Organization & Architecture

Architecture: Computer Architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have direct impact on the logical execution of a program.

Examples include the instruction set, the number of bits used to represent various data types (e.g., number, character), I/O mechanisms and techniques for addressing memory.

Organization: Computer Organization refers to the operational units and their Interconnections that realize the architectural specifications

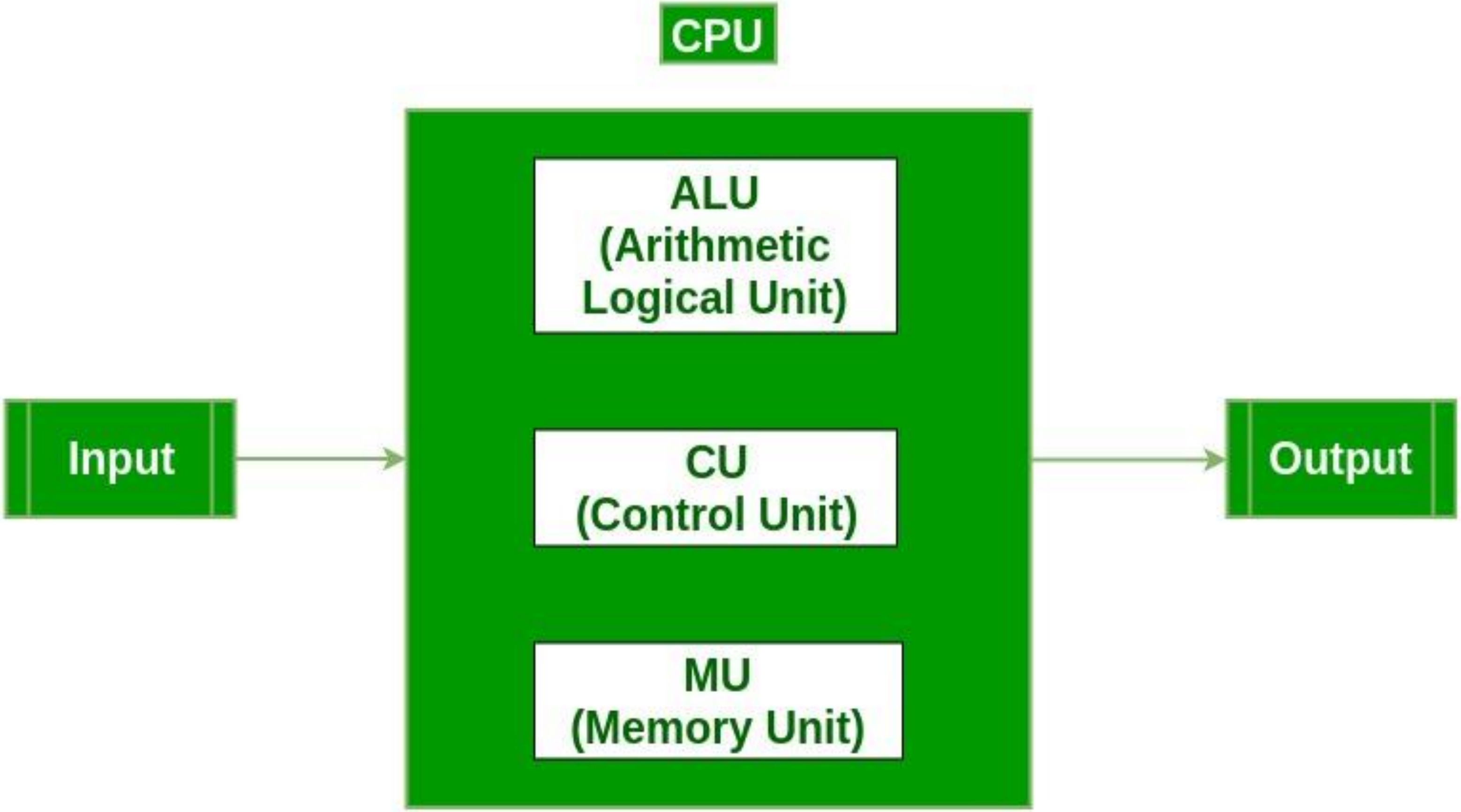
Examples include those hardware details transparent to the programmer, such as control signals, interfaces between the computer and peripherals and memory technology used.

INTRODUCTION

Introduction to digital computer

A computer is an electronic device which works under the given instruction. it takes the input from the user process that input and gives the output.

- **Input device:-**used to give input to the computer.
- **Output device**:-gives pressed data to the user.
- **Memory**:- it is a storage unit which stored data and information in the foam of binary(0,1).
- **Control unit**:-it controls all the components of the computer i.e it devices when to start receiving data ,when to stop it where to store data etc.
- **Arithmetic logical unit(A.L.U)**:- all the arithmetic and logical operation are perform here.
- **C.P.U**:- control process unit is the brain of the computer. It combine CU+ALU.



Digital computer

1. **Input Unit :**

- The input unit consists of input devices that are attached to the computer.
- These devices take input and convert it into binary language that the computer understands.
- Common input devices are keyboard, mouse, joystick, scanner etc.

2. **Central Processing Unit (CPU) :**

- The CPU is called the brain of the computer because it is the control centre of the computer.
- It first fetches instructions from memory and then interprets them so as to know what is to be done.
- CPU executes or performs the required computation and then either stores the output or displays on the output device

3. **Arithmetic and Logic Unit (ALU) :**

- It performs mathematical calculations and takes logical decisions.
- Arithmetic calculations include addition, subtraction, multiplication and division.
- Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.

4. **Control Unit :**

- It coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and input/output units.
- It decodes the fetched instruction, interprets it and sends control signals to input/output devices until the required operation is done properly by ALU and memory.

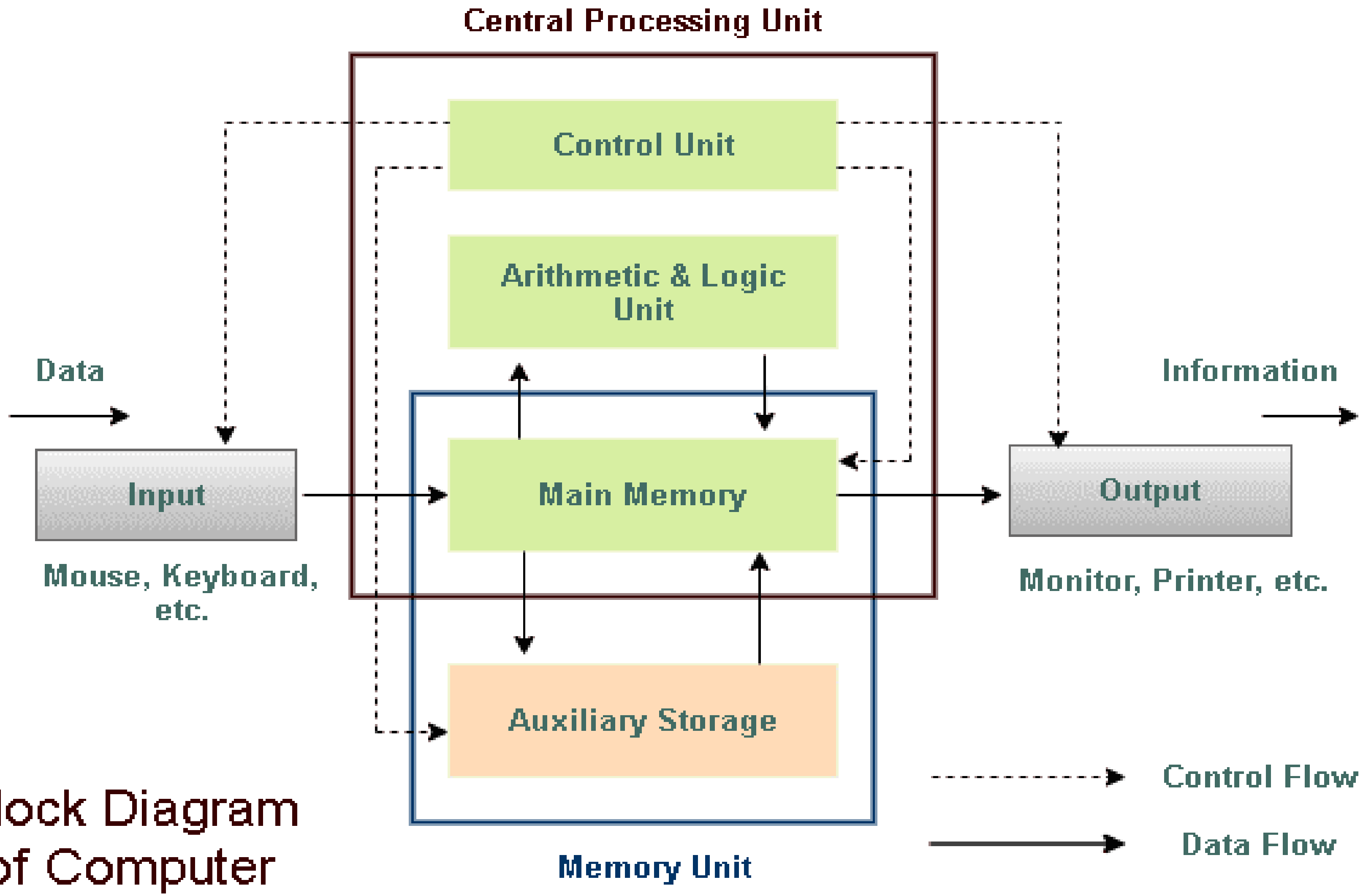
5. **Memory :**

- Memory attached to the CPU is used for storage of data and instructions and is called internal memory.
- The internal memory is divided into many storage locations, each of which can store data or instructions

6. **Output Unit :**

- The output unit consists of output devices that are attached with the computer.
- It converts the binary data coming from CPU to human understandable form.
- The common output devices are monitor, printer, plotter etc.

Block Diagram of Computer



1. **Primary / Main memory**:

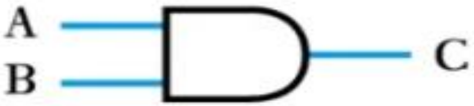
- The memory unit that establishes direct communication with the CPU is called **Main Memory.** The main memory is often referred to as RAM (Random Access Memory).
- **It holds the data and instructions that the processor is currently working on.**

2. **Secondary Memory / Mass Storage**:

- The contents of the secondary memory first get transferred to the primary memory and then are accessed by the processor, this is because the processor does not directly interact with the secondary memory.
- The memory units that provide backup storage are called **Auxiliary Memory**. For instance, magnetic disks and magnetic tapes are the most commonly used auxiliary memories.

Logic Gates

The AND gate



(a) Circuit symbol

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

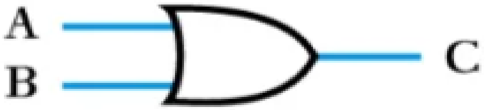
(b) Truth table

$$C = A \cdot B$$

(c) Boolean expression

Logic Gates

The OR gate



(a) Circuit symbol

A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

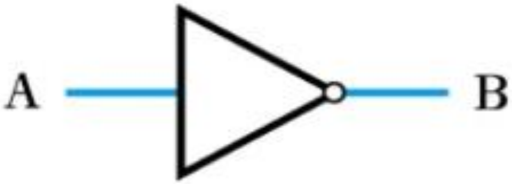
(b) Truth table

$$C = A + B$$

(c) Boolean expression

Logic Gates

The NOT gate (or inverter)



(a) Circuit symbol

A	B
0	1
1	0

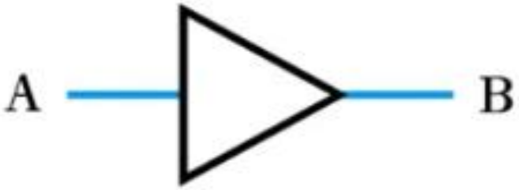
(b) Truth table

$$B = \overline{A}$$

(c) Boolean expression

Logic Gates

A logic buffer gate



(a) Circuit symbol

A	B
0	0
1	1

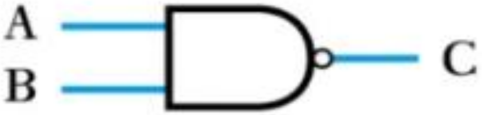
(b) Truth table

$B = A$

(c) Boolean expression

Logic Gates

The NAND gate



(a) Circuit symbol

A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

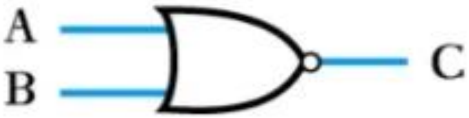
(b) Truth table

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

(c) Boolean expression

Logic Gates

The NOR gate



(a) Circuit symbol

A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

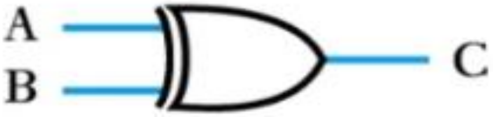
(b) Truth table

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

(c) Boolean expression

Logic Gates

The Exclusive OR gate



(a) Circuit symbol

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

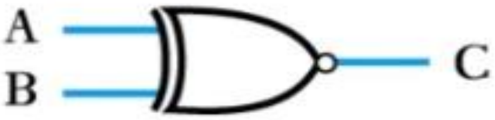
(b) Truth table

$$C = A \oplus B$$

(c) Boolean expression

Logic Gates

The Exclusive NOR gate



(a) Circuit symbol

A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

(b) Truth table

$$C = \overline{A \oplus B}$$

(c) Boolean expression

Multiplexer

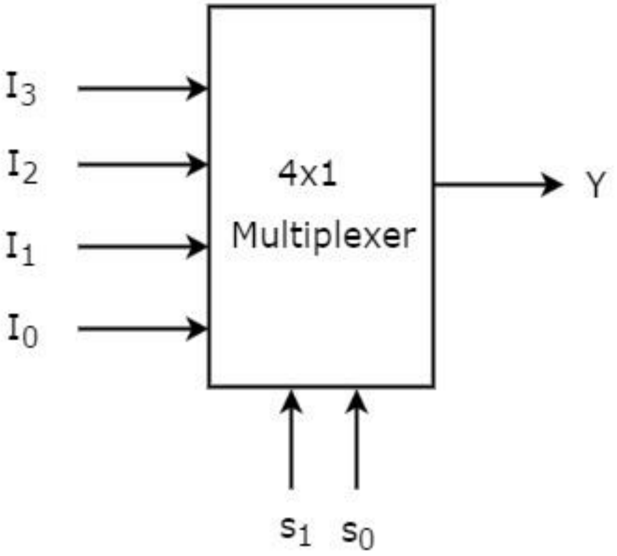
Multiplexer (MUX)

- It is a combinational circuit that has maximum of 2^n data inputs, ‘n’ selection lines and single output line.
- One of these data inputs will be connected to the output based on the values of selection lines.
- Multiplexers are also known as **“Data n selector, parallel to serial convertor, many to one circuit, universal logic circuit”**

4x1 Multiplexer

4x1 Multiplexer

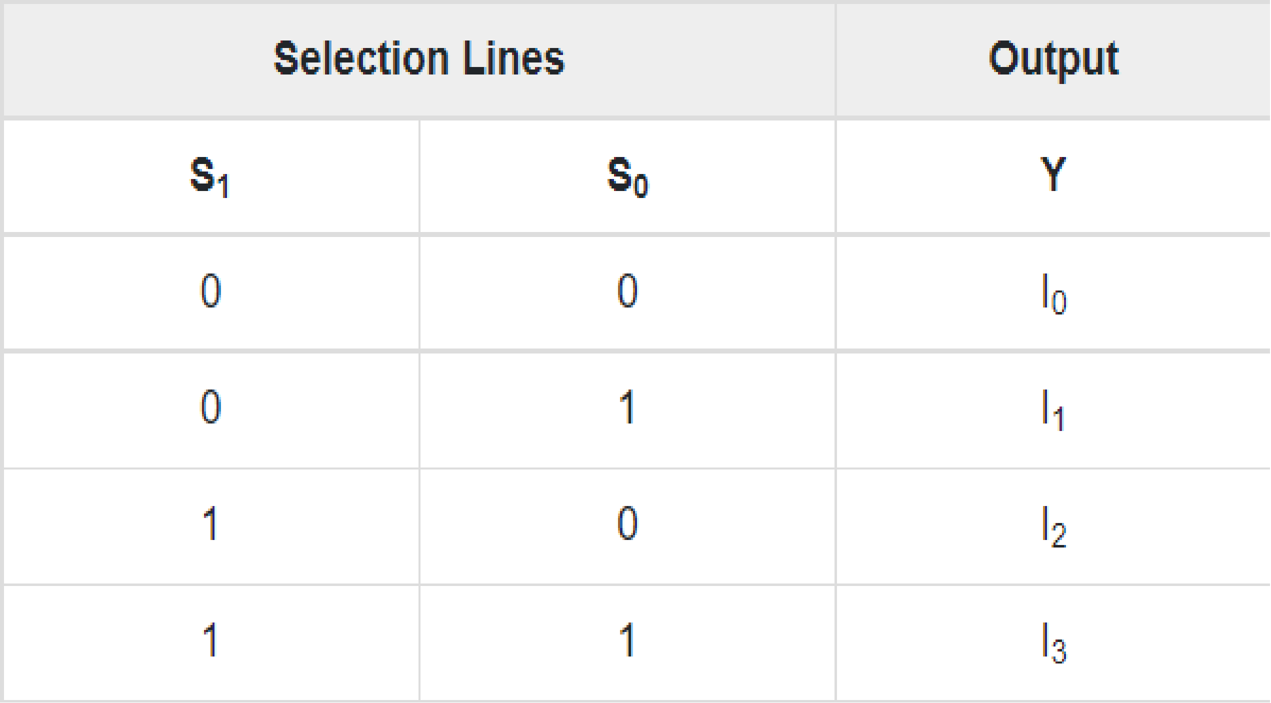
- 4x1 Multiplexer has four data inputs I_3, I_2, I_1 & I_0, two selection lines s_1 & s_0 and one output Y.
- The block diagram of 4x1 Multiplexer is shown in the following figure.



4x1 Multiplexer

- One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines.

- **Truth table** of 4x1 Multiplexer is shown below.



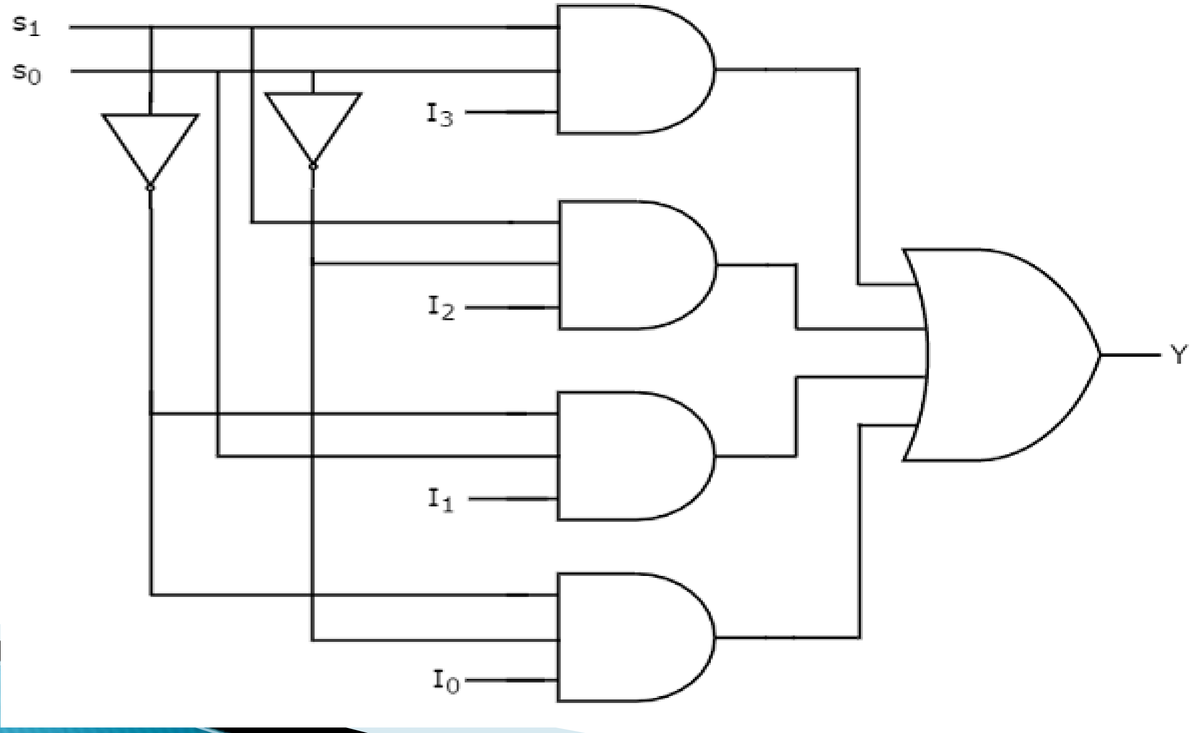
Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

4x1 Multiplexer

- From Truth table, we can directly write the **Boolean function** for output, Y as

$$Y = S_1{}'S_0{}'I_0 + S_1{}'S_0 I_1 + S_1 S_0{}'I_2 + S_1 S_0 I_3$$

- This Boolean function can be implemented using Inverters, AND gates & OR gate. The **circuit diagram** of 4x1 multiplexer is shown in the following figure.



De-Multiplexer

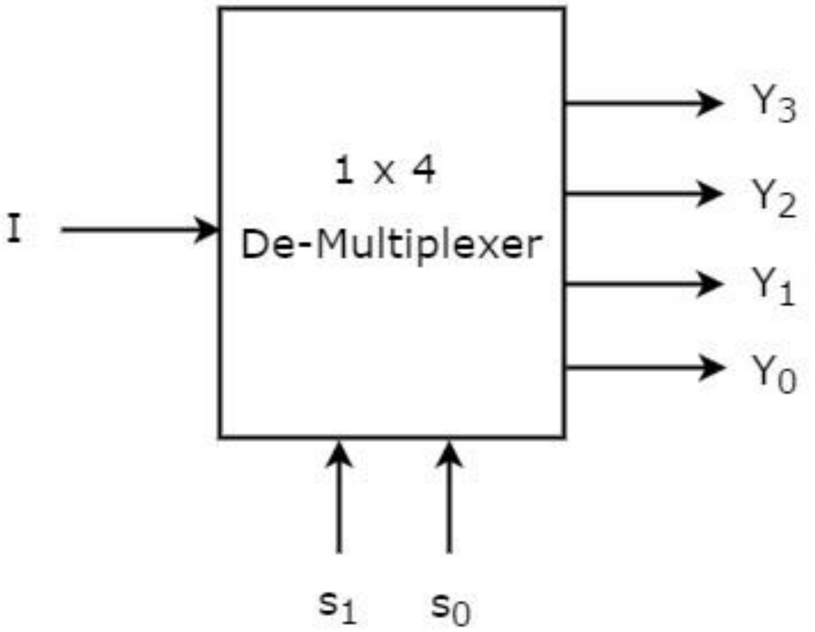
De-Multiplexer (De-MUX)

- It is a combinational circuit that performs the reverse operation of Multiplexer.
- It has single input, ‘n’ selection lines and maximum of 2^n outputs.
- The input will be connected to one of these outputs based on the values of selection lines.
- They are also known as **“Data distributor, serial to parallel convertor, one to many circuit”**

1x4 De-Multiplexer

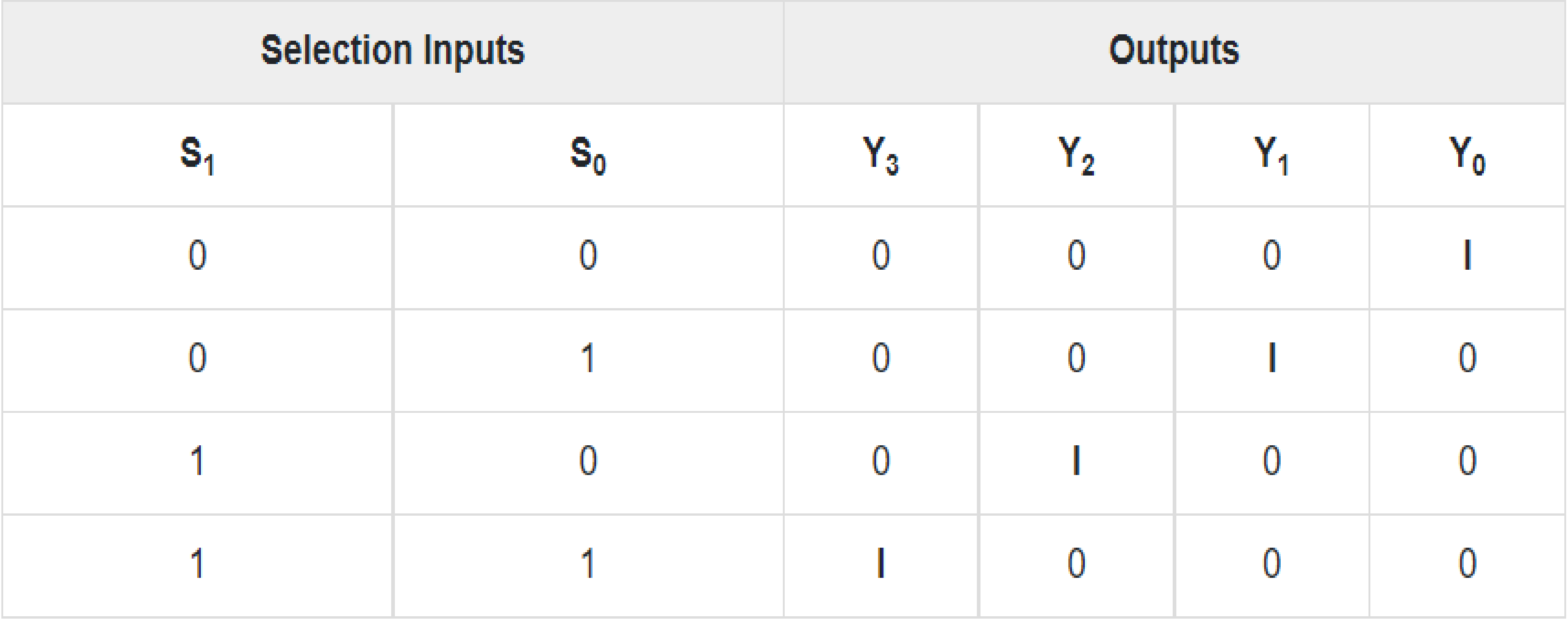
1x4 De-Multiplexer

- 1x4 De-Multiplexer has one input I, two selection lines, s_1 & s_0 and four outputs Y_3, Y_2, Y_1 &Y_0.
- The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



1x4 De-Multiplexer

- The single input ‘I’ will be connected to one of the four outputs, Y_3 to Y_0 based on the values of selection lines s_1 & s0.
- The **Truth table** of 1x4 De-Multiplexer is shown below.



Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

1x4 De-Multiplexer

- From the above Truth table, we can directly write the **Boolean functions** for each output as

$$Y_3 = s_1 s_0 I$$

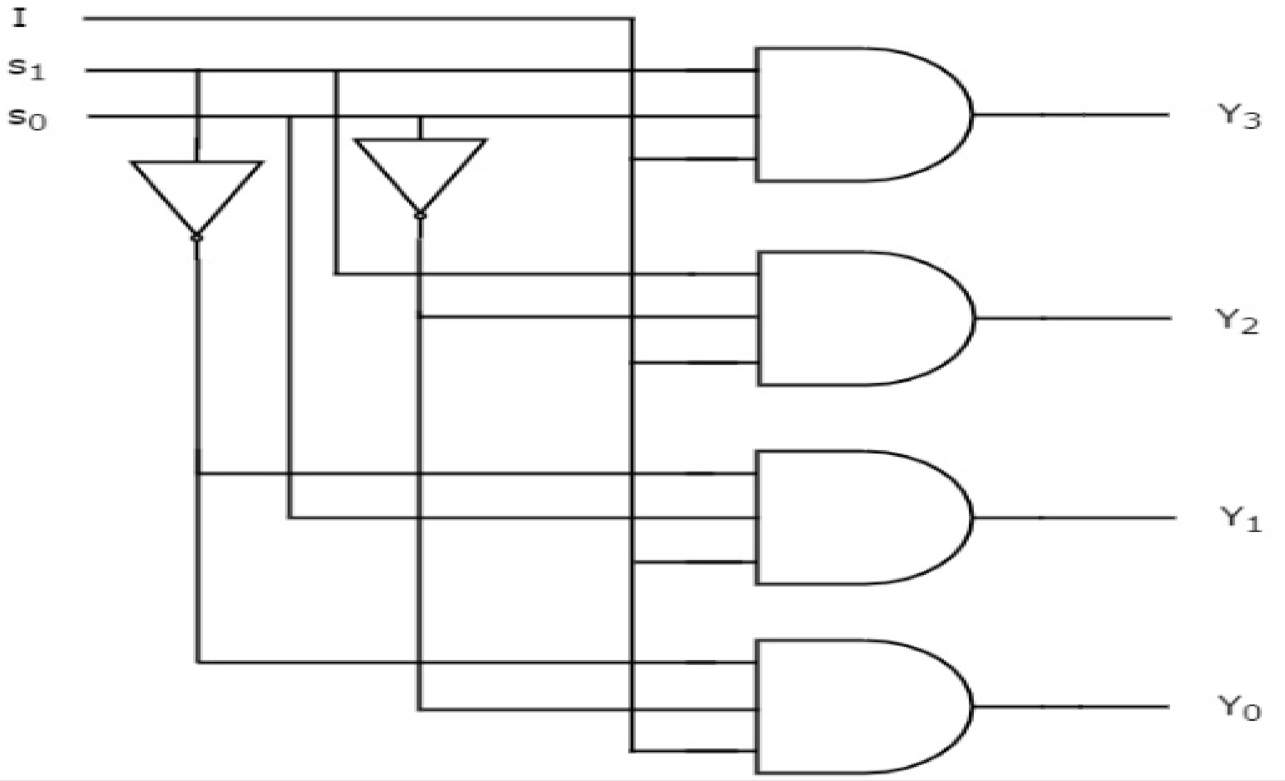
$$Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I$$

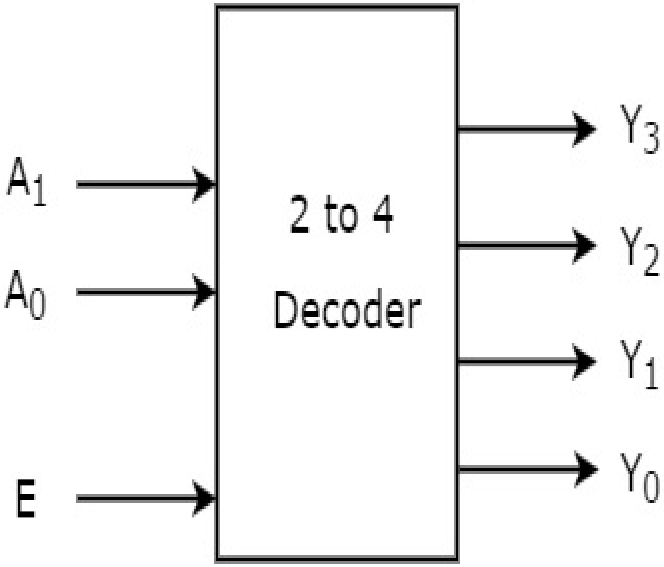
1x4 De-Multiplexer

- These Boolean functions are implemented using Inverters & 3-input AND gates.
- The circuit diagram of 1x4 De-Multiplexer is shown in the following figure.



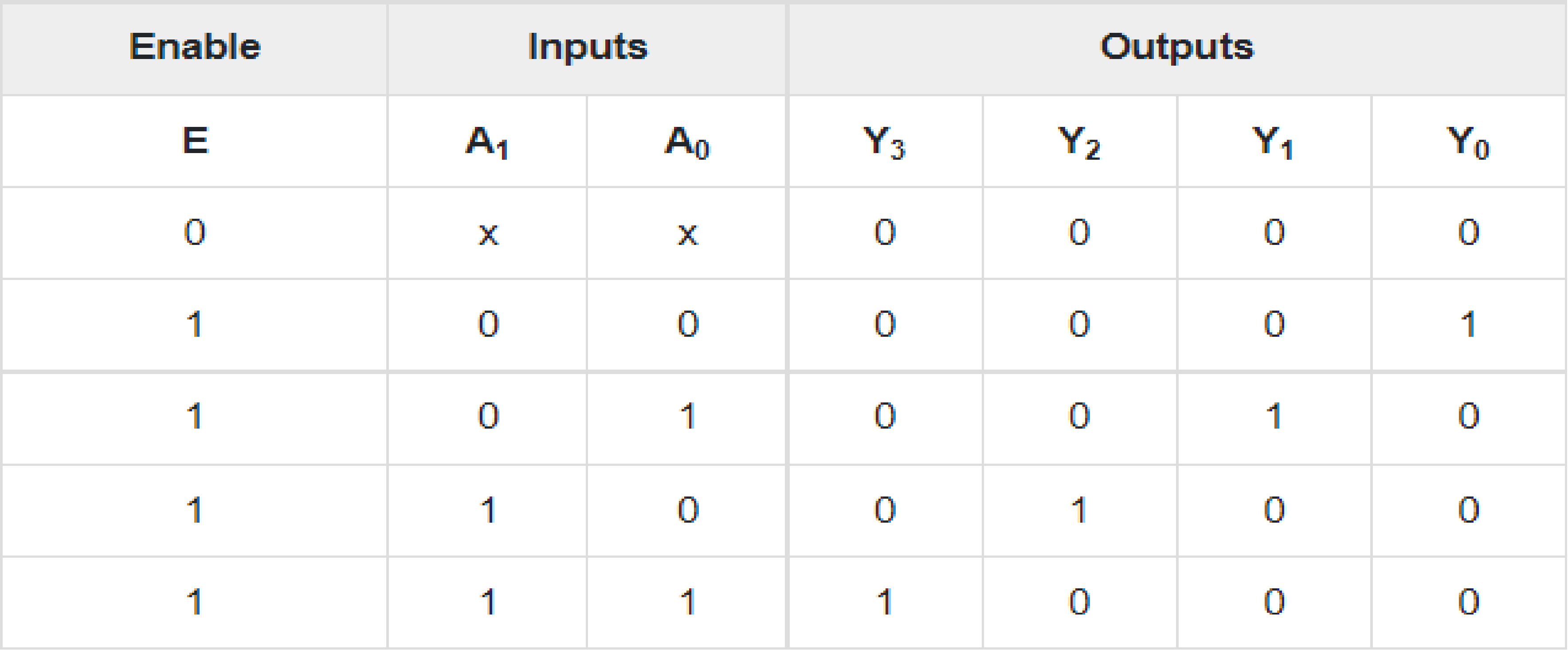
2 to 4 Decoder

- Let 2 to 4 Decoder has two inputs A_1 & A_0 and four outputs Y_3, Y_2, Y_1 & Y_0.
- The **block diagram** of 2 to 4 decoder is shown in the following figure.



2 to 4 Decoder

- One of these four outputs will be ‘1’ for each combination of inputs when enable, E is ‘1’.
- The Truth table of 2 to 4 decoder is shown below-



Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

2 to 4 Decoder

- From Truth table, the **Boolean functions** for each output is

$$Y_3 = E.\,A_1\,.\,A_0$$

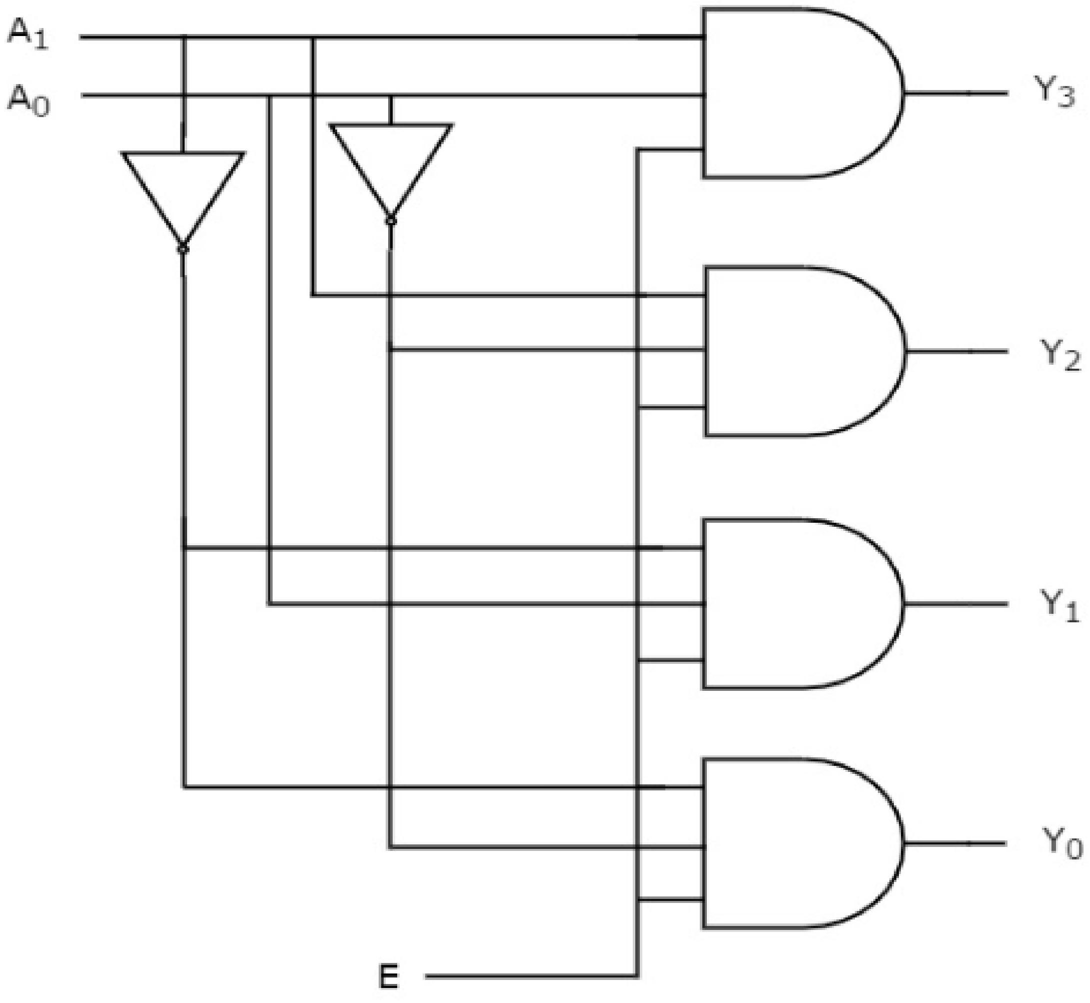
$$Y_2 = E.\,A_1\,.\,A_0{}'$$

$$Y_1 = E.\,A_1{}'.\,A_0$$

$$Y_0 = E.\,A_1{}'.\,A_0{}'$$

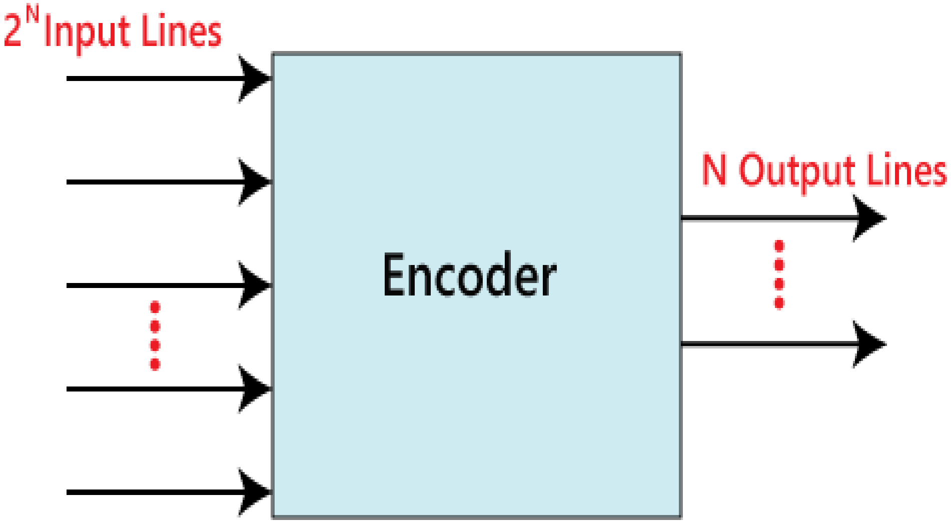
2 to 4 Decoder

- The circuit diagram of 2 to 4 decoder is shown in the following figure



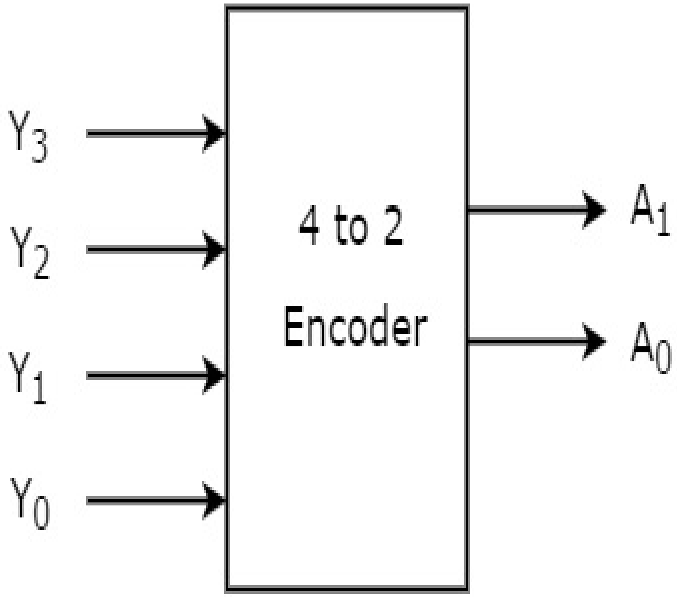
Encoder

- An **Encoder** is a combinational circuit that performs the reverse operation of Decoder.
- It has maximum of 2^n input lines and ‘n’ output lines.
- It will produce a binary code equivalent to the input, which is active High.
- Therefore, the **encoder encodes 2^n input lines with ‘n’ bits.** It is optional to represent the enable signal in encoders.



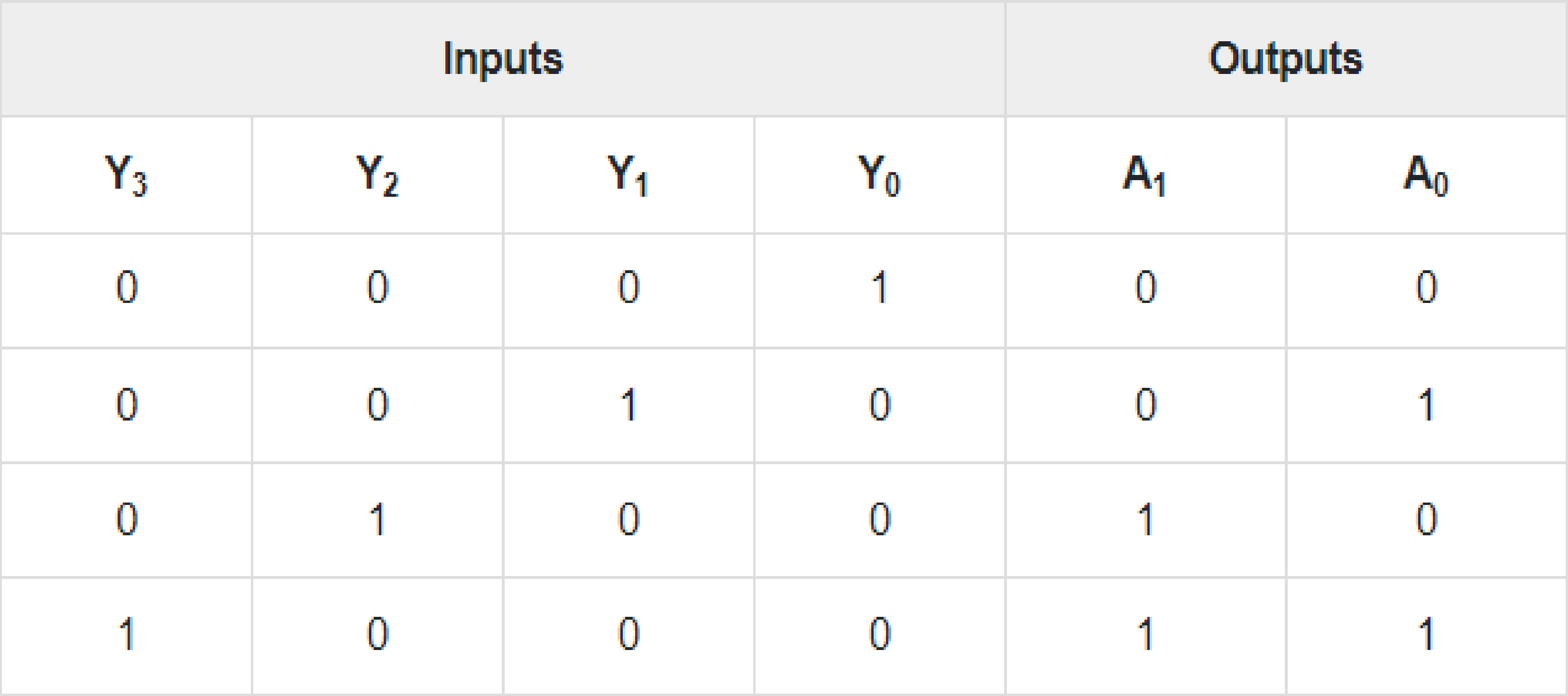
4 to 2 Encoder

- Let 4 to 2 Encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0.
- The block diagram of 4 to 2 Encoder is shown in the following figure.



4 to 2 Encoder

- **At any time, only one of these 4 inputs can be '1' in order to get the respective binary code at the output**.
- The Truth table of 4 to 2 encoder is shown below



Inputs				Outputs	
Y_3	Y_2	Y_1	Y_0	A_1	A_0
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

4 to 2 Encoder

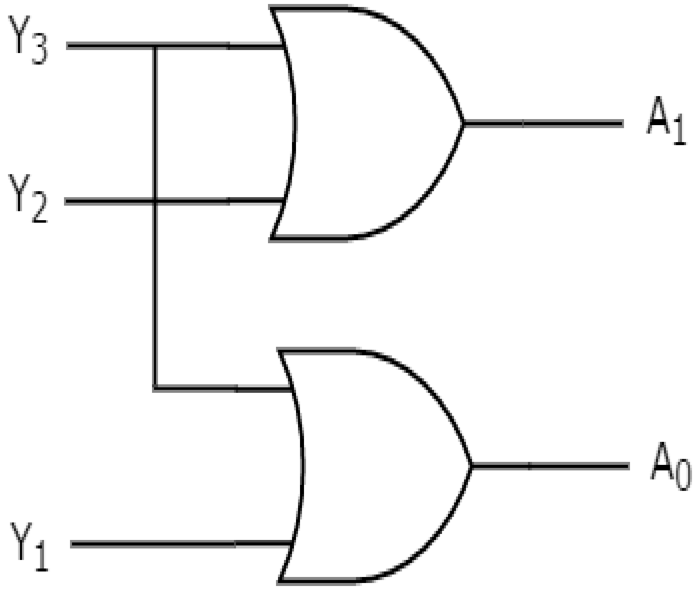
- From Truth table, the Boolean functions for each output is

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

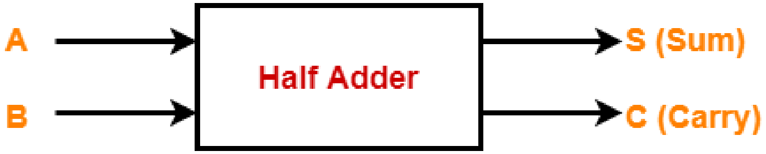
4 to 2 Encoder

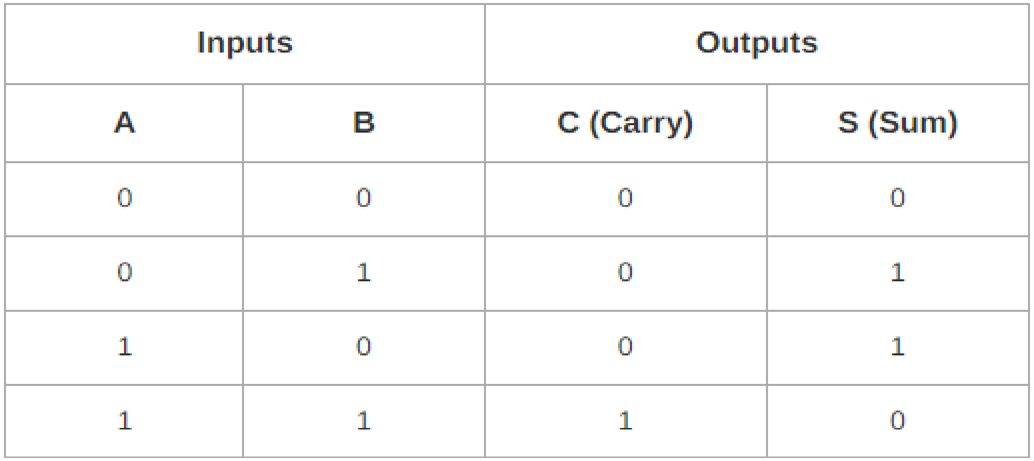
- We can implement the above two Boolean functions by using two input OR gates.
- The circuit diagram of 4 to 2 encoder is shown in the following figure



HALF ADDER

- Half Adder is a combinational logic circuit.
- It is used for the purpose of adding two single bit numbers.
- It contains 2 inputs and 2 outputs (sum and carry).





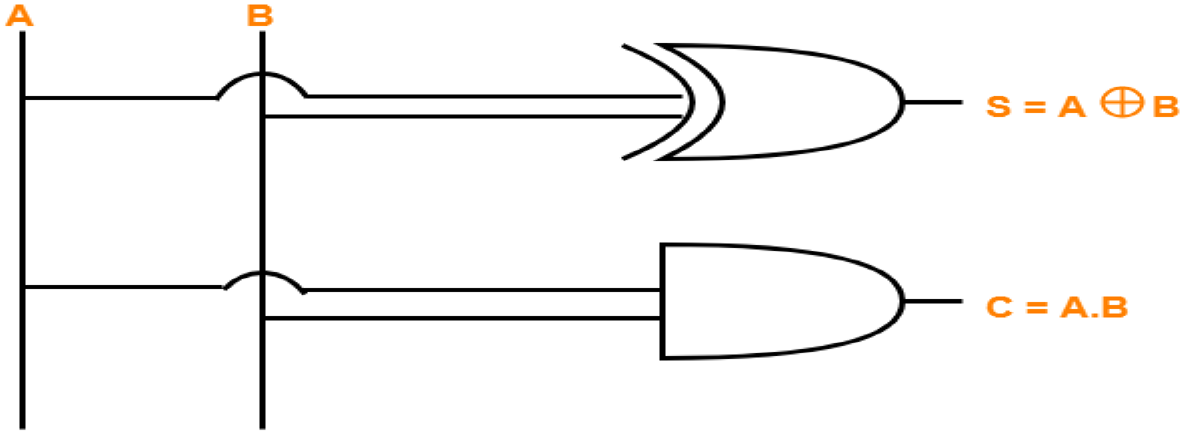
Inputs		Outputs	
A	B	C (Carry)	S (Sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Truth Table

HALF ADDER

Limitation of Half Adder-

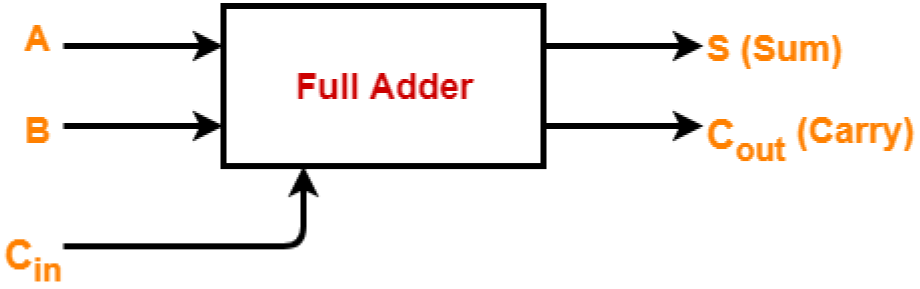
- Half adders have no scope of adding the carry bit resulting from the addition of previous bits.
- This is a major drawback of half adders.
- This is because real time scenarios involve adding the multiple number of bits which can not be accomplished using half adders.



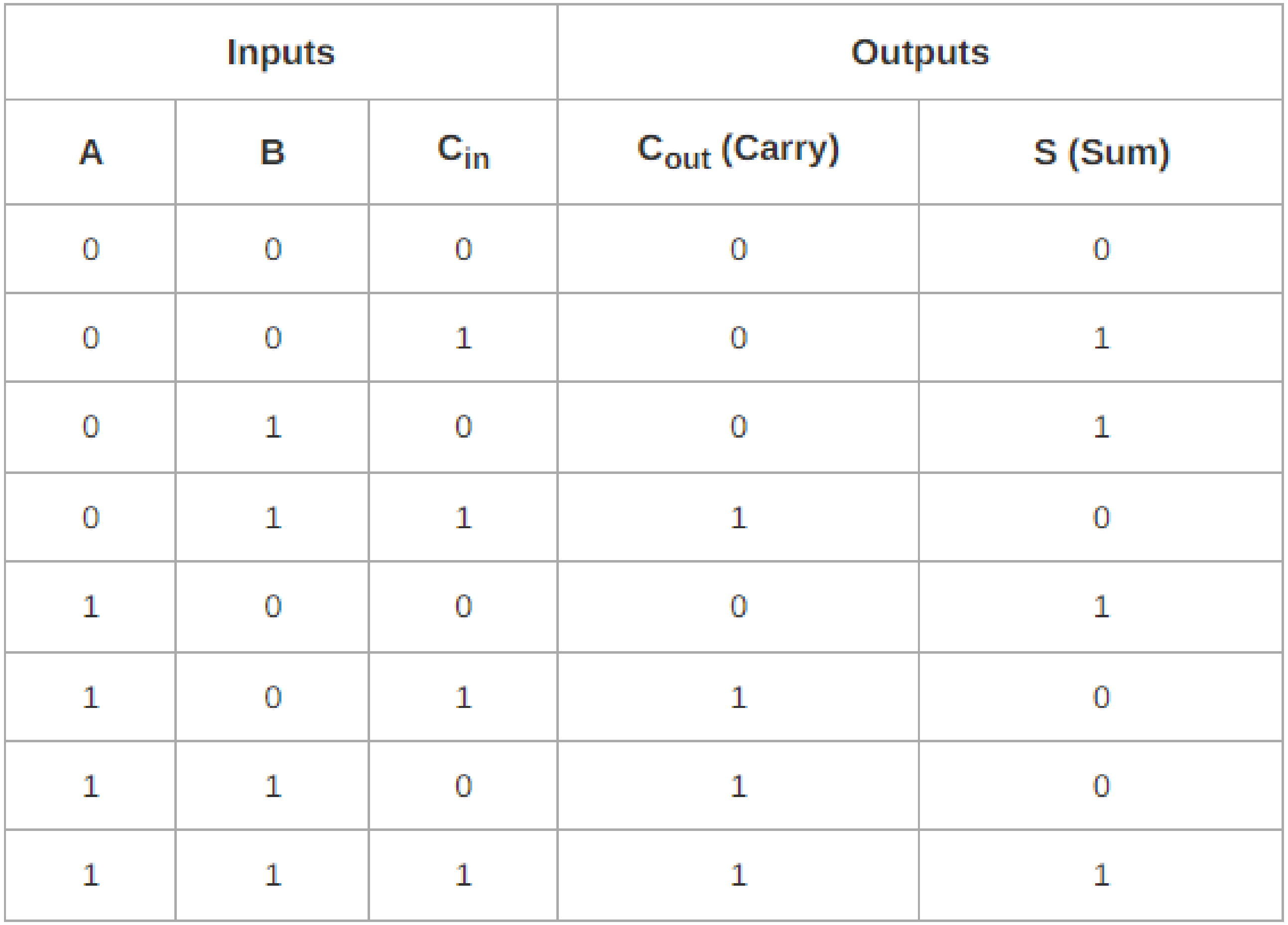
Half Adder Logic Diagram

FULL ADDER

- Full Adder is a combinational logic circuit.
- It is used for the purpose of adding two single bit numbers with a carry.
- Thus, full adder has the ability to perform the addition of three bits.
- Full adder contains 3 inputs and 2 outputs (sum and carry) as shown-



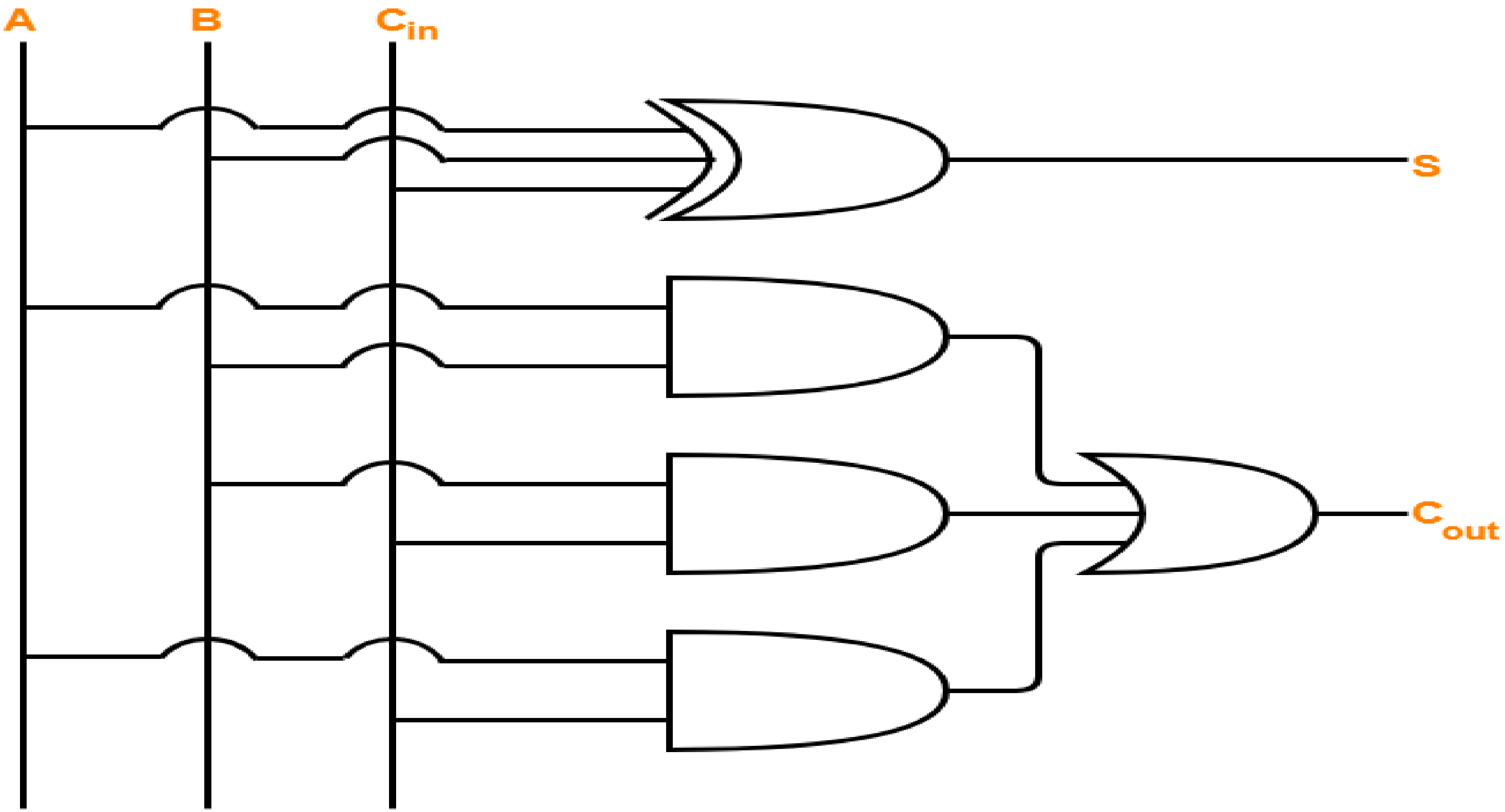
FULL ADDER



Inputs			Outputs	
A	B	C_{in}	C_{out} (Carry)	S (Sum)
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Truth Table

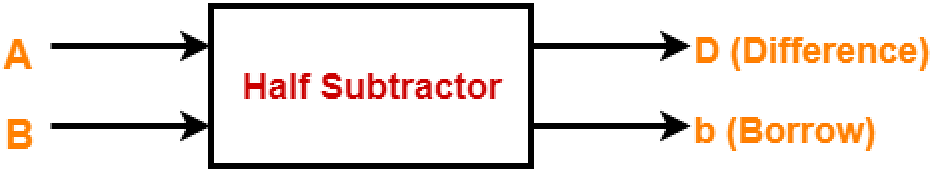
FULL ADDER

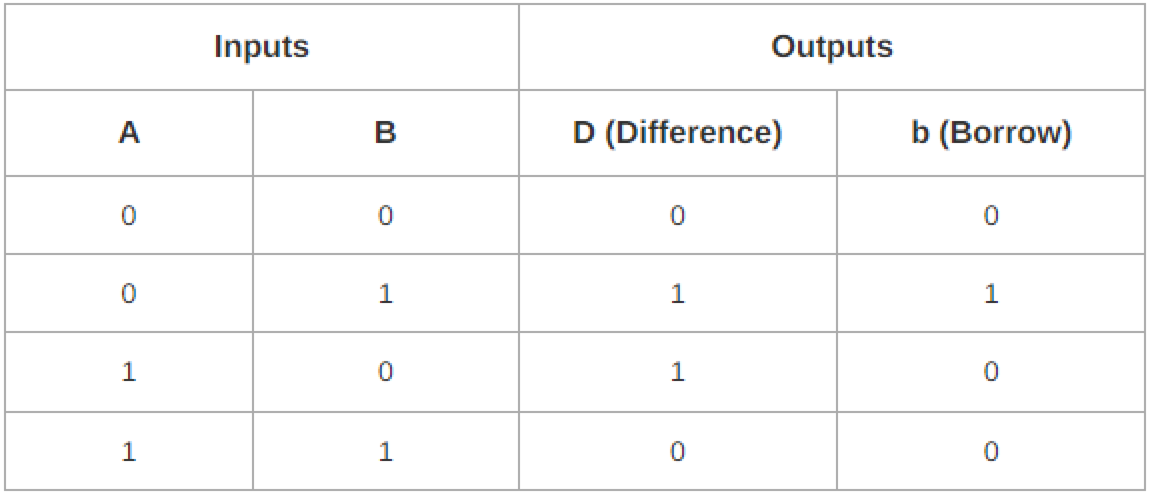


Full Adder Logic Diagram

HALF SUBTRACTOR

- Half Subtractor is a combinational logic circuit.
- It is used for the purpose of subtracting two single bit numbers.
- It contains 2 inputs and 2 outputs (difference and borrow).



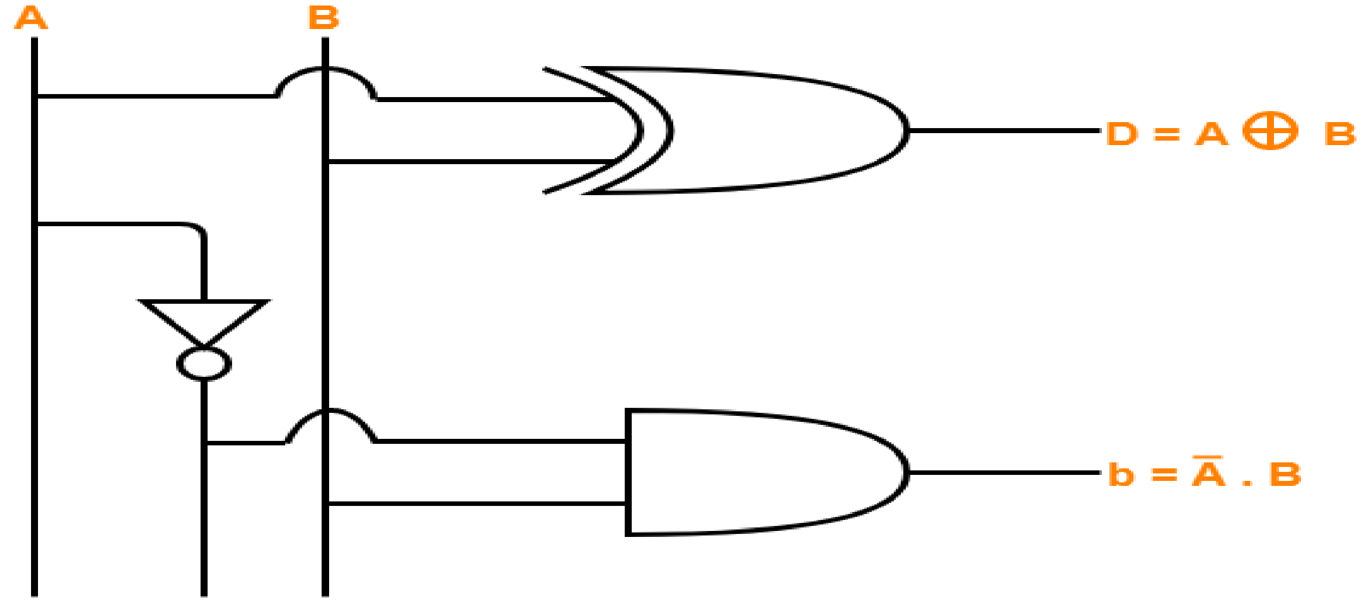


Inputs		Outputs	
A	B	D (Difference)	b (Borrow)
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Truth Table

HALF SUBTRACTOR

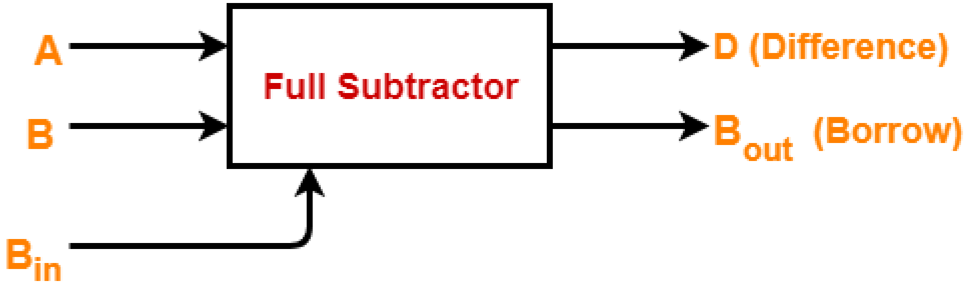
- Half Subtractor is a combinational logic circuit.
- It is used for the purpose of subtracting two single bit numbers.
- It contains 2 inputs and 2 outputs (difference and borrow).



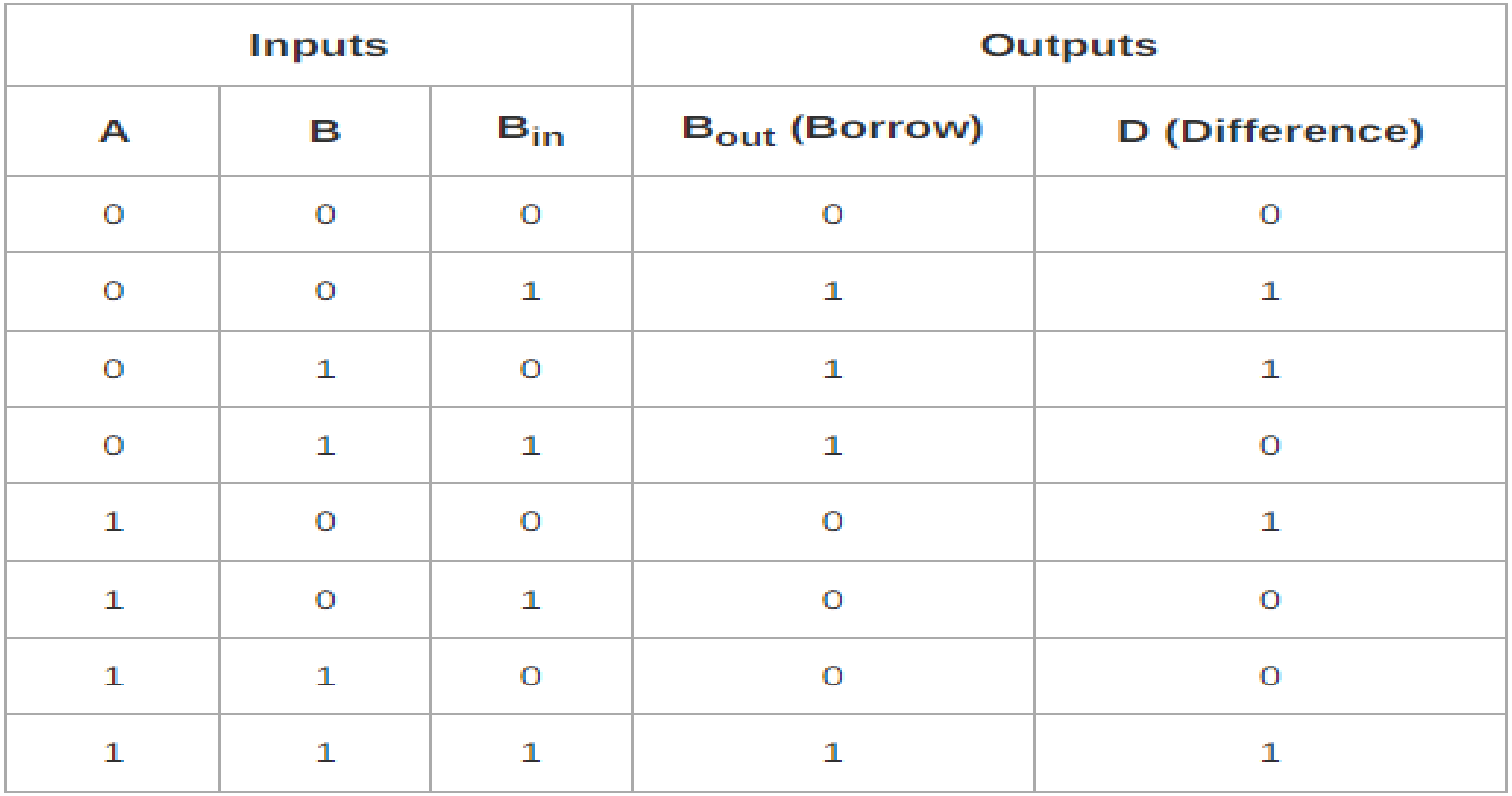
Half Subtractor Logic Diagram

FULL SUBTRACTOR

- Full Subtractor is a combinational logic circuit.
- It is used for the purpose of subtracting two single bit numbers.
- It also takes into consideration borrow of the lower significant stage.
- Thus, full subtractor has the ability to perform the subtraction of three bits.
- Full subtractor contains 3 inputs and 2 outputs (Difference and Borrow) as shown-



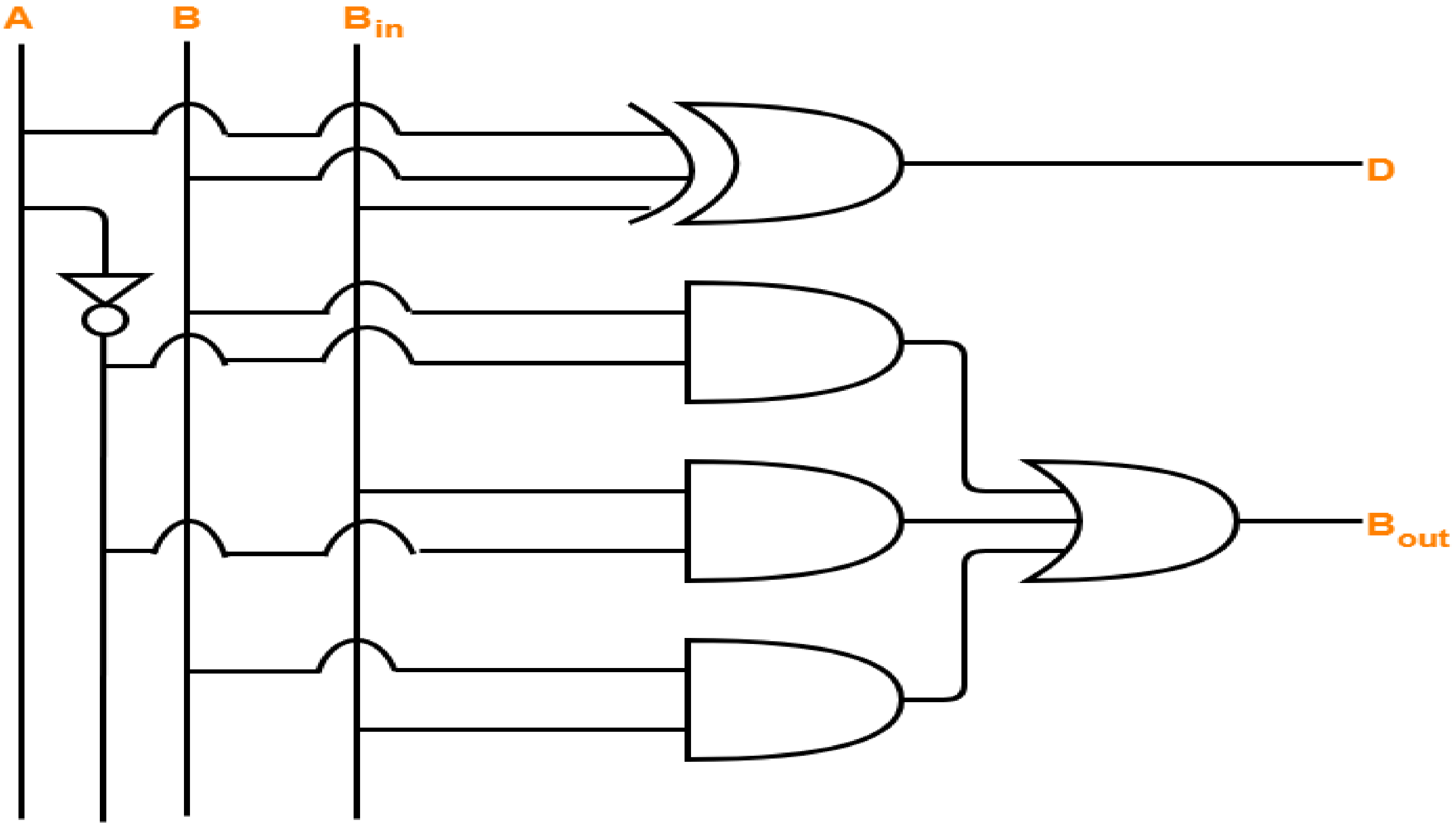
FULL SUBTRACTOR



Inputs			Outputs	
A	B	B_{in}	B_{out} (Borrow)	D (Difference)
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Truth Table

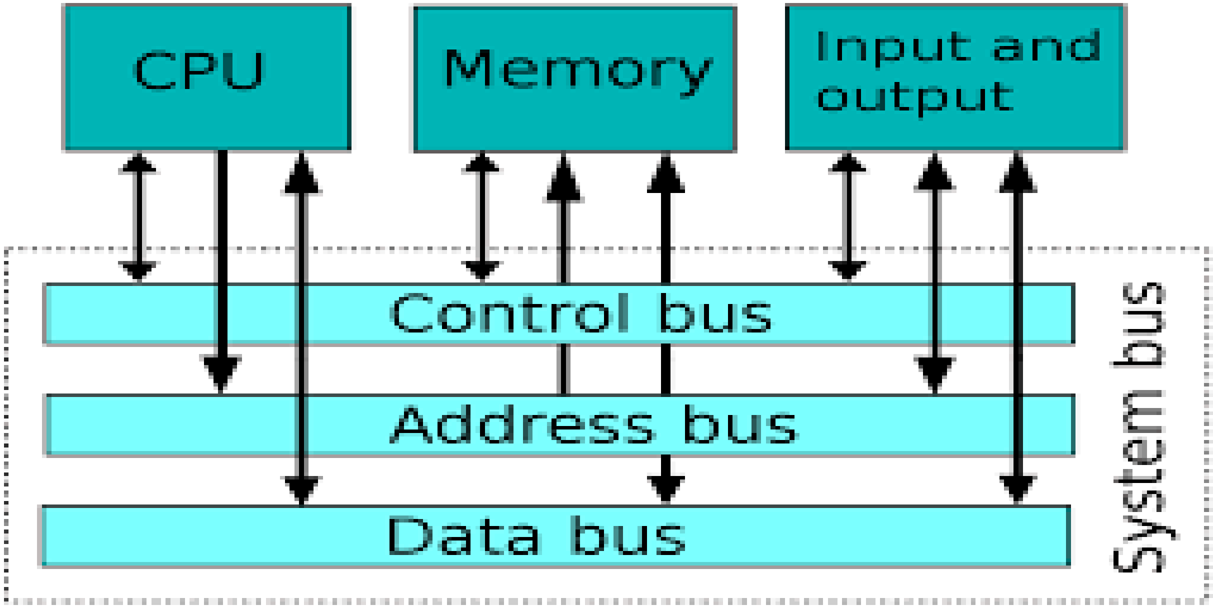
FULL SUBTRACTOR



Full Subtractor Logic Diagram

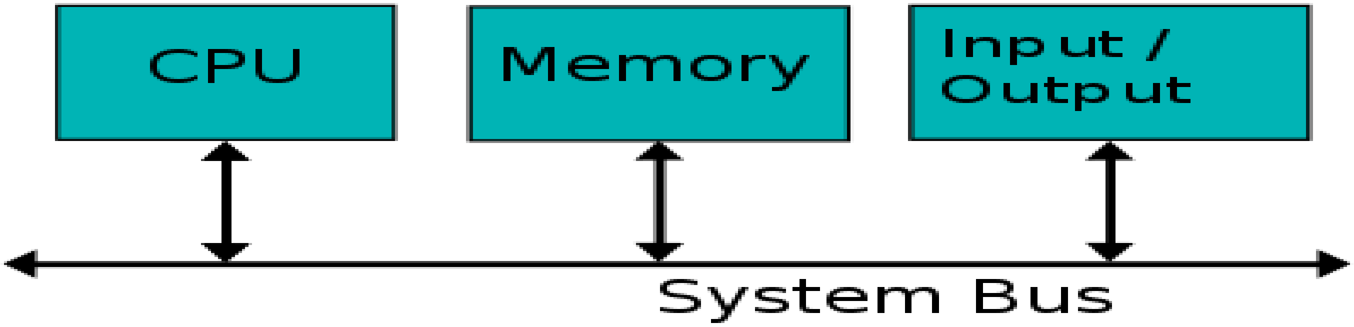
BUS AND MEMORY TRANSFER

- **BUS : A bus** is a **common pathway** through which information flows from one computer component to another.
- It is a subsystem that is used to transfer data and other information between devices.
- Means various devices in computer like(Memory, CPU, I/O and Other) are communicate with each other through buses.
- A shared communication path consisting of one or more connections lines is known as bus and the transfer of data through this bus is known as bus transfer.



BUS AND MEMORY TRANSFER

- A **system bus** is a single computer bus that connects the major components of a computer system, combining the functions of a data bus to carry information, an address bus to determine where it should be sent or read from, and a control bus to determine its operation.



BUS AND MEMORY TRANSFER

Types of Computer BUS:

1. **Data Bus**
2. **Address Bus**
3. **Control Bus**

1. **Data bus**

- It is a **bidirectional** pathway that carries the actual data (information) to and from the main memory.
- Data Lines provide a path for moving data between system modules.
- It is bidirectional which means data lines are used to transfer data in both directions.
- CPU can read data on these lines from memory as well as send data out of these lines to a memory location or to a port.
- The no. of lines in data lines are either 8,16,32 or more depending on architecture.

BUS AND MEMORY TRANSFER

2. **Address bus**

- Address Lines are collectively called as address bus.
- It is a **unidirectional** pathway that allows information to travel in only one direction.
- No. of lines in address are usually 16,20,24, or more depending on type and architecture of bus
- It is an internal channel from CPU to Memory across which the address of data(not data) are transmitted.
- It is used to identify the source or destination of data.
- Here the communication is one way that is, the address is send from CPU to Memory and I/O Port but not Memory and I/O port send address to CPU on that line and hence these lines are unidirectional.

BUS AND MEMORY TRANSFER

3. **Control bus**

- It **carries the control and timing signals** needed to coordinate the activities of the entire computer.
- They are used by CPUs for Communicating with other devices within the computer.
- They are **bidirectional.**
- Typical Control Lines signals are

  Memory Read

  Memory Write

  I/O Read

  I/O Write ,etc

BUS AND MEMORY TRANSFER

1. **Single Bus Structure –**

   All units are connected to the same bus.

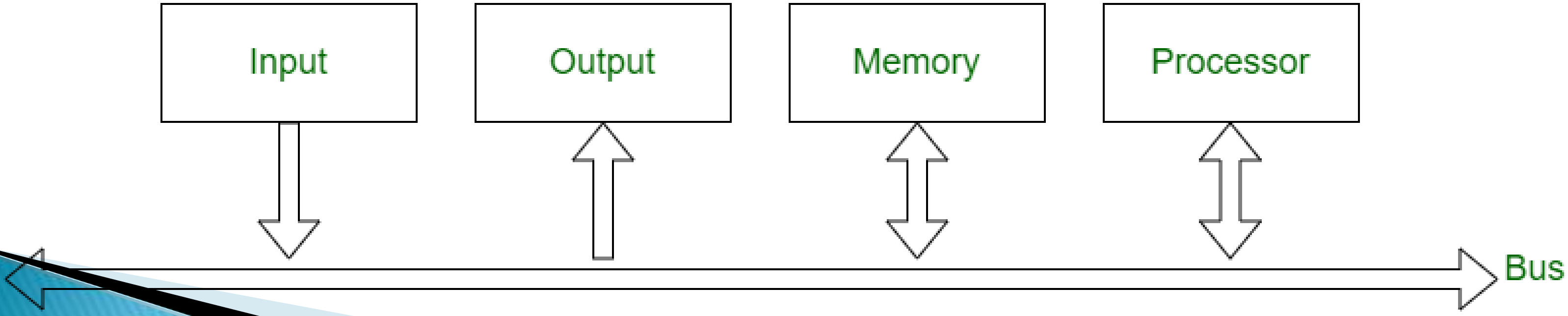
2. **Multiple Bus Structure**

   a) **Traditional Configuration**

   Uses three buses – local bus, system bus and expanded bus.

   a) **High Speed BUS Configuration**

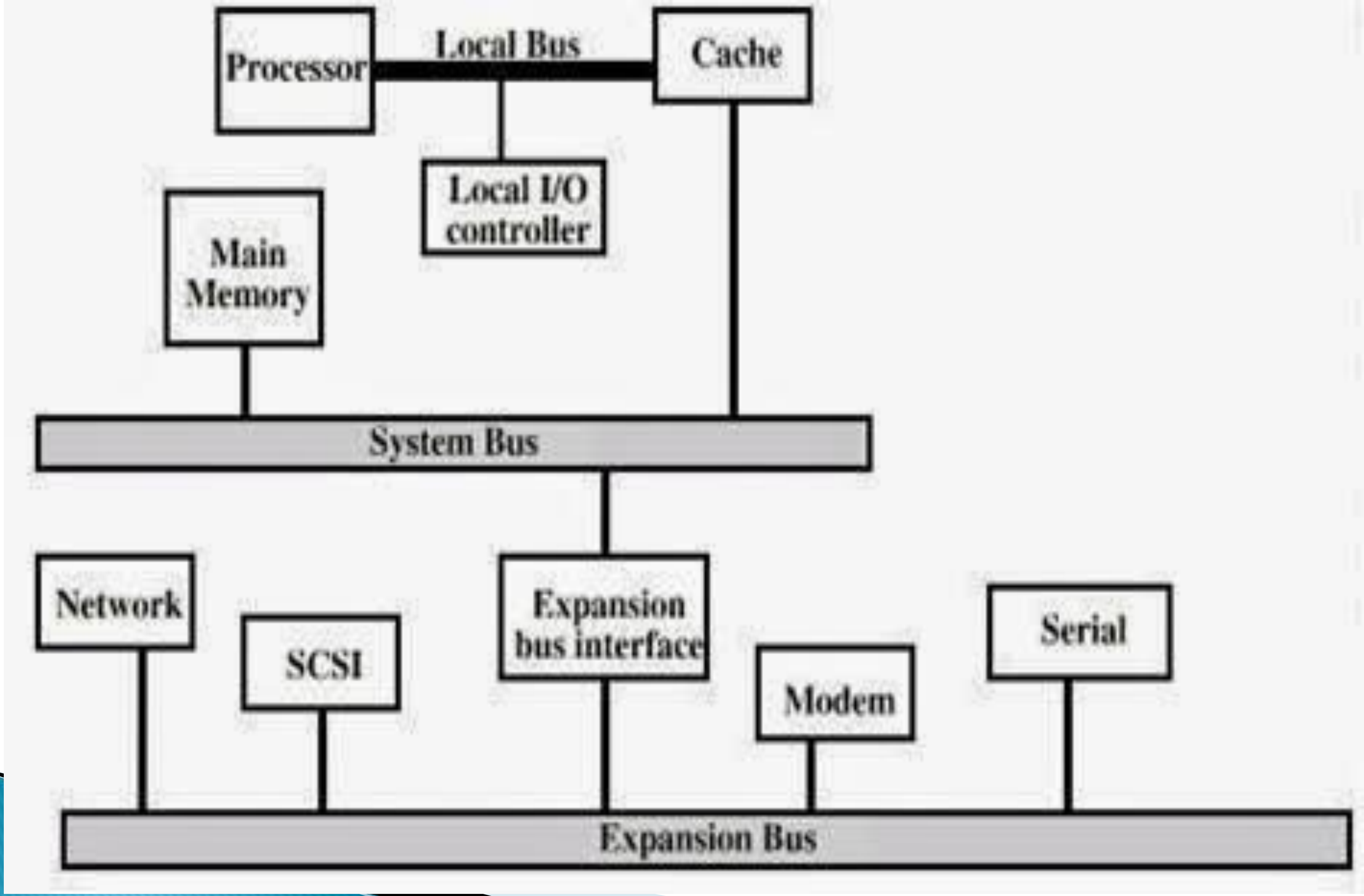
   Uses high speed bus along with three buses – local bus, system bus and expanded bus used in traditional configuration.



Single Bus Structure

BUS AND MEMORY TRANSFER

Traditional Bus Configuration



BUS AND MEMORY TRANSFER

BUS : A shared communication path consisting of one or more connections lines is known as bus and the transfer of data through this bus is known as bus transfer.

Memory Transfer: When a data is read from the memory or is stored in memory is referred to as memory transfer.

Def. : A bidirectional bus used to carry **data between two units is data bus**.

Def. : A unidirectional bus used to carry **memory addresses is called memory bus.**

Def. : The way in which different bus are connected to form common bus, so that CPU, memory and I/O devices can used common bus(Using multiplexer) when required is called **bus organization.**

For a general Bus organization System:-

In general a bus system will multiplex K registers of N bits each.

No of Multiplexers =N

Size of multiplexers =K × 1.

No of selection Lines =m (2^m)=K

Size of decoder will be m× K.

Example : Construct a common bus of 4 registers of 4 bits.

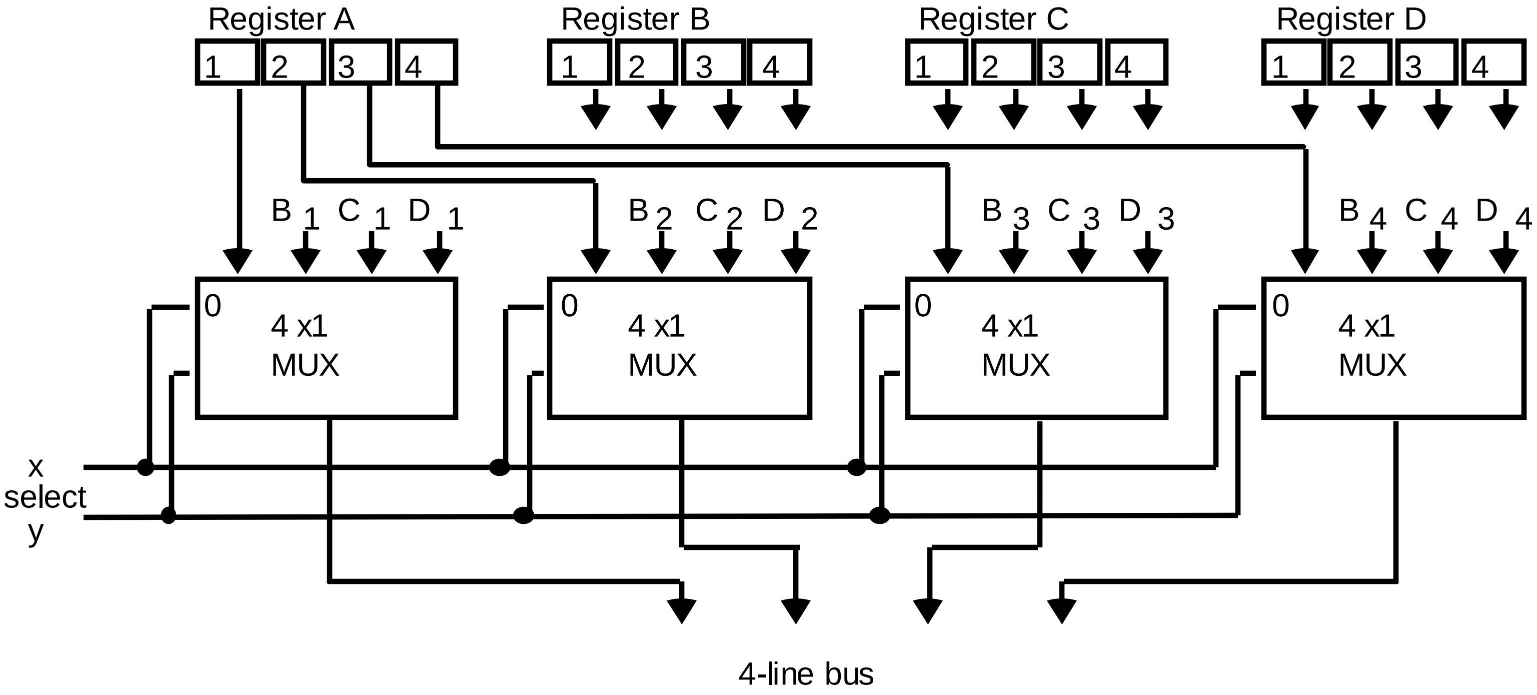
No of Multiplexers =4

Size of multiplexers =4 × 1.

No of selection Lines =2(2^2 =4(size of Multiplexer)

Size of decoder will be 2× 4.

BUS AND MEMORY TRANSFER

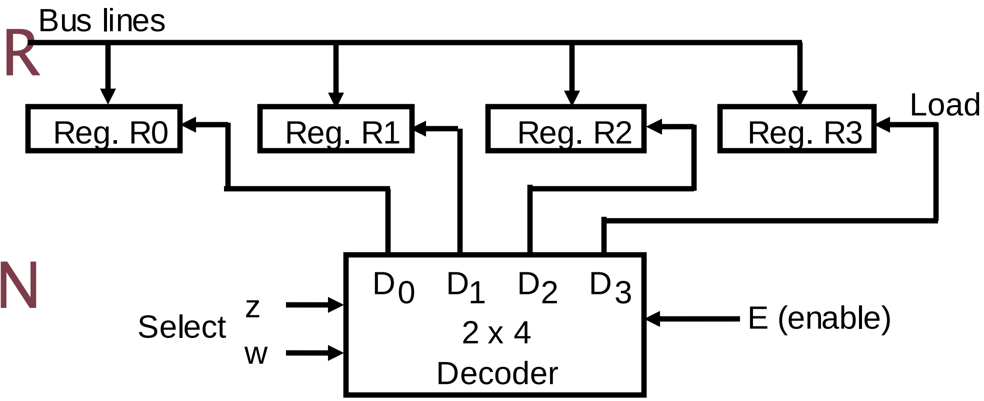


Functional Table for Register Selection

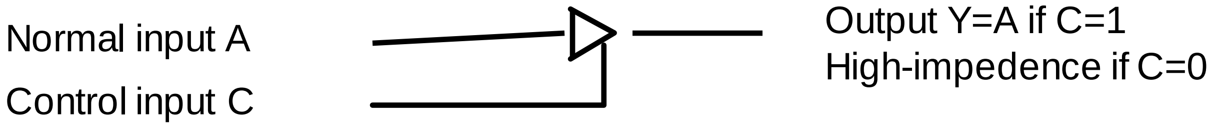
S1	S0	Register Selected
0	0	A
0	1	B
1	0	C
1	1	D

BUS AND MEMORY TRANSFER

TRANSFER FROM BUS TO A DESTINATION REGISTER



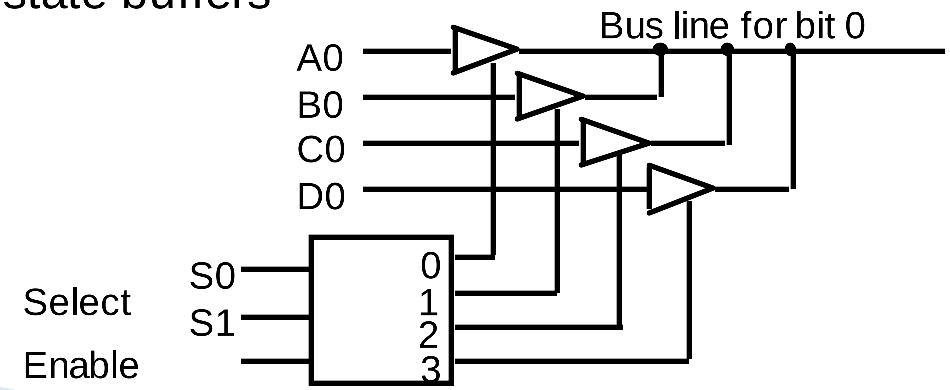
Three-State Bus Buffers


Normal input A

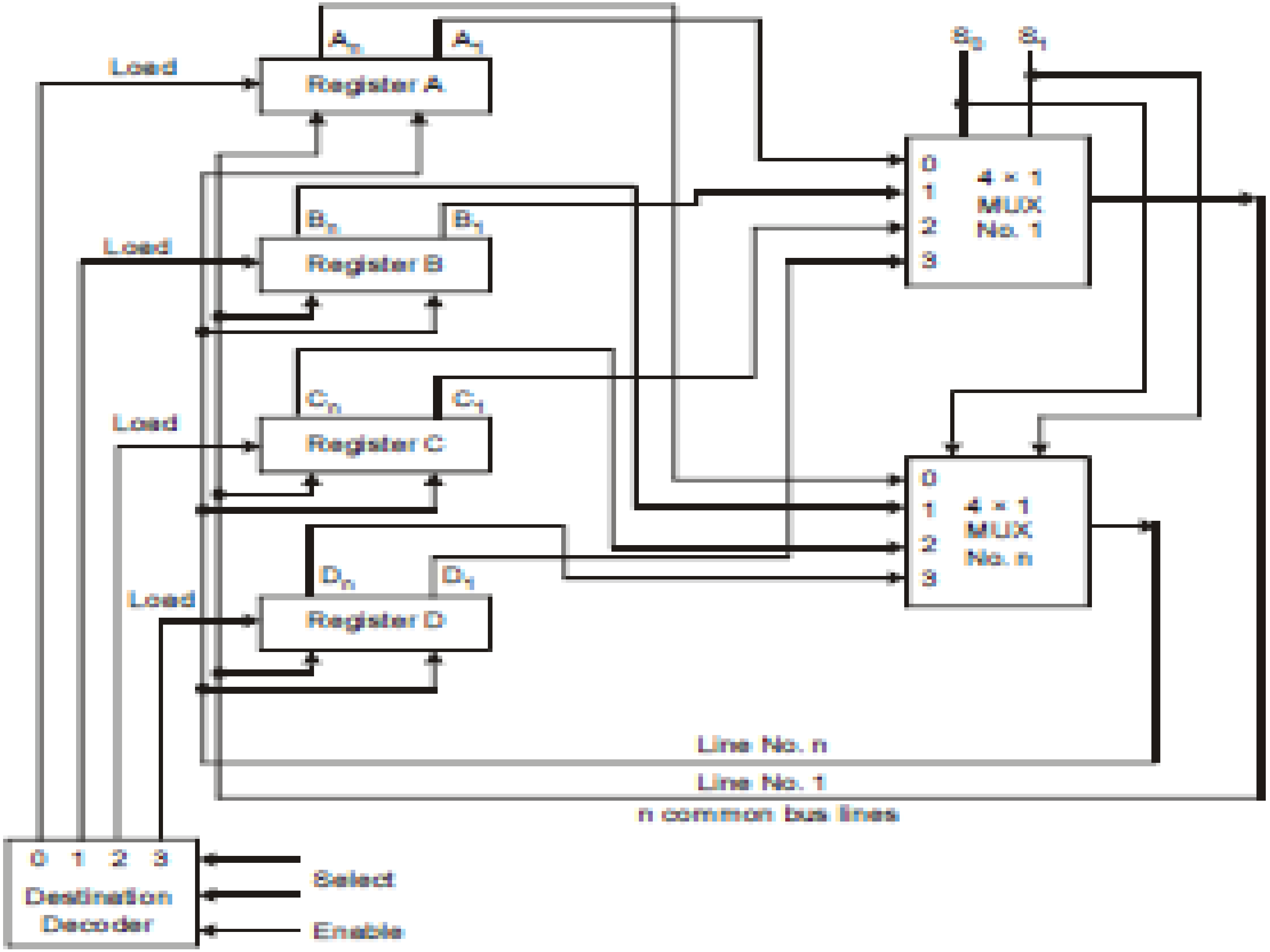
Control input C

Output Y=A if C=1
High-impedence if C=0

Bus line with three-state buffers



BUS AND MEMORY TRANSFER



BUS ARBITRATION

BUS ARBITRATION: refers to the process by which the a device accesses and then leaves the control of the bus and passes it to another bus requesting processor unit. The controller that has access to a bus at an instance is known as a **Bus master**.

OR

The process by which the requesting process may be granted the access to the bus.

There are two approaches to bus arbitration:

Centralized bus arbitration –

A single bus arbiter performs the required arbitration.

(i) Daisy Chaining method

(ii) Polling or Rotating Priority method

(iii) Fixed priority or Independent Request method

Distributed bus arbitration –

All devices participating in the selection of the next bus master.

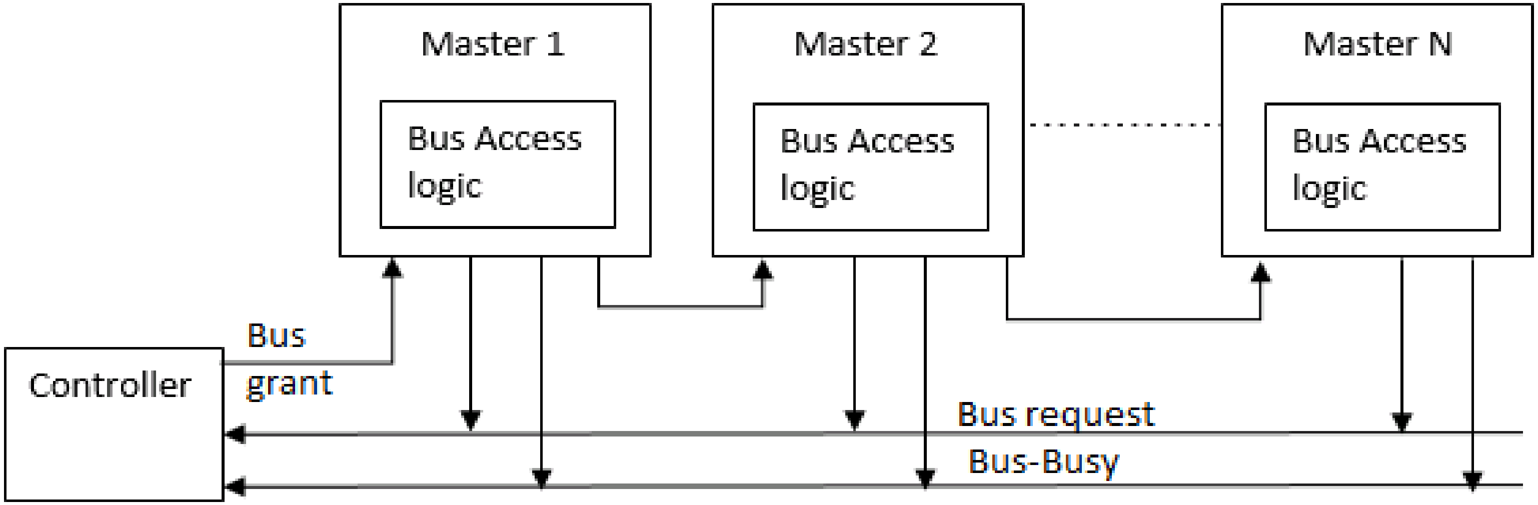
BUS ARBITRATION

Centralized bus arbitration

i) Daisy Chaining method –

It is a simple and cheaper method where all the bus masters use the same line for making bus requests. The bus grant signal serially propagates through each master until it encounters the first one that is requesting access to the bus. This master blocks the propagation of the bus grant signal, therefore any other requesting module will not receive the grant signal and hence cannot access the bus.

During any bus cycle, the bus master may be any device – the processor or any DMA controller unit, connected to the bus.



BUS ARBITRATION

Advantages –

a) Simple design

b) Less no. of control lines.

Disadvantages –

a) Priority depends on the physical location of master.

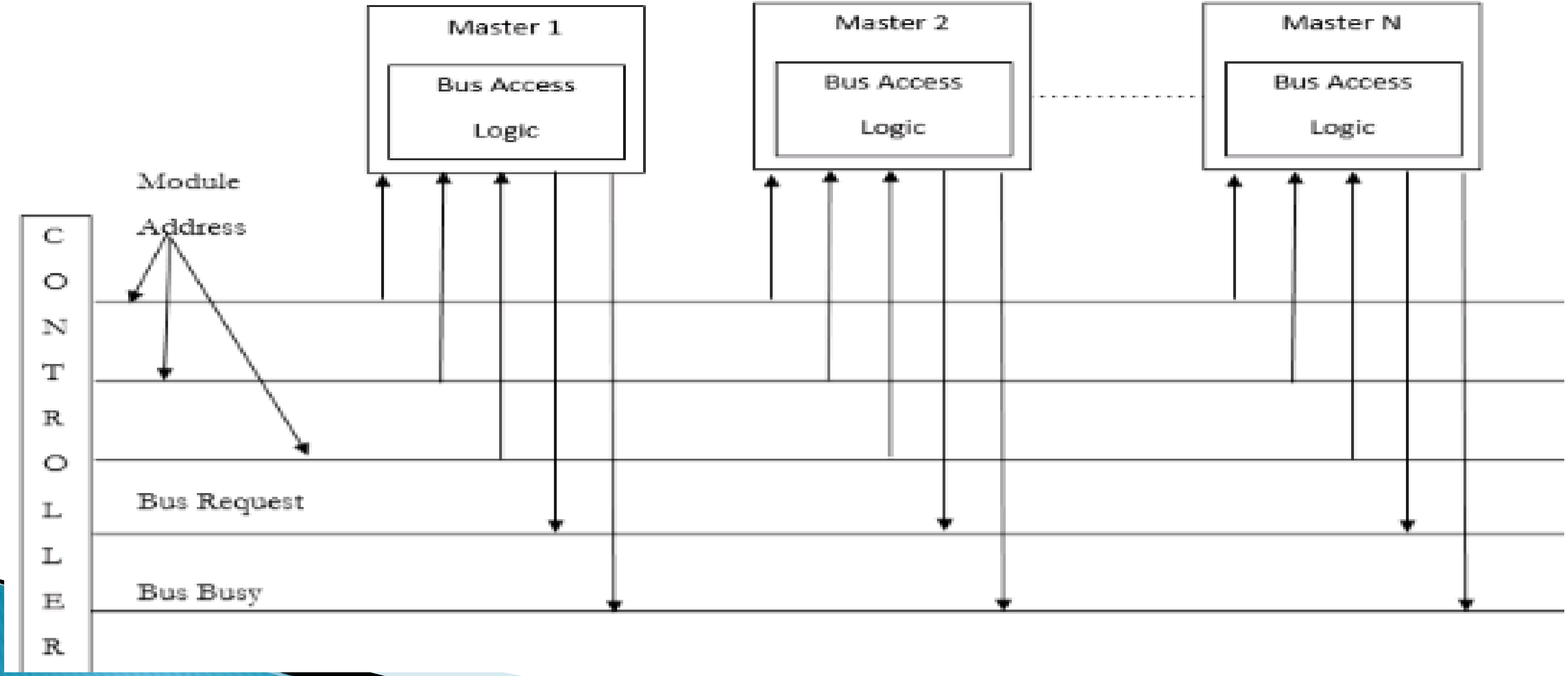
b) Propagation delay due to serially granting of bus.

c) Failure of one of the devices may fail entire system.

BUS ARBITRATION

(ii) Polling or Rotating Priority method –

Here all bus masters use the same line for bus request. Here controller generate binary address for the master. (To connect 8 bus master we need 3 address lines 2323 = 8).In response to a bus request, the controller "polls" the bus masters by sending a sequence of bus master address on address lines.When requesting master recognizes its address, it activates the bus busy lines and takes control of the bus.



BUS ARBITRATION

Advantage:

a) Priority flexible.

b) One module fails, entire system does not fail.

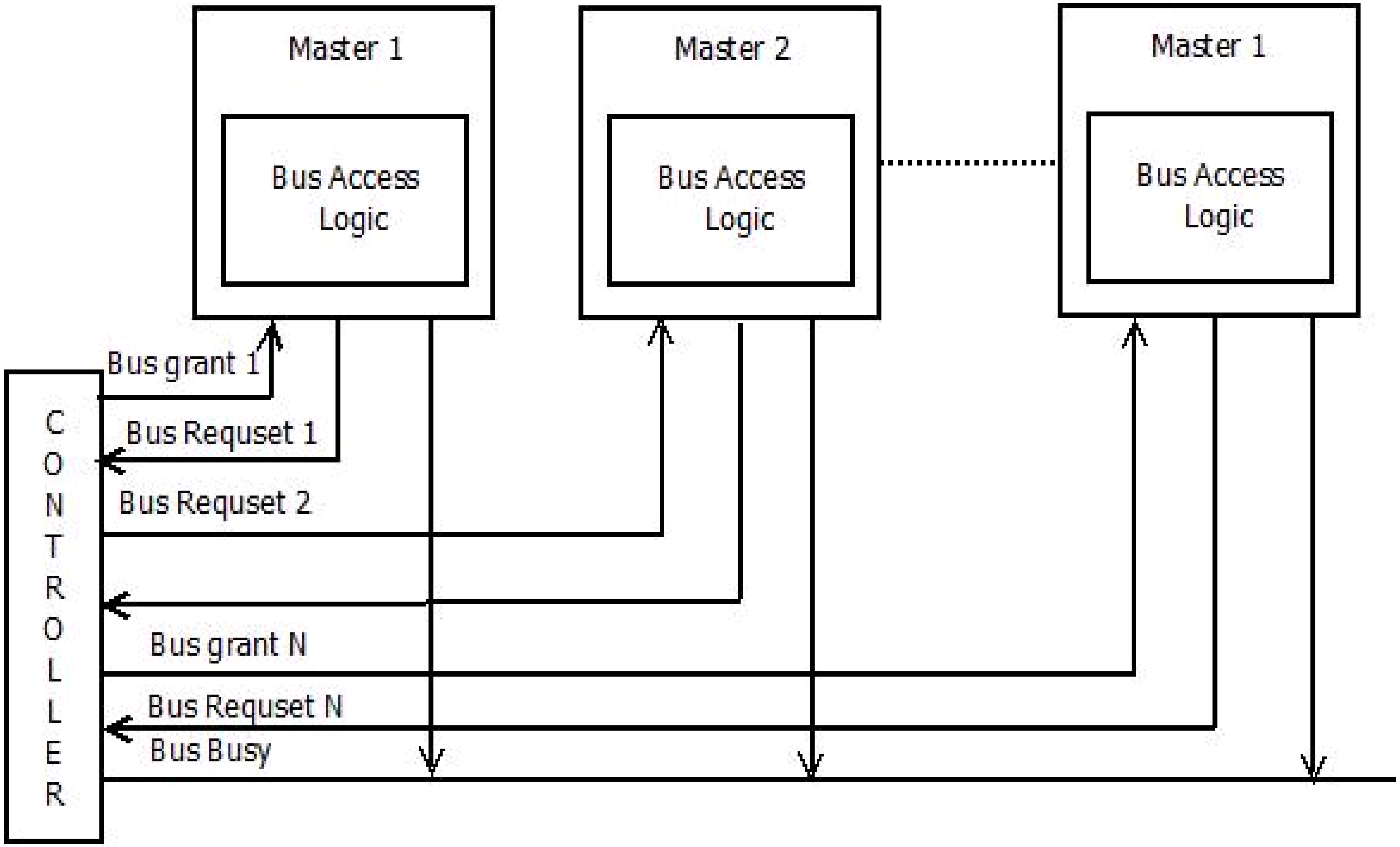
Disadvantage:

a) Adding bus masters is different as increases the number of address lines of the circuit.

BUS ARBITRATION

(iii) Fixed priority or Independent Request method

All bus masters have their individual bus request and bus grant lines. The controller thus knows which master has requested, so bus is granted t that master. Priorities of the masters are predefined so on simultaneous bus requests, the bus is granted based on the priority, provided the bus busy line is not active. The controller consists of encoder and decoder logic for priorities



BUS ARBITRATION

Advantages:

a) Bus arbitration is fast.

b) Speed independent of no. of devices connected.

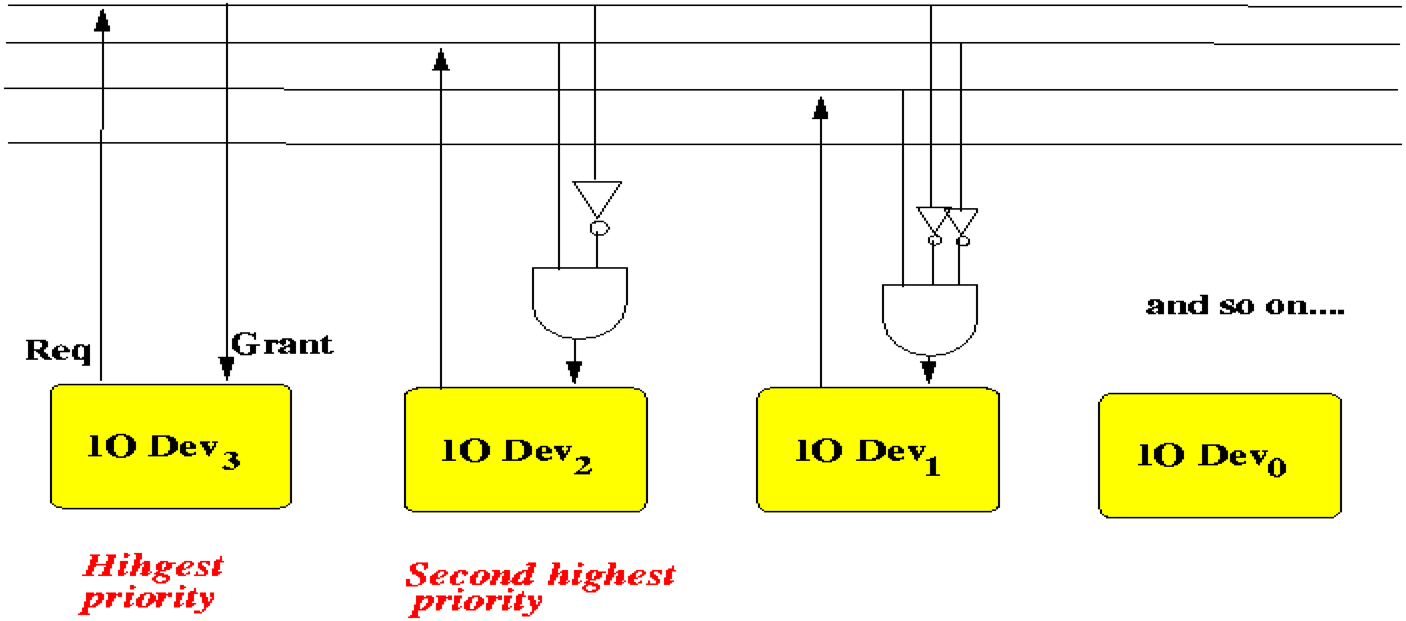
Disadvantages:

a) No. of control lines required is more. Hence connecting a large number of bus masters is difficult.

BUS ARBITRATION

Distributed bus arbitration –

Here, all the devices participate in the selection of the next bus master. Each device on the bus is assigned a 4 bit identification number. When one or more devices request a control of the bus, they assert the start arbitration signal and place their 4-bit identification numbers on arbitration lines through ARB3.Each device compares the code and changes its bit position accordingly. It does so by placing a 0 at the input of their drive. The distributed arbitration is highly reliable because the bus operations are not dependent on devices.



BUS ARBITRATION

- **What is the difference between centralized processing and distributed processing?**
- In centralized computing all the processing is handled by a central system. It is more secure as all the data and processing is handled at single place . But if the central system is down the whole system crashes.
- In distributed computing a problem is divided into many tasks and these task are completed by different systems connected through each other over the network. It is less secure than centralized computing as data is distributed across several computers. But even if one or two computer goes down the entire system does not crash. The other computers continue their work. So you can upgrade your system without completely shutting it down.

Register, Bus and Memory Transfer

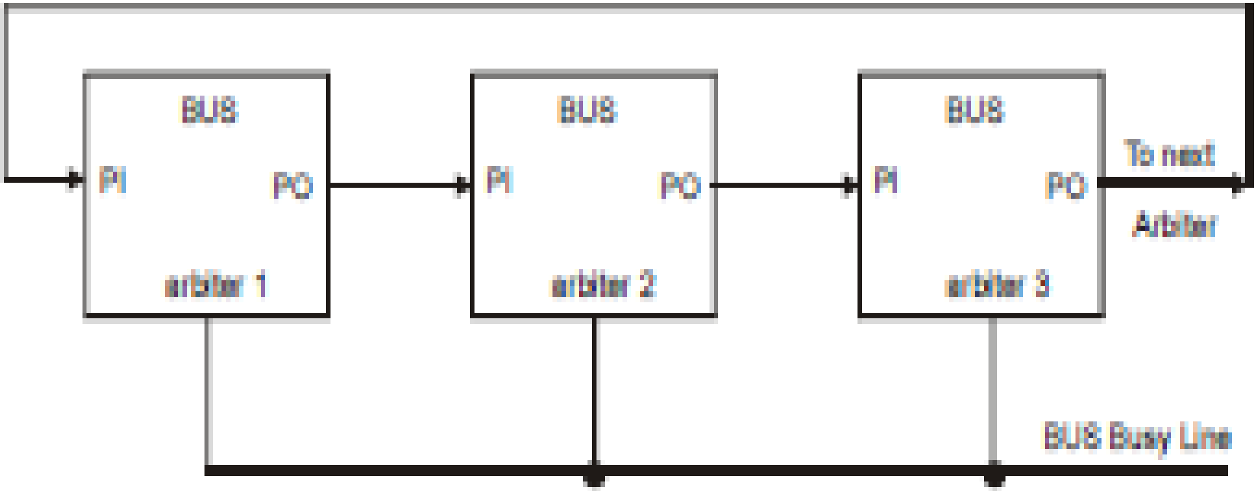
Register

- They are used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU.
- They are used to hold the temporary data.
- There are various types of Registers those are used for various purpose.

Register Symbol	Number of bits	Register Name	Register Function
DR	16	Data register	Holds memory operands
AR	12	Address register	Holds address for memory
AC	16	Accumulator	Processor register
IR	16	Instruction register	Holds instruction code
PC	12	Program counter	Holds address of instruction
TR	16	Temporary register	Holds temporary data
INPR	8	Input register	Holds input character
OUTR	8	Output register	Holds output character

BUS ARBITRATION

Rotating Daisy-chain : Rotating daisy-chain is the dynamic version of the static daisy-chain algorithm. In this algorithm, the priority line is connected to the priority out (PO) of the last processor which is connected to the priority-in of the first processor forming a loop. The processor that has the access to the system bus acts as bus controller. Each arbiter priority is determined by its position along the bus priority line from the arbiter whose processor is using the system bus. Once the arbiter releases the system bus, it has the lowest priority.

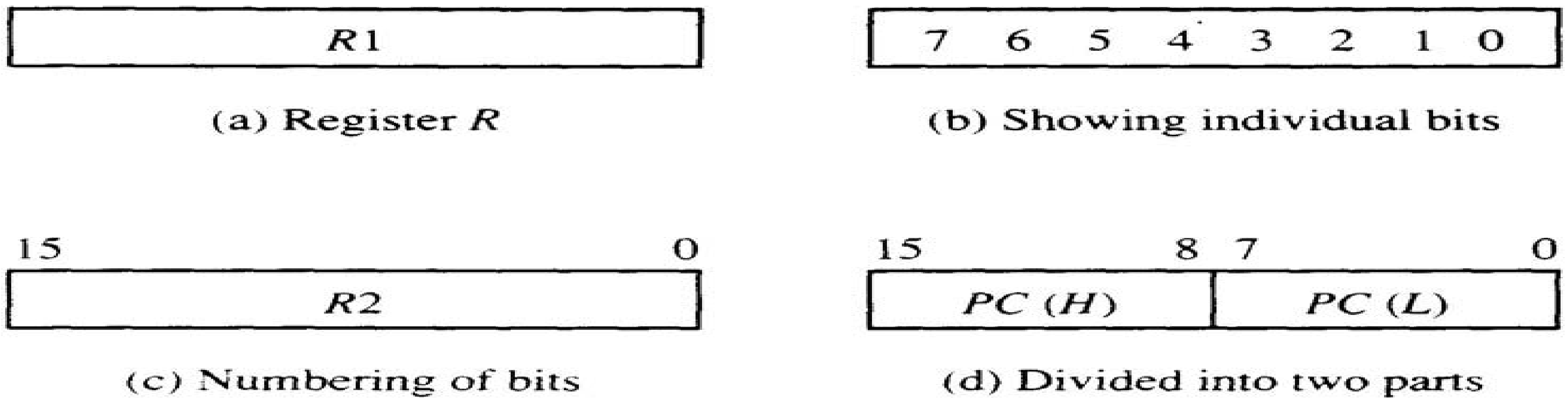


Register, Bus and Memory Transfer

Register

- Computer registers are designated by capital letters (sometimes followed by numerals) to denote the function of the register.
- The register that holds an address for the memory unit is memory address register and is designated by the name **MAR**.
- The program counter register is called **PC**, **IR** is the instruction register and **R1** is a processor register

Block diagram of register.



(a) Register *R*

(b) Showing individual bits

(c) Numbering of bits

(d) Divided into two parts

Register, Bus and Memory Transfer

Register Transfer

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

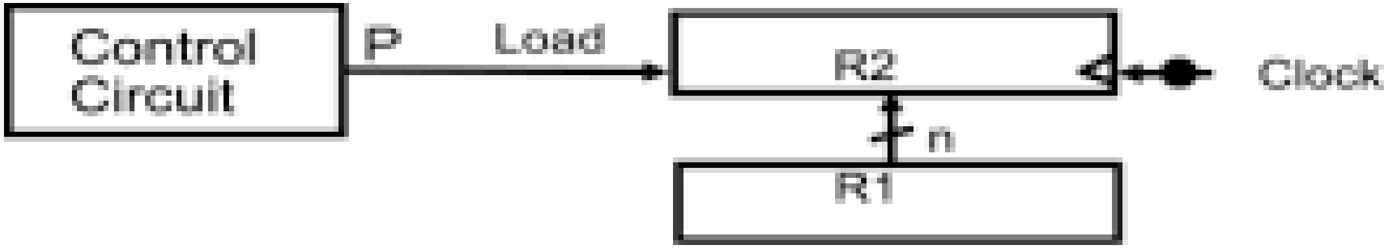
$$R2 \leftarrow R1$$

- It denotes a transfer of the content of register $R1$ into register $R2$. It designates a replacement of the content of $R2$ by the content of $R1$ without changing the content of R1 after transfer.
- If the Register transfer is to occur only under a predetermined control condition, this can be shown by means of an *if-then* statement.

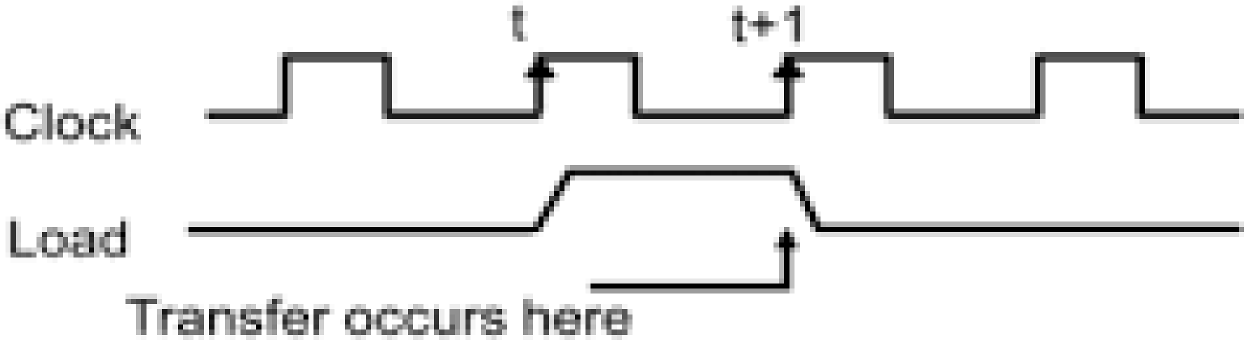
- *If* (P = 1) *then* (R2 ← R1)
- P: R2 ← R1,

where **P is a control function** that can be either 0 or 1

Register, Bus and Memory Transfer

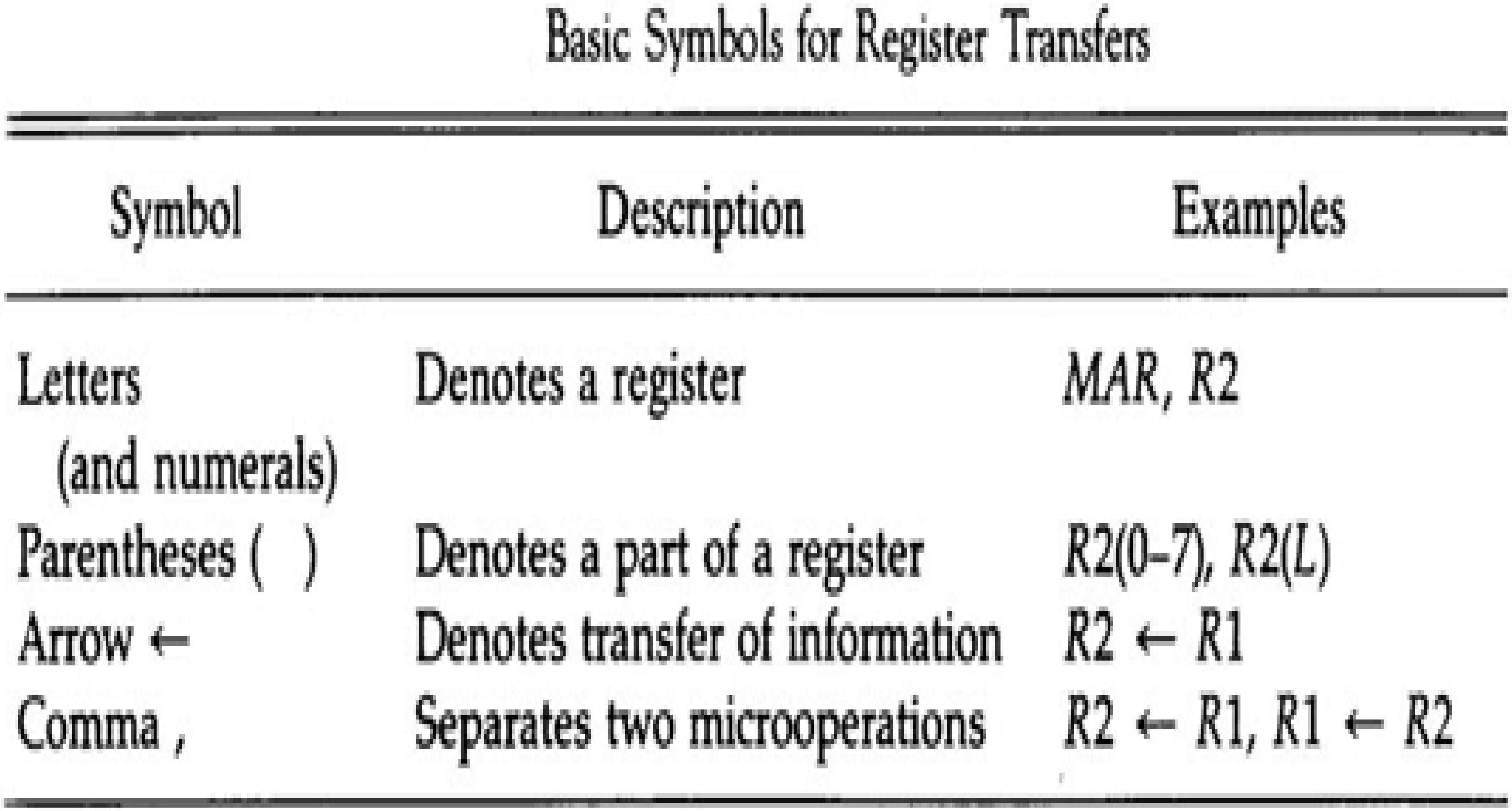


Transfer from R1 to R2 when P = 1



Timing diagram

Register, Bus and Memory Transfer

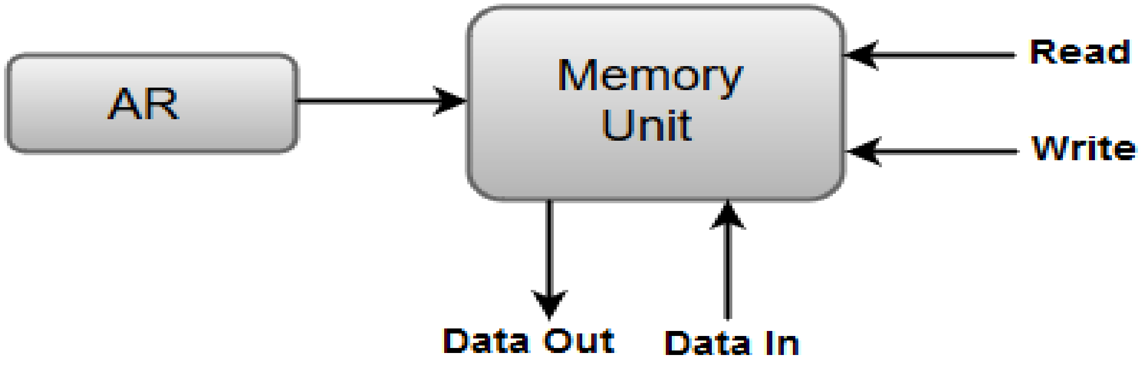


Basic Symbols for Register Transfers

Symbol	Description	Examples
Letters (and numerals)	Denotes a register	MAR, $R2$
Parentheses ( )	Denotes a part of a register	$R2(0–7)$, $R2(L)$
Arrow ←	Denotes transfer of information	$R2 \leftarrow R1$
Comma ,	Separates two microoperations	$R2 \leftarrow R1$, $R1 \leftarrow R2$

Memory Transfer

Memory Transfer Block diagram

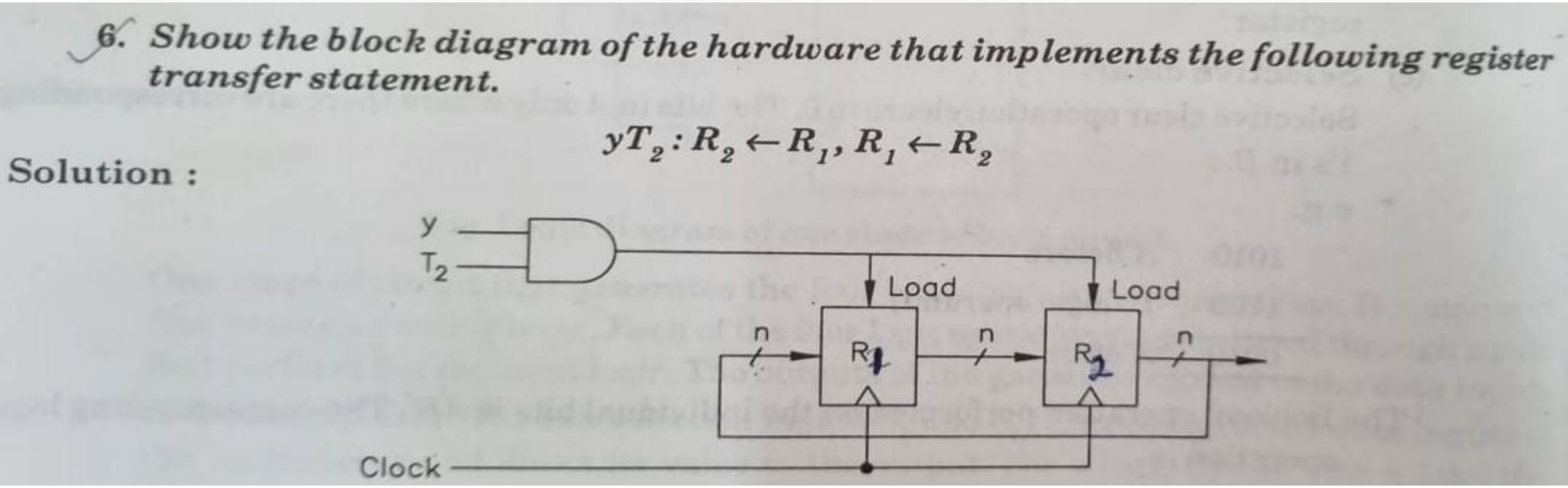


Above Diagram showing connections to memory unit.

Write: M[AR] ← DR

Read: DR ← M[AR]

Memory Transfer

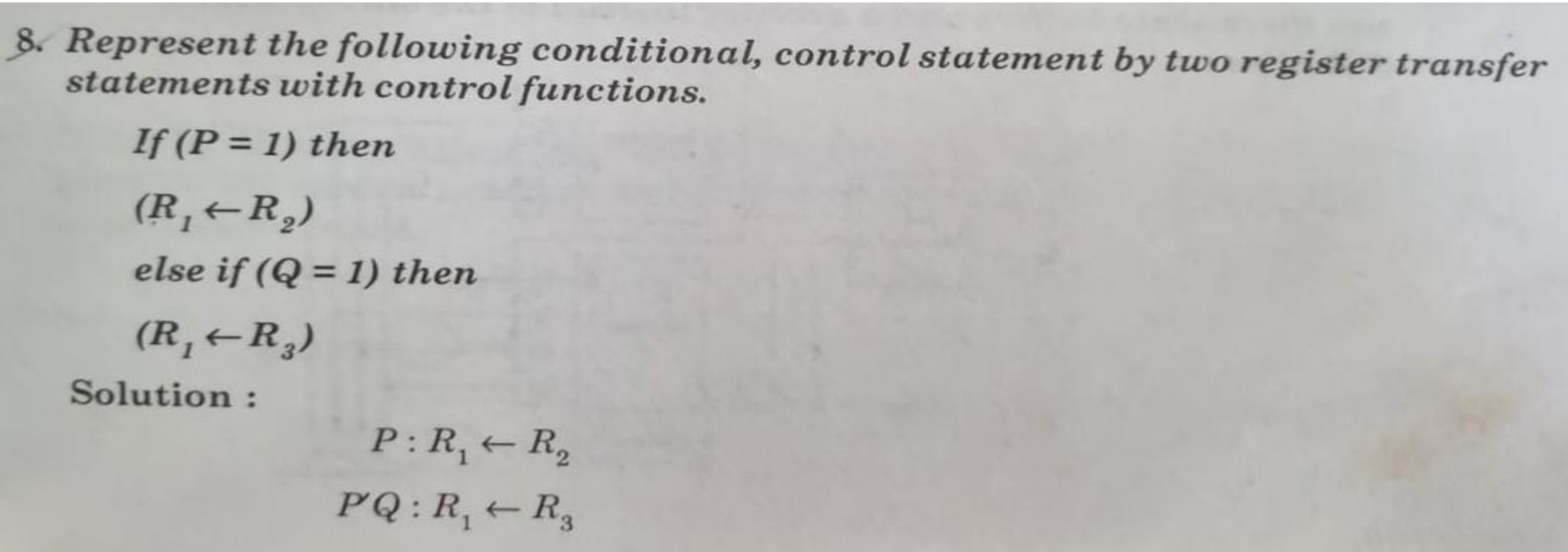


6. *Show the block diagram of the hardware that implements the following register transfer statement.*

$$yT_2 : R_2 \leftarrow R_1, R_1 \leftarrow R_2$$

Solution :

y
T2
Load
Load
n
n
n
R1
R2
Clock



8. *Represent the following conditional, control statement by two register transfer statements with control functions.*

If (P = 1) then

$(R_1 \leftarrow R_2)$

else if (Q = 1) then

$(R_1 \leftarrow R_3)$

Solution :

$$P : R_1 \leftarrow R_2$$

$$P'Q : R_1 \leftarrow R_3$$

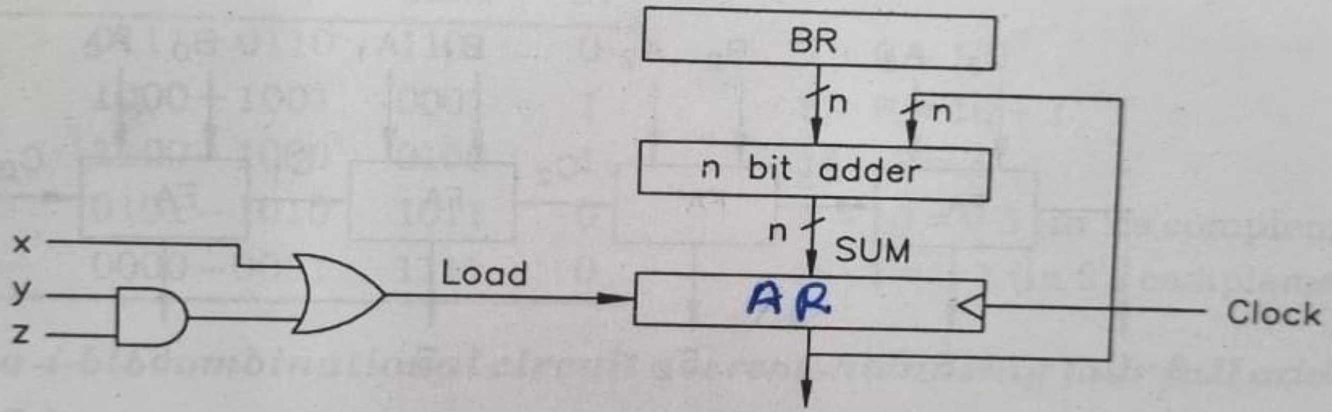
Memory Transfer

13. ***Draw the block diagram for the hardware that implements the following statements :***

$$x + yz : AR \leftarrow AR + BR$$

where AR and BR are two n-bit registers and x, y and z are control variables. Include the logic gates for the control function. (Remember that the symbol + designates an OR operation in a control or Boolean function but that it represents an arithmetic plus in a microoperation.)

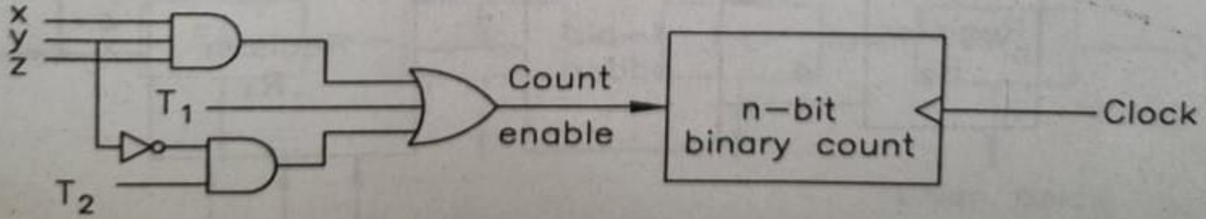
Solution :



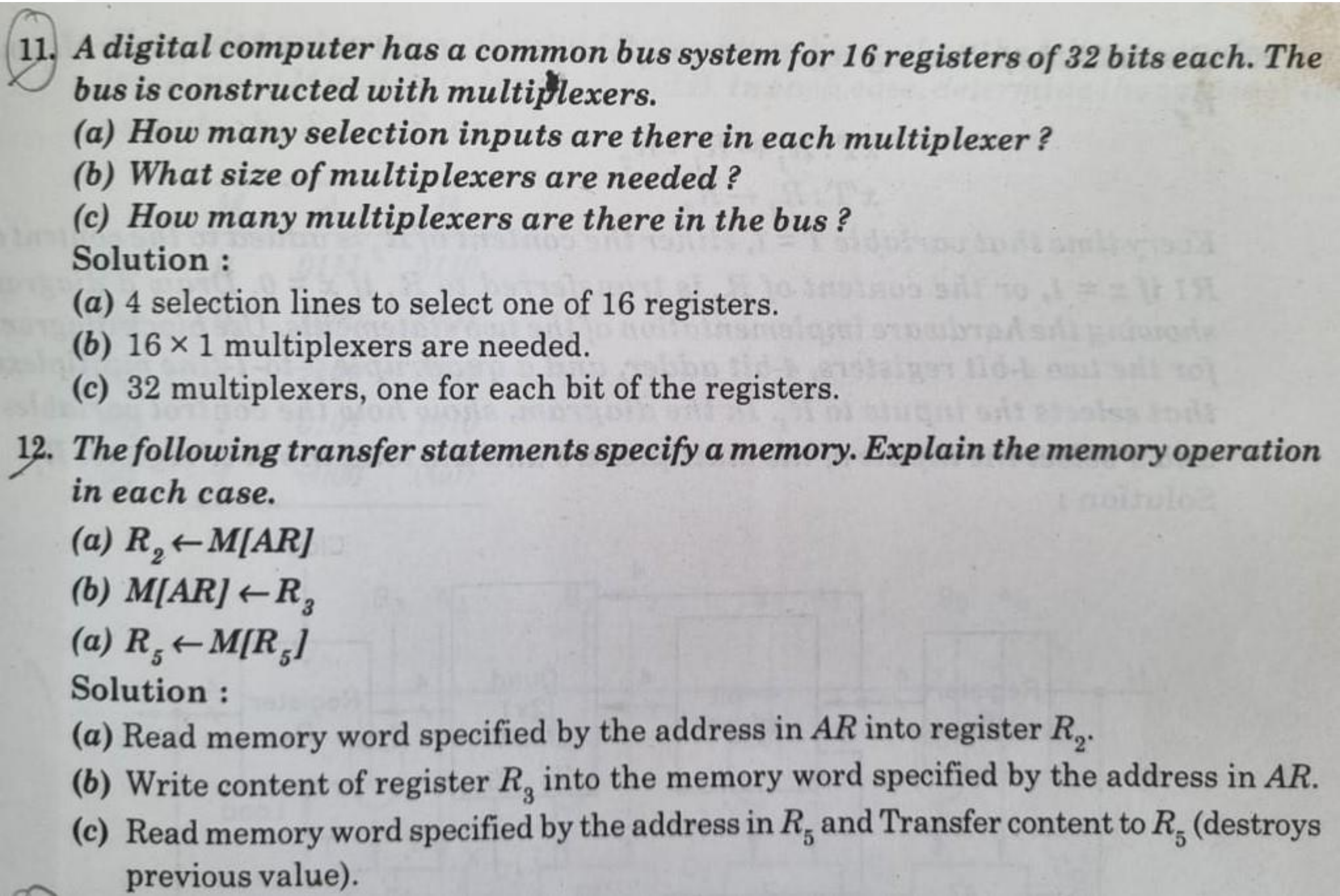
14. ***Show the hardware that implements the following statement. Include the logic gates for the control function and a block diagram for the binary counter with a count enable input.***

$$xyT_0 + T_1 + y'T_2 : AR \leftarrow AR + 1$$

Solution :



Memory Transfer



11. ***A digital computer has a common bus system for 16 registers of 32 bits each. The bus is constructed with multiplexers.***

(a) How many selection inputs are there in each multiplexer ?

(b) What size of multiplexers are needed ?

(c) How many multiplexers are there in the bus ?

Solution :

(*a*) 4 selection lines to select one of 16 registers.

(*b*) 16 × 1 multiplexers are needed.

(c) 32 multiplexers, one for each bit of the registers.

12. ***The following transfer statements specify a memory. Explain the memory operation in each case.***

(a) $R_2 \leftarrow M[AR]$

(b) $M[AR] \leftarrow R_3$

(a) $R_5 \leftarrow M[R_5]$

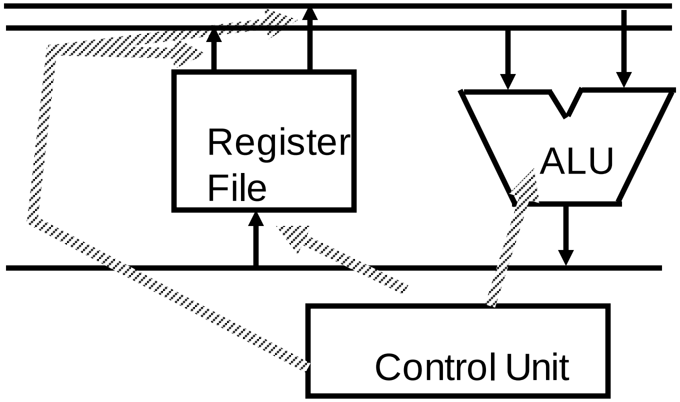
Solution :

(*a*) Read memory word specified by the address in AR into register R_2.

(*b*) Write content of register R_3 into the memory word specified by the address in AR.

(c) Read memory word specified by the address in R_5 and Transfer content to R_5 (destroys previous value).

GENERAL REGISTERS ORGANIZATION



MAJOR COMPONENTS OF CPU

- Storage Components
  - Registers
  - Flags
- Execution (Processing) Components
  - Arithmetic Logic Unit(ALU)
    - Arithmetic calculations, Logical computations, Shifts/Rotates
- Transfer Components
  - Bus
- Control Components
  - Control Unit

In Basic Computer, there is only one general purpose register, the Accumulator (AC)

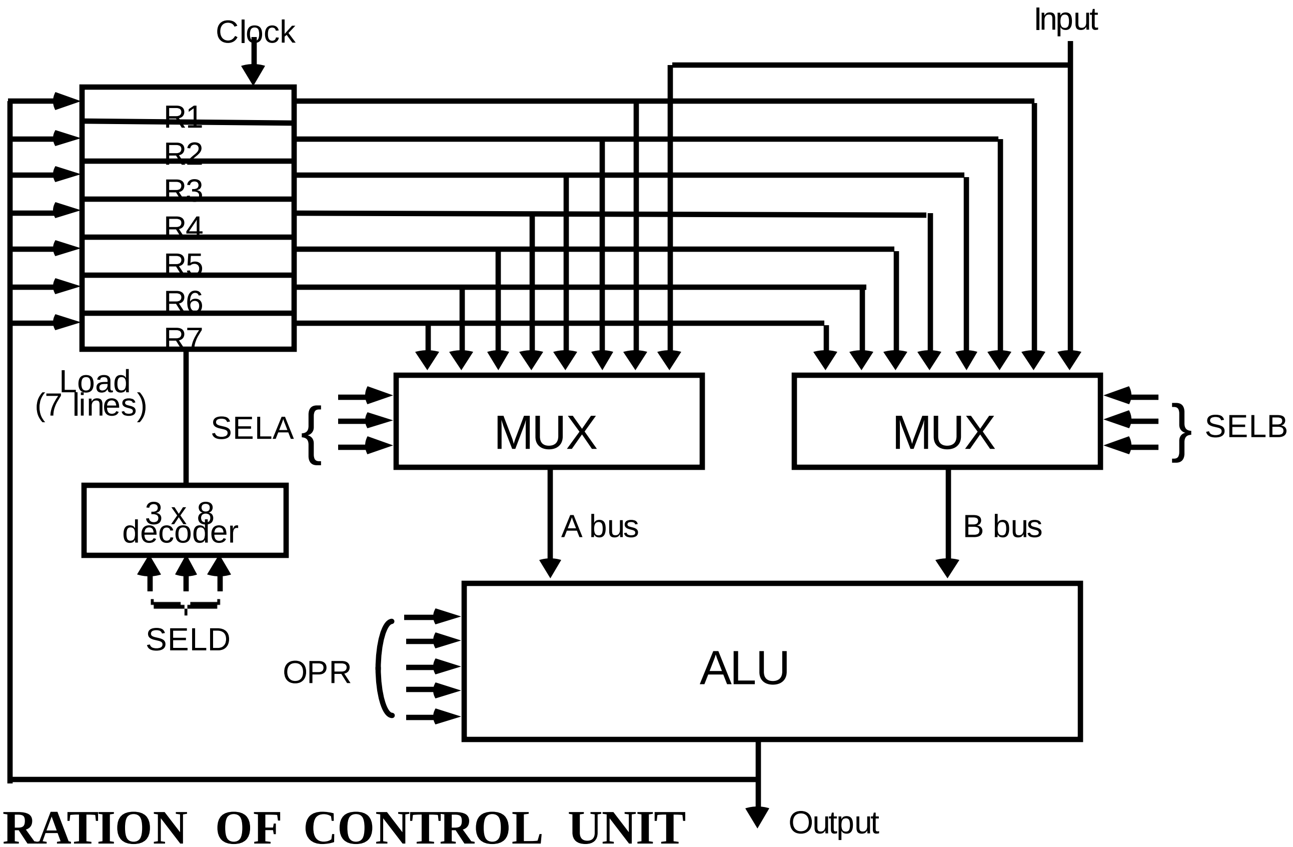
In modern CPUs, there are many general purpose registers

It is advantageous to have many registers

Transfer between registers within the processor are relatively fast

Going “off the processor” to access memory is much slower

GENERAL REGISTERS ORGANIZATION



OPERATION OF CONTROL UNIT

The control unit

Directs the information flow through ALU by

- Selecting various *Components* in the system
- Selecting the *Function* of ALU

GENERAL REGISTERS ORGANIZATION

Control Word : The group of binary bits assigned to perform a specified operation is known as control word

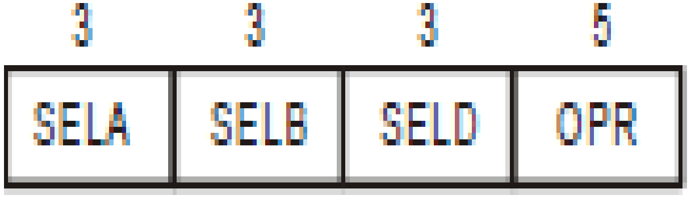
To perform ANY operation, the control must provide

(*i*) SELA: place the contents of Any Register *R* into bus *A*.

(*ii*) SELB: place the contents of Any Register *R* into bus *B*.

(*iii*) ALU operation selector OPR : provide the arithmetic operation

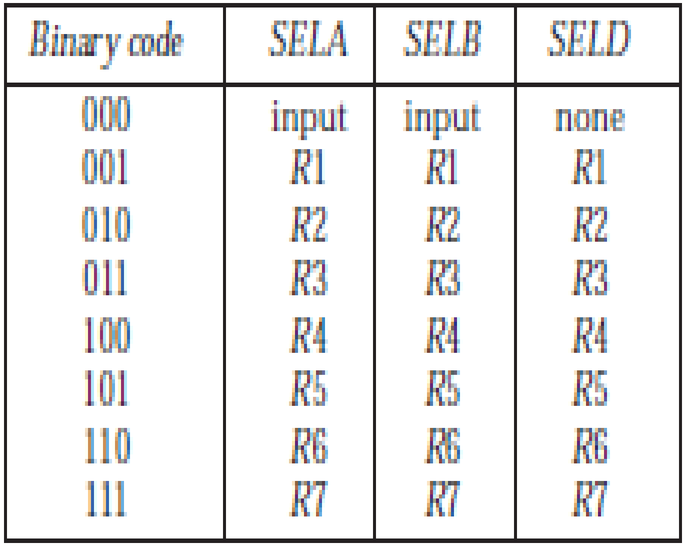
(*iv*) SELD: transfer the contents of the output bus into any destination Register *R*.



3	3	3	5
SELA	SELB	SELD	OPR

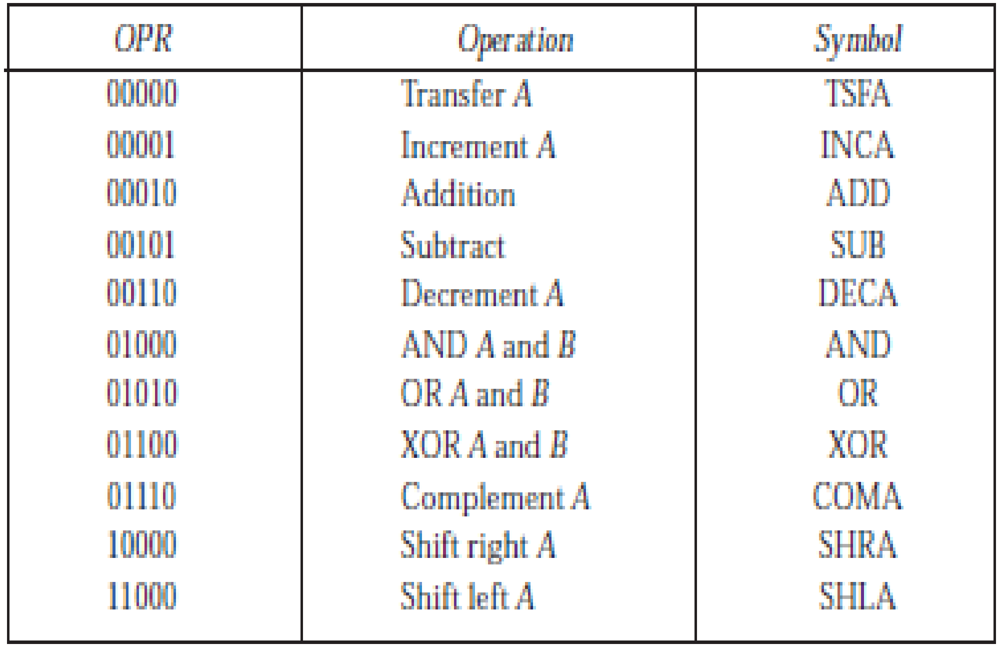
GENERAL REGISTERS ORGANIZATION

Encoding of Register Selection Fields



Binary code	SELA	SELB	SELD
000	input	input	none
001	*R1*	*R1*	*R1*
010	*R2*	*R2*	*R2*
011	*R3*	*R3*	*R3*
100	*R4*	*R4*	*R4*
101	*R5*	*R5*	*R5*
110	*R6*	*R6*	*R6*
111	*R7*	*R7*	*R7*

Encoding of ALU



OPR	Operation	Symbol
00000	Transfer *A*	TSFA
00001	Increment *A*	INCA
00010	Addition	ADD
00101	Subtract	SUB
00110	Decrement *A*	DECA
01000	AND *A* and *B*	AND
01010	OR *A* and *B*	OR
01100	XOR *A* and *B*	XOR
01110	Complement *A*	COMA
10000	Shift right *A*	SHRA
11000	Shift left *A*	SHLA

Let the operation be

R5←*R1* **.** *R4*

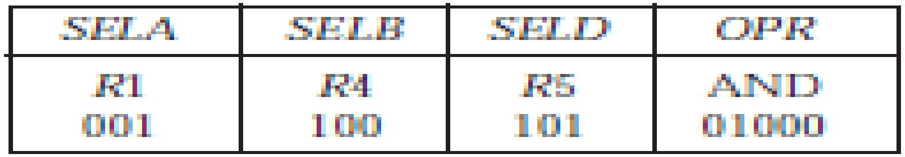
To perform this operation, the control must provide

(*i*) SELA: place the contents of *R1* into bus *A*.

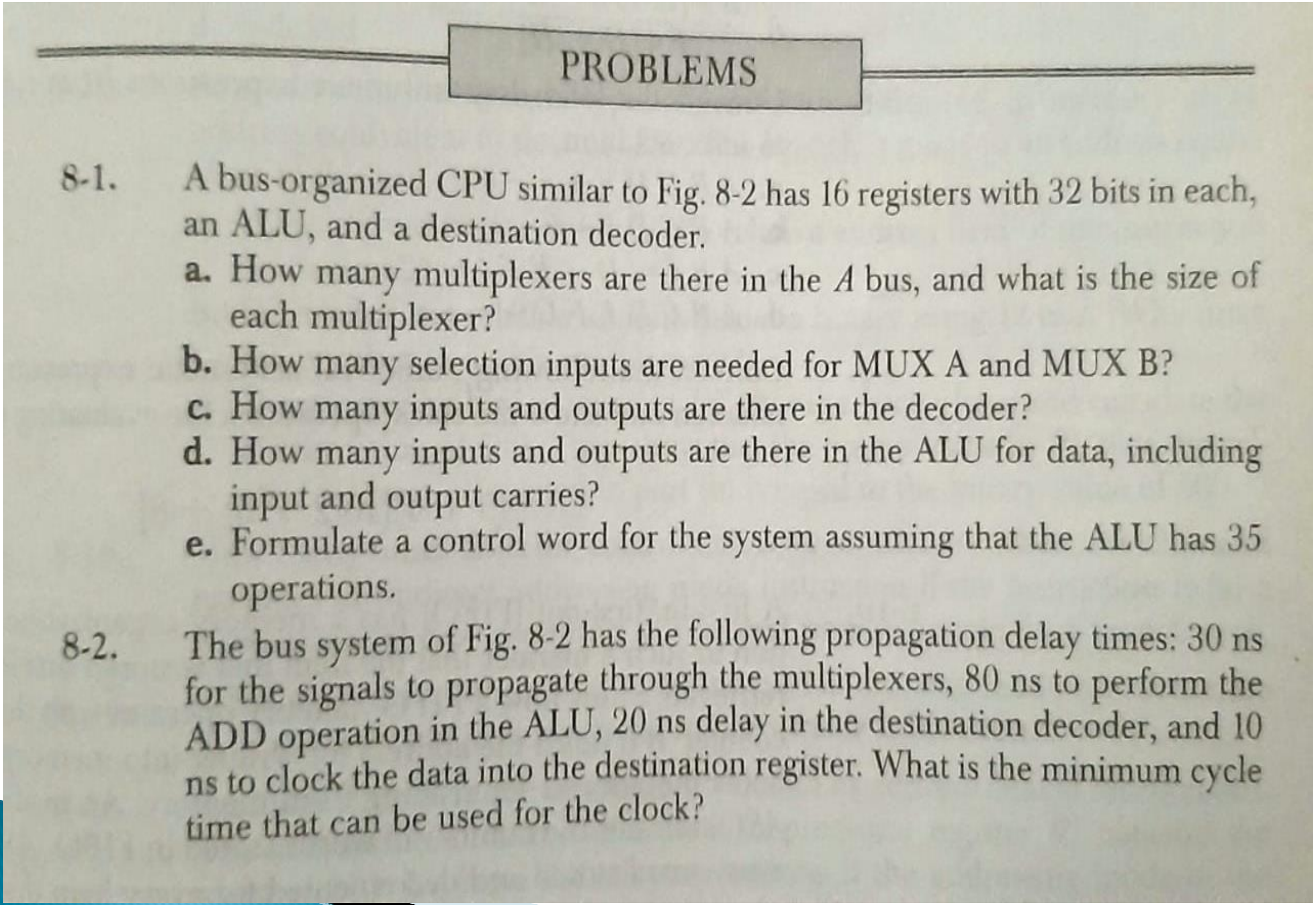
(*ii*) SELB: place the contents of *R4* into bus *B*.

(*iii*) ALU operation selector OPR : provide the arithmetic multiplication *A* AND *B*.

(*iv*) SELD: transfer the contents of the output bus into *R5*.



SELA	SELB	SELD	OPR
R1 001	*R4* 100	*R5* 101	AND 01000



PROBLEMS

8-1. A bus-organized CPU similar to Fig. 8-2 has 16 registers with 32 bits in each, an ALU, and a destination decoder.

a. How many multiplexers are there in the *A* bus, and what is the size of each multiplexer?

b. How many selection inputs are needed for MUX A and MUX B?

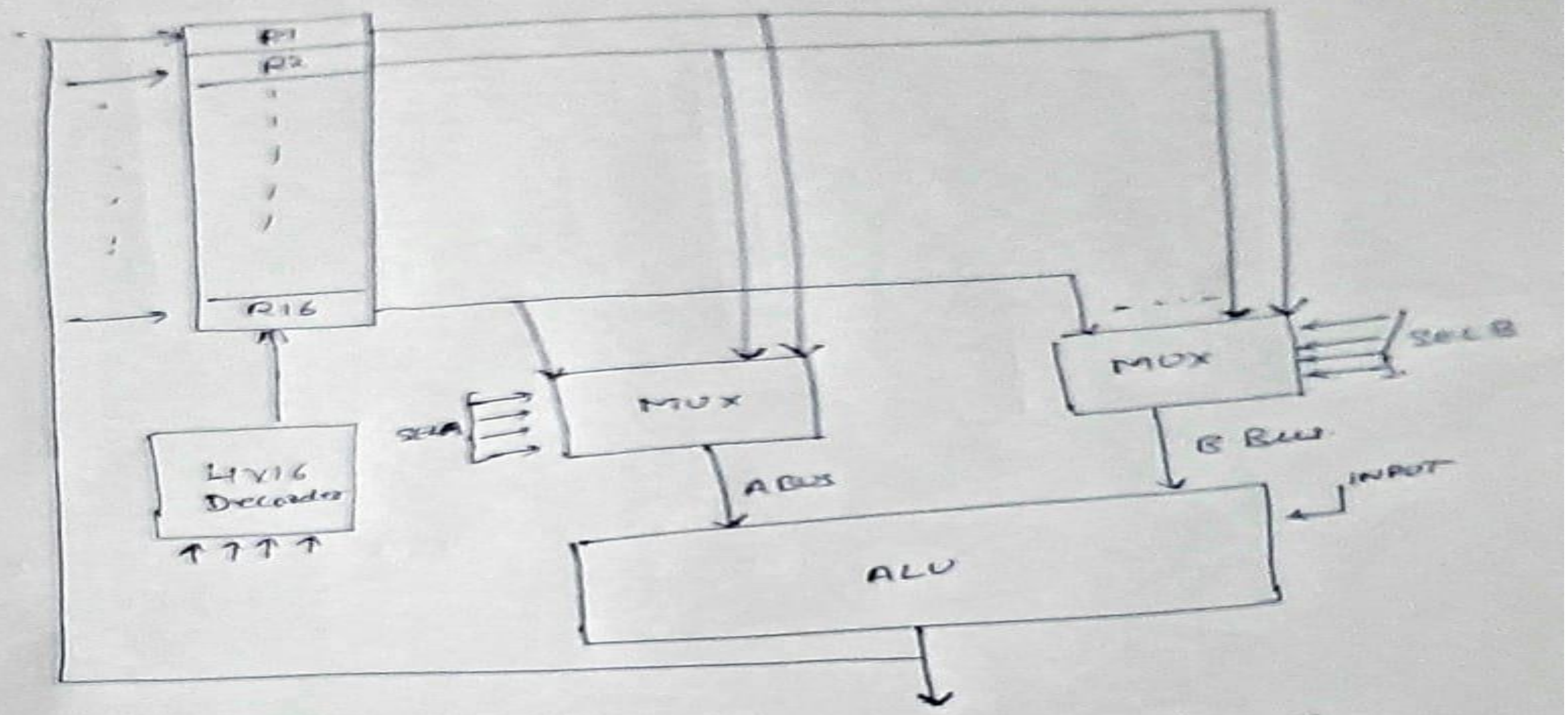
c. How many inputs and outputs are there in the decoder?

d. How many inputs and outputs are there in the ALU for data, including input and output carries?

e. Formulate a control word for the system assuming that the ALU has 35 operations.

8-2. The bus system of Fig. 8-2 has the following propagation delay times: 30 ns for the signals to propagate through the multiplexers, 80 ns to perform the ADD operation in the ALU, 20 ns delay in the destination decoder, and 10 ns to clock the data into the destination register. What is the minimum cycle time that can be used for the clock?

Sol 8.1



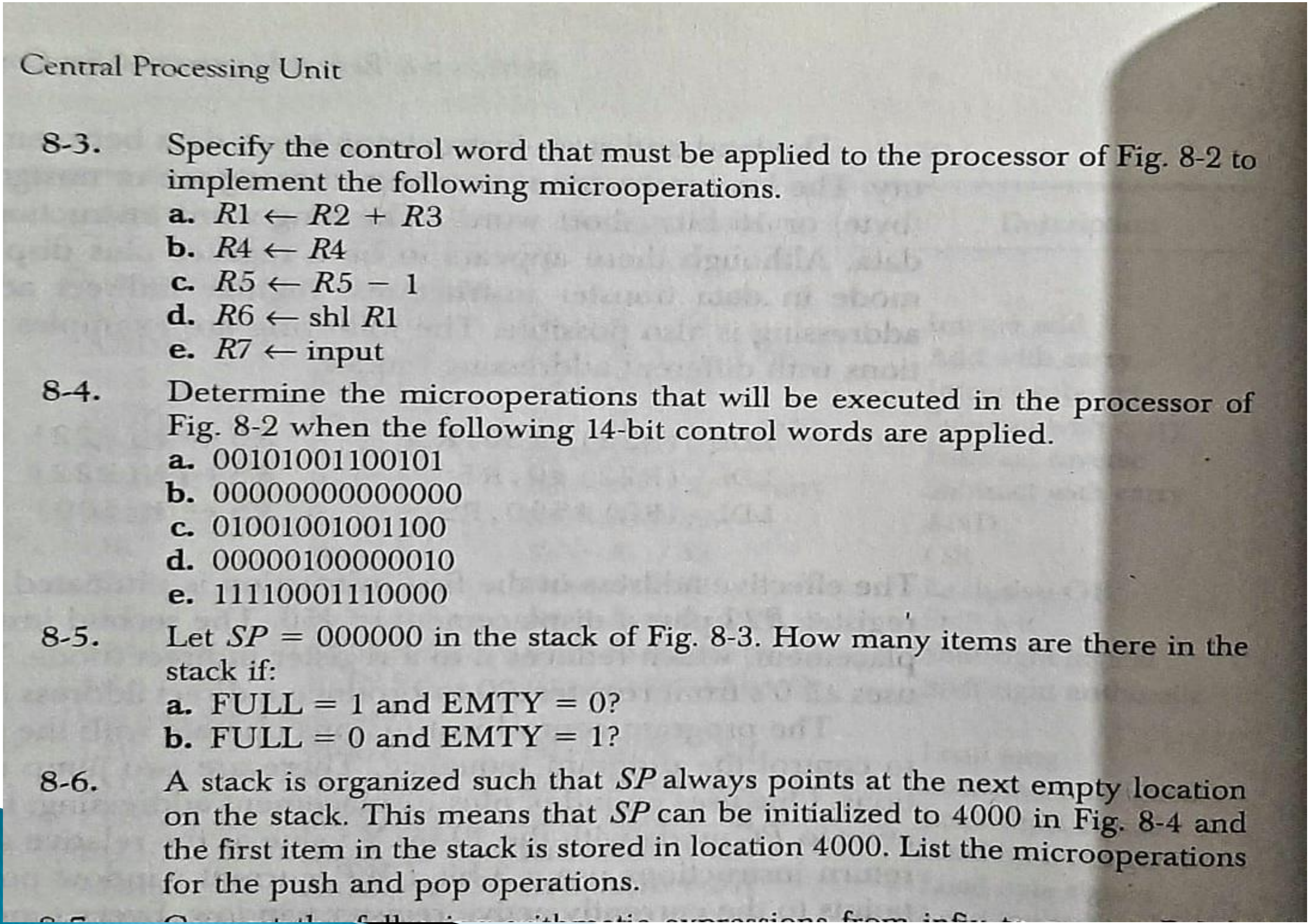
② A Common Bus System having K-Registers of n-bits produce n-line common bus.

# No of Multiplexers needed = n
Size of each Multiplexer = K x 1.

(a) No of Multiplexer = 32 (n- No of bits), Size = 16 x 1 (K x 1).
(b) 4 inputs to each to Selection one amongs 16 Reg.
(c) 4 to 16 (to Select one amongs 16).
(d) 32 (A) + 32 (B) + 1 (INPUT) = 65
32 (O) + 1 (O. Carry) = 33
(e)

4	4	4	6
SELA	SEL B	SEL D	OPR

= 18 bits



8-3. Specify the control word that must be applied to the processor of Fig. 8-2 to implement the following microoperations.

a. $R1 \leftarrow R2 + R3$

b. $R4 \leftarrow R4$

c. $R5 \leftarrow R5 - 1$

d. $R6 \leftarrow \text{shl } R1$

e. $R7 \leftarrow \text{input}$

8-4. Determine the microoperations that will be executed in the processor of Fig. 8-2 when the following 14-bit control words are applied.

a. 00101001100101

b. 00000000000000

c. 01001001001100

d. 00000100000010

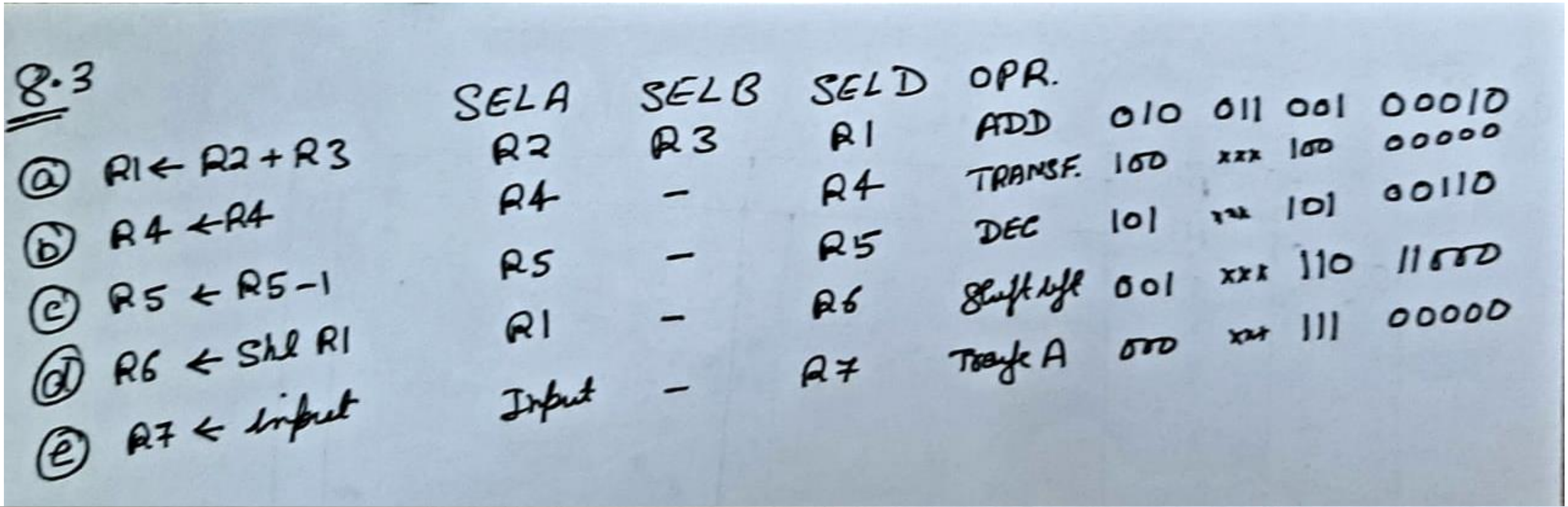
e. 11110001110000

8-5. Let $SP = 000000$ in the stack of Fig. 8-3. How many items are there in the stack if:

a. FULL = 1 and EMTY = 0?

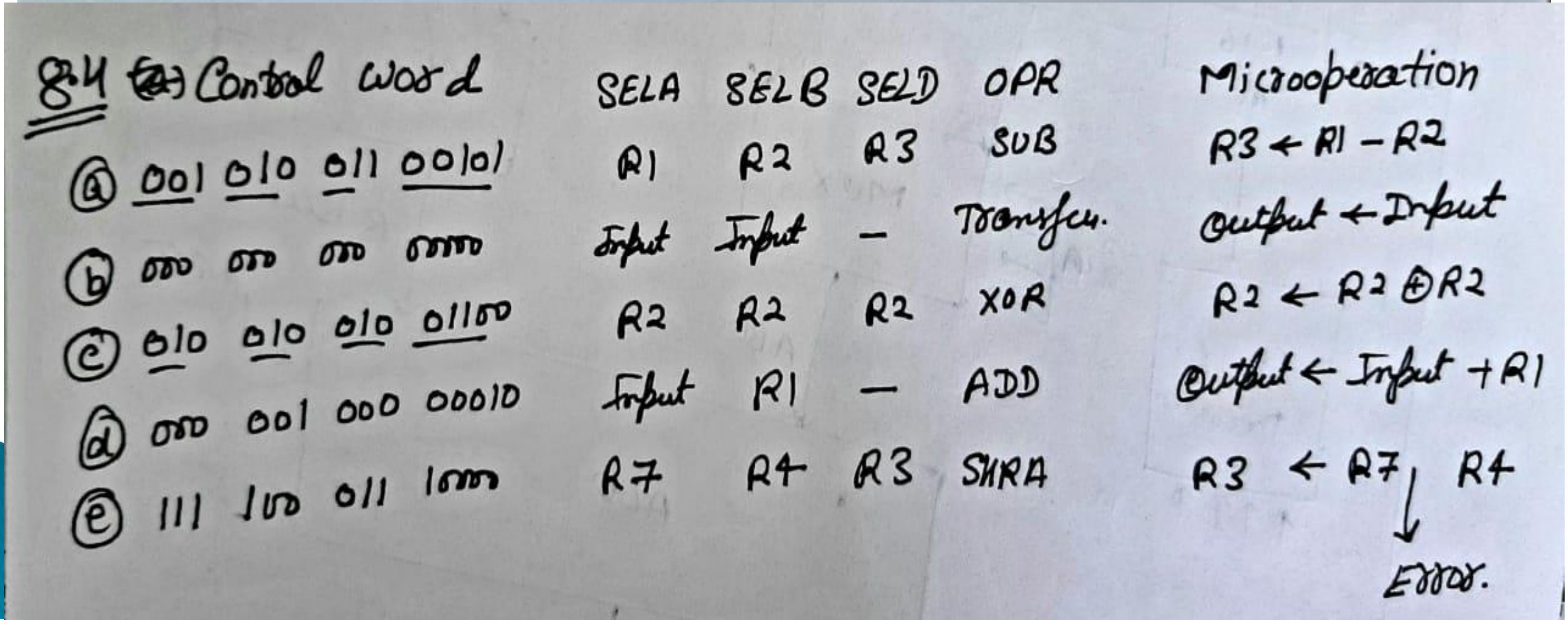
b. FULL = 0 and EMTY = 1?

8-6. A stack is organized such that *SP* always points at the next empty location on the stack. This means that *SP* can be initialized to 4000 in Fig. 8-4 and the first item in the stack is stored in location 4000. List the microoperations for the push and pop operations.



8.3

	SELA	SELB	SELD	OPR.				
(a) R1 ← R2 + R3	R2	R3	R1	ADD	010	011	001	00010
(b) R4 ← R4	R4	—	R4	TRANSF.	100	xxx	100	00000
(c) R5 ← R5 − 1	R5	—	R5	DEC	101	xxx	101	00110
(d) R6 ← Shl R1	R1	—	R6	Shift left	001	xxx	110	11000
(e) R7 ← input	Input	—	R7	Transfe A	000	xxx	111	00000



8.4 ~~(a)~~ Control word

Control word	SELA	SELB	SELD	OPR	Microoperation
(a) 001 010 011 00101	R1	R2	R3	SUB	R3 ← R1 − R2
(b) 000 000 000 00000	Input	Input	—	Transfer.	Output ← Input
(c) 010 010 010 01100	R2	R2	R2	XOR	R2 ← R2 ⊕ R2
(d) 000 001 000 00010	Input	R1	—	ADD	Output ← Input + R1
(e) 111 100 011 10000	R7	R4	R3	SHRA	R3 ← R7, R4 ↓ Error.

STACK ORGANIZATION

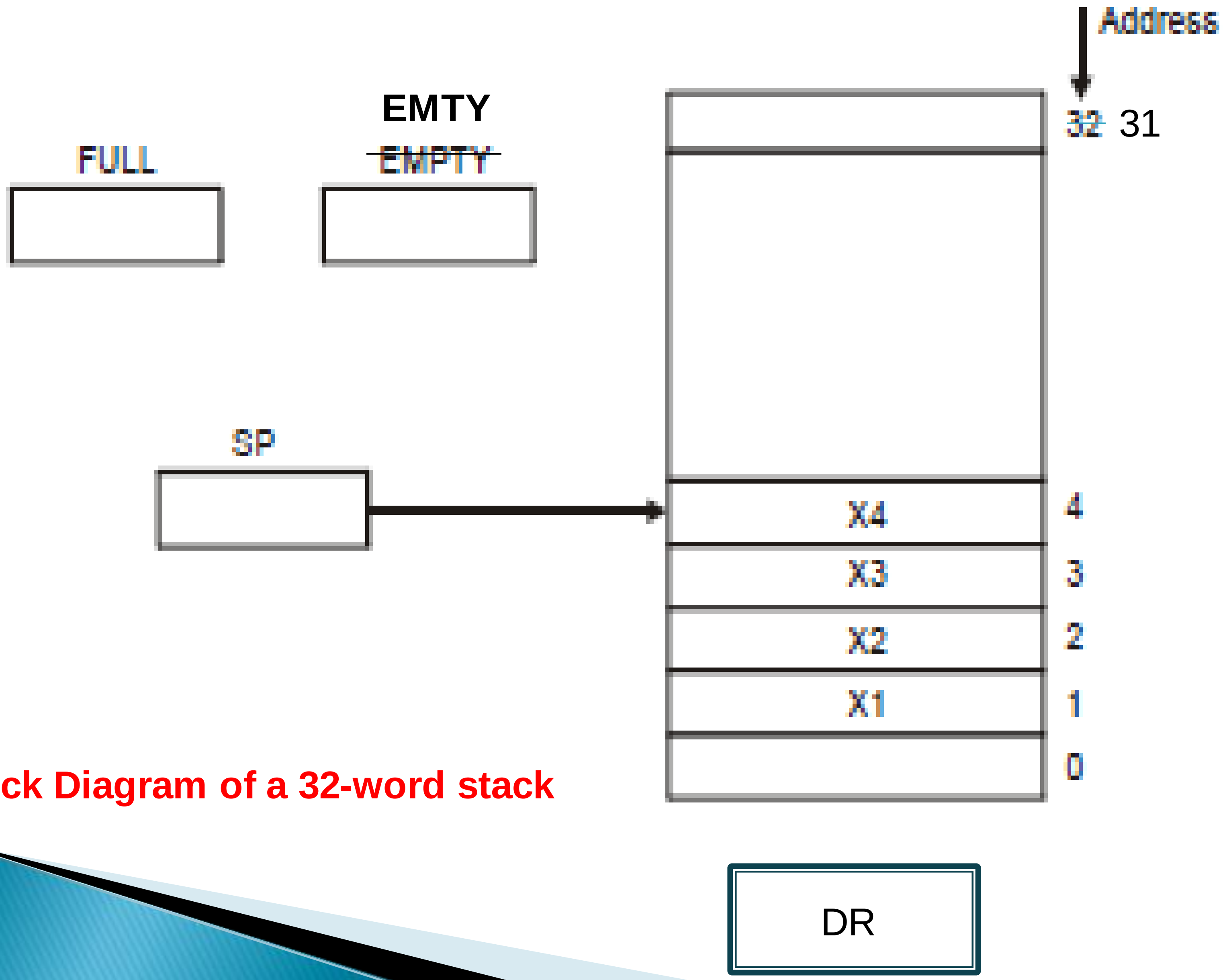
A stack is an ordered collection of item which permits the insertion or detection of an item to occur only at one end. The stack is also known as last-in first-out list. The stack can be considered as a storage method in which the items that stored last is the first item to be removed ex: stack of tray. Operations performed on Stack are:

PUSH: The insertion operation is known as push .

POP: The deletion operation is known as pop.

The stack in a digital computer is a part of memory unit. Also, with the stack an address register is associated that holds the address of the last element stored in the stack. This address register is known as Stack Pointer (SP). Thus, the stack pointer always points to the top most element of the stack.

STACK ORGANIZATION



Block Diagram of a 32-word stack

REGISTER STACK ORGANIZATION

Initially, the SP is cleared to 0 so the stack pointer points to the word at address 0. Also, the one-bit register FULL is cleared to 0, indicating that the stack is not full and the register EMPTY is set to 1. A new item is inserted into the stack by push operation. **The PUSH operation will be the set of following micro operations :**

SP ← SP + 1	Increment stack pointer
M[SP] ← DR	Add item on the top of stack
If (SP = 0) then (FULL ← 1)	Check if stack is full
EMPTY ←0	Mark the stack not empty.

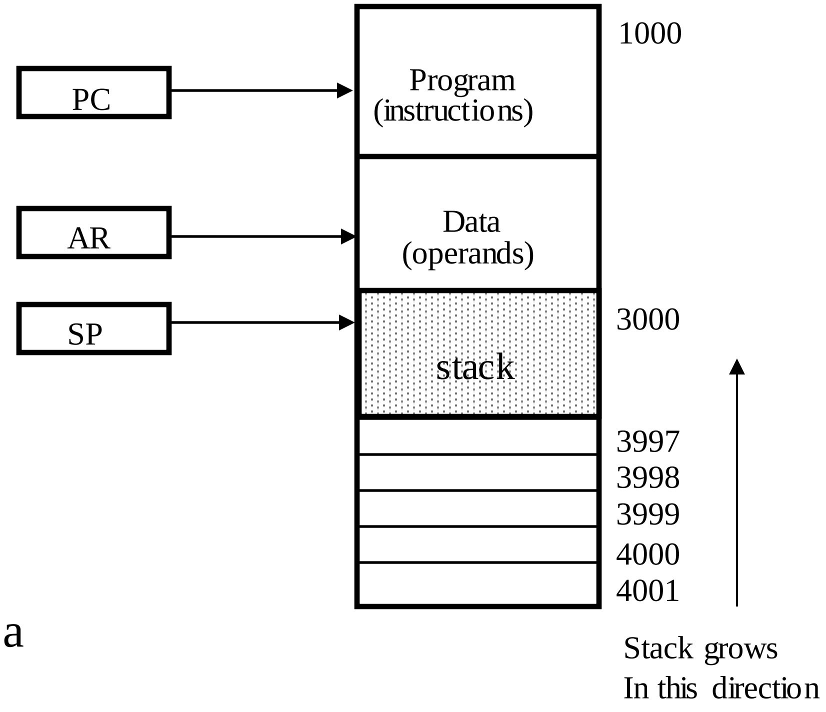
If the stack is not empty, an item can be deleted from the stack using the POP operation. **The POP operation is implemented by the following set of micro operations.**

DR ← M [SP]	Read item from the top of stack
SP ← SP – 1	Decrement stack pointer
If (SP = 0) then (EMPTY← 1)	Check if stack is empty
FULL ← 0	Mark the stack not full.

The top item is read from the stack into DR, then the SP is decremented by 1 so that it points to top of stack. The SP is checked whether it is zero on not. If zero, EMPTY sets to 1 indicating that the stack is empty.

MEMORY STACK ORGANIZATION

Memory with Program, Data, and Stack Segments



- A portion of memory is used as a stack with a processor register as a stack pointer

- PUSH: SP ← SP - 1
  M[SP] ← DR
- POP: DR ← M[SP]
  SP ← SP + 1

- Most computers do not provide hardware to check stack overflow (full stack) or underflow (empty stack) → must be done in software

REVERSE POLISH NOTATION

- Arithmetic Expressions: A + B

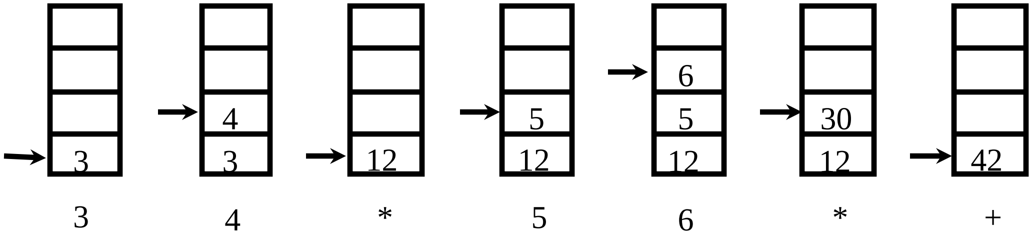
A + B	Infix notation
+ A B	Prefix or Polish notation
A B +	Postfix or reverse Polish notation

- The reverse Polish notation is very suitable for stack manipulation

- Evaluation of Arithmetic Expressions

Any arithmetic expression can be expressed in parenthesis-free Polish notation, including reverse Polish notation

$$(3 * 4) + (5 * 6) \quad \Rightarrow \quad 3\ 4\ *\ 5\ 6\ *\ +$$



PROCESSOR ORGANIZATION

- In general, most processors are organized in one of 3 ways
  - Single register (Accumulator) organization
    - Basic Computer is a good example
    - Accumulator is the only general purpose register
  - General register organization
    - Used by most modern computer processors
    - Any of the registers can be used as the source or destination for computer operations
  - Stack organization
    - All operations are done using the hardware stack
    - For example, an OR instruction will pop the two top elements from the stack, do a logical OR on them, and push the result on the stack

INSTRUCTION FORMAT

- Instruction Fields

OP-code field - specifies the operation to be performed

Address field - designates memory address(es) or a processor register(s)

Mode field - determines how the address field is to be interpreted (to get effective address or the operand)

- The number of address fields in the instruction format depends on the internal organization of CPU

- The three most common CPU organizations:

Single accumulator organization:

```
ADD     X                 /* AC ← AC + M[X]  */
```

General register organization:

```
ADD     R1, R2, R3          /* R1 ← R2 + R3  */
ADD     R1, R2            /* R1 ← R1 + R2  */
MOV     R1, R2            /* R1 ← R2  */
ADD     R1, X             /* R1 ← R1 + M[X]  */
```

Stack organization:

```
PUSH    X                 /* TOS ← M[X]  */
ADD
```

THREE, AND TWO-ADDRESS INSTRUCTIONS

- Three-Address Instructions

Program to evaluate X = (A + B) * (C + D) :

```
ADD    R1, A, B   /*  R1 □ M[A] + M[B]     */
ADD    R2, C, D   /*  R2 □ M[C] + M[D]     */
MUL    X, R1, R2          /*  M[X] □ R1 * R2              */
```

- Results in short programs
- Instruction becomes long (many bits)

- Two-Address Instructions

Program to evaluate X = (A + B) * (C + D) :

```
MOV    R1, A          /* R1 □ M[A]         */
ADD    R1, B          /* R1 □ R1 + M[A]   */
MOV    R2, C          /* R2 □ M[C]         */
ADD    R2, D          /* R2 □ R2 + M[D]   */
MUL    R1, R2         /* R1 □ R1 * R2      */
MOV    X, R1          /* M[X] □ R1          */
```

ONE, AND ZERO-ADDRESS INSTRUCTIONS

- One-Address Instructions

- Use an implied AC register for all data manipulation
- Program to evaluate X = (A + B) * (C + D) :

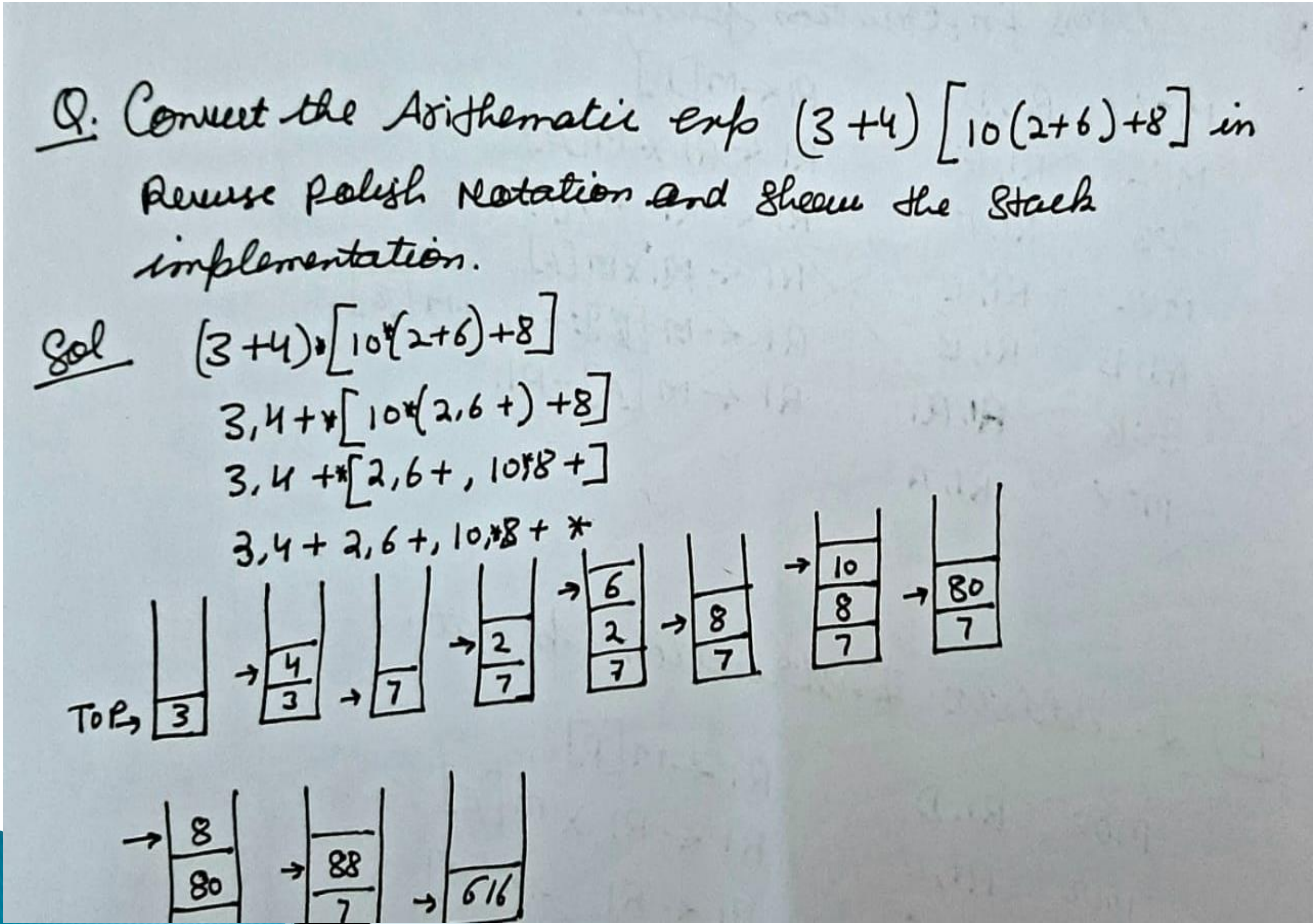
```
LOAD    A       /*  AC □ M[A]       */
ADD     B       /*  AC □ AC + M[B]  */
STORE   T       /*  M[T] □ AC       */
LOAD    C       /*  AC □ M[C]       */
ADD     D       /*  AC □ AC + M[D]            */
MUL     T       /*  AC □ AC * M[T]            */
STORE   X       /*  M[X] □ AC       */
```

- Zero-Address Instructions

- Can be found in a stack-organized computer
- Program to evaluate X = (A + B) * (C + D) :

```
PUSH        A           /*  TOS □ A                      */
PUSH        B           /*  TOS □ B                      */
ADD                     /*  TOS □ (A + B)                */
PUSH        C           /*  TOS □ C                      */
PUSH        D           /*  TOS □ D                      */
ADD                     /*  TOS □ (C + D)                */
MUL                     /*  TOS □ (C + D) * (A + B)  */
POPX        /*  M[X] □ TOS                               */
```

INSTRUCTION FORMAT



Q. Convert the Arithematic exp (3+4)[10(2+6)+8] in Reverse polish Notation and show the Stack implementation.

Sol

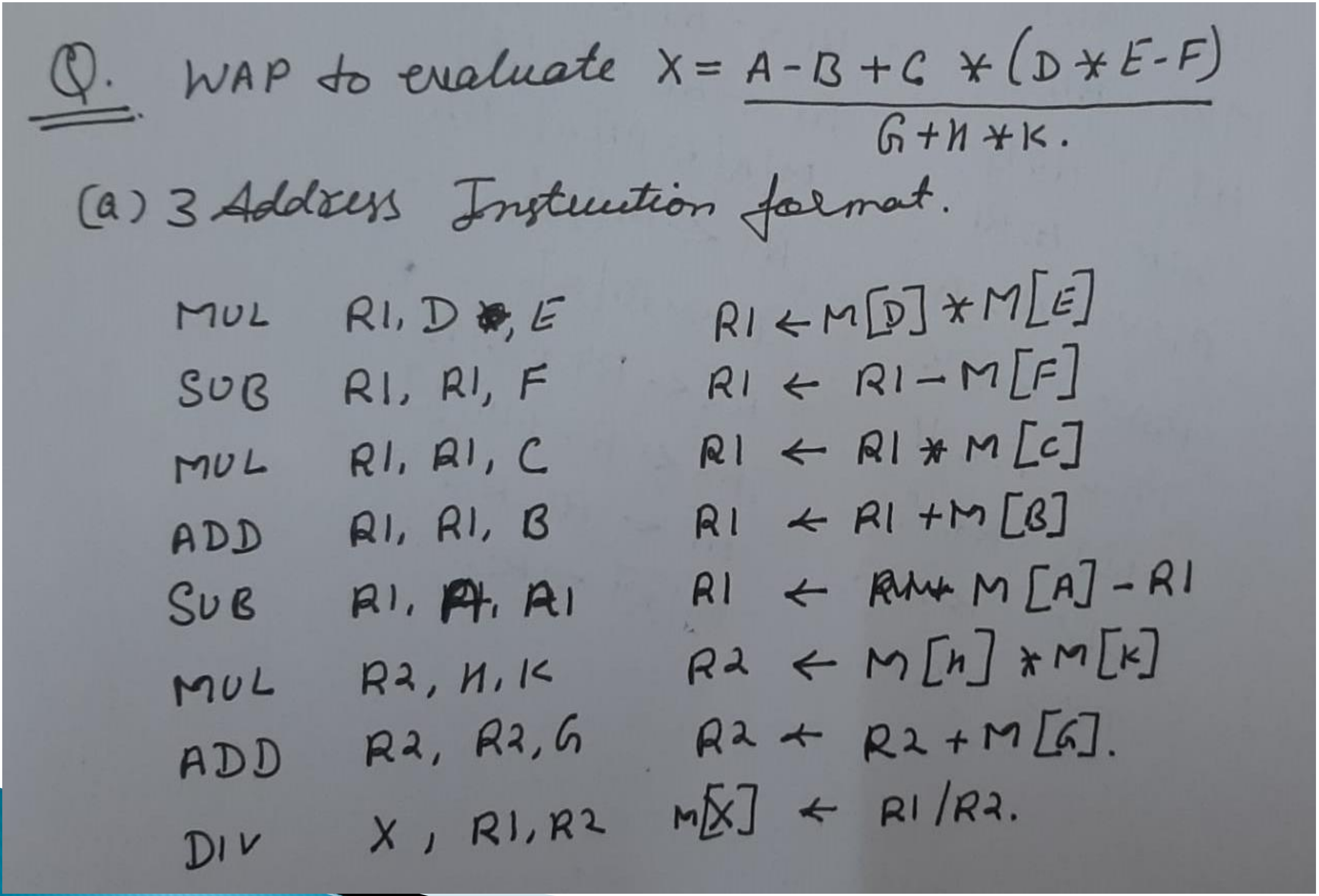
(3+4)*[10*(2+6)+8]

3,4+*[10*(2,6+)+8]

3,4+*[2,6+, 10*8+]

3,4+2,6+,10*8+*

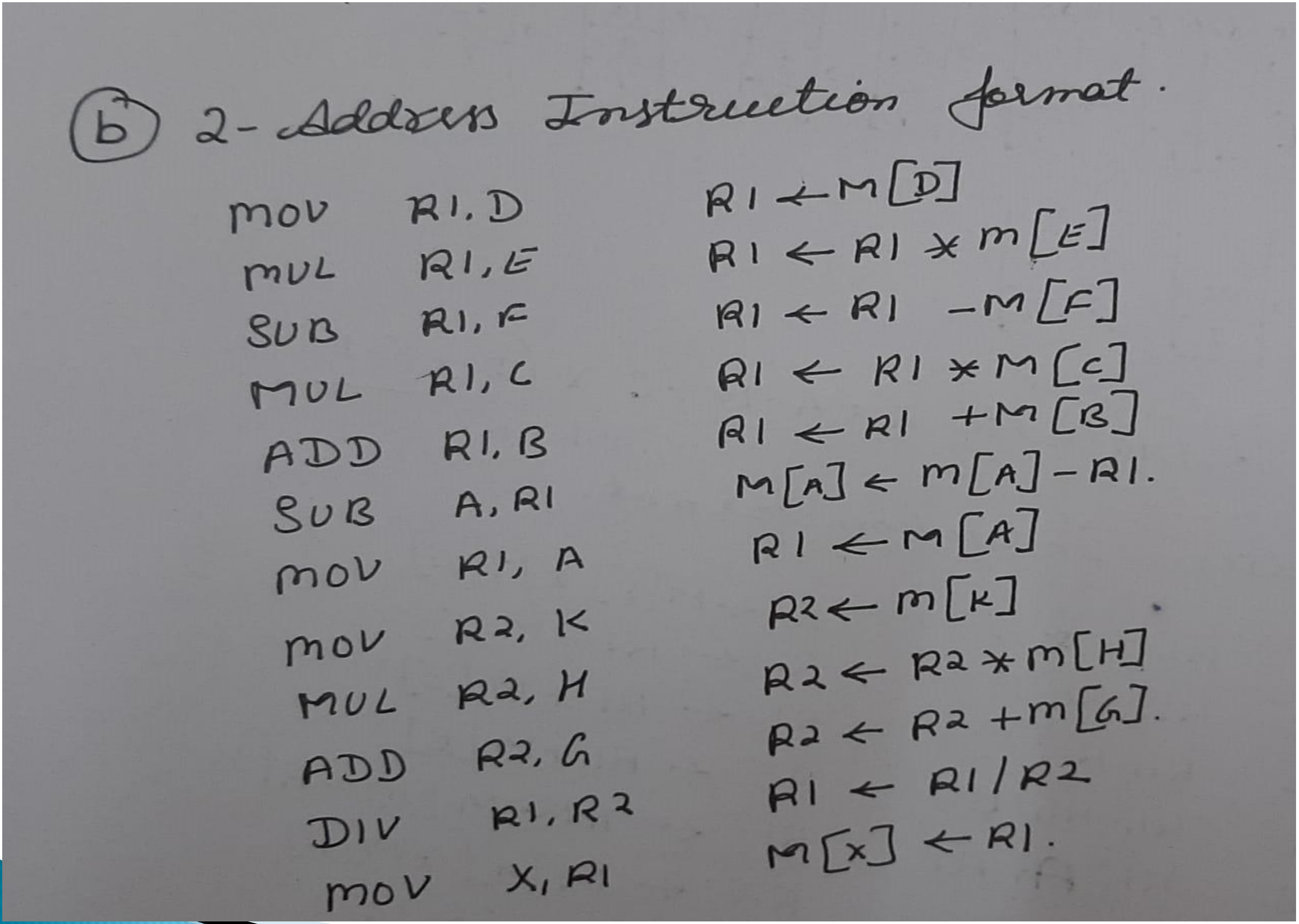
TOP→ 3
→ 4
3
→ 7
→ 2
7
→ 6
2
7
→ 8
7
→ 10
8
7
→ 80
7
→ 8
80
→ 88
7
→ 616



Q. WAP to evaluate $X = \dfrac{A - B + C * (D * E - F)}{G + H * K}$.

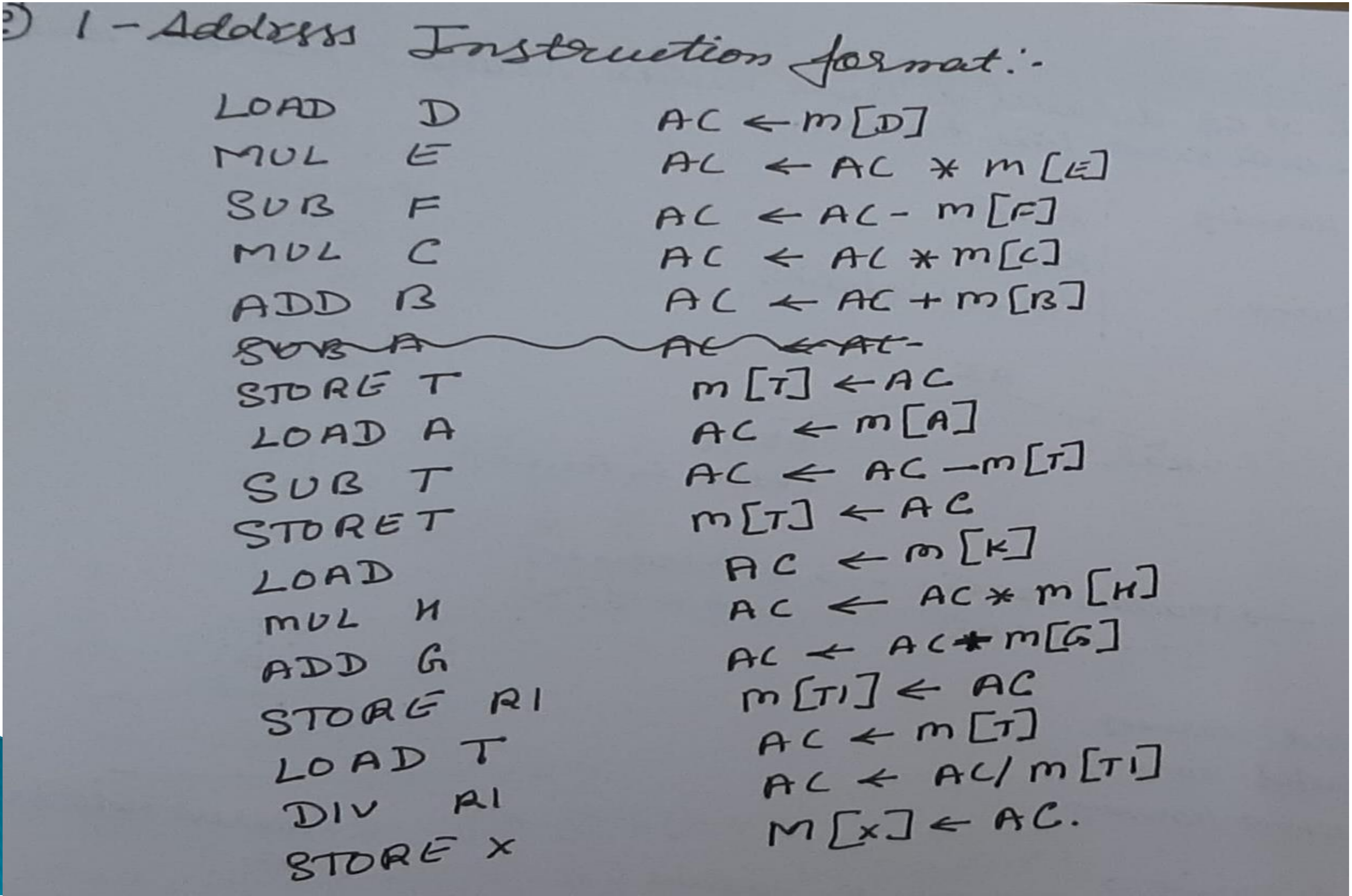
(a) 3 Address Instruction format.

```
MUL   R1, D, E       R1 ← M[D] * M[E]
SUB   R1, R1, F      R1 ← R1 − M[F]
MUL   R1, R1, C      R1 ← R1 * M[C]
ADD   R1, R1, B      R1 ← R1 + M[B]
SUB   R1, A, R1      R1 ← M[A] − R1
MUL   R2, H, K       R2 ← M[H] * M[K]
ADD   R2, R2, G      R2 ← R2 + M[G].
DIV   X, R1, R2      M[X] ← R1/R2.
```



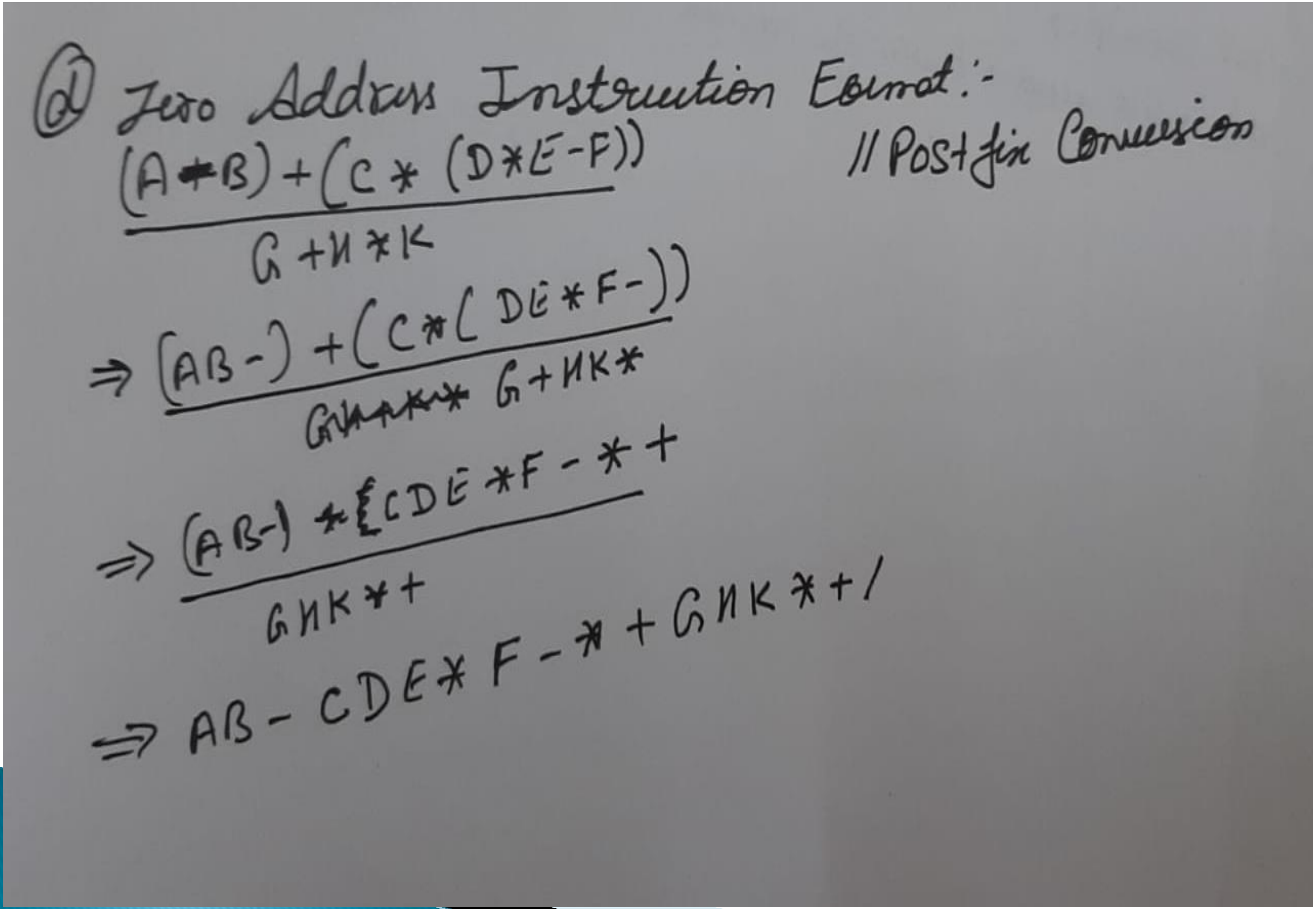
(b) 2- Address Instruction format.

mov R1. D	R1 ← M[D]
MUL R1, E	R1 ← R1 * m[E]
SUB R1, F	R1 ← R1 − M[F]
MUL R1, C	R1 ← R1 * M[C]
ADD R1, B	R1 ← R1 + M[B]
SUB A, R1	M[A] ← m[A] − R1.
mov R1, A	R1 ← M[A]
mov R2, K	R2 ← m[K]
MUL R2, H	R2 ← R2 * m[H]
ADD R2, G	R2 ← R2 + m[G].
DIV R1, R2	R1 ← R1 / R2
mov X, R1	M[X] ← R1.



2) 1-Addrsss Instruction format:-

LOAD D	AC ← m[D]
MUL E	AC ← AC * m[E]
SUB F	AC ← AC - m[F]
MUL C	AC ← AC * m[C]
ADD B	AC ← AC + m[B]
~~SUB A~~	~~AC ← AC -~~
STORE T	m[T] ← AC
LOAD A	AC ← m[A]
SUB T	AC ← AC - m[T]
STORE T	m[T] ← AC
LOAD	AC ← m[K]
MUL H	AC ← AC * m[H]
ADD G	AC ← AC * m[G]
STORE R1	m[T1] ← AC
LOAD T	AC ← m[T]
DIV R1	AC ← AC / m[T1]
STORE X	M[x] ← AC.



(2) Zero Address Instruction Format:-

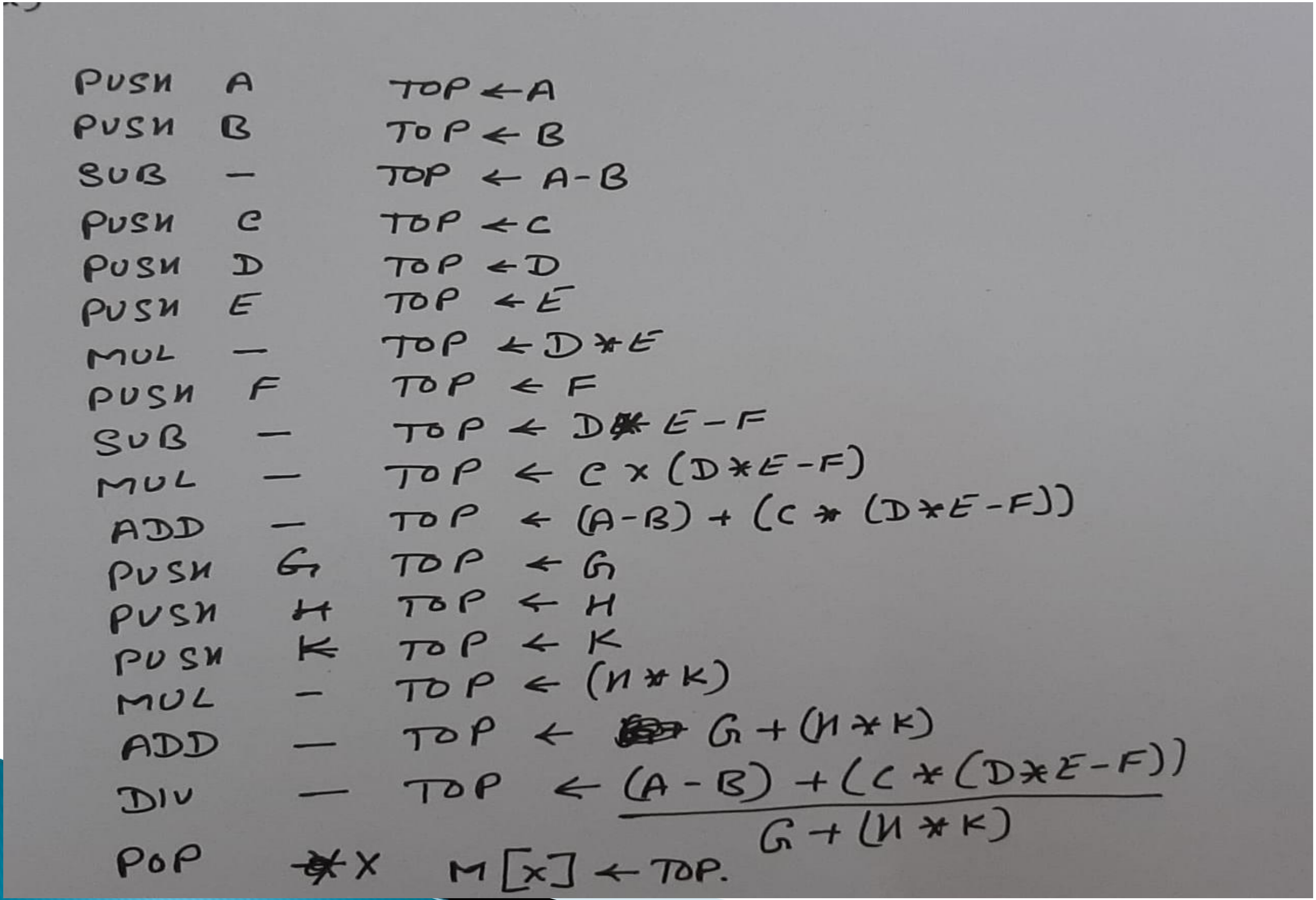
// Postfix Conversion

$$\frac{(A-B)+(C*(D*E-F))}{G+H*K}$$

$$\Rightarrow \frac{(AB-)+(C*(DE*F-))}{\cancel{GHK*}\ G+HK*}$$

$$\Rightarrow \frac{(AB-)*\{CDE*F-*+}{GHK*+}$$

$$\Rightarrow AB-CDE*F-*+GHK*+/$$



PUSH	A	TOP ← A
PUSH	B	TOP ← B
SUB	—	TOP ← A-B
PUSH	C	TOP ← C
PUSH	D	TOP ← D
PUSH	E	TOP ← E
MUL	—	TOP ← D*E
PUSH	F	TOP ← F
SUB	—	TOP ← D*E-F
MUL	—	TOP ← C x (D*E-F)
ADD	—	TOP ← (A-B) + (C * (D*E-F))
PUSH	G	TOP ← G
PUSH	H	TOP ← H
PUSH	K	TOP ← K
MUL	—	TOP ← (H*K)
ADD	—	TOP ← G+(H*K)
DIV	—	TOP ← $\frac{(A-B)+(C*(D*E-F))}{G+(H*K)}$
POP	X	M[X] ← TOP.

ADDRESSING MODES

- **Addressing Modes:-** Specifies a rule for interpreting or modifying the address field of the instruction (before the operand is actually referenced)

Advantage of addressing modes

- to give programming flexibility to the user
- to use the bits in the address field of the instruction efficiently

- **Implied Mode:-** Address of the operands are specified implicitly in the definition of the instruction. Ex:- CLA, CME, INP

- **Immediate Mode :-** Instead of specifying the address of the operand, operand itself is specified in the instruction. No need to specify address in the instruction.

  MVI A,34H

- **Register Mode or Register Direct :-** The operands are in the register that reside within the CPU. Faster to acquire an operand than the memory addressing

- **Register Indirect Mode:-**Instruction specifies a register in the CPU whose contents give the address of the operand in memory.

- **Autoincrement or Autodecrement Mode:-**Similar to register Indirect Mode except that the register is incremented or decremented after (or Before)by 1 automatically when its value is used to access memory.

- **Direct Address Mode :-** In this the effective address is equal to the address part of the instruction. Faster than the other memory addressing modes

- **Indirect Addressing Mode :-** In this the address field of the instruction gives the address where the effective address is stored in the memory.

- **Relative Addressing Modes :-**

  (R = PC, where PC: Program Counter)

  - EA = PC + IR(address)

- **Indexed Addressing Mode :-**

  (R = IX, where IX: Index Register)

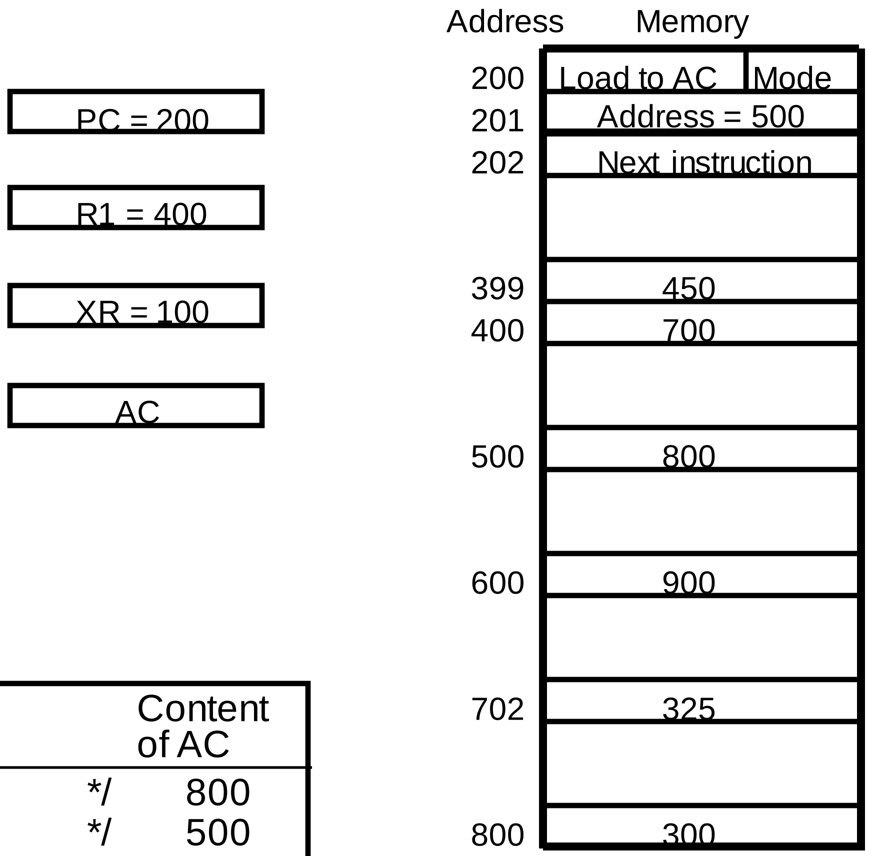
  - EA = IX + IR(address)

- **Base Register Addressing Mode:-**

  (R = BAR, where BAR: Base Address Register)

  - EA = BAR + IR(address)

ADDRESSING MODE EXAMPLES



PC = 200

R1 = 400

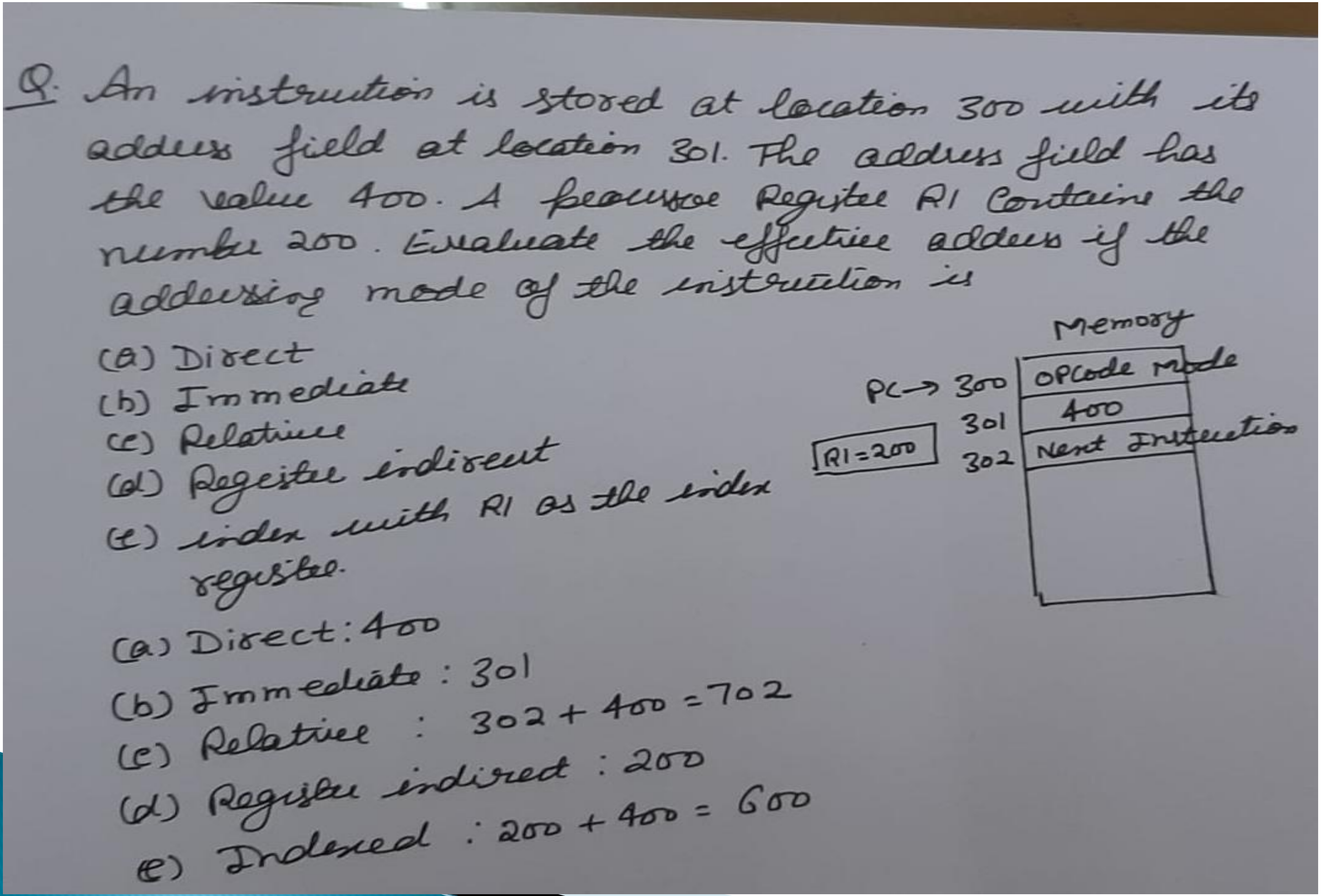
XR = 100

AC

Address	Memory
200	Load to AC Mode
201	Address = 500
202	Next instruction
399	450
400	700
500	800
600	900
702	325
800	300

Addressing Mode	Effective Address			Content of AC
Direct address	500	/* AC ← (500)	*/	800
Immediate operand	-	/* AC ← 500	*/	500
Indirect address	800	/* AC ← ((500))	*/	300
Relative address	702	/* AC ← (PC+500)	*/	325
Indexed address	600	/* AC ← (RX+500)	*/	900
Register	-	/* AC ← R1	*/	400
Register indirect	400	/* AC ← (R1)	*/	700
Autoincrement	400	/* AC ← (R1)+	*/	700
Autodecrement	399	/* AC ← -(R)	*/	450

ADDRESSING MODES - EXAMPLES



Q. An instruction is stored at location 300 with its address field at location 301. The address field has the value 400. A processor Register R1 contains the number 200. Evaluate the effective address if the addressing mode of the instruction is

(a) Direct
(b) Immediate
(c) Relative
(d) Register indirect
(e) index with R1 as the index register.

Memory

PC → 300	OPCode Mode
301	400
302	Next Instruction

R1 = 200

(a) Direct: 400
(b) Immediate : 301
(c) Relative : 302 + 400 = 702
(d) Register indirect : 200
(e) Indexed : 200 + 400 = 600