SIDDARTHA INSTITUTE OF SCIENCE AND TECHNOLOGY
(AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMPUTER ORGANIZATION & ARCHITECTURE
(20CS0504)

COURSE OBJECTIVES

COURSE OBJECTIVES

1. Explain the **fundamentals of computer organization** and its relevance to classical and Modern problems of computer design
2. Make the students **understand the structure and behavior** of various functional modules of a computer.
3. Understand the **techniques that computers use to communicate** with I/O devices.
4. Illustrate the concepts of **pipelining and the way it can speed** up processing.
5. Understand the basic **characteristics of multiprocessors**

COURSE OUTCOMES

On successful completion of the course, the student will be able to

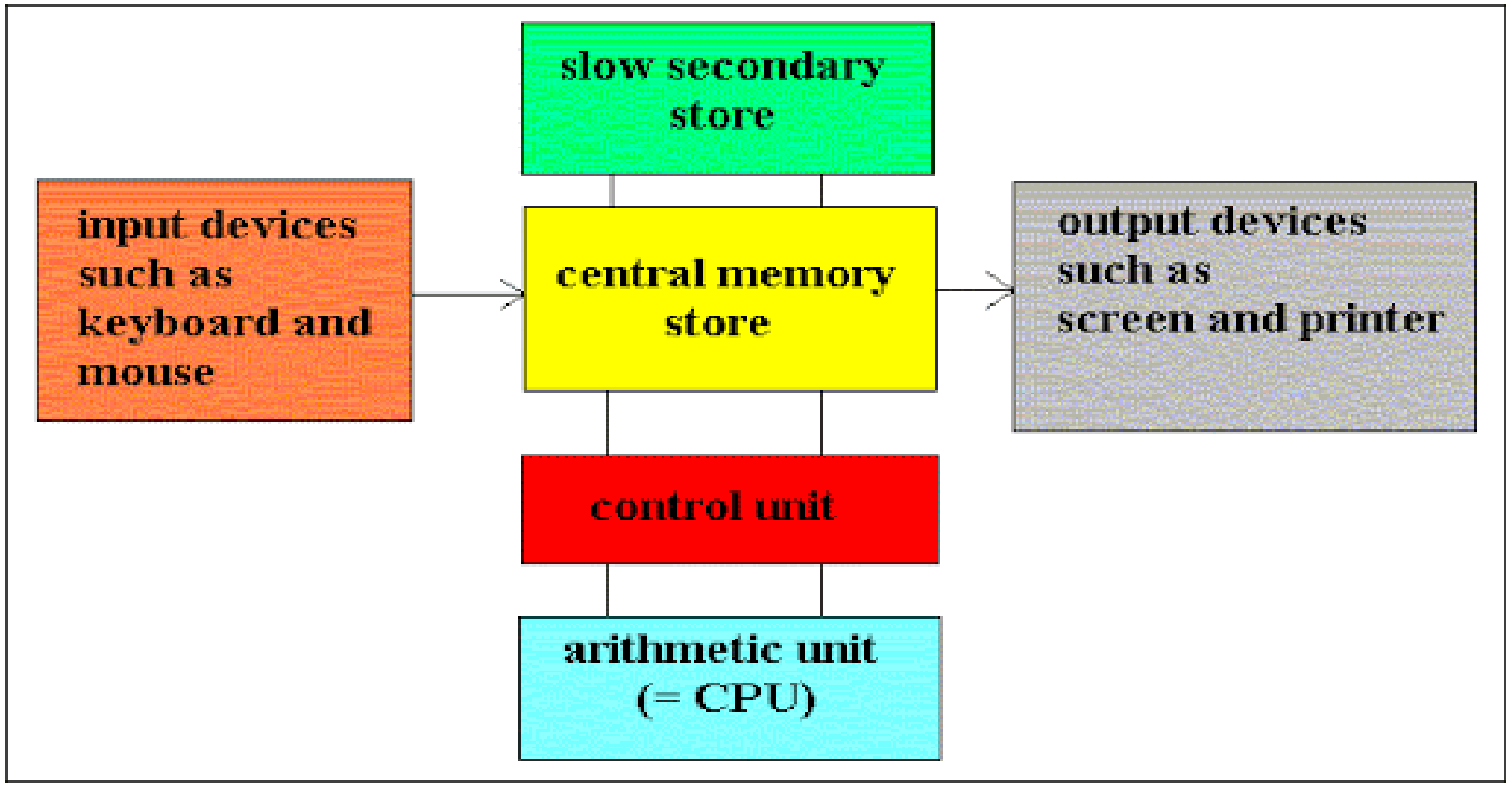
1. Describe the fundamental operations of computer
2. Explain the functional units of computer system
3. Understand the memory hierarchy and its impact of cost and performance.
4. Discuss hardware requirements for cache memory and virtual memory.
5. Design algorithms to exploit pipelining and multiprocessors
6. Compare memory and I/O devices

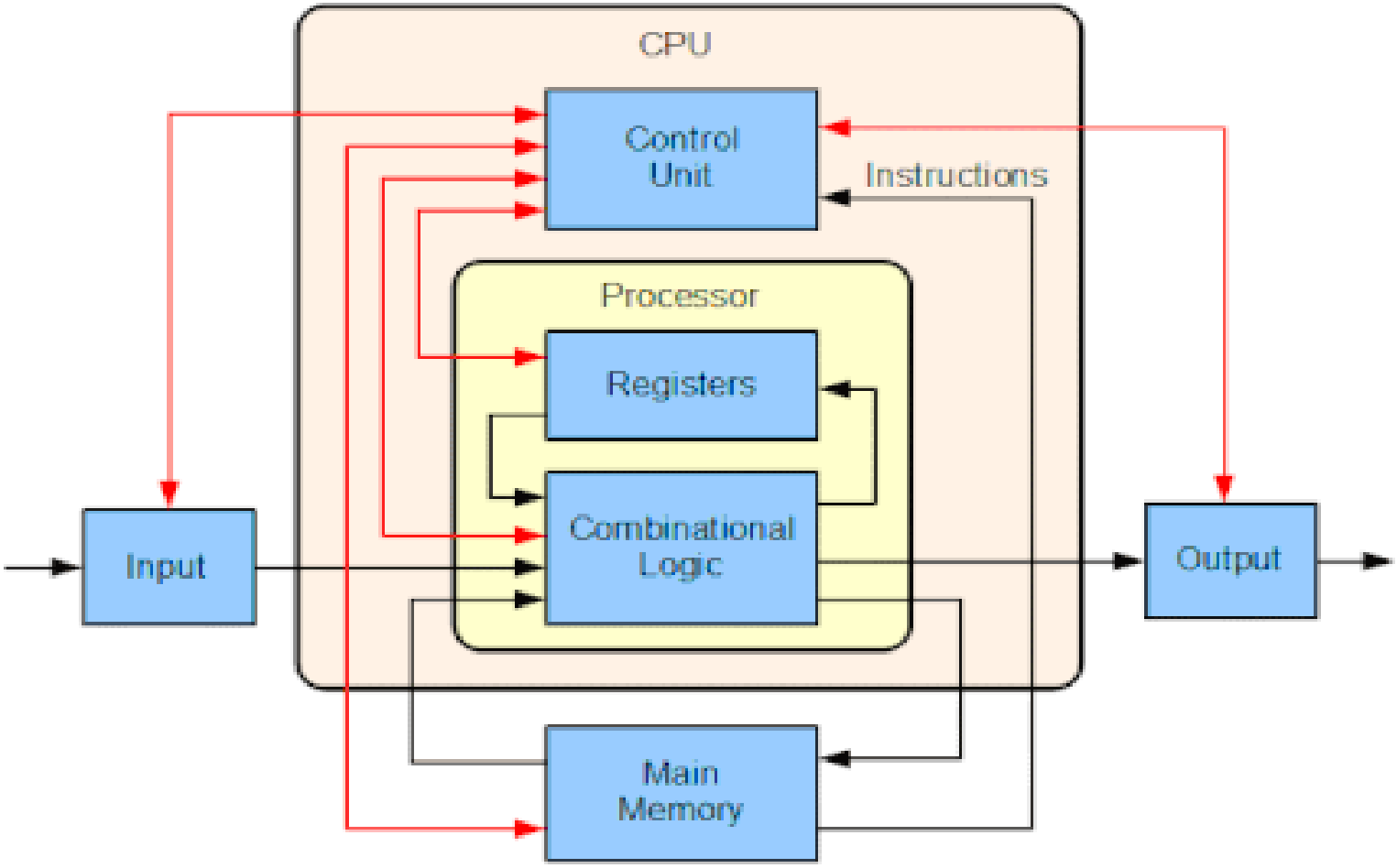
UNIT- I

Basic Structure of Computers: Functional Units, Basic Operational Concepts, Basic I/O Operations, Bus Structures, Instruction Cycle, Data Transfer, Data Manipulation and Program Control, Addressing Modes.

Basic Structure of Computers :

Functional Units:




CPU
Control Unit
Instructions
Processor
Registers
Combinational Logic
Input
Output
Main Memory

Functional units:

- CPU
- The Arithmetic / Logic Unit (ALU)
- The Control Unit
- Main Memory
- External Memory
- Input / Output Devices
- The System Bus

Basic operational concepts:

The fundamental operation of most CPUs

- To execute a sequence of stored instructions called a program.

1. The program is represented by a series of numbers that are kept in some kind of computer memory.

2. There are four steps that nearly all CPUs use in their operation: fetch, decode, execute, and write back.

Cont..

Anything that feeds the data into the computer. This data can be in alpha-numeric form which needs to be keyed-in or in its very basic natural form i.e. hear, smell, touch, see; taste & the sixth sense

…feel?

Typical input devices are:

1. Keyboard
2. Mouse
3. Joystick
4. Digitizing Tablet
5. Touch Sensitive Screen
6. Light Pen

Cont..

Output devices display information in a way that you can you can understand. The most common output device is a monitor. It looks a lot a like a TV and houses the computer screen. The monitor allows you to 'see' what you and the computer are doing together.

Basic I/O operations:

- Programmed I/O.
- Interrupt- initiated I/O.
- Direct memory access(DMA).

Programmed I/O: It is due to the result of the I/O instructions that are written in the computer program.

- Each data item transfer is initiated by an instruction in the program. Usually the transfer is from a CPU register and memory. In this case it requires constant monitoring by the CPU of the peripheral devices.

Interrupt- initiated I/O: Since in the above case we saw the CPU is kept busy unnecessarily. This situation can very well be avoided by using an interrupt driven method for data transfer.

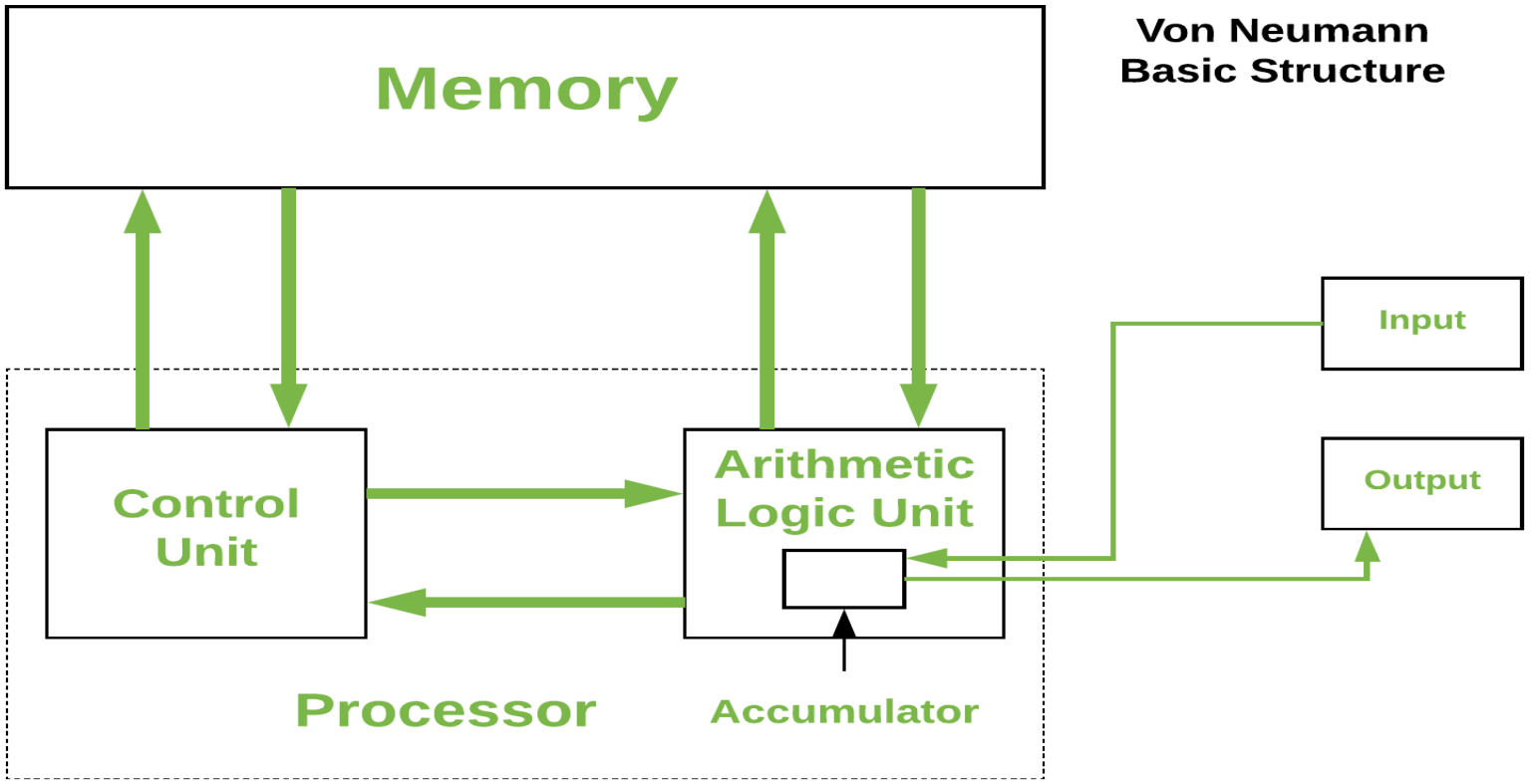
By using interrupt facility and special commands to inform the interface to issue an interrupt request signal whenever data is available from any device.

Direct Memory Access: The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU.

Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU.

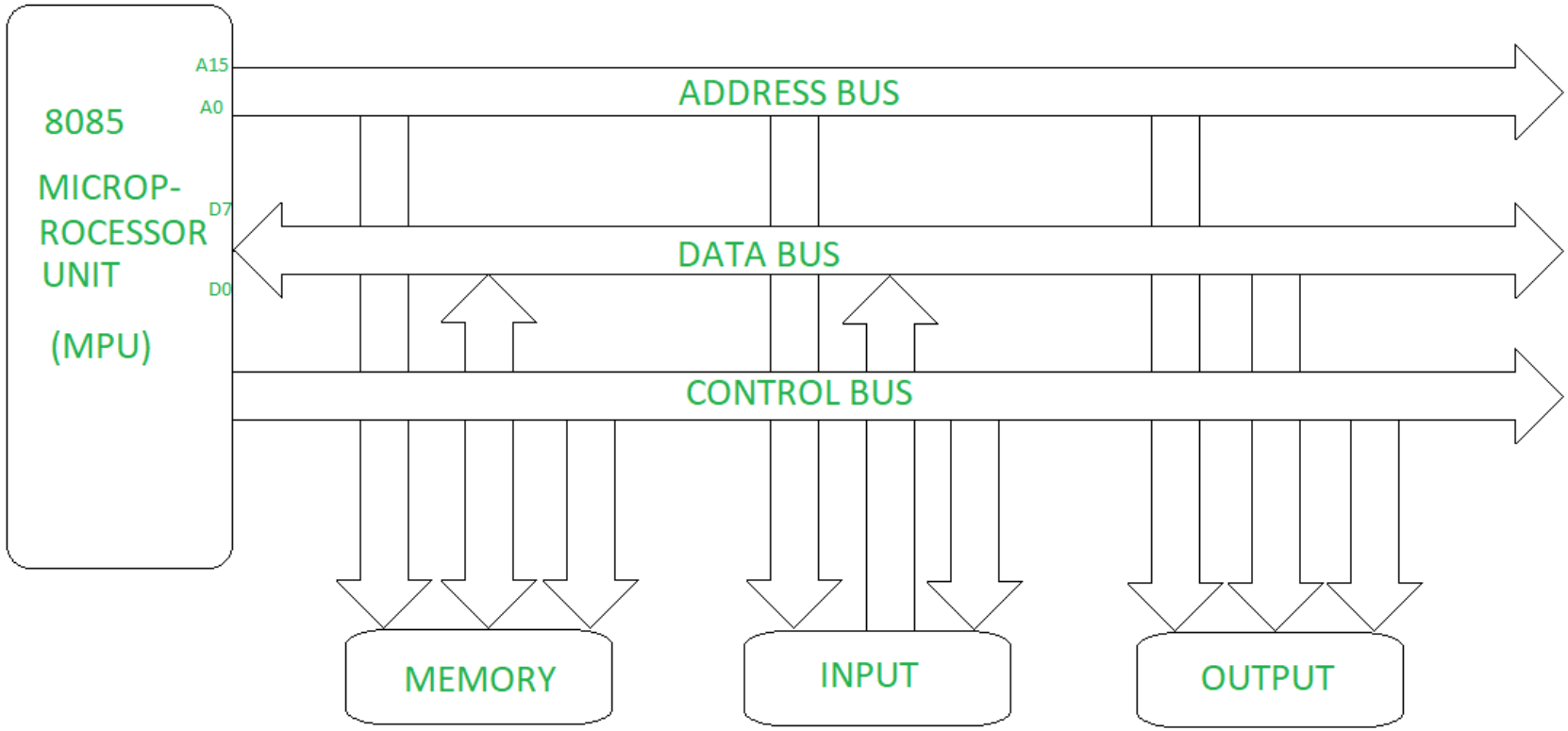
This type of data transfer technique is known as DMA or direct memory access.

Basic structure of computer



Bus structure:

- Bus is a group of conducting wires which carries information, all the peripherals are connected to microprocessor through Bus.
- Diagram to represent bus organization :



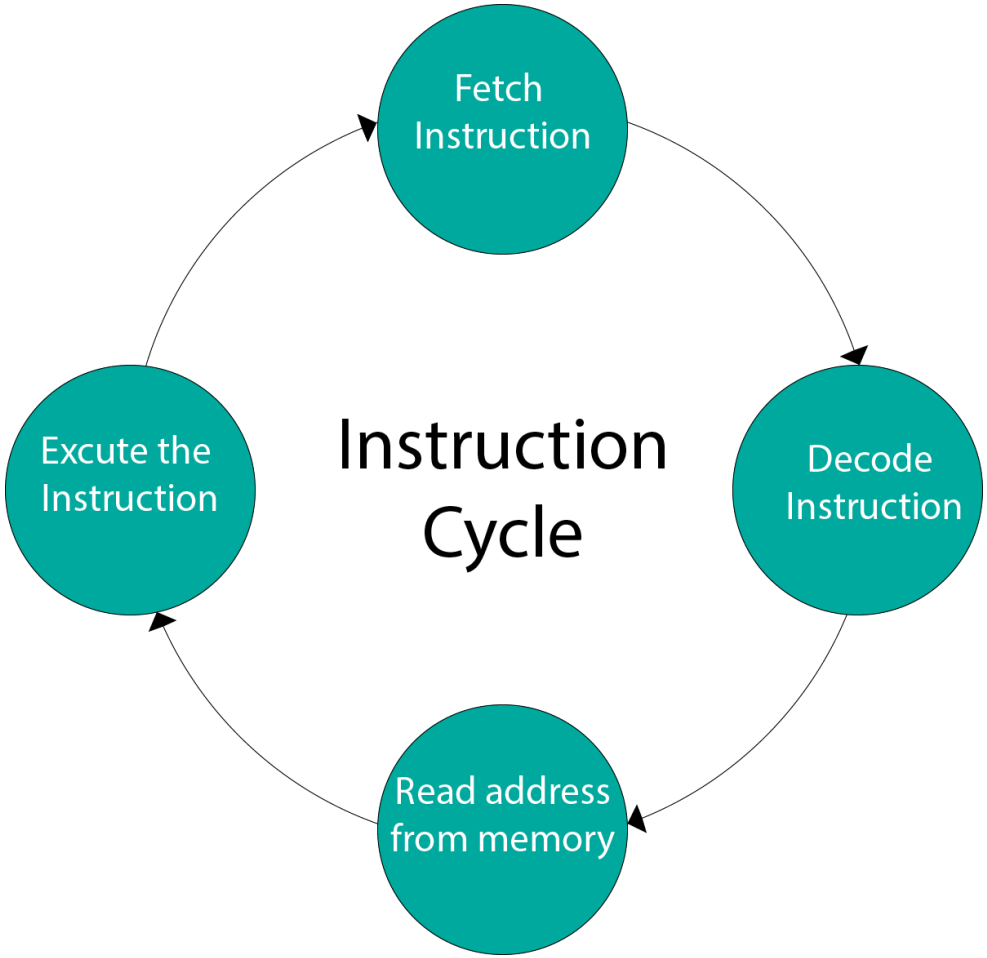
Bus organization system of 8085 Microprocessor

Instruction Cycles:

Registers Involved In Each Instruction Cycle:

- **Memory address registers(MAR)** : It is connected to the address lines of the system bus. It specifies the address in memory for a read or write operation.
- **Memory Buffer Register(MBR)** : It is connected to the data lines of the system bus. It contains the value to be stored in memory or the last value read from the memory.
- **Program Counter(PC)** : Holds the address of the next instruction to be fetched.
- **Instruction Register(IR)** : Holds the last instruction fetched.

https://www.javatpoint.com/instruction-cycle



Data Transfer

- **Data transfer** instructions move **data** from one place in the computer to another without changing the data.
- Typical transfers are between memory and processor registers, between processor registers and input and output registers, and among the processor registers themselves.

Synchronous Data Transfer in Computer Organization

In **Synchronous data transfer**, the sending and receiving units are enabled with same clock signal. It is possible between two units when each of them knows the behavior of the other.

The master performs a sequence of instructions for data transfer in a predefined order. All these actions are synchronized with the common clock.

- **Data transfer instructions** move **data** from one place in the **computer** to another without changing the **data**.
- Typical **transfers** are between memory and processor registers, between processor registers and input and output registers, and among the processor registers themselves.

- MOV - MOV AX, [SI]
- PUSH - PUSH DX
- POP - POP AS
- PUSHA - PUSH A
- XCHG - XCHG [2050], AX

Data Manipulation :

- **Data manipulation instructions** perform operations on **data** and provide the computational capabilities for the **computer**.
- There are three types of **data manipulation instructions**: Arithmetic **instructions**, Logical and bit **manipulation instructions**, and Shift **instructions**.
-

Arithmetic instructions – add, subtract, increment, decrement, convert byte/word and compare.

- ADD, SUB: Add, subtract byte or word
- ADC, SBB :Add, subtract byte or word and carry (borrow).
- INC, DEC: Increment, decrement byte or word.
- NEG: Negate byte or word (two's complement).
- CMP: Compare byte or word (subtract without storing).

Logic instructions – AND, OR, exclusive OR, shift/rotate and test

- NOT : Logical NOT of byte or word (one's complement)
- AND: Logical AND of byte or word
- OR: Logical OR of byte or word.
- XOR: Logical exclusive-OR of byte or word

String manipulation instructions – load, store, move, compare and scan for byte/word

- MOVS: Move byte or word string
- MOVSB, MOVSW: Move byte, word string.
- CMPS: Compare byte or word string.
- SCAS S: can byte or word string (comparing to A or AX)

Program control instructions:

- LOOP: Loop unconditional, count in CX, short jump to target address.
- LOOPE (LOOPZ): Loop if equal (zero), count in CX, short jump to target address.
- LOOPNE (LOOPNZ): Loop if not equal (not zero), count in CX, short jump to target address.
- JCXZ: Jump if CX equals zero (used to skip code in loop).

Processor control instructions:

- Flag manipulation:
- STC, CLC, CMC: Set, clear, complement carry flag.
- STD, CLD: Set, clear direction flag.STI, CLI: Set, clear interrupt enable flag.
- PUSHF, POPF: Push flags onto stack, pop flags off stack.

ADDRESSING AND ADDRESSING MODES

To perform any operation, the corresponding instruction is to be given to the microprocessor. In each instruction, programmer has to specify 3 things:

- Operation to be performed.
- Address of source of data.
- Address of destination of result.

Definition:

- The different ways in which the location of an operand is specified in an instruction are referred to as addressing modes.

- The method by which the address of source of data or the address of destination of result is given in the instruction is called Addressing Modes

.

.

1 Implied addressing mode

- In this mode the operands are specified implicitly in the definition of the instruction. For example the ‘complement accumulator’ instruction is an implied mode instruction because the operand in the accumulator register is implied in the definition of the instruction itself.

Example:CMA

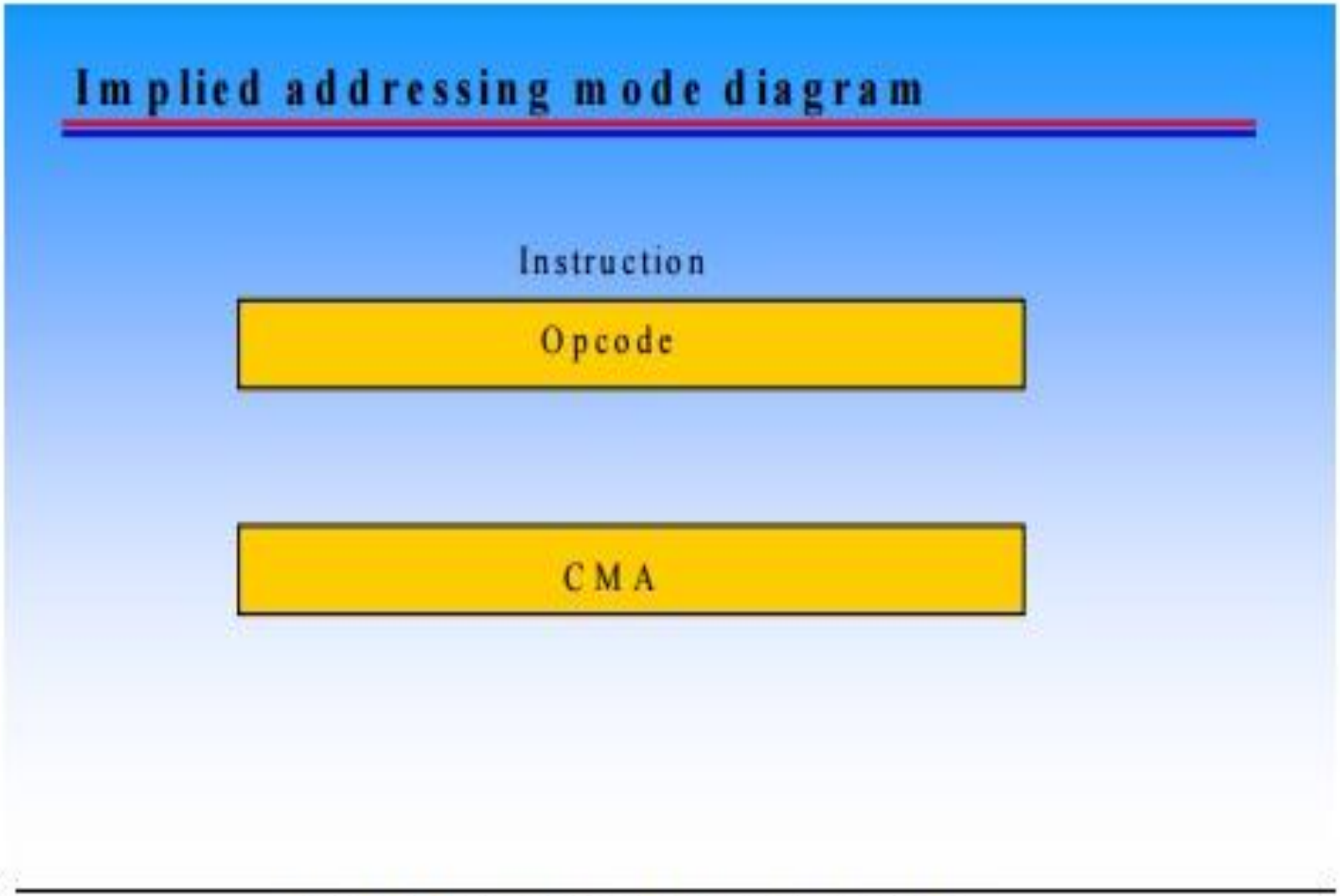
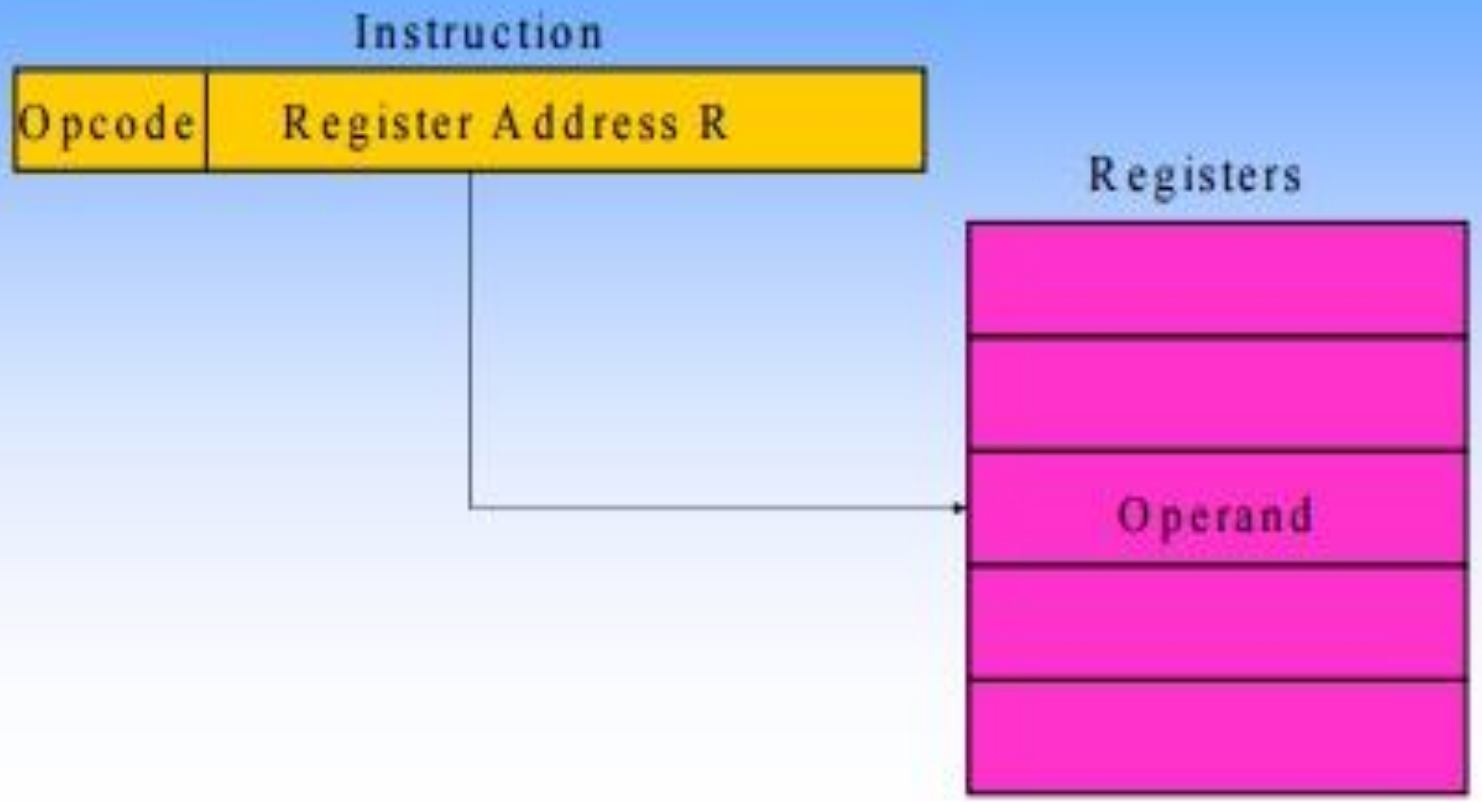


Fig1.8 Implied addressing mode

2. **Register addressing mode** - The operand is the contents of a processor register; the name (address) of the register is given in the instruction.

- Example: **MOVE R1,R2**
- This instruction copies the contents of register R2 to R1.

Register addressing mode diagram



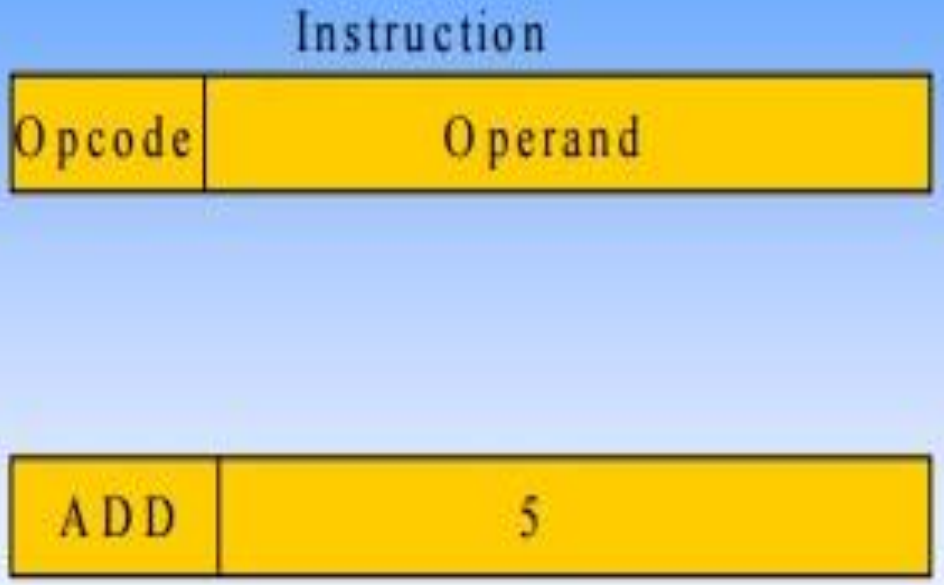
3. **Absolute addressing mode** - The operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct.)

- Example: **MOVE LOC,R2**
- This instruction copies the contents of memory location of LOC to register R2.

4. Immediate addressing mode - The operand is given explicitly in the instruction.

- Example: **MOVE #200 , R0**
- The above statement places the value 200 in the register R0. A common convention is to use the sharp sign (#) in front of the value to indicate that this value is to be used as an immediate operand.

Immediate addressing mode diagram

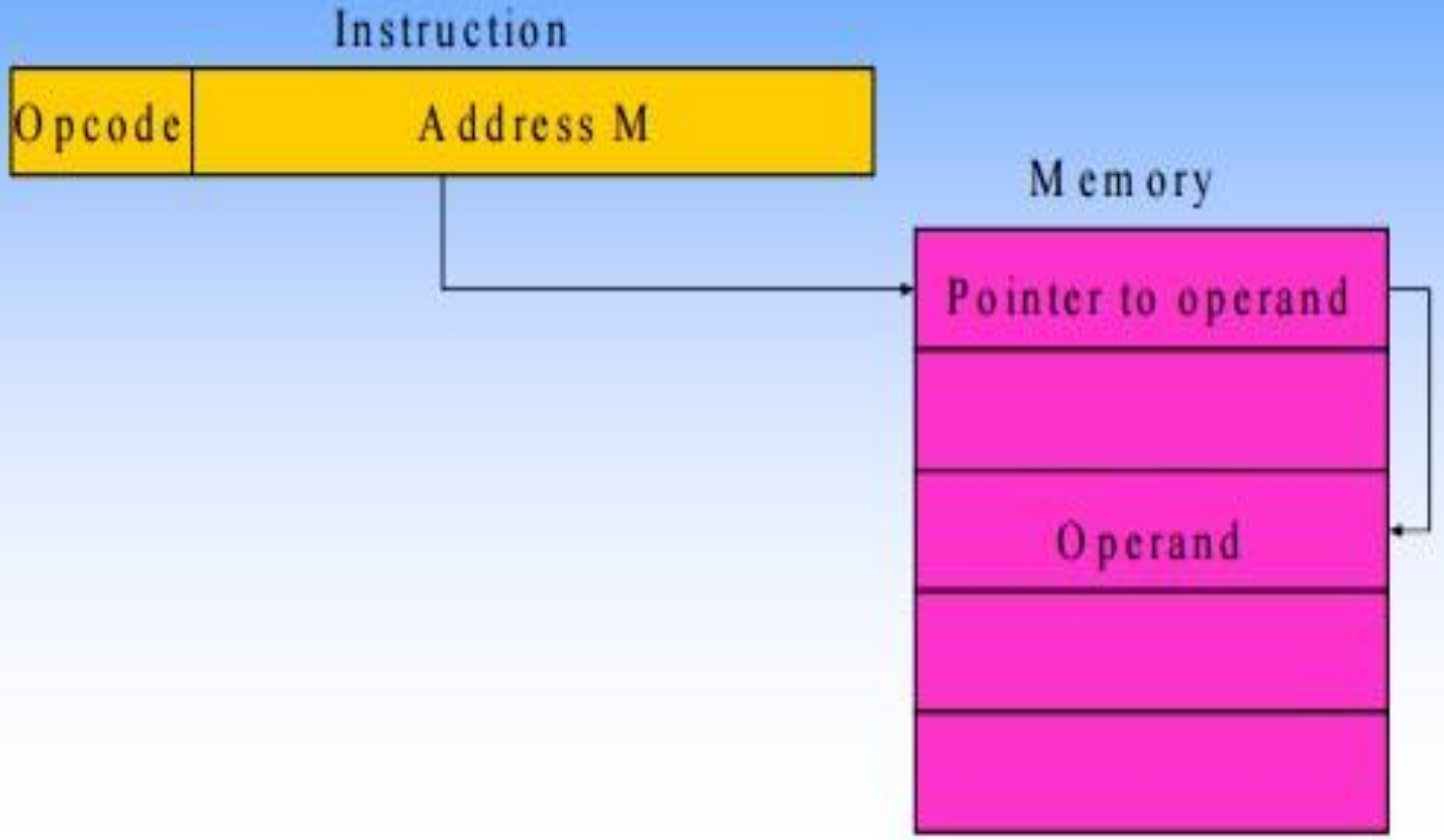


- **INDIRECTION AND POINTERS**
- In the addressing modes that follow, the instruction does not give the operand or its address explicitly. Instead, it provides information from which the memory address of the operand can be determined. We refer to this address as the effective address (EA) of the operand.

5. Indirect addressing mode

- The effective address of the operand is the contents of a register or memory location whose address appears in the instruction.
- Example **Add (R2),R0**
- Register R2 is used as a pointer to the numbers in the list, and the operands are accessed indirectlythroughR2.

Indirect addressing mode diagram



6. Index mode

- The effective address of the operand is generated by adding a constant value to the contents of a register. We indicate the Index mode symbolically as **X(Ri).**
- Where X denotes the constant value contained in the instruction and Ri is the name of the register involved. The effective address of the operand is given by **EA = X + [Ri].**

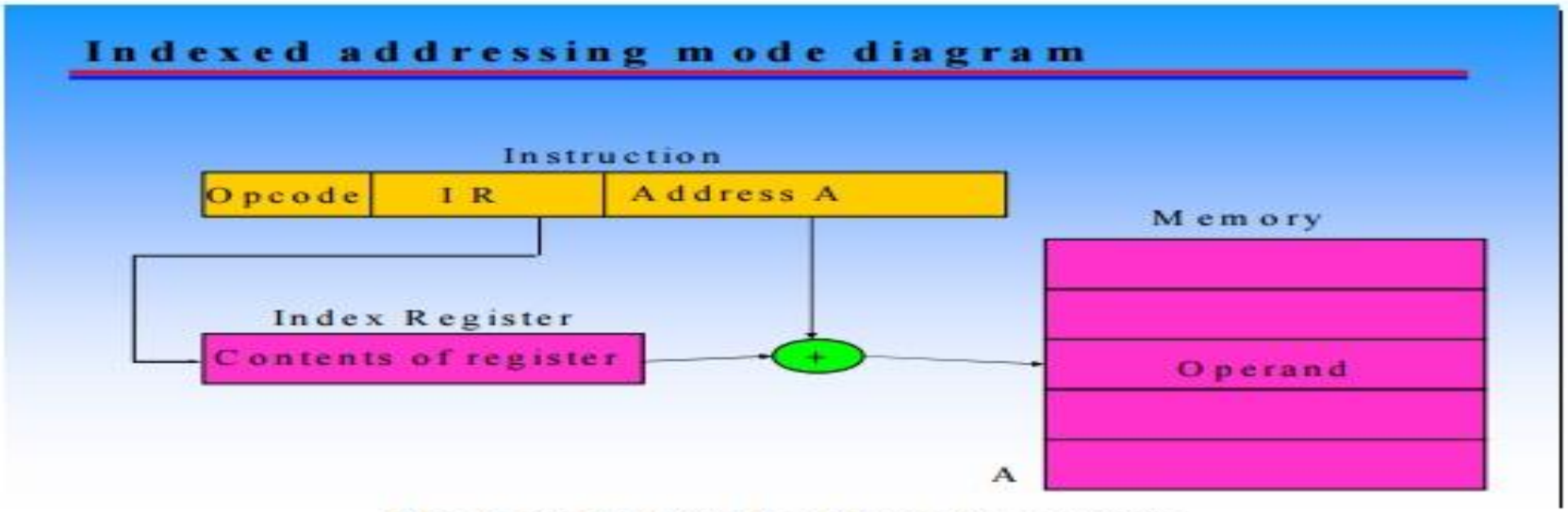


Fig.1.16 Indexed addressing mode

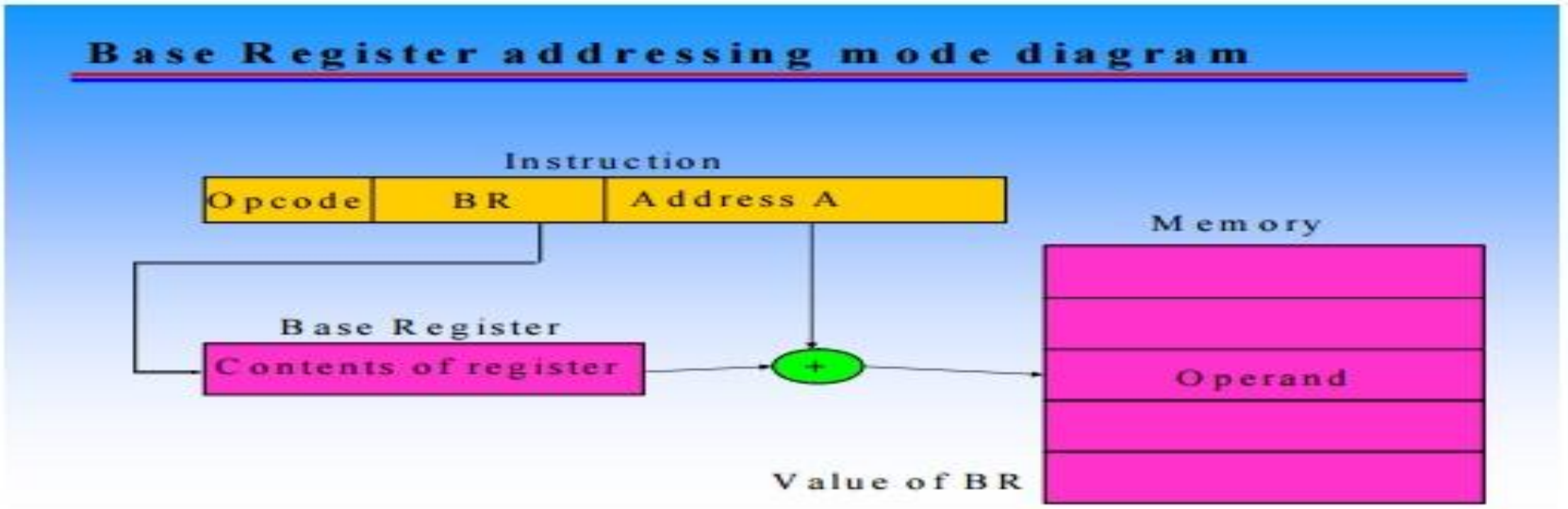


Fig.1.17 Base register addressing mode

7. Base register addressing mode

- In this mode the content of base register is added to the address part of the instruction to obtain the effective address.
- This is similar to the indexed addressing mode except that the register is now called a base register instead of an index register.
- The difference between the two modes is in the way they are used rather than in the way that they are computed.

8. Autoincrement mode - The effective address of the operand is the contents of a register specified in the instruction.

After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list.

The Auto increment mode is written as **(Ri) +.**

9.Autodecrement mode - The contents of a register specified in the instruction is first automatically decremented and is then used as the effective address of the operand.

We denote the Autodecrement mode by putting the specified register in parentheses, preceded by a sign to indicate that the contents of the register are to be decremented before being used as the effective address.

Thus, we write **- (Ri)**

Web references:

https://www.studytonight.com

https://www.javatpoint.com

https://www.tutorialspoint.com/

Unit- II - Data Representation

- **Data Representation:** Signed number representation, Fixed and Floating point representations, Character representation.
- **Computer Arithmetic**: Integer addition and subtraction, Multiplication – shift and add, Booth multiplication, Division, Signed operand multiplication, Floating point arithmetic.

Number Systems

- Human beings use *decimal* (base 10) because we have 10 figures such as 0, 1, 2, Up to 9).
- Computers euse *binary* (base 2) number system, as they are made from binary digital components (known as transistors) operating in two states - on and off.
- In computing, we also use *hexadecimal* (base 16) or *octal* (base 8) number systems, as a *compact* form for representing binary numbers.

Decimal (Base 10) Number System

- Decimal number system has ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9, called *digit*s.
- It uses *positional notation*. That is, the least-significant digit (right-most digit) is of the order of 10^0 (units or ones), the second right-most digit is of the order of 10^1 (tens), the third right-most digit is of the order of 10^2 (hundreds), and so on, where ^ denotes exponent. For example,
- 735 = 700 + 30 + 5 = 7×10^2 + 3×10^1 + 5×10^0

Binary (Base 2) Number System

- Binary number system has two symbols: 0 and 1, called *bits*. It is also a *positional notation*, for example,

101102

= 10000 + 0000 + 100 + 10 + 0 = 1×2^4 + 0×2^3 + 1×2^2 + 1×2^1 + 0×2^0

A binary digit is called a *bit*. Eight bits is called a *byte* (why 8-bit unit? Probably because $8=2^3$).

Hexadecimal (Base 16) Number System

Hexadecimal number system uses 16 symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F, called *hex digits*. It is a *positional notation*, for example,

A3EH = A00H + 30H + EH = 10×16^2 + 3×16^1 + 14×16^0

Signed number representation:

n-bit Sign Integers in 2's Complement Representation

- In 2's complement representation:
- Again, the most significant bit (msb) is the *sign bit*, with value of 0 representing positive integers and 1 representing negative integers.
- The remaining *n*-1 bits represents the magnitude of the integer

Fixed point number representation

For positive integers, the absolute value of the integer is equal to "the magnitude of the (*n*-1)-bit binary pattern".

For negative integers, the absolute value of the integer is equal to "the magnitude of the *complement* of the (*n*-1)-bit binary pattern *plus one*" (hence called 2's complement).

https://www3.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html

- Example 1: Suppose that *n*=8 and the binary representation 0 100 0001B.
  Sign bit is 0 ⇒ positive
  Absolute value is 100 0001B = 65D
  Hence, the integer is +65D
- Example 2: Suppose that *n*=8 and the binary representation 1 000 0001B.
  Sign bit is 1 ⇒ negative
  Absolute value is the complement of 000 0001B plus 1, i.e., 111 1110B + 1B = 127D
  Hence, the integer is -127D

- Example 3: Suppose that *n*=8 and the binary representation 0 000 0000B.

  Sign bit is 0 ⇒ positive

  Absolute value is 000 0000B = 0D

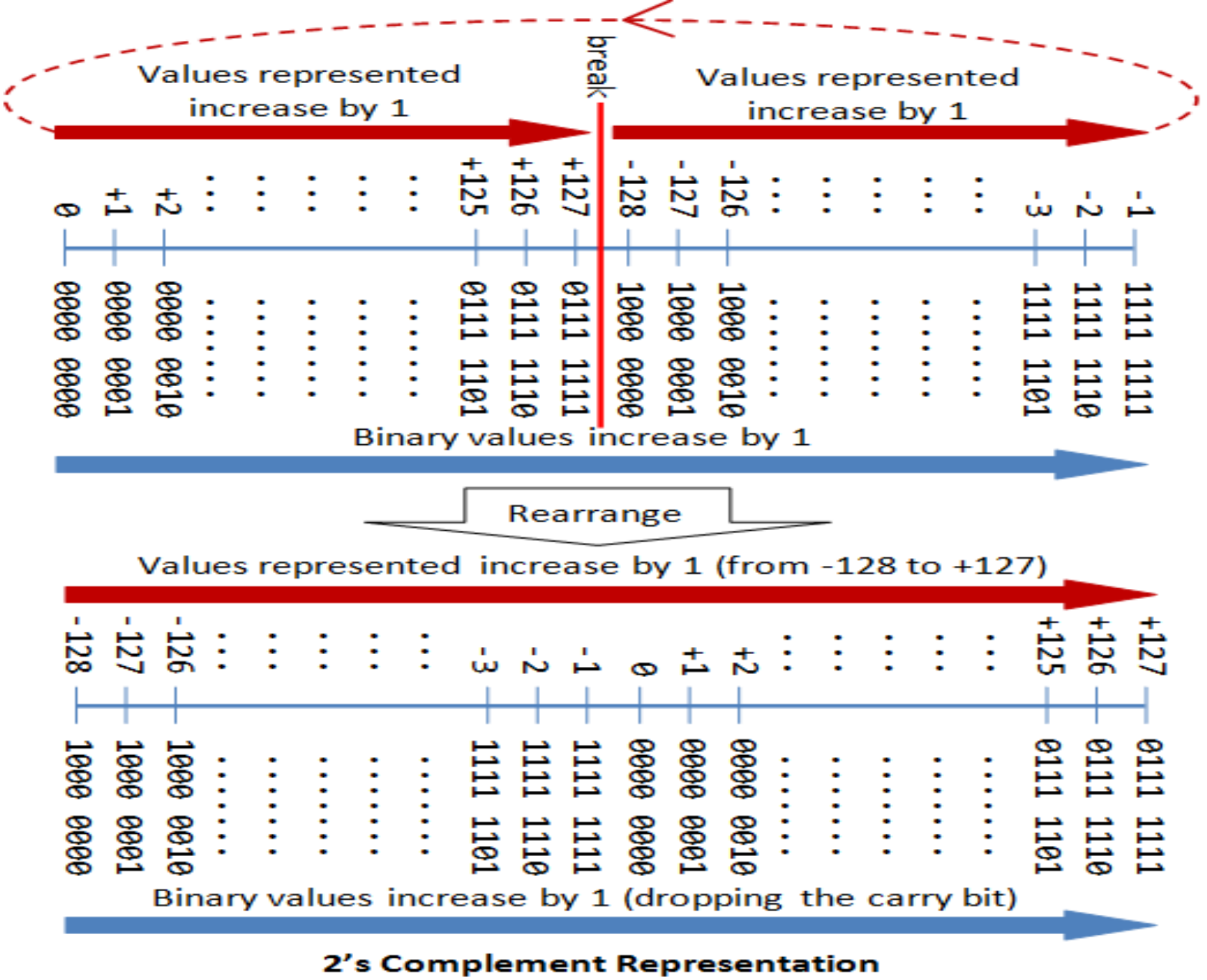
  Hence, the integer is +0D

- Example 4: Suppose that *n*=8 and the binary representation 1 111 1111B.

  Sign bit is 1 ⇒ negative

  Absolute value is the complement of 111 1111B plus 1, i.e., 000 0000B + 1B = 1D

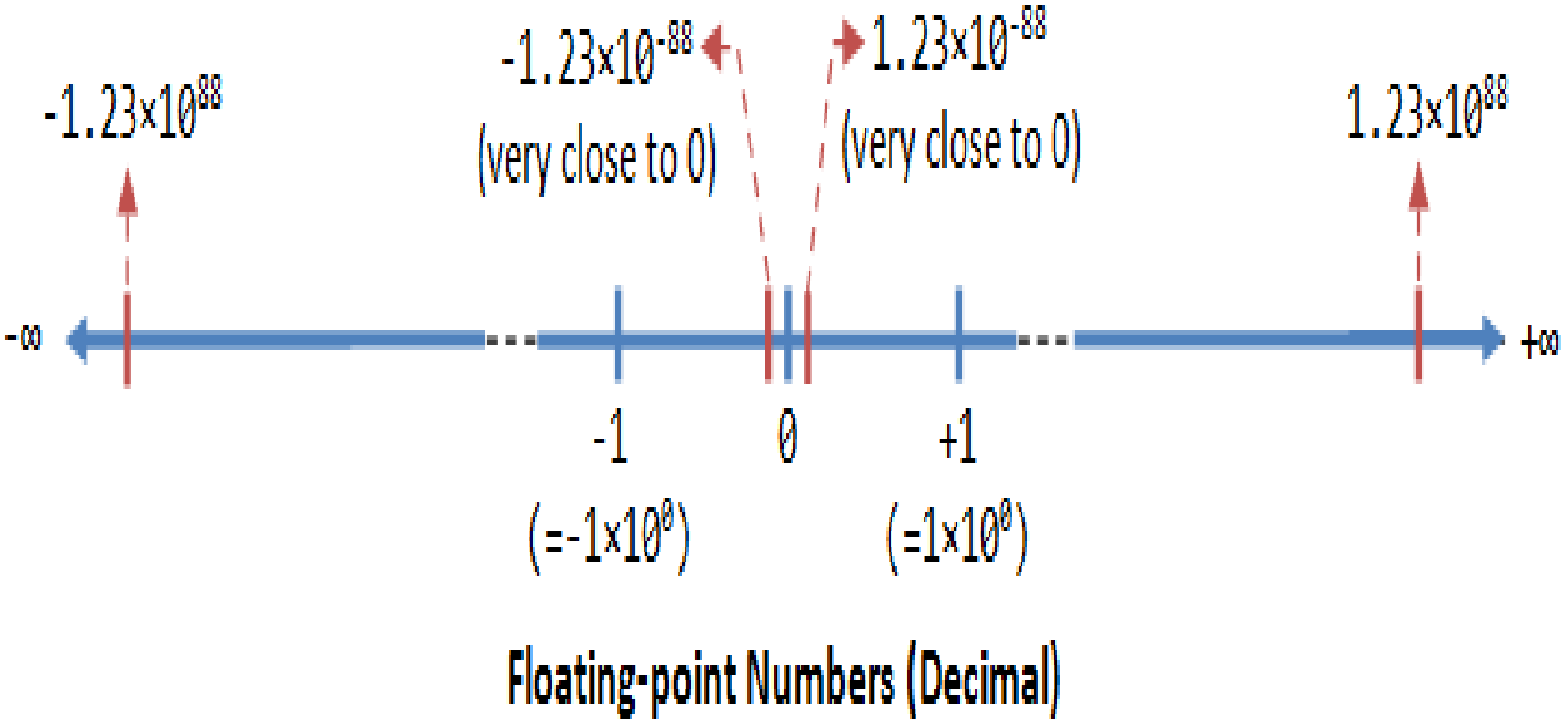
  Hence, the integer is -1D



2's Complement Representation

Floating-Point Number Representation:

- A floating-point number (or real number) can represent a very large (1.23×10^88) or a very small (1.23×10^-88) value.
- It could also represent very large negative number (-1.23×10^88) and very small negative number (-1.23×10^88), as well as zero,



Floating-point Numbers (Decimal)

Ex: The number 55.66 can be represented as 5.566×10^1, 0.5566×10^2, 0.05566×10^3, and so on.

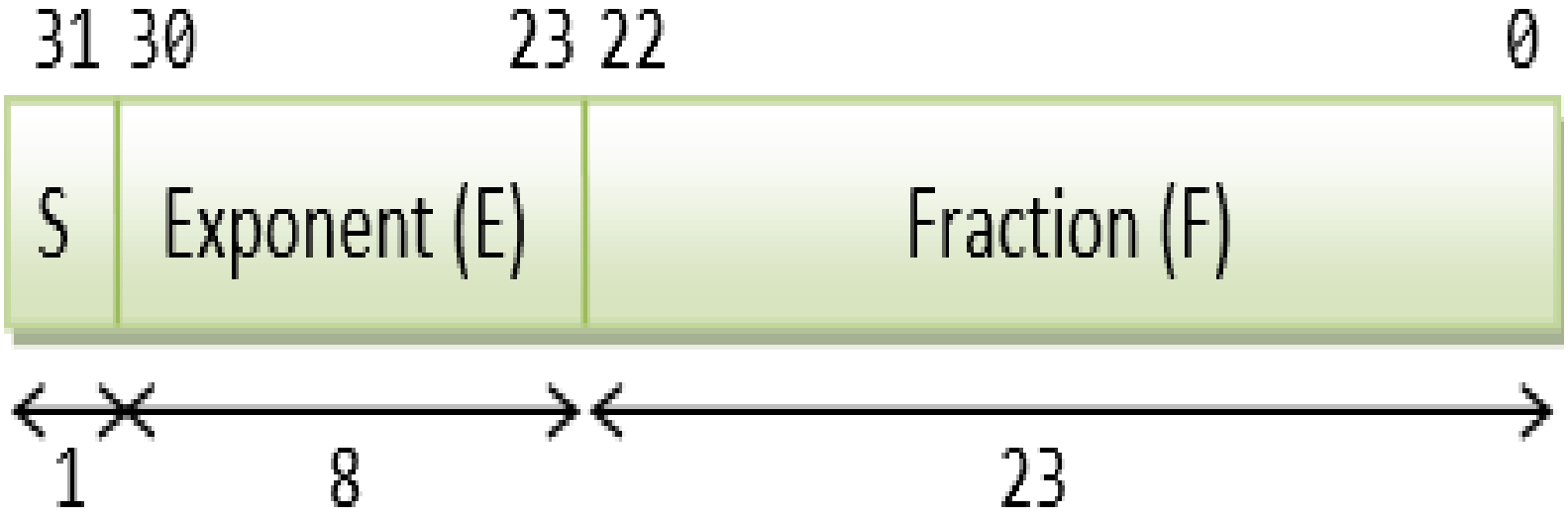
The fractional part can be ***normalized***. In the normalized form, there is only a single non-zero digit before the radix point.

For example, decimal number 123.4567 can be normalized as 1.234567×10^2;

binary number 1010.1011B can be normalized as 1.0101011B×2^3.

IEEE-754 32-bit Single-Precision Floating-Point Numbers

- In 32-bit single-precision floating-point representation:
- The most significant bit is the *sign bit* (S), with 0 for positive numbers and 1 for negative numbers.
- The following 8 bits represent *exponent* (E).
- The remaining 23 bits represents *fraction* (F).



32-bit Single-Precision Floating-point Number

Normalized Form

- Let's illustrate with an example, suppose that the 32-bit pattern is 1 1000 0001 011 0000 0000 0000 0000 0000, with:
- S = 1
- E = 1000 0001
- F = 011 0000 0000 0000 0000 0000

- In the *normalized form*, the actual fraction is normalized with an implicit leading 1 in the form of 1.F. In this example, the actual fraction is 1.011 0000 0000 0000 0000 0000 = 1 + 1×2^-2 + 1×2^-3 = 1.375D.
- The sign bit represents the sign of the number, with S=0 for positive and S=1 for negative number. In this example with S=1, this is a negative number, i.e., -1.375D.

- In normalized form, the actual exponent is E-127 (so-called excess-127 or bias-127). This is because we need to represent both positive and negative exponent.
- With an 8-bit E, ranging from 0 to 255, the excess-127 scheme could provide actual exponent of -127 to 128. In this example, E-127=129-127=2D.
- Hence, the number represented is -1.375×2^2=-5.5D.

Character Representation:

In computer memory, character are "encoded" (or "represented") using a chosen "character encoding schemes”

For example, in ASCII (as well as Latin1, Unicode, and many other character sets):

code numbers 65D (41H) to 90D (5AH) represents 'A' to 'Z', respectively.

code numbers 97D (61H) to 122D (7AH) represents 'a' to 'z', respectively.

code numbers 48D (30H) to 57D (39H) represents '0' to '9', respectively.

Computer Arithmetic:

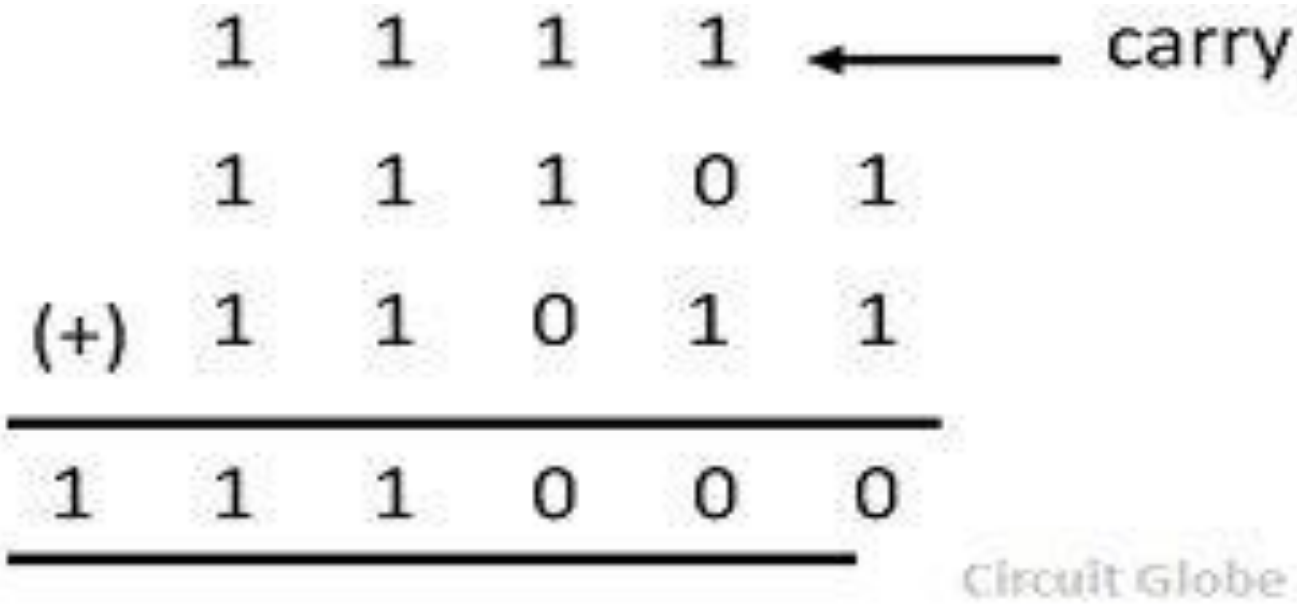
Integer Addition and Subtraction:

- The binary number system uses only two digits 0 and 1 due to which their addition is simple. There are four basic operations for binary addition, as mentioned above.
- 0+0=0
  0+1=1
  1+0=1
  1+1=10

Rules for binary addition :

Case	A + B	Sum	Carry
1	0 + 0	0	0
2	0 + 1	1	0
3	1 + 0	1	0
4	1 + 1	0	1

Example for binary Addition:



Binary Addition

Addition Rules:

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

Example 1:

```
  1
  0 1 0 1
+ 0 1 1 0
---------
  1 0 1 1
```

Example 2:

```
    1 1
  1 0 1 1 1
+ 1 0 0 1 0
-----------
1 0 1 0 0 1
```

OVERFLOW

Binary subtraction rules:

Procedures for Binary Subtraction by 1's Complement

Write the 1's complement of the subtrahend

Then add the 1's complement subtrahend with the minuend

If the result has a carryover, then add that carry over in the least significant bit

If there is no carryover, then take the 1's complement of the resultant, and it is negative.

Binary Subtraction Questions Using 1's Compement

Question 1:

$(110101)_2 - (100101)_2$

Solution:

$(1\ 1\ 0\ 1\ 0\ 1)_2 = 53_{10}$

$(1\ 0\ 0\ 1\ 0\ 1)_2 = 37_{10}$ – subtrahend

Now take the 1's complement of the subtrahend and add with minuend.

1 carry

1 1 0 1 0 1

(+) 0 1 1 0 1 0

———————

0 0 1 1 1 1

1 carry

———————

0 1 0 0 0 0

Therefore, the solution is 010000

$(010000)_2 = 16_{10}$

Binary representation of 5 is: 0 1 0 1

1's Complement of 5 is: 1 0 1 0

2's Complement of 5 is: (1's Complement + 1) i.e.

1 0 1 0 (1's Compliment)

+ 1

1 0 1 1 (2's Complement i.e. -5)

Binary multiplication Shift and Add:

```
        1 0 0 1
      x 1 0 1
-------------
        1 0 0 1
      0 0 0 0
+   1 0 0 1
-------------
  1 0 1 1 0 1
```

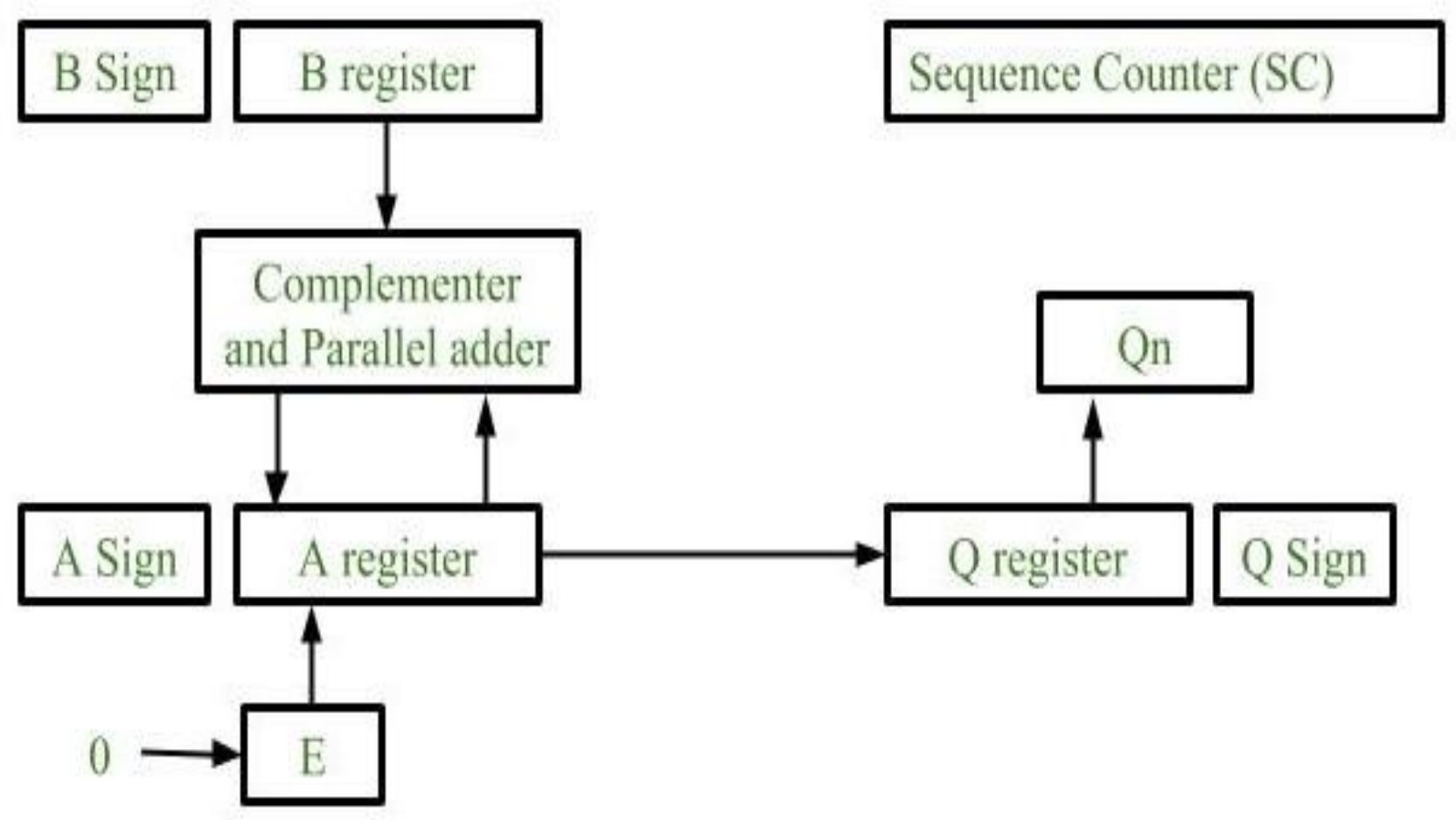
Example:

0011010 x 001100 = 100111000

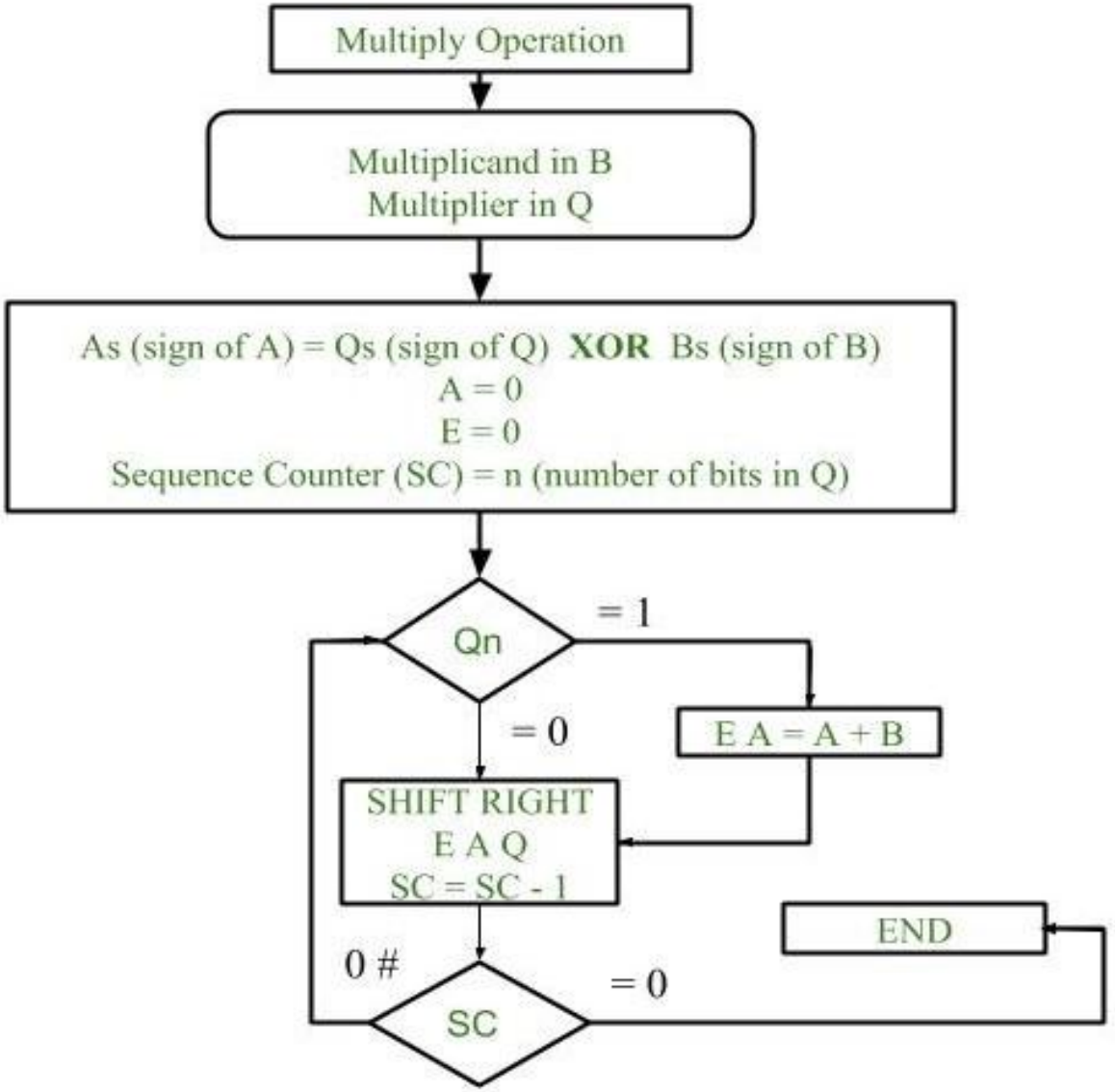
$$
\begin{array}{rl}
0011010 & = 26_{10} \\
\times 0001100 & = 12_{10} \\
\hline
0000000 & \\
0000000 & \\
0011010 & \\
0011010 & \\
\hline
0100111000 & = 312_{10}
\end{array}
$$

https://www.geeksforgeeks.org/multiplication-algorithm-in-signed-magnitude-representation/

Unsigned multiplication:



Flow chart

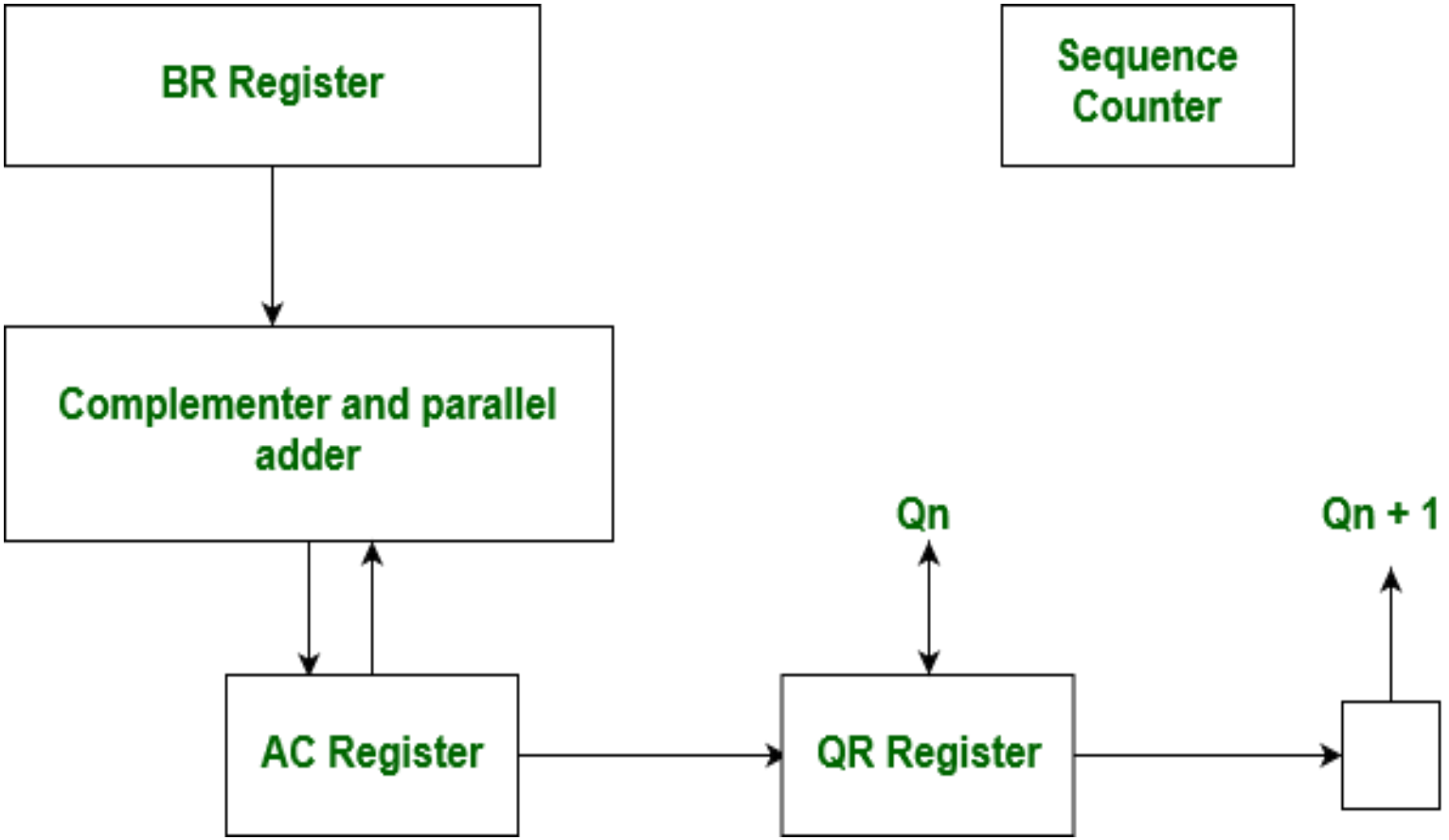

Multiply Operation
Multiplicand in B
Multiplier in Q
As (sign of A) = Qs (sign of Q) XOR Bs (sign of B)
A = 0
E = 0
Sequence Counter (SC) = n (number of bits in Q)
Qn
= 1
= 0
E A = A + B
SHIFT RIGHT
E A Q
SC = SC - 1
END
0 #
SC
= 0

Example

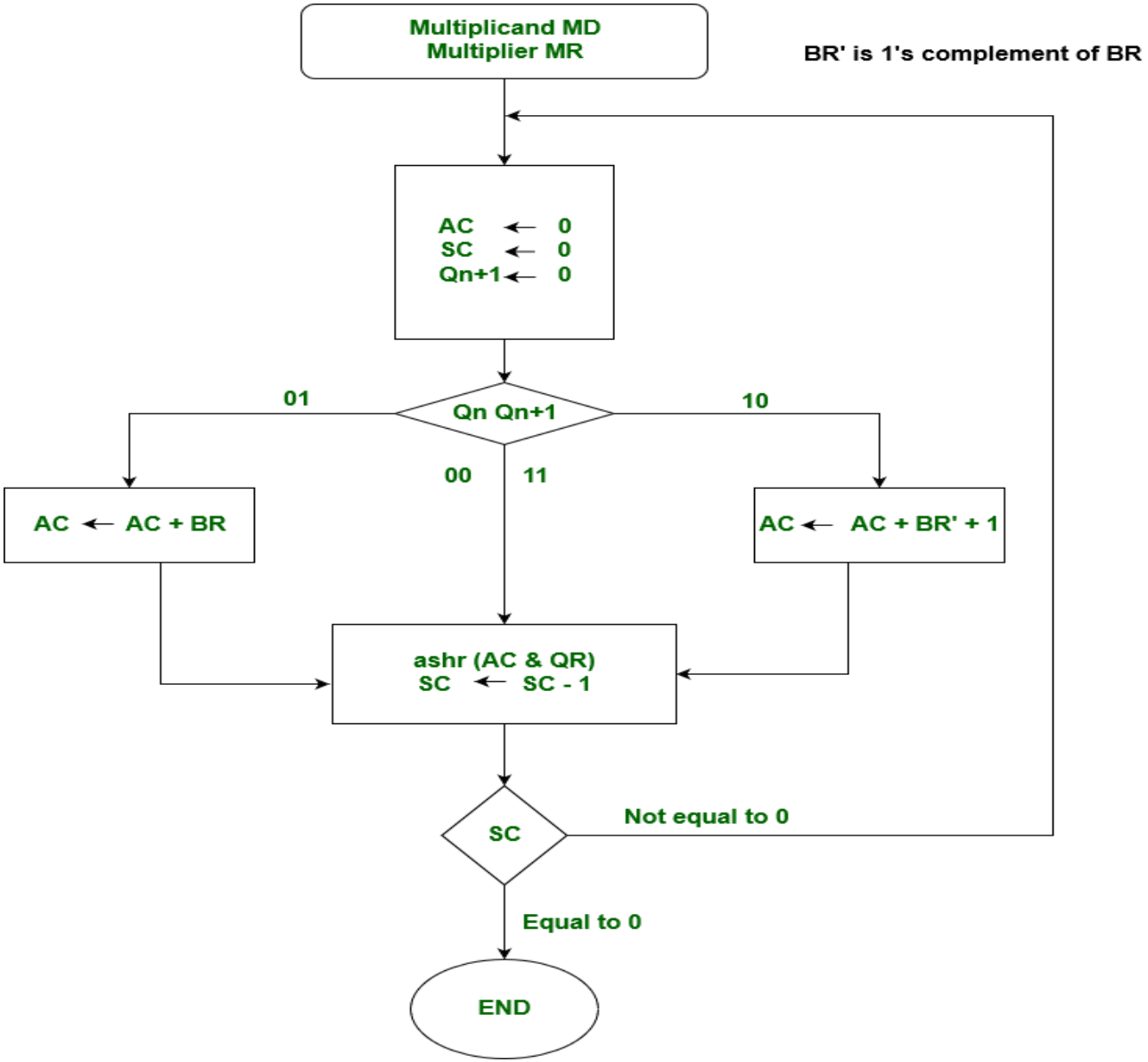
Multiplicand B = 10111	E	A	Q	SC
Multiplier in Q	0	00000	10011	101
Qn = 1; add B		10111		
First partial product	0	10111		
Shift right EAQ	0	01011	11001	100
Qn = 1; add B		10111		
Second partial product	1	00010		
Shift right EAQ	0	10001	01100	011
Qn = 0; shift right EAQ	0	01000	10110	010
Qn = 0; shift right EAQ	0	00100	01011	001
Qn = 1; add B		10111		
Fifth partial product	0	11011		
Shift right EAQ	0	01101	10101	000

Final product in AQ
0110110101

Booth multiplication



Flow chart


Multiplicand MD
Multiplier MR
BR' is 1's complement of BR
AC ← 0
SC ← 0
Qn+1 ← 0
Qn Qn+1
01
10
00
11
AC ← AC + BR
AC ← AC + BR' + 1
ashr (AC & QR)
SC ← SC - 1
SC
Not equal to 0
Equal to 0
END

- Example – A numerical example of booth's algorithm is shown below for n = 4. It shows the step by step multiplication of -5 and -7.
- MD = -5 = 1011, MD = 1011, MD'+1 = 0101
- MR = -7 = 1001
- The explanation of first step is as follows: Qn+1
- AC = 0000, MR = 1001, Qn+1 = 0, SC = 4
- Qn Qn+1 = 10
- So, we do AC + (MD)'+1, which gives AC = 0101
- On right shifting AC and MR, we get
- AC = 0010, MR = 1100 and Qn+1 = 1

Example: Multiply the two numbers 7 and 3 by using the Booth's multiplication algorithm.

- **Ans**. Here we have two numbers, 7 and 3. First of all, we need to convert 7 and 3 into binary numbers like 7 = (0111) and 3 = (0011).
- Now set 7 (in binary 0111) as multiplicand (M) and 3 (in binary 0011) as a multiplier (Q). And SC (Sequence Count) represents the number of bits, and here we have 4 bits, so set the SC = 4.

- Also, it shows the number of iteration cycles of the booth's algorithms and then cycles run SC = SC - 1 time.
- Q_n Q_{n+1} M = (0111)

Division:

- They are generally of two types of division algorithm -

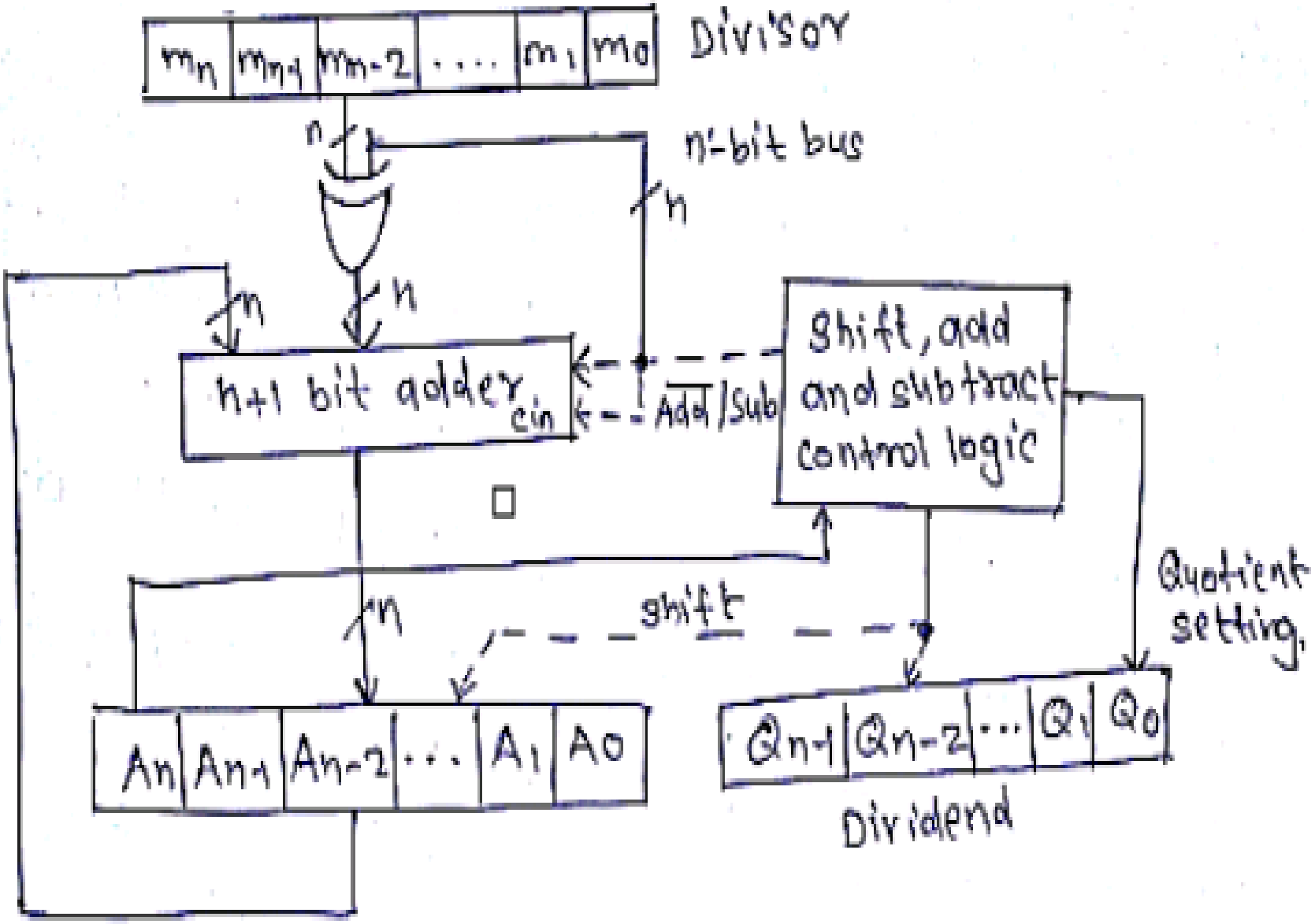
  Restoring division algorithm

  Non-restoring division algorithms

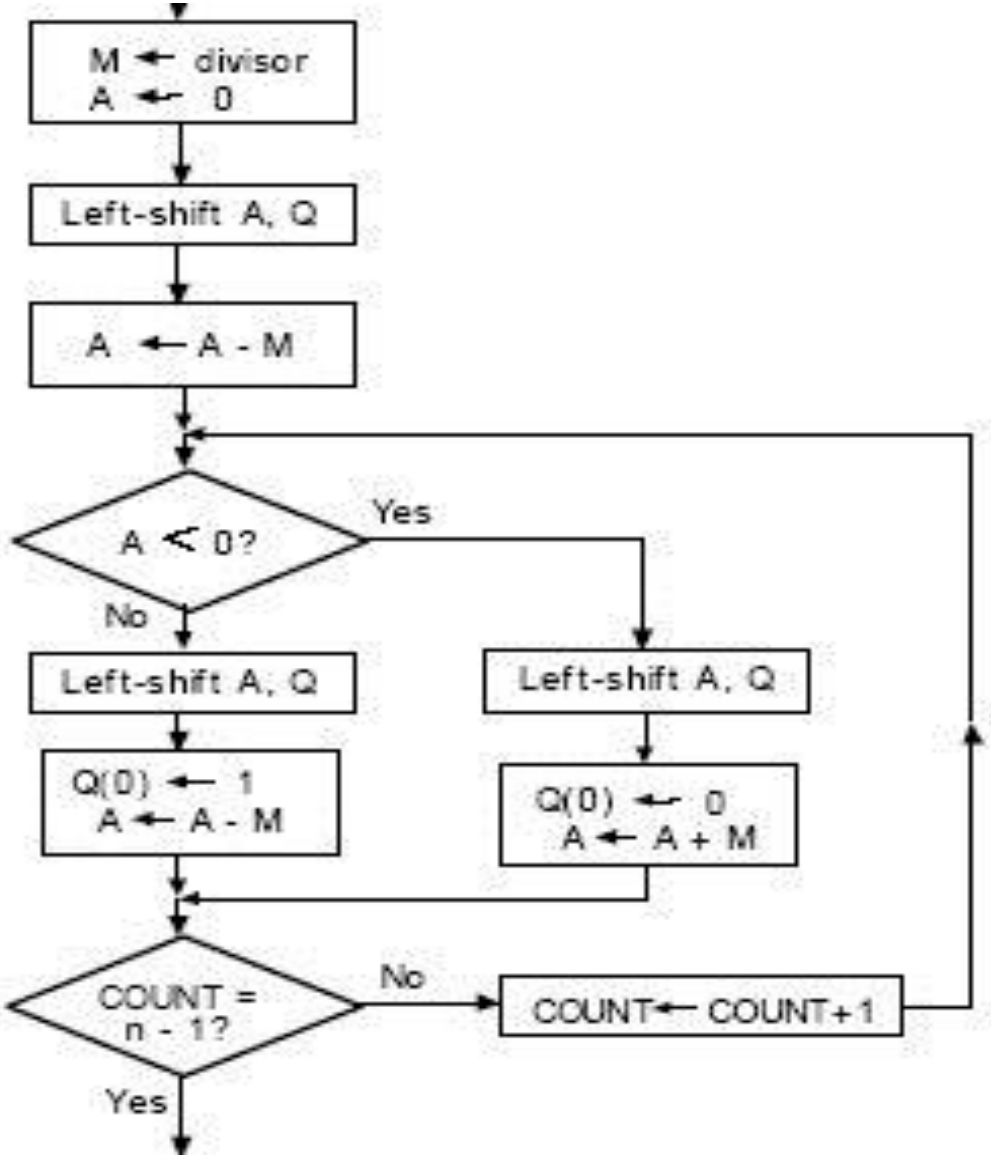
Restoring Division Algorithm For Unsigned Integer

- A division algorithm provides a quotient and a remainder when we divide two number.

Hardware implementation



- Flow chart:


M ← divisor
A ← 0
Left-shift A, Q
A ← A - M
A < 0?
Yes
No
Left-shift A, Q
Left-shift A, Q
Q(0) ← 1
A ← A - M
Q(0) ← 0
A ← A + M
COUNT = n - 1?
No
COUNT ← COUNT+1
Yes
Quotient in Q

Steps in restoring division algorithms:

Step-1: First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = number of bits in dividend)

Step-2: Then the content of register A and Q is shifted right as if they are a single unit

Step-3: Then content of register M is subtracted from A and result is stored in A

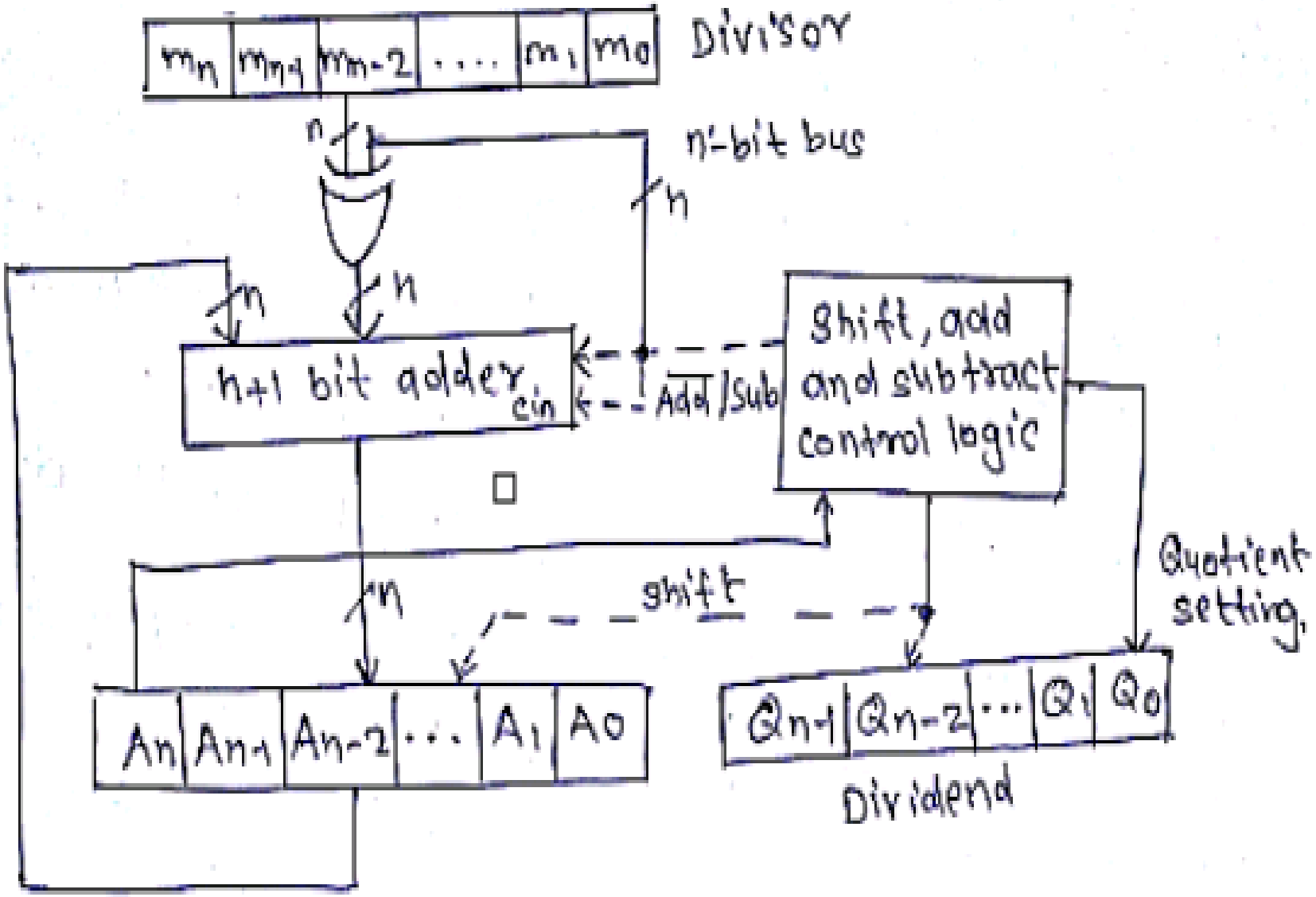
Step-4: Then the most significant bit of the A is checked if it is 0 the least significant bit of Q is set to 1 otherwise if it is 1 the least significant bit of Q is set to 0 and value of register A is restored i.e the value of A before the subtraction with M

Step-5: The value of counter n is decremented

Step-6: If the value of n becomes zero we get of the loop otherwise we repeat fro step 2

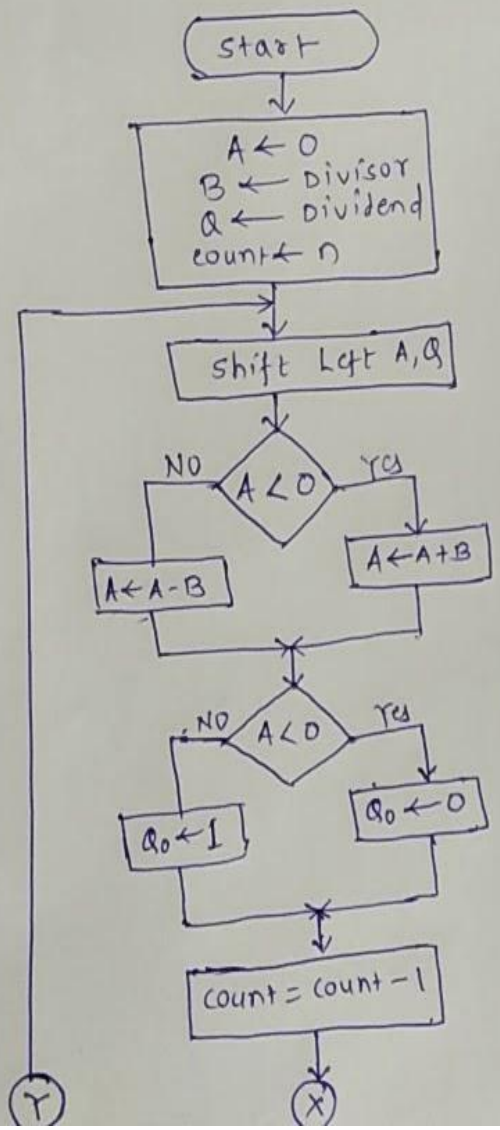
Step-7: Finally, the register Q contain the quotient and A contain remainder

Non restoring – division
Hardware implementation



Non-Restoring Division &

Flowchart-



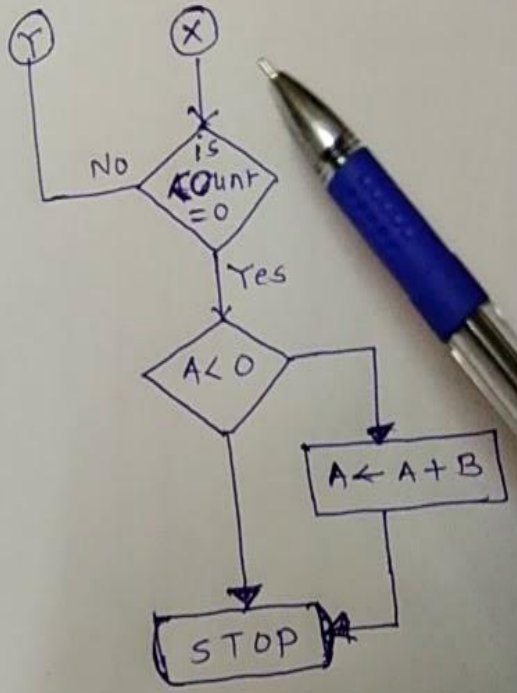


Fig:- Flowchart for non-restoring division operation

Example : Non-restoring Division

Dividend = 1011 ⇒ Q n=4

Divisor = 0101 ⇒ B = 00101 ⇒ $\bar{B}+1$ = 11011

Operation	A	Q	
Initial	00000	1011	1st cycle
shift	00001	011 □	
subtract	11011		
	11100	011 0	
set Q_0			
shift	11000	110 □	2nd cycle
Addition	00101		
	11101	110 0	
set Q_0			
shift	11011	100 □	3rd cycle
Addition	00101		
	00000	100 1	
set Q_0			
shift	00001	001 □	4th cycle
+ subtract	11011		
	11100	001 0	
set Q_0			

A Q

op.	A	Q
	11100	0010
Addition	00101	= Q
	00001	

A

= Remainder

FLOATING POINT ARITHMETIC:

Addition and Subtraction

To understand floating point addition, first we see addition of real numbers in decimal as same logic is applied in both cases.

For example, we have to add **$1.1 * 10^3$** and **50.**

We cannot add these numbers directly.

After aligning exponent, we get **$50 = 0.05 * 10^3$**

Now adding significand, **$0.05 + 1.1 = 1.15$**

So, finally we get **$(1.1 * 10^3 + 50) = 1.15 * 10^3$**

Here, notice that we shifted **50** and made it **0.05**

to add these numbers.

Web references:

https://www.studytonight.com

https://www.javatpoint.com

https://www.tutorialspoint.com/

https://www.youtube.com/results?search_query=unsigned+multiplication+

UNIT- III

Register Transfer and Micro operations: Register Transfer Language, Register Transfer, Bus and Memory Transfers: Arithmetic Micro operations, Logic Micro operations, Shift Micro operations.

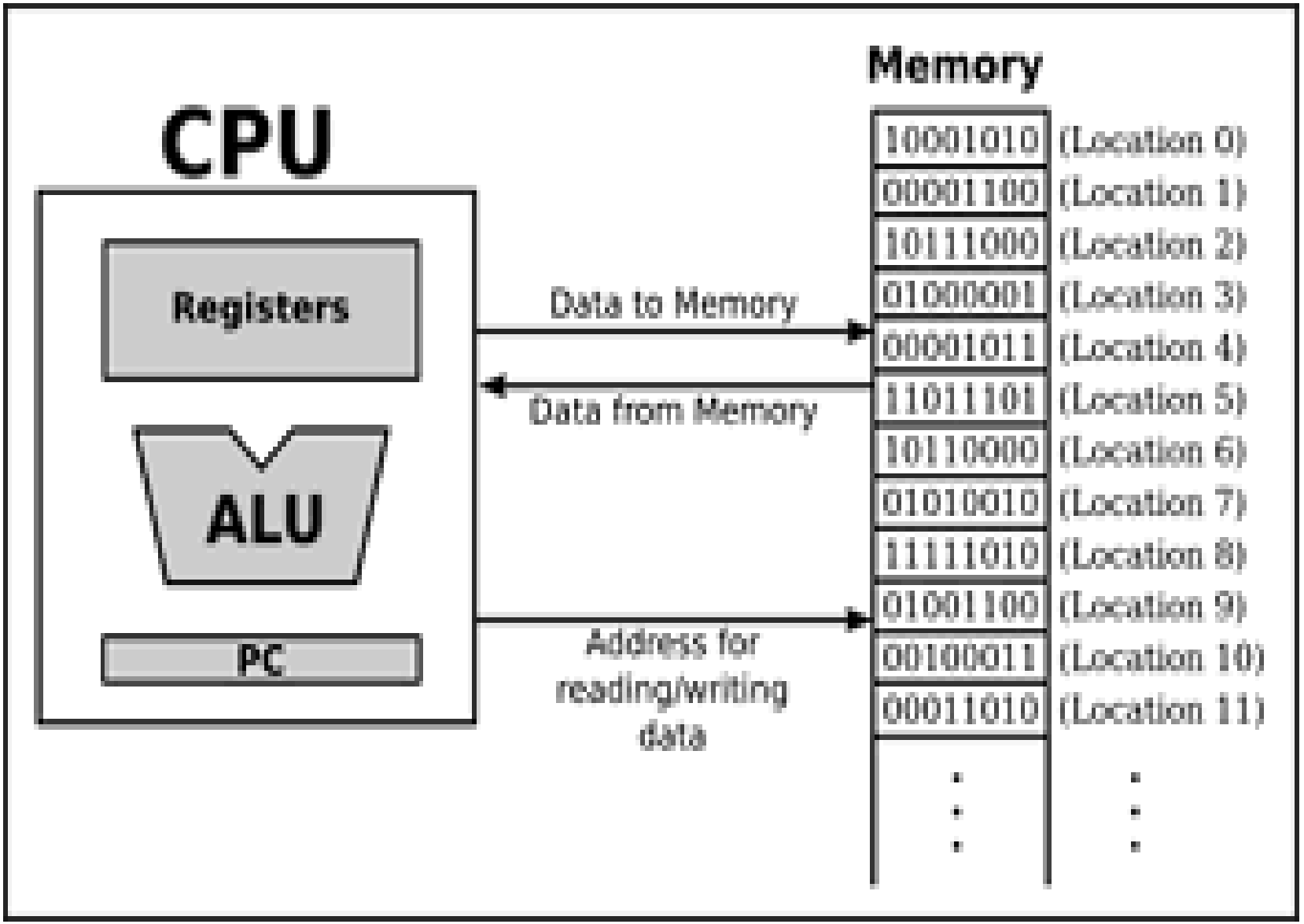
CPU control unit design: Hardwired control, Micro-programmed control, Address Sequencing.

Registers in Computer Architecture

Register is a very fast computer memory, used to store data/instruction in-execution.

A **Register** is a group of flip-flops with each flip-flop capable of storing **one bit** of information.

An *n-bit* register has a group of *n flip-flops* and is capable of storing binary information of *n-bits*.


CPU
Registers
ALU
PC
Data to Memory
Data from Memory
Address for reading/writing data
Memory
10001010 (Location 0)
00001100 (Location 1)
10111000 (Location 2)
01000001 (Location 3)
00001011 (Location 4)
11011101 (Location 5)
10110000 (Location 6)
01010010 (Location 7)
11111010 (Location 8)
01001100 (Location 9)
00100011 (Location 10)
00011010 (Location 11)

A register consists of a group of flip-flops and gates.

The flip-flops hold the binary information and gates control when and how new information is transferred into a register.

Various types of registers are available commercially.

The simplest register is one that consists of only flip-flops with no external gates.

Loading the Registers

- The transfer of new information into a register is referred to as loading the register.

- If all the bits of register are loaded simultaneously with a common clock pulse then the loading is said to be done in parallel.

Register Transfer Language

- The symbolic notation used to describe the micro-operation transfers amongst registers is called **Register transfer language**.

- The term **register transfer** means the availability of **hardware logic circuits** that can perform a stated micro-operation and transfer the result of the operation to the same or another register.

Following are some commonly used registers:

Accumulator: This is the most common register, used to store data taken out from the memory.

General Purpose Registers: This is used to store data intermediate results during program execution. It can be accessed via assembly programming.

Special Purpose Registers: Users do not access these registers. These registers are used by Computer system.

Special Purpose Registers:

MAR: Memory Address Register are those registers that holds the address for memory unit.

MBR: Memory Buffer Register stores instruction and data received from the memory and sent from the memory.

PC: Program Counter points to the next instruction to be executed.

IR: Instruction Register holds the instruction to be executed.

Register Transfer

Information transferred from one register to another is designated in symbolic form by means of replacement operator.

R2 ← R1

The transfer to occur only in predetermined control condition.

This can be shown by following **if-then** statement: if (P=1) then (R2 ← R1)

Control Function

A control function is a Boolean variable that is equal to 1 or 0. The control function is shown as:

P: R2 ← R1

The control condition is terminated with a colon. It shows that transfer operation can be executed only if P=1.

Micro-Operations

The operations executed on data stored in registers are called micro-operations.

A micro-operation is an elementary operation performed on the information stored in one or more registers.

Example: Shift, count, clear and load.

Types of Micro-Operations

- The micro-operations in digital computers are of 4 types:

1 Register transfer micro-operations transfer binary information from one register to another.

2 Arithmetic micro-operations perform arithmetic operations on numeric data stored in registers.

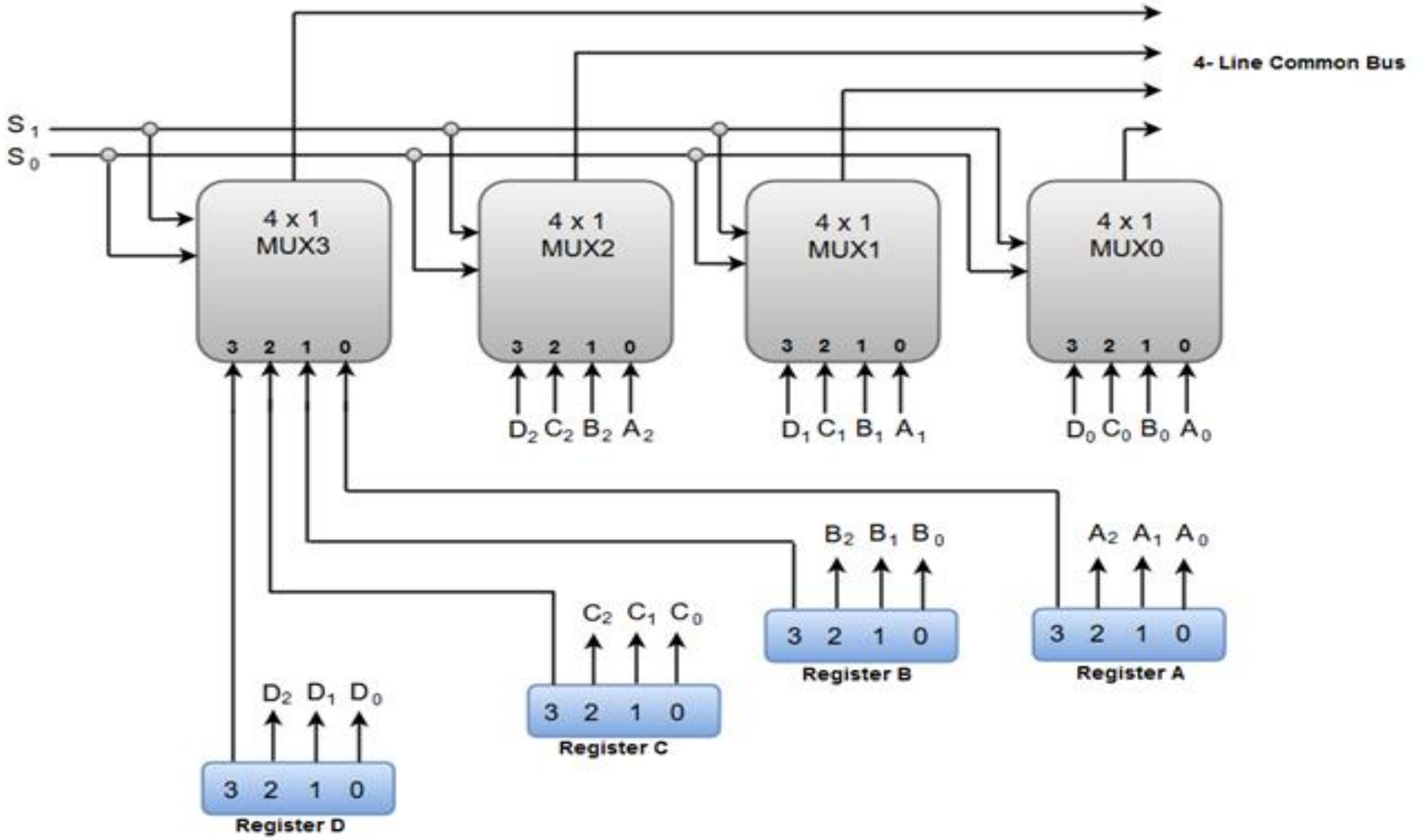
3. Logic micro-operations performs bit manipulation operation on non-numeric data stored in registers.

4. Shift micro-operations perform shift micro-operations performed on data.

Bus and memory transfer

- A digital system composed of many registers, and paths must be provided to transfer information from one register to another.
- A bus structure is more efficient for transferring information between registers in a multi-register configuration system.

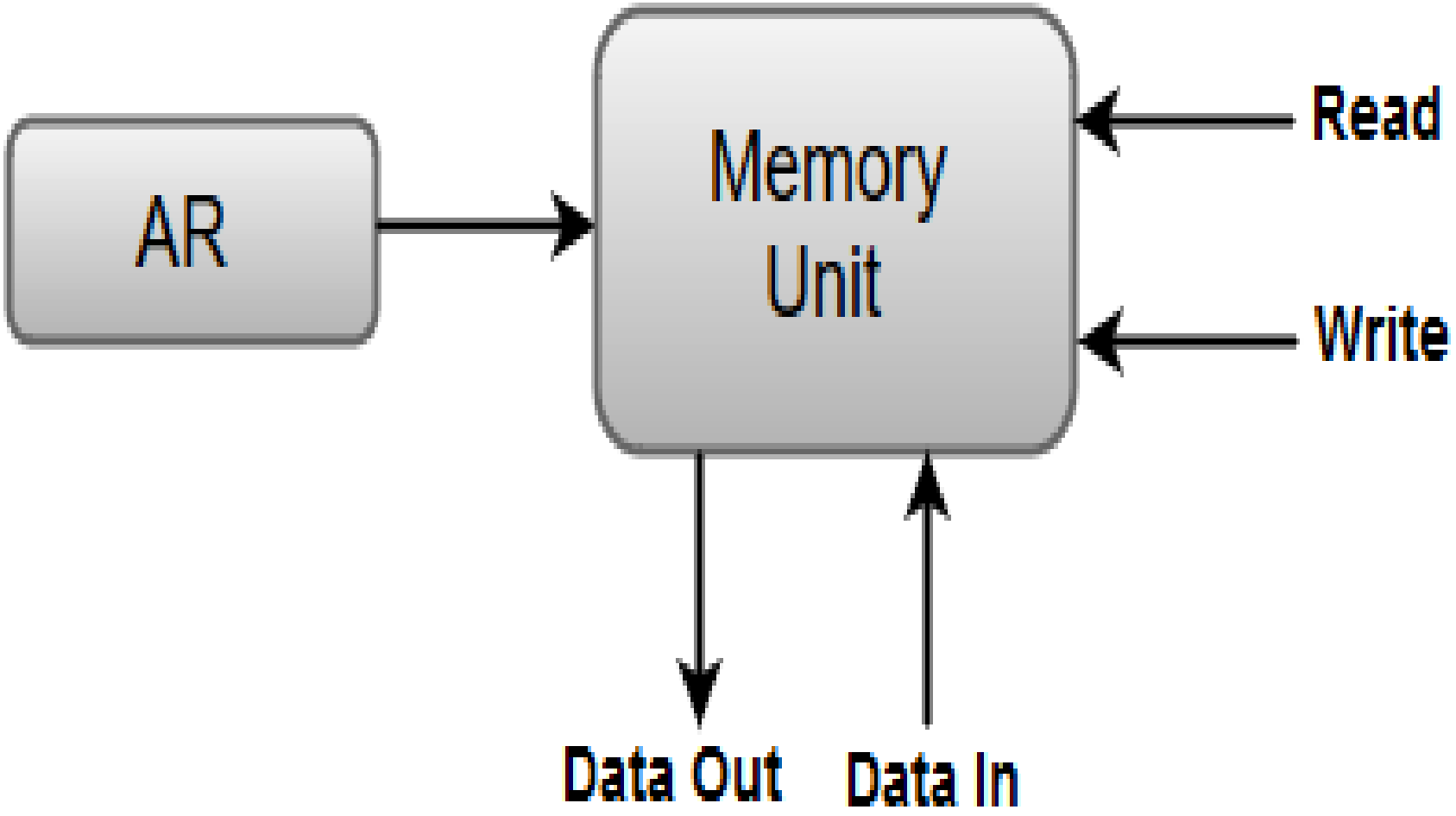
Bus System for 4 Registers:



Memory Transfer

- The transfer of information from a memory unit to the user end is called a **Read** operation.
- The transfer of new information to be stored in the memory is called a **Write** operation.
- A memory word is designated by the letter **M**.

- **Read:** DR ← M [AR]
- The **Read** statement causes a transfer of information into the data register (DR) from the memory word (M) selected by the address register (AR).
- And the corresponding write operation can be stated as:
- **Write**: M [AR] ← R1
- The **Write** statement causes a transfer of information from register R1 into the memory word (M) selected by address register (AR).


AR
Memory Unit
Read
Write
Data Out
Data In

Arithmetic Micro-Operations:

Let us again take an example:

R3 → R1 + R2' + 1

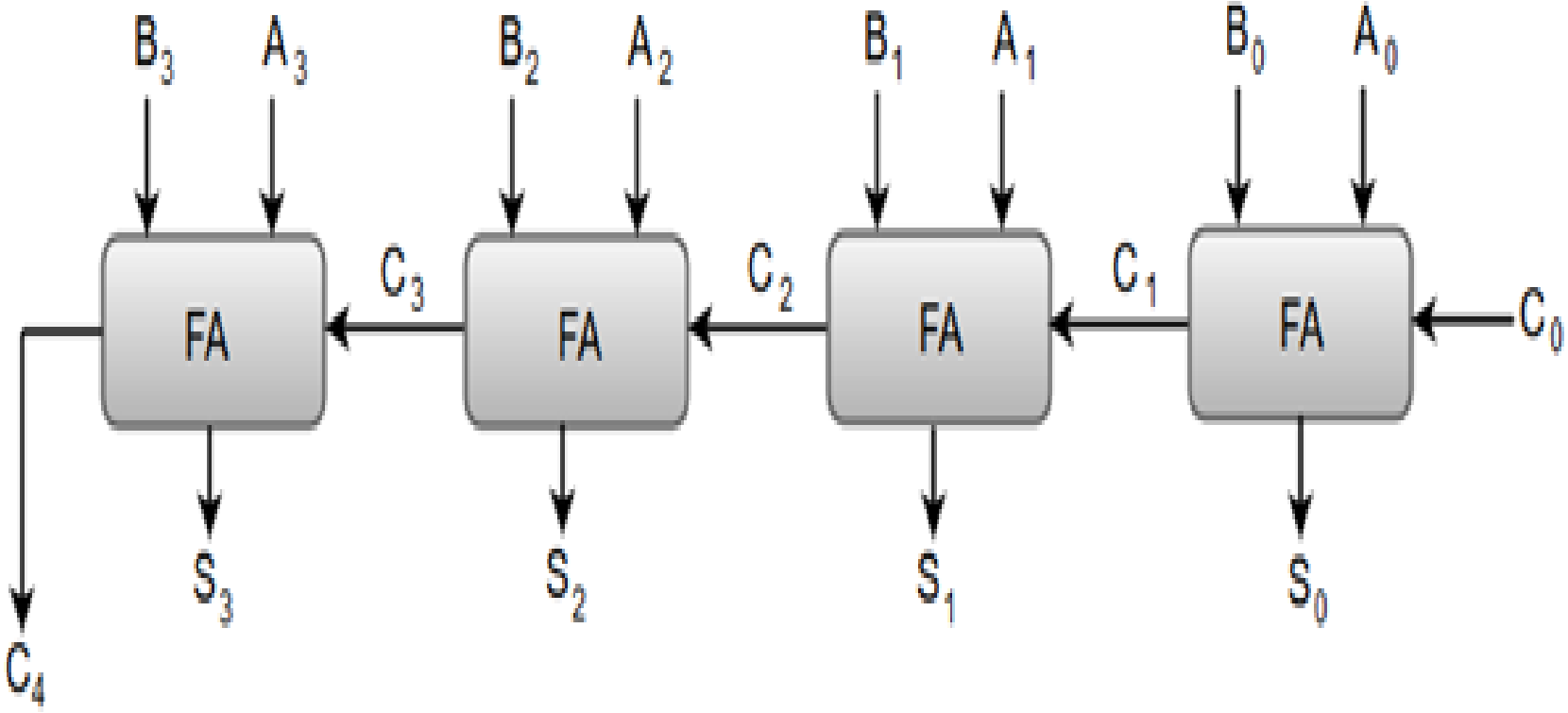
In subtract micro-operation, instead of using minus operator we take **1's compliment** and add 1 to the register which gets subtracted, i.e **R1 - R2** is equivalent to **R3 → R1 + R2' + 1**

Increment/Decrement Micro-Operation

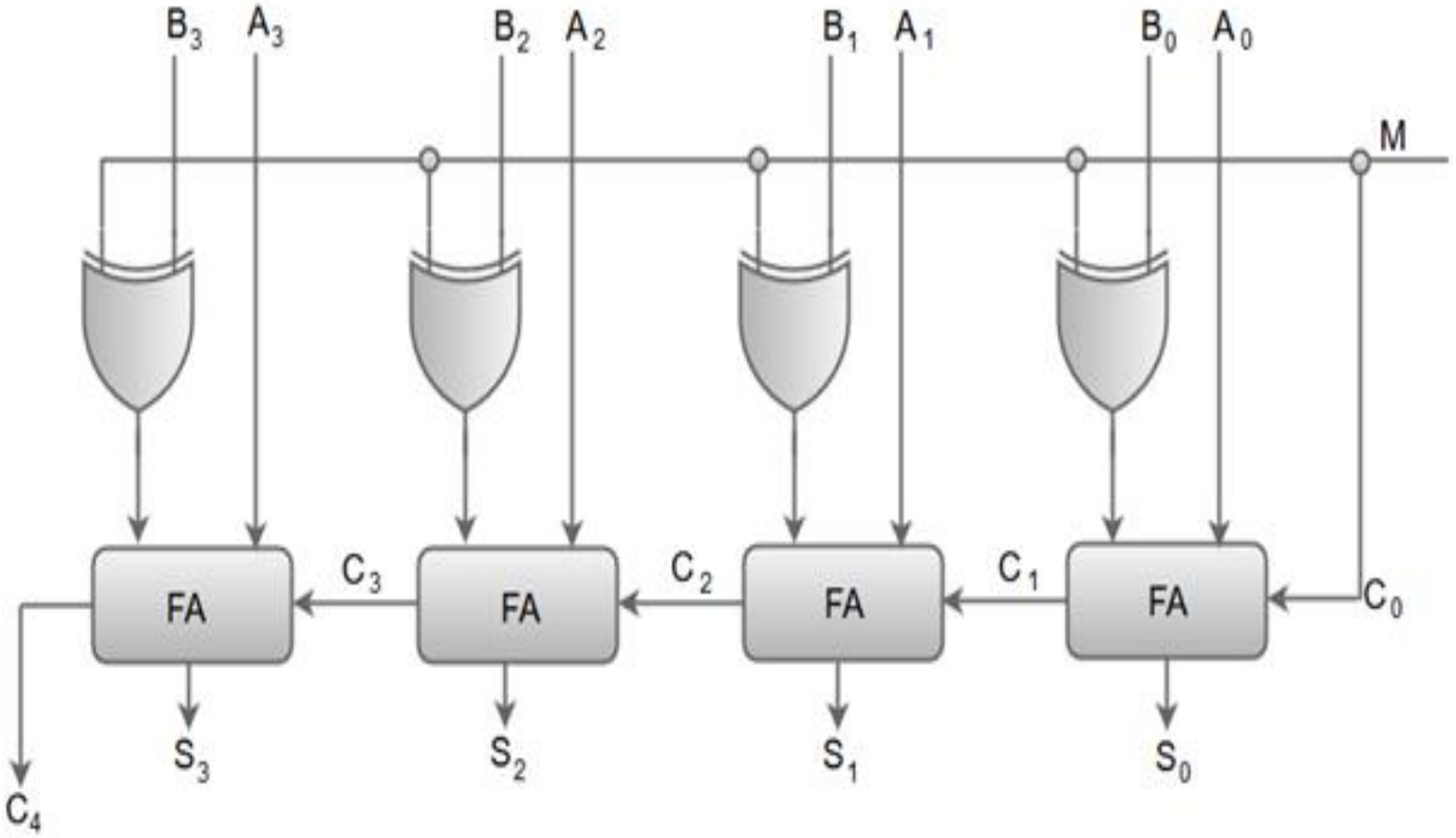
- $R1 \rightarrow R1 + 1$
- $R1 \rightarrow R1 - 1$
- **Arithmetic operations:**

Symbolic Designation	**Description**
R3 ← R1 + R2	Contents of R1+R2 transferred to R3.
R3 ← R1 - R2	Contents of R1-R2 transferred to R3.
R2 ← (R2)‘	Compliment the contents of R2.
R2 ← (R2)' + 1	2's compliment the contents of R2.
R3 ← R1 + (R2)' + 1	R1 + the 2's compliment of R2 (subtraction).
R1 ← R1 + 1	Increment the contents of R1 by 1.
R1 ← R1 – 1	Decrement the contents of R1 by 1.

4 bit binary adder:



4 bit adder-subtractor:



Logic Micro-Operations:

These are binary micro-operations performed on the bits stored in the registers. These operations consider each bit separately and treat them as binary variables.

Let us consider the X-OR micro-operation with the contents of two registers R1 and R2.

P: R1 ← R1 X-OR R2

- Assume that each register has 3 bits.
- Let the content of R1 be **010** and R2 be **100**. The X-OR micro-operation will be:

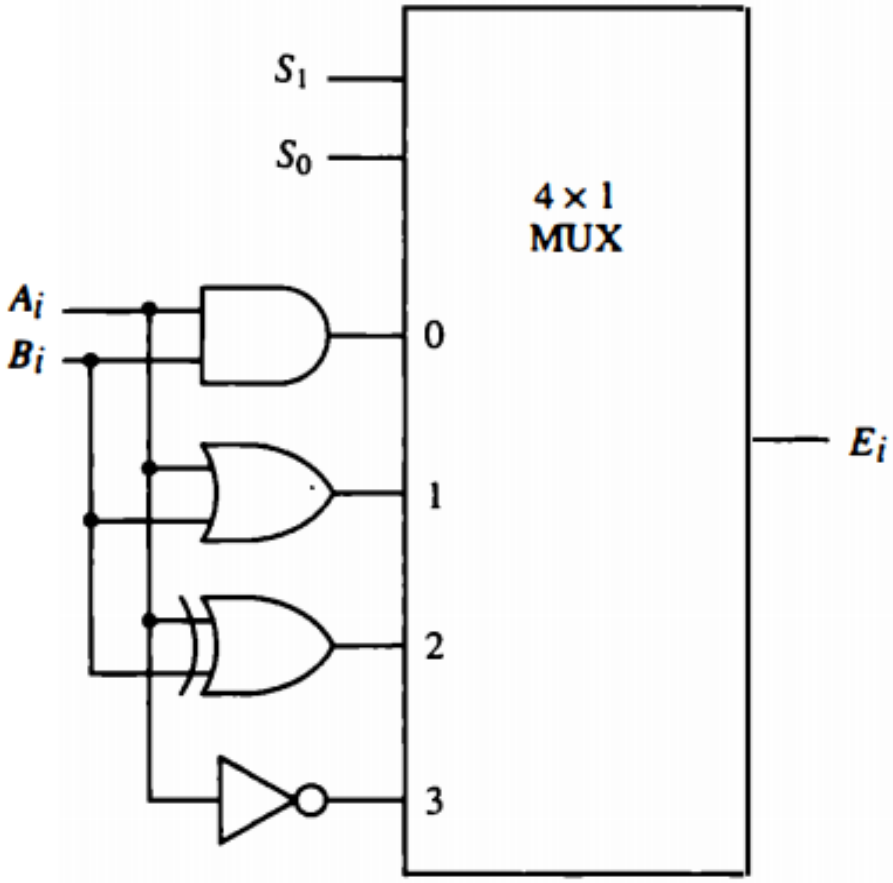
$$010 \rightarrow R1$$

$$\underline{100 \rightarrow R2}$$

$$\underline{110} \rightarrow R1 \text{ after } P=1$$

- Logical operations:

Figure One stage of logic circuit.



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

(b) Function table

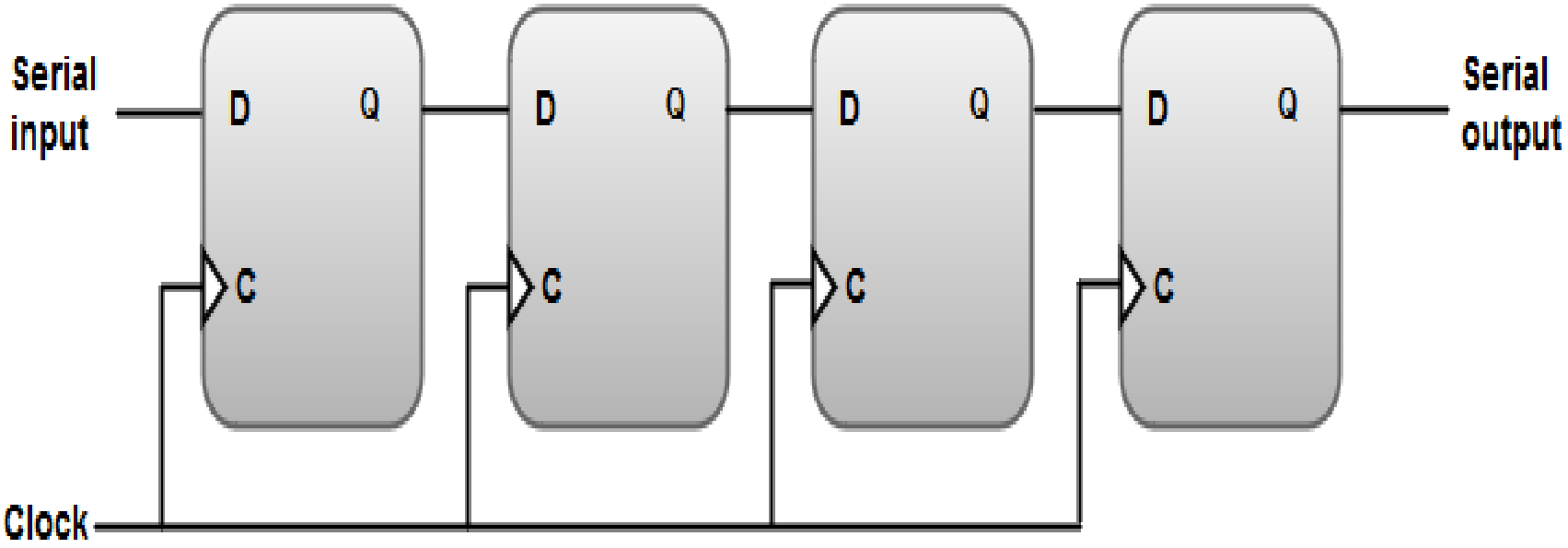
Shift – Micro operations:

Shift - Registers are capable of shifting their binary information in one or both directions.

The logical configuration of a Shift - Register consists of a series of flip-flops, with the output of one flip-flop connected to the input of the next flip-flop.

The following image shows the block diagram of a Shift - Register and its configuration.

4 - bit Shift-Register:



The basic configuration of a Shift - Register contains the following points:

The most general Shift - Registers are often referred to as **Bidirectional Shift Register with parallel load**.

A **common clock** is connected to each register in series to synchronize all operations.

A **serial input** line is associated with the left-most register, and a serial output line is associated with the right-most register.

CPU control unit design:

The Control Unit is classified into two major categories:

Hardwired Control

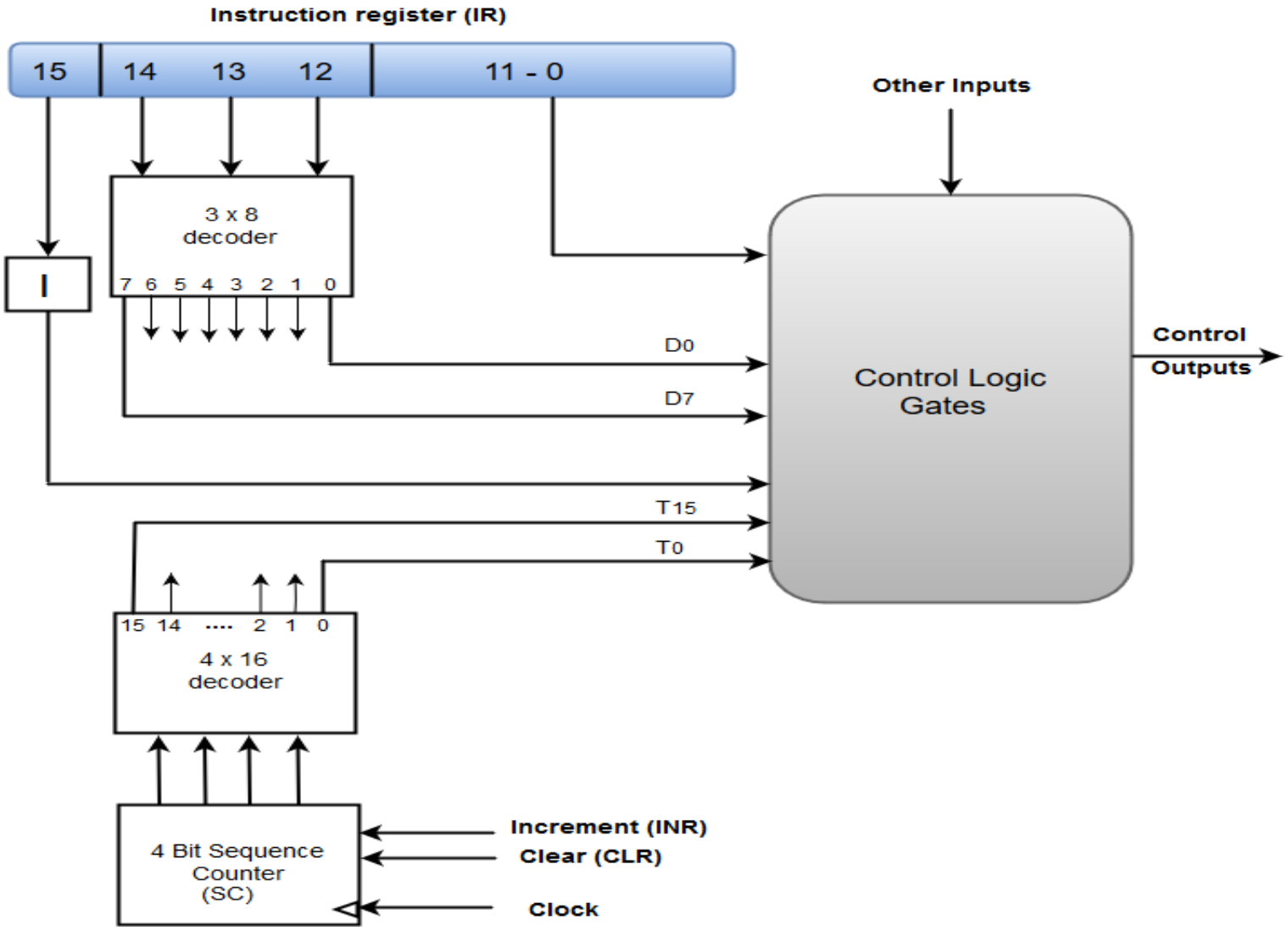
Micro programmed Control

Hardwired Control:

The Hardwired Control organization involves the control logic to be implemented with gates, flip-flops, decoders, and other digital circuits.

Hardwired Control

Control Unit of a Basic Computer:



Hard-wired Control consists of two decoders, a sequence counter, and a number of logic gates.

An instruction fetched from the memory unit is placed in the instruction register (IR).

The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.

The operation code in bits 12 through 14 are coded with a 3 x 8 decoder.

The outputs of the decoder are designated by the symbols D0 through D7.

The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.

The operation codes from Bits 0 through 11 are applied to the control logic gates.

The Sequence counter (SC) can count in binary from 0 through 15.

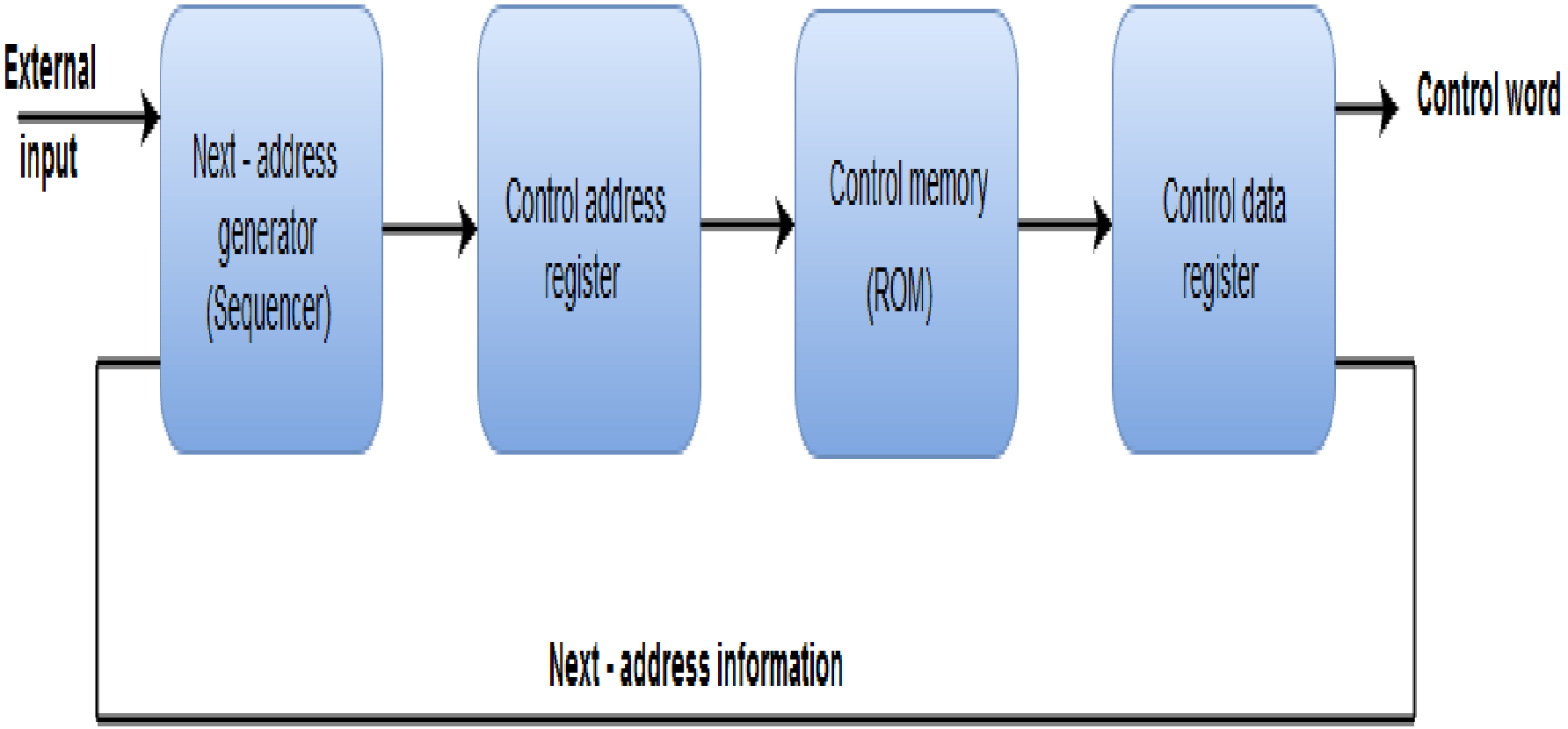
Micro-programmed Control

The Micro programmed Control organization is implemented by using the programming approach.

In Micro programmed Control, the micro-operations are performed by executing a program consisting of micro-instructions.

The following image shows the block diagram of a Micro programmed Control organization.

Microprogrammed Control Unit of a Basic Computer:



Address Sequencing

A program residing in the memory unit of a computer consists of a sequence of instructions. These instructions are executed by the processor by going through a cycle for each instruction.

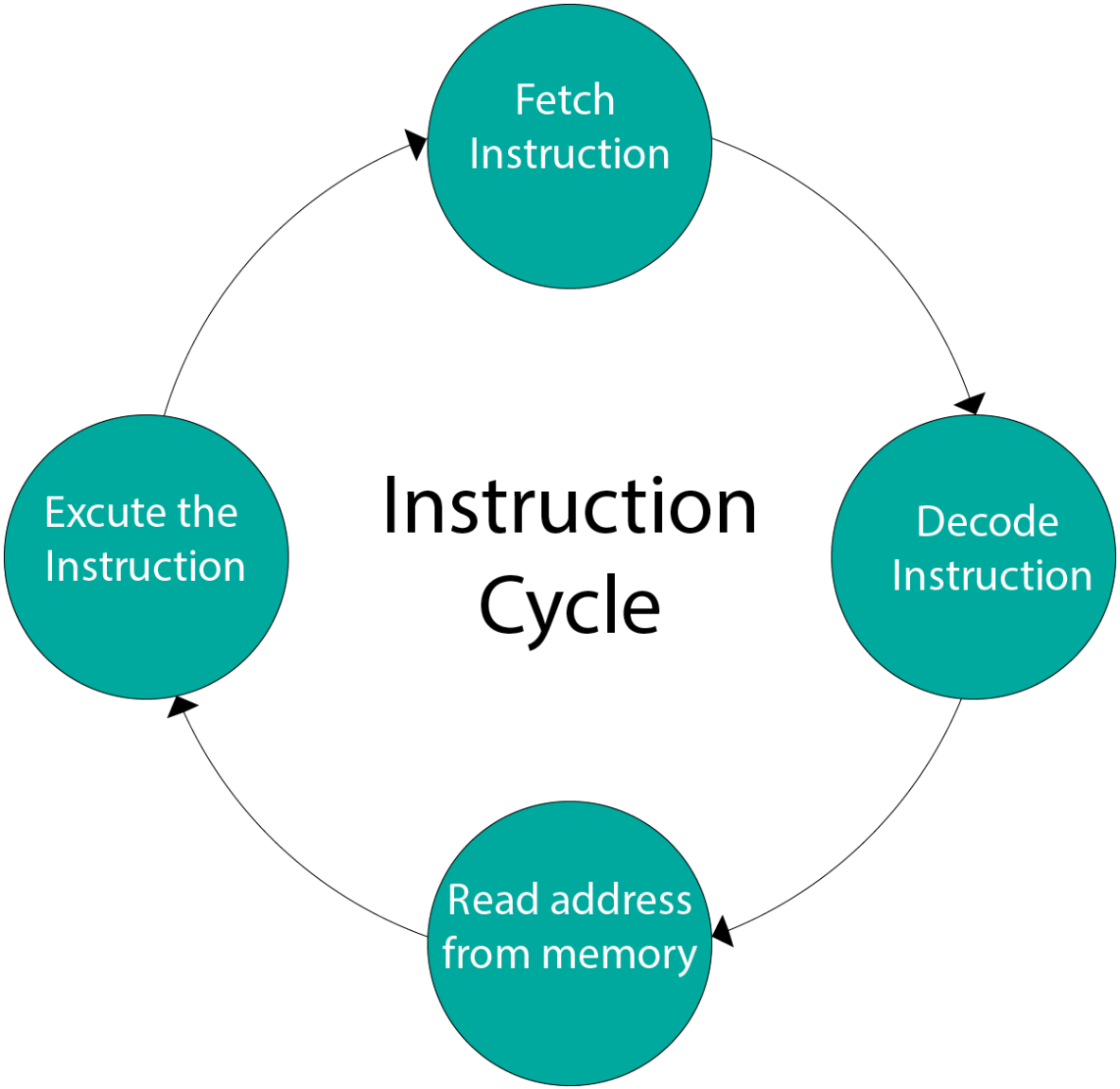
In a basic computer, each instruction cycle consists of the following phases:

Fetch instruction from memory.

Decode the instruction.

Read the effective address from memory.

Execute the instruction.


Fetch Instruction
Decode Instruction
Read address from memory
Excute the Instruction
Instruction Cycle

Web references:

https://www.studytonight.com

https://www.javatpoint.com

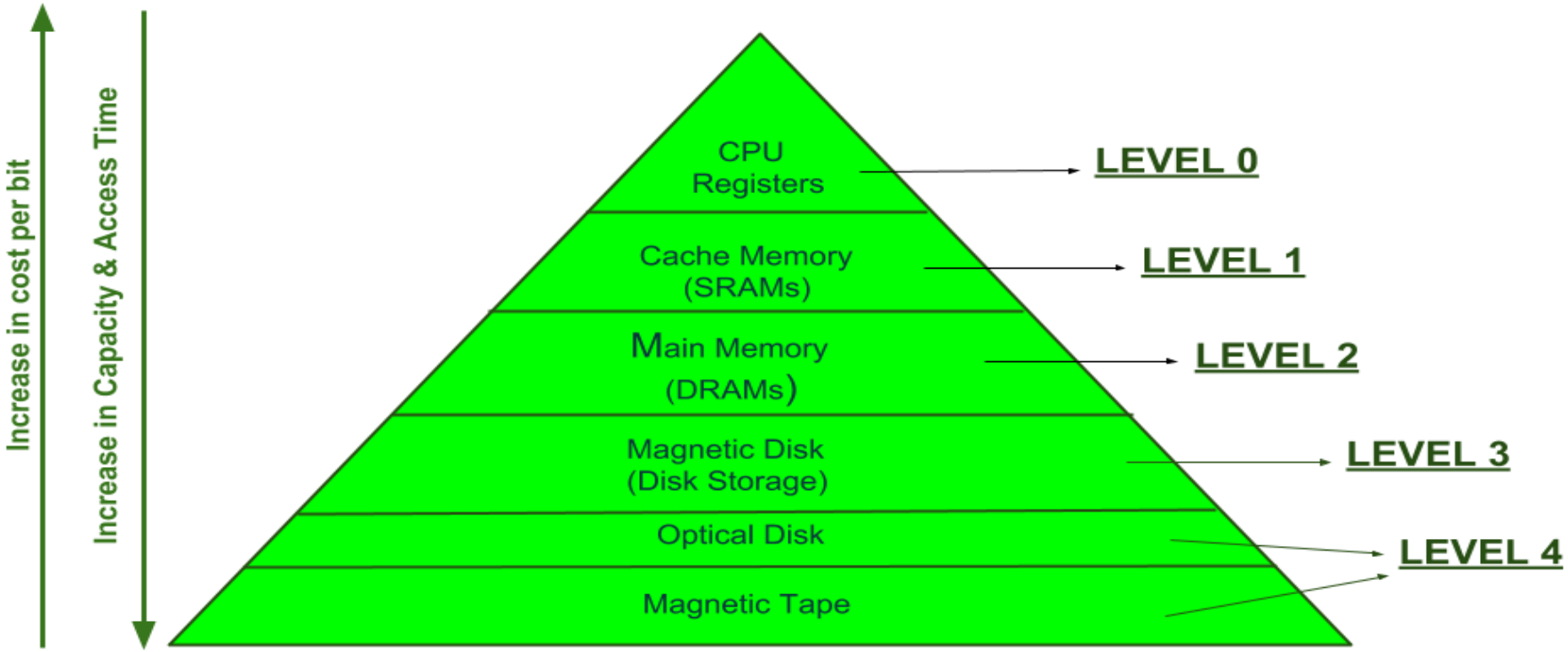
https://www.tutorialspoint.com/

UNIT- IV

Memory Organization: Concept of hierarchical memory organization, Main memory, Semiconductor memory technologies, Cache memory, Virtual memory, Auxiliary memory, Direct Memory Access(DMA).

Memory Organization

Concept of hierarchical memory organization



MEMORY HIERARCHY DESIGN

Main Memory

- The main memory acts as storage unit in a computer system. It is a relatively large and fast memory which is used to store programs and data during the run time operations.
- Technology used for the main memory:
- RAM (Random Access Memory) integrated circuit chips
- ROM (Read Only Memory) integrated circuit chips

Primary Memory: It is also known as the main memory of the computer system.

Primary memory is of two types:

- **(i) RAM (Random Access Memory):** It is a volatile memory. Volatile memory stores information based on the power supply.

  RAM is of two types:

- **S RAM (Static RAM):** It uses transistors and the circuits of this memory are capable of retaining their state as long as the power is applied.
- **D RAM (Dynamic RAM):** It uses capacitors and transistors and stores the data as a charge on the capacitors. They contain thousands of memory cells.

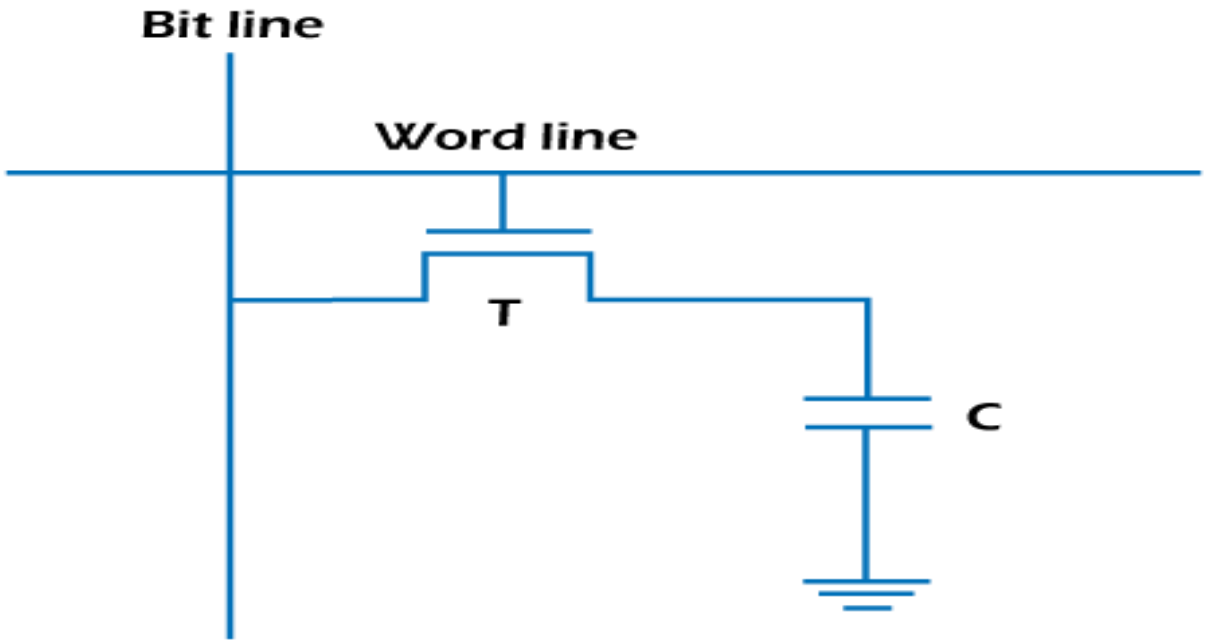
- **(ii) ROM (Read Only Memory):** It is a non-volatile memory.
- ROM is used to store information that is used to operate the system.
- ROM is of four types:
- **MROM(Masked ROM):** Hard-wired devices with a pre-programmed collection of data or instructions were the first ROMs.
- **PROM (Programmable Read Only Memory):** This read-only memory is modifiable once by the user.

- **EPROM (Erasable Programmable Read Only Memory):** It is an extension to PROM where you can erase the content of ROM by exposing it to Ultraviolet rays for nearly 40 minutes.
- **EEPROM (Electrically Erasable Programmable Read Only Memory):** Here the written contents can be erased electrically.

Semiconductor memory technologies

DRAM

- DRAM will store bits of data in a storage or memory cell, consisting of a capacitor and a transistor.



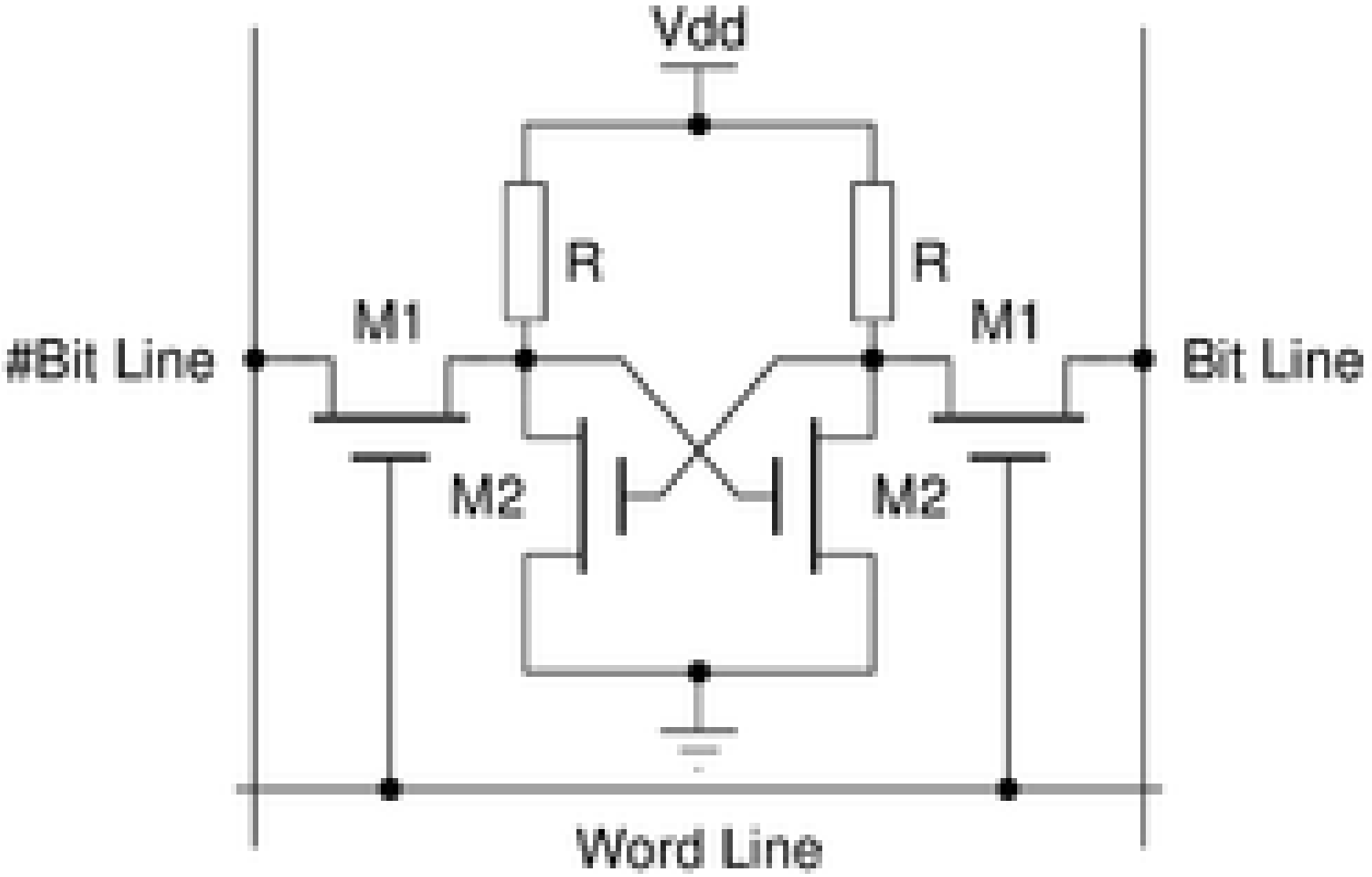
- A DRAM storage cell is dynamic, meaning that it needs to be refreshed or given a new electronic charge every few milliseconds to compensate for charge leaks from the capacitor.
- The memory cells will work with other circuits that can be used to identify rows and columns,

- DRAM is one option of semiconductor memory that a system designer can use when building a computer.
- Alternative memory choices include static RAM (SRAM), electrically erasable programmable read-only memory (EEPROM), NOR flash, and NAND flash.

SRAM

- A typical SRAM cell is made up of six MOSFETS. Each bit in an SRAM is stored on four transistors (M1, M2, M3, M4) that form two cross-coupled inverters.
- Two additional *access* transistors serve to control the access to a storage cell during read and write operations.

SRAM cell

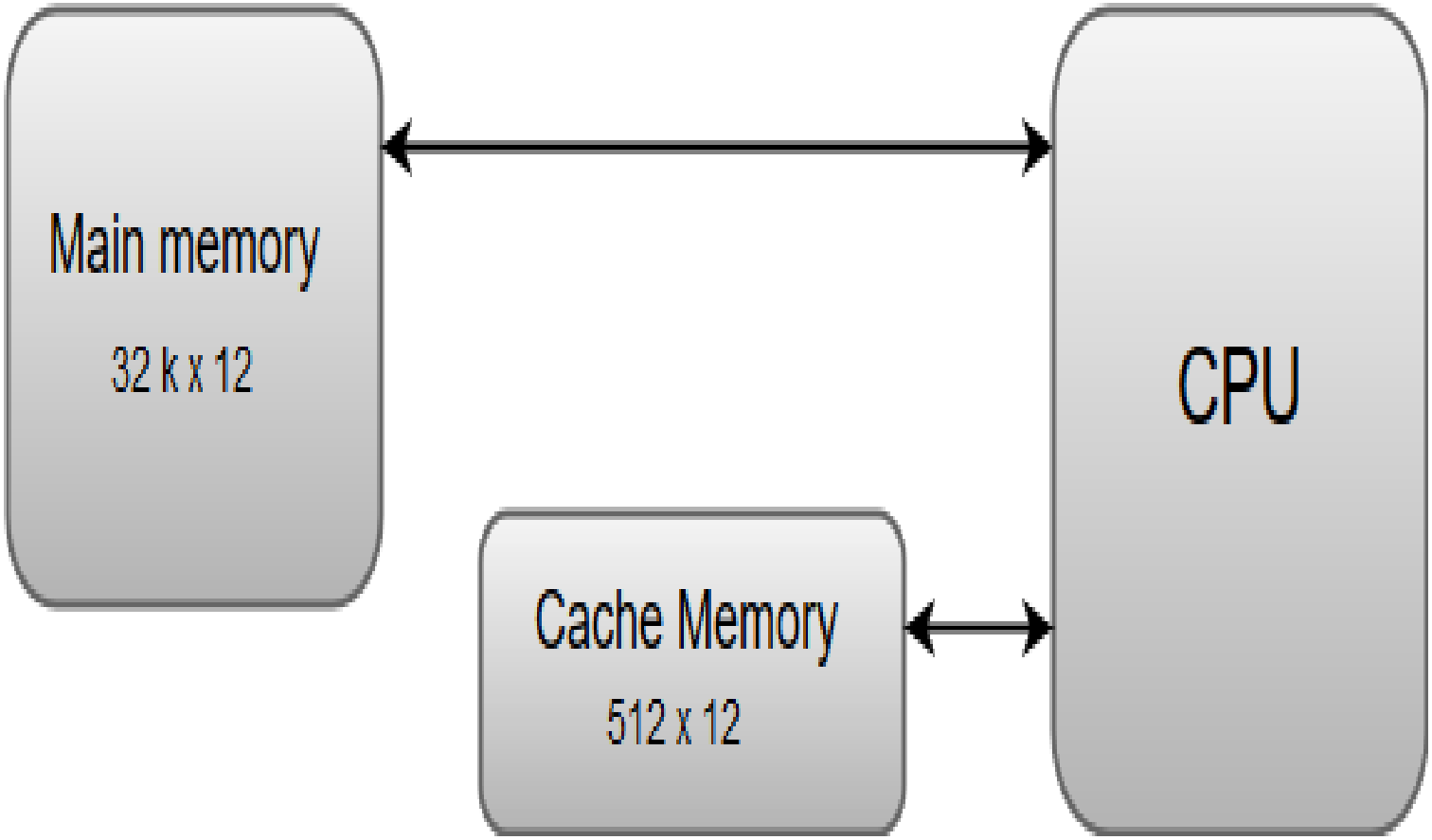


- Four-transistor SRAM is quite common in stand-alone SRAM devices implemented in special processes with an extra layer of polyvision allowing for very high-resistance pull-up resistors.

Cache Memory

- The data or contents of the main memory that are used frequently by CPU are stored in the cache memory so that the processor can easily access that data in a shorter time.
- Whenever the CPU needs to access memory, it first checks the cache memory.

Cache memory



- **Types of Cache**

PrimaryCache

Primary cache is always located on the processor chip. This cache is small anditsaccess time is comparable to that of processor registers.

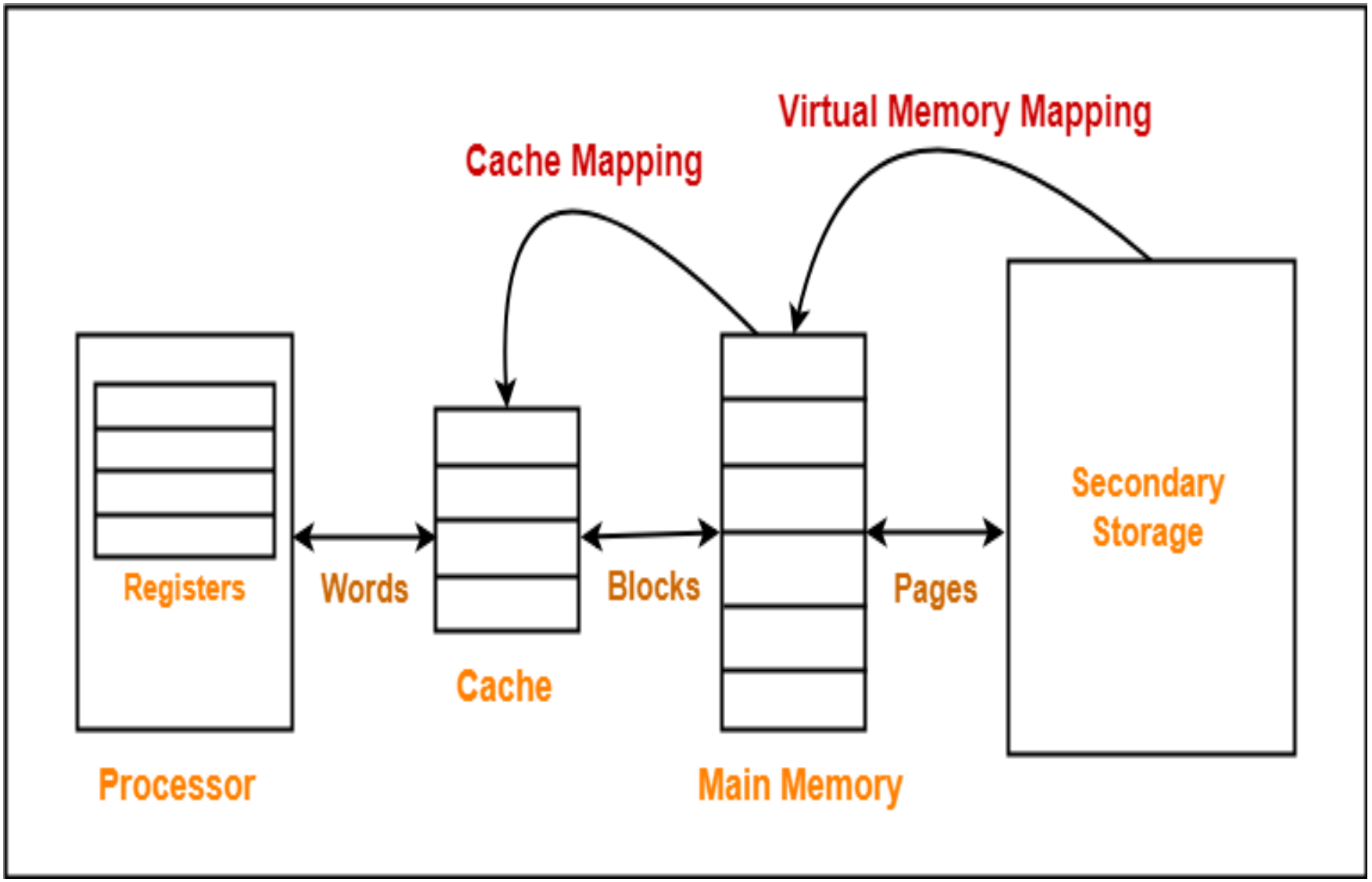
SecondaryCache

Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.

Cache Mapping

- Cache mapping defines how a block from the main memory is mapped to the cache memory in case of a cache miss.
- Cache mapping is a technique by which the contents of main memory are brought into the cache memory.
- The following diagram illustrates the mapping process

Cache mapping



- Direct Mapping-

  A particular block of main memory can map only to a particular line of the cache.

  The line number of cache to which a particular block can map is given by

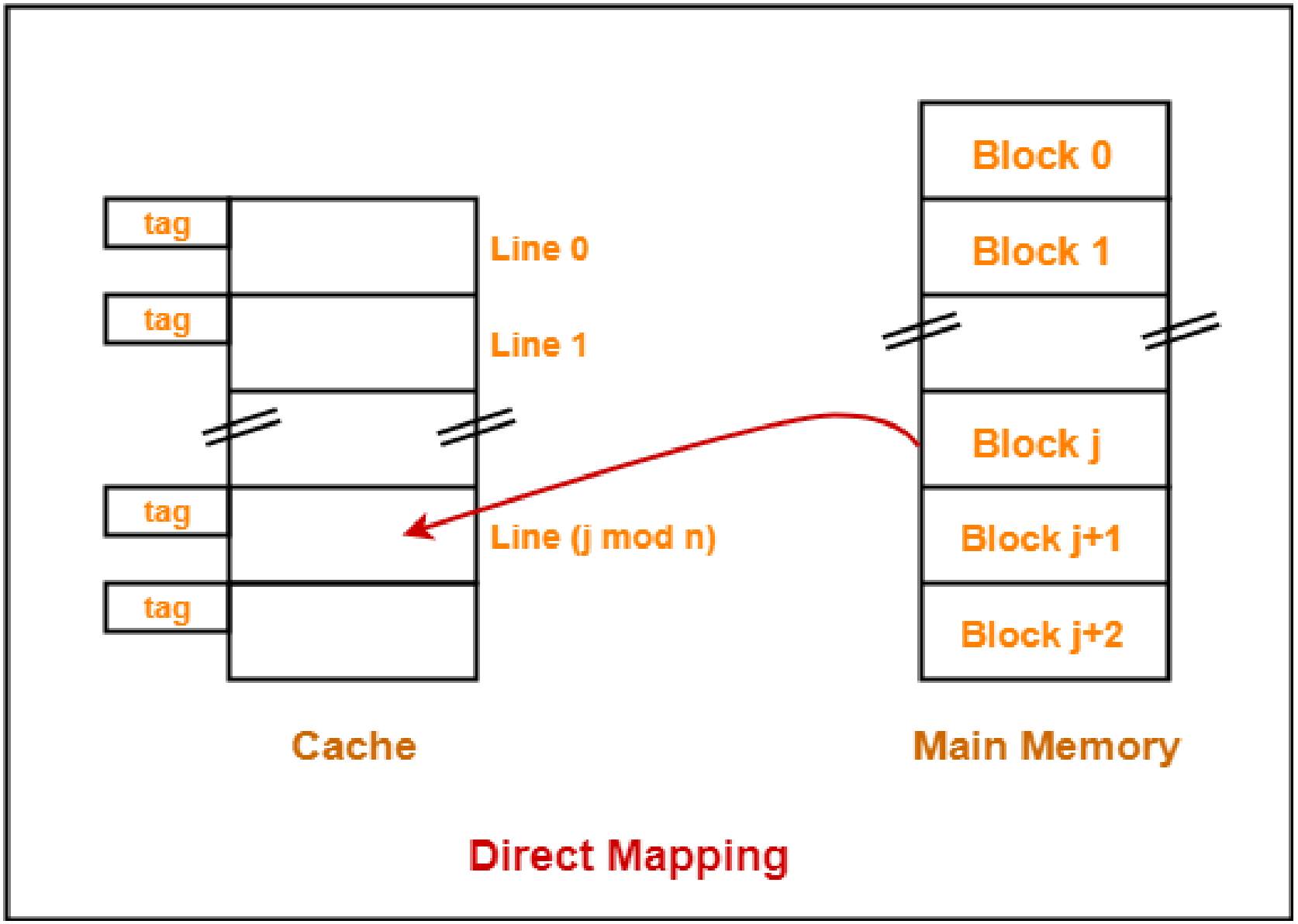
  Cache line number

  = (Main Memory Block Address) Modulo (Number of lines in Cache)

  Example:

  Consider cache memory is divided into ‘n’ number of lines.

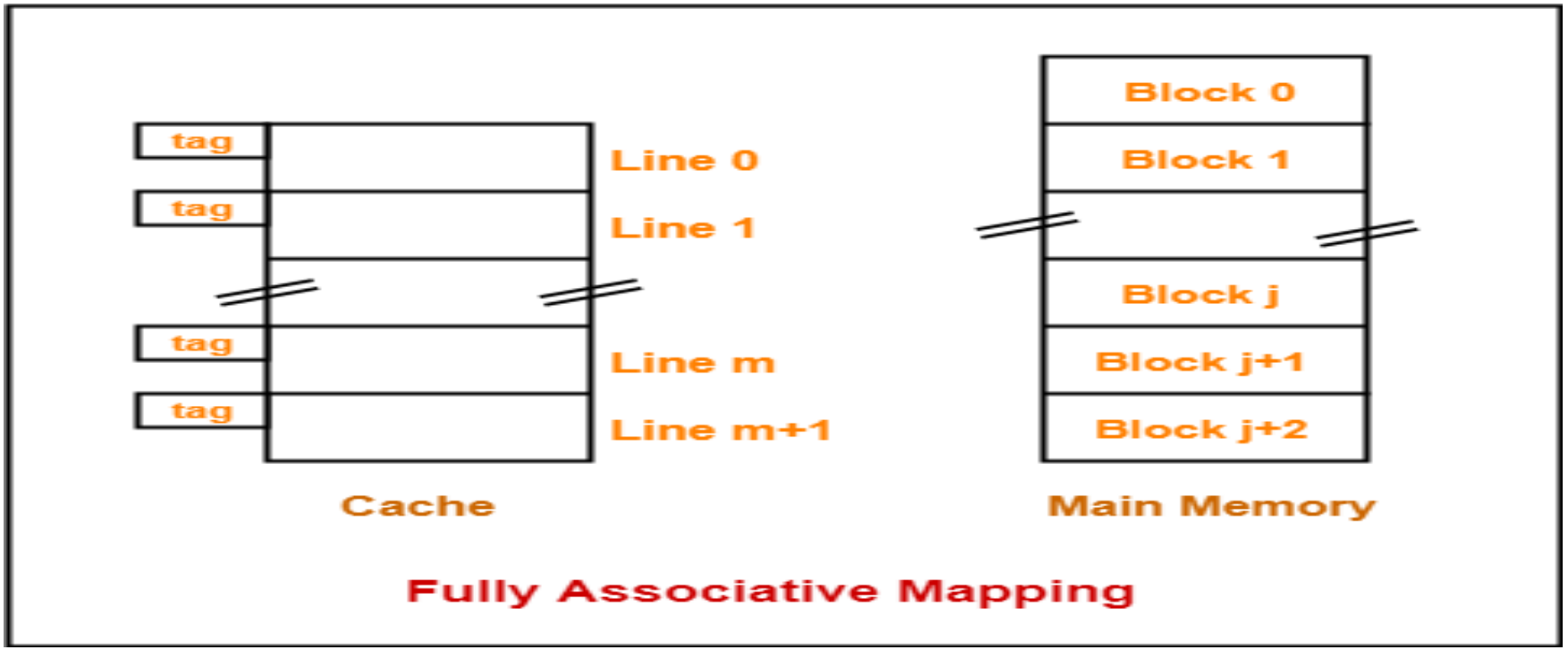
  Then, block ‘j’ of main memory can map to line number (j mod n) only of the cache.


tag
tag
tag
tag
Line 0
Line 1
Line (j mod n)
Block 0
Block 1
Block j
Block j+1
Block j+2
Cache
Main Memory
Direct Mapping

Fully Associative Mapping:

A block of main memory can map to any line of the cache that is freely available at that moment.

This makes fully associative mapping more flexible than direct mapping.



Virtual Memory

Virtual Memory is a storage scheme that provides user an illusion of having a very big main memory.

In this scheme, User can load the bigger size processes than the available main memory.

Instead of loading one big process in the main memory, the Operating System loads the different parts of more than one process in the main memory.

Demand Paging

In demand paging, the pages of a process which are least used, get stored in the secondary memory.

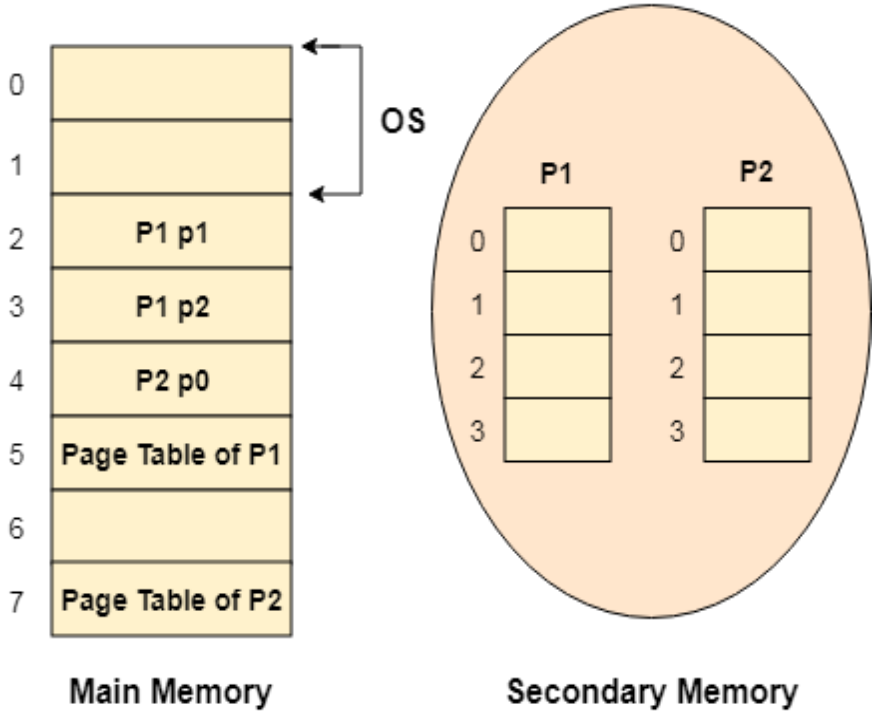
A page is copied to the main memory when its demand is made or page fault occurs.

Page replacement algorithms are used to determine the pages which will be replaced.

Virtual memory management

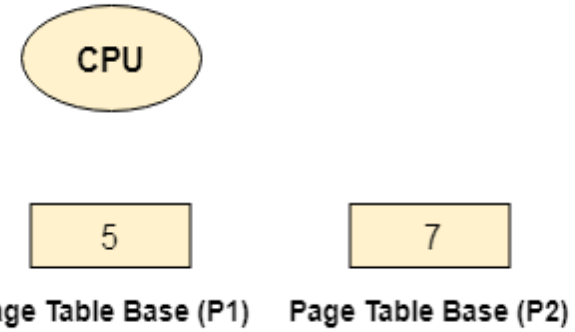
	Frame	Present / Absent	D Bit	Reference bit	Protection
0		0	0	0	
1	2	1	0	1	
2	3	1	1	0	
3		0	0	0	

Page Table of P1

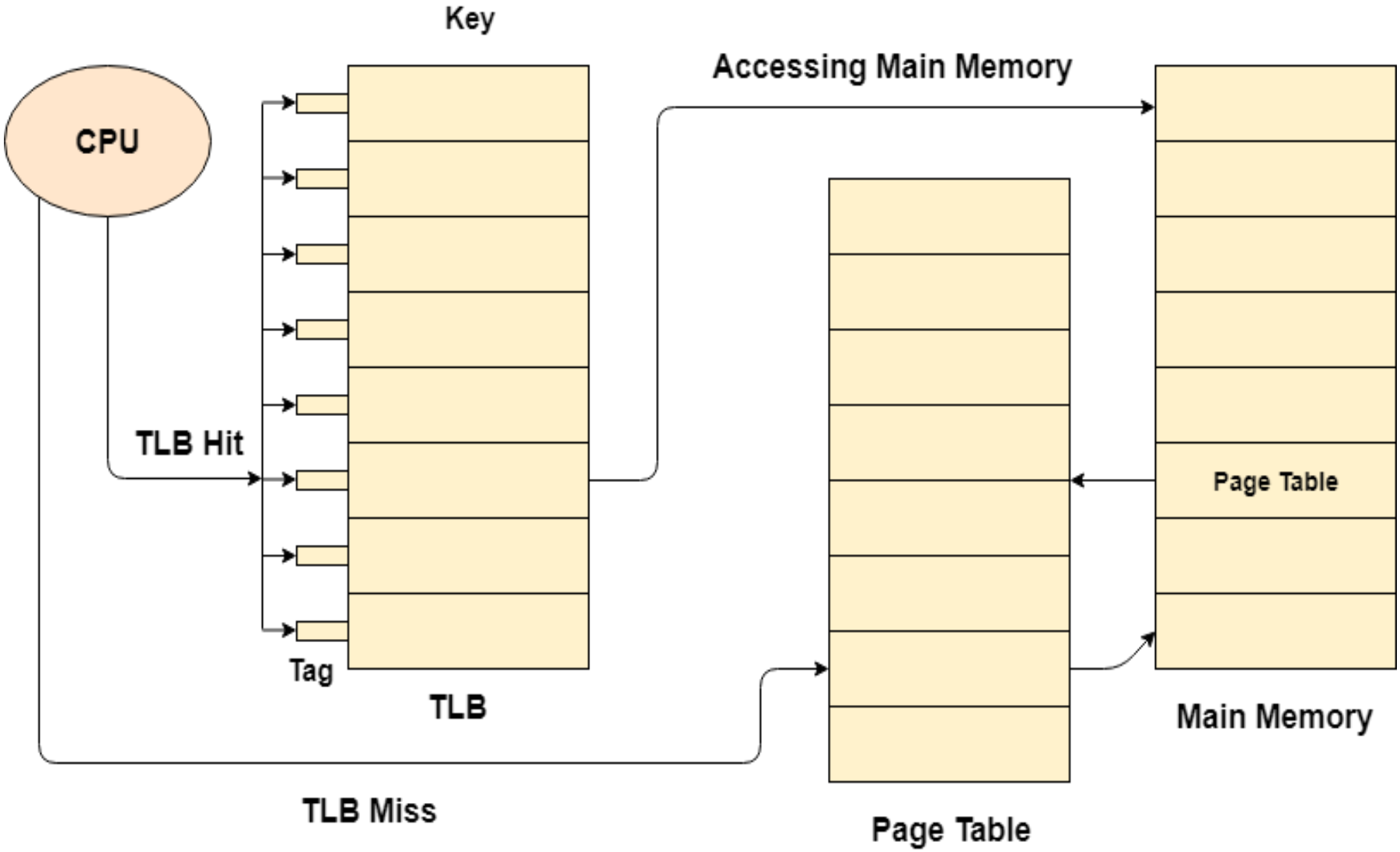


	Frame	Present / Absent	D Bit	Reference bit	Protection
0	4	1	0	1	
1		0	0	0	
2		0	0	0	
3		0	0	0	

Page Table of P2



Virtual memory management

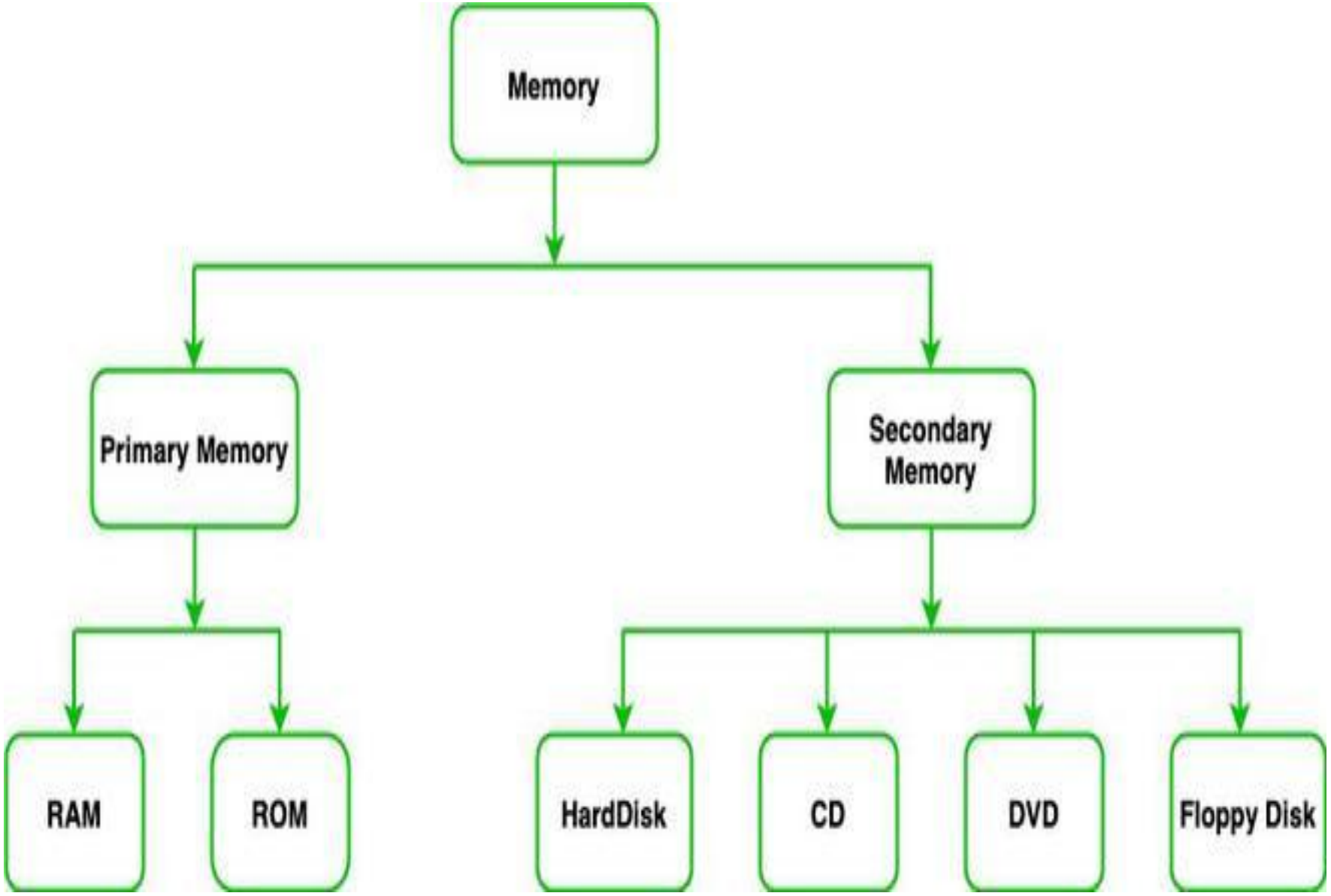


Auxiliary memory:

Auxiliary memory (also referred to as *secondary storage*) is the non-volatile memory lowest-cost, highest-capacity, and slowest-access storage in a computer system.

It is where programs and data kept for long-term storage or when not in immediate use.

Classification of Auxiliary memory



Direct Memory Access

DMA controller is a **hardware unit** that allows I/O devices to access memory directly without the participation of the processor.

The unit that controls the activity of accessing memory directly is called a **DMA controller**.

The processor **relinquishes the system bus** for a few clock cycles. So, the DMA controller can accomplish the task of data transfer via the system bus.

Direct memory access (DMA) is a **mode of data transfer** between the memory and I/O devices. This happens **without the involvement** of the processor.

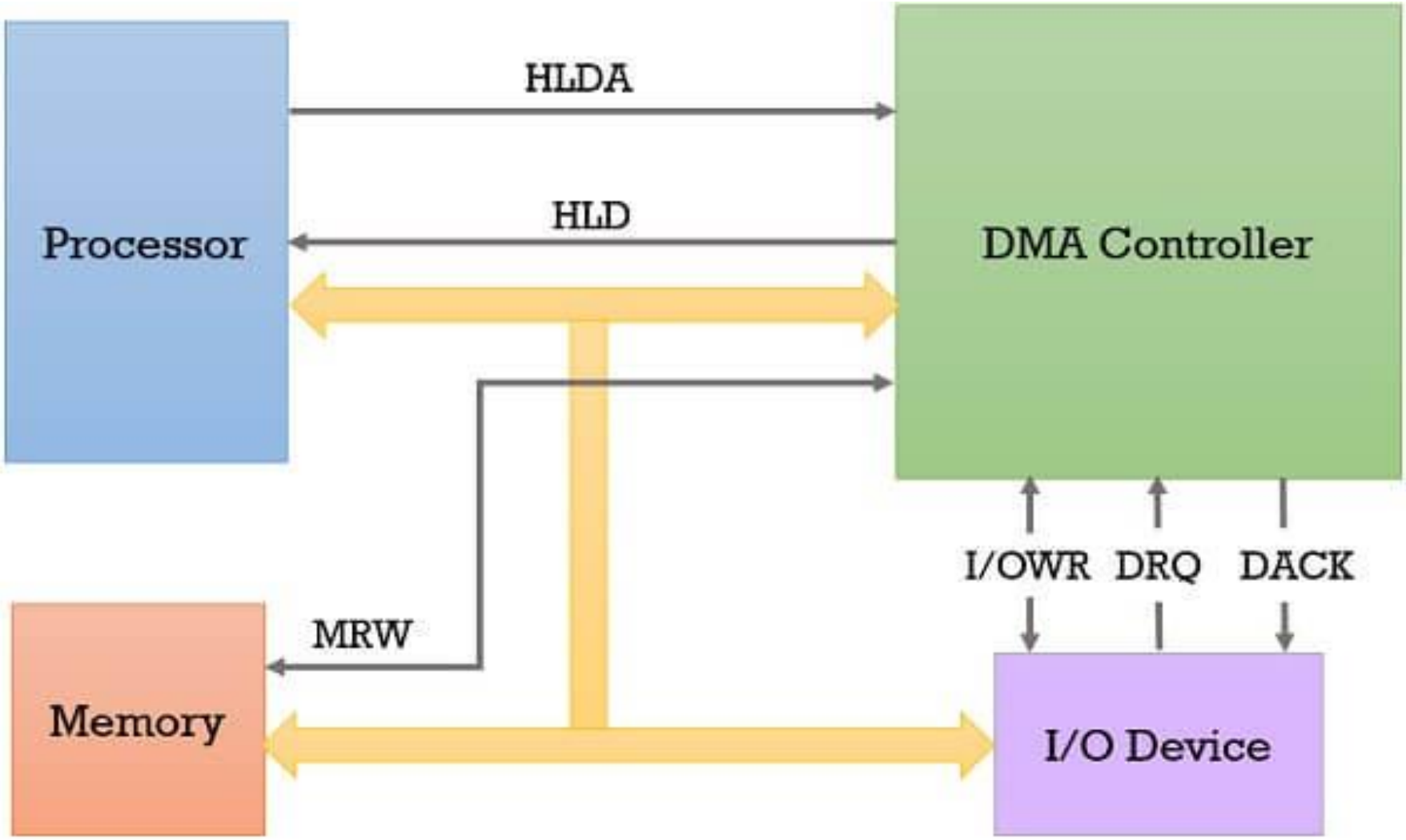
The DMA controller transfers the data in three modes:

Burst Mode: Here, once the DMA controller gains the charge of the system bus, then it releases the system bus only after **completion** of data transfer. Till then the CPU has to wait for the system buses.

Cycle Stealing Mode: In this mode, the DMA controller **forces** the CPU to stop its operation and **relinquish the control over the bus** for a **short term** to DMA controller.

After the **transfer of every byte**, the DMA controller **releases** the **bus** and then again requests for the system bus. In this way, the DMA controller steals the clock cycle for transferring every byte.

Transparent Mode: Here, the DMA controller takes the charge of system bus only if the **processor does not require the system bus**.



DMA Controller Data Transfer

Web references:

https://www.studytonight.com

https://www.javatpoint.com

https://www.tutorialspoint.com/

UNIT- V

Pipelining: Basic concepts of pipelining, Arithmetic pipeline, Instruction pipeline, Instruction Hazards.

Parallel Processors: Introduction to parallel processors, Multiprocessor, Interconnection structures and Cache coherency.

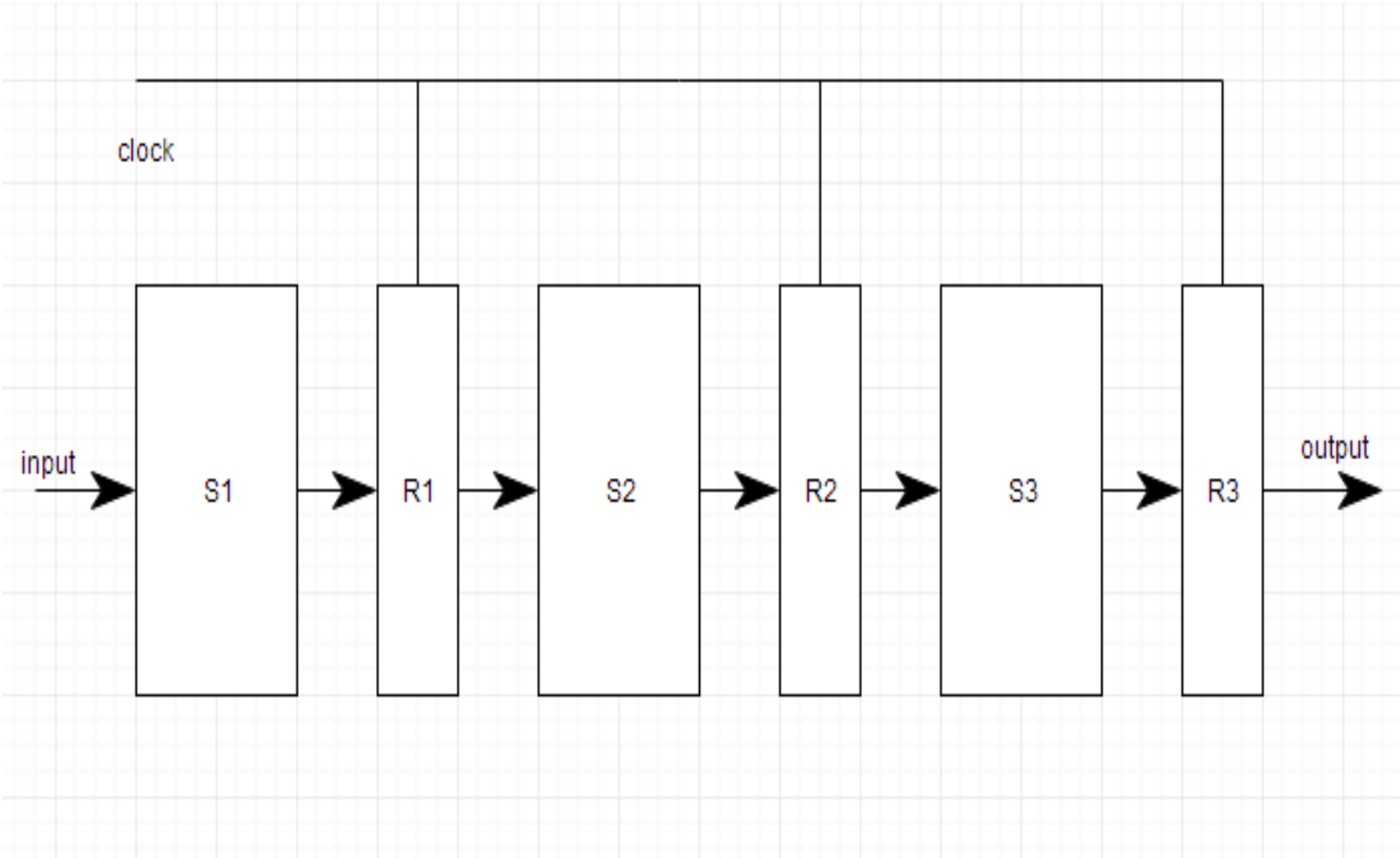
Pipelining: Basic concepts of pipelining:

Pipelining is the process of accumulating instruction from the processor through a pipeline.

Pipelining is a technique where multiple instructions are overlapped during execution. Pipeline is divided into stages and these stages are connected with one another to form a pipe like structure.

Pipelining increases the overall instruction throughput.

Pipeline stages



Types of Pipeline

It is divided into 2 categories:

Arithmetic Pipeline

Instruction Pipeline

Arithmetic Pipeline

Arithmetic Pipelines are mostly used in high-speed computers. They are used to implement floating-point operations, multiplication of fixed-point numbers

Example of a pipeline unit for floating-point addition and subtraction.

The inputs to the floating-point adder pipeline are two normalized floating-point binary numbers defined as:

$X = A * 2^a = 0.9504 * 10^3$

$Y = B * 2^b = 0.8200 * 10^2$

Where **A** and **B** are two fractions that represent the mantissa and **a** and **b** are the exponents.

The combined operation of floating-point addition and subtraction is divided into four segments.

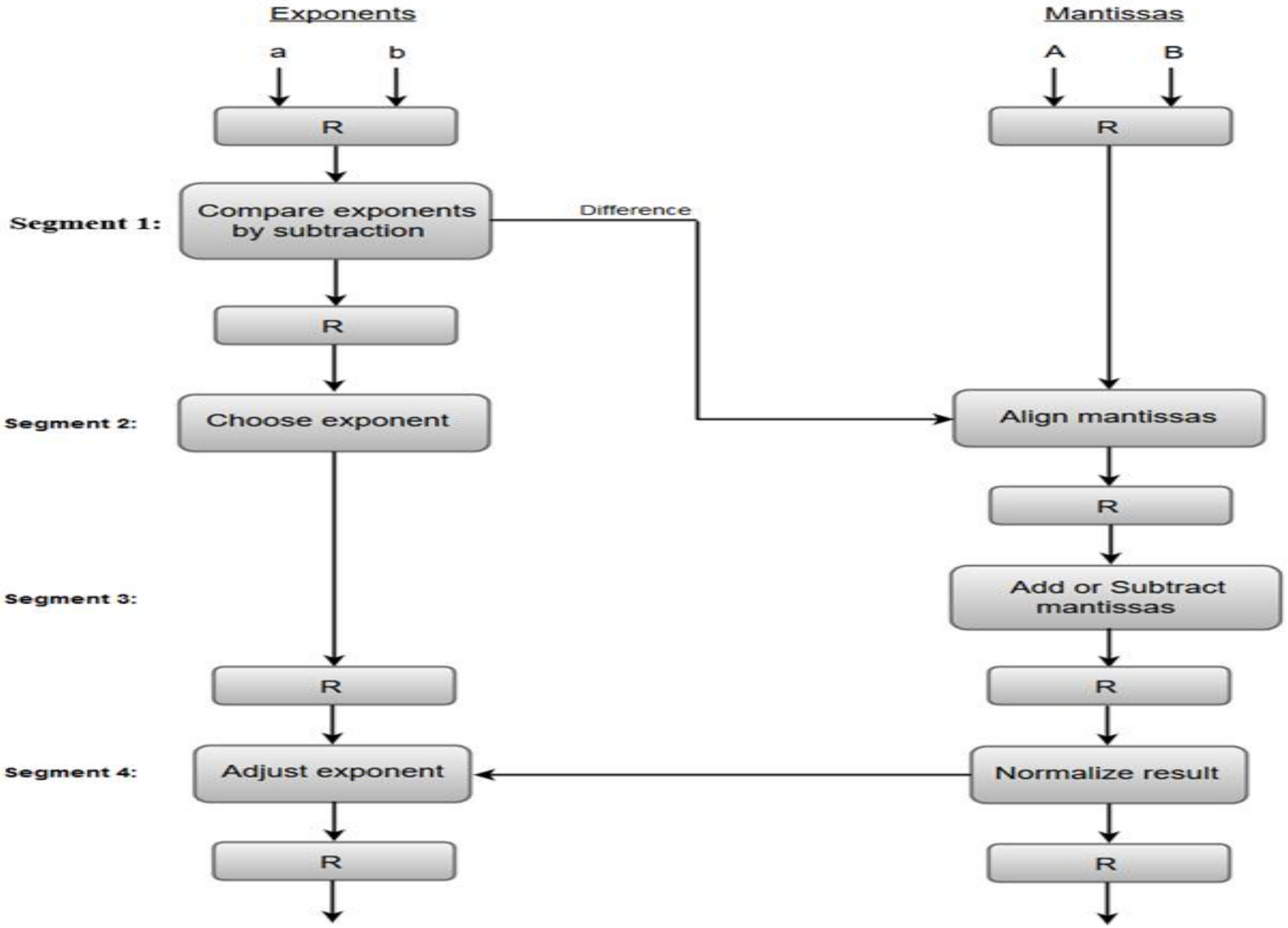
Compare the exponents by subtraction.

Align the mantissas.

Add or subtract the mantissas.

Normalize the result.

Pipeline organization for floating point addition and subtraction:



Instruction Pipeline

In general, the computer needs to process each instruction with the following sequence of steps.

Fetch instruction from memory.

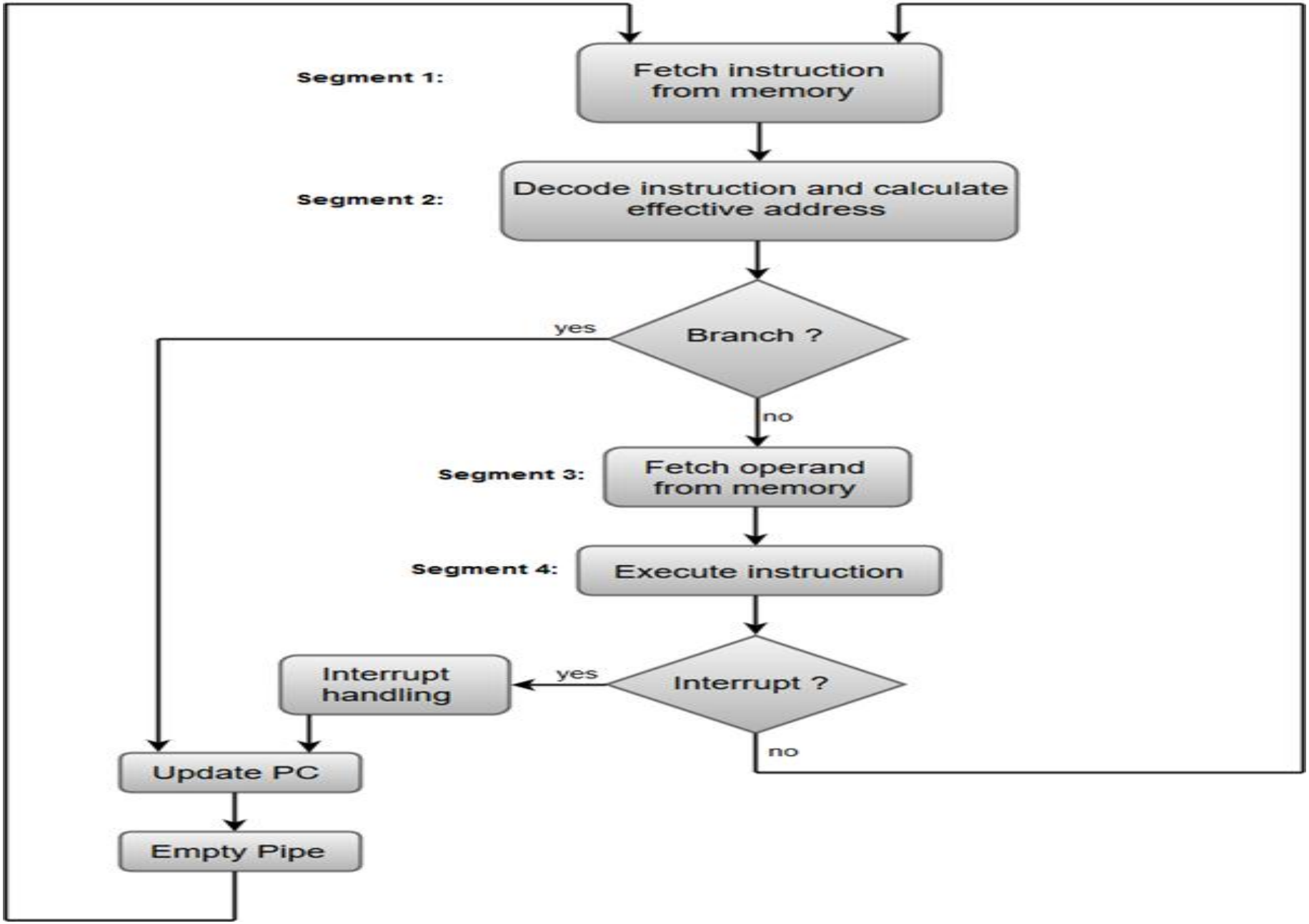
Decode the instruction.

Calculate the effective address.

Fetch the operands from memory.

Execute the instruction.

Store the result in the proper place.


Segment 1:
Fetch instruction from memory
Segment 2:
Decode instruction and calculate effective address
Branch ?
yes
no
Segment 3:
Fetch operand from memory
Segment 4:
Execute instruction
Interrupt ?
yes
Interrupt handling
no
Update PC
Empty Pipe

Pipeline Conflicts

There are some factors that cause the pipeline to deviate its normal performance.

Timing Variations

All stages cannot take same amount of time. This problem generally occurs in instruction processing

Data Hazards

When several instructions are in partial execution, and if they reference same data then the problem arises.

3. Branching

In order to fetch and execute the next instruction, we must know what that instruction is. If the present instruction is a conditional branch, and its result will lead to the next instruction.

4. Interrupts

Interrupts set unwanted instruction into the instruction stream. Interrupts effect the execution of instruction.

5. Data Dependency

It arises when an instruction depends upon the result of a previous instruction

Instruction Hazards:

Dependencies in a pipelined processor

1) Structural Dependency
   2) Control Dependency
   3) Data Dependency

Structural dependency

This dependency arises due to the resource conflict in the pipeline. A resource conflict is a situation when more than one instruction tries to access the same resource in the same cycle. A resource can be a register, memory, or ALU.

Control Dependency (Branch Hazards)

This type of dependency occurs during the transfer of control instructions such as BRANCH, CALL, JMP, etc.

Consider the following sequence of instructions in the program:

100: I_1
101: I_2 (JMP 250)
102: I_3
.
.
250: BI_1

Expected output: I_1 -> I_2 -> BI_1

Data Dependency (Data Hazard)

I1 : ADD R1, R2, R3

I2 : SUB R4, R1, R2

When the above instructions are executed in a pipelined processor, then data dependency condition will occur, which means that I_2 tries to read the data before I_1 writes it, therefore, I_2 incorrectly gets the old value from I_1.

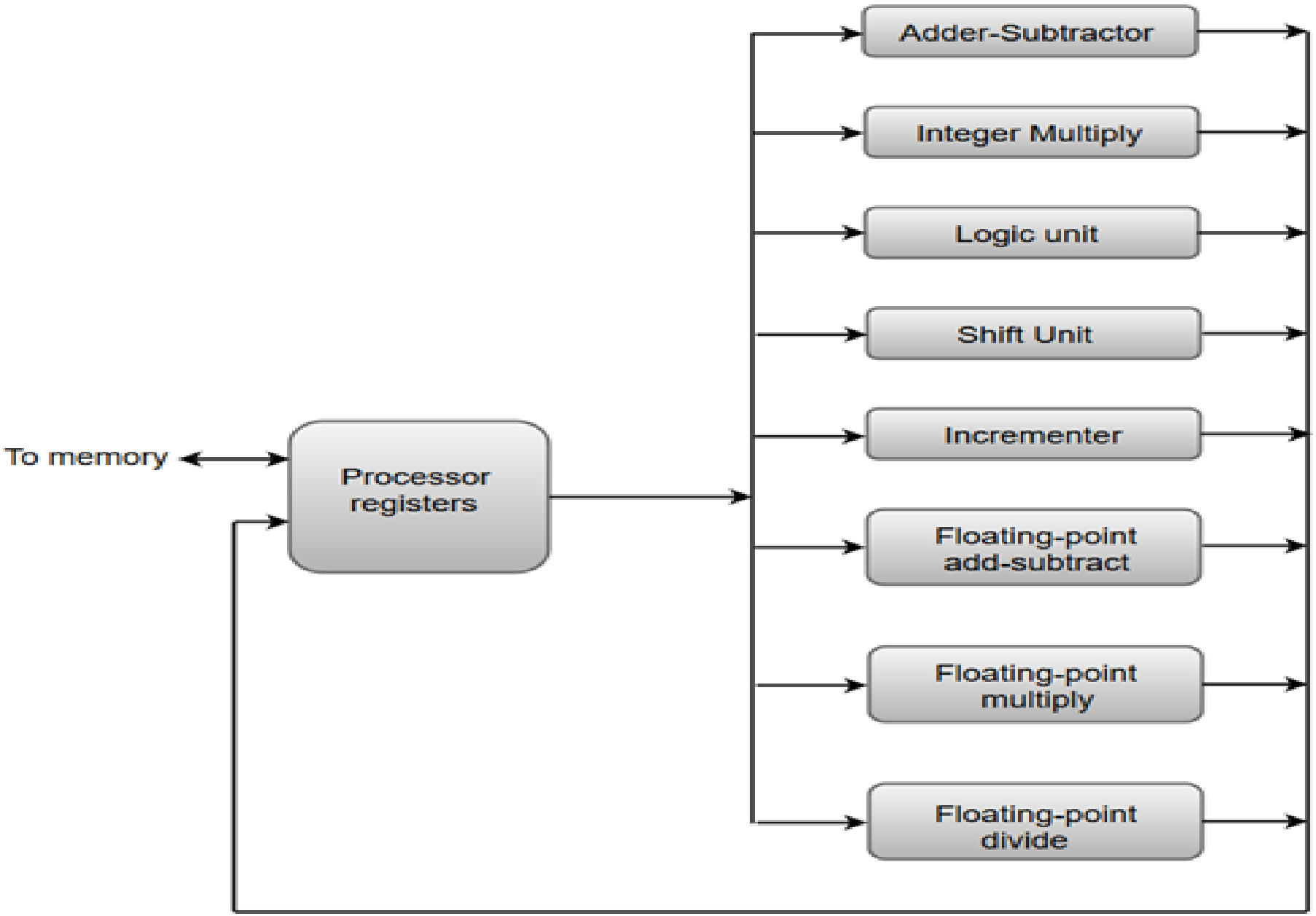
Parallel Processors:

Introduction to parallel processors:

Parallel processing can be described as a class of techniques which enables the system to achieve simultaneous data-processing tasks to increase the computational speed of a computer system.

A parallel processing system can carry out simultaneous data-processing to achieve faster execution time. For instance, while an instruction is being processed in the ALU component of the CPU, the next instruction can be read from memory.

Eight functional units operating in Parallel



Multiprocessor:

A Multiprocessor is a computer system with two or more central processing units (CPUs) share full access to a common RAM.

Two types of multiprocessors:

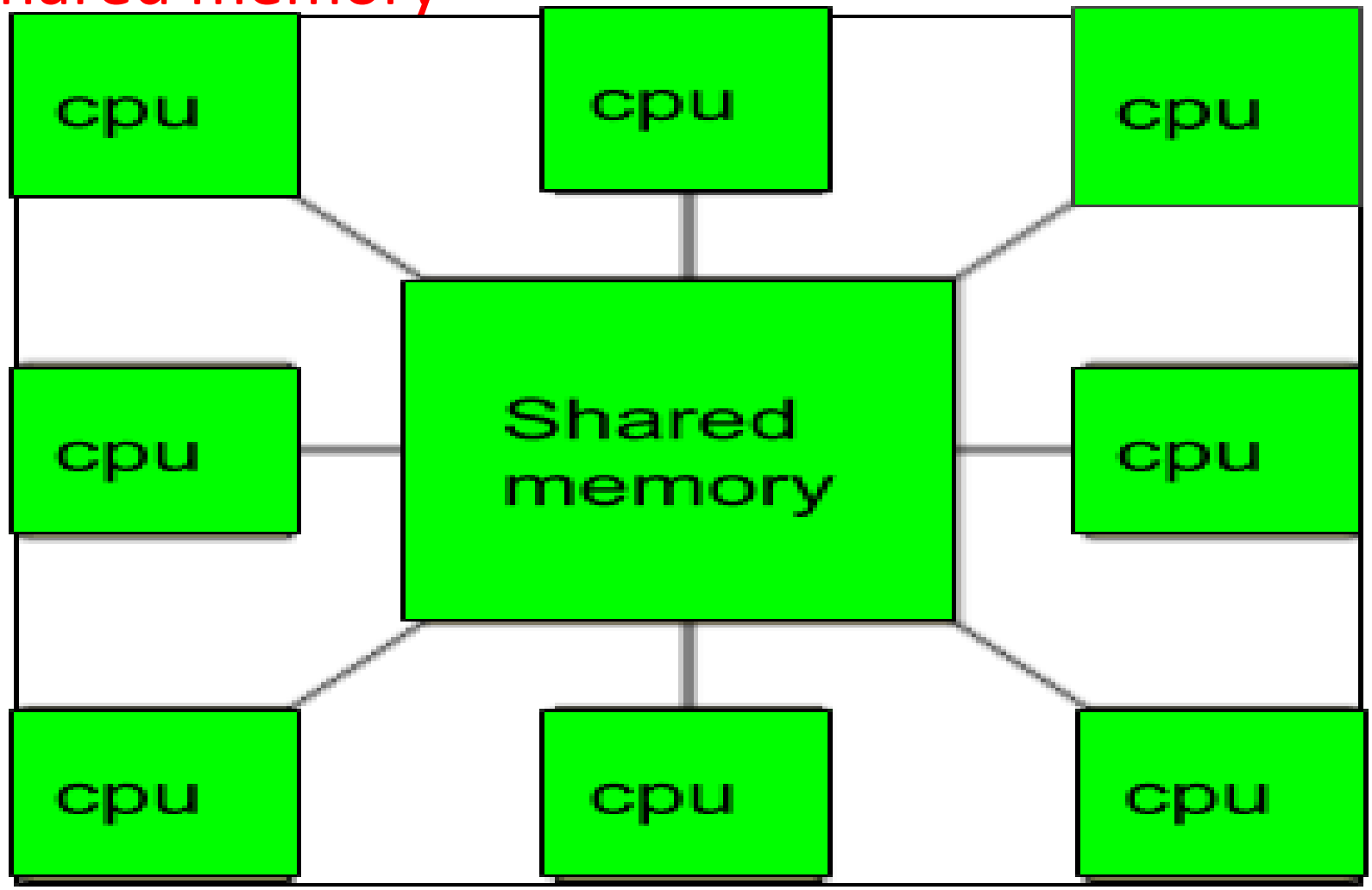
Shared memory multiprocessor

Distributed memory multiprocessor.

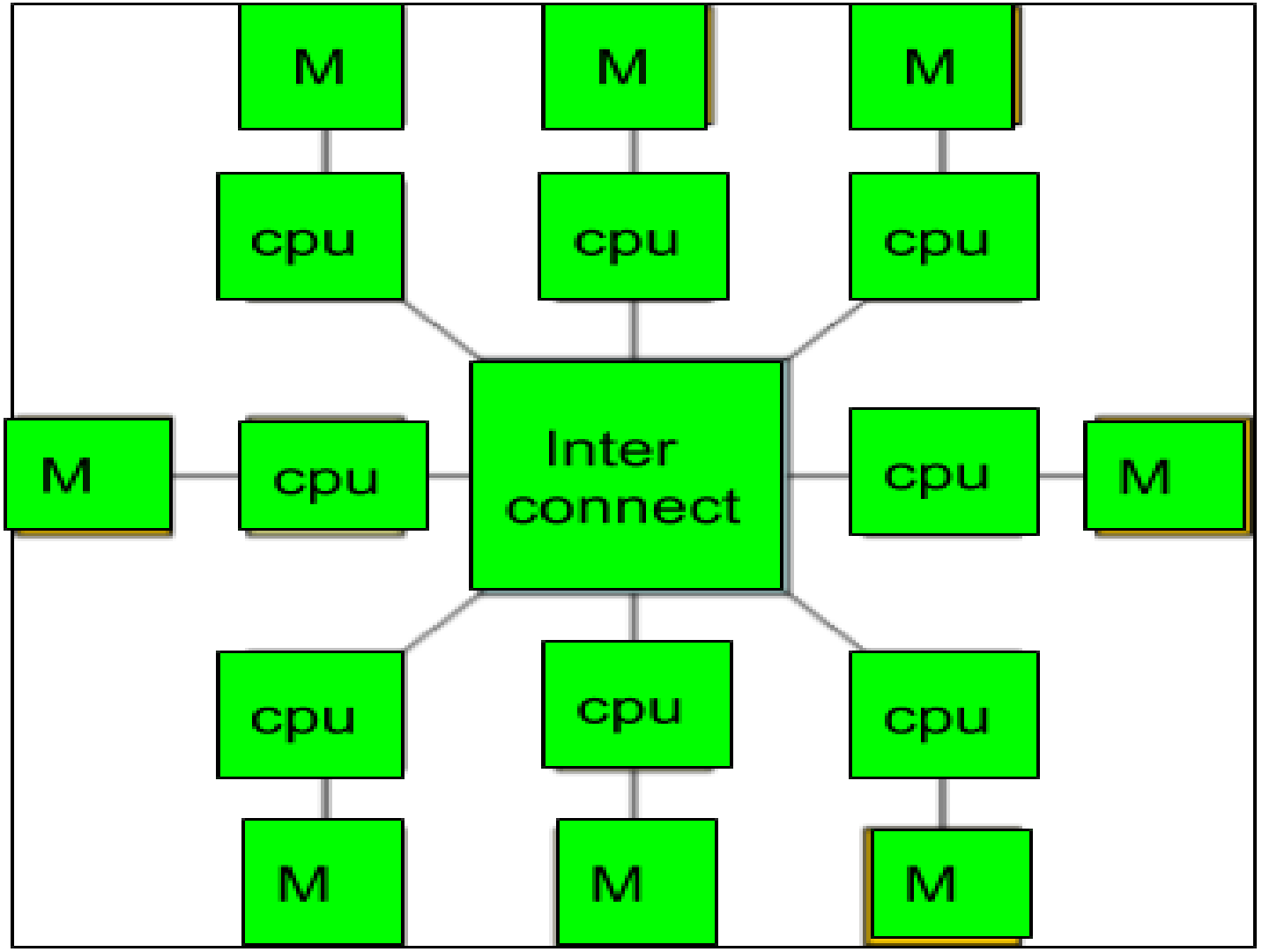
In shared memory multiprocessors, all the CPUs shares the common memory.

Multiprocessor

Shared memory



Multi computer



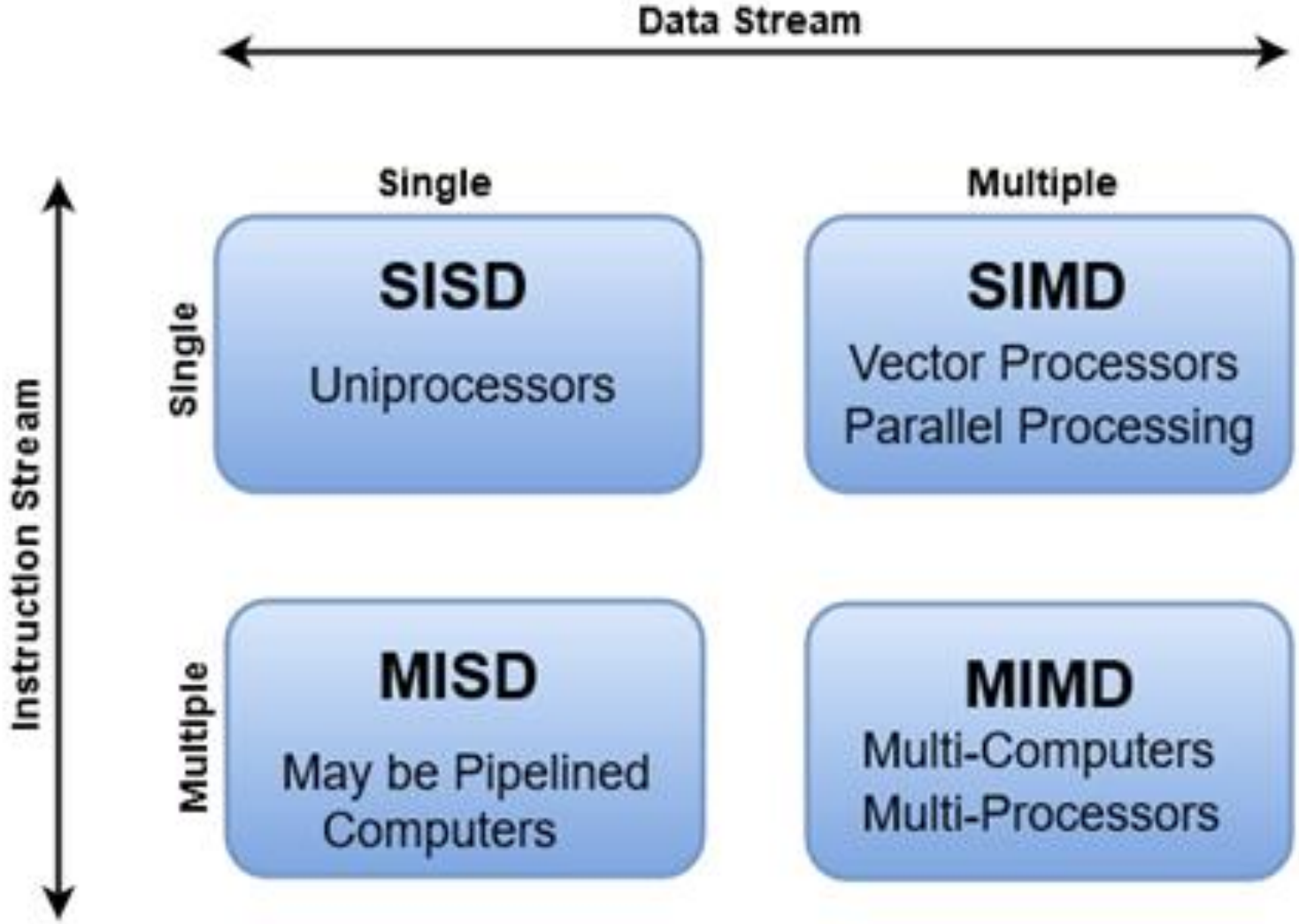
Flynn's Classification of Computers

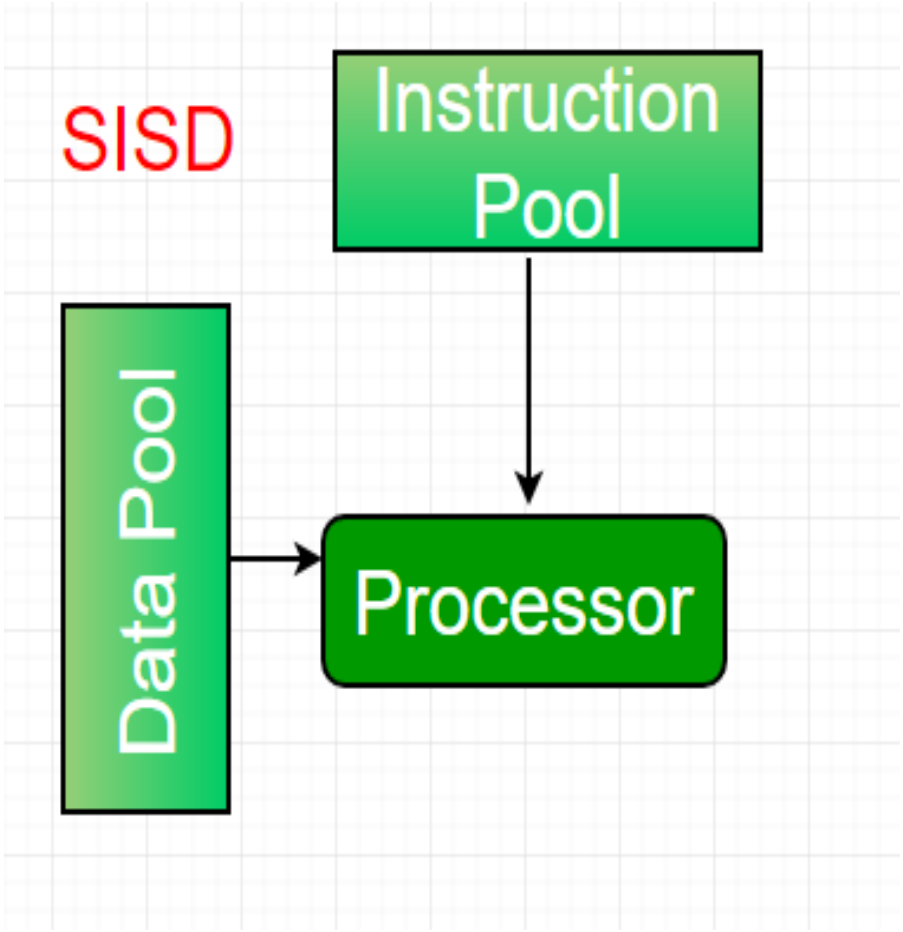
M.J. Flynn proposed a classification for the organization of a computer system by the number of instructions and data items that are manipulated simultaneously.

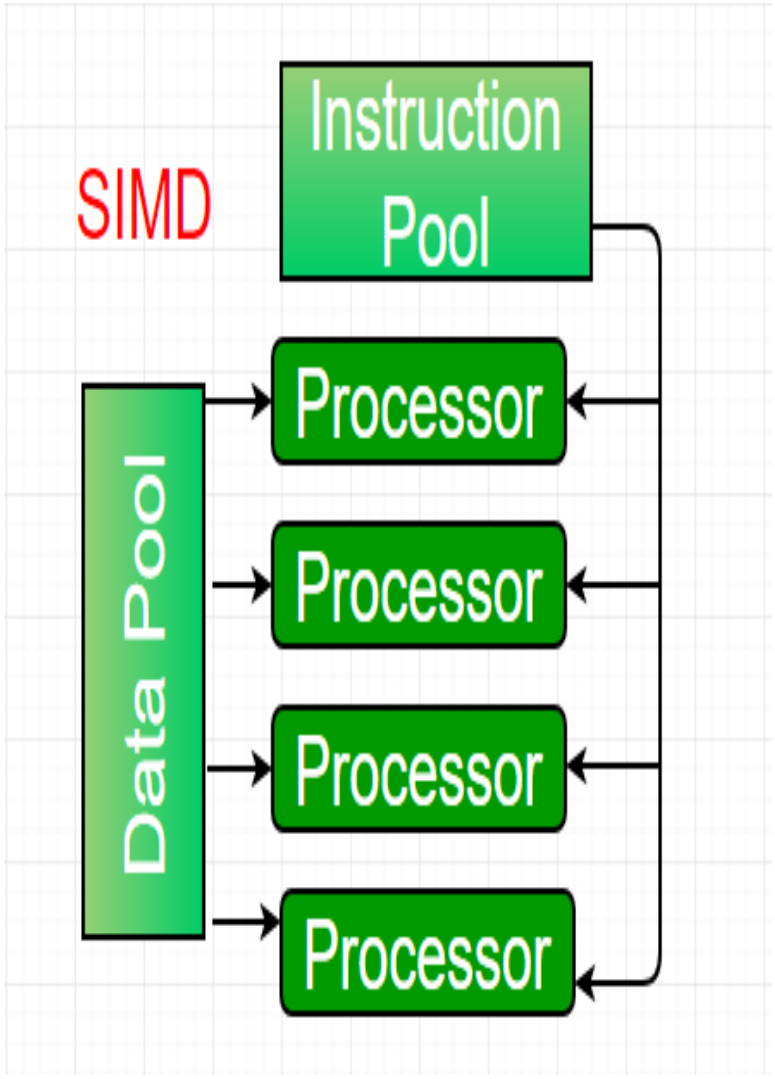
The sequence of instructions read from memory constitutes an **instruction stream**.

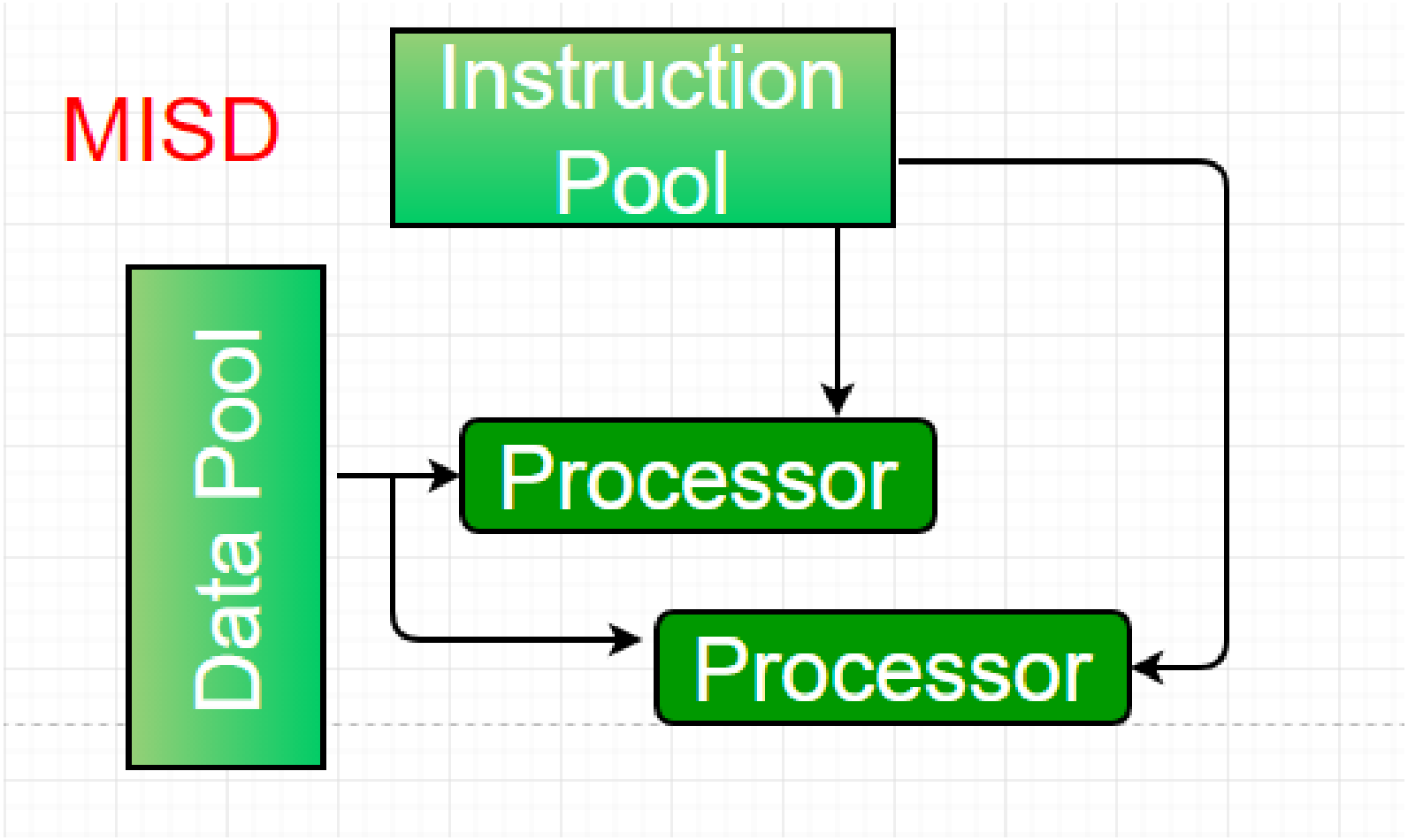
The operations performed on the data in the processor constitute a **data stream**.

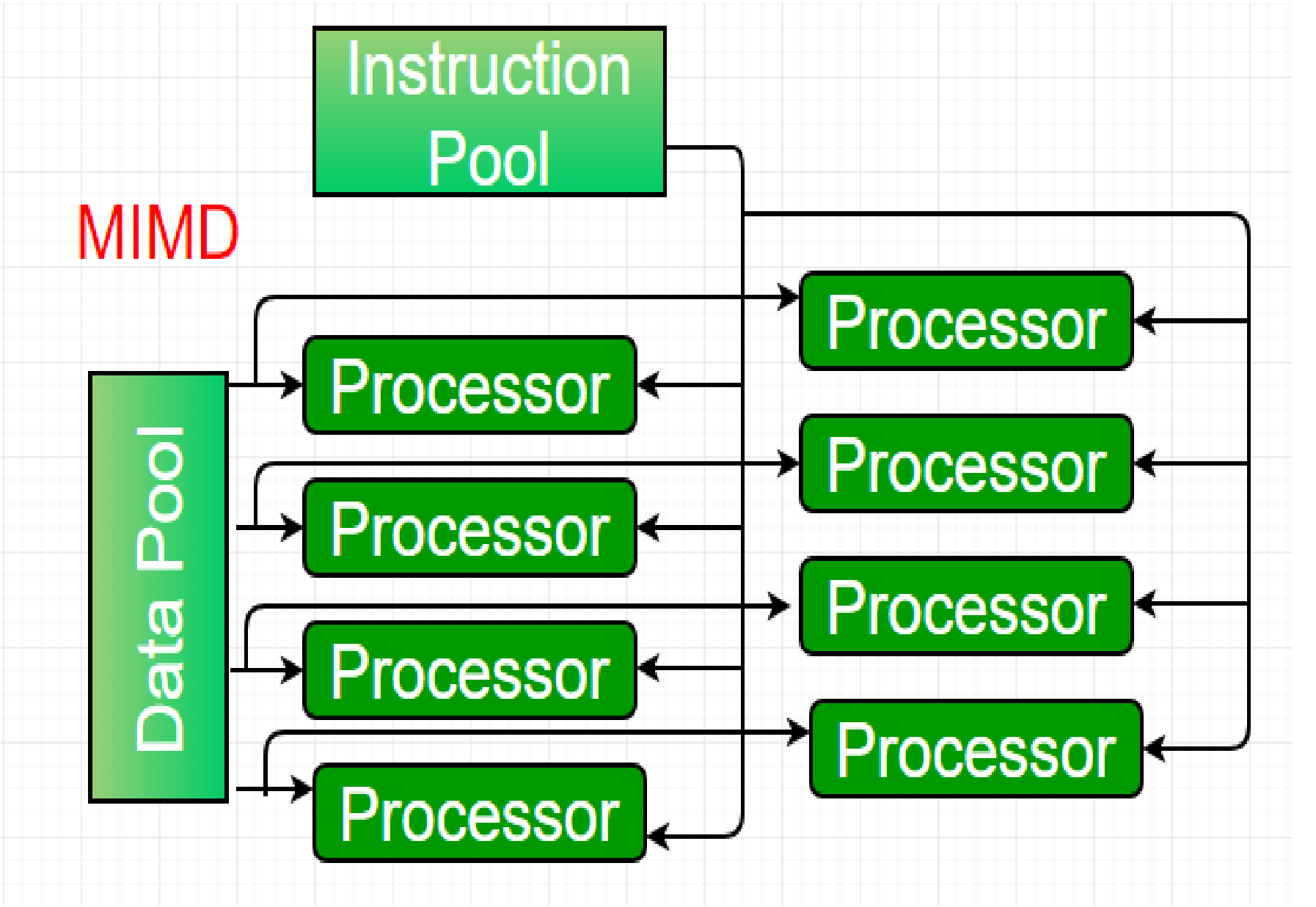
Flynn's Classification of Computers




SISD
Instruction Pool
Data Pool
Processor


SIMD
Instruction Pool
Data Pool
Processor
Processor
Processor
Processor


MISD
Instruction Pool
Data Pool
Processor
Processor


Instruction Pool
MIMD
Data Pool
Processor
Processor
Processor
Processor
Processor
Processor
Processor
Processor

Interconnection Structures:

The interconnection between the components of a multiprocessor System can have different physical configurations depending on the number of transfer paths that are available between the processors and memory in a shared memory system.

Some of the schemes are as: -

Time-Shared Common Bus

System bus for multiprocessor

Multiport memory system

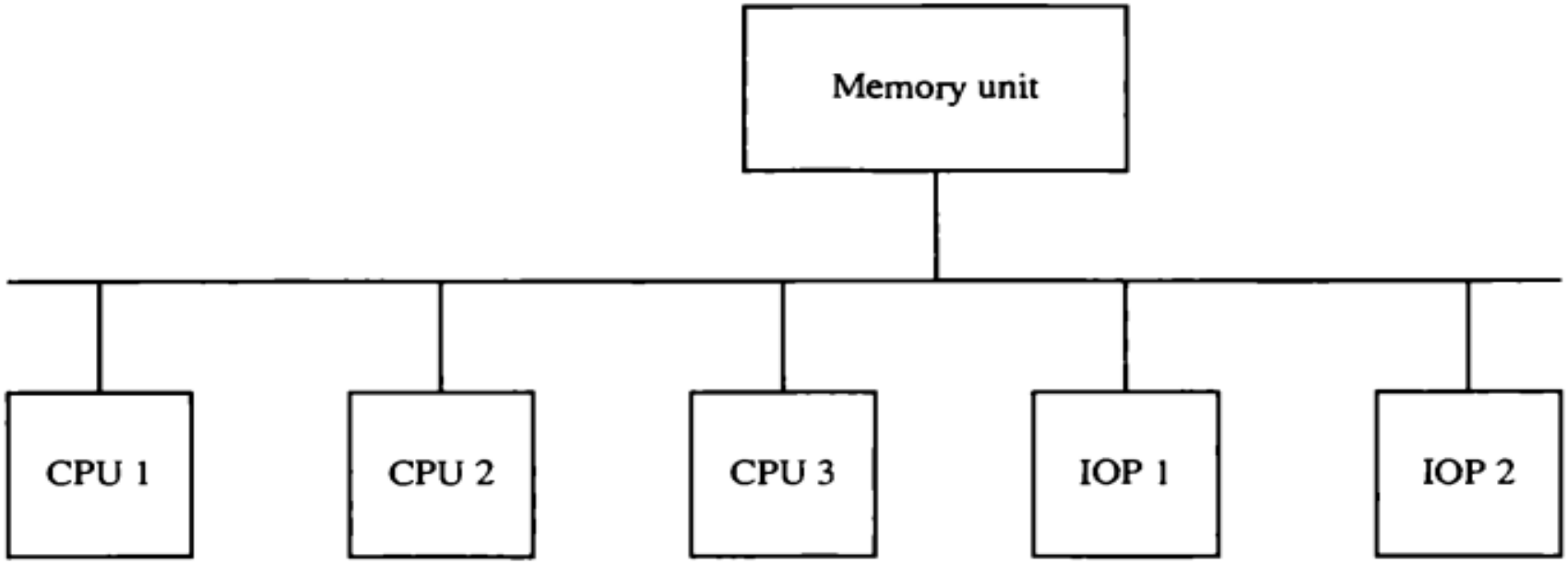
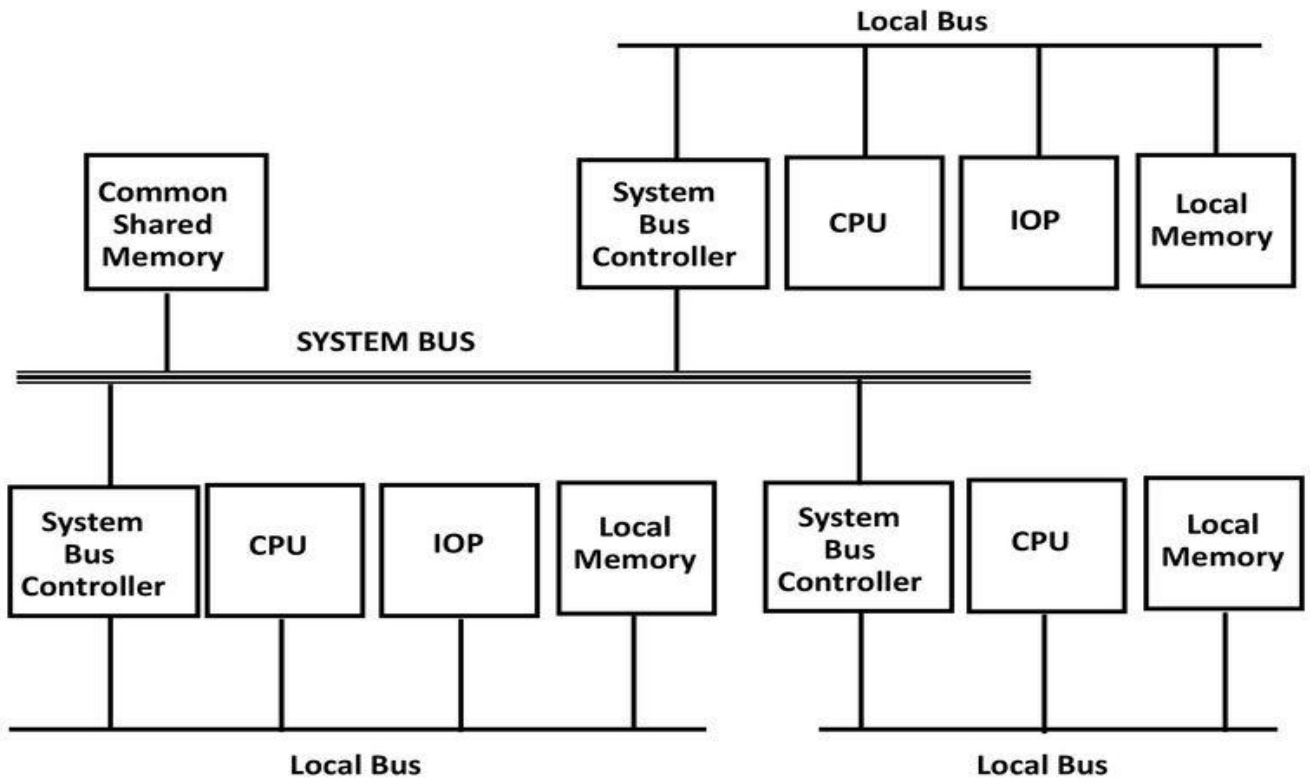


Figure 1 Time-shared common bus organization.

System Bus Structure for Multiprocessor



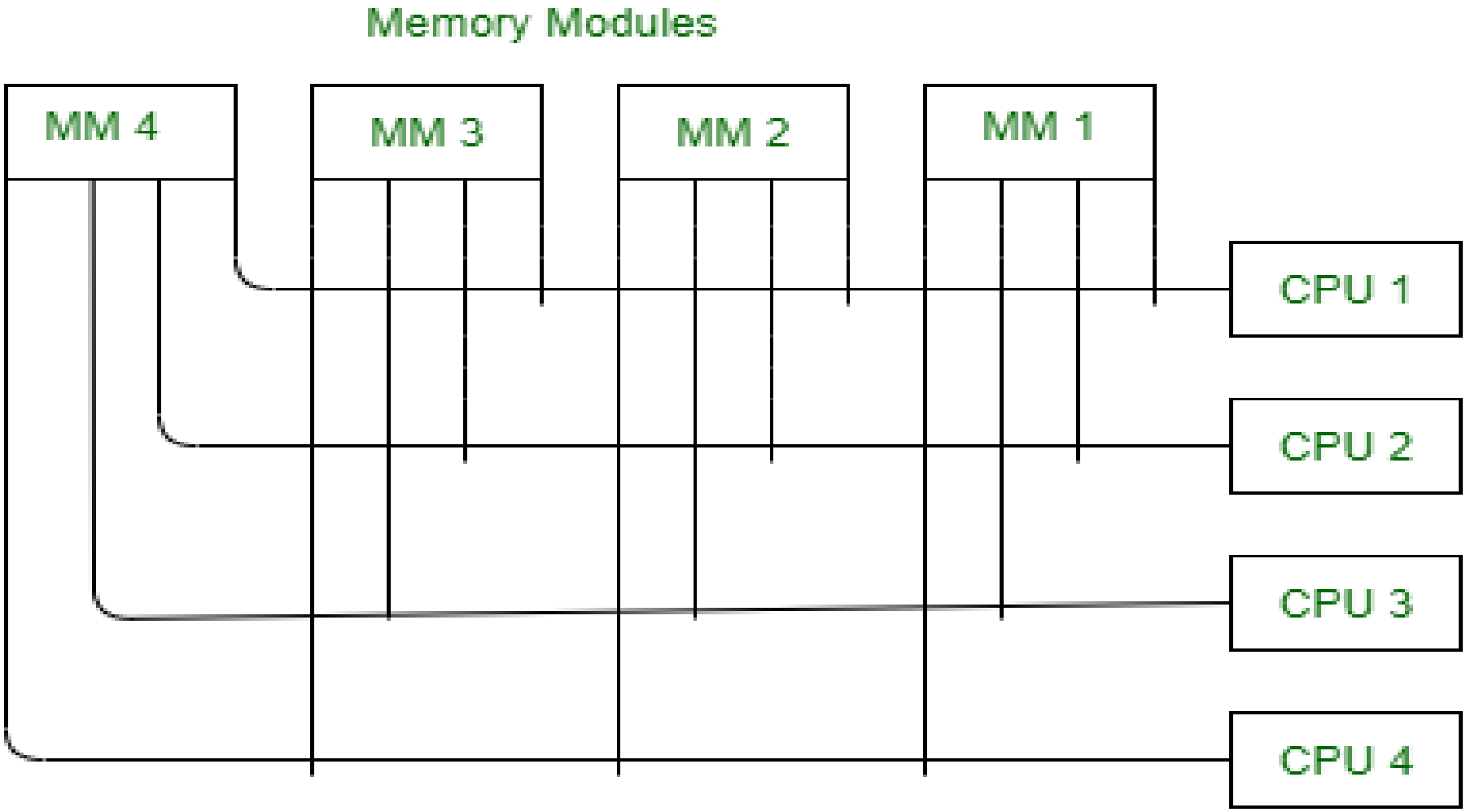
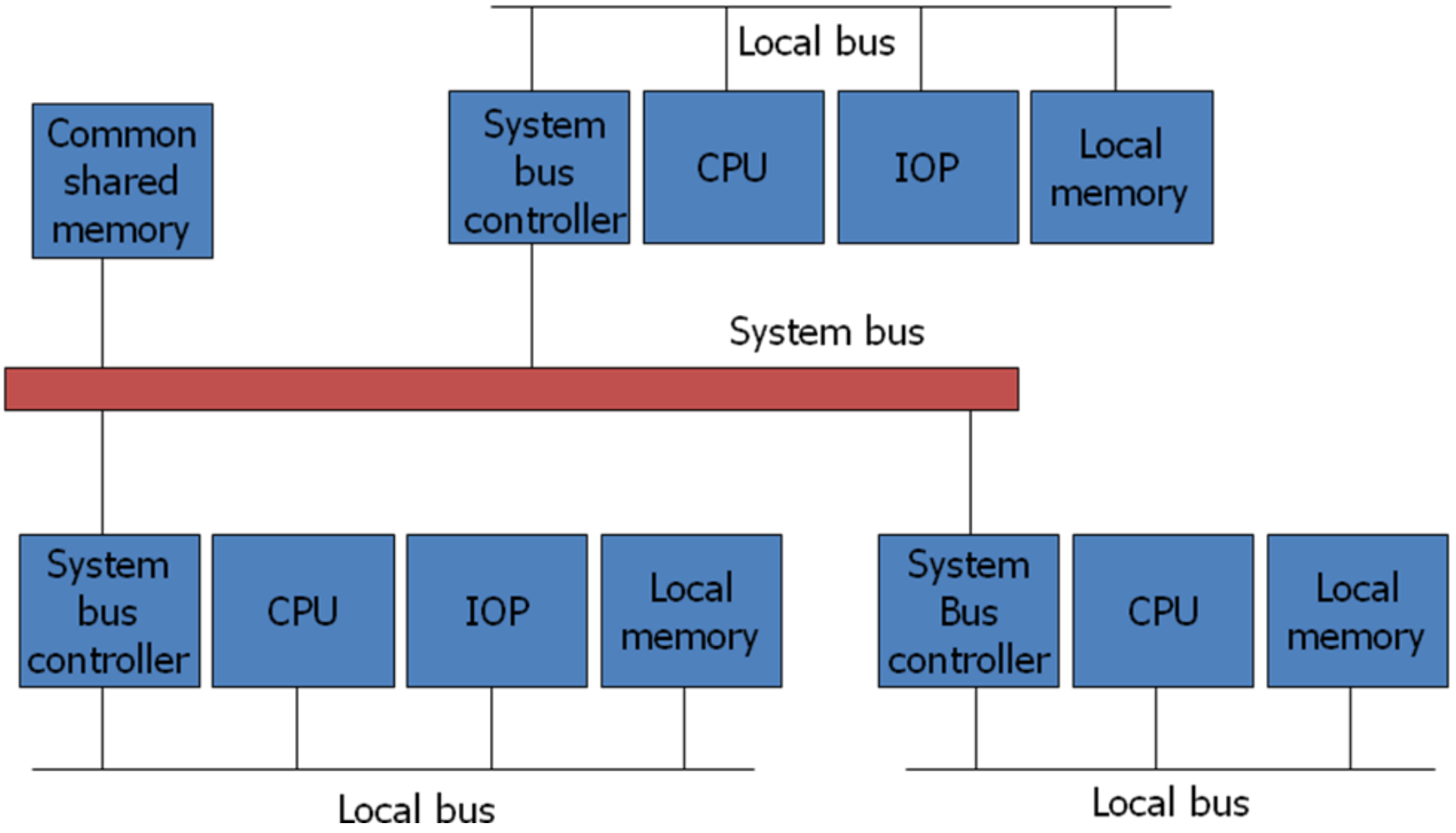
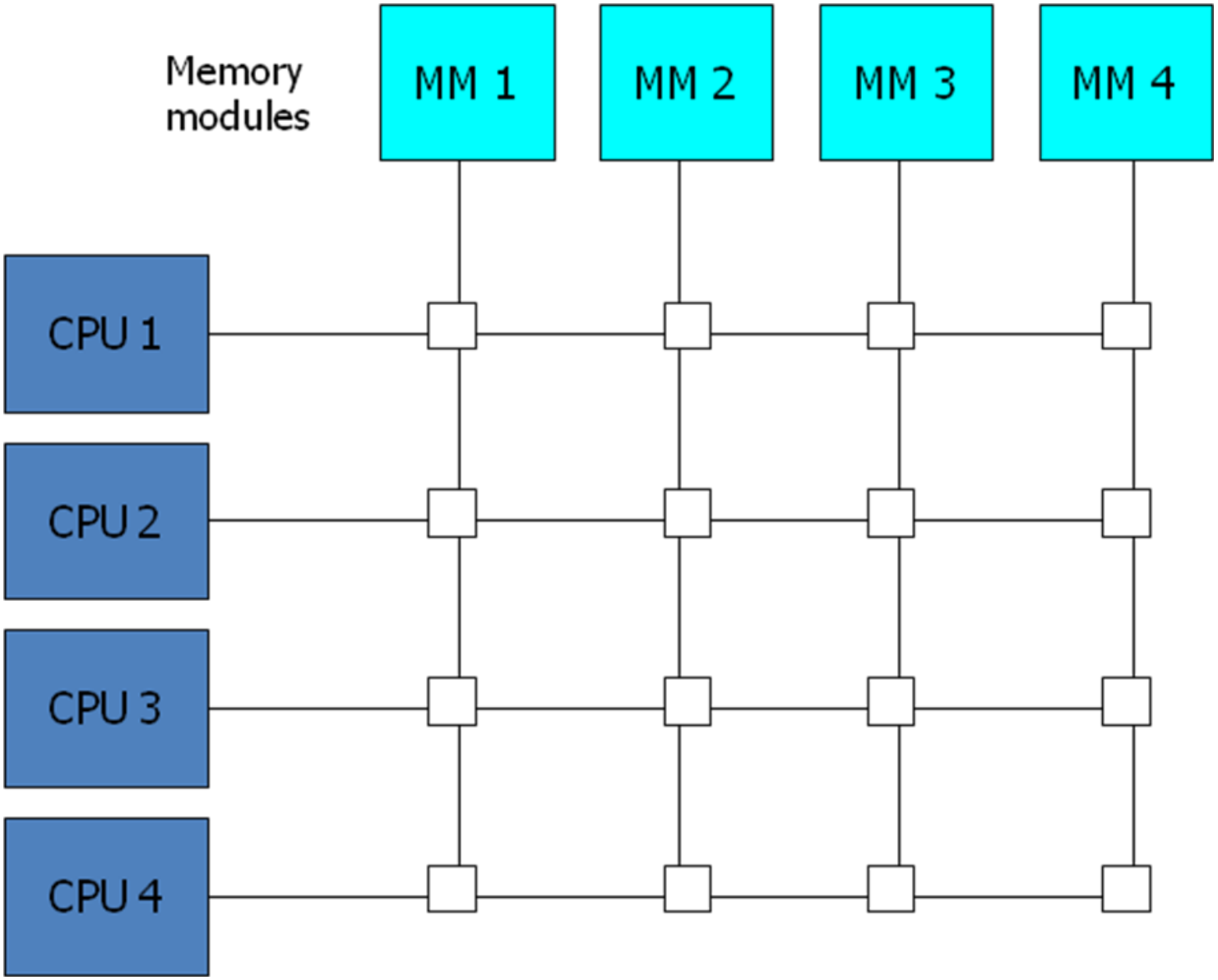


Figure - Multiport Memory System

System bus structure for multiprocessors

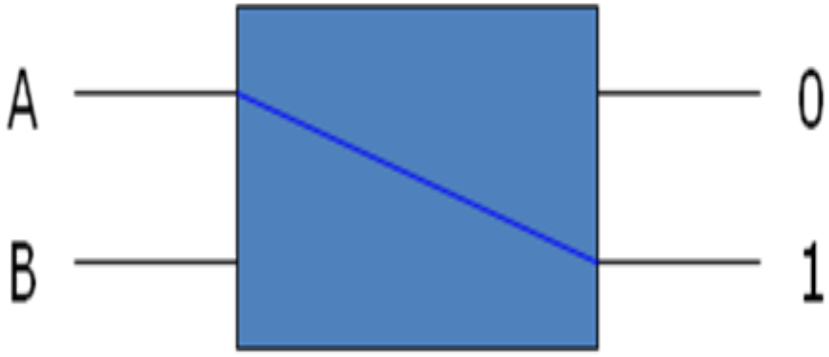


Crossbar switch

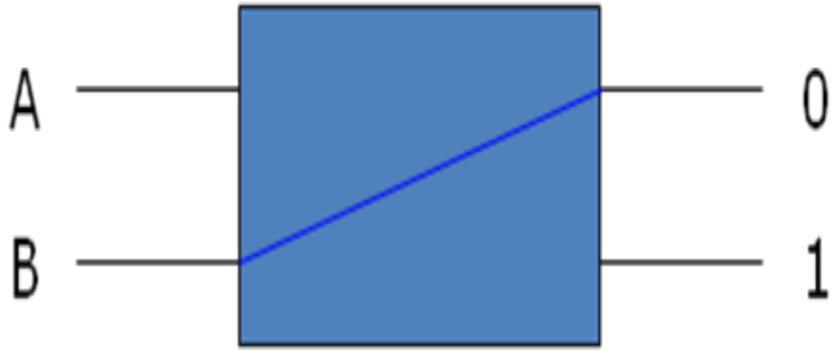


Multistage Switching Network

A connected to 0

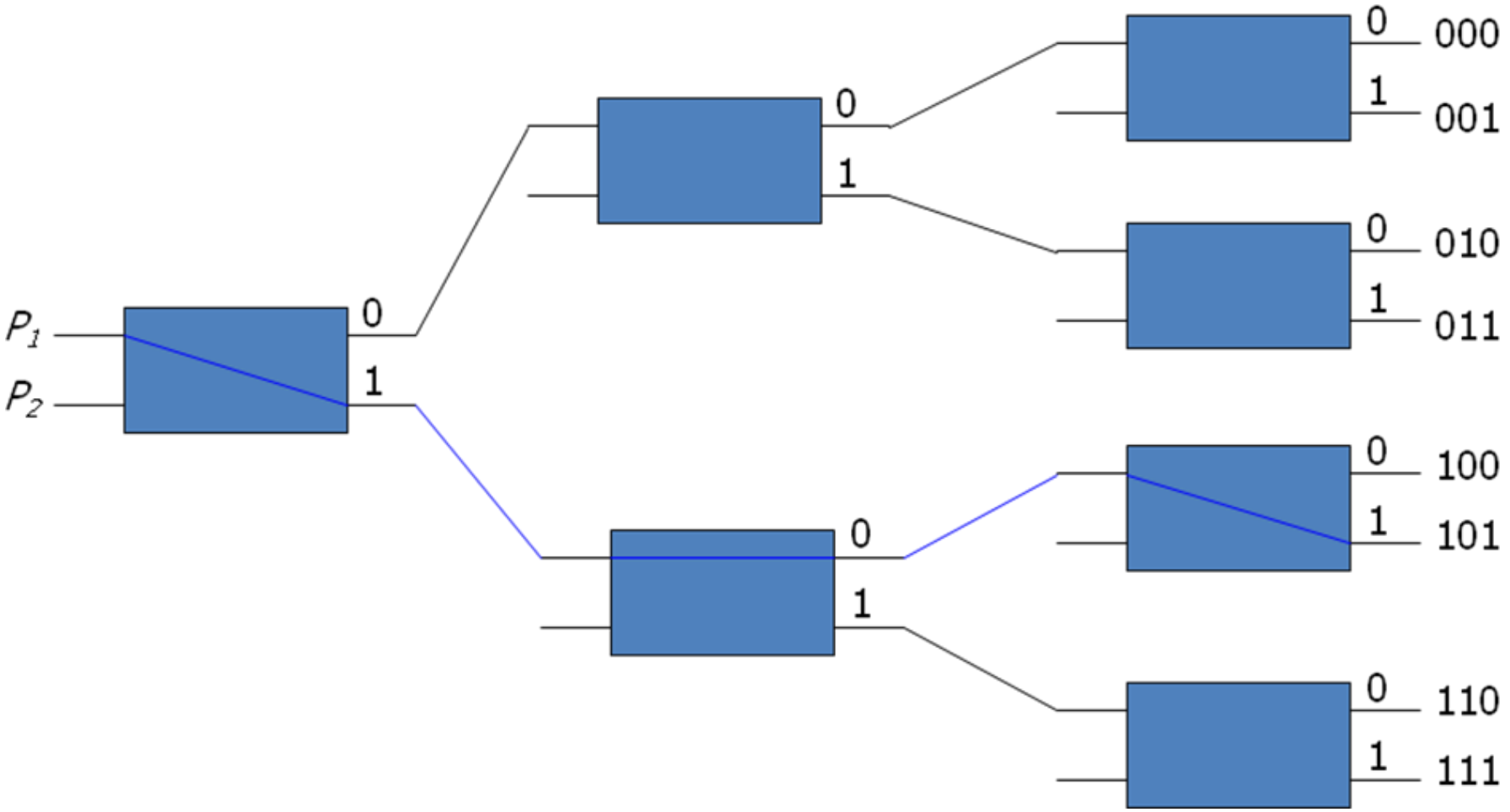


A connected to 1

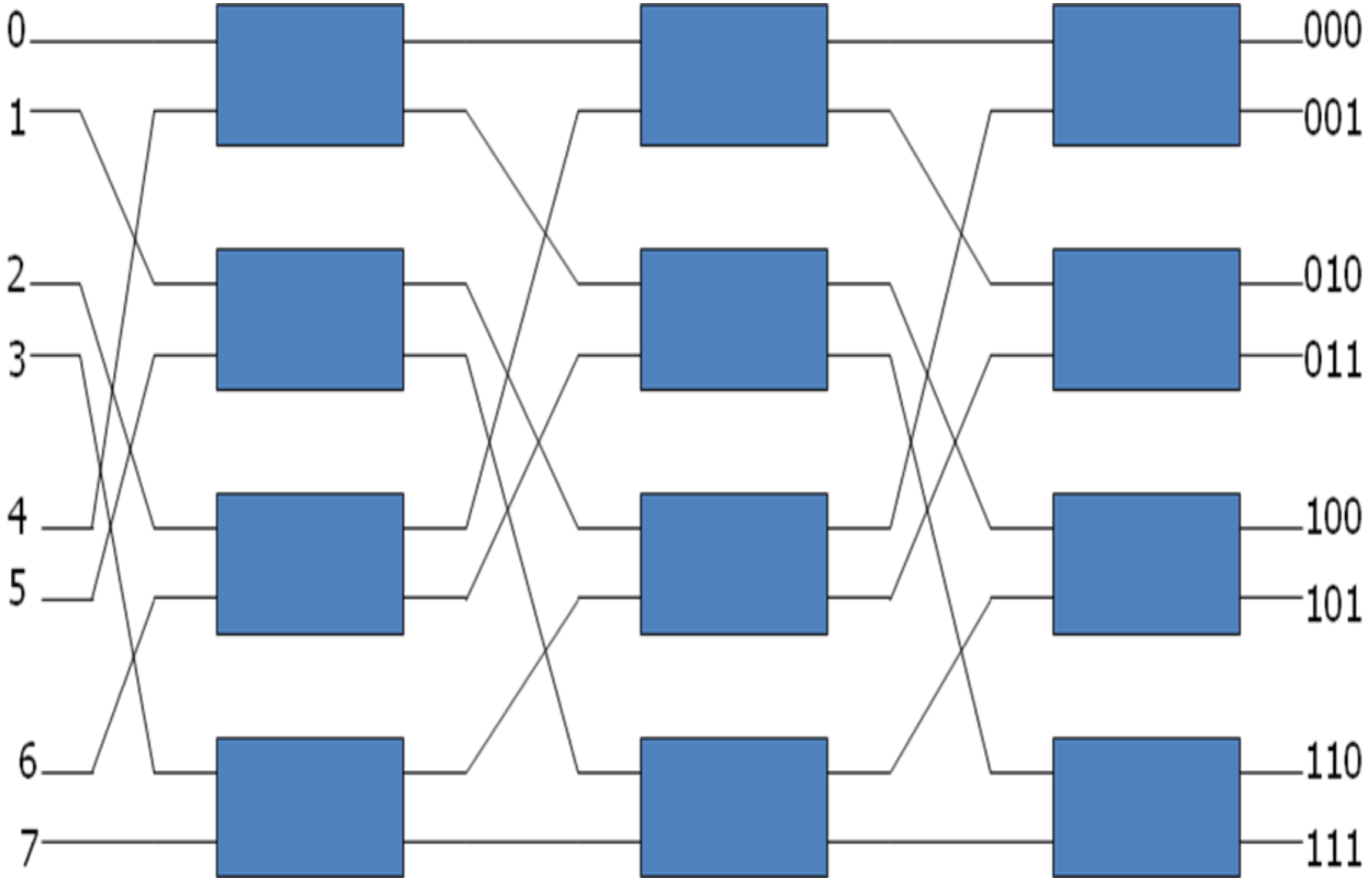


B connected to 0

B connected to 1


P_1
P_2
0
1
0
1
0
1
0
1
0
1
0
1
000
001
010
011
100
101
110
111

Omega switching network (8X8)



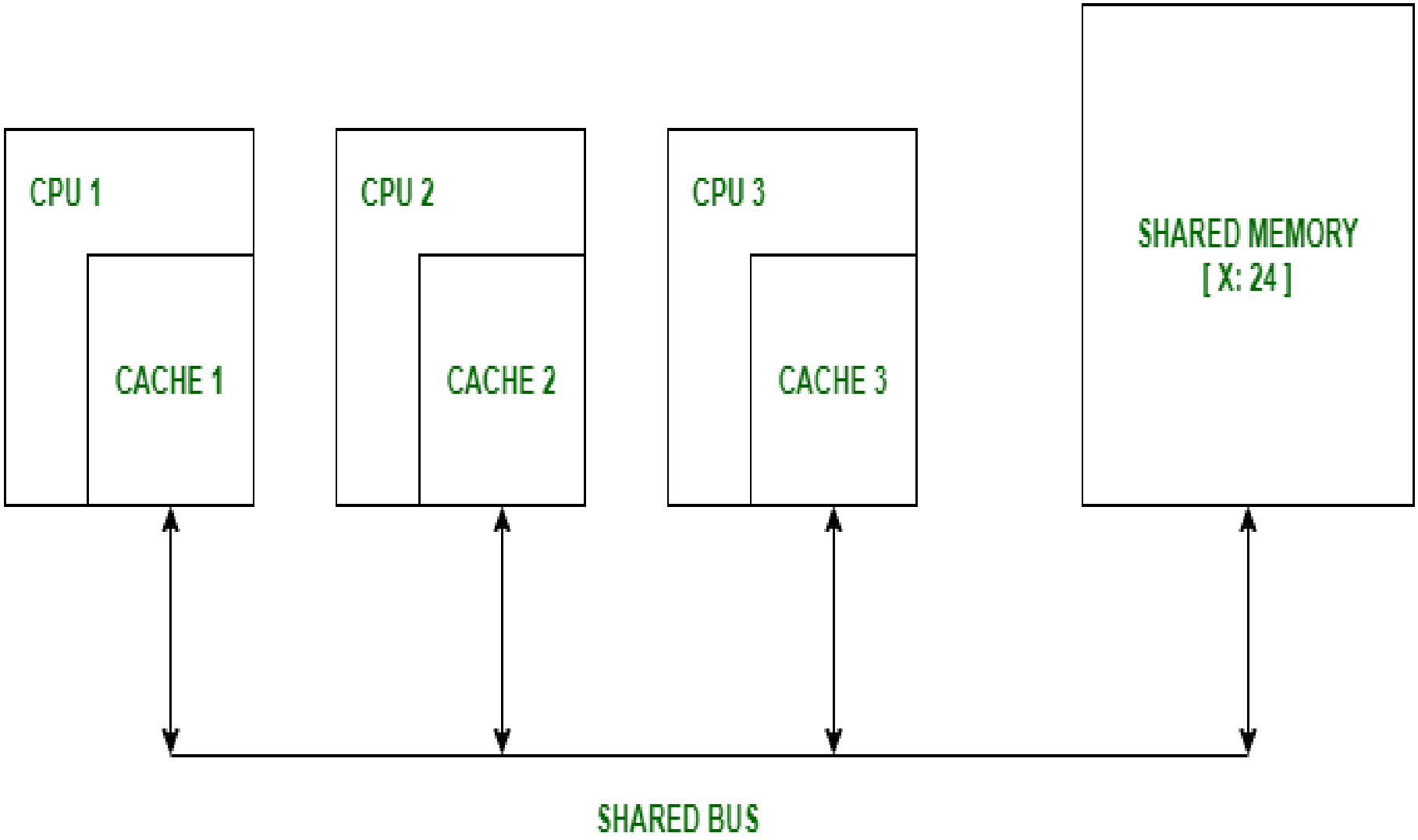
Cache coherence :

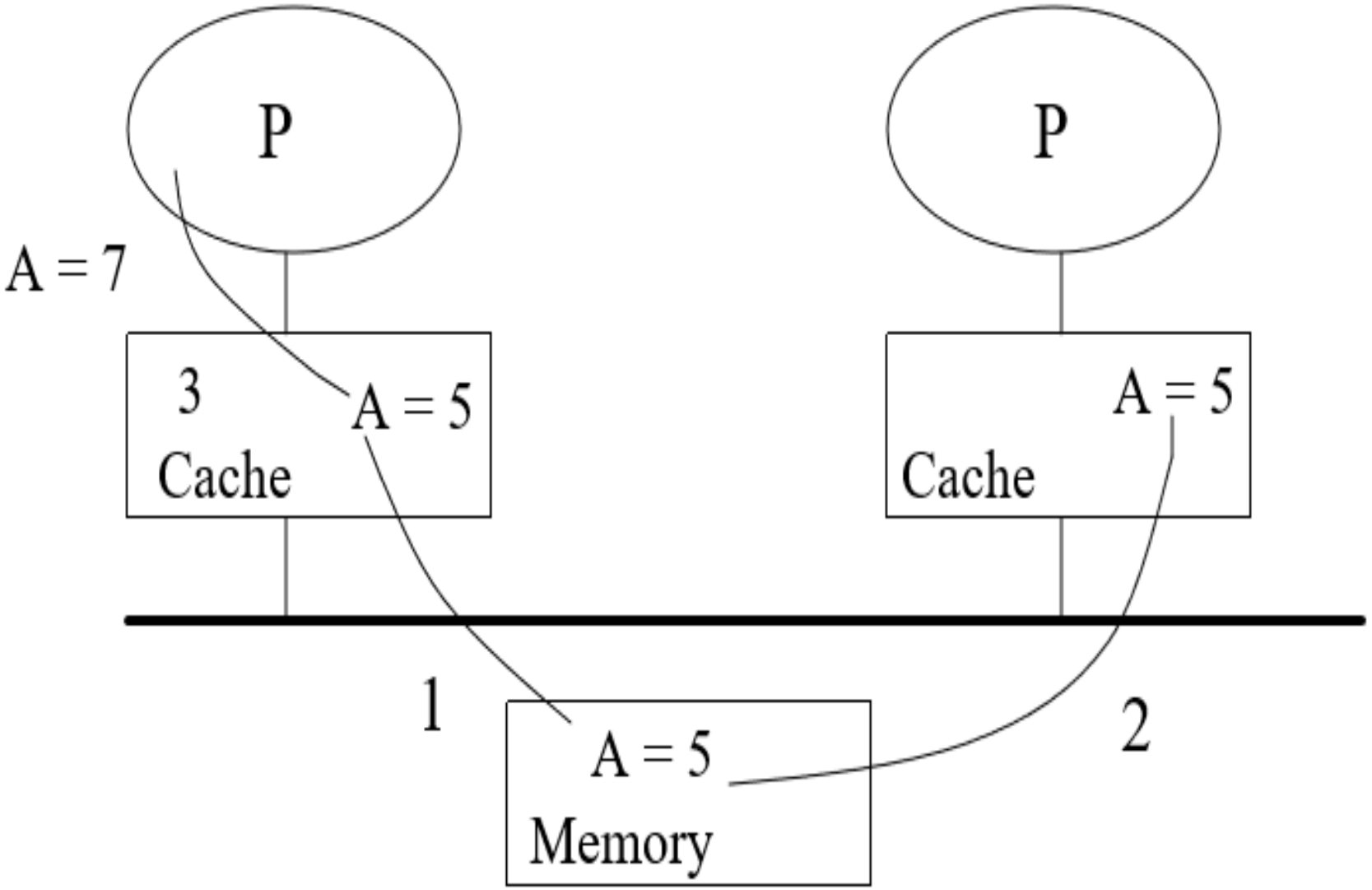
In a multiprocessor system, data inconsistency may occur among adjacent levels or within the same level of the memory hierarchy.

In a shared memory multiprocessor with a separate cache memory for each processor, it is possible to have many copies of any one instruction operand

One copy in the main memory and one in each cache memory. When one copy of an operand is changed, the other copies of the operand must be changed also.

Cache Coherence




P
P
A = 7
3
A = 5
Cache
A = 5
Cache
1
2
A = 5
Memory

Solutions to cache coherence

- **<u>Write Through Schemes:</u>**
- All processor writes result in :
- – update of local cache and a global bus write that :
  - updates main memory
  - invalidates/updates all other caches with that item
- Advantage : Simple to implement
- Disadvantages : Since ~15% of references are writes, this scheme consumes tremendous bus
- bandwidth. Thus only a few processors can be supported.