

For integer $(379)_{10}$

$379 \div 16 = 23$ remainder **11= B**

$23 \div 16 = 1$ remainder **7**

$1 \div 16 = 0$ remainder **1**

For fractional part $(0.02)_{10}$

$0.02 \times 16 = 0.32$ integer **0**

$0.32 \times 16 = 5.12$ integer **5**

$0.12 \times 16 = 1.92$ integer **1**

$0.92 \times 16 = \mathbf{14}.72$ integer **14 = E**

$0.72 \times 16 = \mathbf{11}.52$ integer **11 = B**

$0.52 \times 16 = 8.32$ integer **8**

Operation is recurring hereafter, hence stop

Therefore, $(379.02)_{10} = (17B.051EB8)_H$

Decimal to Base-3 number conversion: Example $(33.45)_{10}$

For integer $(33)_{10}$

$33 \div 3 = 11$ remainder **0**

$11 \div 3 = 3$ remainder **2**

$3 \div 3 = 1$ remainder **0**

$1 \div 3 = 0$ remainder **1**

For fractional part $(0.45)_{10}$

$0.45 \times 3 = 1.35$ integer **1**

$0.35 \times 3 = 1.05$ integer **1**

$0.05 \times 3 = 0.15$ integer **0**

$0.15 \times 3 = 0.45$ integer **0**

Here, the sequence is recurring, hence stop

Hence, $(33.45) = (1020.1100)$

Conversion of Base-N system to Decimal number system

- Simply add the weights together
- Rewrite the decimal equivalent number

Conversion of $(120.02)_3$ into decimal

$$(120.02)_3 = (1\times3^2) + (2\times3^1) + (0\times3^0) + (0\times3^{-1}) + (2\times3^{-2})$$
$$= 9 + 6 + 0 + 0 + 0.22$$
$$= (15.22)_{10}$$

Conversion of $(A5.7C)_H$ into decimal

$$(A5.7C)_H = (10\times16^1) + (5\times16^0) + (7\times16^{-1}) + (12\times16^{-2})$$
$$= 160 + 5 + 0.44 + 0.05$$
$$= (165.49)_{10}$$

Conversion of Base-M to Base-N number system (M & N ≠ 10)

Conversion of Binary $(1101.01)_2$ to Base-5 system

Step 1: **Binary to decimal**. Hence $(1101.01)_2 = (13.25)_{10}$

Step 2: **Decimal to Base-5**. Perform by successive division by 5 to 13, & successive multiplication by 5 to 0.25

Hence, $(1101.01)_2 = (13.25)_{10} = (23.111111)_5$

Conversion of Binary (Base-2) to Octal (Base-8)

Above method can be used, as it is **"Universal"**

But, follow the shortcut as **Octal = Base '8' = 2^3**

View binary in groups of **3 bit** & express the groups in **octal by adding {4,2,1} weightages** where logic'1' is present

$(10110.01)_2 = (010\ \ 110\ .\ 010)_2$:Complete groups of 3 bits each

$(2\ \ 6\ .\ 2)_8$

Base-M to Base-N conversion contd......

Conversion of Octal (Base-8) to Binary (Base-2)

“Universal” method is: **Octal to Decimal to Binary**

But, follow the shortcut method as **Octal = Base '8' = 2^3**

Expand each **Octal digit** in group of three binary bits in {4:2:1} manner: $(1 \quad 7 \quad 5 \;.\; 2 \quad 6)_8$

$$(001\ 111\ 101\ .\ 010\ 110)_2$$

Remove the redundant '0's and rewrite the result.

Hence, $\mathbf{(1111101.01011)_2}$

Base-M to Base-N conversion contd……

Conversion of Binary (Base-2) to Hex (Base-16)

Hexadecimal Base = 16 = 2^4

View binary in **groups of 4 bit** & express the groups in Hex by adding **{8,4,2,1} weightages** where logic '1' is present

Example: Convert $(1010111001.011011)_2$ to Hex

0010 1011 1001 . 0110 1100

Hex Result = (**2** **B** **9** **.** **6** **C** $)_H$

Hex to Binary conversion

Expand the hex digit in 4 bit binary using {8,4,2,1} weightages

$(2AC.78)_H = (0010\ 1010\ 1100.\ 0111\ 1000)_2$

Hence, binary after removing redundant '0's:

(2AC.78) = (10 1010 1100. 0111 1)

BCD (Binary Coded Decimal) system

- Decimal digits expressed in 4 bit binary form using {8,4,2,1} weightages

Binary: Operating number system of Digital computers

Decimal: Natural number system that human understands easily

- Advantages of both number system are collected together in BCD
- BCD is also **referred as 8421-BCD (as it is weighted)**

Decimal	BCD	Decimal	BCD
0	**0000**	**1**	**0001**
2	**0010**	**3**	**0011**
4	**0100**	**5**	**0101**
6	**0110**	**7**	**0111**

BCD vs Excess-3 code system

- Excess-3 (XS-3) is extension of BCD: adds $(3)_{10}$ to the BCD value
- BCD is weighted (8421), but XS-3 is **Non-weighted, Inverted Reflected code**

Decimal	8421-BCD	XS-3 code
0	0000	**0011**
1	0001	**0100**
2	0010	**0101**
3	0011	**0110**
4	0100	**0111**
5	0101	**1000**
6	0110	**1001**
7	0111	**1010**
8	1000	**1011**
9	1001	**1100**

Gray code

- **Non-weighted, n-bit code** (not restricted to only 4 bits like BCD or XS-3 code)
- **Reflected code** except the MSB (left most bit)
- **Differs only at 1 bit position** from the previous combination

Decimal to 2 bit Binary to Gray code conversion:

Decimal	Binary	Gray code
0	00	00
1	01	01
2	10	11
3	11	10

Successive Gray code entries differ only at 1 bit:

00 → 01 → 11 → 10

Note: Above reflect plane, MSB = 0 & below plane, MSB = 1 (lower bits are the true reflections about the plane)

2 bit to 3 bit Gray code conversion

Step 1: Write 2 bit Gray code in sequence & reflect all entries below

00
01
11
10

Reflect plane

10
11
01
00

Step 2: Append **MSB = 0 above plane**, **'1' below** & rewrite the 3 bit Gray code

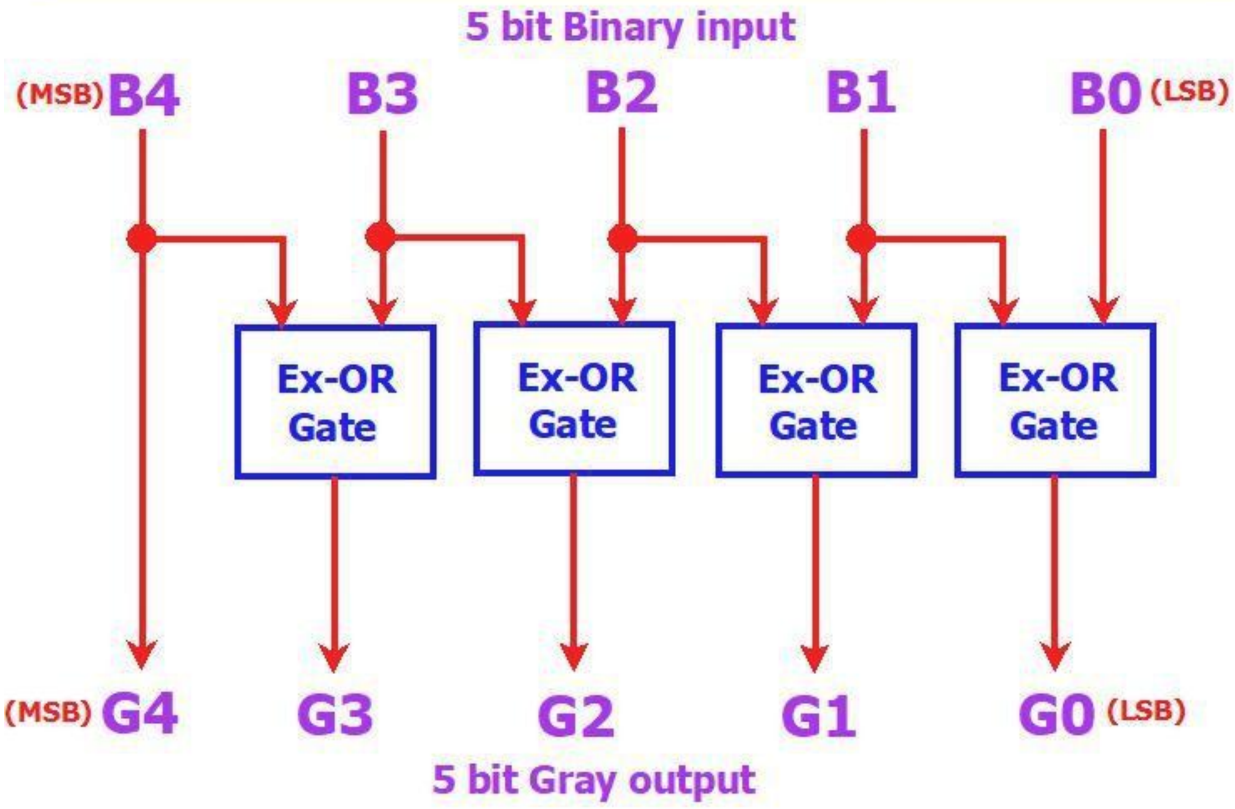
3 bit Gray

000
001
011
010

110
111
101
100

N- bit Binary to Gray code conversion (using Ex-OR gate)

5 bit Binary to Gray code conversion setup



Note: For equal inputs, output of Ex-OR gate is '0'

Consider binary input: 10010

Gray output in bit by bit is:

G4 = B4 = 1

G3 = 1 (as '1' & '0' are Ex-OR i/ps)

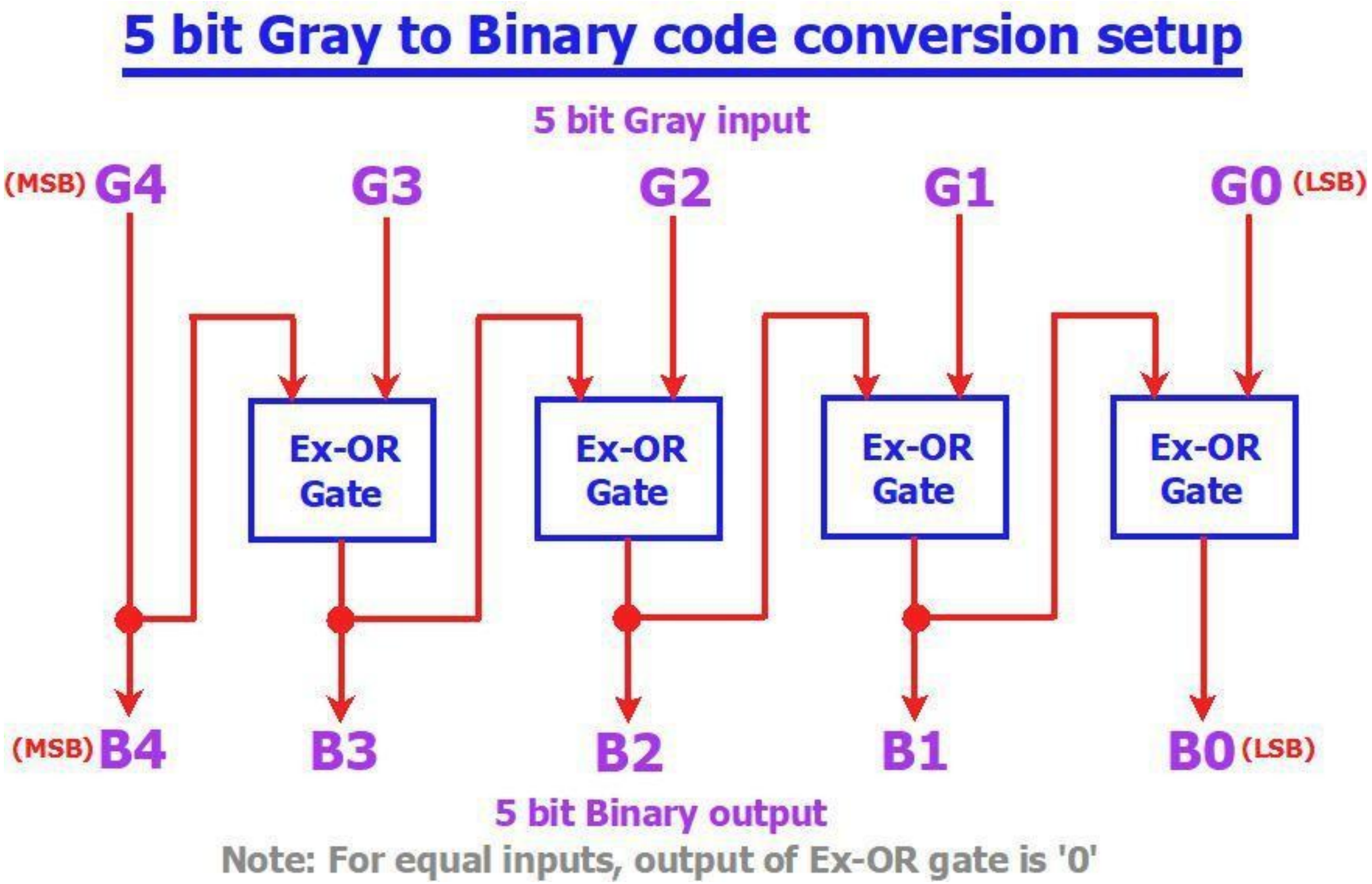
G2 = 0 (as '0' & '0' are Ex-OR i/ps)

G1 = 1 (as '0' & '1' are Ex-OR i/ps)

G0 = 1 (as '1' & '0' are Ex-OR i/ps)

Hence, Gray output = 11011

N-bit Gray to Binary code conversion (using Ex-OR gate)



Consider Gray input: 10101

Binary output in bit by bit is:

B4 = G4 = 1

B3 = 1 (as '1' & '0' are Ex-OR i/ps)

B2 = 0 (as '1' & '1' are Ex-OR i/ps)

B1 = 0 (as '0' & '0' are Ex-OR i/ps)

B0 = 1 (as '0' & '1' are Ex-OR i/ps)

Hence, Binary output = 11001

Tricky problem on number conversions

Problem:

The gray code is "10010. 01101". Convert it into Octal, BCD & Excess-3 code.

Solution hints:

Octal: First convert gray code into Binary without consideration of the binary point, but inserting the binary point at last & then this binary converted into Octal

Binary: $(11100.\ 01001)_2$

Octal: $(34.\ 22)_8$

BCD: Binary or octal obtained is converted into decimal & then into BCD (Octal is preferred over binary as it is fast)

Decimal: $(28.\ 28)_{10}$

BCD: **0010 1000 . 0010 1000**

Excess-3: Decimal obtained is converted into the Excess-3

Excess-3: **0101 1011 . 0101 1011**

All in one problem on number conversions

Problem statement:

Convert the decimal number $(163.13)_{10}$ into Binary, Octal, Hexadecimal, Ternary, BCD, Excess-3 & Gray code.

Solution hints:

Decimal to Binary: Using "Double-Dabble method" on decimal number

Octal: Using short cut method (making the groups of 3 binary bits & expressing them in octal)

Hexadecimal: Making groups of 4 binary bits & expressing them in Hex

Ternary: Using "Ternary-Dabble method" on given decimal number

BCD: Expressing each decimal digit of given number in 4 bit binary form

Excess-3: Add '3' to each decimal digit & express sum in 4 bit binary form

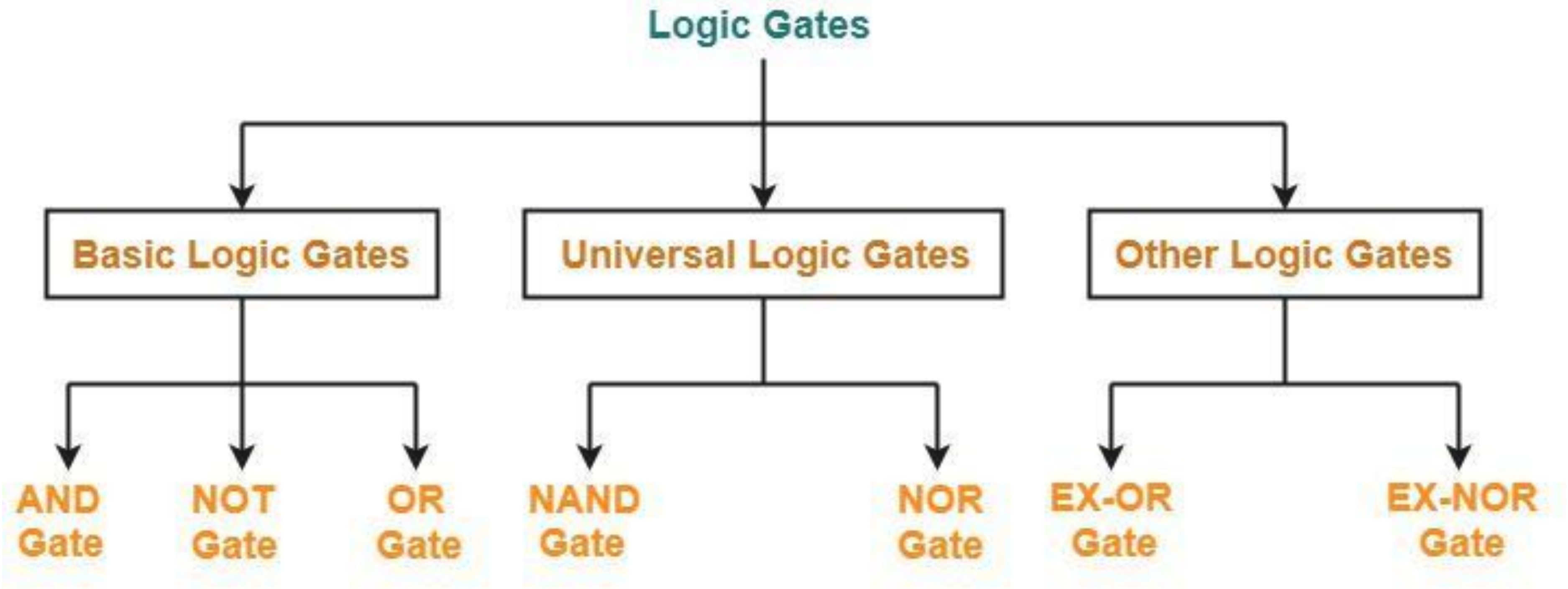
Gray code: Converting the binary directly into Gray code using "Binary to Gray code conversion logic" explained earlier

Solution to the previous problem

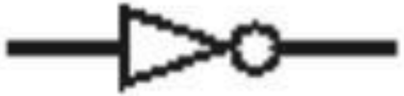
Given decimal number: $(163.13)_{10}$

- **Binary**: $(10100011.\ 00100001)_2$
- **Octal**: $(243.\ 102)_8$
- **Hex**: $(A3.\ 21)_H$
- **Ternary**: $(20001.\ 0101112)_3$
- **BCD**: 0001 0110 0011. 0001 0011
- **Excess-3**: 0100 1001 0110. 0100 0110
- **Gray**: 11110010. 10110001

Classification of the Logic gates

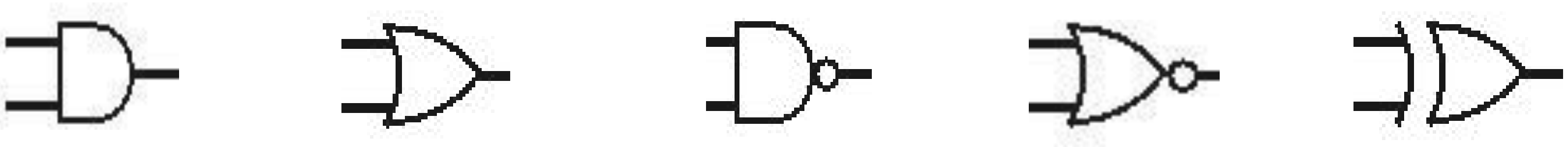


NOT

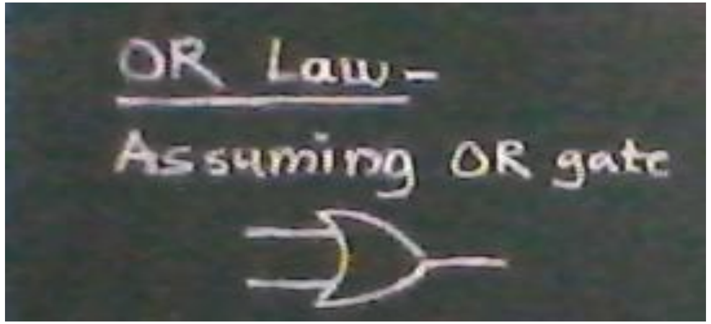


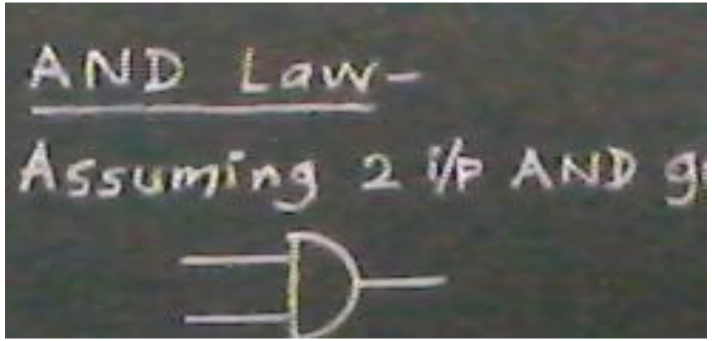
Summary of all 2-input gates

Inputs A	Inputs B	AND	OR	NAND	NOR	Ex- OR
0	0	0	0	1	1	0
0	1	0	1	1	0	1
1	0	0	1	1	0	1
1	1	1	1	0	0	0

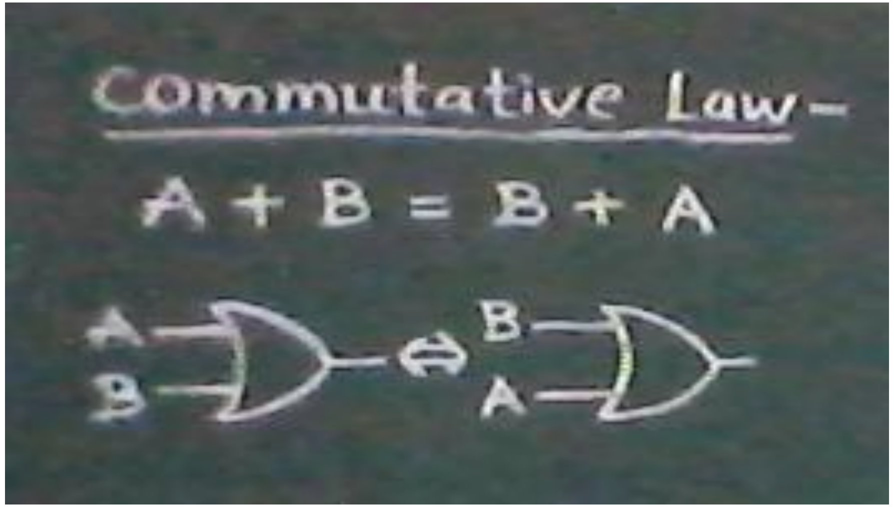


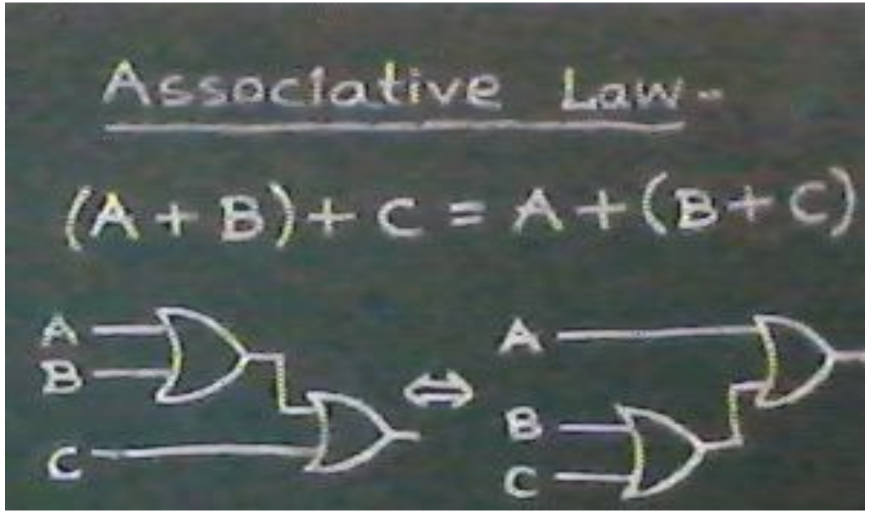
Boolean Algebra: OR & AND Laws

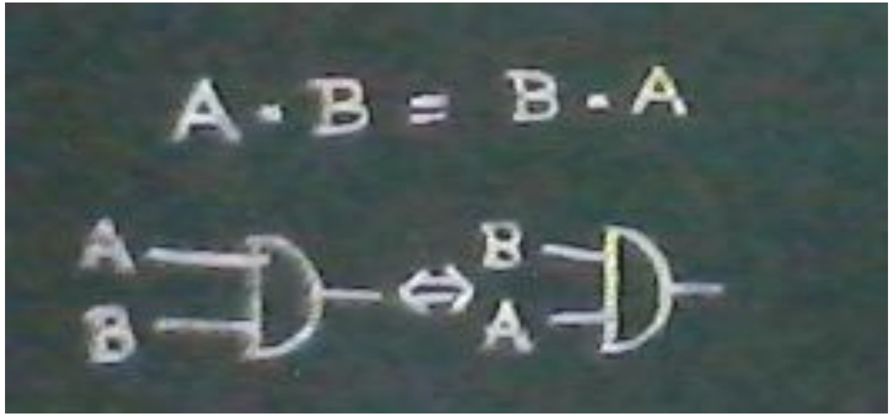


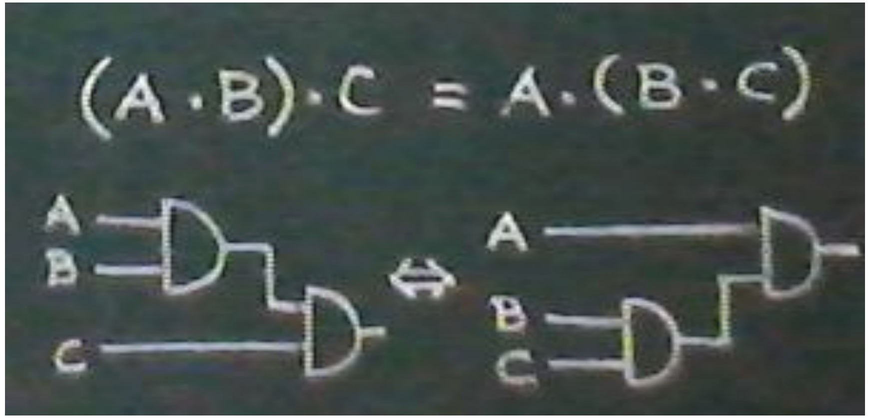


Boolean Algebra: Commutative & Associative Laws









DeMorgan's, Distributive & Other Boolean Laws

DeMorgan's Law -

$$\overline{A+B} = \overline{A} \cdot \overline{B}$$

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

Double Inversion -

$$\overline{\overline{A}} = A$$

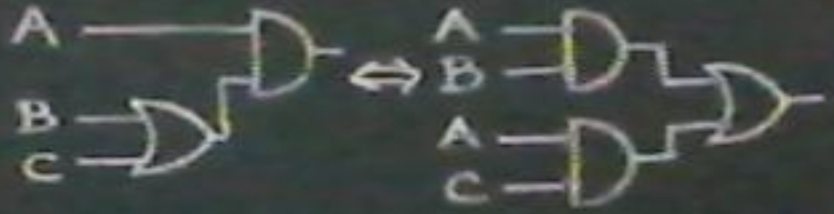
Other Law -

$$A + (A \cdot B) = A$$

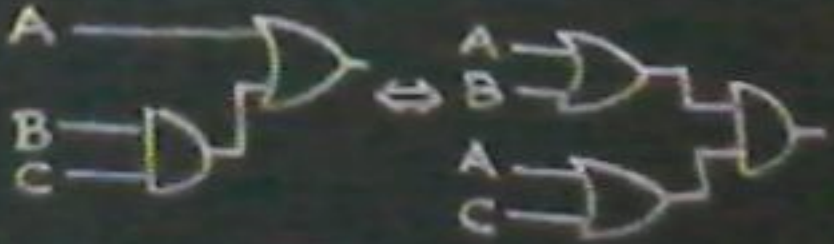
$$A \cdot (A+B) = A$$

Distributive Law -

$$A \cdot (B+C) = A \cdot B + A \cdot C$$



$$A + B \cdot C = (A+B) \cdot (A+C)$$



Proof of DeMorgan's theorem by "Perfect Induction"

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

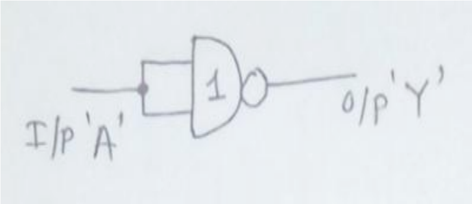
A	B	A+B	$\overline{A+B}$	A.B	$\overline{A.B}$	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$	$\overline{A}.\overline{B}$
0	0	0	1	0	1	1	1	1	1
0	1	1	0	0	1	1	0	1	0
1	0	1	0	0	1	0	1	1	0
1	1	1	0	1	0	0	0	0	0

First Theorem proved

Second Theorem proved

NAND gate as Universal Gate

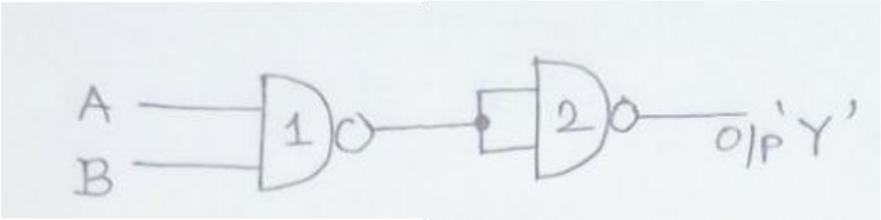
- **Inverter/NOT gate using NAND:**



$$Y = \overline{A \cdot A}$$

$$\boxed{\therefore Y = \overline{A}}$$

- **AND gate using NAND:**

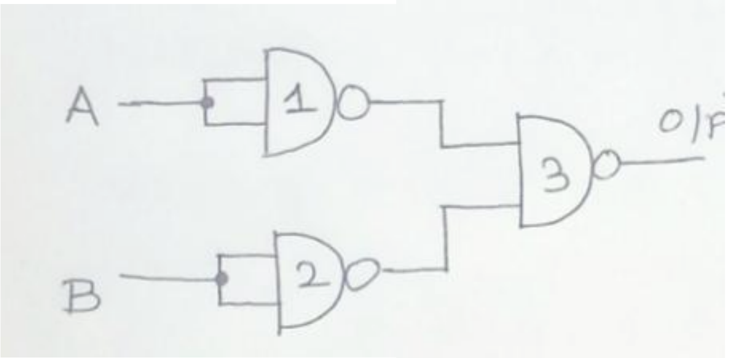


$$\text{NAND } ① = \overline{A \cdot B}$$

$$\text{NAND } ② = \overline{\overline{A \cdot B}}$$

$$\boxed{\therefore Y = A \cdot B}$$

- **OR gate using NAND:**



$$\text{NAND } ① = \overline{A}$$

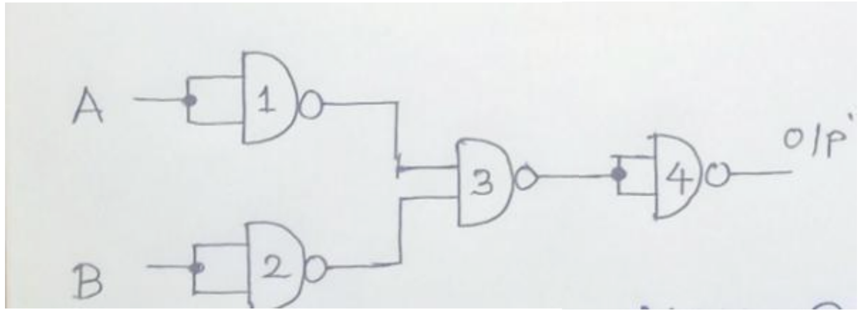
$$\text{NAND } ② = \overline{B}$$

$$\text{NAND } ③ = \overline{\overline{A} \cdot \overline{B}} = \overline{\overline{A}} + \overline{\overline{B}}$$

$$\boxed{\therefore Y = A + B}$$

NAND gate as Universal Gate contd....

- **NOR gate using NAND:**

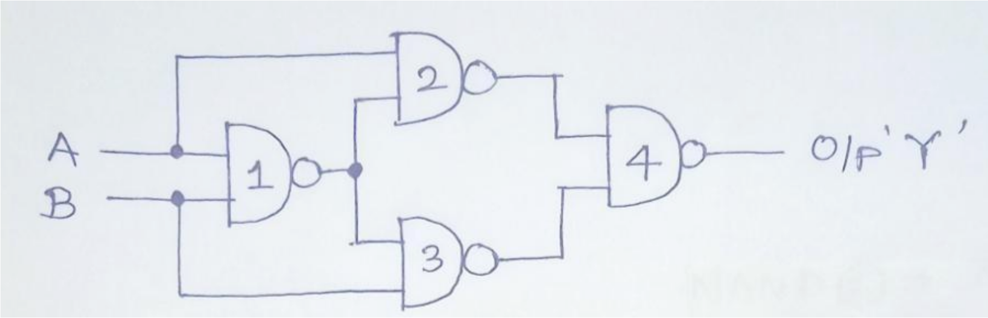


$$NAND\ ③ = A + B$$

$$NAND\ ④ = \overline{A + B}$$

$$\boxed{\therefore Y = \overline{A + B}}$$

- **Ex-OR gate using NAND:**



$$NAND\ ① = \overline{A \cdot B} = \overline{A} + \overline{B}$$

$$NAND\ ② = \overline{A \cdot (\overline{A} + \overline{B})} = \overline{A \cdot \overline{A} + A \cdot \overline{B}} = \overline{A \cdot \overline{B}}$$

$$NAND\ ③ = \overline{B \cdot (\overline{A} + \overline{B})} = \overline{\overline{A} \cdot B + B \cdot \overline{B}} = \overline{\overline{A} \cdot B}$$

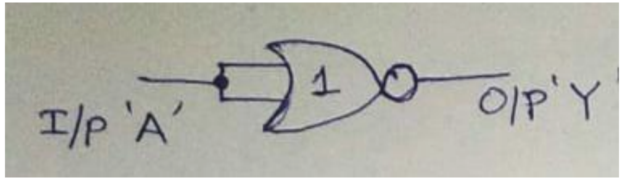
$$NAND\ ④ = \overline{(\overline{A \cdot \overline{B}}) \cdot (\overline{\overline{A} \cdot B})} = \overline{\overline{A \cdot \overline{B}}} + \overline{\overline{\overline{A} \cdot B}}$$

$$= A \cdot \overline{B} + \overline{A} \cdot B$$

$$\boxed{\therefore Y = A \cdot \overline{B} + \overline{A} \cdot B}$$

NOR gate as Universal Gate

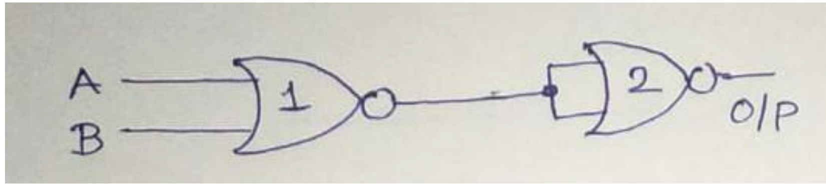
- **Inverter/NOT gate using NOR:**



$$Y = \overline{A + A}$$

$$\boxed{\therefore Y = \overline{A}}$$

- **OR gate using NOR:**

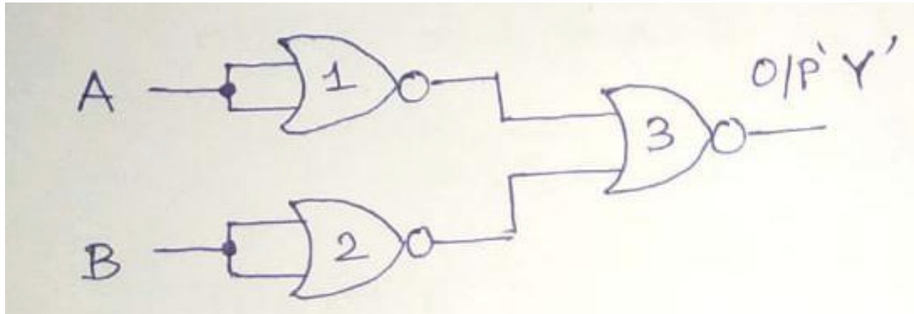


$$\text{NOR}(1) = \overline{A + B}$$

$$\text{NOR}(2) = \overline{\overline{A + B}}$$

$$\boxed{\therefore Y = A + B}$$

- **AND gate using NOR:**



$$\text{NOR}(1) = \overline{A}$$

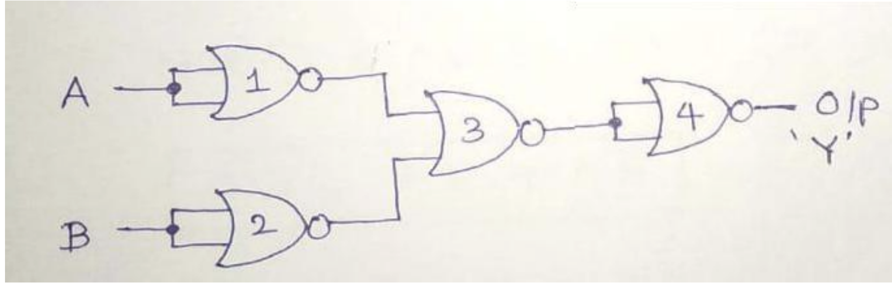
$$\text{NOR}(2) = \overline{B}$$

$$\text{NOR}(3) = \overline{\overline{A} + \overline{B}} = \overline{\overline{A}} \cdot \overline{\overline{B}}$$

$$\boxed{\therefore Y = A \cdot B}$$

NOR gate as Universal Gate contd....

NAND gate using NOR:



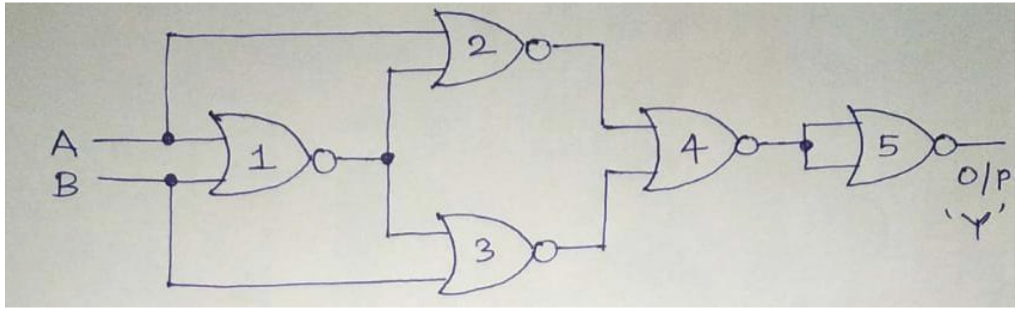
NOR ①, ②, ③ form 'AND'

NOR ③ $= A \cdot B$

NOR ④ $= \overline{A \cdot B}$

$$\boxed{\therefore Y = \overline{A \cdot B}}$$

Ex-OR gate using NOR:



NOR ① $= \overline{A + B} = \overline{A} \cdot \overline{B}$

NOR ② $= \overline{A + \overline{A} \cdot \overline{B}} = \overline{(A + \overline{A})(A + \overline{B})}$

$= \overline{(A + \overline{B})} = \overline{A} \cdot B$

NOR ③ $= \overline{B + \overline{A} \cdot \overline{B}} = \overline{(\overline{A} + B)(B + \overline{B})}$

$= \overline{(\overline{A} + B)} = A \cdot \overline{B}$

NOR ④ & NOR ⑤ form OR gate

$$\boxed{\therefore Y = \overline{A} B + A \overline{B}}$$

Computer Architecture vs Computer Organization

Computer Architecture

- Defines the parameters/attributes visible to the programmer
- Architecture includes: Instruction set, No. of bits, Data transfer mechanisms etc.
- Includes simply the instructions
- Remains same for many years like IBM PC, Apple Mac etc.

Computer Organization

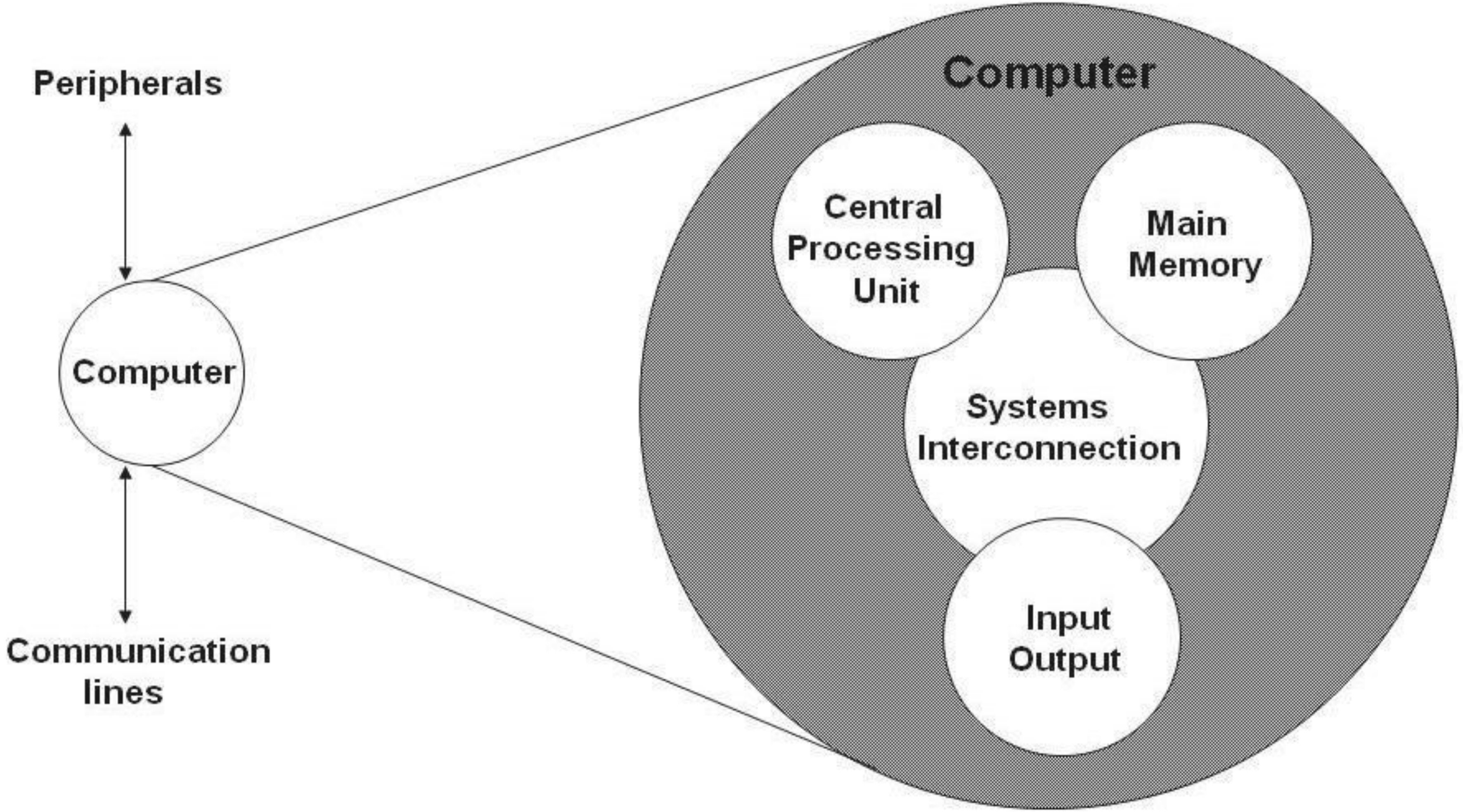
- Defines the attributes which are invisible/ non-available to the programmer
- Organization includes: Control signals, Interfacing methods, Details of control unit designs etc.
- Includes the methods to implement that instruction
- Architecture has many organizational models with different price and performance characteristics

Hierarchical structure of Digital Computer system

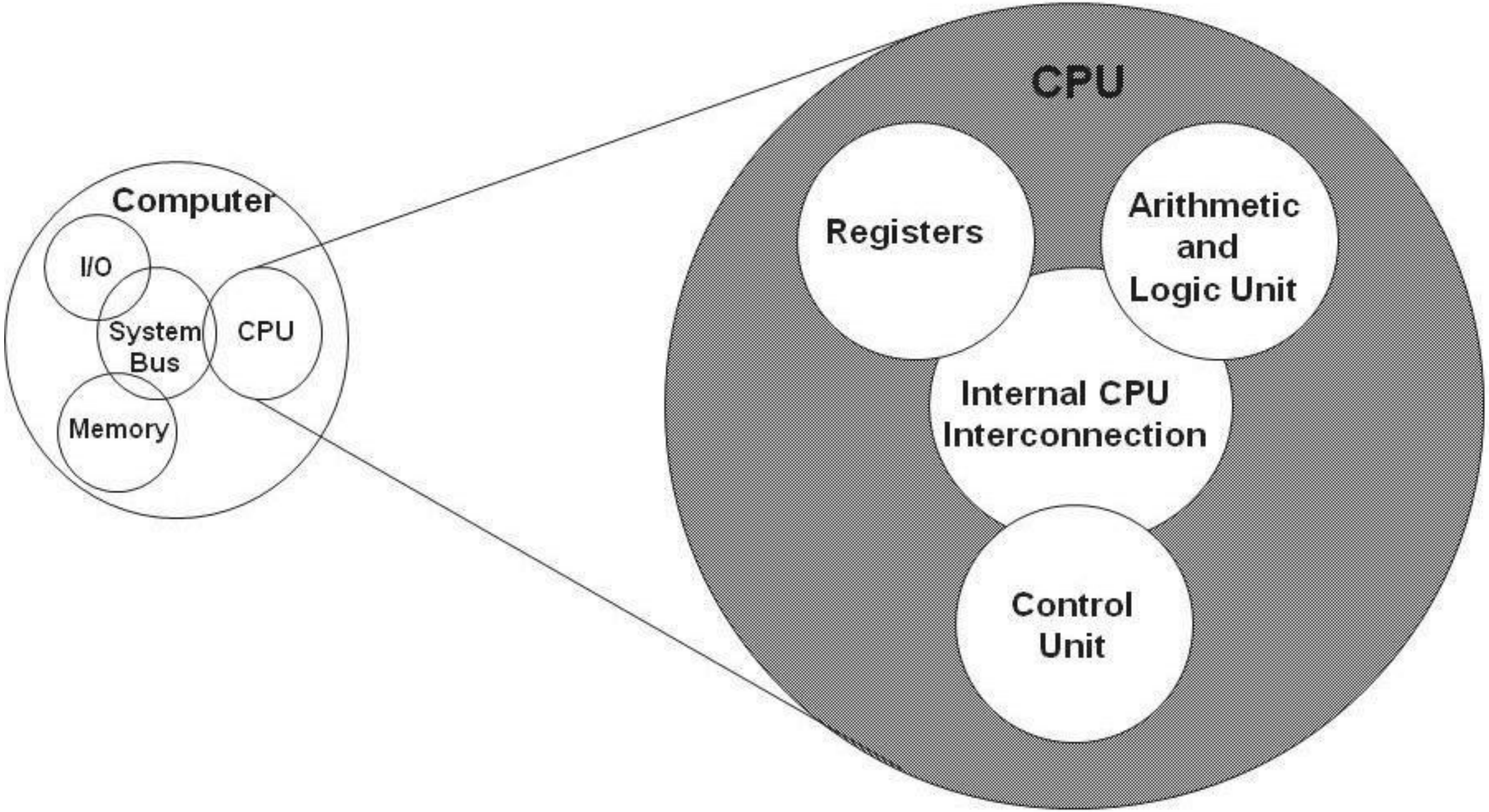
3 levels in hierarchical structure of computer organization:

1) Digital computer- Top level structure
2) CPU level structure
3) CU level structure

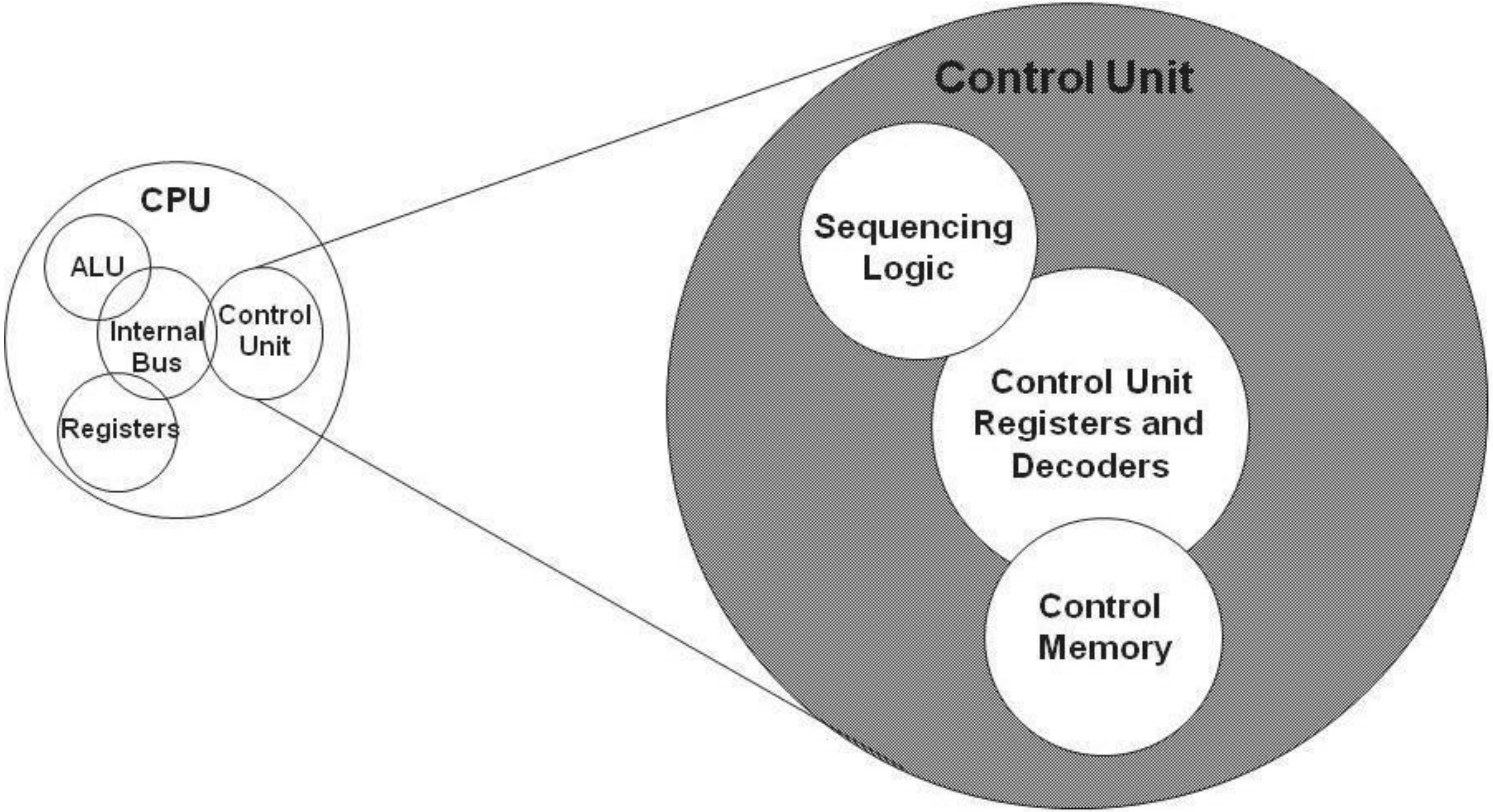
Digital Computer- Top Level Structure



CPU Level Structure

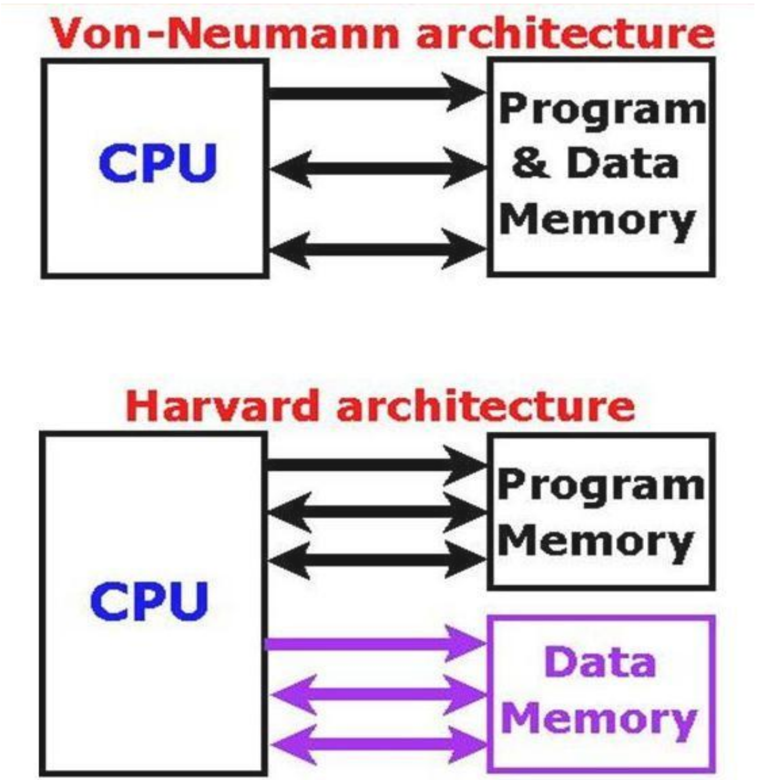


Control Unit (CU) Level Structure

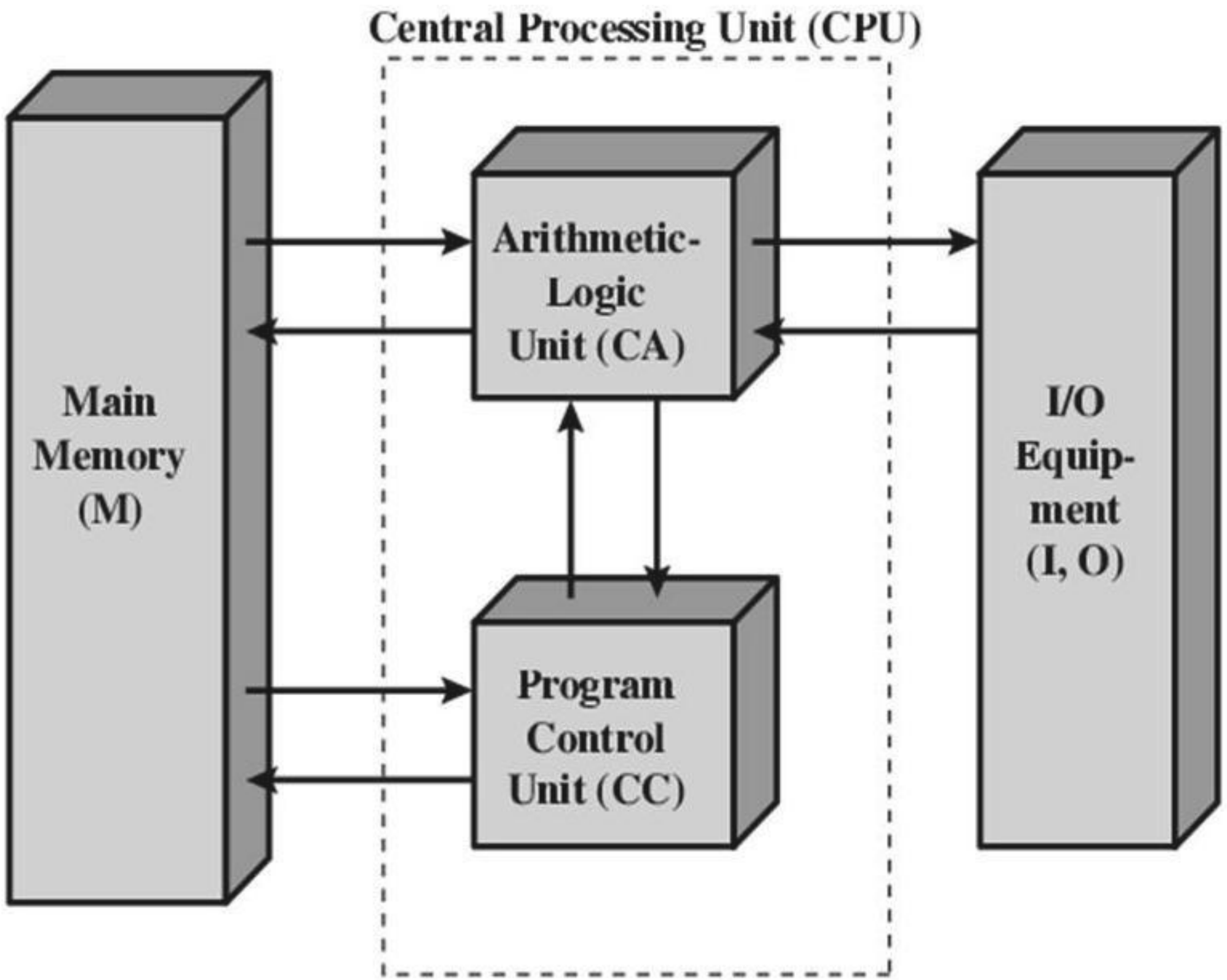


Von-Neumann vs Harvard Architectures

Von-Neumann (Princeton)	Harvard
- Common memory for Program & Data	- Separate memory
- No protection amongst Program & Data	- Natural protection
- Simple, Inexpensive Hardware	- Complex & Costly H/W
- Complex O.S.	- Simple O.S.
- Less execution speed	- Higher execution speed



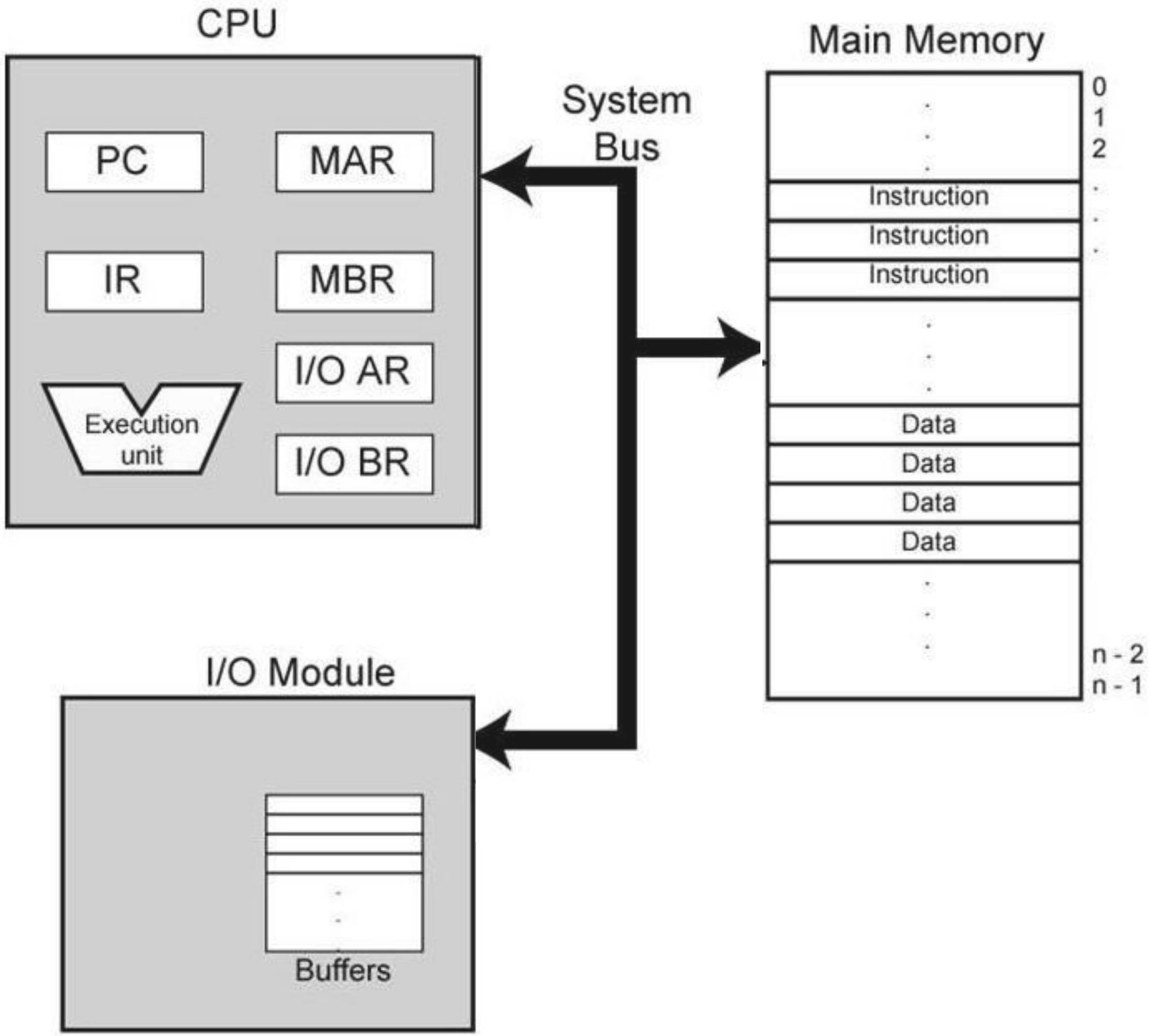
Digital Computer Structure: Von-Neumann machine



Operational Principle of Von-Neumann machine

- Works on "Stored Program" concept
- Main memory stores programs and data in a single memory (common RAM)
- Control unit interprets/decodes the instructions fetched from memory
- ALU processes data by executing the decoded instructions
- Generated results can be copied into memory (RAM) or sent to an output device

Functional block diagram of Von-neumann Digital Computer



PC	=	Program counter
IR	=	Instruction register
MAR	=	Memory address register
MBR	=	Memory buffer register
I/O AR	=	Input/output address register
I/O BR	=	Input/output buffer register

CPU Internal Registers (other than General data)

- **MAR:** Holds memory address in read or write operation
- **MBR:** Holds the data which is read from or written into the addressed memory location
- **I/O AR:** Holds I/O device address in read or write operation
- **I/O BR:** Holds the data which is read from or written into the addressed I/O device
- **PC:** Points the address of next instruction to be executed
- **IR:** Holds the instruction op-code while execution