Instruction Codes:- An instruction code is a group of bits that instruct the computer to perform a specific operation.

- An instruction is stored in binary code.
- An instruction code is a command or statement to perform some action in the computer. As we know a computer can understand only two symbols that is 0 & 1.
- So every instructions must be given to the computer in the form of a group of 0's and 1's.
- An instruction code is a group of 16-bits, which is divided into three parts.

→ 0/1
→ OP Code
→ Address or operand.

Copyrighted by Codeleap.io
All rights reserved.

Addressing Modes ←

15	14 13 12 11	0
0/1	OP Code	Address

→ Instruction format

- Now, name the first heading address considering our Memory of 4096 words [2^{12} words] we need atleast 12 bytes identify a Memory words uniquely.
- So address part of our instruction code contains 12-bits from [0 to 11]

0/1 → It specify the Addressing mode (0 for direct addressing and 1 for indirect addressing)
[Mode] indicates how data is represented.

In the 15th bit of the instruction code The Method in which a target address or effective address is recognized within the

instruction.

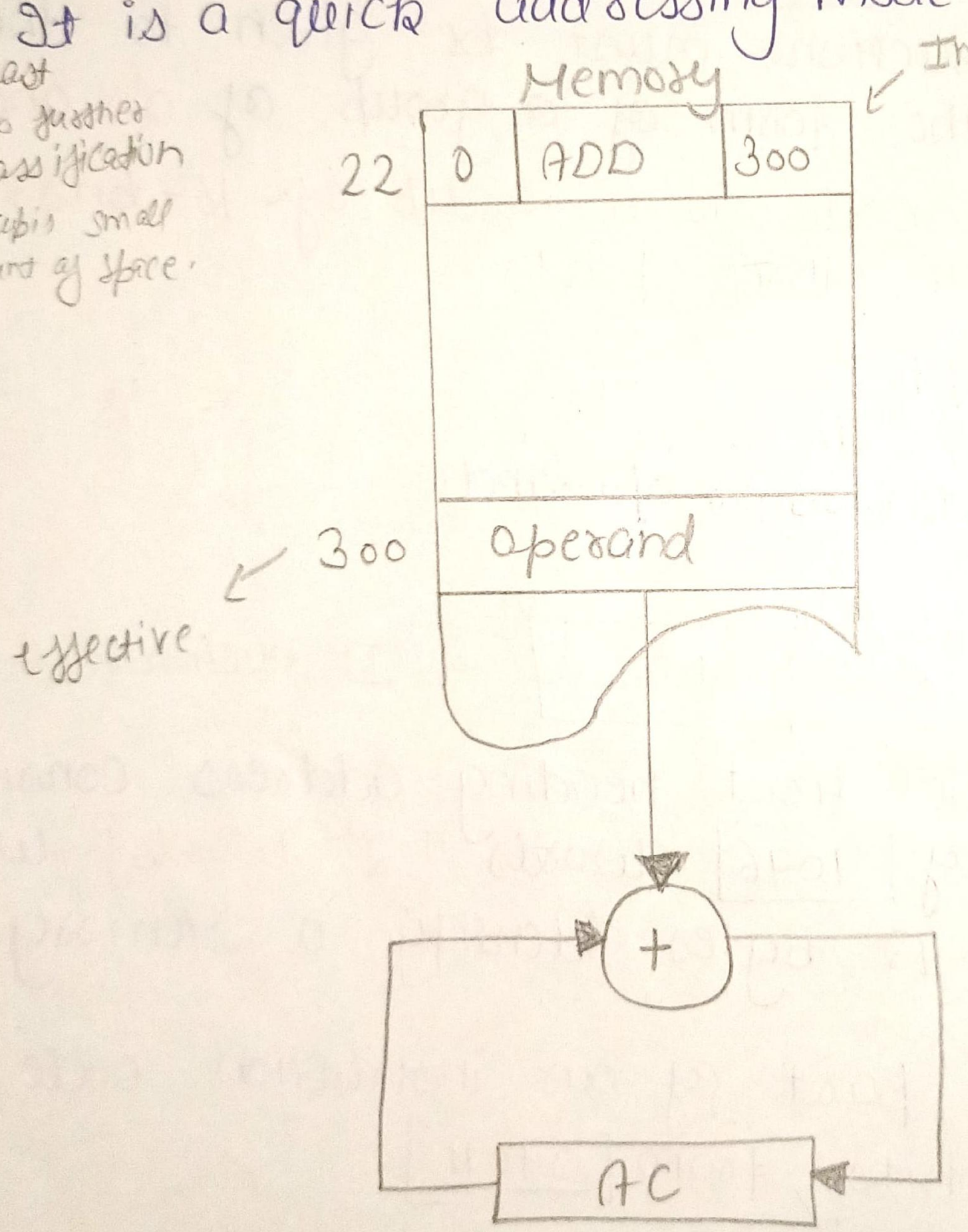
- They Classified into two parts
  - a) Direct addressing
  - b) Indirect addressing

Copyrighted by Codeleap.io
All rights reserved.

a) Direct addressing (Absolute) :-> The address field contain the effective address of th operand.

- This addressing mode requires one memory reference only.
- It is a quick addressing mode.

- fast
- No further classification
- occupies small amount of space.



[• Placed in address in Memory]

[• The I bit is 0, the instruction i recognised as d direct address]

[• The opcode specify an ADD instruction the addressing par the binary equiva of 300]

[• The control fir the operand in m at address 457 adds it to the of AC.]

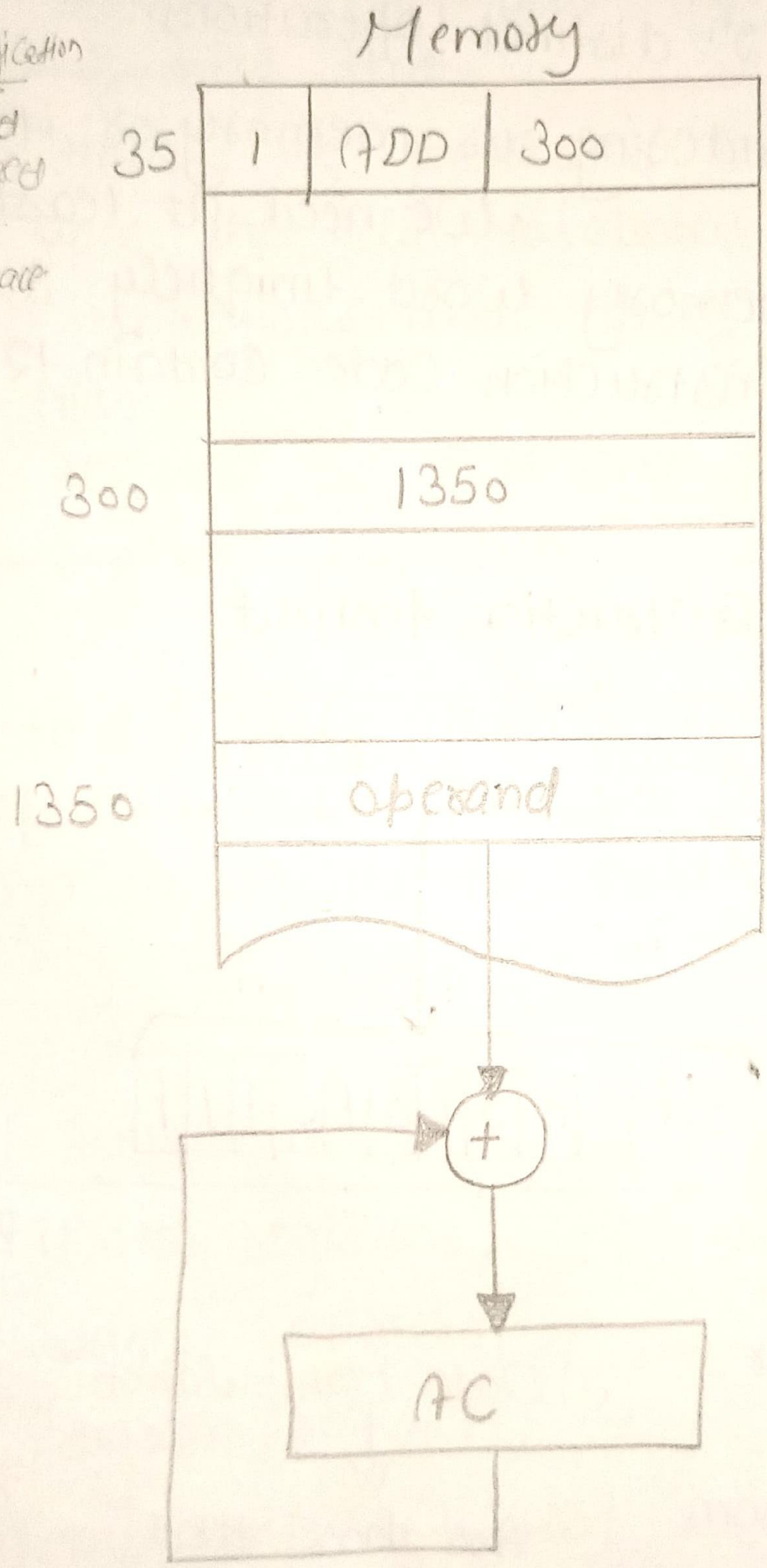
Effective address (EA) [offset] :-> effective address is the address of the exact m location where the value the operand is present

(b) **Indirect addressing** :-> In this mode, the address field contains reference of the effective address.

- It requires two memory references.
- It is a slower in comparison to direct addressing mode.

further Classification
- Memory Indirect
- Register Indirect

occupies large amount of space



[· The instruction in address 35 has a mode bit I = 1.]

[· The address part is the binary equivalent of 300.

[· The operand found in address 1350 is then added to the content of AC.

Copyrighted by Codeleap.io
All rights reserved.

Accumulator register :-> This register is found in single register processors and it performs all operations with memory operands.

OP Code :→ It Stands for operational Code)
→ 3- bits [12, 13, 14] are used to specify the operation to be performed.

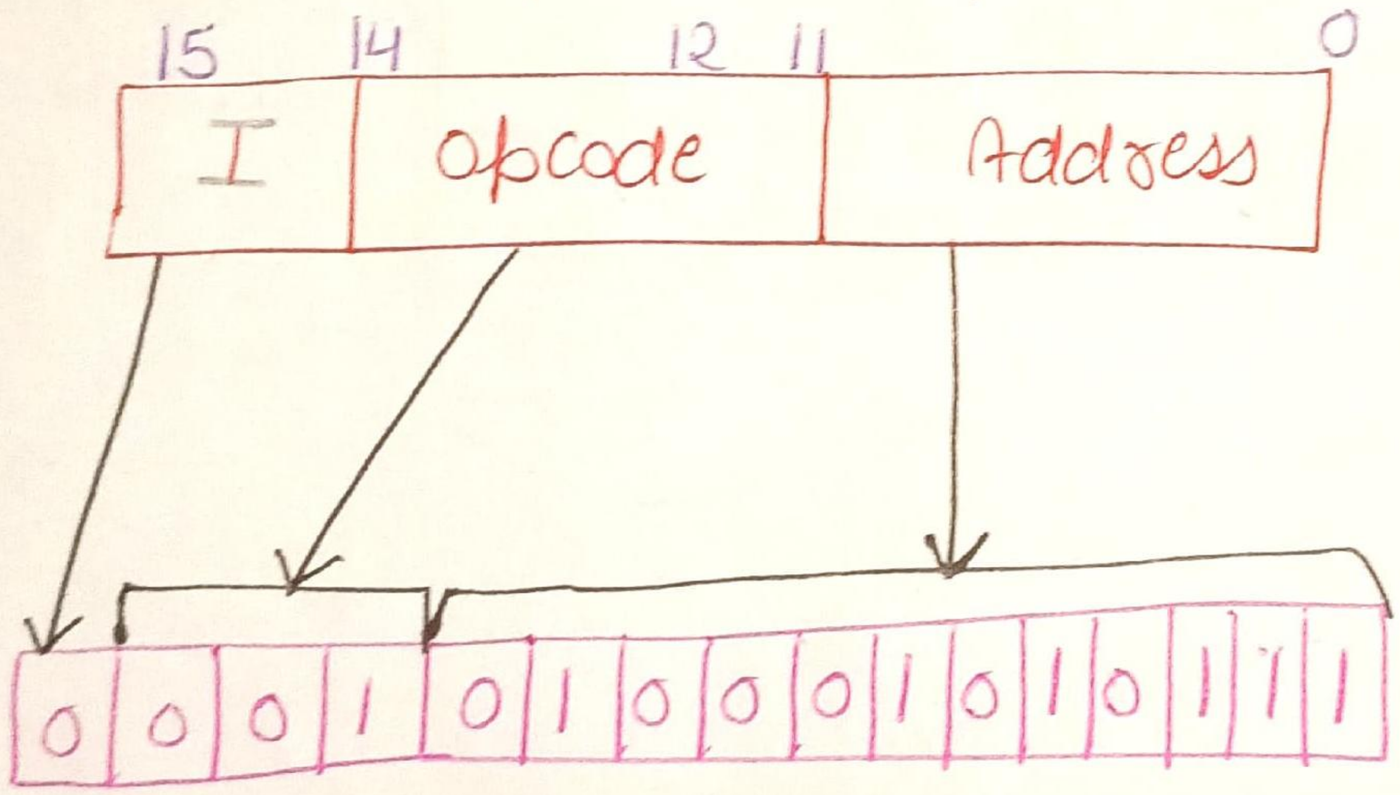
- Such operations like add, subtract, Multiply, Shift operations.
- The operation Code must consists of atleast n-bi for a given 2^n distinct operations.

Address :→ Considering our memory of 4096 words (2^{12} words). We need at least 12 bits to identify a memory word uniquely so address part of our instruction Code contain 12- bit for

$\boxed{0 \text{ to } 11}$

Copyrighted by Codeleap.io
All rights reserved.

Instruction format



Instruction types

1) Data transfer
↓
Cause of data from one Location to another without Changing the binary int content.

2) Data Manipulation
↓
Are those that perform arithmetic, logic, shift operation

3) Program con
↓
This instruc provide decis making capab & Change the path taken by the pro when executed in the comp

Computer Registers :→ Registers are inbuilt memory location inside the CPU and provide the fastest access.

- Registers are limited in number due to their very high cost.
- Its an electronic component that stores information in form of bits that is 0 and 1.
- Register makes the processor faster.
- Register uses the cells to store bit that is called flip flop. Register made up of multiple cells that contain bits.
- Usually registers are 8, 16 and 32 bit but commonly 32 bits registers are used.
- Registers are a type of memory used to quickly accept, store, and transfer data and instructions.
- The computer needs processor register for Manipulating Data. and a register for holding the Memory address.

⇒ Types of Registers

1) AC [Accumulator] [→ Storing instructions]
2) DR [Data registers] [the registers are very fast memory which are used to execute programs and operations.]
3) AR [Address registers]
4) PC [Program registers]
5) TR [Temporary registers]
6) IR [Instruction registers]
7) INPR [Input registers]
8) OUTR [output registers]

Copyrighted by Codeleap.io
All rights reserved.

1) Accumulator: → Bits :-> 16

Purpose :-> It's a processor register.

- This is the most frequently used register used to store data taken from memory.
- Size would be 16 bits starts from 0 to 15.

2) Data Register: → Bits :-> 16

Purpose :-> holds memory operand

- The operand is stored in the Data Register from memory.
- When a direct or indirect addressing operand is found, it is placed in the Data register.
- Word size in RAM is 16 bits so Data register size would be 16.

3) Address Register: → Bits :-> 12

Purpose :-> holds an address for Memory.

- CPU needs read and write data on memory.
- This register is used to store memory address of data for CPU.
- It will store the address of operands, on which operations to be performed As it stores the address, there are 12 bits to identify a particular address in memory.

(Considering our memory of $4096 = 2^{12}$ words)

- The size of AR is equal to width of Memory address is directly related to size of the Memory because it contains an address

4) Program Counter:

Bits:→ 12

Purpose:→ It holds the address of the instruction.

- Stores address of next instruction place in memory. It points only that location where instruction exist. If location is empty it will never points it. Its size would be 12 bits.
- Every program in computer has many instruction But all the instructions are executed in sequence, one by one from top to bottom, to run a program. When the CPU is executing any particular instruction is to be executed nett. The address of the nett instruction is stored in a special register known as Program Counter because the program counter stores the address, we need 12 bytes to identify a unique address in memory. [considering our memory is of 4096 words = 2^{12} words]

5) Temporary Register:

Bits: 16

Purpose: hold Temporary data

- The temporary register is used to hold Temporary data.
- As Temporary register stores data, the no of bits it contains is the same as the number of bits in the data word.
- It is also the only register that can be read and written more than once in a single instruction.

Copyrighted by Codeleap.io
All rights reserved.

6) Instructions Register:-
Bits:-> 16
Purpose:-> hold an instruction c

- I is most significant bit, if 0 it means direc address, if its 1 then it shows indirect addres
- If value of 0 then directly go on that locatio fetch data and gives to accumulator.
- If value is 1 means indirect addressing, first u have to go 1st location which point to 2nd lo that points to 3rd location then here u will find data.
- operand has address of data on which we have to perform operation.

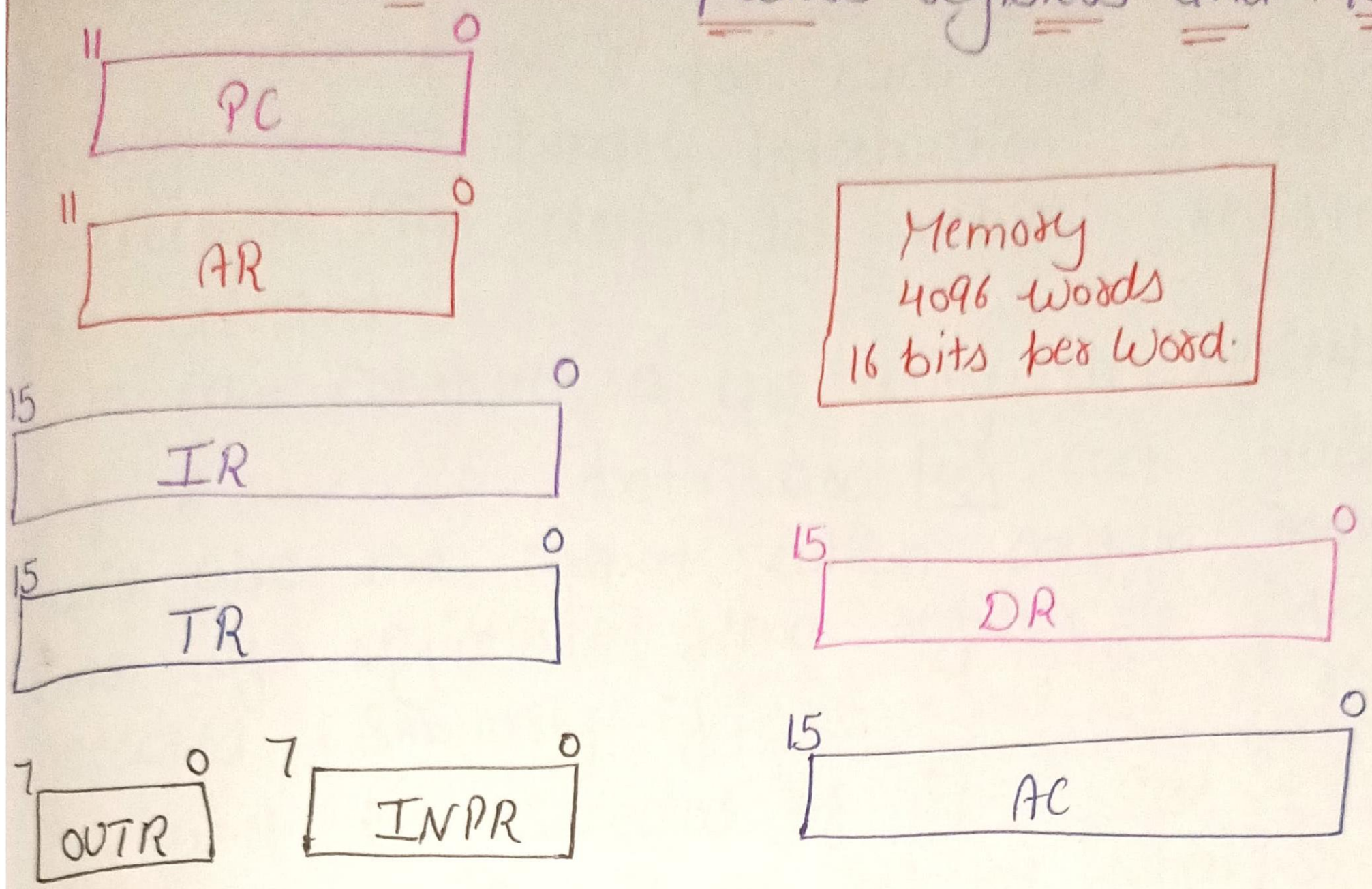
7) Input Register:-Bits:-> 8
Purpose:-> holds input character

- It stores the character which is currently ente or inputed as the character is of one byte. [1- byte = 8 bits] INPR take 8 bytes.
- Recieves data from input devices like mouse, keyboard etc and provides that data to accum and ALU.

8) Output Register:-> Bits:-> 8
Purpose:-> holds output character.

- It stores the character to be outputed [ma printed —] as the character is of one byte [1 byte = 8 bite]
- receives data from ALU and gives to output devices like printer scanners etc.

Basic computers registers and Memory



Common Bus System

- Basic computer units
  - Eight registers
  - Memory unit
  - Control unit

Copyrighted by Codeleap.io
All rights reserved.

- A bus is a pair of signal lines that allows multi-bit data to be transferred from one system to another.
- A common Bus is a more efficient Method of sending data in a system with multiple registers. The common bus connects the outputs of seven registers and Memory.
- A bus provides a means for people to communicate with one another
- The basic computer registers and Memory connection to a common bus system are depicted.

6)

- A common bus system is used to transfer the data b/w register. It consist of a set of common line, 1 for each bit of register, through which binary information is transfered.
- Selection line determined which register will be selected.
- We can construct a bus using the Multiplexer.
- If S1 and S0 both are $\boxed{0}$, each Multiplexer will give the output which contain indrisual bits of register $\boxed{A}$ 1) Hence register A [A] will be selected. Similarly if $\boxed{S1 = 0 \text{ and } S0 = 1}$ register $\boxed{B}$ 2) will be selected if $\boxed{S1 = 1 \text{ and } S0 = 0}$ [B]. the register $\boxed{C}$ 3) will be selected when $\boxed{S1 = 1, S0 = 1}$ [C] register $\boxed{d}$ will be selected.
- Number of Multiplexer needed will be = The Number of bits in register
- The size of MUX will depend upon the no. of register size will be $\boxed{n \times 1}$ for n is number of register.

Copyrighted by Codeleap.io
All rights reserved.

Computer Instructions

The basic computer has 16 bit instruction register which can denotte either three types instruction.

a) Memory reference
b) Register reference
c) Input / output

→ are those commands or instructions which are in the custom to generate a refer to Memory

- Computer instructions are a set of Machine Language instructions that a particular processor understands and executes.
- An instruction comprises of groups called fields. These fields include.

→ The operation Code (opcode) field which specifi the operation to be performed

→ The Address field which contains the locatio of the operand. i.e, register or Memory Location.

→ The Mode field which specifies how the operand will be Located.

Copyrighted by Codeleap.io
All rights reserved.

Mode	opcode	Operand / address

a) Memory reference instruction :→ These instructions refers to Memory address as an operand. The other operand is always accumulator. Specifies 12 bit address, 3 bit opcode (other than 111) and 1 bi addressing Mode for direct and indirect addressi

Ex: IR Contains = 0111001000000000, i.e. CMA after fetch and decode cycle we find out that it is a register reference instruction for complement accumulator.

Hence, AC <- ~AC

Code	Symbol	Description
7800	CLA	Clear AC
7400	CLE	Clear E
7200	CMA	Complement AC
7100	CME	" " E
7080	CIR	Circular right AC and E
7040	CIL	" " Left AC and E
7020	INC	Increment AC
7010	SPA	Skip next instruction if AC positive
7008	SNA	" " " " AC Negative
7004	SZA	" " " " AC zero
7002	SZE	" " " " E is 0
7001	HLT	Halt Computer

(c) Input - output Instruction

These instructions are for communication b/w computer and outside environment. The IR (14-12) is 111 and IR (15) is 1 [Differentiates it MR from register reference instructions]. The rest 12 bit specify I/O operation.

15	12 11	0
1 1 1 1	I/O operation	

[opcode = 111, I = 1]

Copyrighted by Codeleap.io
All rights reserved.

★

15	14 … 12	11 … 0
I	Opcode	Address

[opcode = 000 through 110]

- 0 Means direct address
  1 Means indirect address

Ex:→ IR register contains = 0001XXXXXXXXXXX, ie ADD after fetching and decoding of instruction we find out that is a memory reference instruction for ADD operation.

Hence, DR <- M[AR]
AC <- AC + DR, SC <- 0

↗ Basic Computer Instruction.

Hemadecimal Code I = 0	Hemadecimal Code I = 1	Symbol	Description
0XXX	8XXX	AND	AND Memory word to AC
1XXX	9XXX	ADD	Add Memory word to AC
2XXX	AXXX	LDA	Load memory word to AC
3XXX	BXXX	STA	Store content of AC in me
4XXX	CXXX	BUN	Branch unconditionally
5XXX	DXXX	BSA	Branch and Save return
6XXX	EXXX	ISZ	Increment and Skip if z

B) Register Reference :→ Register reference instructior have zero in the 15th bits 111 in the opcode code part and the register address in the remaining 12 bytes.

15	12 11 … 0
0 1 1 1	Register operation

[opcode = 111, I = 0]

Copyrighted by Codeleap.io
All rights reserved.

Ex:- IR register Contains =
1111100000000000, i.e. INP after fetch and decode cycle we find out that it is an Input/output instruction for inputing character. Hence, Input Character from peripheral device.

Instruction Set Completeness

A set of instructions is said to be complete if the computer includes a sufficient number of instructions in each of the following categories:

functional instructions

- Arithmetic, logical and Shift Instructions

→ Arithmetic, logical and shift instructions provide computational capabilities for processing the type of data the user may wish to employ.

- A set of instructions for moving information to and from memory and processor registers

→ A huge amount of binary information is stored in the Memory unit, but all computations are done in processor registers.

- Instructions which controls the program together with instructions that check status conditions

Program control instructions such as branch instructions are used change the sequence in which the program is executed.

- Input and output Instructions

act as an interface b/w the computer and the user. Programs and data must be transferred into Memory.

Timing and Control

- The timing for all registers is controlled by a master clock generator.
- Clock pulses are applied to all flip-flops and register in the system, including all flip-flops and registers in the control unit.
- Clock pulse do not change the state of a register unless the register is enabled by a control signal.
- The control signals are generated in the control unit and provide control inputs for the Multiplexers in the common bus, control inputs in processor registers and Microoperations for the accumulator.

Copyrighted by Codeleap.io
All rights reserved.

⇒ Control Unit

- The control unit or CU is programmed to control all the activites performed inside the computer system.
- The computer system's Main memory transfers information and instructions to the control unit

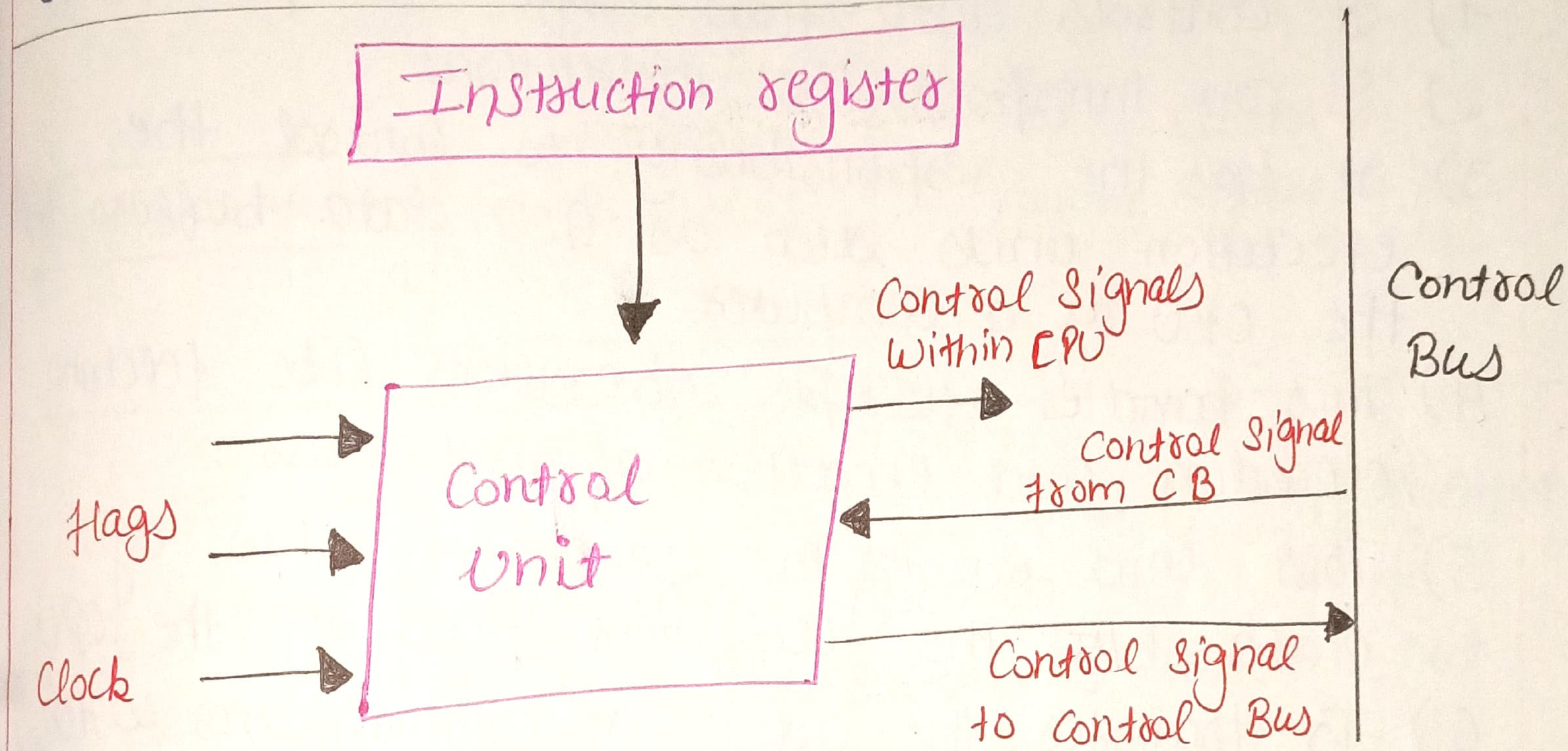
Components of Control Unit

1) Input Components

Input Components include flags, Clock and control signals.

- Clock :→ The Clock enables the control unit to Maintain time. One microinstruction can be performed in one clock pulse.

- **Flags:-** Flags are required to determine the outcome of the preceding ALU operation and the exact status of the processor.
- **Control Signal from the Control Bus:-** The control bus generates control signal such as acknowlegment and interrupts signals. These signals are sent to the control unit.



[Block Diagram of Control Unit]

- **Instruction register:-** It is used to storing the instruction which is currently being executed.
- **Output Components:-** The outputs include control signals inside the processor and the control signals to the control bus.

a) **Control Signals inside the processor:-** These signals are used to transfer the data from one register to another.

Some control signals are also responsible for activating certain ALU function.

· Control signals to the control bus: This includes control signals to the memory and control signals to the input/output devices.

function of Control Unit

1) It controls data flow inside the processor.
2) It can interpret the instructions
3) It has the responsibility to control the execution units such as ALU, data buffers in the CPU of a computer.
4) This handles multiple operations like fetching, decoding and execution of instructions.
5) This unit enables the sequential flow of data b/w the different part of the CPU
6) To transfer the data, it communicates with the input and output devices and controls all the units of the computer.

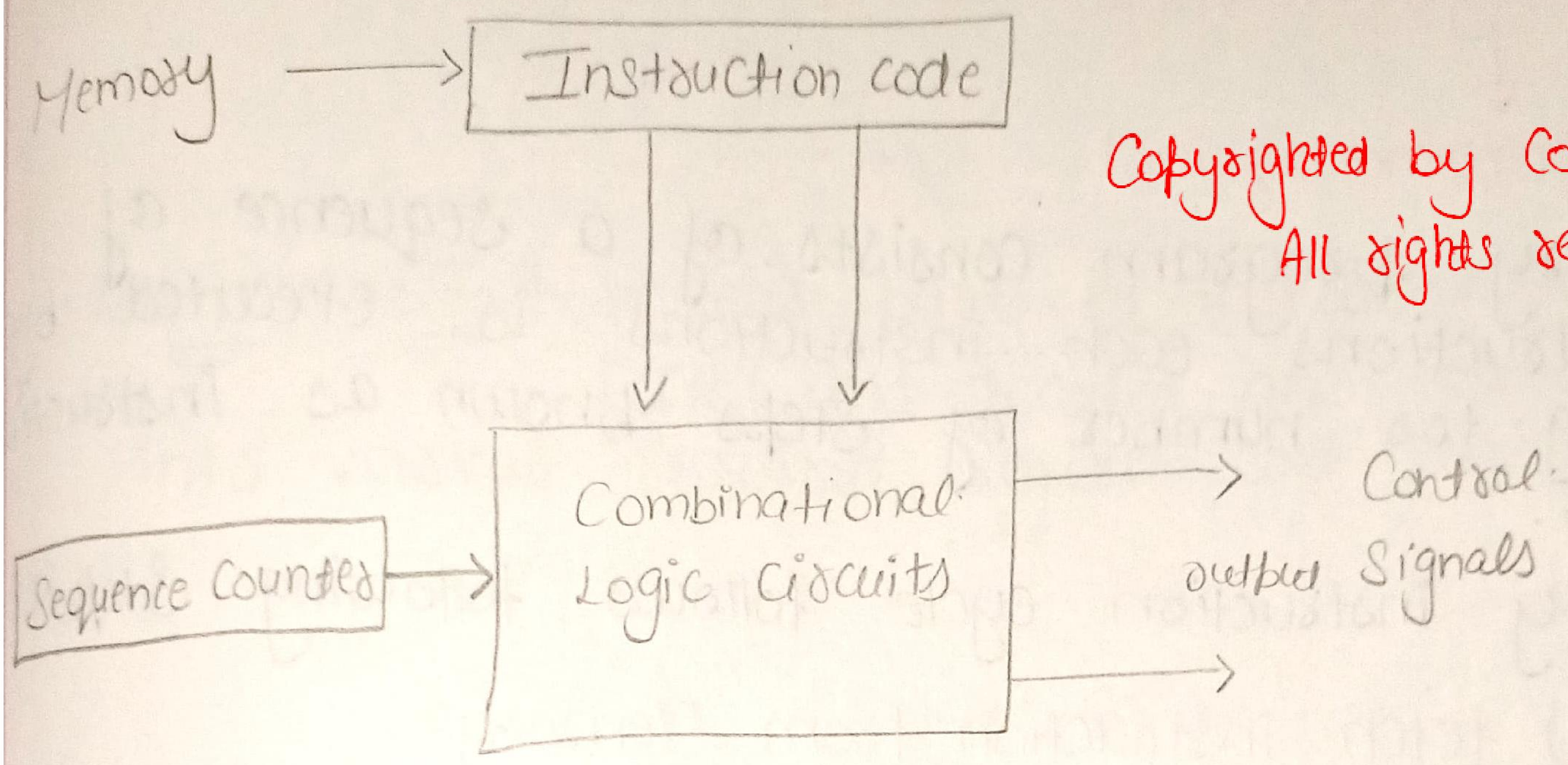
Design of Control Unit / Approaches

Two types of a control unit.

a) Hardwise based.
b) Microprogrammed based (single-level and two level)

a) Hardwise Based :-> It is implemented with the help of physical component which is (gates, flip-flops, decords).

- If design is modified or changed, all the combinational circuits should be modified which is a very difficult task. (wiring) (Any change by wiring)

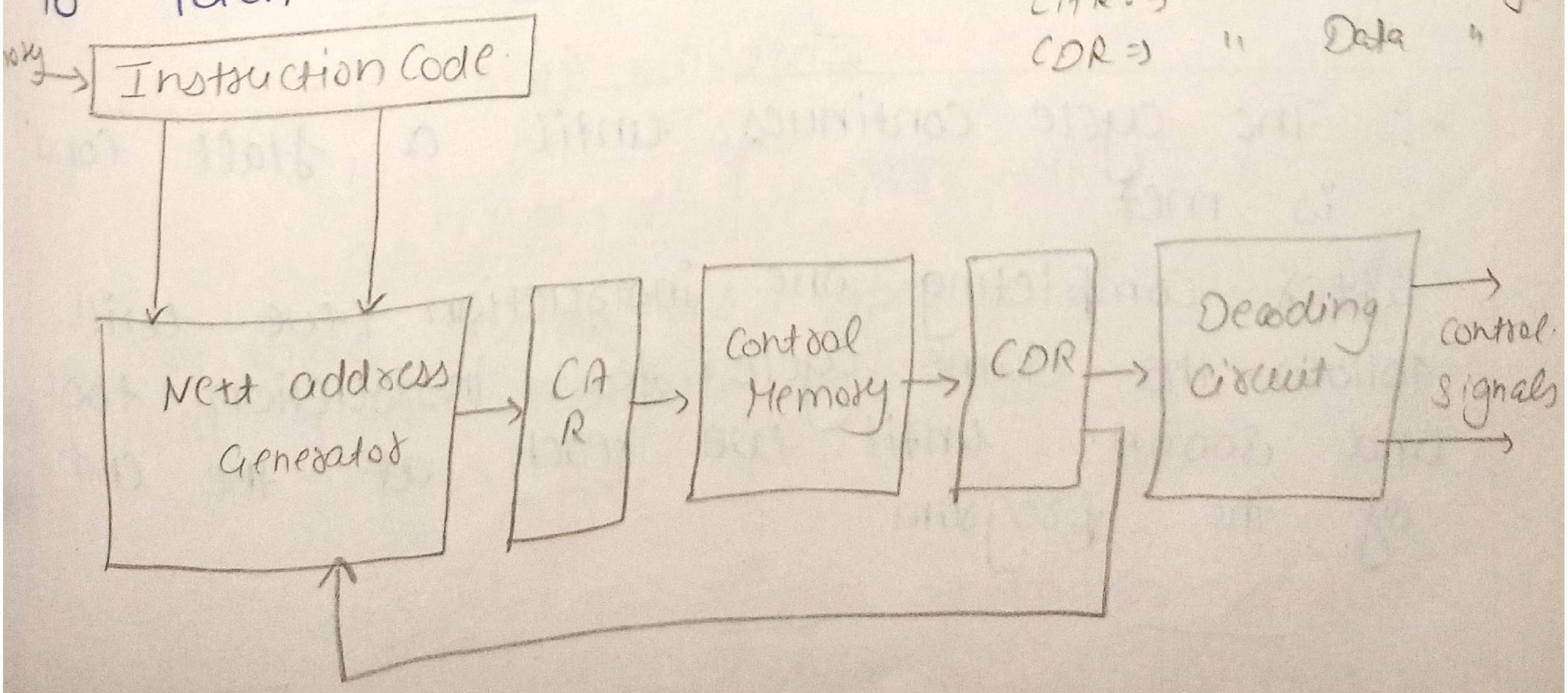


Copyrighted by Codeleap.io
All rights reserved.

b) Microprogrammed Control Unit: It is implemented (Any change by update) by using programming approach.

- A sequence of Micro-operations is carried out by executing a program which consist of micro-instructions.
- Its speed is slow because of the time it takes to fetch microinstructions from control memory.

CAR :- Control Address Regis
CDR => " Data "



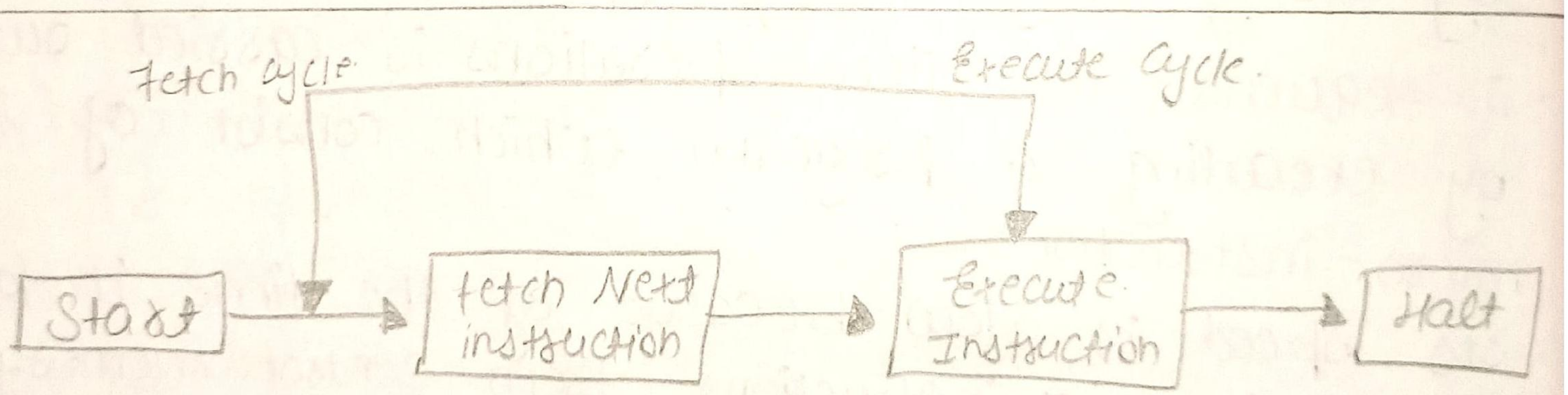
Instruction Cycle

- Set of instruction
- Program
  ↓
  Sequence of inst.

Copyrighted by Codeleap.io
All rights reserved.

- Every program consists of a sequence of instructions each instructions is executed by going too number of steps known as instruction cycle.
- Every instruction cycle follows following steps.

1) fetch instruction from Memory
2) Decode the instruction
3) Read the effective address from memory
4) Execute the instruction.



Instruction cycle.

∴ The cycle continues until a Halt Cond^n is met.

- after completing one instruction, we will follow the same cycle for instruction tool. and sooon. until we reach at the end of the program.

- At time $[T_1]$:→ Memory contents of [AR] is loaded into instruction register [IR] and PC is increment by $\boxed{1}$

T_2 : $D_0, \ldots D_7 \leftarrow IR[12-14]$

$AR \leftarrow IR(0-11)$

$I \leftarrow IR(15)$

- At time $[T_2]$ various parts of instruction register [IR] and decoded and transferred into various required register and variable.

if $D_7 \neq 1$ → It is a Memory reference

a) $I = 0$ [Direct address]

b) $I = 1$ [Indirect address]

$D=0$

15	14 12	11 0
I	opcode	address

opcode = 000 through 110

At Time $[T_3]$ if $D = 1$ → It is a register reference or I/O instruction.

] 16 bits formats

a) $I = 0$, it is register reference.

15	12 11 0
0111	Register oper

opcode = 111, $I = 0$

b) $I = 1$, it is I/O instruction

15	12 11 0
1111	I/O oper

opcode = 111, $I = 1$

- The three instructions types are subdivided into four separate paths.

$D_7' I T_3$: $AR \leftarrow M[AR]$

$D_7' I' T_3$: Nothing

$D_7 I' T_3$: Execute a register-reference.

$D_7 I T_3$: Execute an I/O instruction

Fetch & Decode

→ Fetch the instruction :→ The instruction is fetched from memory address the is stored in PC (Program Counter) and stored in the instruction register IR. At the end of the fetch operation, PC is incremented by 1 and it then points to the next instruction to be executed.

→ Decode the instruction :→ In this step the instruction is decoded by the processor. The processor get any operand if required by the instruction.

→ Execute Instruction :→ In the last phase the processor execute the instruction

---

⇒ Initially, the program counter PC is loaded with the address of first instruction in the program

⇒ SC is cleared to 0, it provides the timing signal T

⇒ SC is incremented after clock pulse, so that the timing signal go through T0, T1, T2, T3 and So on

⇒ Microoperations associated with fetch and decode phase can be represented with following register transfer statement

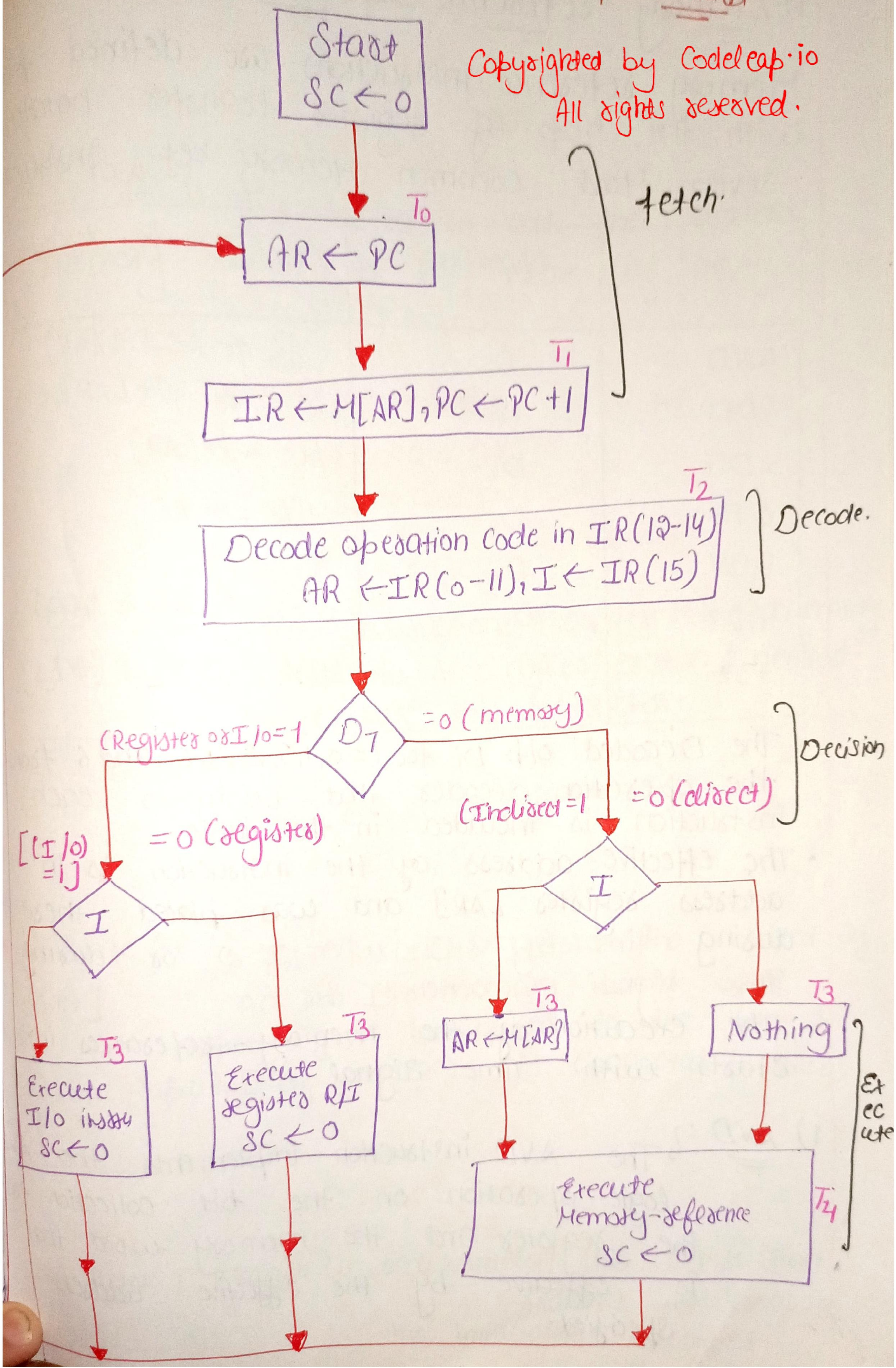
#

$$\left.\begin{array}{l} SC \leftarrow 0 \\ T_0 : AR \leftarrow PC \\ T_1 : IR \leftarrow M[AR],\ PC \leftarrow PC+1 \end{array}\right\} \text{fetch from Memory}$$

→ At time (T_0), the contents of [PC] is transfered into address register (AR).

Flow Chart

Copyrighted by Codeleap.io
All rights reserved.



Memory - reference Instruction

Memory reference instructions are defined precisely with the help of register transfer notatation

· Seven Most common Memory ref. Instructions a

Symbol	Operation Decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR], E \leftarrow Cou$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC \leftarrow AR + 1$
ISZ	D_6	$M[AR] \leftarrow M[AR] + 1$

- The Decoded o/p D_i for $i = 0, 1, 2, 3, 4, 5$ and 6 from the operation decoder that belong to each instruction is included in the table.
- The effective address of the instruction is in the address register [AR] and was placed there during time signal T_2 when $I = 0$ or during time signal T_3 when $I = 1$.
- The execution of the memory - references instruc Starts with Time signal T_4

1) AND :- The AND instruction implements the AN logic operation on the bit collection fo the register and the memory word that is ~~effective~~ by the effective address specified.

. Microoperations

$D_0 T_4 : DR \leftarrow M[AR]$

$D_0 T_5 : AC \leftarrow AC \wedge DR \quad SC \leftarrow 0$

2) ADD To AC :- This instruction add the content of memory word to the value of AC.

. Microoperation

Copyrighted by Codeleap.io
All rights reserved.

$D_1 T_4 : DR \leftarrow M[AR]$

$D_1 T_5 : AC \leftarrow AC + DR$

$E \leftarrow Cout$

$SC \leftarrow 0$

3) Load to AC :- This instruction transfer the memory [LDA] word to AC microoperation needed to execute this instruction.

. Microoperation

$D_2 T_4 : DR \leftarrow M[AR]$

$D_2 T_5 : AC \leftarrow DR, SC \leftarrow 0$

4) Store AC :- This instruction stores the content of [STA] AC into the memory word. Since output of AC is applied to the bus & data input of memory is connected to bus.

. Microoperation

$D_3 T_4 : M[AR] \leftarrow AC, SC \leftarrow 0$

5) BUN [Branch Unconditionally] This instruction transfer the Program to the instruction specified.

$D_4 T_4$:- $PC \leftarrow AR$, $SC \leftarrow 0$

- PC hold the address of the next instruction to be executed.

6. BSA → Branch and Save return Address

These instructions can branch a part of the program. When this instruction is performed, BSA will store the address of the next instruction from PC into a memory location that is determined by the effective address.

$D_5 T_4$: $M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$

$D_5 T_5$: $PC \leftarrow AR$, $SC \leftarrow 0$

7) ISZ :- The increment if zero (ISZ) instruction increments the word determined by effective address.

- If the incremented cost is zero, thus PC is incremented by 1
- A negative value is saved in the memory word through the programmer.
- It can influence the zero value after getting incremented repeatedly. Thus, the PC is incremented and the next instruction is skipped.

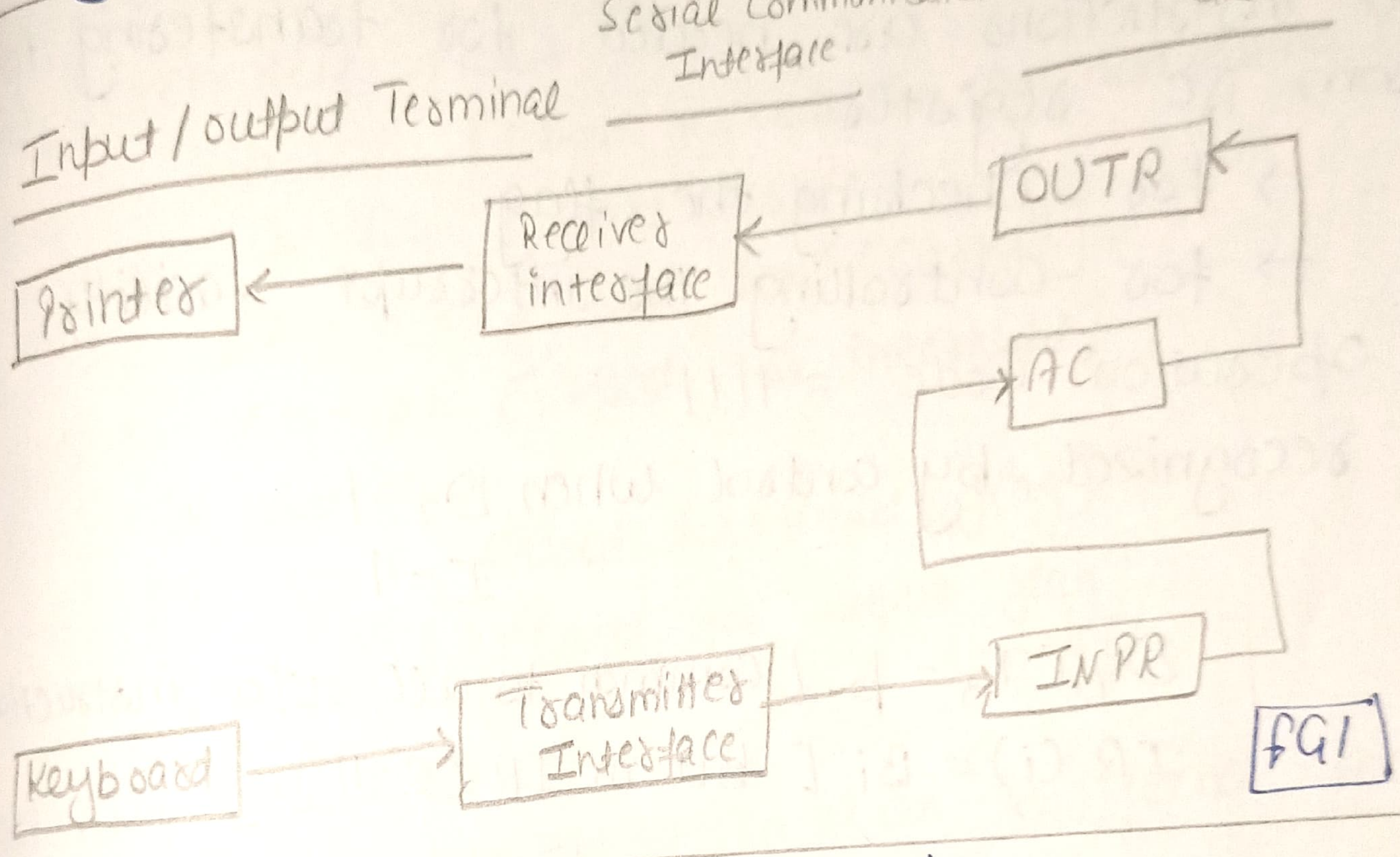
$D_6 T_4$: $DR \leftarrow M[AR]$

$D_6 T_5$: $DR \leftarrow DR + 1$

$D_6 T_6$: $M[AR] \leftarrow DR$, if $(DR = 0)$ then $(PC \leftarrow PC+1)$, $SC \leftarrow 0$

Input - Output and Interrupt

Instructions and data from the user come to the computer through some input device and the computational results are transmitted to the user through some output device. We have many input and output devices



I/O configuration

INPR → input register - consists eight bits and holds an alphanumeric I/p information.

FGI → 1- bit I/p flag is a control flip - flop
= 1, when new information is available in the I/p device
= 0, when the information is accepted by the computer.

- Initially FGI is cleared to 0.

OUTR :→ Output register - consists eight - bits

FGO → 1- bit o/p flag is a control flip-flop.

0 → do not load new thing
1 → information from AC is transferred in parallel to OUTR.

- initally FGO = 1, information from AC is transfered in parallel to OUTR and FGO is cleared to 0.

1) Input - Output Instructions

I/O instructions are needed for transfering to and from AC register
→ for checking the flag
→ for controlling the interrupt facility

operation Code = 1111

recognized by control when $D_7 = 1$
$I = 1$

[$D_7 I T_3 = p$ [Common to all I/O instruction]
$IR(i) = B_i$ [bit in IR (6-11)]]

← Micro operation

2) Program interrupt

The process of commn described b/w keyboard and printer was programmed control transfer where the computer keeps checking the flag bit, and when it finds it set, it initiates an information transfer.

when IEN = 0, the flag cannot interrupt the computer
when IEN = 1, the computer can be interrupted.

Copyrighted by Codeleap.io
All rights reserved.

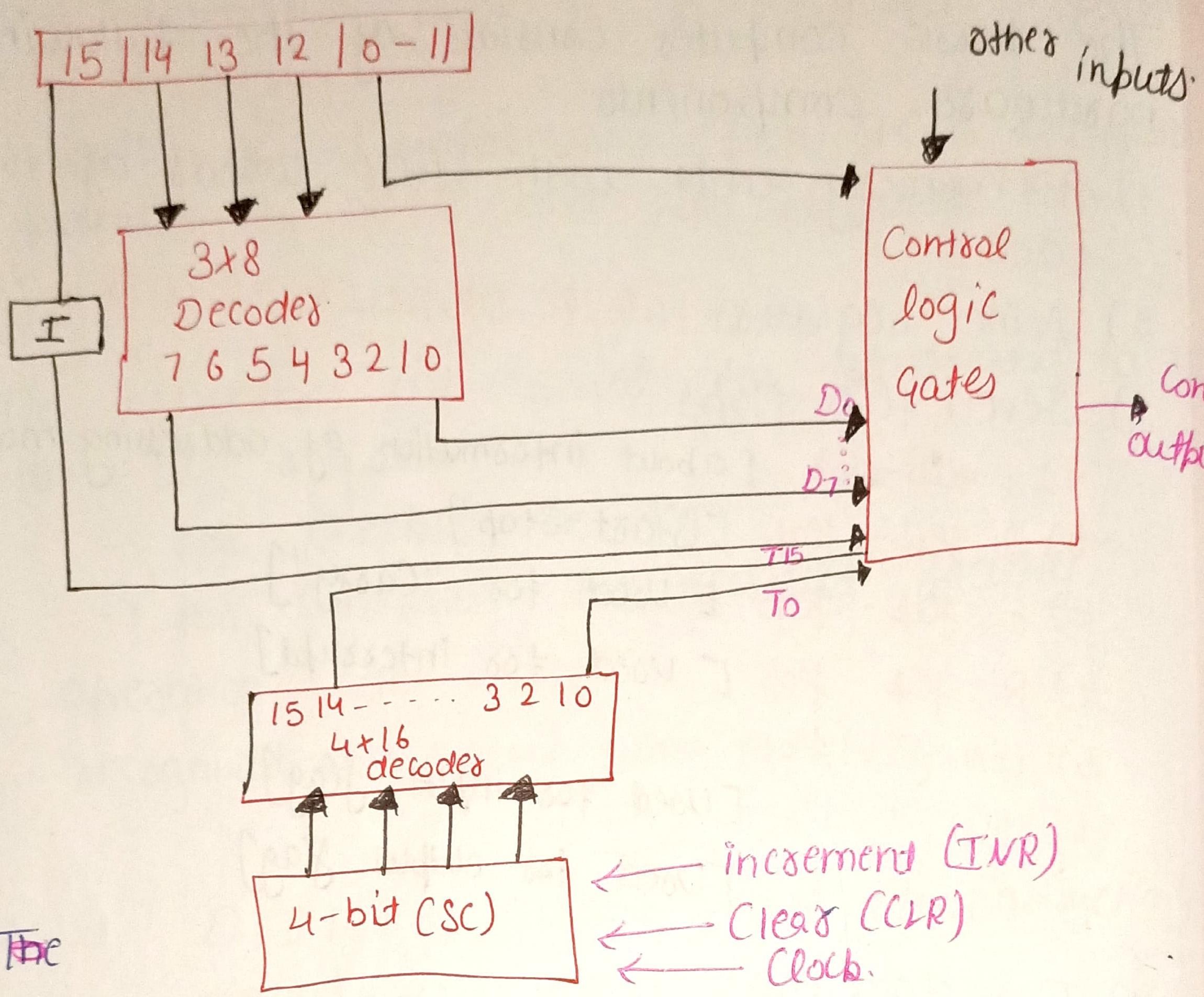
Design of Basic Computer

The basic computer consists of the following hardware components.

1) A memory unit with 4096 words of 16 bit each
2) Nine registers [AR, PC, DR, AC, IR, TR, OUTR, INPR, SC]
3) Seven flip-flops
   1) I flip-flop [about information of addressing mode]
   2) S flip-flop [Start-Stop]
   3) E flip-flop [Used for "Carry"]
   4) R flip-flop [used for interrupt]
   5) IEN flip-flop
   6) FGI " " [Used for input flag]
   7) FGO " " [Used for output flag]
4) Two decoders
   1) 3×8 operation decoder :→ Used to decode the output of IR
   2) 4×6 timing decoder :→ Used to decode the time signal generated by sequence counter.
5) A 16-bit common bus

The basic computer has eight registers, a memory unit, and a control unit and there is a requirement of a path to transfer information from one register to another and b/w memory and register. So to provide this path we use a common bus.

7) Control logic gates



The

- The inputs for the control logic gates comes from
  1. input from I flip-flops
  2. Input from the two decoders
  3. Input from 0-11 bits of IR
  4. Other inputs to the control logic gate.
     1) AC bits 0-15, to check if AC = 0 and to detch the sign bit in AC (15)
     2) DR (Data register) bits 0-15, to check if DR = 0 and check the values of Seven flip-flop.
- The output of the control logic circuit
  1) Signals to control the read and write inputs of Memory.

2. Signals to control the inputs of the eight registers.

3. Control the AC adder and logic circuit.

4. Signal to set, clear or complement of f-f.

Control of (Single) flip-flops

The control gates for the seven flip-flops can be determined in a similar manner.

for eg: for IEN flip-flop.

$PB_7 : IEN \leftarrow 1$

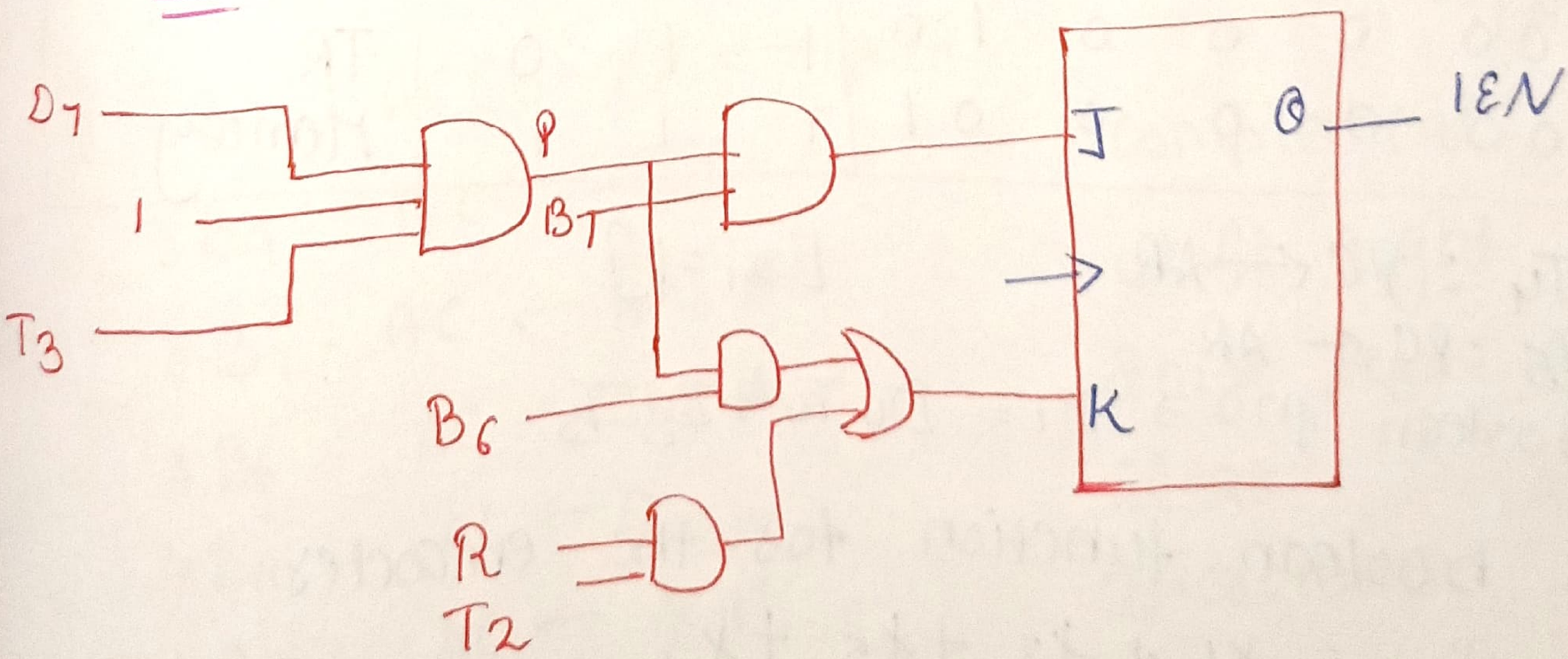
$PB_6 : IEN \leftarrow 0$

When $P = D_7 I T_3$

- And B7 B6 are bits 7 and 6 of IR
- At the end of interrupt cycle, IEN is cleared to 0.

$RT_2 : IEN \leftarrow 0$

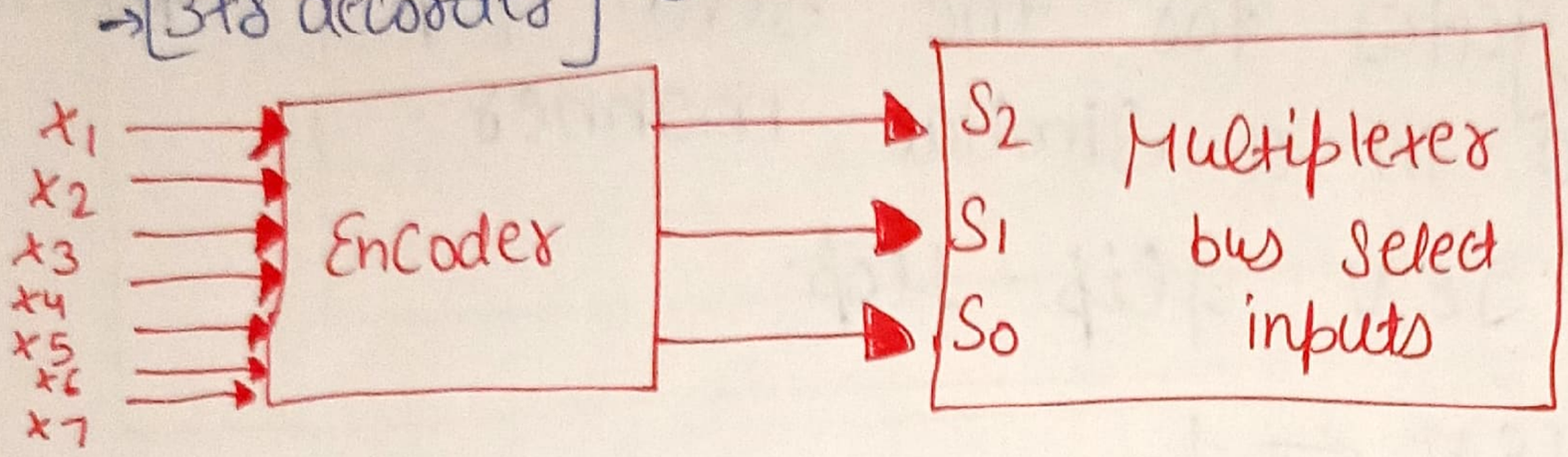
→ Control logic gate for IEN



Control of Common Bus

- The 16-bit common Bus is controlled by the selection inputs S_2, S_1 and S_0.
- The three selection variable S_0, S_1 and S_2 are used to select the corresponding register or memory.

- The block Diagram of the encoder used.
  → [3×8 decorder]



- The input and output of the above encoder are as shown below in the table.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	Selected register
0	0	0	0	0	0	0	0	0	0	none
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

- $D_4 T_4$: $PC \leftarrow AR$ $[x_1 = 1]$
- $D_5 T_5$: $PC \leftarrow AR$
  ∴ Boolean fn → $x_1 = D_4 T_4 + D_5 T_5$

- The boolean function for the encoder.

$$S_0 = x_1 + x_3 + x_5 + x_7$$

$$S_1 = x_2 + x_3 + x_6 + x_7$$

$$S_2 = x_4 + x_5 + x_6 + x_7.$$

Design of Accumulator logic

accumulator is a register for a short-term, intermediate storage of arithmetic and logic data a computers CPU

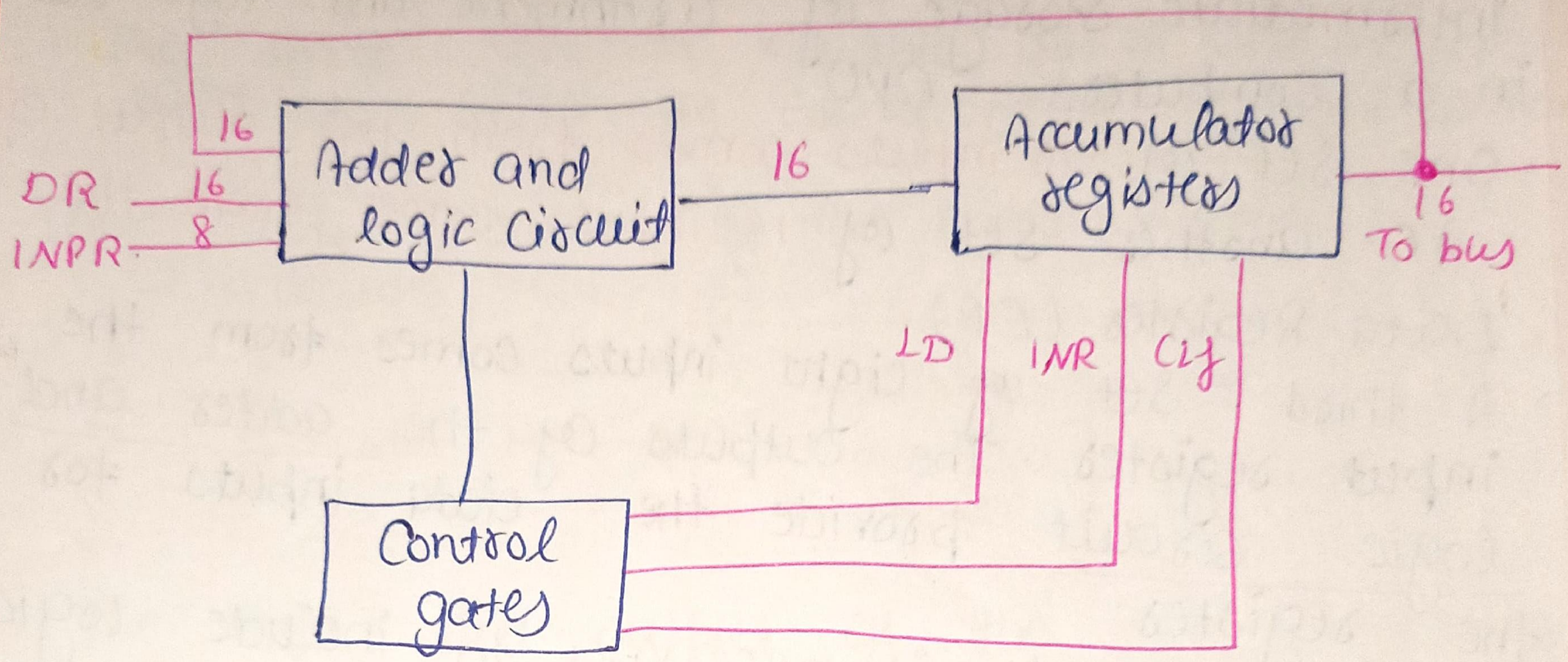
one set of 16 inputs comes from the output of AC. Another set of 16 inputs comes from the Data Register (DR)

A third set of eight inputs comes from the input register. The outputs of the adder and logic circuit provide the data inputs for the register.

In addition, it is necessary to include logic gates for controlling the LD, INR and CLR in the register and for controlling the operation the adder and logic circuit.

$D_0 T_5 : AC \leftarrow AC \wedge DR$	AND with DR.
$D_1 T_5 : AC \leftarrow AC + DR$	Add with DR.
$D_2 T_5 : AC \leftarrow DR$	Transfer from DR
$PB_{11} : AC(0-7) \leftarrow INPR$	" " INPR.
$rB_9 : AC \leftarrow \overline{AC}$	complement
$rB_7 : AC \leftarrow shr\ AC$	Shift right
$rB_6 : AC \leftarrow shl\ AC$	Shift left
$rB_{11} : AC \leftarrow 0$	Clear
$rB_5 : AC \leftarrow AC+1$	Increment

- Circuits associated with AC



Copyrighted by Codeleap.io
All rights reserved.

Register Transfer Language

It is the symbolic representation of notations used to specify the sequence of Micro-operations.

- information Transfer from one register to another register is called Register Transfer.
- Registers are designated by capital letters, sometimes followed by numbers (eg - A, R1, R13, IR).
- RTL is used to describe Data flow at register level.

Ex:→ $R_1 \leftarrow R_2$

This statement denotes a transfered of the content of register R_2 in to register R1

- Here content of R1 is replaced by content of R2
- Content of R2 is not affected by this transfer

Basic Symbol of Register Transfer Memory

Symbol	Description	Examples
Letters & Numerals	Denotes a register	MAR, R_2
Parenthesis ()	Denotes a part of a register	$R_2(0-7)$, $R_2(L)$
Arrow ←	Denotes Transfer of information	$R_2 \leftarrow R_1$
Comma ,	Separates two	$R_2 \leftarrow R_1$, $R_2 \leftarrow R_3$

- The operation performed on the data stored in registers are called micro-operations

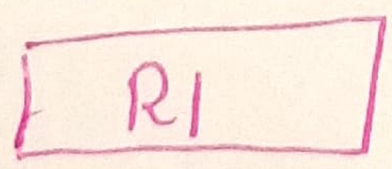
$T=1 : R2 \leftarrow R1 , R1 \leftarrow R2$

Control function

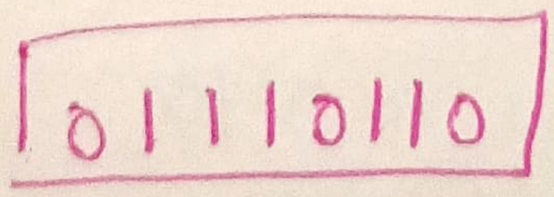
- Registers (R1, R2) are denoted by capital letter and numerals.
- Parentheses ( ) are used to denote a part of a register.
- The arrow ($\leftarrow$) denotes a transfer of information and the direction of transfer.
- A comma (,) is used to separate two or more operation that are executed at the same time.
- The control condition 'T=1' is terminated with a colon. It symbolizes the requirement that the transfer operation be executed by the hardware only if $T=1$

There are various methods of RTL

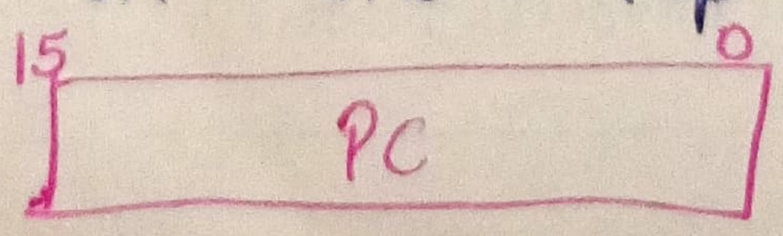
1) General way of representing a register is by the name of the register enclosed in a rectangular box



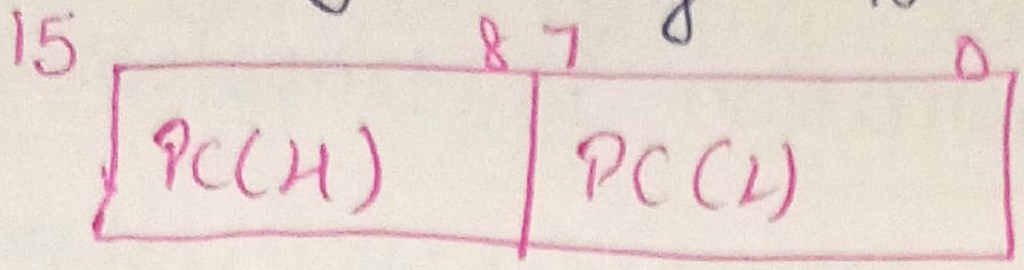
2). Register is numbered in a sequence of 0 to (n-1).



3) The numbering of bits in a register can be marked on the top of the box.



4) A 16-bit register PC is divided into 2 parts- bits (0 to 7) are assigned with lower byte of 16-bit address and bits (8 to 15) are assigned with higher bytes of 16-bit address-



Register transfer : Means the availability of hardware Logic circuits that can perform a stated micro-operations and transfer the result of the operation to the same or another register

⇒ Some of the commonly used register are

1) Accumulator :→ This is the most common register, used to store data taken out from the Memory.

2) General Purpose Registers :→ This is used to store data intermediate results during program execution.

3) Special Purpose Registers: Users do not access these registers. These registers are for computer system

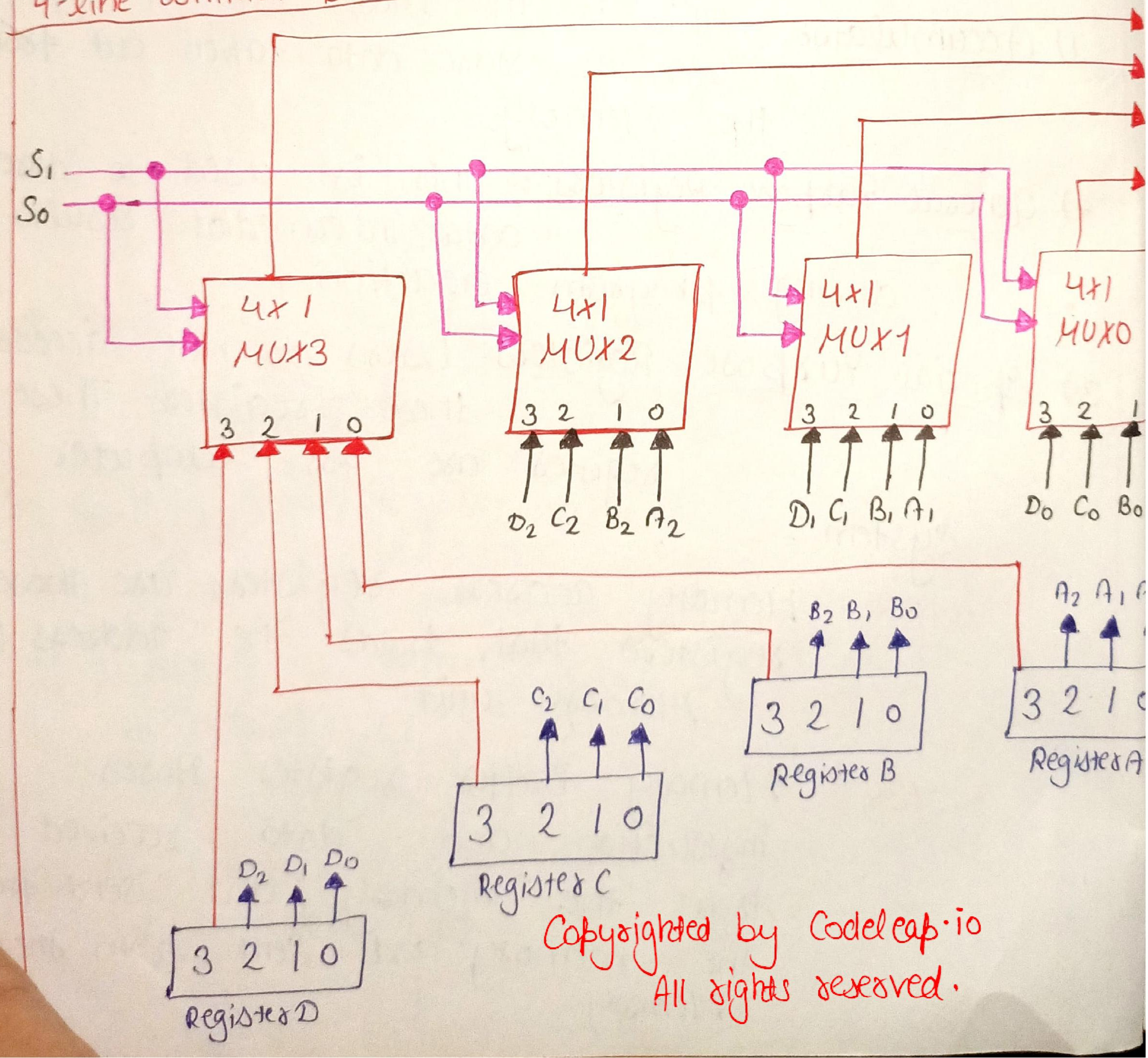
- MAR :→ Memory address registers are those registers that holds the address for Memory unit.

- MBR :→ Memory Buffer register stores instruction and data received from the memory and send from the memory and send from the Memory.

Bus and Memory Transfers

- A bus structure consists of a set of common lines, one for each bit of a register, through which binary information is transferred one at a time.
- Control signal determine which register is selected by the bus during each particular register transfer.
- One way of constructing a common bus system is with multiplexers. The Multiplexers select the source register whose binary information is then placed on the bus.

4-line common Bus.



Copyrighted by Codeleap.io
All rights reserved.

- The following block Diagram shows a Bus System for four registers.
- It is constructed with the help of four 4×1 Multiplexers each having four data inputs (0 through 3) and two Selection inputs (S1 and S2)
- These inputs are connected to the selection inputs of all multiplexers.
- When both of the Select line are at low logic i.e S1S0 = 00, the 0 data inputs of all four Multiplexers are selected and applied to the outputs that forms the bus. Thus, causes the bus line to recieve the content of register A. Since the output of this register are connected to the 0 data inputs of the Multiplexers.
- Illy when S1S0 = 01, the register B is selected, and the bus lines will receive the content provided by register B.

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

In general :-> • a bus System will multiplex k registers of n bits each to produce an n-line common bus.

- The no. of Multiplexers needed to construct the bus in equal to n, the no. of bits in each register

- The size of each multiplexer must be $\boxed{k \times 1}$ since it multiplexes k data lines.
- The symbolic statement for a bus transfer:

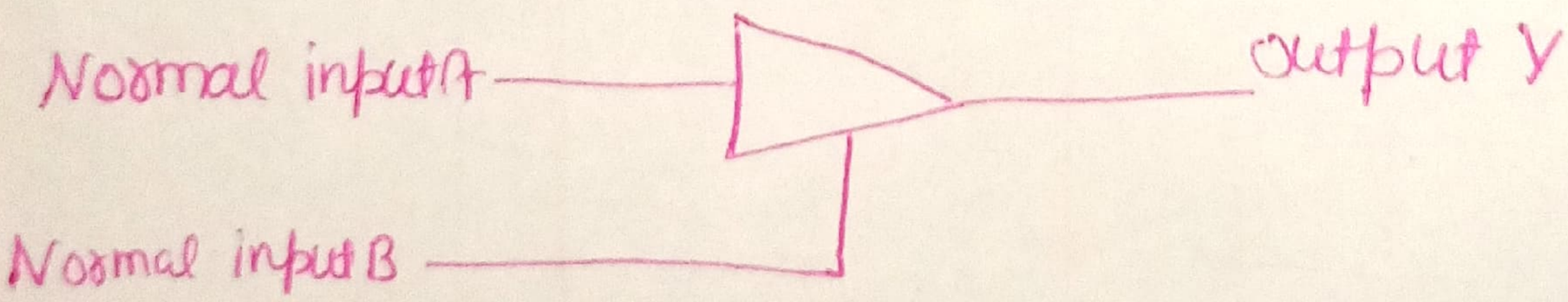
$$BUS \leftarrow C, R1 \leftarrow BUS.$$

- To show a direct transfer like $\boxed{R1 \leftarrow C}$

---

3 - State Bus Buffers

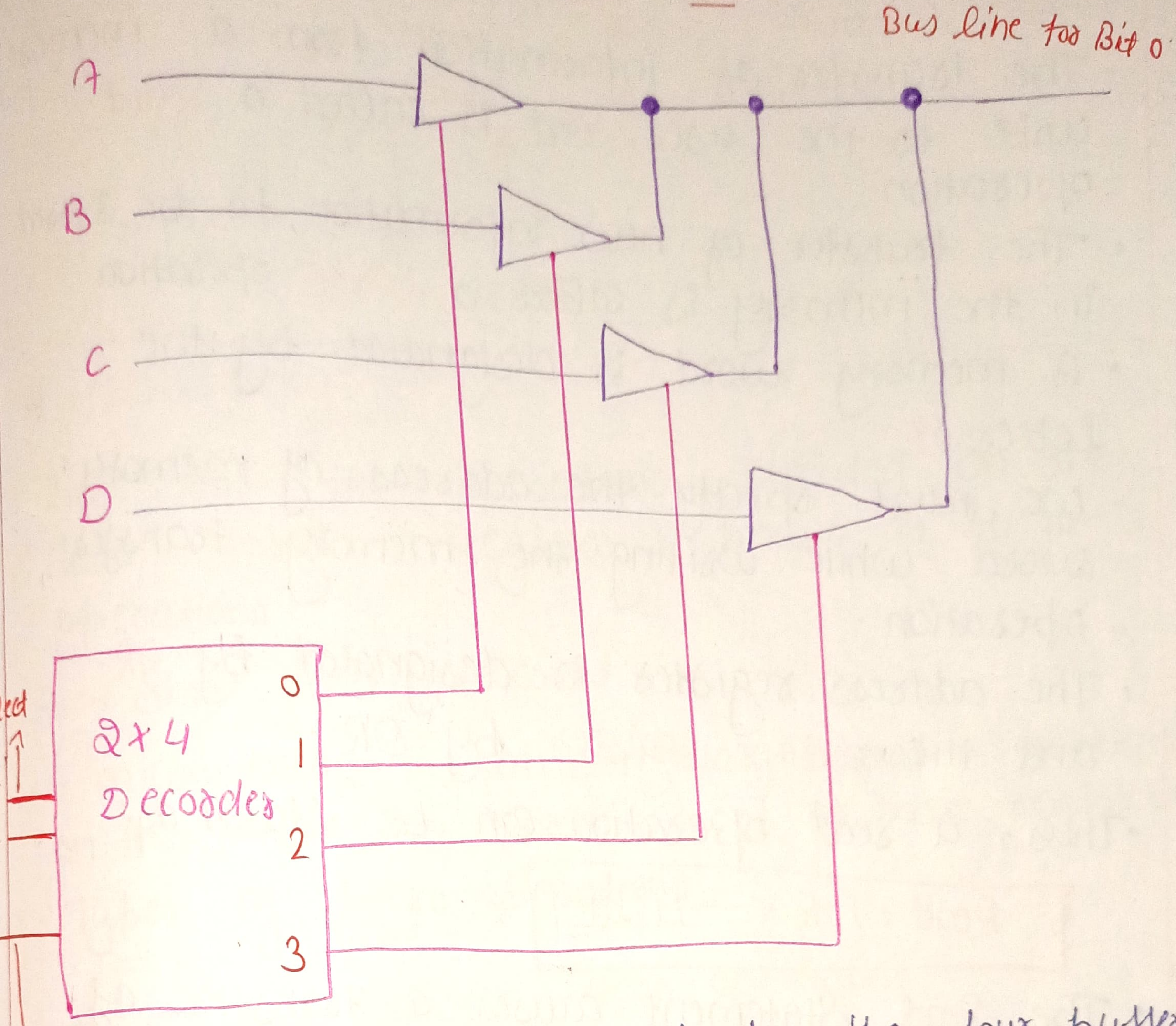
- A bus system can also be constructed using three - state gates instead of Multiplexers.
- The Three State gates can be considered as a digital circuit that has three gates, two of which are signals equivalent to logic 1 and 0 as in a conventional gate. However, the third gate exhibits a high - impedance state.
- The Mostly commonly used three states gates in case of the bus system is a buffer gate
- The graphical symbol of a three - state buffer gate can be represented as:



Output $Y = A$ if $C = 1$

High - Impedance if $C = 0$.

→ Bus line with three state buffer.



- The outputs generated by the four buffers are connected to form a single bus line.
- Only one buffer can be in active state at a given point of time.
- The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.
- A $\boxed{2 \times 4}$ decoder ensures that no more than one control input is active at any given point of time.

Memory Transfer

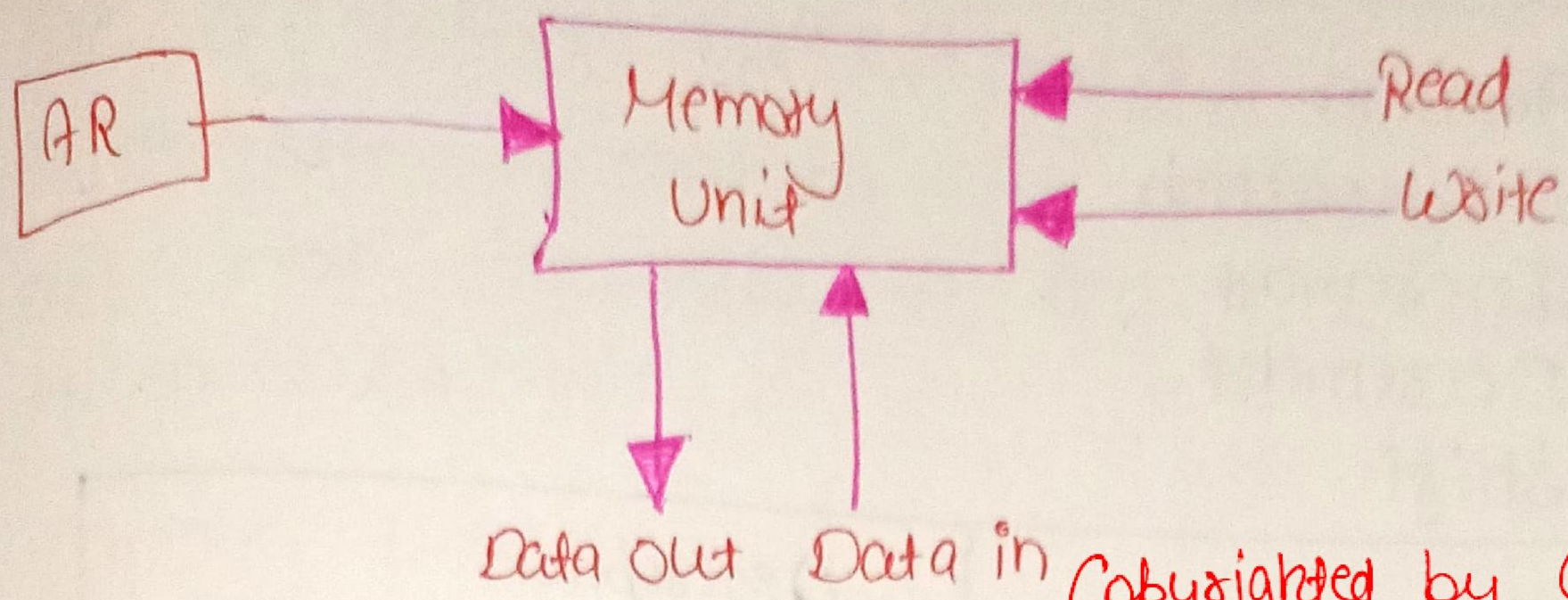
- The transfer of information from a memory unit to the user end is called a Read operation.
- The transfer of new information to be stored in the memory is called a Write operation.
- A memory word is designated by the letter M.
- We must specify the address of memory word while writing the memory transfer operation.
- The address register is designated by AR. and the data register by DR.
- Thus, a read operation can be stated as:

$$\boxed{\text{Read} : DR \leftarrow M[AR]}$$

- The Read Statement causes a transfer of information into the data register (DR). from the memory word (M) selected by the address register (AR)
-

$$\boxed{\text{Write} : M[AR] \leftarrow R1}$$

- The Write Statement causes a transfer of information from register R1 into the memory word [M] selected by address register (AR)



Copyrighted by Codeleap.io
All rights reserved.

Microoperations

- These are the operations executed on the data stored in registers is known as Micro-operations.

Ex:→ Shift, load, inc, count.

- A microoperation is an elementary operation performed on the information stored in one or more registers.
- Types of Micro-operations

1) Register Transfer Micro-operations.
[Transfer binary information from one register to another, R2 ← R1]

2) Arithmetic Micro-operations

3) Logical Micro-operations

4) Shift Micro-operation

1) Arithmetic Micro-operations:→ Perform arithmetic operation on numeric data stored in registers.

- The basic arithmetic micro-operation are.
  - Addition [$R_3 \leftarrow R_1 + R_2$]
  - Substraction
  - Increment
  - Decrement
  - Shift

Symbolic representation	Description
$R3 \leftarrow R1 + R2$	The contents of R1 Plus R2 are transferred to R3
$R3 \leftarrow R1 - R2$	The contents of R1 Minus R2 are transferred to R3
$R2 \leftarrow R2'$	Complement the content of R2 (1's complement)
$R2 \leftarrow R2' + 1$	2's Complement the Contents of R2
$R3 \leftarrow R1 + R2' + 1$	R1 plus the 2's Compl. of R2 [Substra]
$R1 \leftarrow R1 + 1$	Increment the contents of R1 by one.
$R1 \leftarrow R1 - 1$	Decrement the contents of R1 by One.

- The increment and decrement micro-operations are symbolized by '+1' and '?1'.
- Arithmetic operations like Multiply and divide are not included in the basic set of micro-operation

Binary adder

To implement the add microoperation with H/W we need

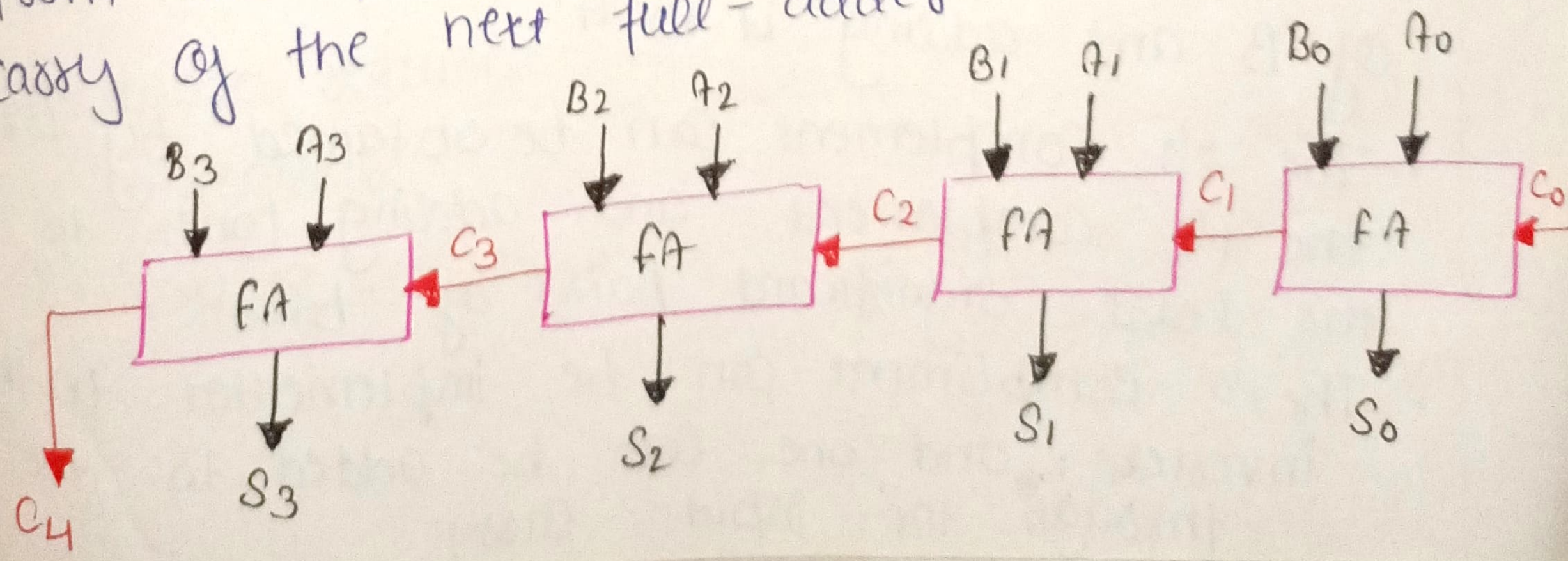
→ register that hold the data.

→ Digital component that performs the arithmetic addition.

Full adder :→ The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full-adder [add two bits and one carry]

Binary adder :→ The digital circuit that generates the arithmetic sum of two binary numbers of any length is called binary adder.

- The binary adder is constructed with full-adder connected to cascade, with the output carry from one full-adder connected to the input carry of the next full-adder.

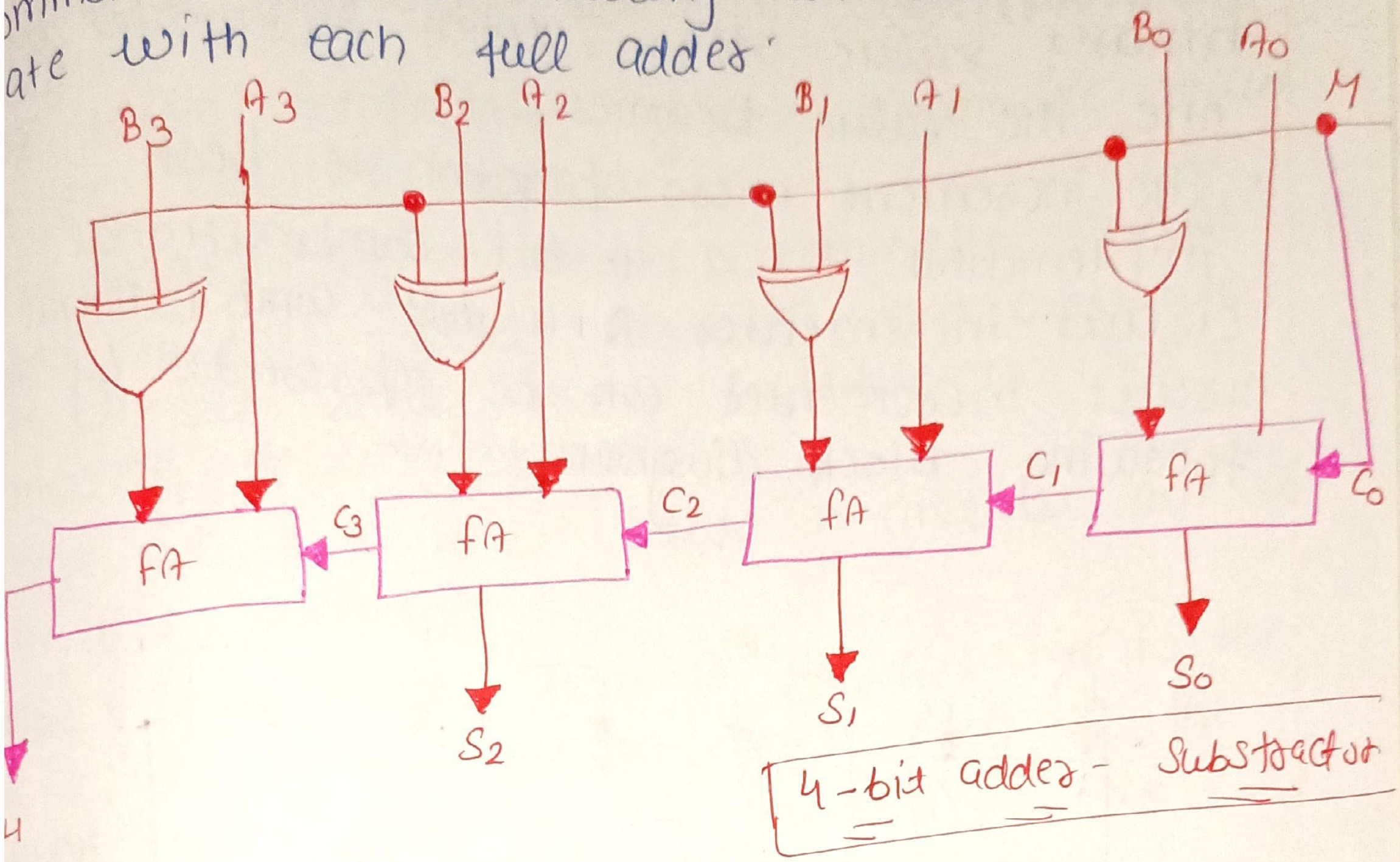


- (A) → Augend bits
- (B) → addend bits
- The carry inputs Starts from C_0 to C_3 connected in a chain through the full-adders. C_4 is a resultant output carry generated by the last full-adder circuit
- The output carry from each full-adder is connected to the input carry of the next-high-order full-adder.
- The sum outputs (S_0 to S_3) generates the required arithmetic sum of augend and addend bits.

Binary Adder - Subtractor

→ Hardware requirement
- registers to hold data.
- Digital component to perform arithmetic addition (full adder)
- exclusive-or gate with each full adder
- The substraction of binary numbers can be done most convienty by means of complement
- A - B be cond by taking the 2's complement of B and adding it to A.
- The 2's complement can be obtained by taking the 1's complement and adding one to the least significant pairs of bits.
- The 1's complimend can be implemented with inverters, and one can be added to the su through the input carry.

The arithmetic micro-operations like addition and substraction can be combined into one common circuit be including an exclusive-OR gate with each full adder.



4-bit adder - Substractor

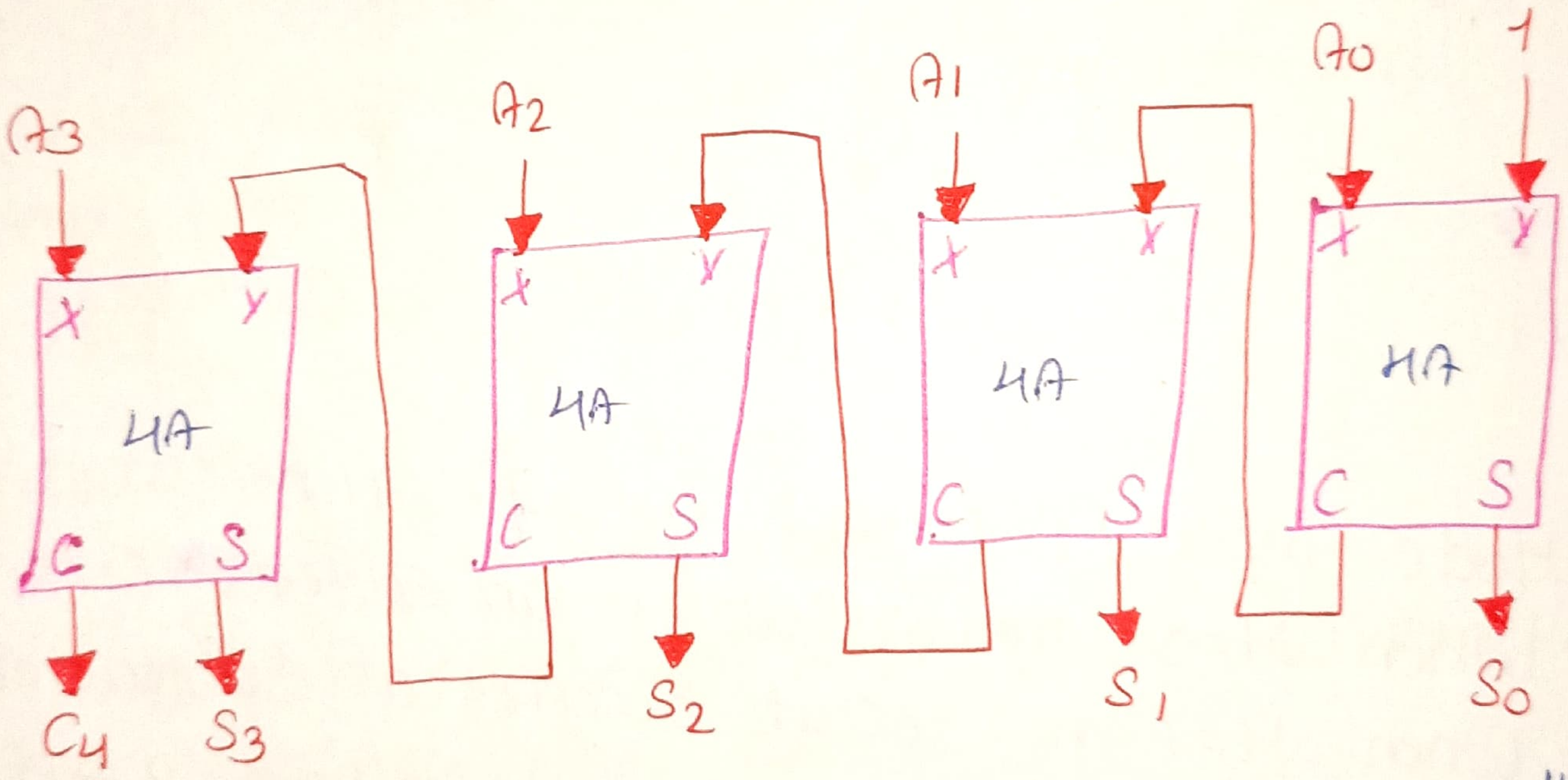
Mode input M control the operation.
When M=0 the circuit is an adder.
When M=1 the circuit becomes a Subtractor.
When M is at low logic, we have $B \oplus 0 = B$.
When M is at high level, we have $B \oplus 1 = B'$
and $C0 = 1$

Binary incrementor

- Hardware required.
  - → Register.
  - → Half-adder.

Copyrighted by Codeleap.io
All rights reserved.

- The increment micro-operation add one binary value to the value of binary variables stored in a register for instance - 4-bit register has a binary value 0110, when incremented by one the value becomes 0111.
- The increment micro-operation is best implemented by a 4-bit combinational circuit incrementer. A 4-bit combinational circuit incrementer can be represented by the following block diagram.



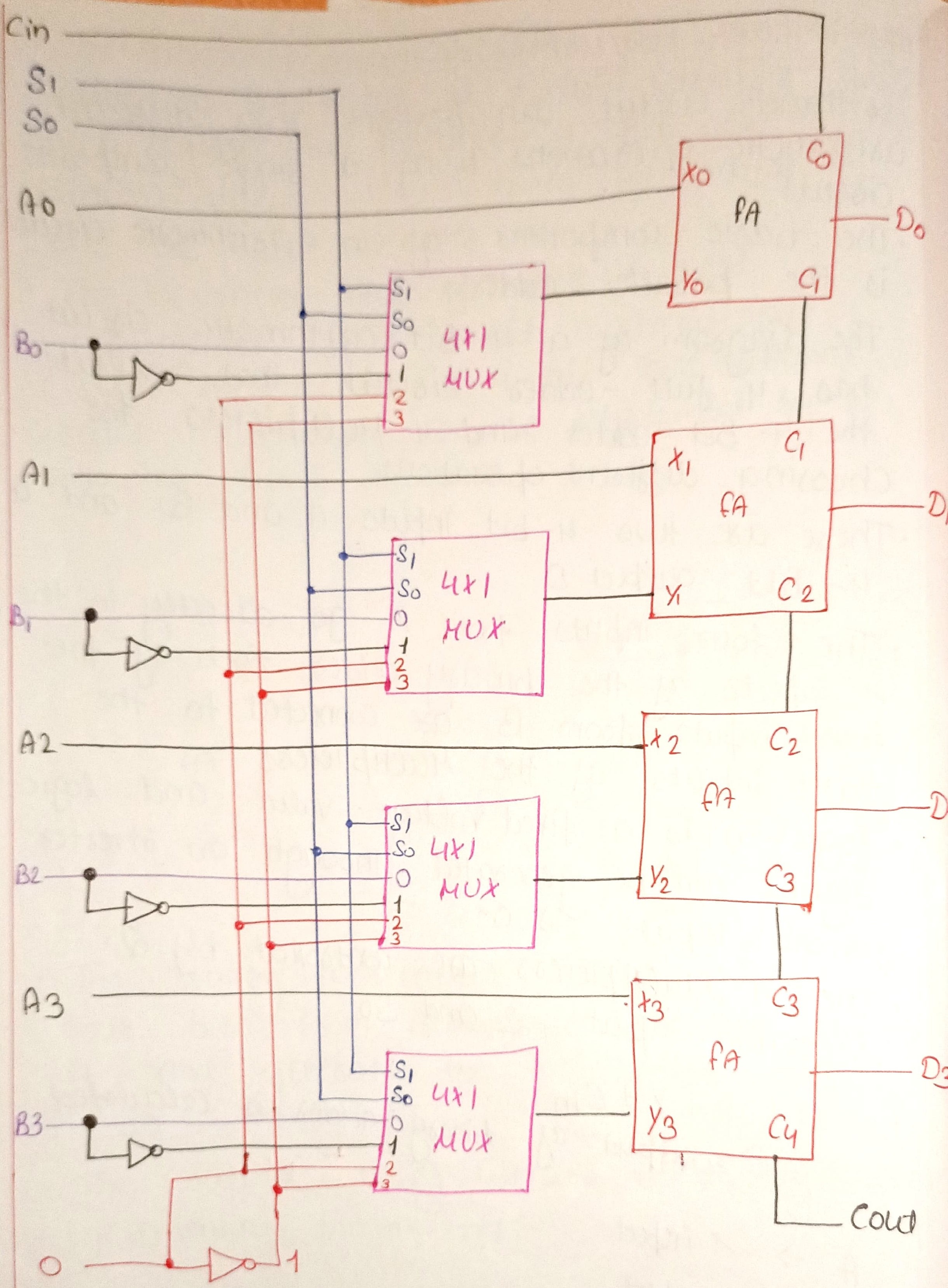
- The binary incrementer circuit receives the four bits from A0 through A3, add on to it, and generates the incremented output in S0 through S3
- The output carry C4 will be 1 only after incrementing binary 1111.

Arithmetic Circuit

Arithmetic circuit can perform seves different arithmetic operations using a single composite circuit

- The basic component of an arithmetic circuit is the parallel adder.
- The diagram of a 4-bit arithmetic circuit has 4 full-adder circuit that constitute the 4-bit adder and 4 Multipleters for choosing different operations.
- There are two 4-bit inputs A and B and a 4-bit output D
- The four inputs from A go directey to the x inputs of the binary adder. Each of the four inputs from B are connected to the data inputs of the Multipleters
- Logic-0 is a fixed voltage value and logic-1 signal can be generated through an inverter whose input is 0.
- The 4 Multipleters are controlled by 2 Selection inputs, S_1 and S_0
- $D = A + Y + C_{in}$
  → output of binary adder is calculated

A → x input
B → y input
Cin → input carry can be 1 or 0.



Hardware implementation

logic Micro-operation

The logic Micro-operation specify binary operations for string of bits stored in registers

eg:→ $P: R1 \leftarrow R1 \oplus R2$

It specifies a logic micro-operation to be executed individual bits of the registers provided that the control variable $P=1$.

Special Symbols

$\vee$ → or Microoperation

$\wedge$ → AND Microoperation

- The complement microoperation is the same as the 1's complement and uses a bar on top of the symbol that denotes the register name.
  $[\overline{A}, A']$

Symbol has two Meaning

→ When + occurs in a Microoperation, it will denote an arithmetic plus

→ When it occurs in a control (or Boolean) function it will be denote OR operation.

→ $P+Q: R1 \leftarrow R2 + R3, R4 \leftarrow R5 \vee R6$

↳ is an OR operation b/w two binary variable of a control function.

- The + b/w R2 and R3 specifies an add Micro-operation.
- The OR Microoperation is designated by the symbol $\vee$ b/w registers R5 and R6.

list of logic Microoperations

- There are 16 different logic operations that can be performed with two binary variables
- They can be determined from all possible truth table obtained with two binary variable.

x	y	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

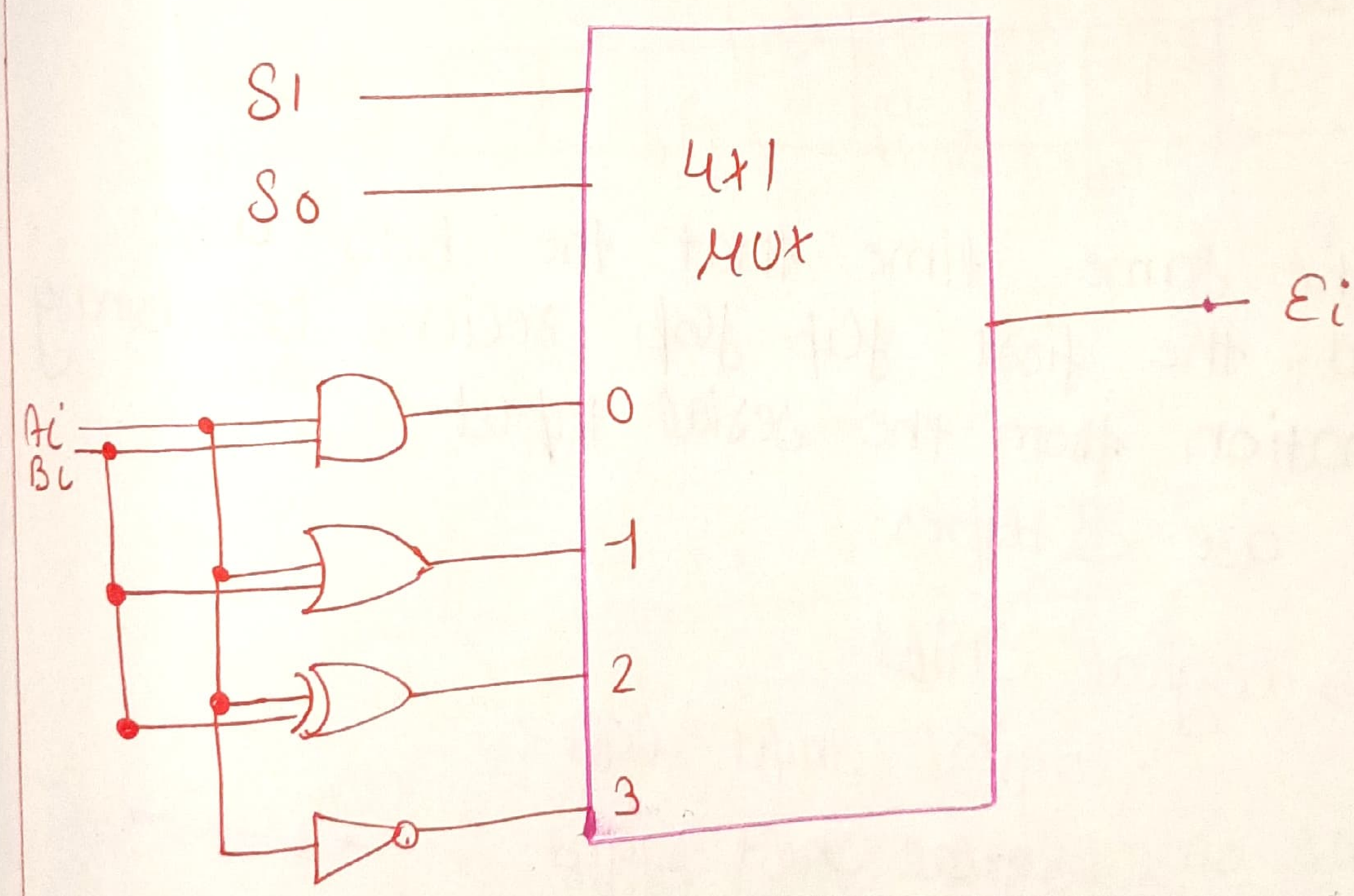
- Each of the 16 columns f_0 through f_{15} represents a truth table of one possible boolean function for the two variable x and y.

Boolean function	Microoperation	Name.
$f = 0$	$f \leftarrow 0$	Clear
$f_1 = xy$	$f \leftarrow A \wedge B$	AND
$f_2 = xy'$	$f \leftarrow A \cap B'$	
$f_3 = x$	$f \leftarrow A$	transfer A.
$f_4 = x'y$	$f \leftarrow A' \cap B$	AND
$f_5 = y$	$f \leftarrow B$	Transfer B
$f_6 = x \oplus y$	$f \leftarrow A \oplus B$	Exclusive -OR.
$f_7 = x + y$	$f \leftarrow A \vee B$	OR
$f_8 = (x+y)'$	$f \leftarrow \overline{A \vee B}$	NOR
$f_9 = (x \oplus y)'$	$f \leftarrow \overline{A \oplus B}$	Exclusive NOR.
$f_{10} = y'$	$f \leftarrow \overline{B}$	Complement B
$f_{11} = x + y'$	$f \leftarrow A \vee \overline{B}$	" OR

$f_{12} = x'$	$f \leftarrow \bar{A}$	Complement A
$f_{13} = x' + y$	$f \leftarrow \bar{A} \vee B$	
$f_{14} = (xy)$	$f \leftarrow \overline{A \wedge B}$	NAND
$f_{15} = 1$	$f \leftarrow$ all 1's	Set of all's

Hardware implementation

- It requires that logic gates be inserted for each bit or pair of bits in the register to perform the required logic function.
- Although there are 16 logic microoperation, Most computers use only for AND, OR, XOR. from which all others can be derived



S1	S0	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	$E = \bar{A}$	Complement

Shift Micro-operations

- Shift micro-operation are used for serial transfer of data.
- The content of a register can be shifted to the left or the right
- During a shift left operation, the serial input transfer a bit into the rightmost position.

  Eg:→ A = 1101

  after shift → 1010

- During a shift right operation, the serial input transfer a bit into the leftmost position.

  Eg:→ A = 1101

  after shift = 0110

- At the same time that the bits are shifted, the first flip-flop recieves its binary information from the serial input.

→ There are 3 types

Copyrighted by Codeleap.io
All rights reserved.

→ logical shift
- logical shift left
- logical shift right

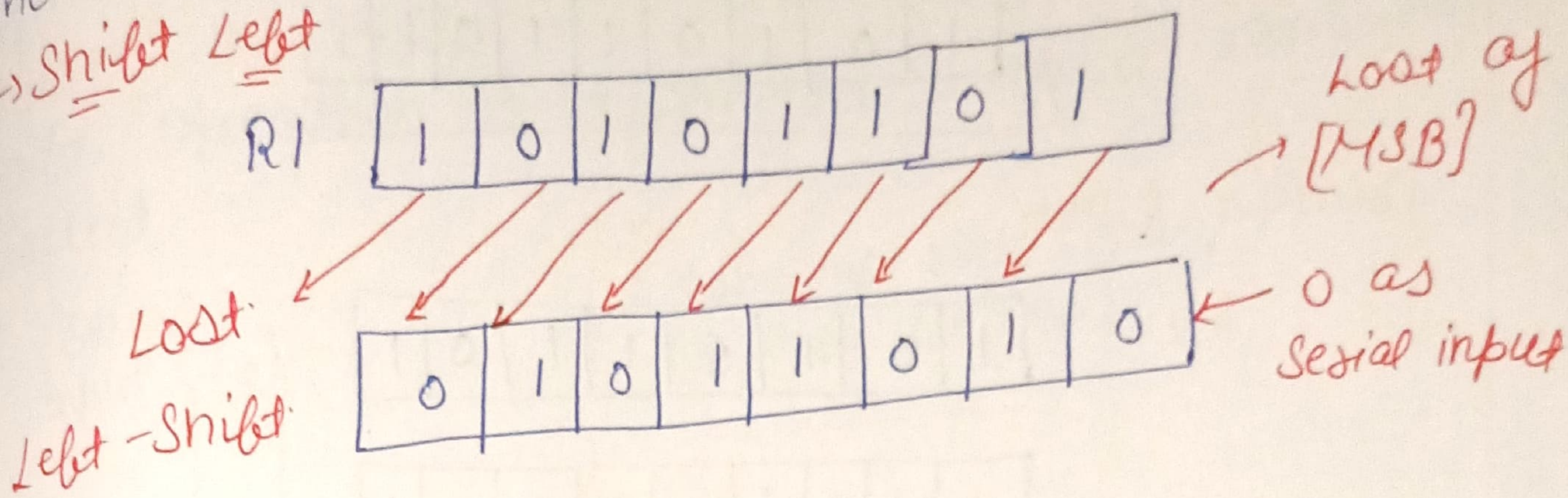
→ Circular shift
- Circular shift left
- Circular shift right

→ Arithmetic shift
- Arithmetic " left
- " " right

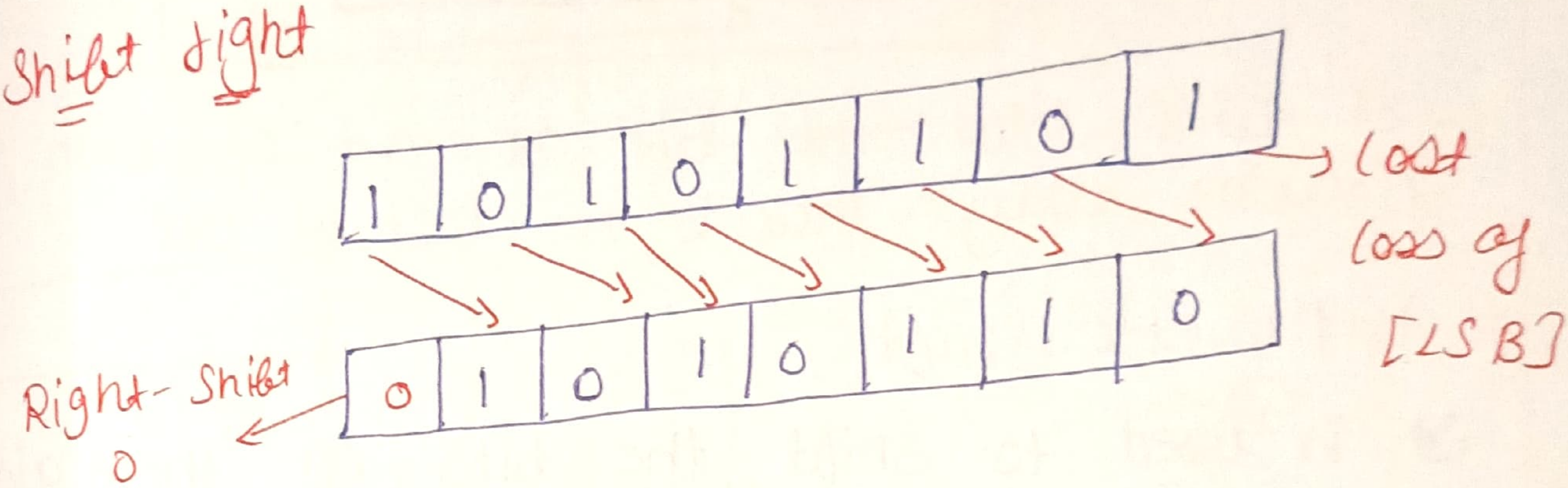
1) Logical Shift

A logical shift is one that transfer $\boxed{0}$ through the serial input.

→ Shift Left



→ Shift right



[Shl → Shift left]
[Shr → Shift right]

$$\boxed{\begin{aligned} R1 &\leftarrow Shl\ R1 \\ R2 &\leftarrow Shr\ R2 \end{aligned}}$$

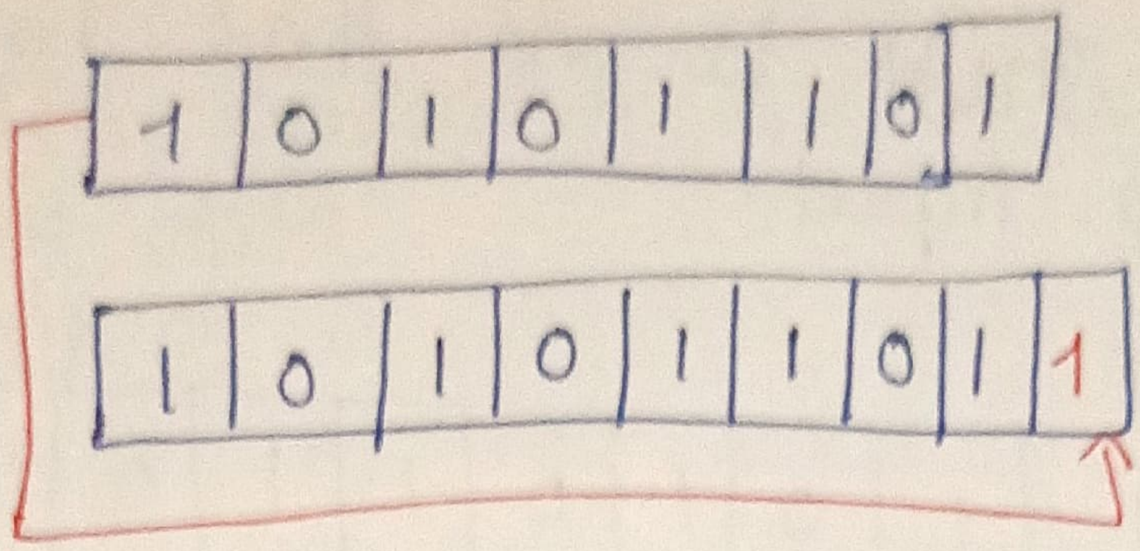
2) Circular Shift

The circular shift (also known as rotate operation) circulates the bit of the register around the two ends without loss of information.

$$\boxed{\begin{aligned} R1 &\leftarrow Cil\ R1 \\ R2 &\leftarrow Cil\ R2 \end{aligned}}$$

Circular - left

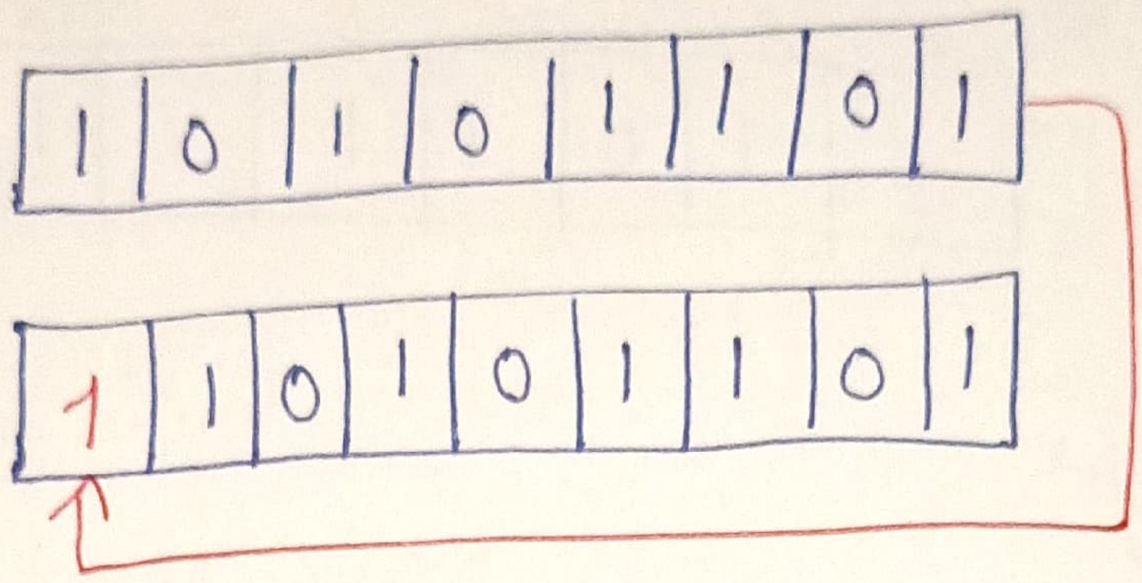
R1



[MSB lost]

Circular - Right

R2.



- It will shift the bit at end position in the circular way. There is no lost of bit

Arithmetic Shift

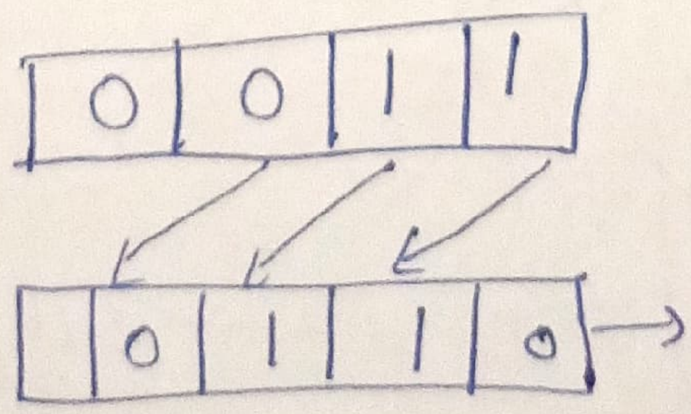
It is used to shift the bits of the registers, to keep its sign unchanged.

R1 ← ashl R1

R1 ← ashr R2

- Shift - left Multiples by 2.
- Shift - right Divides by 2.
- The sign- bit is 0 for positive and 1 for negative.

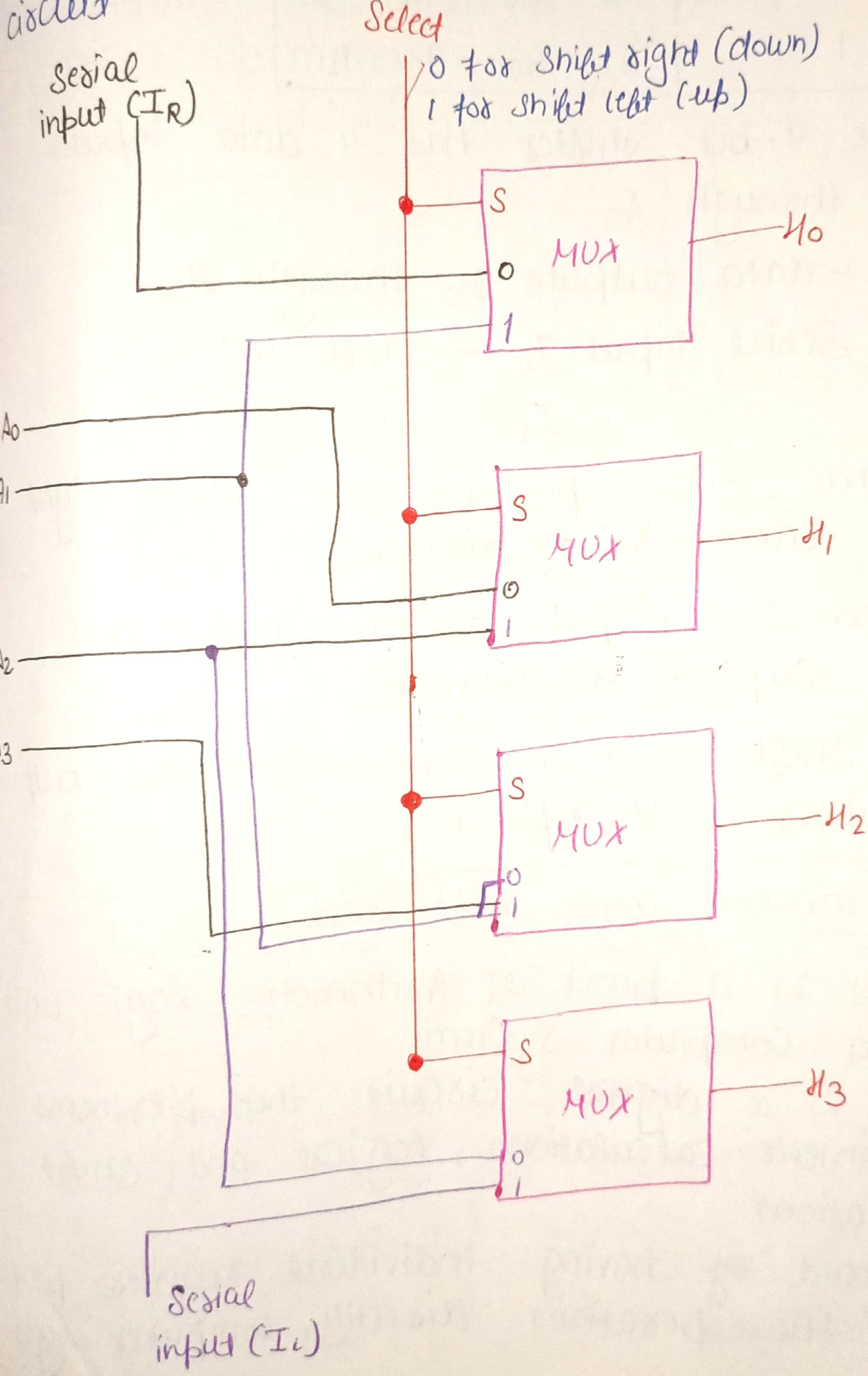
Ex:- Shift - left No = +3



→ as serial input.

Hardware implementation

A clock pulse is needed to load data into the register and another clock pulse is needed to initiate the shift.
Shift operation is implemented with a combinational circuit



Function Table

Select S	Output H_0	H_1	H_2	H_3
0	I_R	A_0	A_1	A_2
1	A_1	A_2	A_3	I_L

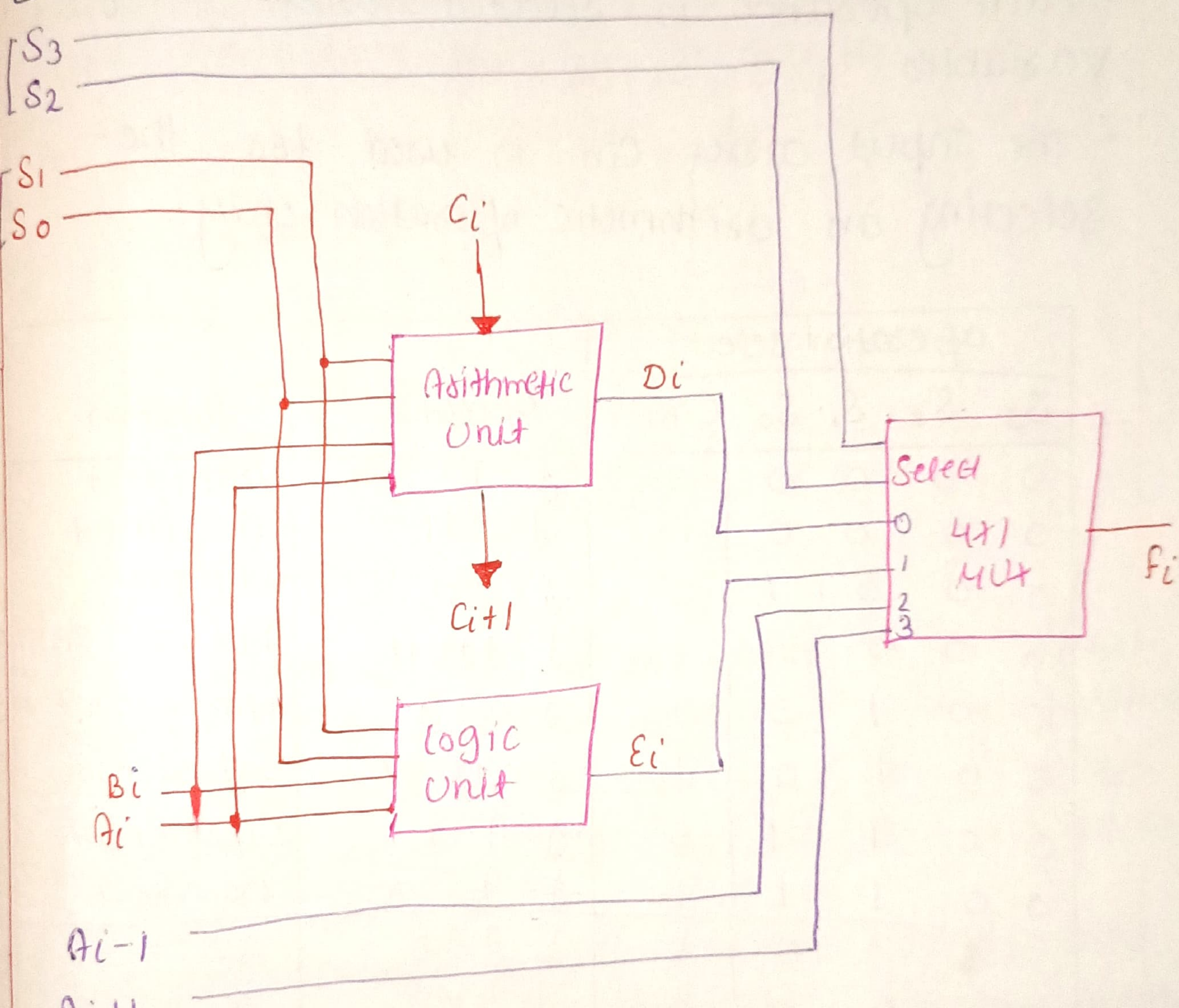
- The 4-bit shifter has 4 data inputs A_0 through A_3
- 4 - data outputs H_0 through H_3
- 2 Serial input $I_L \longrightarrow$ shift left
  $I_R \longrightarrow$ Shift right
- when $S = 0$, input data are shifted right (down in the diagram)
- when $S = 1$, input data are shifted left (up in the diagram)
- A Shifter with n data inputs and outputs requires n Multipleres.

#Arithmetic logic Shift unit

- ALSU is a part of Arithmetic logic unit of a computer system
- It is a digital circuit that performs arithmetic calculations, logical and shift operations.
- Instead of having individual registers performing the Microoperations directly, computer system

employ a number of storage registers connected to a common operational unit called an Arithmetic logic unit

- ALSO consists of arithmetic, logical and shift circuits. In each stage, the circuit can perform
  8 arithmetic operations, 16 logical operations and
  2 shift operations



- S_0, S_1] Microoperation selected.
  - → A_{i-1} = Shift - Right operation
  - → A_{i+1} = Shift - Left operation
- S_3, S_2] data in MUX
  - → C_{i+1} = output carry of A stage
  - → C_i = Input carry

- The stage of the ALU is shown. Inputs A_i and B_i are applied to both the arithmetic and logic units. A particular microoperation is selected with inputs S_1 and S_0.
- A 4×1 Multiplexer at the output chooses b/w an arithmetic output D_i and a logic circuit a logic output in E_i
- Each operation is selected with the five variables S_3, S_2, S_1, S_0 and C_{in}.
- The input carry C_{in} is used for the selecting an arithmetic operation only.

Operation select					Operation	function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	$f = A$	Transfer A
0	0	0	0	1	$f = A + 1$	Increment A
0	0	0	1	0	$f = A + B$	addition
0	0	0	1	1	$f = A + B + 1$	Add with carry
0	0	1	0	0	$f = A + \overline{B}$	Subtract with borrow
0	0	1	0	1	$f = A + \overline{B} + 1$	Subtraction
0	0	1	1	0	$f = A - 1$	Decrement A
0	0	1	1	1	$f = A$	Transfer A
0	1	0	0	x	$f = A \wedge B$	AND
0	1	0	1	x	$f = A \vee B$	OR
0	1	1	0	x	$f = A \oplus B$	XOR
0	1	1	1	x	$f = \overline{A}$	Complement A
1	0	x	x	x	$f = shr\, A$	Shift right A into f
1	1	x	x	x	$f = shl\, A$	" left A into f

- 8 arithmetic operation are selected with $S_3 S_2 = 00$
- 4 logic operation are selected with $S_3 S_2 = 01$
- Input carry has no effect during the logic operation and is marked with don't - case $\boxed{x's}$
- Last 2 operation are shift operations and are selected with $S_3 S_2 = 10$ and 11.

→ The ALSU is responsible for the condn.

1) Equal to
2) Less than
3) Greather than.

Copyrighted by Codeleap.io
All rights reserved.

Microprogrammed Control

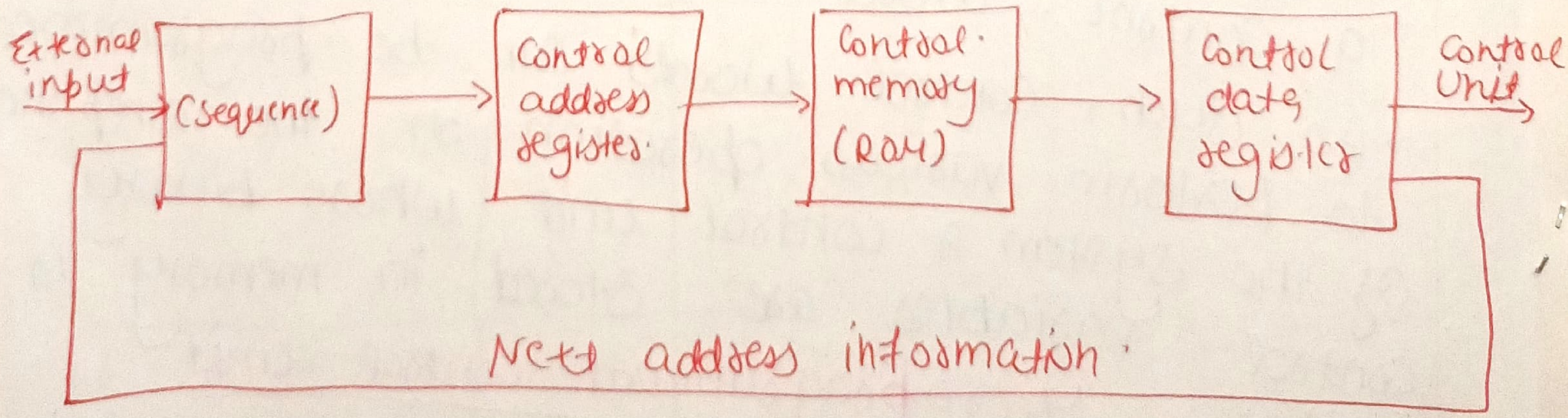
1) Control Memory

- The function of the control unit in a digital computer is to initate sequences of Microoperations.
- The control variables at any given time can be represented by a string of 0's and 1's called a control word.
- As such, control words can be programmed to perform various operation on the components of the system A control unit whose binary control variables are stored in memory is called a Microprogrammed control unit
- Each word in control memory contains within it a Microinstruction

- A sequence of Microinstructions constitutes a Microprogram
- can be either read-only memory (ROM) or writable control Memory (dynamic micro-programming)
- A computer that employs a Microprogrammed control unit will have two separate memories
  1) Main Memory
  2) Control Memory
- The general configuration of a Microprogrammed control unit
  1) The control Memory is assumed to be a ROM. within which all control information is permantly Stored.
  2) The control address register specifies the address of Microinstruction.
  3) The control data register hold the Micro-operations read from memory

Microprogram

1) program Stored in memory that generates all the control signals required to execute the instruction set correctly.
   - Consists of Microinstructions.



Microprogrammed control organization.

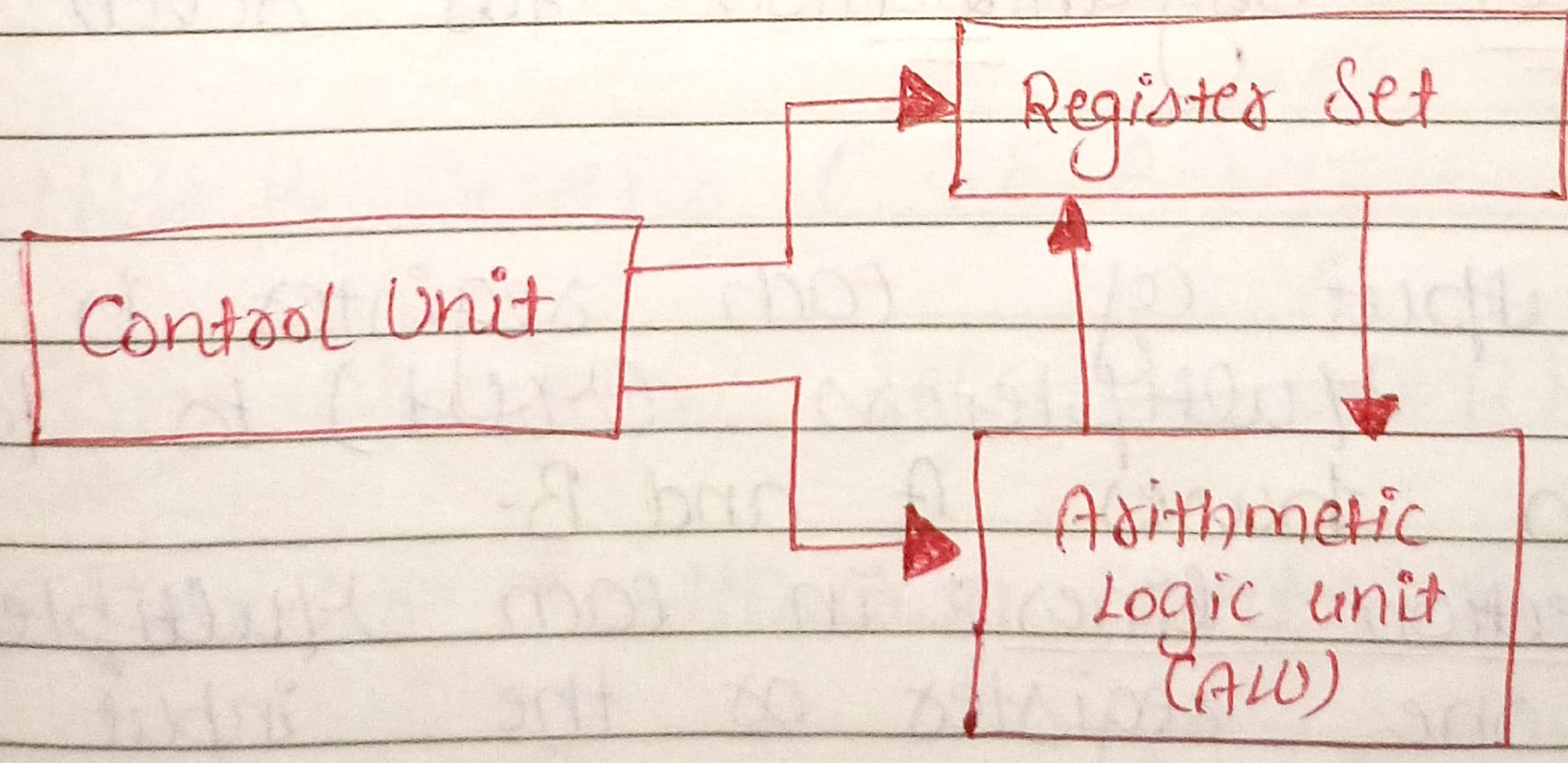
Central processing unit

The part of the computer that performs all data - processing operations is called CPU. CPU is made up of three major parts.

1) Registers Set :→ Store intermediate data using during the execution.

2) ALU :→ performs microoperations for executing the instructions.

3) Control unit :→ supervises the transfer of information among registers and instructions the ALU as to which operation to perform



Major components of CPU

Date.....

General register organization

- Memory locations are needed for -
  Storing pointers
  counters
  return address
  temporary results.

- When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system
- The register communicate with each other not only for direct data transfers, but also while performing various Microoperation.
- Here it is necessary to provide a comm unit that can perform all the arith logic and Shift Microoperation in the processor.

→ A bus organization for seven CPU registers

- The output of each register is conne to two Multiplexers (MUX) to form the two buses A and B.
- The Selection lines: in each Multiplexer select one register as the input data for the particular bus.
- The A and B: buses form the inputs

Date.....

to a common ALU

The operation: Selected in the ALU determines the arithmetic or logic-Micro-operation that is to be performed.

The result of the Microoperation: is available for output data and also goes into the inputs of all the registers.

The register: that recieves the information from the output bus is selected by a decoder.

The decoder: activities one of the register load inputs, thus providing a transfer path b/w the data in the output bus and the inputs of the selected destination register.

=> The control provide four selection variables in the control unit and mus be available at the begining of a Clock Style.

1) MUX A Selector (SELA)

2) MUX B Selector (SELB)

3) ALU Operation Selector (OPR)

4) Decoder destination Selector (SELD)

Date.....

⇒ Control word

There are 14 binary selection inputs in the unit and their combined value specifies a control word.

3	3	3	5
SELA	SELB	SELD	OPR

- 3-bits SELA :→ Select a source register for A input of ALU.
- 3-bit SELB :→ Select a source register for B input of ALU
- 3-bit SELD :→ Select a destination register using decoder and its 7 load outputs
- 5-bits OPR :→ Select one of the operations in the ALU.
- 14-bit control word when applied to selection inputs specify a particular microoperations.

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	

Date.....

Ex: R1 ← R2 — R3

R1 → SELD
R2 → SELA
R3 → SELB

Field :>	SELA	SELB	SELD	OPR
Symbol :>	R2	R3	R1	SUB
Binary :	010	011	001	00101.

Stack Organization

[Last - In - first - out]
Lifo principle follow
A Stack is a Storage device that Stores information in such a manner that the item Stored last in the first item retrieved.
The Stack in digital computers is a group of Memory locations with a register that holds the address of top of element
This register that holds the address of top of element of the Stack is called Stack pointer

Stack operations

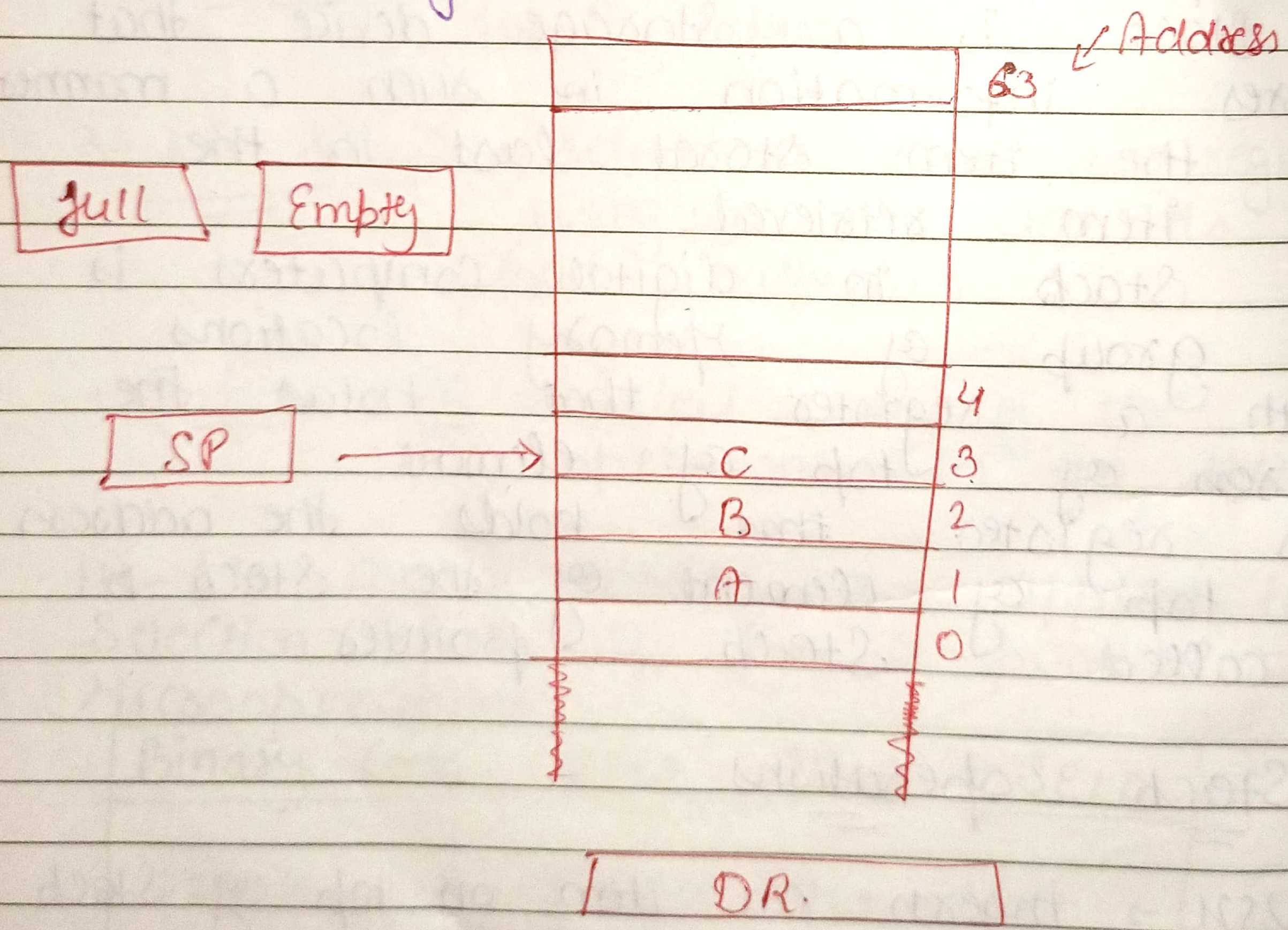
PUSH → inserts an item on top of Stack.
POP → delete " " from " " of "

Date.....

Implementation

1) Register Stack :→ A Stack can be implemented in a ~~ser~~ organized as a collection of finite number of registers that are used to store temporary information during execution of a program.

- The Stack pointer (SP) is a register that holds the address of top of element of the Stack.



→ Block diagram of 64-Word Stack

Date.....

→ 3 item → A, B, C
→ C is on top of Stack.
→ SP = 3
→ To remove top item, the Stack is popped by reading the memory at address 3 and decrement SP

SP = 2, TOP = B

→ To insert new item D, SP = 2+1
TOP = D

→ If Stack is full, FUL ← 1
" " is Empty, EMPTY ← 1.

a) PUSH Operation

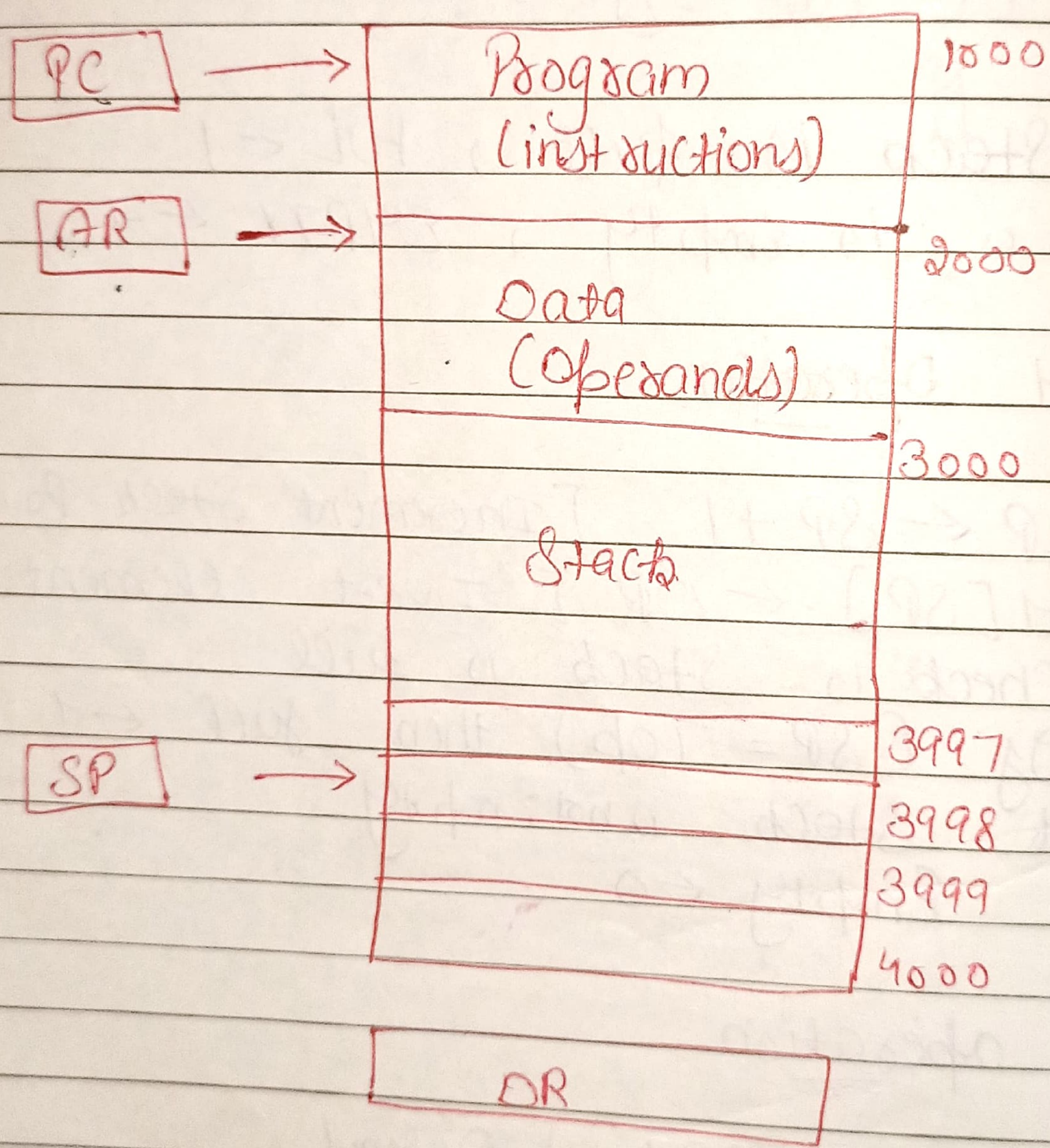
1) SP ← SP + 1 [Increment Stack Pointer]
ii) M[SP] ← DR [Insert element.
iii) Check is Stack is full.
If (SP = Top) then full ← 1.
iv) If Stack is not Empty
Empty ← 0

b) POP operation

1) DR ← M[SP] [Delemt.
2) SP ← SP - 1
3) Check if Stack is empty
SP = 0 then Empty ← 1
4) not full.
full ← 0

Memory Stack

A Stack can be implemented in a random access memory attached to CPU. The implementation of a Stack in the CPU is done by assigning portion of Memory to a Stack operation and using a processor register as a Stack pointer.



- PC points at the address of the next instruction in the program
- AR points at an array of data
- SP points at top of Stack

Date.....

PC is used during fetch phase to read an instruction.
AR is used during execute phase to read an operand.
SP is used to push or pop items into or from the Stack.

Stack limits :- Most computers do not provide hardware to check for Stack overflow or underflow
The Stack limits can be checked by using 2 processor register:
→ One to hold the upper limit (3000) and the other to hold the lower limit (4000).
→ After a push operation, SP is compared with the upper-limit register and after a pop operation.
→ SP is compared with the lower-limit register

Use of Stack

1) Reverse Polish Notation:

A+B Infix
+AB Prefix
AB+ Postfix or reverse Polish Notation.

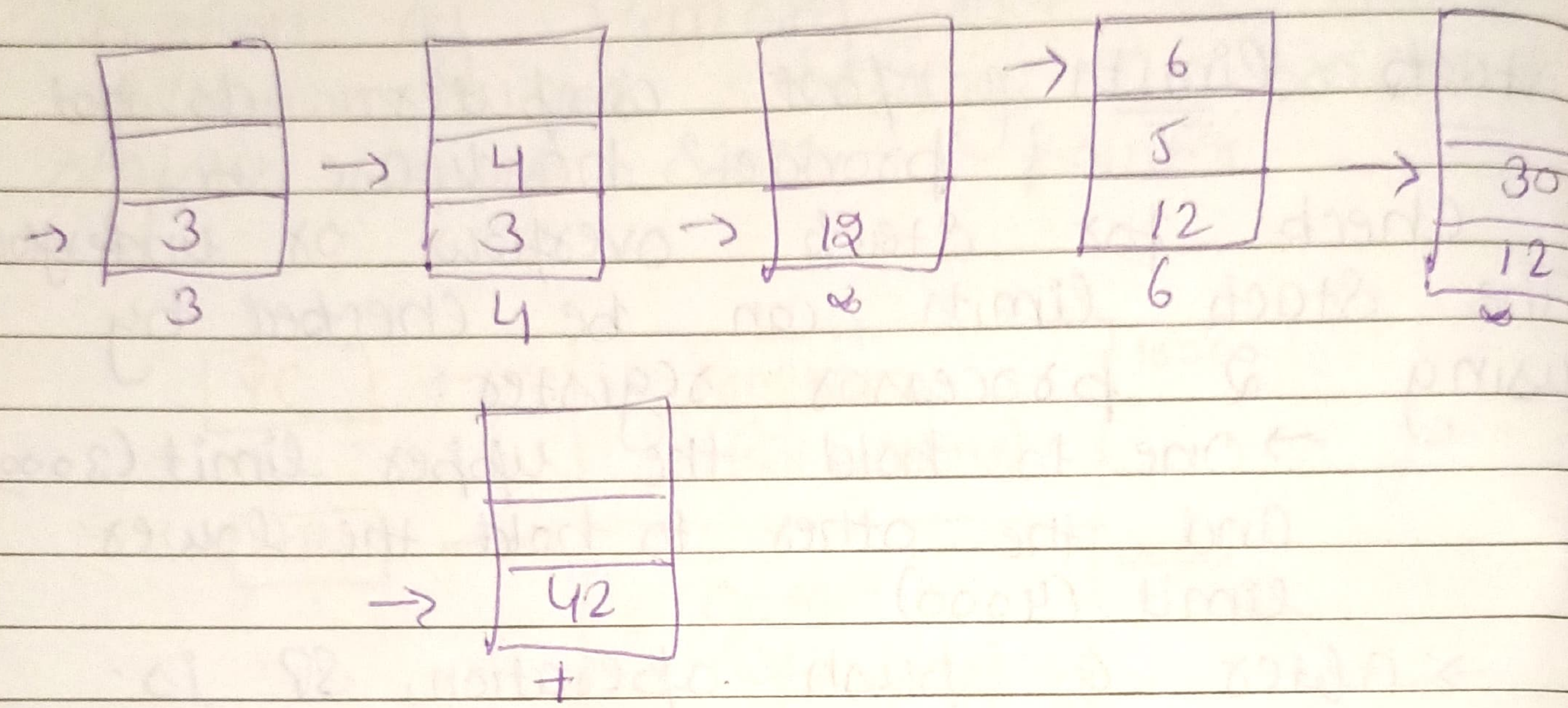
A*B + C*D
RPN = AB* CD* +.

Date.....

2) for storing return address for sub-routines.

3) Stack operations

$3*4+5*6 \rightarrow [34*56*+] \rightarrow$ RPN



Instruction format

Instruction format is a binary format which specifies a computer instruction

- The bits of the instructions are divided into groups called fields
- The Most Common fields in instruction.

1) opcode (operation code):→ It is a operation code.

- every code in form of binary [001, 01
- Specifies the operation to be performe such as add, Mult, Shift etc.

Date.....

Eg: A + B
↳ operand.

Address :→ Specifies a memory address or processor register, where operand is stored.

Mode :→ direct or indirect
Memory or Register } Types

[now the located operand.]

The effective address of the operand is determined.

Mode	opcode	Address

computer may have instruction of several different length containing no. of address (address filed may be 3, 2, 1, 0)

The no. of address filed in the instruction format of a computer depends on the internal organization of it register

Most computers fall into one of the three type of CPU organization
1) Single Accumulator organization
2) General register organization.
3) Stack organization.

Date.....

→ Type of instruction / format

1) Three - address instruction format
2) Two - address instruction format } Single Accumulator
3) One - address instruction format [General]
4) zero - address instruction format [Stack]

Single - Accumulator

1) All operations are performed on implied AC register.
2) instruction format used only one address field.
3) all operands implicity.

Eg: LOAD X // AC ← M(X)
ADD X // AC ← AC + M(X)

General - register

1) All It use set of general purpose register.

2) Two or three instruction format use.

3) all operands explicity.

Eg. ADD R1, R2, R3 // R1 ← R2 + R3.

Date.....

Stack organization

→ operation are carried out the top of stack.

operations are implicity.

→ It does not use any address field for the instruction like ADD, MULT [zero - address instruction format].

→ PUSH and POP instructions are used to communicate with stack.

Ex:
PUSH A
PUSH B
ADD
POP C

Types of Instruction format

1) Three address

opcode	Destination address	source address 1	Sou ad 2

→ Three - address instruction format has three address fields.
→ one address field is used for destination and two address fields for source.
→ Each address field may specify a processor register or a memory operand.
→ also called General register organization

Date.....

Assembly language Notation	Equivalent Register Transfer Notation
Ex ADD R1, R2, R3	// R1 ← R2 + R
ADD R1, A, B	// R1 ← M[A] +

→ Advantage → result in short programs when evalute arithmetic expression.

→ Disadvantage :→ Binary - coded instruction require too many bits to specify three address, complex.

Ex $C = A + B$

ADD C, A, B // M[C] ← M[A] + M[B]

D S1 S2

→ result is stored into Memory

Two address instructions

opcode	Destination Address	Source address

- Two address instructions are most common in commerical computers.
→ It has two address field.

Date.....

Each address field may specify a processor register or a memory operand.

Ex: ADD R1, R2 ← R1 ← R1 + R2

MOV R1, R2 // R1 ← R2

Ex: C = A + B

MOV C, A // M[C] ← M[A]

ADD C, B // M[C] ← M[C] + M[B]

One address instruction

opcode	operand address

→ only one address field. Depend on instruction.

→ one address instruction use a implied Accumulator (AC) register for all data Manipulation.

- All operation done b/w accumulator register and memory operand.

→ also known as Single accumulator organization.

→ Ex:→ ADD X // AC ← AC + M[X]

Date.....

Load and Store instruction used to transfers to and from memory to AC register.

Zero address

| opcode |

- Stack is used.
- Does not use address filed like ADD, Sub.
- PUSH and POP instruction, needs an address field to specify the operand
- To evaluate a arithmetic expression, first it is converted to reverse Polish Notation.

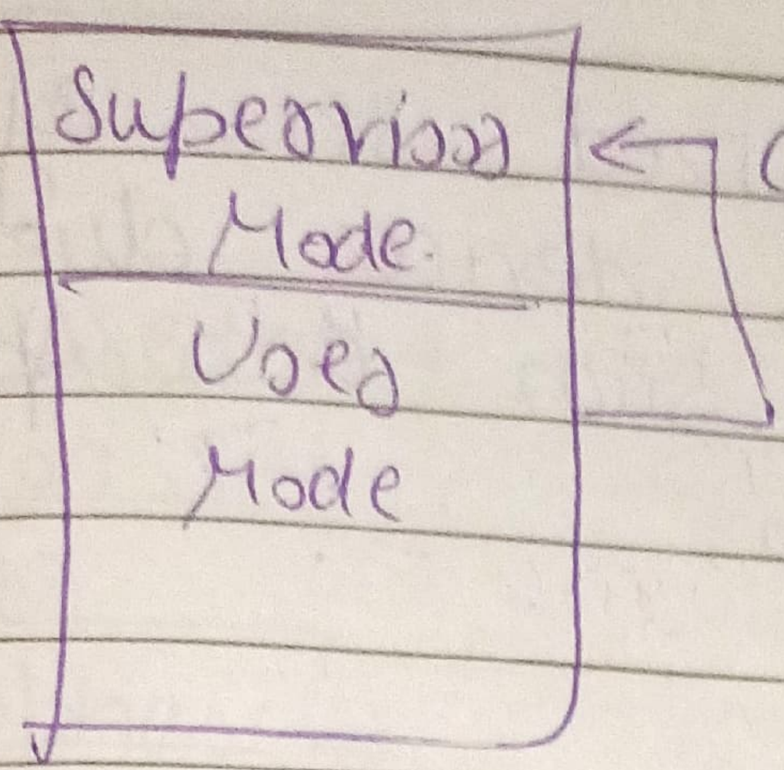
→ Zero address → absence of address filed in an instruction.

Ex → $c = a + b$

Reverse Polish (AB+)

PUSH A	//	TOS ← A
PUSH B	//	TOS ← B
ADD	//	TOS ← (A+B)
POP C	//	M[C] ← TOS

Date.....



CPU → switch CPU from User Mode to Supervisor Mode.

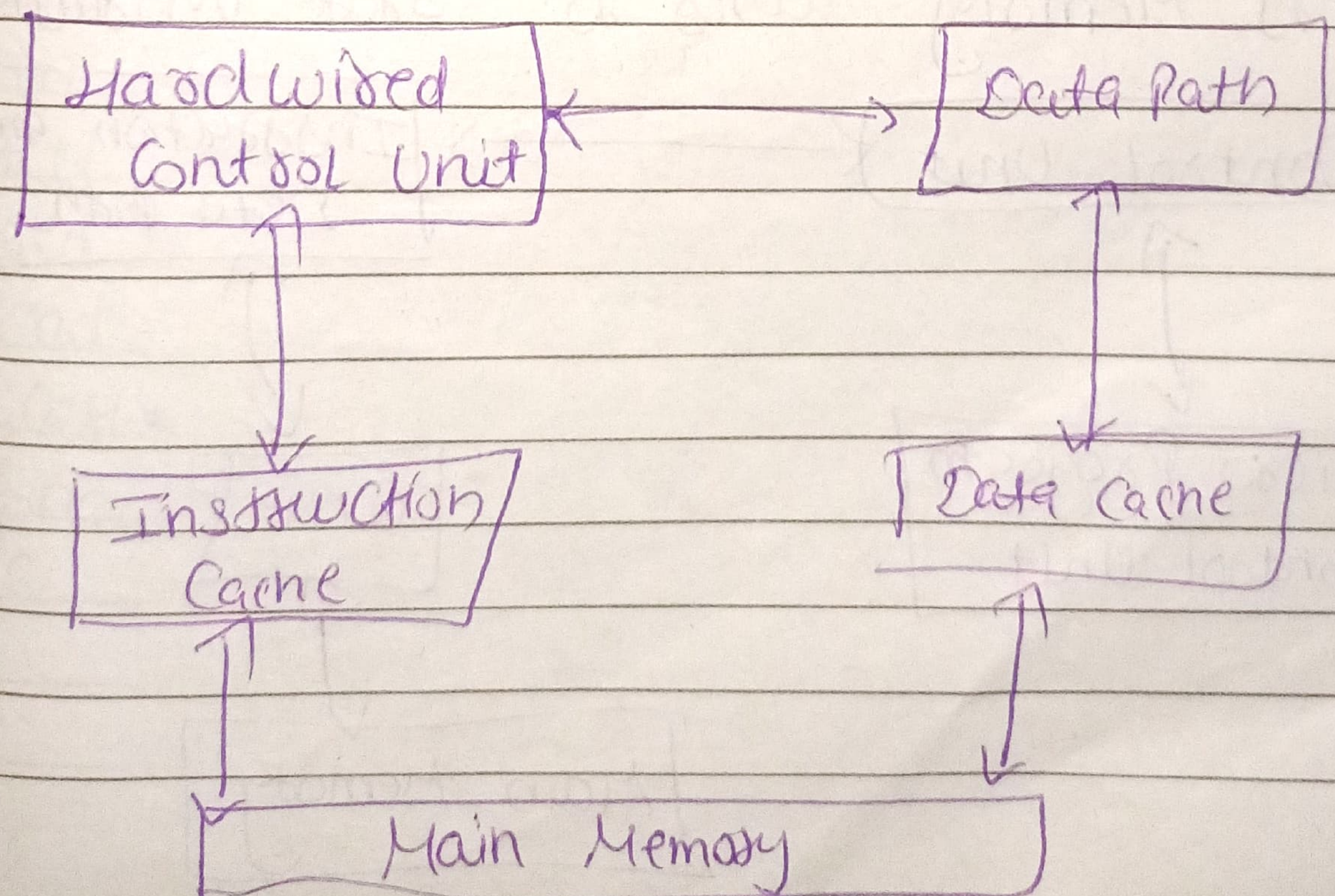
Maskable → Delayed
Non Maskable → cannot be delayed

RISC

Reduced Instruction Set computer.

It is designed to reduce the execution time by simplifying the instruction set.
Each instruction requires only 1 clock cycle. 2) fetch, decode, execute.

Architecture



Date.....

- It is highly customized set of instruction used in portable devices due to system reliability such as Apple iPod, Mobiles / smartphones.

Feature

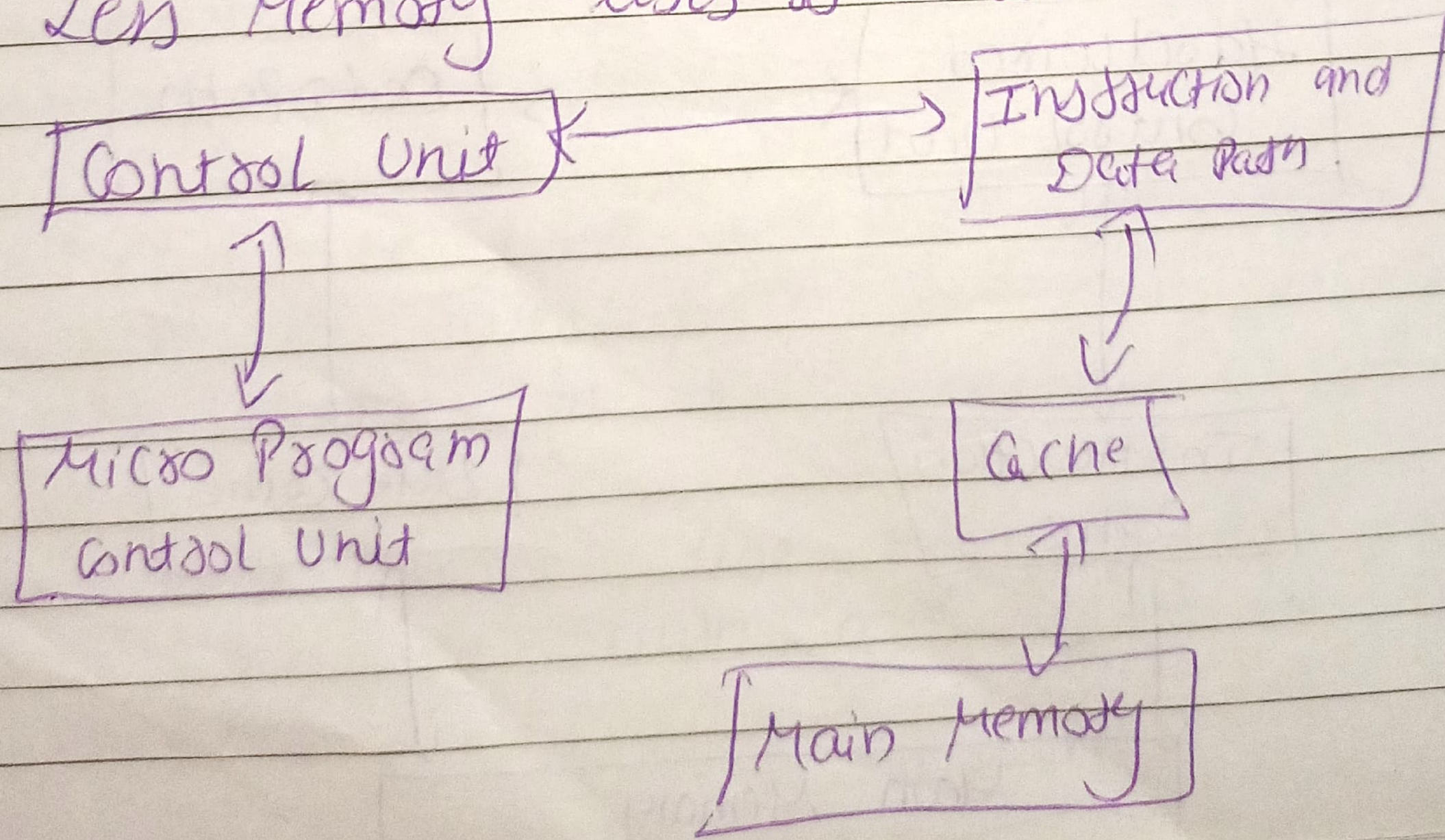
1) One cycle execution time.
2) Pipelining technique.
3) A large no. of registers.
4) Uses LOAD and STORE instruction to access the Memory location

CISC Processor

Complex Instruction Set Computer

To Minimize the no. of instructions program & ignoring the no. of cycles per instructions.

→ Used in desktop, laptops.
→ Less Memory uses to store instruction



Memory Hierarchy

The computer Memory hierarchy looks like a pyramid structure which is used to describe the difference of memory types.

- The Memory in a computer can be divided into five hierarchies based on the speed.
- The processor can move from one level to another based on its requirements.
- The five hierarchies in the memory are registers, cache, main memory, Magnetic disc, and Magnetic Tapes.

This Memory Hierarcy is divided into 2 types

1) External Memory or Secondary Memory

→ accessible by the processor through an I/O Module.

→ Peripheral device include [Magnetic Disk, Magnetic Tape]

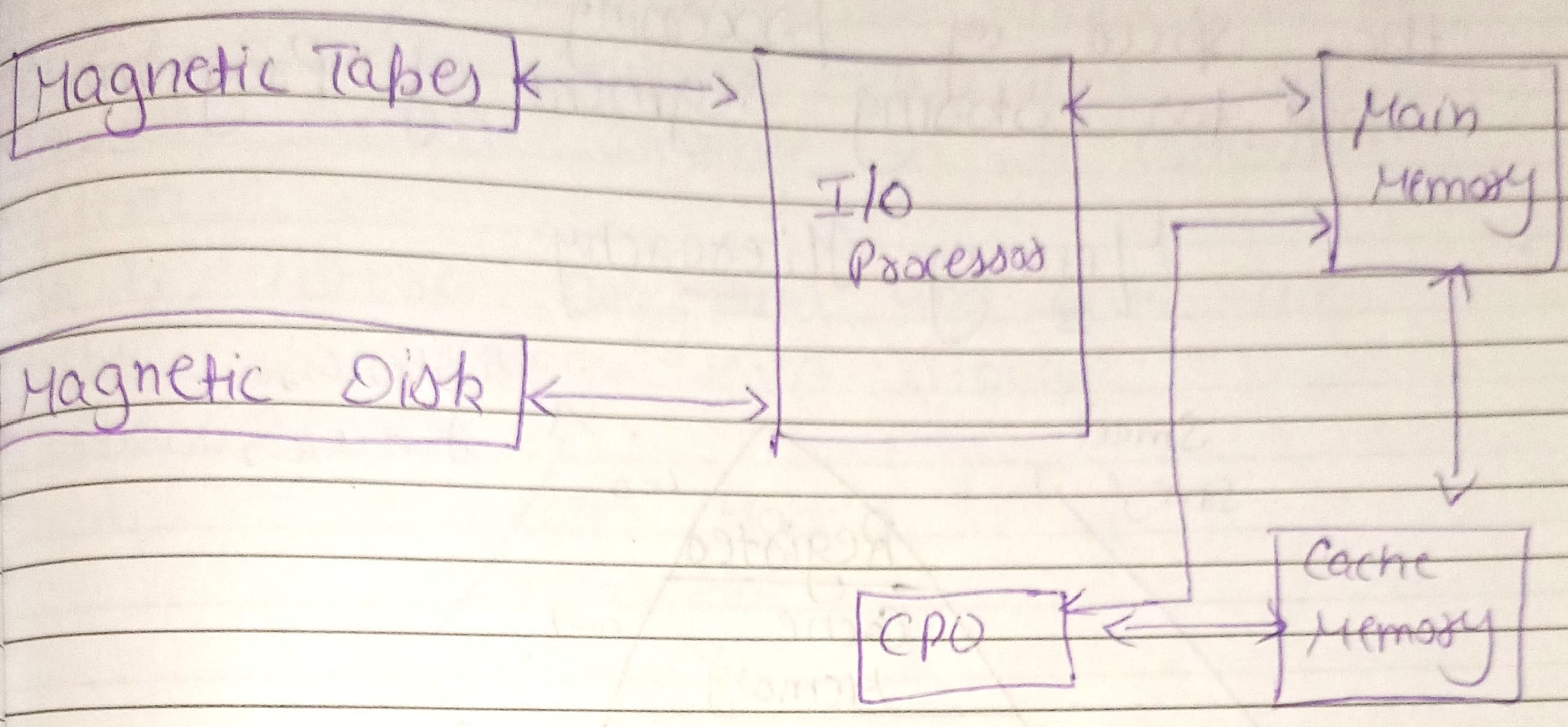
2) Internal Memory or Primary Memory

→ accessible by processor.

→ comparing of Main Memory, Cache Memory CPU registers

Date.....

Components of typical Memory hiearchy



Main Memory :→ The Memory unit that communicates directly with the CPU.

- Refered as RAM

Auxiliary Memory :→ Device that provide backup Storage.

Magnetic tape and disks are used as auxiliary Memory.

- also known as lowest - cost, highest capacity and slowest access Storage.

I/O processor :→ The connect b/w auxiliary Memory and Main Memory

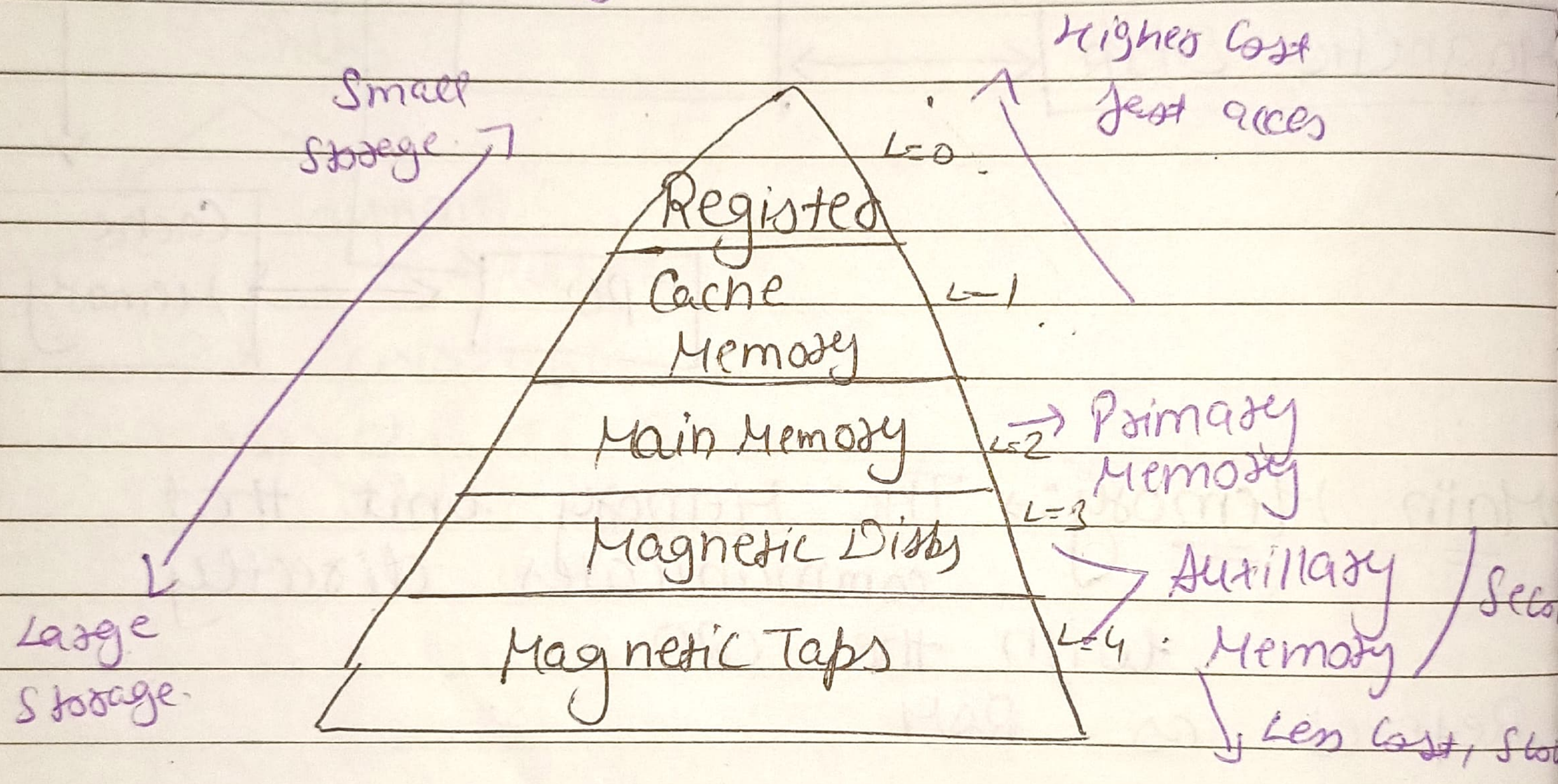
→ Manage the data transfer

Date.....

Cache Memory :- A special very- high speed is called a Cache is used to increase the speed of processing.

- Used for storing segment of Program

Memory Hierarchy



- The first three hierarchies are volatile memories which means when there is no power and then automatically they lose their stored data.

- The last two are non- volatile memo means they store the data permenently.

→ Characterstics

Performance, Ability, Access Time, Cost per bit

Date.....

Auxiliary Memory

auxiliary memory is known as the lowest - cost, highest - capacity and slowest - access storage in a computer system.
It is where programs and data are kept for long - term storage.

Characteristics

- Access Mode.
- Access time.
- Capacity
- Transfer rate.

Access time :→ The average time required to reach a storage location in memory

Seek time :→ Required to specify the position of reader head.

Update

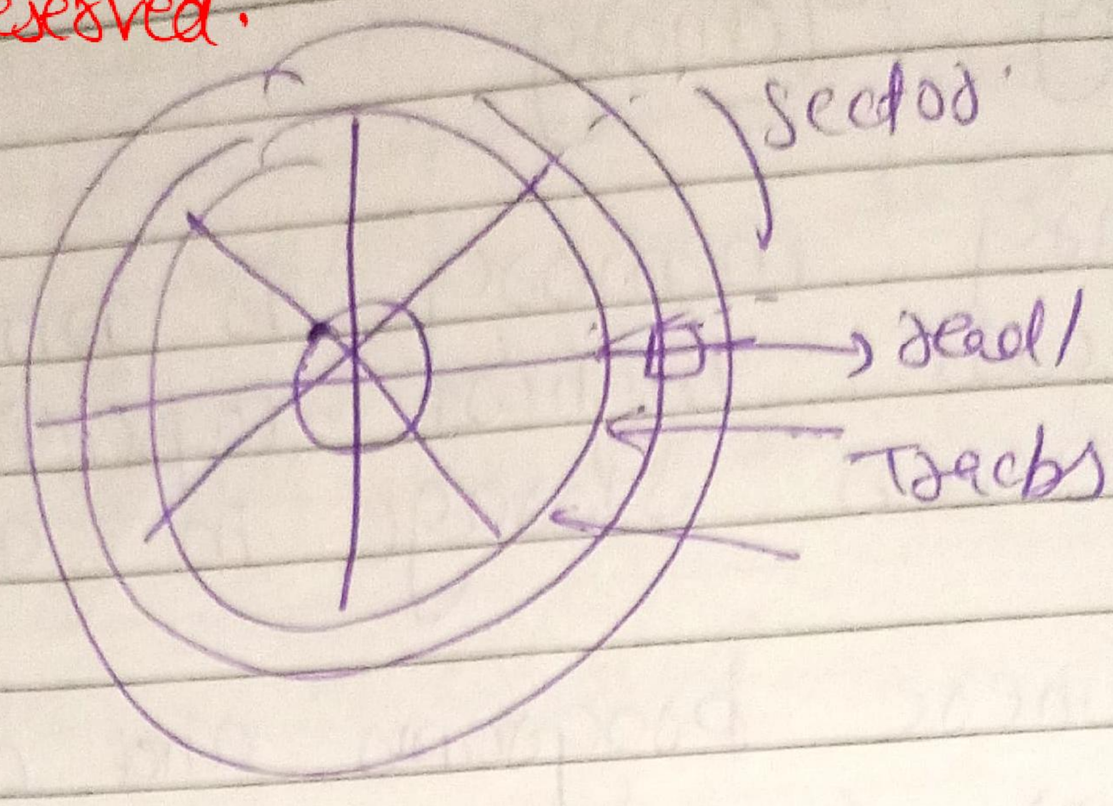
Magnetic Disks → ~~Sequential Access Device~~ Direct Access

Non volatile, large data.

A Magnetic disk is a type of Memory constructed using a circular plate of Metal or plastic coated with magnetized materials.
both sides of the disks are used to carry out read/ write operations.
Several Disks may be stacked on one spindle with read/ write heads available on each surface.

Copyrighted by Codeleap.io
All rights reserved.

Date.....



- All disks rotate together at high speed and are not stopped for aces purpose.

- Bits are stored in the Magnetized surface in spots along concentric circle called Tracks.

- The concentric circle (tracks) are commonly divided into sections called sectors.

→ Type:

Floppy Disk :→ old concept
→ Data Store
→ portable [transfer]
→ Store only some data
→ Made up of plastic coated.
→ A Soft of Magnetic Disk

A Hard Disk → Non - voiler
→ HDD know
→ Permant attack data.
→ 0 & 1 form data.