Date
Page No.

Computer Organization and Architecture

Unit-I

The term data refers to factual information used for analysis or reasoning.
Information is a collection of facts or data that is communicated.

The data types found in registers of digital computers may be classified as being one of the following categories:-
1) Numbers used in arithmetic computations
2) Letters of the alphabet used in data processing
3) Other discrete symbols used for specific purposes.

Decimal to Binary No :- $(15)_{10}$ =

2	15	1
2	7	1
2	3	1
2	1	1
	0	

$= (1111)_2$

(b) $(15.375)_{10} \Rightarrow$

$0.375 \times 2 = 0.750$ — 0
$0.75 \times 2 = 1.5$ — 1
$0.5 \times 2 = 1$ — 1

$= (1111.011)_2$

Binary to decimal no. :- $(11.001001)_2$

$2^1 \; 2^0$

$= 2^0 \times 1 + 2^1 \times 1 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3} + 0 \times 2^{-4} + 0 \times 2^{-5} + 1 \times 2^{-6}$

$= 1 + 2 + \frac{1}{8} + \frac{1}{64} = 3 + \frac{9}{64} = 3.15$

Octal to decimal no :- $(123.75)_8$

$1 \times 8^2 + 2 \times 8^1 + 3 \times 8 + 7 \times 8^{-1} + 5 \times 8^{-2} = 3 + 16 + 64 + \frac{7}{8} + \frac{5}{64}$

$= 83.953$

Date
Page No.

Decimal to Octal :- $(83)_{10}$

8	83	3 ↑
8	10	2
8	1	1
	0	

$= (123)_8$

Binary to Octal :- $(1011010100)_2$

$= \underset{1}{001}\ \underset{3}{011}\ \underset{2}{010}\ \underset{4}{100} = (1324)_8$

Binary to Hex :- $(1011010100)_2$

$= \underset{2}{0010}\ \underset{D}{1101}\ \underset{4}{0100} = (204)_{16}$

Hex to Binary :- $(204)_{16} =$

$\underset{2}{0010}\ \underset{D}{1101}\ \underset{4}{0100} = (001011010100)_2$

Binary Coded Decimal (BCD) :-

Decimal no	B.C.D.
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000
20	0010 0000

Date
Page No.

Alphanumeric Representation :- An alphanumeric character set is a set of elements that includes the 10 decimal digits, the 26 letters of the alphabet and a number of special characters, such as $, +, and =.

Character	Binary Code
A	100 0001
B	100 0010
C	100 0011
D	100 0100
E	100 0101
F	100 0110
G	100 0111
H	100 1000
I	100 1001
J	100 1010
K	100 1011
L	100 1100
M	100 1101
N	100 1110
O	100 1111
P	101 0000
Q	101 0001
R	101 0010
S	101 0011
T	101 0100
U	101 0101
V	101 0110
W	101 0111
X	101 1000
Y	101 1001
Z	101 1010

Character	Binary Code
0	011 0000
1	011 0001
2	011 0010
3	011 0011
4	011 0100
5	011 0101
6	011 0110
7	011 0111
8	011 1000
9	011 1001
Space	010 0000
.	010 1110
(	010 1000
+	010 1011
$	010 0100
*	010 1010
)	010 1001
-	010 1101
/	010 1111
,	010 1100
=	011 1101

Date
Page No.

Complements (r's and $(r-1)$'s

	r's	$(r-1)$'s
$r=10$	10's	9's
$r=8$	8's	7's
$r=2$	2's	1's

r's complement $= r^n - N$ where r = base, n = No. of digits, N = No.

$(r-1)$'s complement $= r^n - N - 1$

eg (a) $(7)_{10}$

10's $= 10 - 7 = 3$

9's $= 10 - 7 - 1 = 2$

(b) $(56)_{10}$

10's $= (10)^2 - 56 = 100 - 56 = 44$

9's $= (10)^2 - 56 - 1 = 100 - 56 - 1 = 43$

$\Rightarrow$ $(r-1)^s$ Complement $= r^n - N - 1$

$(r-1)^s$ Complement $+1 = r^n - N$.

$(r-1)^s$ complement $+1$ = r's complement

eg $(1101)_2$.

2's $= 2^4 - 1101 = (16)_{10} - 1101 = 10000 - 1101$

$= 00011$

1's $= 1111 - 1101 = 0010$

$10000 - 1101 = 10011$

Date
Page No.

eg (563)8.

7's complement = 777 − 563 = 214

8's complement = 214 + 1 = 215

⇒ Fixed-Point representation :—

→ Positive integers, including zero, can be represented as unsigned numbers.
→ To represent negative integers, we need a notation for negative values.
→ It is customary to represent the sign with a bit placed in the left-most position of the number.
→ The sign bit equal to 0 for positive and 1 for negative.

- Integer representation :—

→ For positive integer binary number, the sign is represented by 0 and magnitude by a positive binary numbers.
→ When the number is negtive, the sign is represented by 1 but the rest of the number may be represented in any of three possible ways :—

- Signed-magnitude representations of $(+14)_{10}$: 0 0 0 1 1 1 0
- Signed-1's complement " : 1 1 1 0 0 0 1
- Signed-2's " " : 1 1 1 0 0 1 0

- Arithmetic Addition :—

- Arithmetic Subtraction :—

Date
Page No.

• Decimal Fixed Point Representation :-

→ A four bit decimal code requires four flip flops for each decimal digit. The representation of 4385 in BCD requires 16 flip-flops, four flip flops for each digit.

→ Whereas 4385 in binary uses 13 bits only.

→ By representing numbers in decimal we are wasting a considerable amount of storage.

→ Also the circuits required to perform arithmetic in decimal are more complex.

→ The sign of a decimal number is usually represented by four bits to conform with the 4-bit code of the decimal digits. It is necessary to designate a plus with four 0's and a minus with the BCD equivalent to 9, which is 1001.

eg (+375) = (0000 0011 0111 0101)BCD

(−240) (1001 0010 0100 0000)BCD

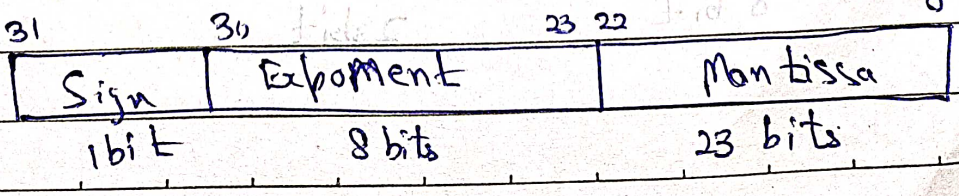
• Floating Point Representation :-

It has three parts :- (a) Mantissa, (b) Base (c) Exponent

Number	Mantissa	Base	Exponent
• 3×10^6	3	10	6
• 110×2^8	110	2	8
• 6132.784	6132784	10	−3

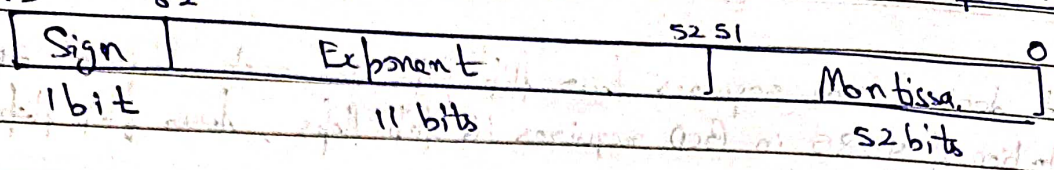
We use IEEE 754 floating point number representation

(a) Single precision format Representation $(1 \cdot N)_2^{E-127}$



Date
Page No.

(b) Double precision format Representation = $(1-N)_2^{E-1023}$



Q Represent $(1259.125)_{10}$ in single and double precision format

Step 1: Convert decimal number to binary number.

$(1259.125)_{10} = (10011101011.001)_2$

Step-2 Normalize the number

$(1\,00\,111\,01\,011.001)_2$

$(1.\underline{00\,111\,0101\,1001})_2 \times 2^{10}$

N

Step-3 Single precision format

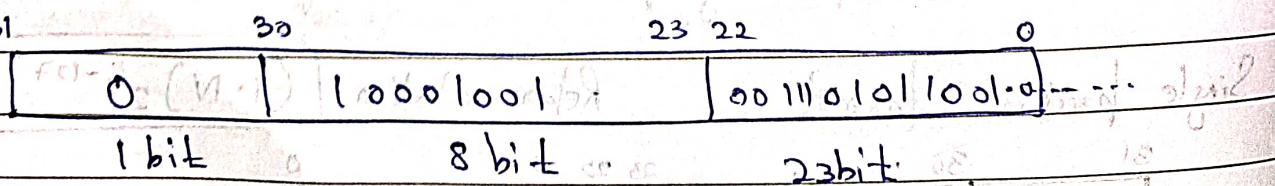
$(1-N)_2^{E-127}$

$(1.00\,111\,0101\,1001)_2 \times 2^{10}$

$E - 127 = 10$

$E = 137$ $\therefore E = (10001001)_2$

S E M



Date
Page No.

Step-4 :- Double precision format

$(1-N)\,2^{E-1023}$

$1.00111010111001 \times 2^{10}$

$E - 1023 = 10$

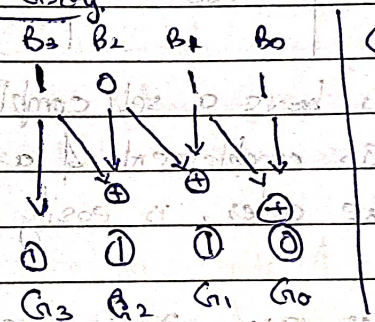
$E = 1033 = (10000001001)_b$

G3	G2	S2 S1
0	10000001001	00111010111001 ....00....
1 bit	11 bit	52 bit

Gray Codes :- Another example of non-weighted codes. It has a special feature that only one bit change each time, the decimal no is incremented.

	Binary	Gray
0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 0
4	0 1 0 0	0 1 1 0
5	0 1 0 1	0 1 1 1
6	0 1 1 0	0 1 0 1
7	0 1 1 1	0 1 0 0
8	1 0 0 0	1 1 0 0
9	1 0 0 1	1 1 0 1
10	1 0 1 0	1 1 1 1
11	1 0 1 1	1 1 1 0
12	1 1 0 0	1 0 1 0
13	1 1 0 1	1 0 1 1
14	1 1 1 0	1 0 0 1
15	1 1 1 1	1 0 0 0

Binary to Gray :-



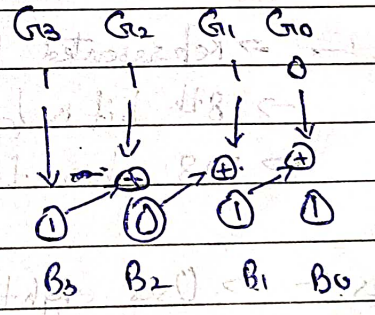
$G_3 = B_3$

$G_2 = B_3 \oplus B_2$

$G_1 = B_2 \oplus B_1$

$G_0 = B_1 \oplus B_0$

Gray to Binary :-



$B_3 = G_3$

$B_2 = G_2 \oplus B_3$

$B_1 = G_1 \oplus B_2$

$B_0 = G_0 \oplus B_1$

Date
Page No.

Other Decimal Codes :—

Decimal digit	BCD	2-4-2-1	Excess 3	Excess-3 Gray
0	0000	0 0 0 0	0 0 1 1	0 0 1 0
1	0001	0 0 0 1	0 1 0 0	0 1 1 0
2	0010	0 0 1 0	0 1 0 1	0 1 1 1
3	0011	0 0 1 1	0 1 1 0	0 1 0 1
4	0100	0 1 0 0	0 1 1 1	0 1 0 0
5	0101	1 0 1 1	1 0 0 0	1 1 0 0
6	0110	1 1 0 0	1 0 0 1	1 1 0 1
7	0111	1 1 0 1	1 0 1 0	1 1 1 1
8	1000	1 1 1 0	1 0 1 1	1 1 1 0
9	1001	1 1 1 1	1 1 0 0	1 0 1 0

→ Two codes have a self complementing property which means that the 9's complement of a decimal number, when represented in one of these codes, is easily obtained by changing 1's to 0's and 0's to 1's.

These codes are 2-4-2-1 and Excess-3 codes.

Alphanumeric Codes :—

- **ASCII codes** :— → Represented by 7-bit code
  → 8th bit is for parity
  → 128 characters

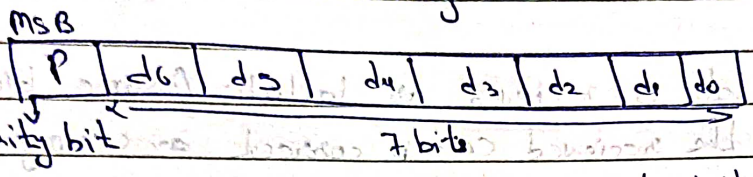
- **EBCDIC codes** :— → Uses eight bits for each character
  → 9th bit is for parity

Date
Page No.

⇒ Error detection and Correction

1. Parity Checker
2. Hamming code
3. Cyclic Redundancy check

Parity Checker :- The MSB of an 8-bit data is used as parity bit and remaining 7-bit are used as data or msg. bit.



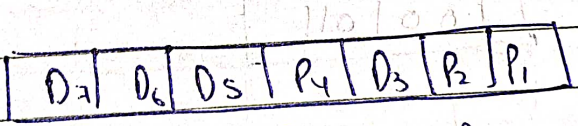
P = 0 Even.
P = 1 odd.

eg:- transmitted code: 0 | 1 0 0 1 0 1 1 ← data bits →

Recieved code: 0 | 0 0 0 1 0 1 1 Error.

drawbacks :- ① It does not detect all types of errors.
② Not suitable for multiple bit errors.

Hamming Codes → It can be applied to data units of any length
→ It is used to detect and correct single bit errors
→ All bit positions that are power of 2 are marked as parity bits (1, 2, 4, 8) other bits are data bits



eg:-

D7	D6	D5	P4	D3	P2	P1
1	1	0		1		

P_1 = check 1 bit, skip 1 bit. . . . .
= (1, 3, 5, 7, 9 . . .)

P_2 = check 2 bits, skip 2 bits . . . .
= (2, 3, 6, 7, 10, 11, 14, 15)

P_4 = check 4 bits, skip 4 bits . . . .
= (4, 5, 6, 7, 12, 13, 14, 15)

DELTA Notebook

Date
Page No.

Oth

$P_1 = 0$ even
$P_2 = 1$ odd
$P_4 = 0$ even.

Recieved code :-	D_7	D_6	D_5	P_4	P_3	P_2	P_1
	1	1	0	0	1	1	0

for ~~even~~ odd parity. there is on error at $(010)_2$ bit.
ie. error at 2nd place.

Corrected code = 1 1 1 0 1 1 0

d) A 7 bit hamming code is recieved as 1011011. Assume Even parity and state whether the recieved code is correct or wrong. If wrong locate the bit in error.

Sol^n

Recieved Code.

D_7	D_6	D_5	P_4	P_3	P_2	P_1
1	0	1	1	0	1	1

$P_1 =$ ~~1 1 0~~ 1 0 1 1 odd parity. 1
$P_2 =$ 1 0 0 1 even parity 0
$P_4 =$ 1 1 0 1 odd parity 1

∴ Recieved code is wrong.

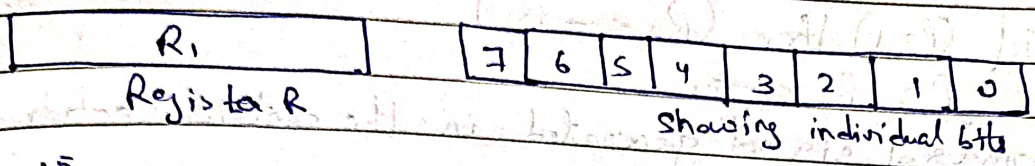
$E = P_4\ P_2\ P_1$

1 0 1 = 5th bit is error

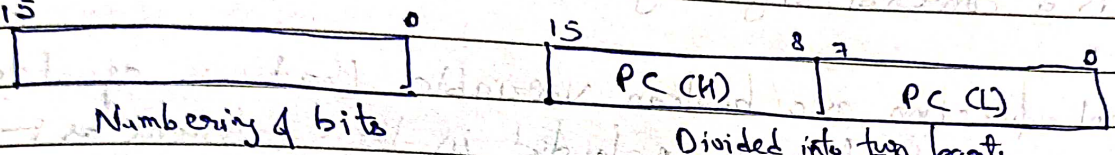
Corrected code = 1 0 0 1 0 1 1

Date
Page No.

Register :- Computer Register are designated by capital letters.
Eg :- MAR - Memory address Register
PC - Program Counter
IR - Instruction Register
R1 - Processor Register



Register R — Showing individual bits

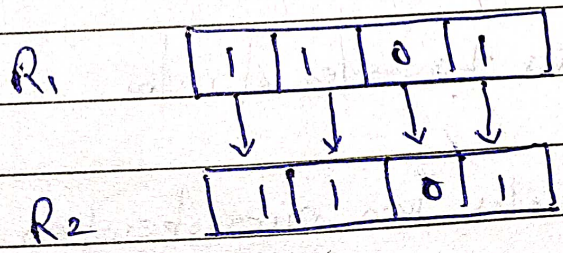


Numbering of bits — Divided into two parts

Register Transfer Language :-

→ The symbolic notation used to describe the microoperation transfer among register is called a register transfer language.
→ The term register transfer implies the availability of hardware logic circuit that can perform a stated microoperation and transfer the result of the operation to the same or another register.
→ A register Transfer language is a system for expressing in symbolic form the microoperation sequences among the registers of a digital module.
→ Information transfer from one register to another is designated in symbolic form by means of a replacement operator is known as Register Transfer.

$R_2 \leftarrow R_1$



Date
Page No.

Register Transfer with Control function :-

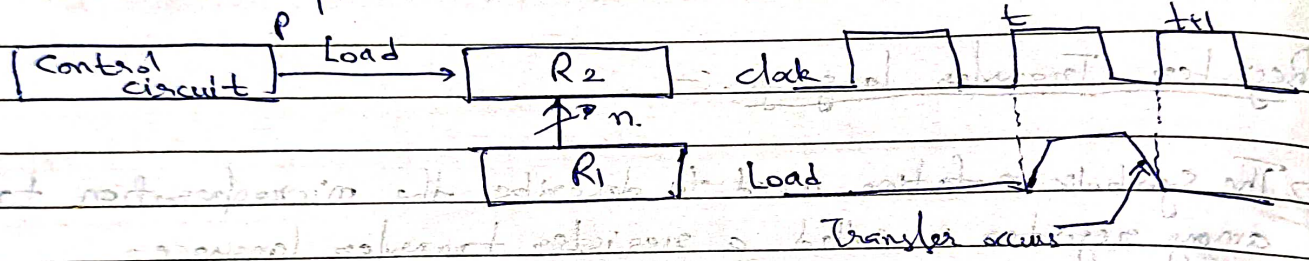
→ Normally, we want the transfer to occur only under a predetermined control condition using if-then statement.

$$\text{if } (P = 1) \text{ then } (R_2 \leftarrow R_1)$$

$$P : R_2 \leftarrow R_1$$

where P is a control signal generated in the control section.

→ A control function is a boolean variable that is equal to 1 or 0. The control function is included in the statement as :-



Microoperations :- The operations executed on data stored in registers are called microoperations.

eg :- shift, count, clear, load.

Basic symbols for Register Transfers :-

(1)	Letters (and numerals)	Denotes a Register	MAR, R_2
(2)	Parenthesis ( )	denotes Part of Register	$R_2(0-7)$
(3)	Arrow $\leftarrow$	denotes transfer of info.	$R_2 \leftarrow R_1$
(4)	Comma ,	Seperates two microoperation	$R_2 \leftarrow R_1, R_3 \leftarrow R_4$

Date
Page No.

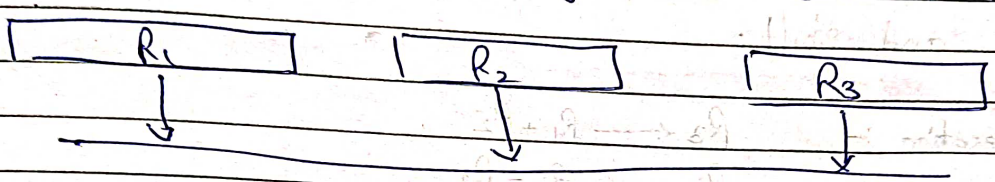
Bus and Memory Transfer :-

In a digital system with many registers, it is impractical to have data and control lines to directly allow each register to be loaded with the contents of every possible other registers.

→ So, there is one centralized set of circuits for data transfer - bus.

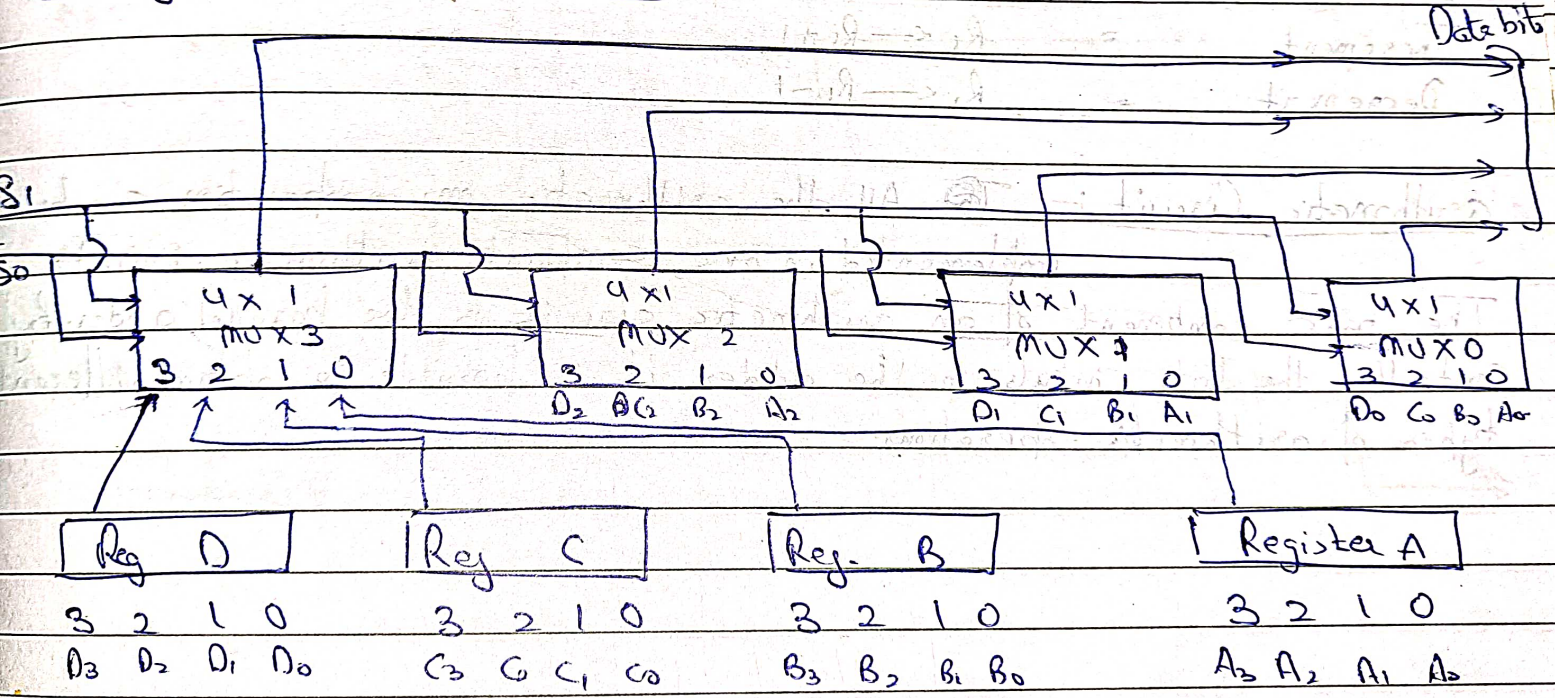
→ It have control circuits to select which register is the source and which is the destination.

→ Can have several sources to any of several destination.



eg:- $R_2 \leftarrow R_1$ or $Bus \leftarrow R_1$, $R_2 \leftarrow Bus$

Memory transfer using Multiplexers:-



S_1	S_0	Register
0	0	A
0	1	B
1	0	C
1	1	D

⇒ Arithmetic Microoperations :-

The basic arithmetic microoperation are addition, subtraction, increment, decrement, and shift.

Add microoperation :- $R_3 \leftarrow R_1 + R_2$

Subtract mi,, = $R_3 \leftarrow R_1 - R_2$

= $R_3 \leftarrow R_1 + \bar{R}_2 + 1$

Complement = $R_2 \leftarrow \bar{R}_2$

2's complement = $R_2 \leftarrow \bar{R}_2 + 1$

Increment = $R_1 \leftarrow R_1 + 1$

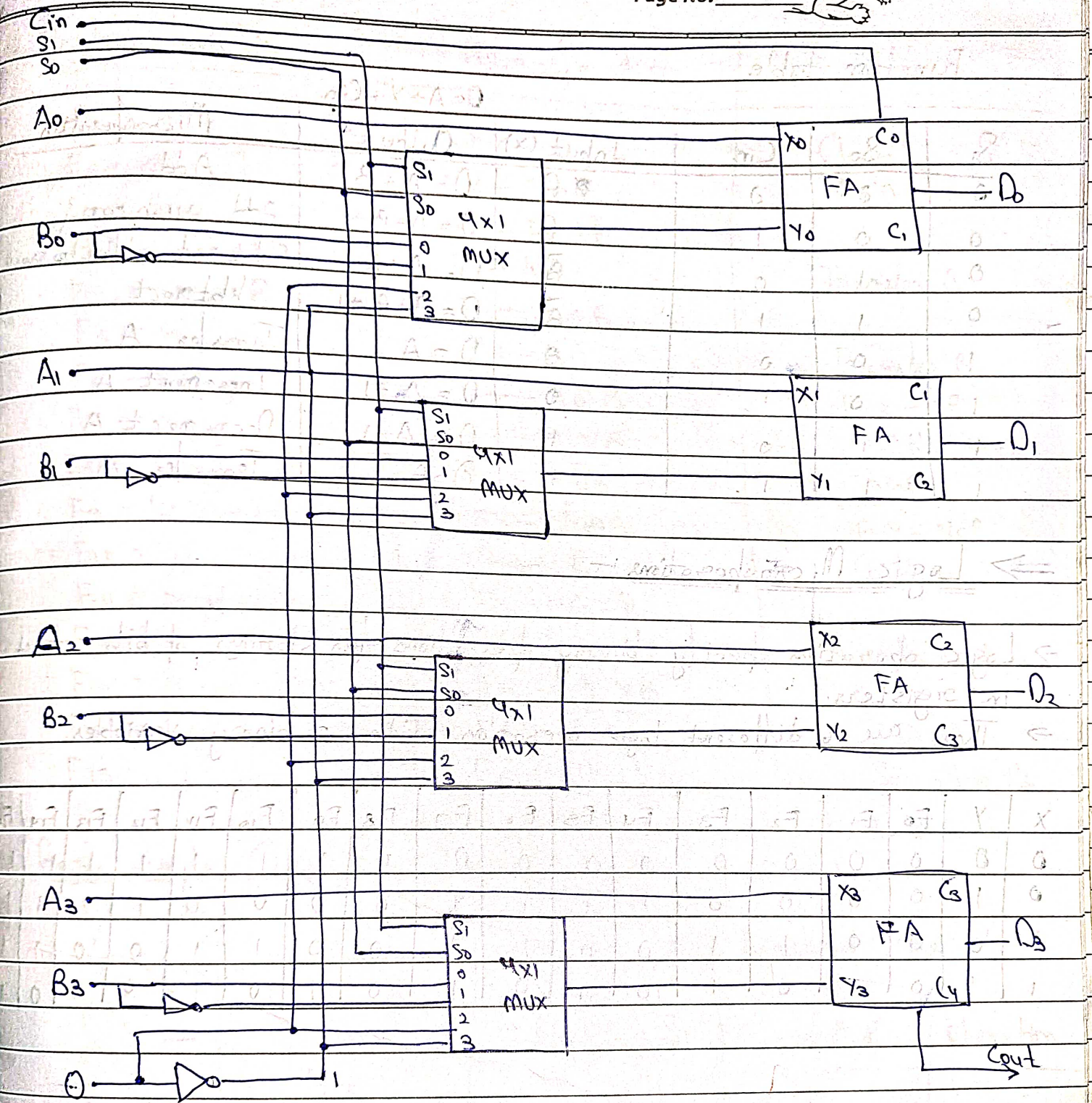
Decrement = $R_1 \leftarrow R_1 - 1$

Arithmetic Circuit :- All the arithmetic microoperation can be implemented in one composite arithmetic circuit.

The basic component of an arithmetic circuit is the parallel adder. By controlling the data inputs to the adder, it is possible to obtain different types of arithmetic operations.

Date _____

Page No. _____


Cin
S1
S0
A0
B0
A1
B1
A2
B2
A3
B3
0
1
S1
S0
4x1
MUX
0
1
2
3
X0
C0
FA
Y0
C1
D0
X1
C1
FA
Y1
C2
D1
X2
C2
FA
Y2
C3
D2
X3
C3
FA
Y3
C4
D3
Cout

Date
Page No.

Function table

$D = A + Y + C_{in}$

S_1	S_0	C_{in}	Input (Y)	Output	Microoperation
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	Add with carry.
0	1	0	$\bar{B}$	$D = A + \bar{B}$	Subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

⇒ Logic Microoperations :-

→ Logic operations specify binary operations for strings of bits stored in registers.

→ There are 16 different logic operations with 2 binary variables.

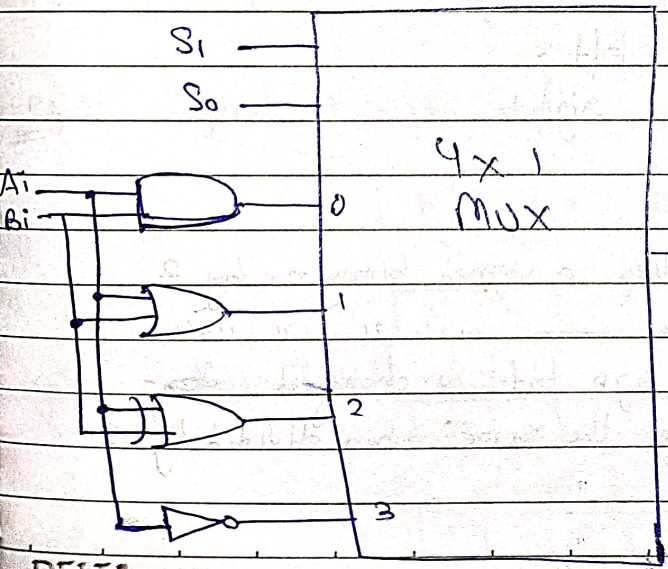
X	Y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

∧ = AND
∨ = OR
— = NOT

Date
Page No.

Boolean function	Microoperation	Name
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow F \wedge B$	AND
$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Ex-OR.
$F_7 = x + y$	$F \leftarrow A \vee B$	OR.
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR.
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Ex-NOR.
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B.
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = x'$	$F \leftarrow \bar{A}$	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 all to 1's

Basic Logic Circuit



Functional Table

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

Date
Page No.

⇒ Shift Microoperation :-

→ Shift Microoperation are used for serial transfer.
→ The content of a register can be shifted to the left or right.
→ At the same time that the bits are shifted, the flip-flop recieves its binary from the serial input.
→ During a shift-left operation the serial input is enters into rightmost position and vice versa.

There are three types of shift :⇒ Logical shift
→ Arithmetic shift
→ Circular Shift

Logical Shift :- A Logical shift is one that transfers 0 through the serial input

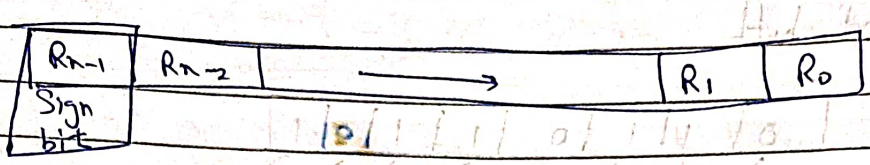
$R_1 \leftarrow shl\ R_1$ shl is logical shift left
$R_2 \leftarrow shr\ R_2$ shr is logical shift right

Circular shift :- It circulates the bits of register around the two ends without loss of information.
→ It is accomplished by connecting the serial o/p of the register to the serial input.

cil ⟵ circular shift left
cir circular shift right

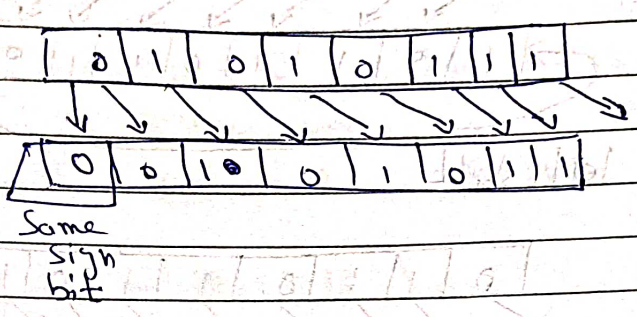
Arithmetic Shift :-
→ An arithmetic shift - left multiplies a signed binary no. by 2
→ An " " right divides " " " " " "
→ Arithmetic shift must leave the sign bit unchanged.
∴ The sign of the number remains the same when divided by 2.

Date
Page No.



ashr →

eg



ashl

→ Arithmetic shift-left inserts a 0 into R_0 and shifts all bits to the left

→ The initial bit R_{n-1} is lost and replaced by the bit R_{n-2}. In case of sign reversal occurs if the Bit R_{n-1} and R_{n-2} are different.

→ This condition is called overflow.

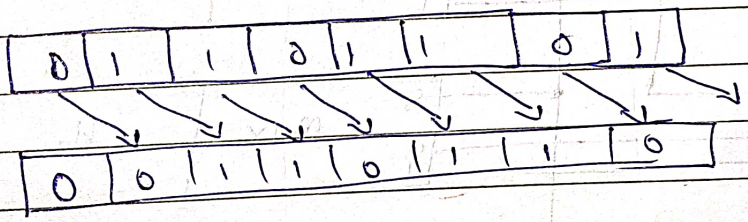
→ To detect this a V_s flip flop is used.

$$V_s = R_{n-1} \oplus R_{n-2}$$

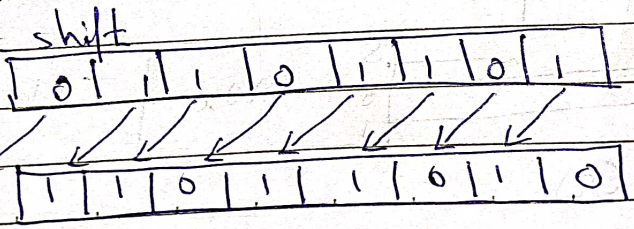
if $V_s = 0$ there is no overflow

$V_s = 1$ There is a overflow and sign is reversed.

eg :- Logical right shift

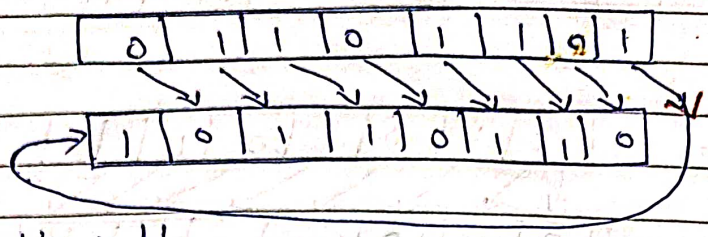


logical left shift



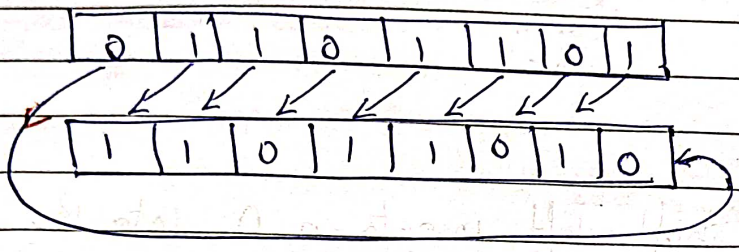
Date
Page No.

Circular right shift



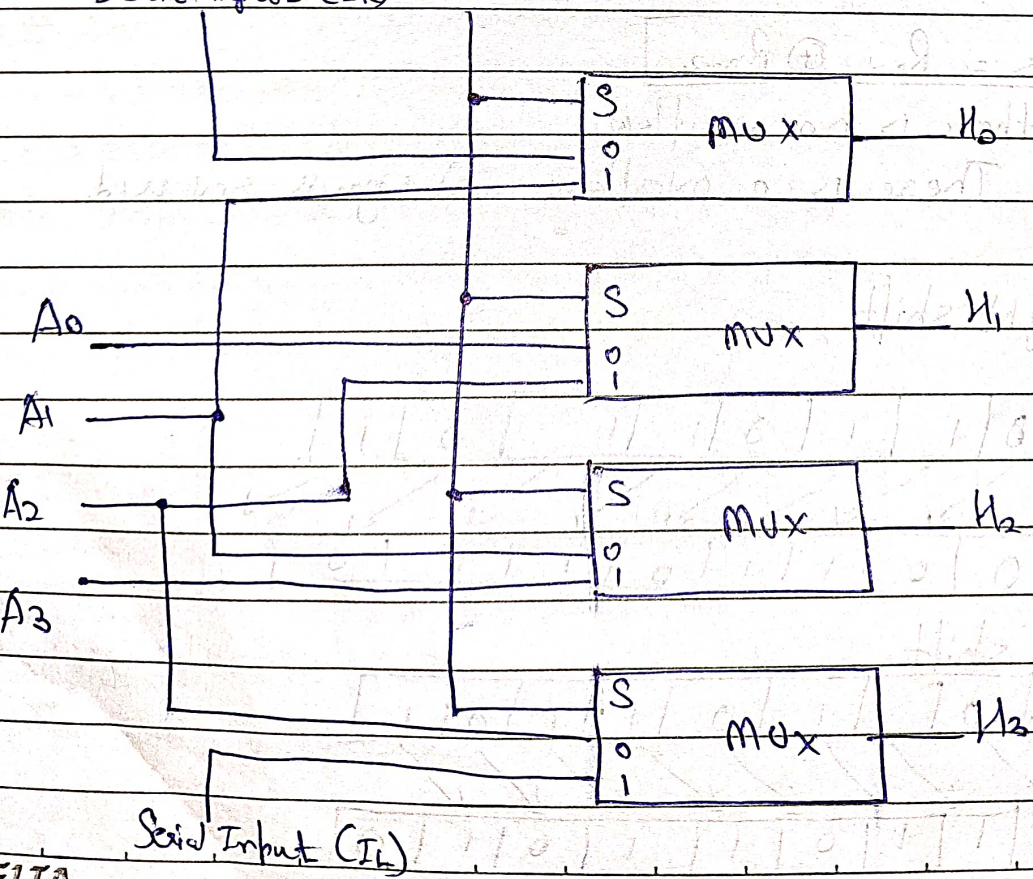
bina

Circular left shift



Hardware implementation :-

S	O_0	O_1	O_2	O_3
0 → R	I_R	A_0	A_1	A_2
1 → L	A_1	A_2	A_3	I_L



Date
Page No.

COA

Unit-2

⟹ Instruction Codes

A group of bits (1010....) -> It tells to the computer to perform some specific tasks.

-> Every computer has their own instruction code format

-> It is divided into two parts → opcode
                                 ↳ address

Memory

Instruction program
operand data

4096 × 16

$2^{12} \times 2^{4}$

12 → address part

4 → opcode

opcode ⇒ operation
ADD, SUB, MUL

operand ⇒ A, B, C

Instruction format

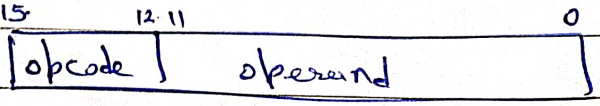
tells the address of the operand on which the operation is going to be performed

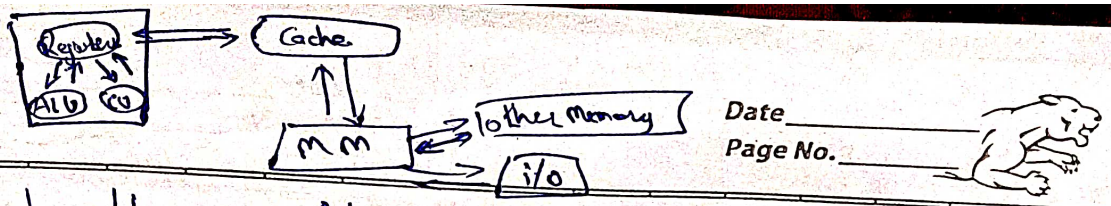
Types of Instruction Codes:-

1. Memory Reference Instruction
2. Register Reference Instruction
3. I/O Reference Instruction

Based on address the Instruction Codes can be of three types -

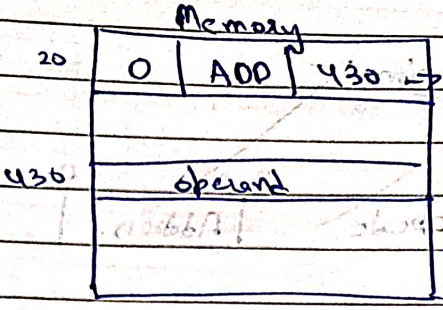
1. Immediate address:- actual operand is present in the address.



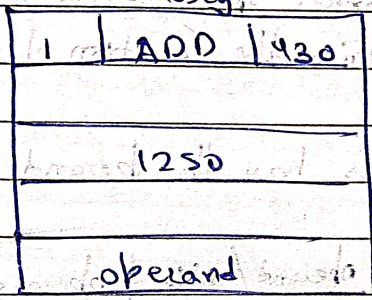


Date
Page No.

② Direct address :- Address of operand present (effective address)



③ Indirect address :- Address of memory word in which address of operand is found.



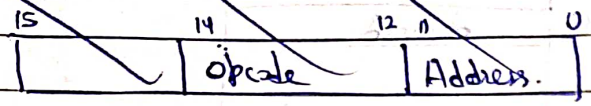
⇒ Computer Registers :- A Register is a very small amount of very fast memory that is built into the CPU in order to speed up its operation by providing quick access to commonly used values.

Register Symbol	Register Name	Number of Bits	Description
AC	Accumulator	16	Processor Register
DR	Data Register	16	Hold memory Data
TR	Temporary Register	16	Holds Temporary Data
IR	Instruction Register	16	Holds Instruction Data/code
AR	Address Register	12	Holds Memory Address
PC	Program Counter	12	Holds Add. of next instruction
INPR	Input Register	8	Holds input data
OUTR	Output Register	8	Holds output data

Date
Page No.

⇒ Computer Instruction :-

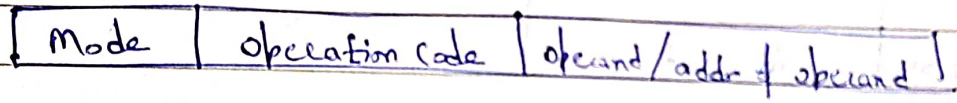
(1) Memory Reference Instruction :-



Computer Instructions are a set of machine language instructions that a particular processor understands and executes. A computer performs tasks on the basis of the instruction provided.

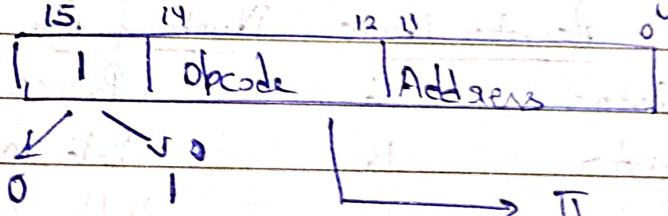
An instruction comprises of :-

- The operation code (opcode) which specifies the operation to be performed.
- The Address field which contains the location of the operand i.e. register or memory location.
- The Mode field which specifies how the operand will be located



(1) Memory-Reference instruction :-

In Memory-reference instruction, 12 bits of memory is used to specify an address and one bit to specify the addressing mode 'I'.

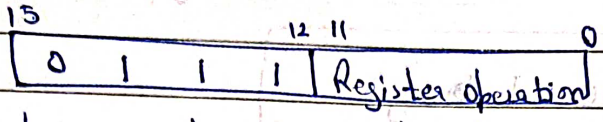


→ The opcode of an instruction refers to a group of bits that define arithmetic and logic operations such as add, subtract, multiply, shift and compliment.

Date
Page No.

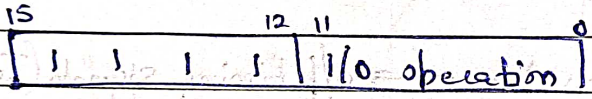
② Register - reference instruction :-

The Register - reference instruction are represented by the Opcode 111 with a 0 in the leftmost bit of the instruction



A Register - reference instruction specifies an operation on or a test of the AC (Accumulator) Register.

③ Input - Output Instruction :- Just like the Reference register instruction, an Input - Output instruction does not need a reference to memory and is recognized by the operation code 111 with a 1 in the leftmost bit. The remaining 12 bits are used to specify the type of input-output operation or test performed.



Note :- The three operation code bits in positions 12 through 14 should be equal to 111. Otherwise, the instruction is a memory - reference type, and the bit in position 15 is taken as the addressing mode I.
:- When the three operation code bits are equal to 111, control unit inspects the bit in position 15. If the bit is 0, the instruction is a register - reference type. Otherwise, the instruction is an input-type having bit 1 at position 15.

Date
Page No.

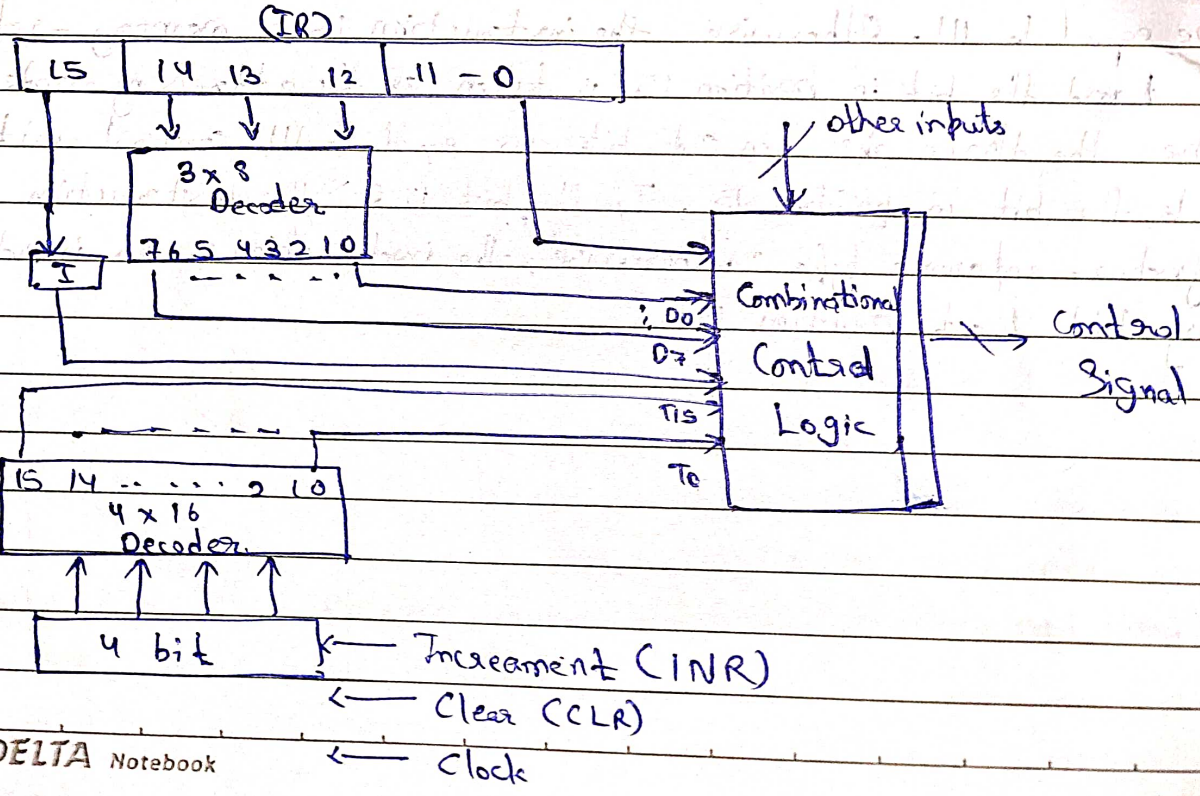
⇒ Timing and Control :—

Components of CU ⟹ 2 decoders
⟹ 1 Seq counter
⟹ 1 Control logic gates

IR (16 bit Reg)
- 1 bit: I = indirect, 0 = direct
- (12-14) bits: opcode → 3 x 8 decoder → 8 o/p of decoder $D_0 - D_7$ → Control logic gates
- (0-11) bits: address of the operand → applied to control logic gates

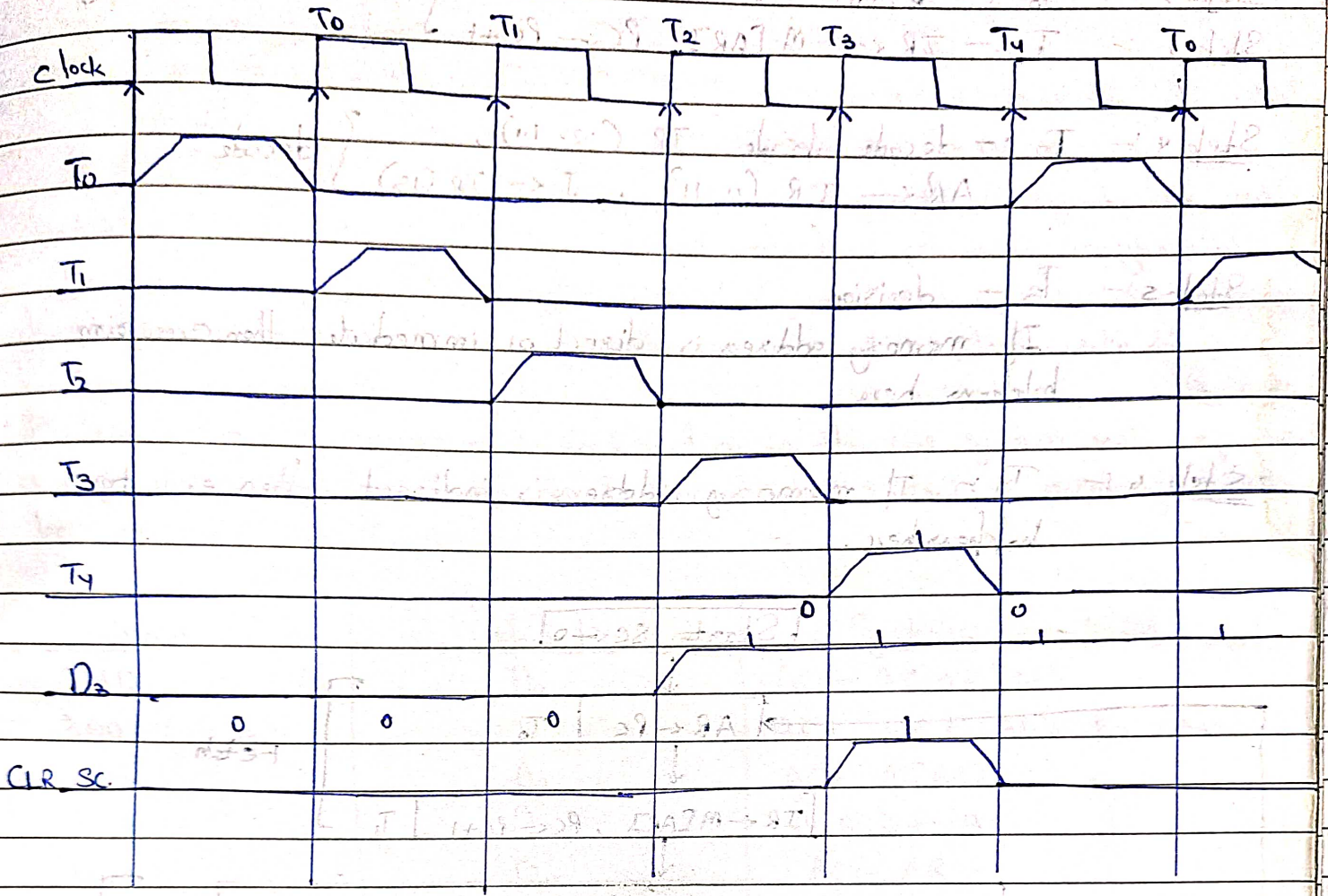
4 bit sequence counter ⟶ o/p timing signals ($T_0 - T_{15}$)

Control unit of basic Computer :—



Date
Page No.

Control timing Signals :-



$D_3 T_4 : SC \leftarrow 0$

⇒ Instruction Cycle :-

Program stored in memory unit

↓

Sequence of inst ⟷ IR

↓

Instruction cycle

Fetch → decode → Execute

Decision

IR → I=0 direct

I=1 Indirect

↓

decision.

Date
Page No.

Step-1 :- Start SC ← 0

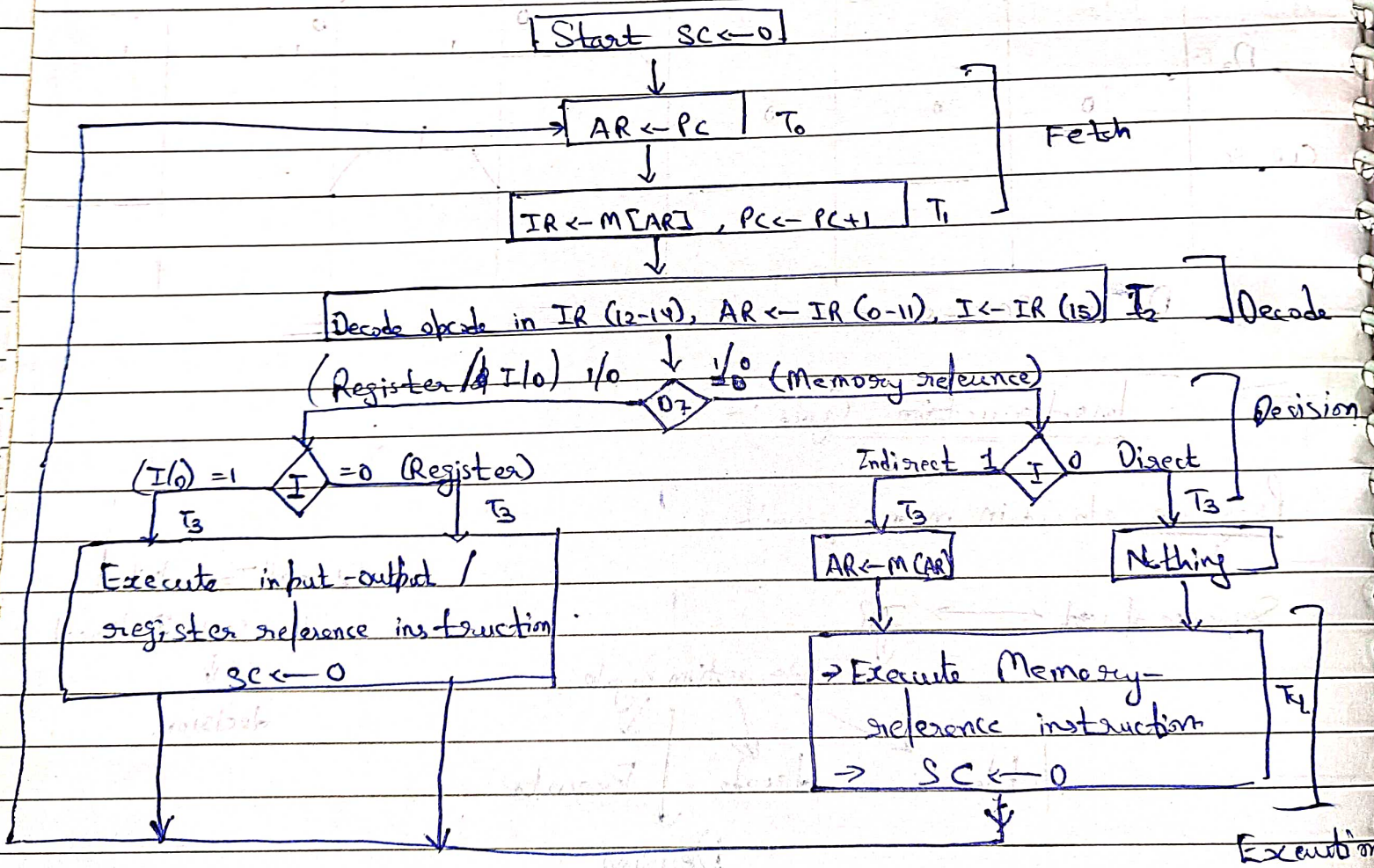
Step-2 :- T_0 :- AR ← PC

Step-3 :- T_1 :- IR ← M[AR], PC ← PC+1 } fetch from memory

Step-4 :- T_2 :- decode opcode IR (12-14), AR ← IR (0-11), I ← IR (15) } decode

Step-5 :- T_3 :- decision
If memory address is direct or immediate then execution happens here

Step-6 :- T_4 :- If memory address is indirect, then execution happens here.



Date
Page No.

⇒ Memory reference instruction :-

-> There are seven memory-reference instruction.
-> The Decoded o/p differ i=0, 1, 2, 3, 4, 5, 6 and 6 from the operation decoder that belongs 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 timing signal T_2 when I=0 or during timing signal T_3 when I=1.
-> The execution of the memory-reference instruction starts with timing signal T_4.
-> The actual execution of the instruction in the bus system will require a sequence of microoperations. This is because data stored in memory cannot be processed directly.

Symbol	Operation decoder	Symbolic description
AND	D_0	AC ← AC ∧ M[AR]
ADD	D_1	AC ← AC + M[AR], E ← Cout
LDA	D_2	AC ← M[AR]
STA	D_3	M[AR] ← AC
BUN	D_4	PC ← AR
BSA	D_5	M[AR] ← PC, PC ← AR+1
ISZ	D_6	M[AR] ← M[AR]+1 if M[AR]+1 = 0, then PC ← PC+1

① AND to AC :-

$D_0 T_4$: DR ← M[AR]
$D_0 T_5$: AC ← AC ∧ DR, SC ← 0

② ADD to AC :-

$D_1 T_4$: DR ← M[AR]
$D_1 T_5$: AC ← AC + DR, E ← Cout, SC ← 0

Date
Page No.

(3) LDA to AC :- $D_2 T_4$: DR <- M[AR]
$D_2 T_5$: AC <- DR, SC <- 0

(4) STA AC :- $D_3 T_4$: M[AR] <- AC, SC <- 0

(5) BUN : Branch Unconditional

$D_4 T_4$: PC <- AR, SC <- 0

(6) BSA :- Branch and Save Return address

$D_5 T_4$: M[AR] <- PC, AR <- AR+1
$D_5 T_5$: PC <- AR, SC <- 0

(7) ISZ :- Increment and skip if Zero

$D_6 T_4$: DR <- M[AR]
$D_6 T_5$: DR <- DR+1
$D_6 T_6$: M[AR] <- DR, if (DR=0) then (PC <- PC+1), SC <- 0

=> Input-Output instruction :-

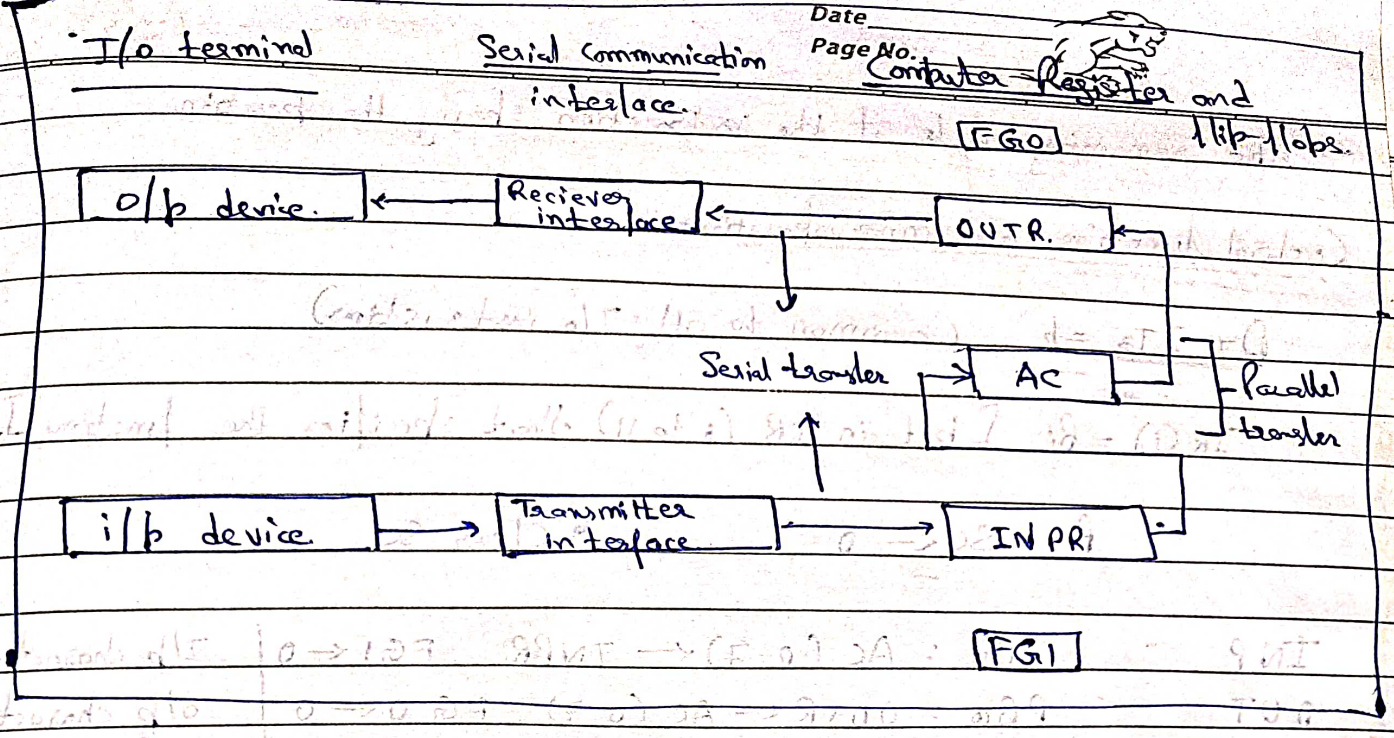
-> Instruction and data stored in memory must come from some input device.
-> Computation results must be transmitted to the user through some output device.

I/o terminal

Serial communication interface.

Date
Page No.

Computer Register and flip flops.



INPR => Input-register - consists eight bits and holds an alphanumeric input information

FG1 :- 1-bit input flag is a control flip-flop.
= 1, when new information is available in the input
= 0, when the information is accepted by the computer.
-> Initially it is cleared to 0.

OUTR :- Output register - consists eight bits

FGO :- 1-bit ~~register~~ output flag is a control flip-flop
= 0, do not load new thing.
= 1, information from AC is transferred in parallel to OUTR.

-> Operation code for i/o instructions is 1111
$D_7 = 111$
$I = 1$

DELTA Notebook

Date
Page No.

→ The remaining bits of the instruction specify the operation.

Control functions and microoperations :-

$D_7 I T_3 = p$ (common to all I/o instructions)

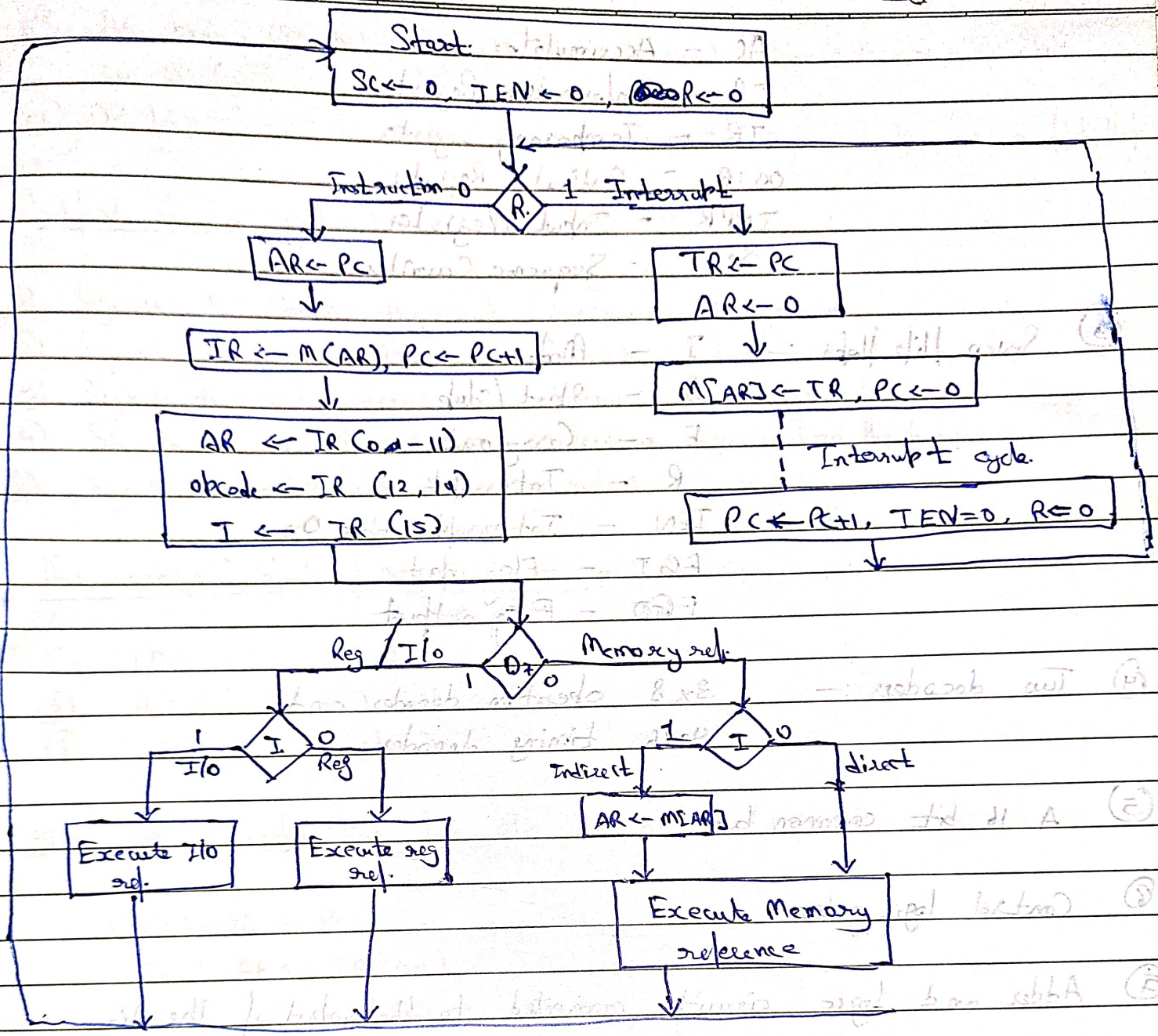
$IR(i) = B_i$ [bit in IR (6 to 11) that specifies the functions].

$p:$ $SC \leftarrow 0$ • clear SC

INP	PB_{11} : $AC(0-7) \leftarrow INPR$, $FGI \leftarrow 0$	I/p character
OUT	PB_{10} : $OUTR \leftarrow AC(0-7)$, $FGO \leftarrow 0$	o/p character
SKI	PB_9 : if $(FGI = 1)$ then $(PC \leftarrow PC+1)$	Skip on i/p flag
SKO	PB_8 : if $(FGO = 1)$ then $(PC \leftarrow PC+1)$	Skip on o/p flag
ION	PB_7 : $IEN \leftarrow 1$	Interrupt enable on
IOF	PB_6 : $IEN \leftarrow 0$	Interrupt enable off

⟹ Complete Computer Description ← The final flowchart of the instruction cycle, including the interrupt cycle for the basic computer.

Date
Page No.



⇒ Design of Basic Computer

H/w components :—

① Memory unit with 4096 words of 16 bits each.
② Nine registers :— AR – address register
:— PC – Program Counter
:— DR – Data Register

Date
Page No.

AC - Accumulator
IR - Information Register
TR - Temporary register
OUTR - Output Register
INPR - Input Register
SC - Sequence Counter

(3) Seven flip flops :- I - Mode
S - Start/Stop
E - Carry out
R - Interrupt
IEN - Interrupt enable On
FGI - Flag input
FGO - Flag output

(4) Two decoders :- 3x8 operation decoder and
4x16 timing decoder

(5) A 16 bit common bus

(6) Control logic gates

(7) Adder and logic circuit connected to the input of the AC.

Inputs to the circuit comes from :-

① Two decoders

② I flip flop

③ 0 - 11 bits of IR

Date
Page No.

④ AC bits (0-15) to check if AC=0 and to detect the sign bit in AC(15)
⑤ DR bits (0-15) to check if DR=0, and the values of the seven flip flop

Output of the control design Circuit are :-

① Signals to control the input of the nine registers
② Signals to control the read and write inputs of memory
③ Signals to set, clear, or complement the flip-flops
④ Signals for S_2, S_1 and S_0 to select a register from the bus
⑤ Signals to control the AC adder and logic circuit.

The control inputs of registers are :-

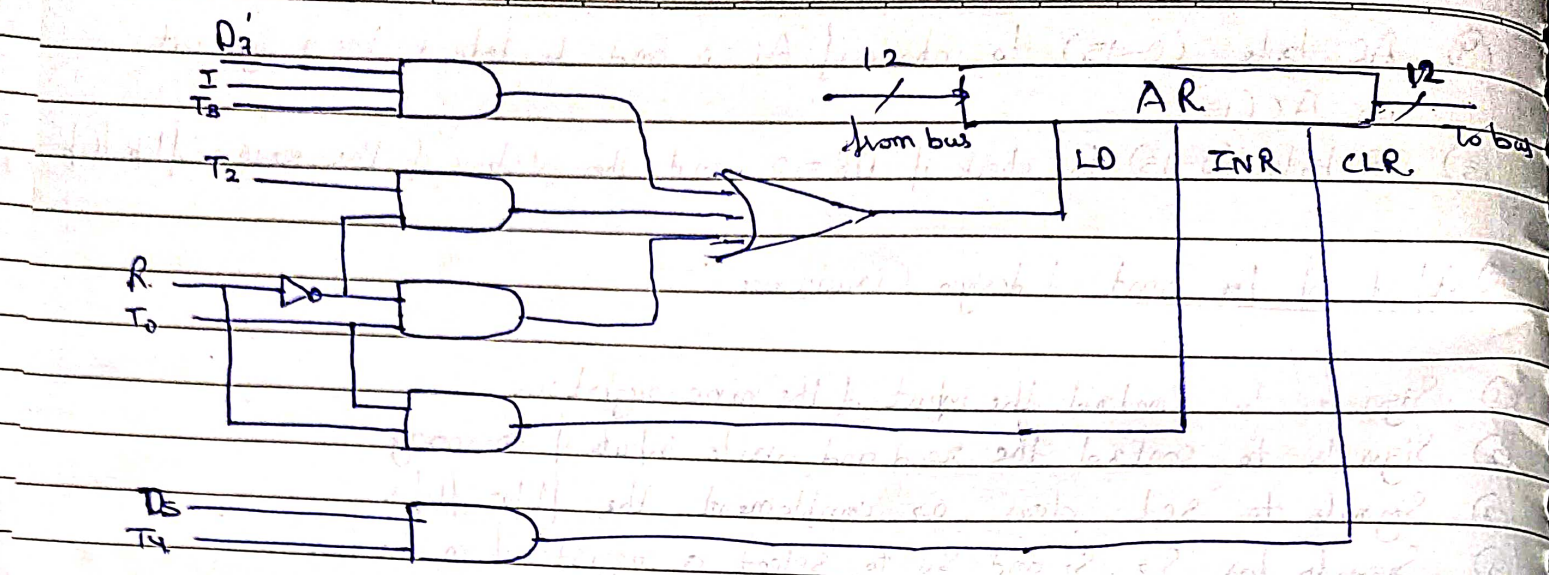
① LDA (load)
② INR (Increament)
③ CLR (clear)

Eg:- The microoperations for the AR are as follows:-

$R'T_0$: $AR \leftarrow PC$
$R'T_2$: $AR \leftarrow IR(0-11)$
$D'_7 I T_3$: $AR \leftarrow M[AR]$
RT_0 : $AR \leftarrow 0$
$D_5 T_4$: $AR \leftarrow AR+1$

The control functions can be combined into three Boolean expression as follows :-

$LD(AR) = R'T_0 + R'T_2 + D'_7 I T_3$
$CLR(AR) = RT_0$
$INR(AR) = D_5 T_4$



Control of Single Flip Flops :-

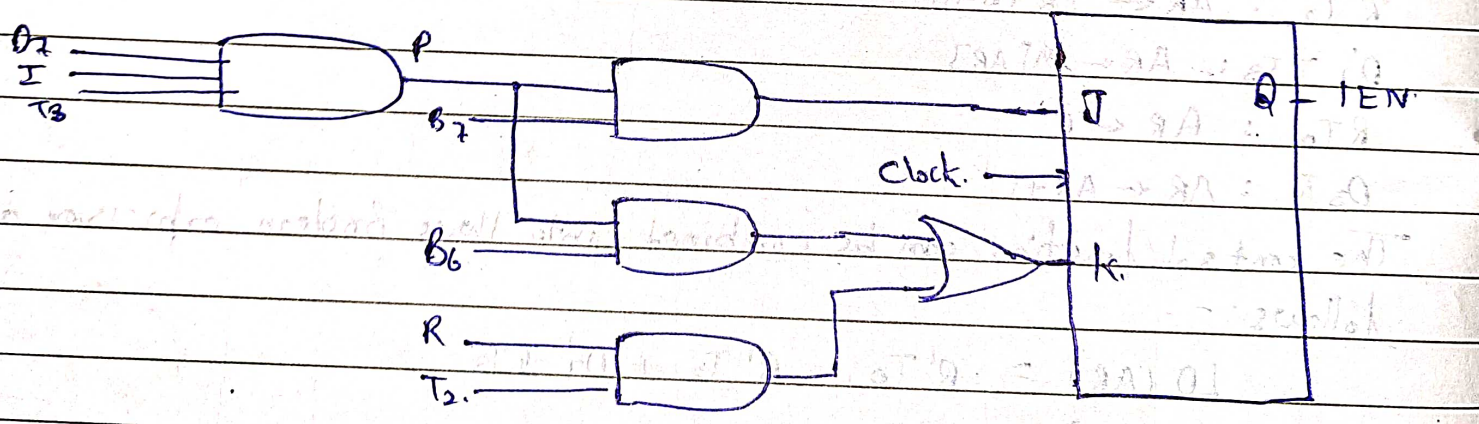
eg :- IEN may change as a result of the two instructions ION and IOF:

$PB_7 : IEN \leftarrow 1$ where $P = D_7 I T_3$

$PB_6 : IEN \leftarrow 0$

and B_7, B_6 are bits 7 and 6 of IR respectively. At the end of the interrupt cycle IEN is cleared to 0

$RT_2 : IEN \leftarrow 0$



Date
Page No.

Encoder for Bus Selection Circuit :-

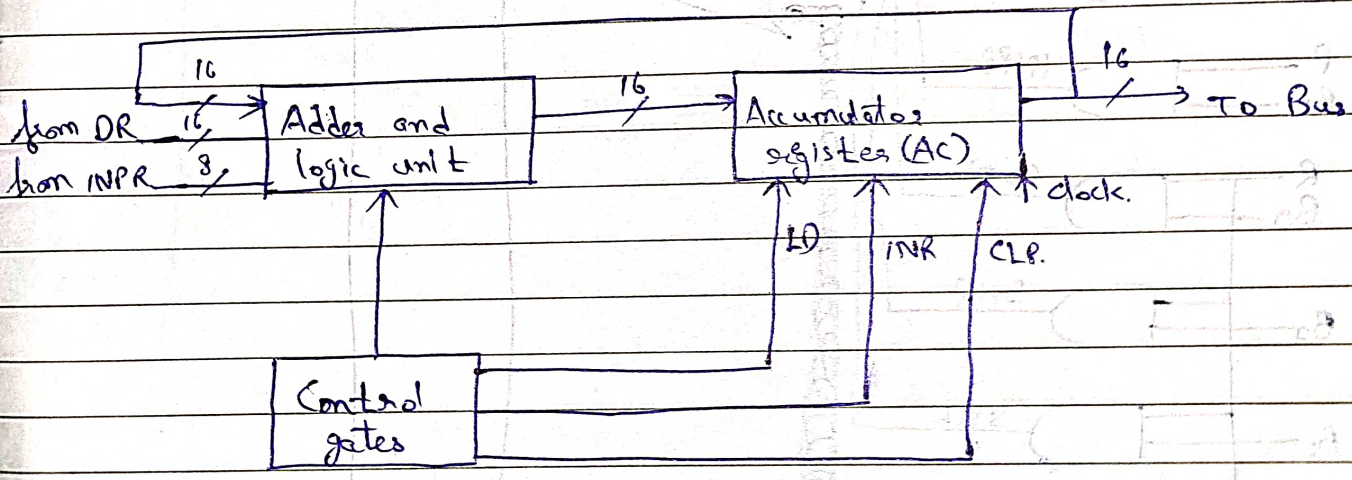
Inputs							Output			Register selected for bus.
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

$S_0 = x_1 + x_3 + x_5 + x_7$

$S_1 = x_2 + x_3 + x_6 + x_7$

$S_2 = x_4 + x_5 + x_6 + x_7$

Design of Accumulator Logic :-

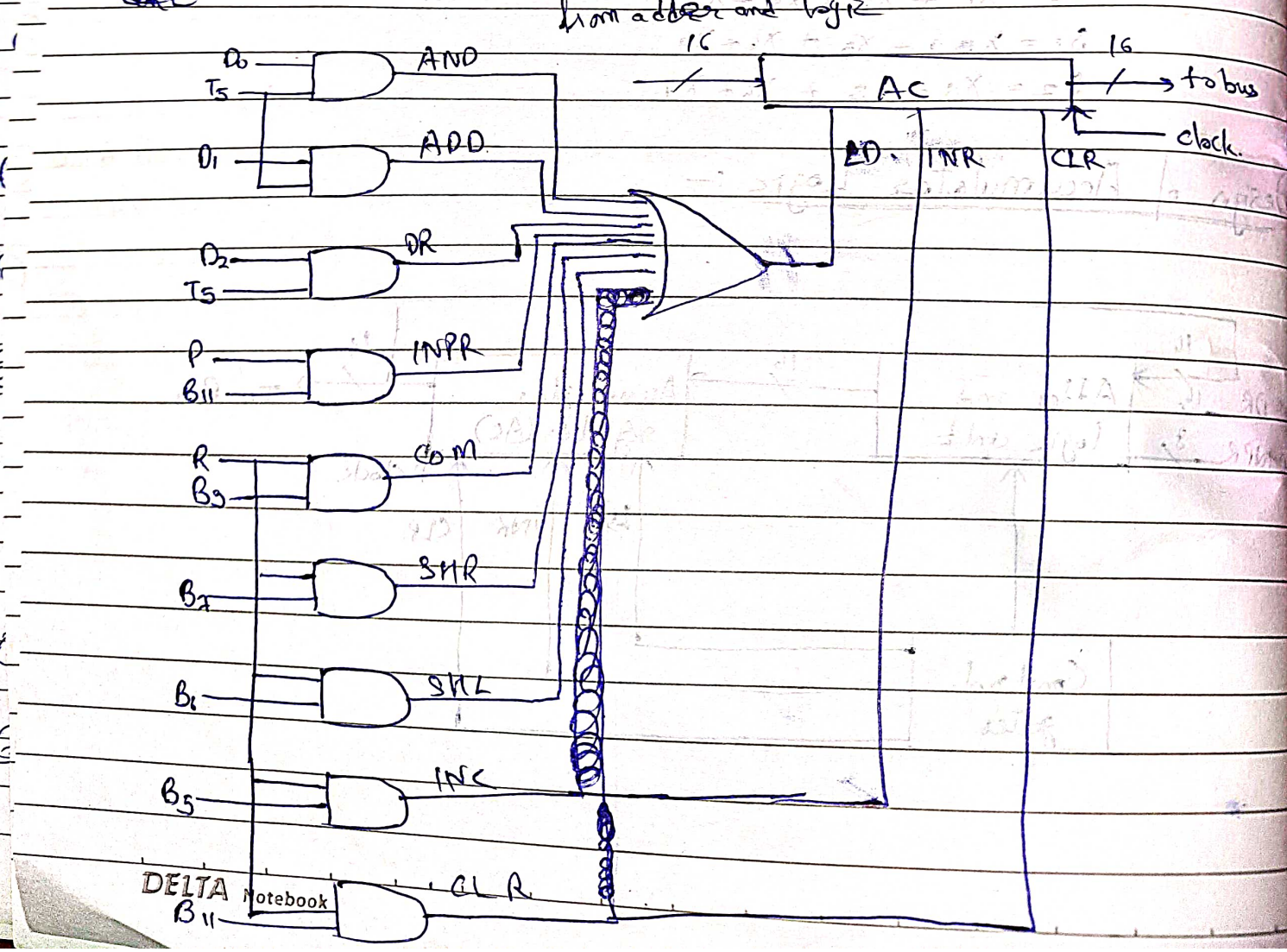


Date
Page No.

Control of AC register :-

AND	$D_0T_5 : AC \leftarrow AC \wedge OR$	Memory reference
ADD	$D_1T_5 : AC \leftarrow AC + OR$	
Load	$D_2T_5 : AC \leftarrow OR$	
Input register	$PB_{11} : AC(0-7) \leftarrow INPR$	I/O Reference
Complement	$RB_9 : AC \leftarrow \overline{AC}$	Register Reference
Shift	$RB_7 : AC \leftarrow shr\ AC,\ AC(15) \leftarrow E$	
Shift	$RB_6 : AC \leftarrow shl\ AC,\ AC(0) \leftarrow E$	
Clear	$RB_{11} : AC \leftarrow 0$	
Increment	$RB_5 : AC \leftarrow AC + 1$	

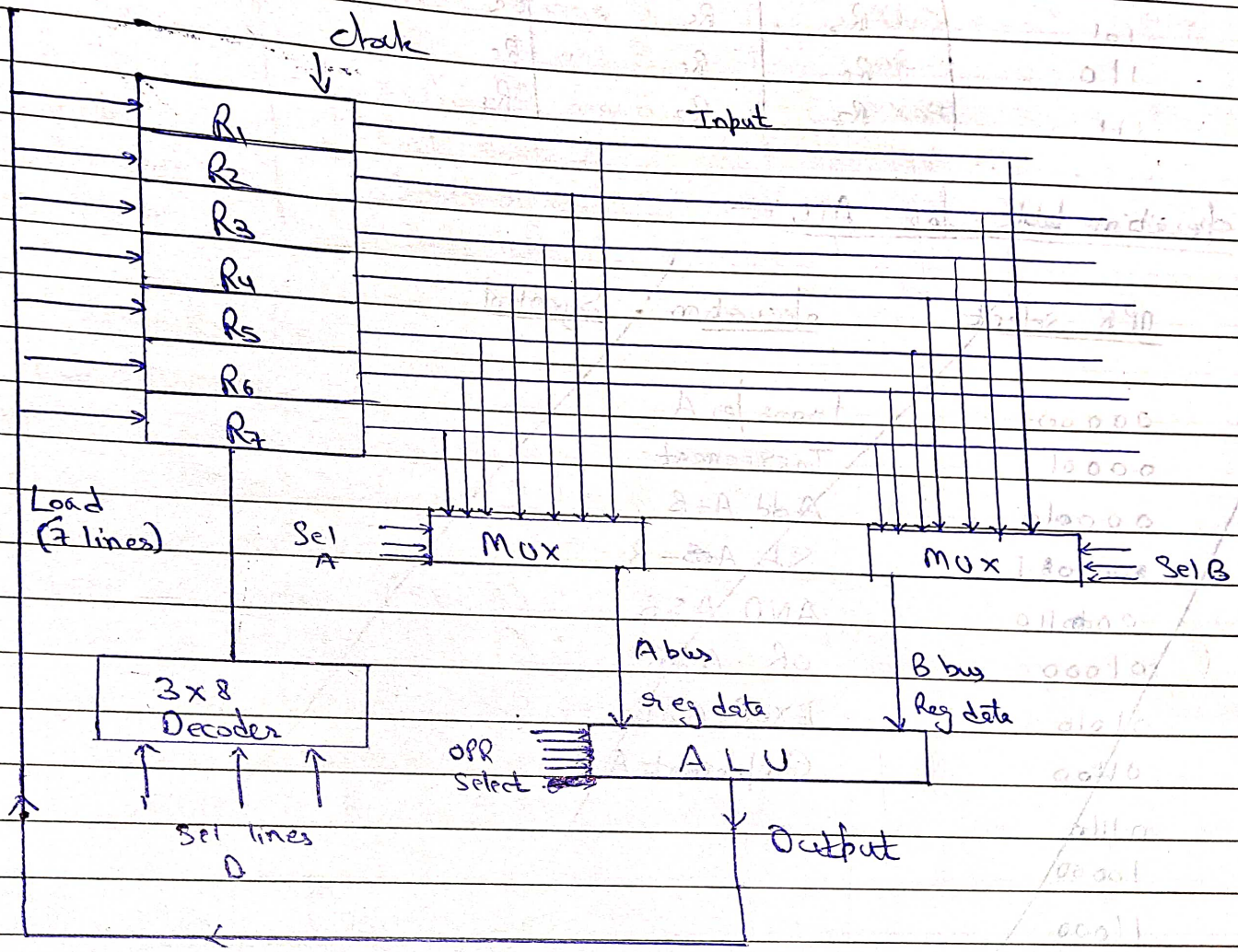
Ckt



DELTA Notebook

General Register Organization

- Important Component of CPU →
  1. Storage Components → Registers, Flip/flops
  2. Execution Components → ALU
  3. Transfer Component → Bus
  4. Control Component → Control Unit.



Date
Page No.

Binary Code	SelA	SelB	SelD
000	-	-	None
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

Operation table for ALU :—

OPR select	operation	Symbol
00000	transfer A	
00001	Increament	
00010	Add A+B	
00011	Sub A-B	
00110	AND A&B	
01000	OR A&B	
01010	EXOR A&B	
01100	Complement A	
01110		
10000		
11000		

Date
Page No.

OPR Select	Operation	Symbol
00000	Transfer A	TSF A
00001	Increament A	INC A
00010	Add A+B	ADD
00101	Subtract A-B	SUB
00110	Decreament A	DEC A
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COM A
10000	Shift right A	SHR A
11000	Shift left A	SHL A

eg:- $R_1 \leftarrow R_2 + R_3$

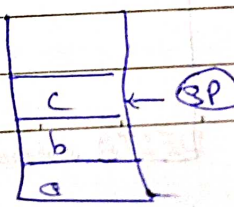
Control word

SelA	SelB	SelD	OPR

1) MUX A Selector (SELA) : Bus A $\leftarrow R_2$
2) MUX B Selector (SELB) : Bus B $\leftarrow R_3$
3) ALU operation $\leftarrow$ ALU to ADD (A+B)
4) Decoder destination selector (SELD) : $R_1 \leftarrow$ ADD (output).

Stack Organization :-

→ Stack is a storage device for storing information in manner LIFO.
→ Stack is a memory unit with an address register called stack pointer (SP)
→ SP always points to the Top of the stack.



Date
Page No.

→ There are two operations of the stack.
PUSH :- Insertion
POP :- Deletion.

⇒ 2 types of stack in Memory :-

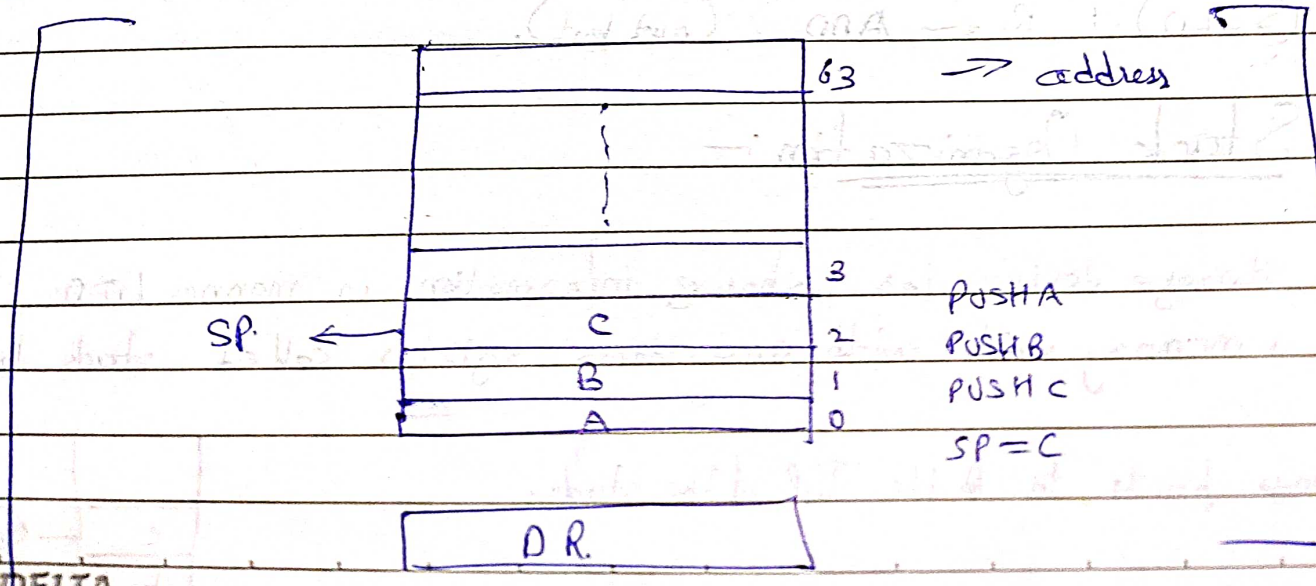
① Register Stack (stack depth: Limited)
② Memory Stack (stack depth: flexible)

① Register Stack :-

Push operation :- SP ← SP + 1 : Increament SP
M[SP] ← DR : Write to the stack
if (SP = 0) then (FULL ← 1) : check if stack is full
EMPTY ← 0 : Mark not empty.

Pop operation :-

DR ← M[SP] : Read item from stack
SP ← SP - 1 : Decreament SP
if (SP = 0) then (Empty ← 1) : check if stack is empty
FULL ← 0 ; Mark not full



Date
Page No.

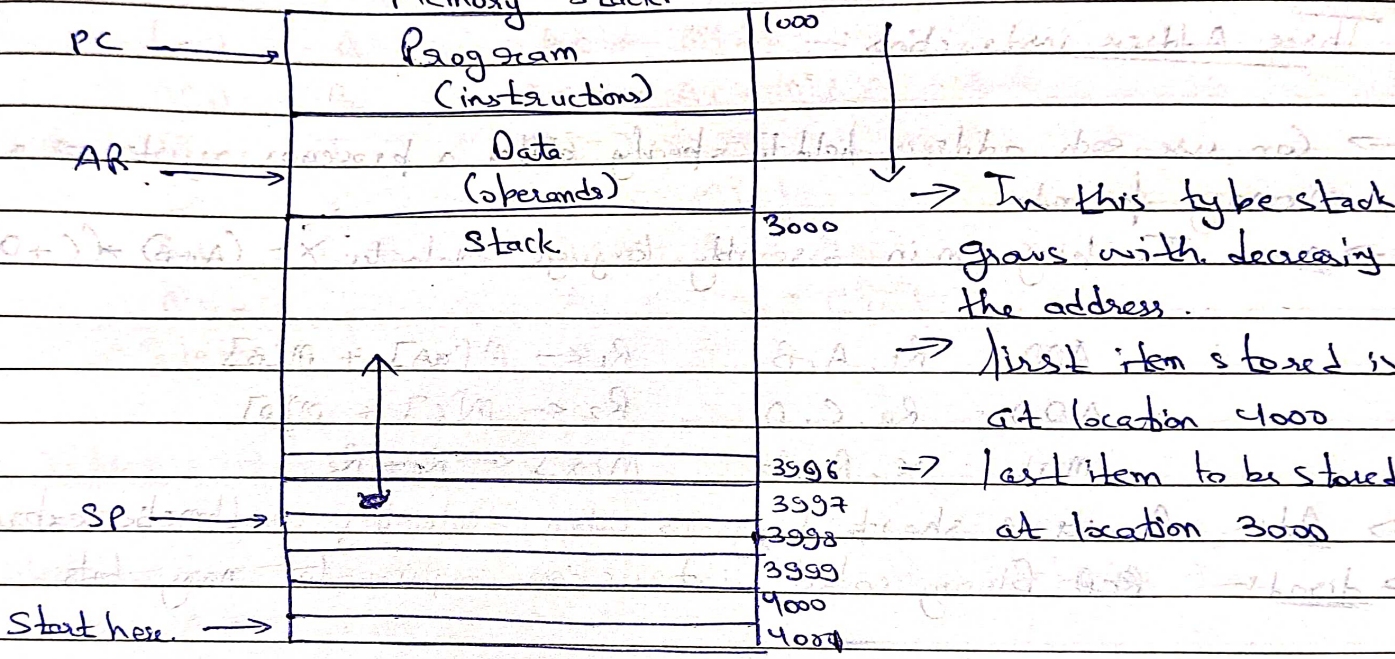
② Memory stack :-

Push operation :-

SP ← SP - 1
M[SP] ← DR

Pop operation :-

DR ← M[SP]
SP ← SP + 1



⇒ **Instruction formats :-** It defines the layout of the bits of an instruction. An instruction format must include an opt code and zero or more operands in one of the addressing modes.

Common field format in instruction format is
① An opcode
② An address field
③ A mode field

Date
Page No.

Instruction formats can be categorized with respect to the operand fields in the instructions.

1. Three Address instructions
2. Two Address instructions
3. One Address instruction
4. Zero Address instruction
5. RISC.

Three Address instructions :-

→ Can use each address field to specify either a processor register or a memory operand.

→ eg:- The program in assembly language evaluates $X = (A+B) * (C+D)$

ADD	R_1, A, B	$R_1 \leftarrow M[A] + M[B]$
ADD	R_2, C, D	$R_2 \leftarrow M[C] + M[D]$
MUL	X, R_1, R_2	$M[X] \leftarrow R_1 * R_2$

→ Adv:- Results in short programs when evaluating arithmetic expressions

→ disadv:- Binary coded instructions require too many bits.

Two address instruction :-

→ Most common in commercial computers.

→ eg:- $X = (A+B) * (C+D)$ is as follows :-

MOV	R_1, A	$R_1 \leftarrow M[A]$
ADD	R_1, B	$R_1 \leftarrow R_1 + M[B]$
MOV	R_2, C	$R_2 \leftarrow M[C]$
ADD	R_2, D	$R_2 \leftarrow R_2 + M[D]$
MUL	R_1, R_2	$R_1 \leftarrow R_1 * R_2$
MOV	X, R_1	$M[X] \leftarrow R_1$

Date
Page No.

One address Instruction :-

-> It uses an implied accumulator (AC) register for all data manipulation.
-> For multiplication and division, we need a second register.
-> However, here we will neglect the second register and assume that the AC contains the result of all operation.

eg :- $X = (A+B) * (C+D)$ is

Load	A	$AC \leftarrow M[A]$
ADD	B	$AC \leftarrow AC + M[B]$
STore	T	$M[T] \leftarrow AC$
Load	C	$AC \leftarrow M[C]$
Add	D	$AC \leftarrow AC + M[D]$
MUL	T	$AC \leftarrow AC * M[T]$
Store	X	$M[X] \leftarrow AC$

Zero address instructions :-

-> A stack organised computer does not use an address field for the instructions ADD and MUL.
-> The PUSH and POP instructions, however, need an address field to specify the operand that communicates with the stack.
-> To evaluate arithmetic expression, in a stack computer, it is necessary to convert the expression into reverse polish notation.

eg :- $X = (A+B) * (C+D)$

RP notation : $X = AB+ CD+ *$

PUSH	A	$TOS \leftarrow M[A]$
PUSH	B	$TOS \leftarrow M[B]$
ADD		$TOS \leftarrow A+B$
PUSH	C	$TOS \leftarrow M[C]$
PUSH	D	$TOS \leftarrow M[D]$
ADD		$TOS \leftarrow C+D$

DELTA Notebook

Date
Page No.

MUL		TOS ← (C+D) * (A+B)
POP	X	M[X] ← TOS

RISC Instruction :-

→ The instruction set of a typical RISC processor is restricted to the use of load and store ~~information~~ instruction when communicating b/w memory and CPU.

→ All other instructions are executed within the registers of the CPU without reffering to memory.

→ eg :- $X = (A+B) * (C+D)$

Load	R_1, A	$R_1 \leftarrow M[A]$
Load	R_2, B	$R_2 \leftarrow M[B]$
Load	R_3, C	$R_3 \leftarrow M[C]$
Load	R_4, D	$R_4 \leftarrow M[D]$
ADD	R_1, R_1, R_2	$R_1 \leftarrow R_1 + R_2$
ADD	R_3, R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL	R_1, R_1, R_3	$R_1 \leftarrow R_1 * R_2$
STORE	X, R_1	$M[X] \leftarrow R_1$

⟹ Addressing Modes :-

→ It specifies operations to be performed on data i.e. operands.

→ It shows how to select operand, how to calculate the effective address of an operand by the instruction stored in register.

→ It gives exact location of an operand.

operand → Memory location / constant / effective add. / direct or indirect add.

→ The way operand is chosen during program execution depends on. Addr modes

Date
Page No.

→ To give facilities as pointer, counters, for loop controls, indexing etc., program relocation.

Types of Addressing modes :-
- Implied
- Immediate
- Direct
- Indirect
- Spatial
  - Index
  - Auto increament
  - Auto decreament

① Implied Addressing Modes.

→ Operand location is already present in the instruction.

eg :- CMA (Complement of Accumulator)
ADD
INCA (Instruction accumulator)
↳ Operand location

② Immediate Addressing Modes :-

→ Value of the operand is available in the instruction

eg :- LOAD R_1, # 100
↳ Represents that the following value is of operand that is to be loaded in the given/specified register

$R_1 \leftarrow 100$
R_1 [100]

Adv :- Faster than other Addressing mode
disad :- Less flexibility
:- Changing of value of repeated.

Date
Page No.

③ Direct / Absolute Addressing Mode :-

Opcode	Address

↳ refers to operand

→ Address specifies the effective address of an operand.

eg :- MUL (X) * 5 Y | 10 |

MUL 10 * 5

eg ADD R1, 300

ADD R1, 20

eg :- LOAD R1, 100

300 | ~~20~~ |

→ the operand to be loaded in R_1, is present at memory location 100.

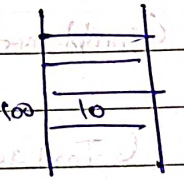
→ Fetch the operand from the memory.

→ Load the operand to Rg R_1

$R_1 \leftarrow M[100]$

| 10 |

R_1



④ Indirect Addressing Mode :-

Opcode	Address

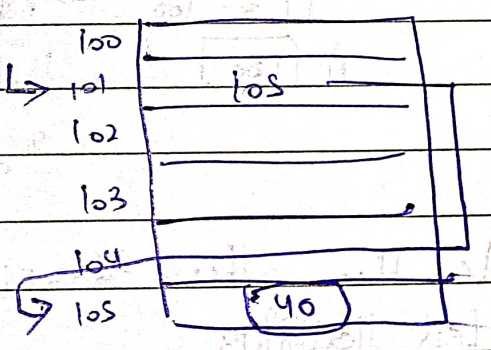
↳ Address of another address that refers to an operand.

eg :- LOAD 101

↳ address

101 | 105 |

105 | 40 | → operand.



→ Similar concept as pointers

→ 2 memory cycle.

→ 2 types → Memory indirect
→ Register indirect

DELTA Notebook

Date
Page No.

(a) Memory Indirect :-

LOAD Ri, (1000) → the paranthesis denote an indirect add.

(b) Register Indirect :-

LOAD Ri, (B) → register = B

→ Here reg contains the address of the operand.

⑤ Spatial Addressing Mode :- Location of the operand is specified with reference to a spatial memory location.

→ 3 types :- (a) Index addressing mode
(b) Auto - Increament Addressing mode
(c) Auto - Decreament Addressing mode

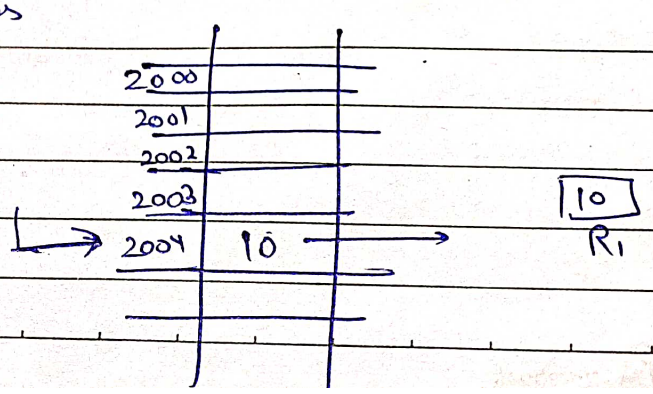
(a) Index Addressing Mode :-

→ The instruction specifies the base address.
→ Index is closed within "[ ]".
→ Index represents the displacement.
→ Displacement is the memory unit we need to move ahead from the base address in order to fetch the operand.

Effective address = Base Address + Offset (Index)

eg :- LOAD R1, 2000[x] → displacement / Index.
↓
base address

eg :- LOAD R1, 2000[4]



Date
Page No.

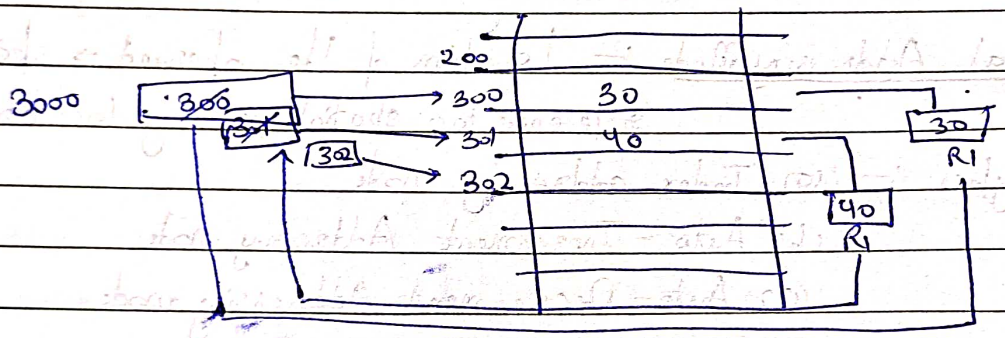
(b) Auto Increament Addressing Mode :- Address of the operand is in a register whose value is increament after fetching the operand from the address

Eg:- LOAD $[R_I]+, R_1$;

1) $R_1 \leftarrow M[R_I]$ → load the content of the memory address stored in reg R_I to Reg R_1.

2) $[R_I]++$ or $[R_I] \leftarrow [R_I]+1$

Increament the memory address present in $[R_I]$ Reg.



(c) Auto Decreament Addressing Mode :- Address of the operand is in a register whose value is decreamented after fetching the operand from the address.

Eg:- LOAD $[R_D]-, R_i$;

1) $R_i \leftarrow M[R_D]$

2) $[R_D]--$ or $[R_D] \leftarrow [R_D]-1$.

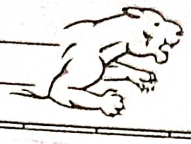
Relative add = PC + address

Index add = XR + address.

Q.

Date

Page No.



50	Load to AC Mode
51	Address = 100
52	Next Instruction
	:
	:
	:
	:
97	450
98	700
99	800
100	105
101	35
102	45
103	65
104	75
105	500
152	70

Memory

$R_1 = 98$

$XR = 3$

Addressing Mode	Effective Address	Content of E.A
Direct	100	105
Immediate	—	100
Indirect	105	500
Relative	152	70
Index	103	65
Register direct	—	98
Register Indirect	98	700
Auto increament	100	105
	101 →	35
Auto decrea ment	100	105
	99 →	800

Computer Instructions

Data Transfer Instruction	Data Manipulation Instruction	Program Control Instruction
Transfer of data from one location to another without changing the binary information content	Performs arithmetic logic and shift operations.	Provide decision making capabilities and change the path taken by the program when executed in the computer

Date
Page No.

⇒ Data Transfer Instructions :- Most common data Transfers are :-

① between Memory and Processor register
② b/w processor register and I/O
③ b/w processor registers themselves.

Typical data Transfer instructions :-

Name	Mnemonic	
Load	LD	To designate a transfer from memory to a process register
Store	ST	from processor register into memory
Move	MOV	from one register to another,
Exchange	XCH	Swapping b/w two registers or register and a memory word.
Input	IN	} A among processor registers and input or
Output	OUT	output terminals
PUSH	PUSH	} b/w processor registers and a memory stack.
POP	POP	

⇒ Data Manipulation Instructions :-

① Arithmetic Instructions
② Logical and bit manipulation instructions
③ Shift instructions

Date
Page No.

Arithmetic instructions :-

Name	Mnemonic	Variations :-
Increment	INC	data type = fixed point or floating point
Decrement	DEC	binary or decimal
Add	ADD	Single-precision or double-precision data
Subtract	SUB	
Multiply	MUL	
Divide	DIV	
Add with carry	ADC	ADD I | Integer
Subtract (borrow)	SUBB	ADD F | Floating point
Negate (2's comp)	NEG	ADD D | decimal numbers

Logical and bit manipulation Instructions :-

Name	Mnemonic	
Clear	CLR	
Complement	COM	
AND	ANL	// clear Selected bit
OR	ORL	// set Selected bit
Exclusive OR	XRL	// complement Selected bit
Clear Carry	CLRC	
Set Carry	SETC	
Complement Carry	COMC	
Enable Interrupt	EI	
Disable Interrupt	DI	

DELTA

Date
Page No.

Shift Instructions :-

Name	Mnemonic
Logical Shift right	SHR
Logical Shift left	SHL
Arithmetic Shift right	SHRA
Arithmetic Shift left	SHRLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

⟹ Program Control Instructions :-

Program control instructions specify conditions for altering the content of the program Counter.

Name	Mnemonic	
Branch	BR	One address Instruction, Conditional / Un Conditional
Jump	JMP	Conditional — True – PC – new effective address; False – PC – next address in sequence
Skip	SKP	Skip the next instruction
CALL	CALL	used with subroutines / Procedures
Return	RET	
Compare (by Subtraction)	CMP	Used to set conditions for branch instructions by updating status bit
Test (by add)	TST	

Date
Page No.

These instructions are used to transfer the program control-
- To jump from one memory location to any other memory location within a program.
- From one program to another program called as Subroutine.

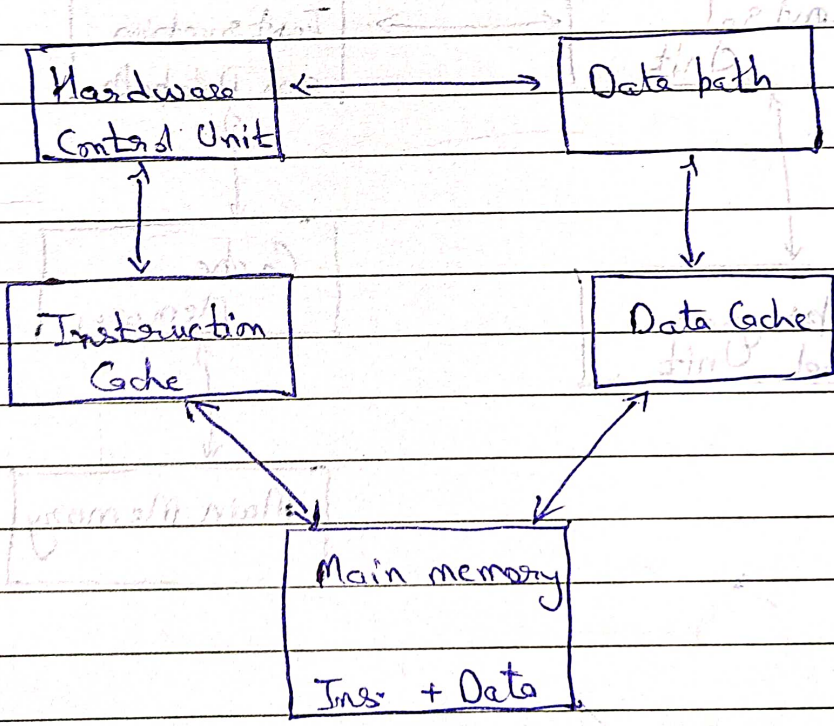
=> RISC (Reduced Instruction Set Computer)

Microprocessor

RISC | CISC | Special processor

RISC :- It is designed to reduce the execution time by simplifying the instruction set.
→ Each instruction requires only 1 clock cycle
↳ fetch, decode and execute

Architecture :- It uses highly optimized set of instruction
:- It is used in tablets, mobiles / smart phones, portable devices.



Date
Page No.

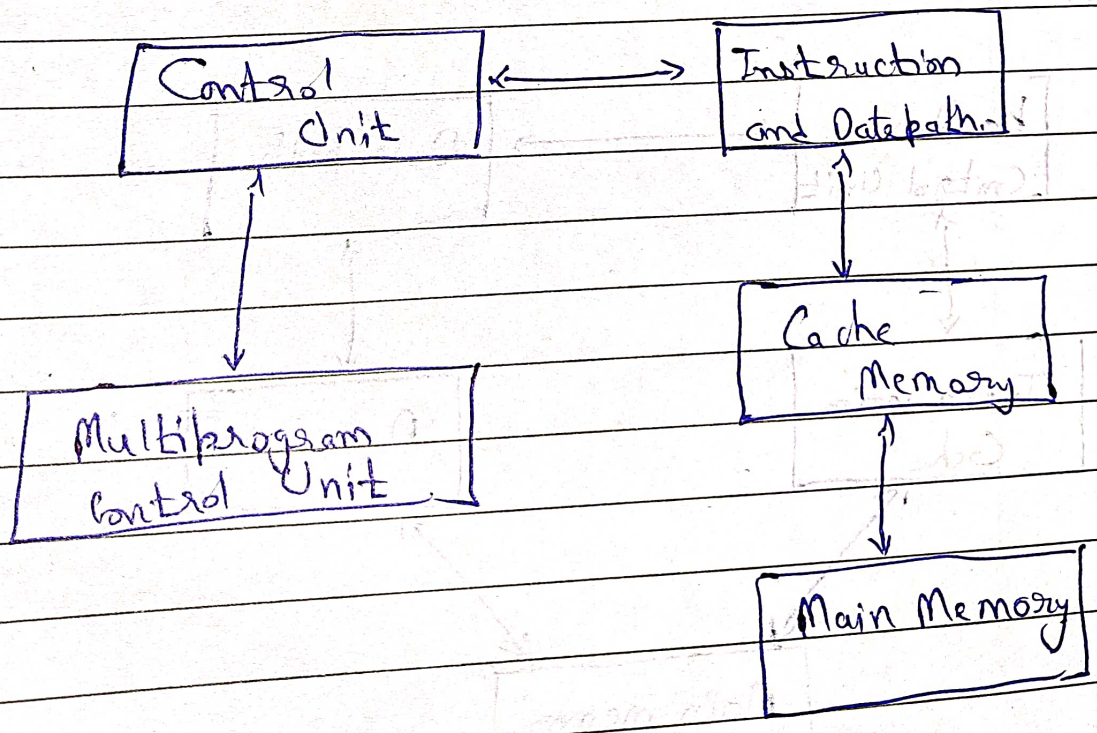
Characteristics :-

:- Relatively few instructions
:- Relatively few addressing modes.
:- Limited memory accesses to LOAD and SAVE instruction
:- All operations are done within the registers of the CPU.
:- Fixed length and easily decoded instruction format
:- Single cycle instruction execution
:- Hardwired rather than microprogrammed controlled.

⇒ CISC (Complex Instruction Set Computer)

-> To minimize the no of instruction program and ignoring the no. of cycles per instructions.
-> Used in desktops, laptops
-> Less Memory (RAM) is required to store instruction.

Architecture :-



Date
Page No.

Characteristics :-

:- Many addressing mode.
:- Large no. of instruction.
:- Variable length of instruction format.
:- Several cycles may be required to execute one instruction
:- Instruction decoding logic is complex.

RISC	CISC
=> Emphasis on Software	=> Emphasis on Hardware
=> Single clock cycle	-> Multiple Clock Cycle
=> Highly pipelined	-> Not or less pipelined
-> Register to Register	-> Memory to Memory
-> Fixed format instruction	-> Variable format instruction
-> Only Load / Store refers memory	-> Any inst. may refer memory.

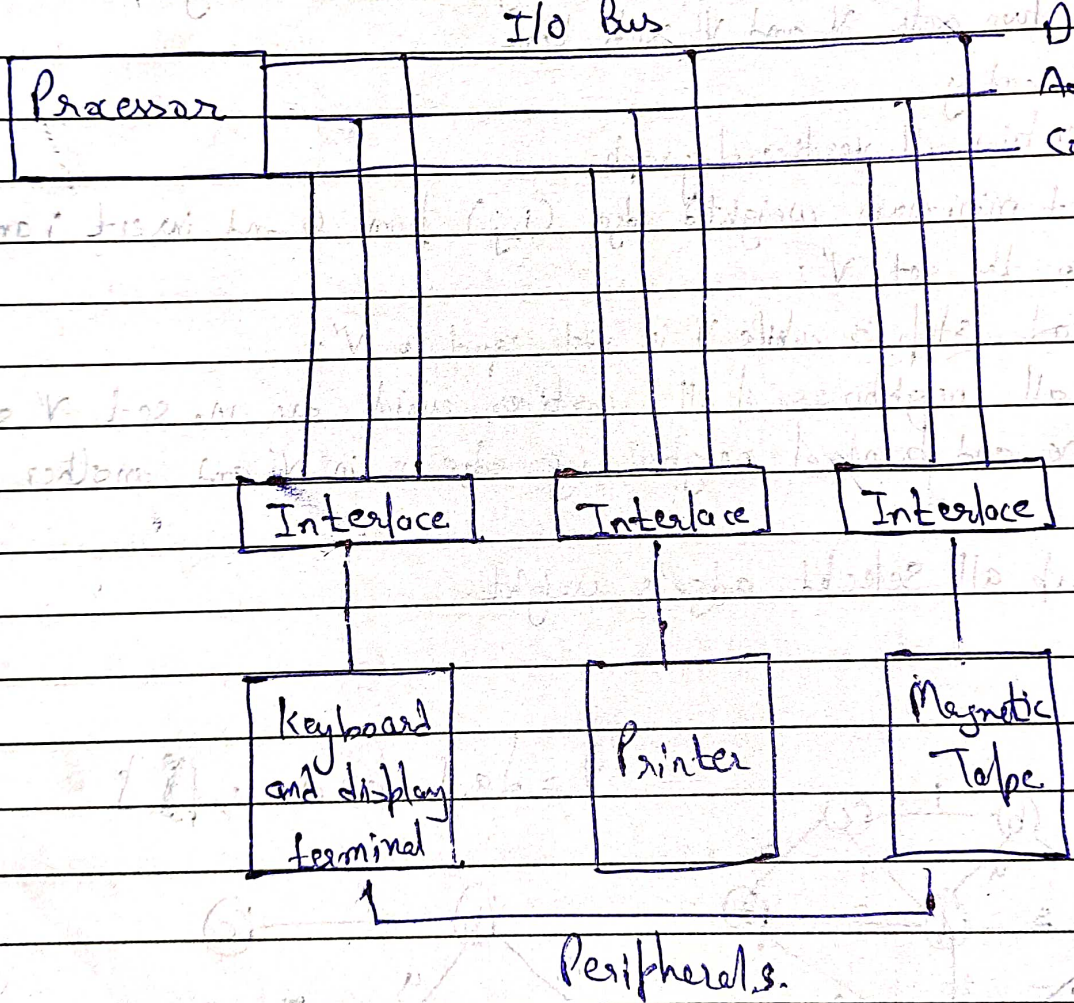
Date.....

Unit -4

I/O Device Interface :- The mode of transferring information between internal storage and external I/O devices is known as I/O interface or input/output interface. ~~The I/O module~~

:- I/O Module Decisions :-

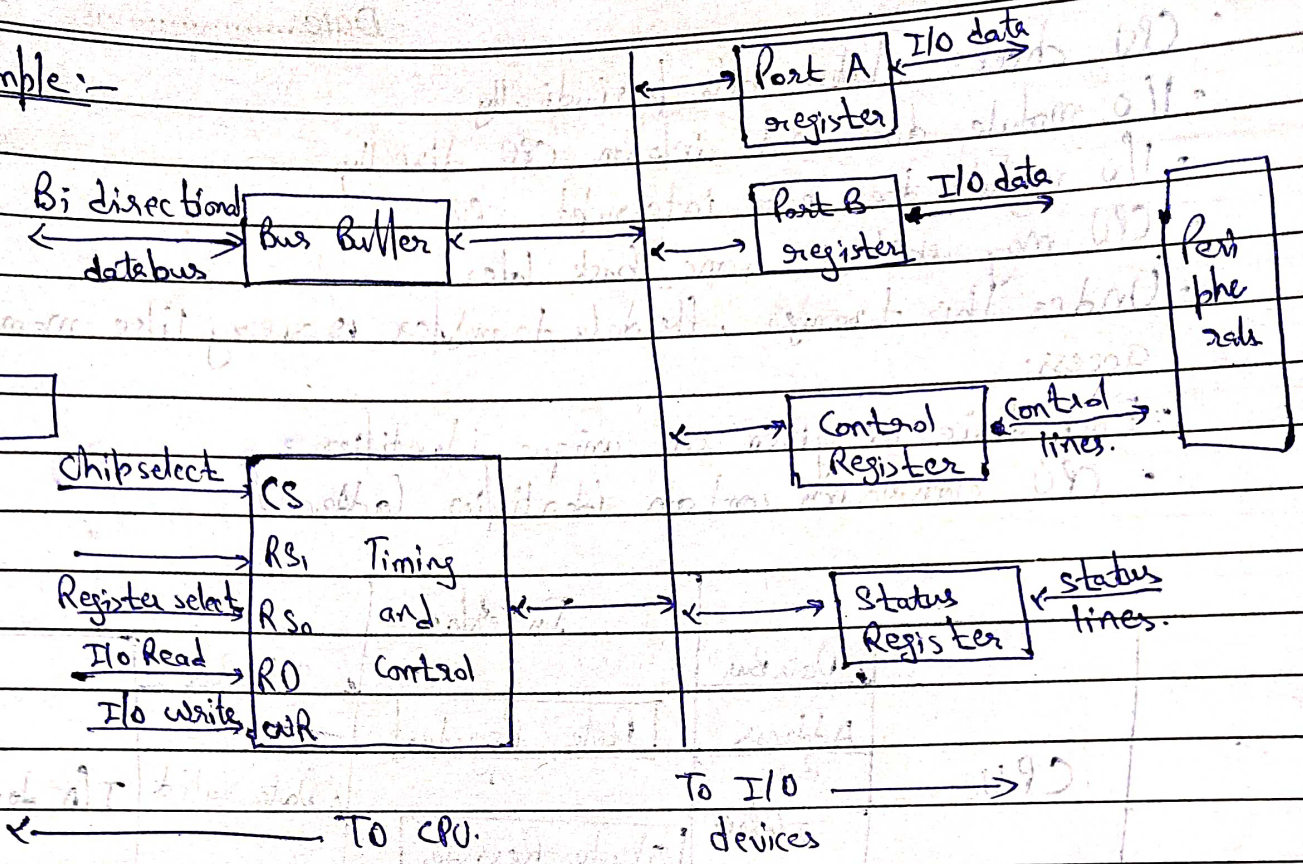
- Hide or reveal device properties to CPU.
- Support multiple or single devices.
- Control device functions or leave for CPU.
- ~~Also, O/S decisions = eg Un~~



Spiral

Date.....

Example :-



CS	RS_1	RS_0	Register Selected
0	X	X	None : data bus in high impedance state
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status Register

⇒ Mode of Transfer :-

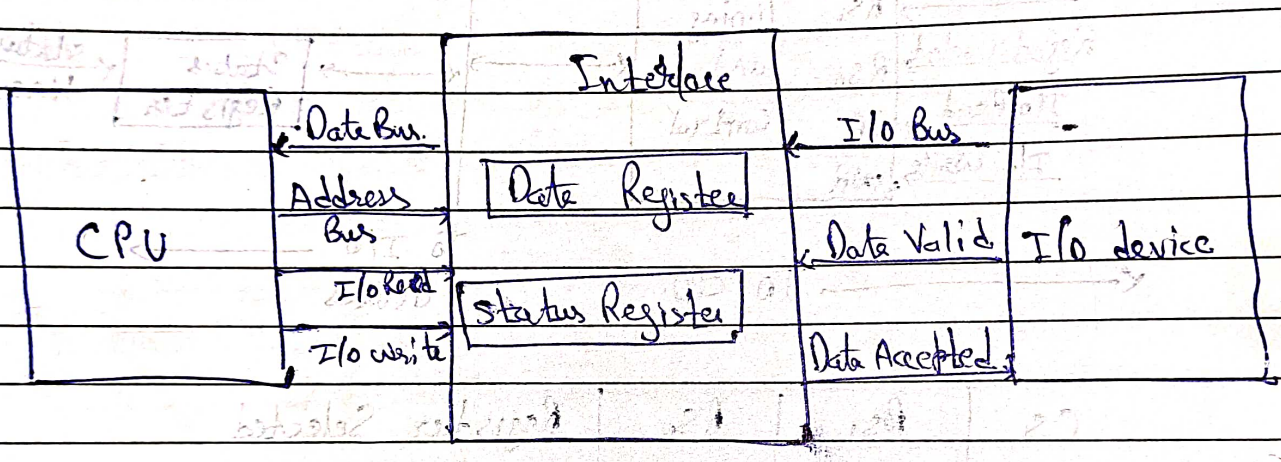
① Programmed I/O :-

- CPU request I/O operation
- I/O module performs operations.
- I/O module sets status bits

Spiral

Date.....

- CPU checks status bits periodically
- I/O module does not inform CPU directly
- I/O module does not interrupt CPU
- CPU may wait or come back later
- Under this transfer, the data transfer is very like memory access.
- Each device is given an unique identifier
- CPU commands contain identifier (address)



disd:- The problem with programmed I/O is that the processor has to wait a long time for the I/O module of concern to be ready for either reception or transmission of data.

:- The processor, while waiting, must repeatedly interrogate the status of the I/O module.

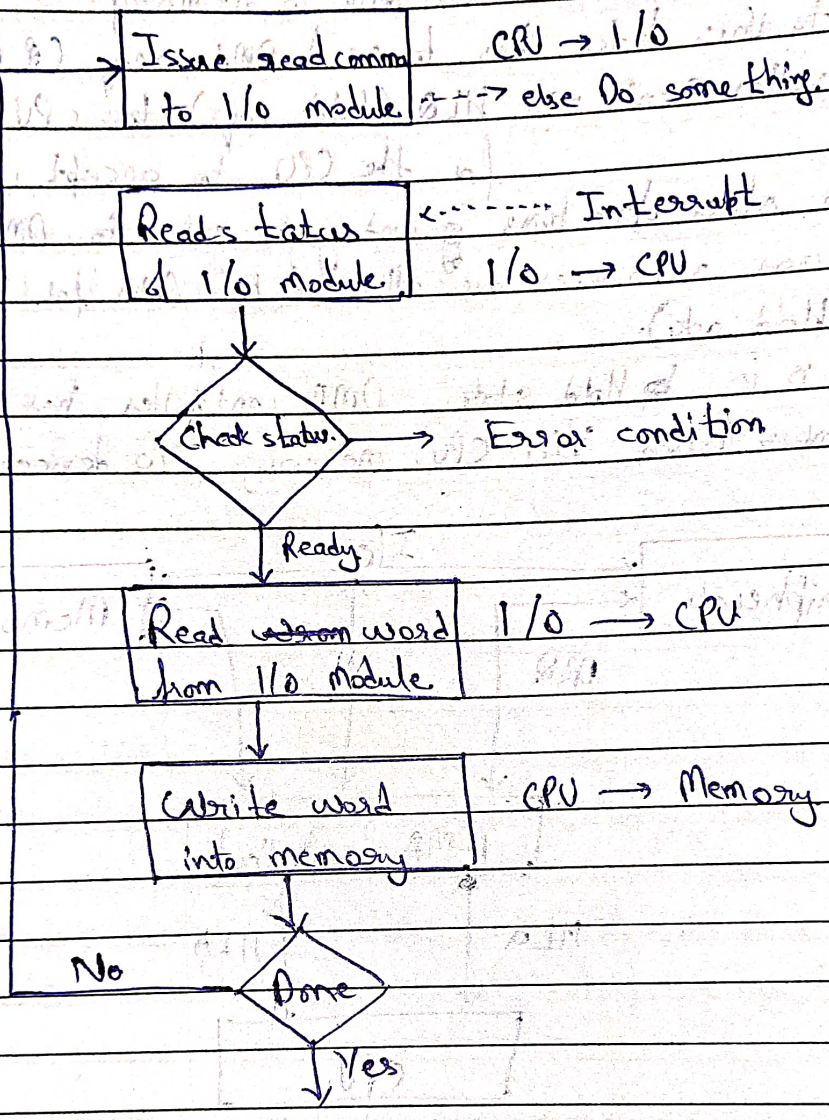
:- Performance of the entire system is severely degraded.

(2) Interrupt driven I/O :-

- CPU issues read command.
- I/O module gets data from peripheral whilst CPU does other work.
- I/O module interrupts CPU.

Date.....

- CPU requests data
- I/O module transfers data



⇒ DMA controller I/O :- It is designed by Intel to transfer data at the fastest rate.

→ It allows the device to transfer data directly to/from memory without CPU involvement.

→ DMA controller takes over from CPU for I/O.

→ It must force the processor to suspend operations temporarily. This technique is called cycle stealing.

Date.....

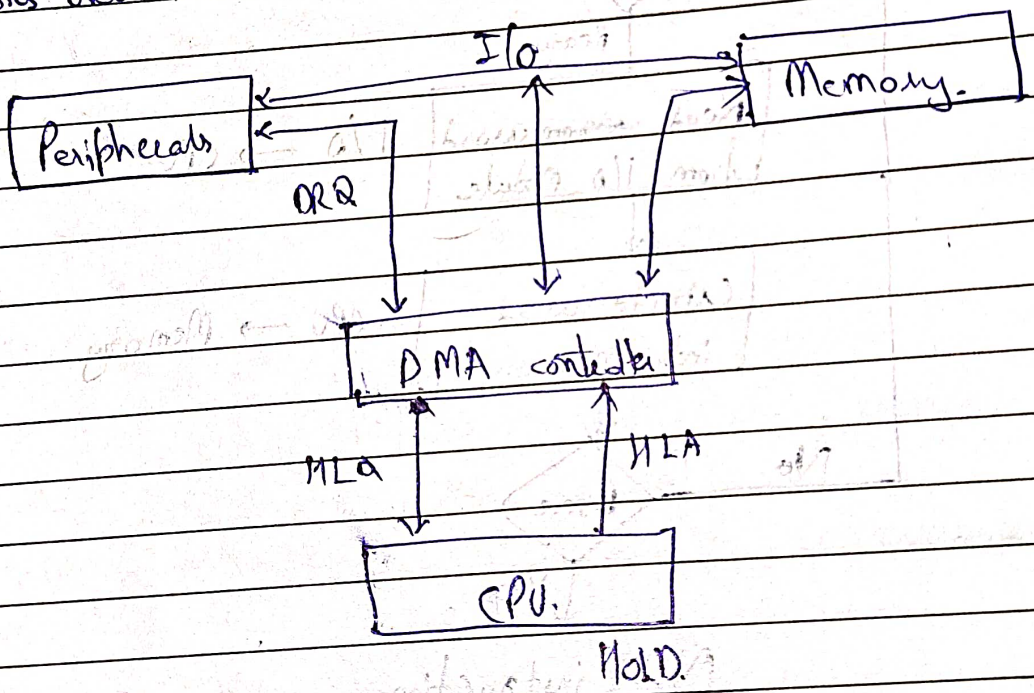
DMA operations :-

① Device (peripheral) wants to send data to memory
In order to this device has to send DMA req (DRQ)

② DMA controller send => HLQ (hold req) to CPU and waits for the CPU to accept it

③ CPU holds other operations and give acks to DMA that all the processor resources are allocated to DMA for I/O and sends HLA (Hold ack).

④ Now CPU is in Hold state, DMA controller has to manage operations over buses via CPU, memory, I/O devices



⇒ **Software interrupt** :- It is a type of interrupt that is caused either by a special instruction in the instruction set or by an exceptional condition in the processor itself. A software interrupt is invoked by software, unlike a hardware interrupt, and is considered one of the ways to communicate with the kernel or to invoke system calls.

Date.....

→ It often occurs when an application software terminates or when it requests the OS for some service.

→ Only one bit of information is communicated during a software interrupt.

→ A software interrupt can also make use of some of the hardware interrupt routines.

→ eg:- Communicating with the disk controller for reading and writing data to and from a disk.

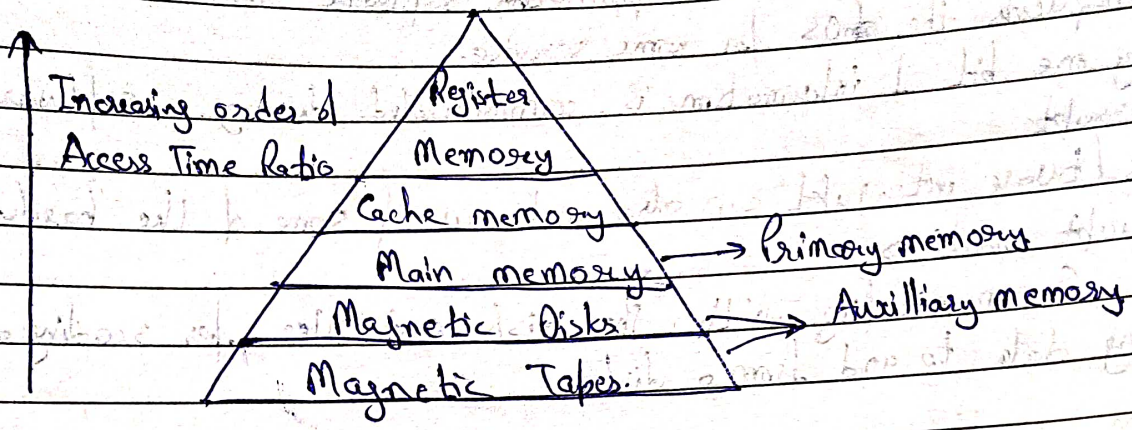
⟹ Memory Heirarchy :-

The memory in a computer can be divided into five heirarchies based on the speed as well as use. The processor can move from one level to another based on its requirements.

The five heirarchies in the memory are registers, cache, main memory, magnetic discs, and magnetic tapes. The first three heirarchies are volatile memories which mean when there is no power, and then automatically they lose their stored data. Whereas the last two heirarchies are not volatile which means they store the data permanently.

The memory heirarchy design in a computer system mainly includes different storage devices. Most of the computers were inbuilt with extra storage to run more powerfully beyond the main memory capacity.

Date.....



⇒ Primary Memory :- Also known as internal memory, and this is accessible by the processor straightly. This memory includes main, cache, as well as CPU registers.

⇒ Secondary Memory :- Also known as external memory, and this is accessible by the processor through an input/output module. This memory includes an optical disk, magnetic disk, and magnetic tape.

⇒ Characteristics of Memory Heirarchy :-

① Performance :- The enhancement of computer system was designed in the memory heirarchy model due to the system's performance increase.

② Ability :- The ability of the memory heirarchy is the total amount of data the memory can store. Because whenever we shift from top to bottom inside the memory heirarchy, then capacity will increase.

Spiral

Date.....

③ Access time :- It is the interval of the time among the data availability as well as request to read or write. Because whenever we shift from top to bottom inside the memory heirarchy, then the access time will increase.

④ Cost per bit :- When we shift from bottom to top inside the memory heirarchy, then the cost for each bit will increase. which means an internal memory is expensive compared to external memory.

⟹ Main Memory :-

The memory unit that communicates directly within the CPU, Auxillary memory, and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations.
Main memory is made up of RAM and Rom, with RAM integrated circuit chips holing the major share.

- RAM :- Random Access Memory
  - → DRAM :- Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10 ~ 100 ms. It is slower and cheaper than SRAM.
  - → SRAM :- Static RAM, has a six transistor circuit in each cell and retains data, until powered off.
  - → NVRAM :- Non-Volatile RAM, retains its data, even when turned off. eg:- Flash Memory.

Spiral

Date.....

- ROM :- Read Only Memory, is non-volatile and is more like a permanent storage for information. It also stores the bootstrap loader program, to load and start the operating system when computer is turned on. PROM (programmable ROM), EEPROM (Electrically Erasable PROM) and EPROM (Erasable PROM) are some commonly used ROMs.

⇒ Auxillary Memory :-

Devices that provide backup storage are called auxillary memory.

eg:- Magnetic disks and tapes are commonly used auxillary devices. Other devices used as auxillary memory are magnetic drums, magnetic bubble memory and optical disks.

It is not directly accessible to the CPU, and is accessed using the INPUT/OUTPUT channels.

⇒ Associative Memory :- It is also known as Content Addressable Memory (CAM).

→ It is a memory chip in which each bit position can be compared. In this the content is compared in each bit cell which allows very fast table lookup.

→ Since the entire can be compared, contents are randomly stored without considering addressing scheme.

→ These chips have less storage capacity than regular memory chips.

Spiral

Date.....

=> **Cache Memory** :- The data or contents of the main memory that are used again by CPU, are stored in the cache memory so that we can easily access that data in shorter time.

-> Whenever the CPU needs to access memory, it first checks the cache memory. If the data is not found in cache memory then the CPU moves onto the main memory.

-> It also transfers block of recent data into the cache and keeps on deleting the old data in cache to accomodate the new one.

Hit Ratio :- The performance of cache memory is measured in terms of a quantity called hit ratio.

-> When the CPU refers to memory and finds the word in cache it is said to produce a hit. If the word is not found in cache; it is in main memory then it counts as a miss.

-> The ratio of the number of hits to the total CPU references to memory is called hit ratio.

$$\text{Hit Ratio} = \text{Hit} / (\text{Hit} + \text{Miss})$$

=> **Cache Mapping** :- It is a technique by which the contents of main memory are brought into the cache memory.

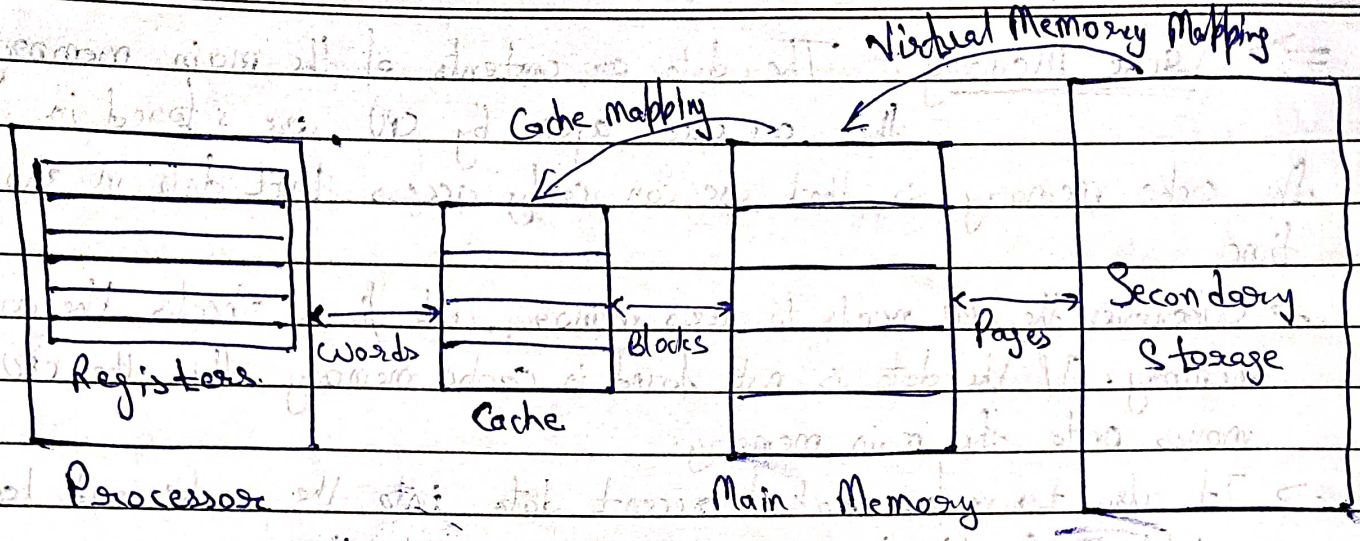
-> Main memory is divided into equal size partitions called as blocks or frames.

-> Cache memory is divided into partitions having same size as that of blocks called as lines.

-> During cache mapping, block of main memory is simply copied to the cache and the block is not actually brought from the main memory.

Spiral

Date.....

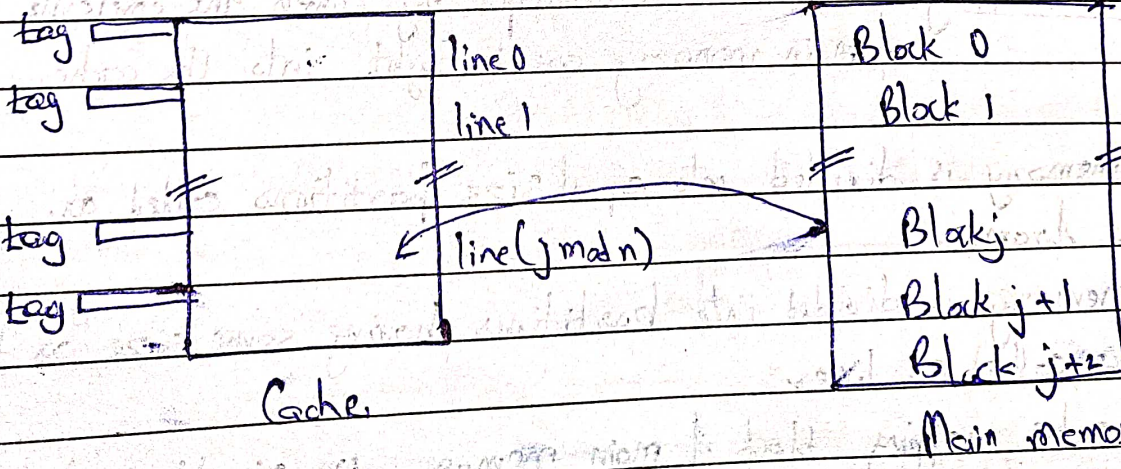


① Direct Mapping :-

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

→ The line number of cache to which a particular block can map is given by. :-

> Cache line no. = (Main memory Block Address) Modulo (No. of lines in Cache)

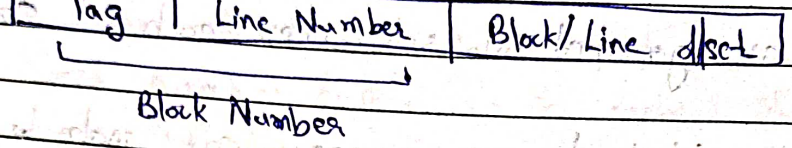


Spiral

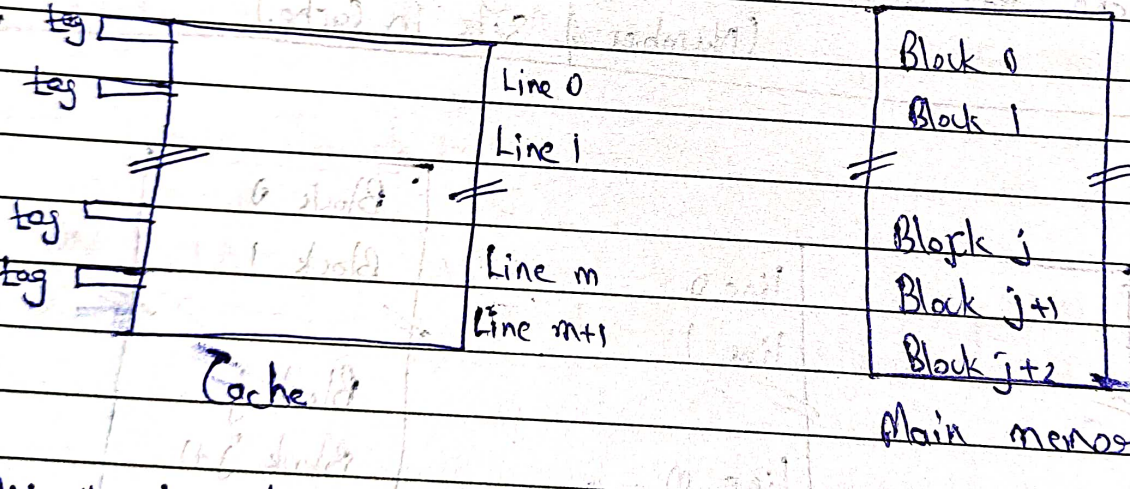
Date.....

→ There is no need of any replacement algorithm because a main memory block can map only to a particular line of the cache. The new incoming block will always replace the existing block in that particular line.

→ The physical address is divided as :-



(2) Associative Mapping :- A block of main memory can map to any line of the cache that is freely available at that moment. This makes fully associative mapping more flexible than direct mapping.



→ All the lines of cache are freely available. Thus, any block of main memory can map to any line of the cache.

→ If all the cache lines been occupied, then one of the existing blocks will have to be replaced.

→ Replacement algorithm like FCFS algorithm, LRU algorithm etc is employed

Date.....

→ The physical address is divided as:-

Block Number / Tag	Block / Line offset

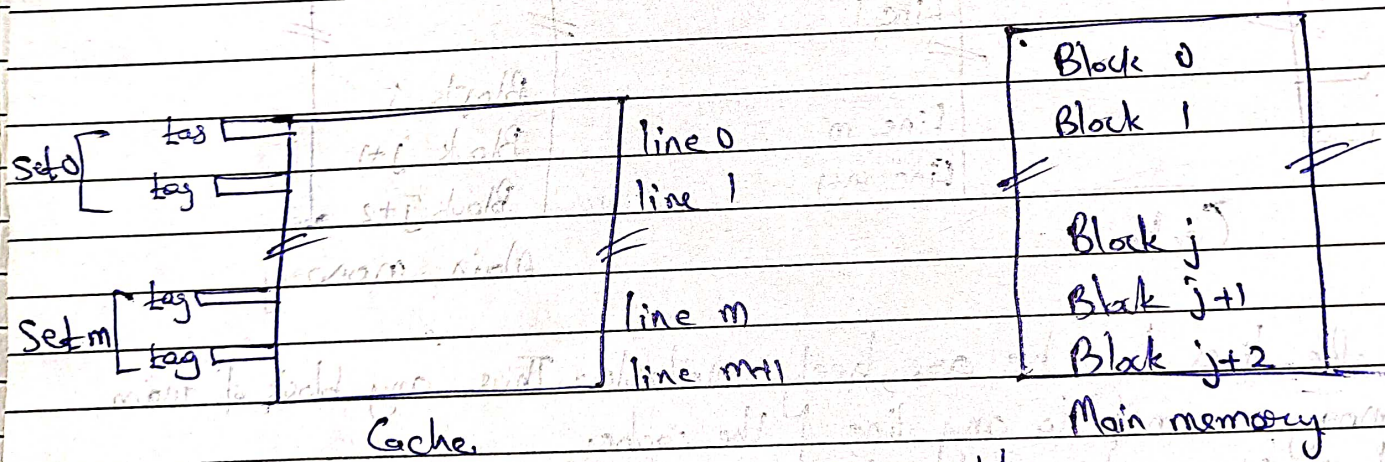
(3) Set Associative Mapping :-

→ Cache lines are grouped into sets where each set contains k number of lines.

→ A particular block of main memory can map to only one particular set of the cache. However, within that set, the memory block can map any cache line that is freely available.

→ The set of the cache to which a particular block of the main memory can map is given by :-

Cache Set No. = (Main Memory Block Address) Modulo (Number of Sets in Cache)



2- way Set Associative Mapping.

→ k=2 suggests that each set contains two cache lines.

→ If all the cache lines are occupied, then one of the existing blocks will have to be replaced.

Date.....

-> It is a combination of direct mapping and fully associative mapping.
-> The physical address is divided as :-

Tag	Set Number	Block / Line offset

-> If k = 1, then it is called direct mapping
-> If k = Total number of lines in the cache, then it becomes associative mapping.

==> Write into Cache Memory :- Whenever a processor wants to write a word, it checks to see if the address it wants to write the data to, is present in the cache or not. If address is present in the cache i.e. Write Hit we can update the value in the cache.
-> But this results in Data Inconsistency Problem.

-> Write Through :-

	V	TAG	DATA
100	1	10101	0303

MAIN MEMORY

1010 1100	0303

↓ after write through

	V	TAG	DATA
100	1	10101	2203

MAIN MEMORY

1010 1100	2203

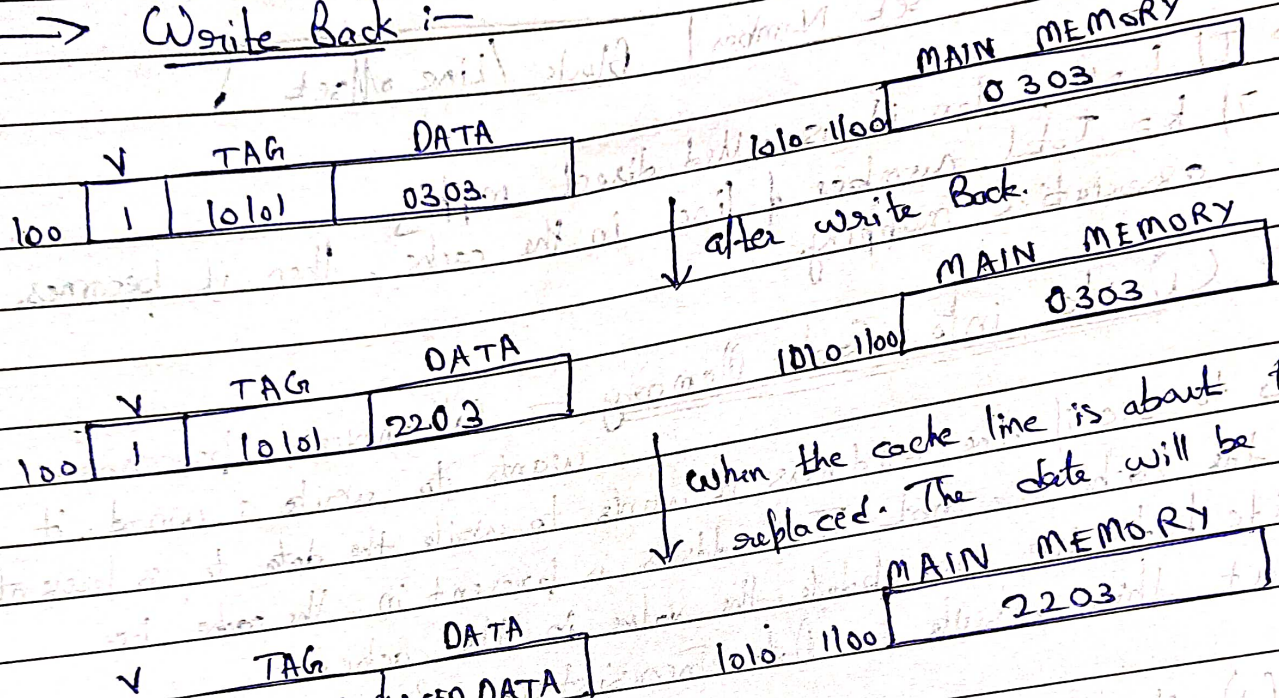
-> Data is simultaneously updated to cache and memory.
-> Process is simpler and reliable.
-> This is done when there are no frequent writes to the cache.
-> In case of power failure it helps in data recovery.

Spiral

Date.....

-> Data write will experience latency.

-> Write Back :-



-> The data is updated only in the cache.

-> Data is updated into the memory only when the cache line is ready to replaced.

-> Also known as Write Deferred.

-> Dirty Bit :- If cache was modified is called Dirty and if not modified called Clean.

-> If write occurs to a location that is not present in the Cache, we use two methods, Write Allocation, and Write Around

Date.....

① → Write Allocation :-

Write Miss occurs

	V	TAG	DATA
100	0	-	-

1010 1100 | MAIN MEMORY | 0303

after write Allocate.

	V	TAG	DATA
100	1	10101	2203

will be updated later if write back and immediately if write through.

→ In write allocation data is loaded from the memory into cache and then updated.

→ Generally used with write back.

→ ② Write Around :-

Write Miss occurs

	V	TAG	DATA
100	0	-	-

1010 1100 | MAIN MEMORY | 0303

after write around

	V	TAG	DATA
100	0	-	-

MAIN MEMORY | 2203

→ Data is directly updated to main memory without disturbing Cache.

→ Better to use when the data is not being immediately used again.

Date.....

⟹ **Virtual Memory** :— It is a valuable concept that allows you to run large, sophisticated programs on a computer even if it has a relatively small amount of RAM.

→ A computer with virtual memory artfully juggles the conflicting demands of multiple programs within a fixed amount of physical memory.

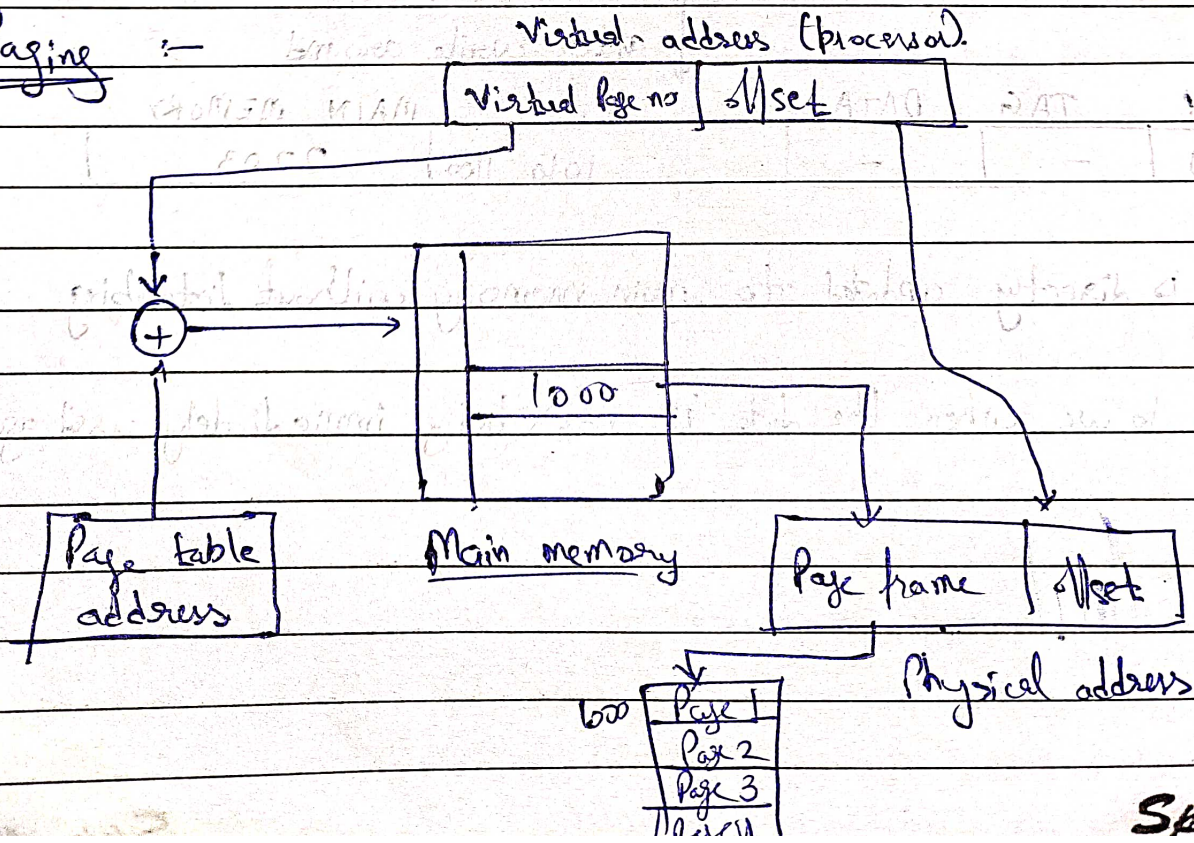
→ First segment is loaded and the next replaces the first one.

→ Techniques that automatically move programs into main memory are virtual memory technique.

→ **Process** :—

- An instruction is refered without looking at its space requirements
- The referenced address is virtual address.
- Virtual address —H/w + S/w components→ Physical address

Paging :—



Spiral

Date.....

-> Virtual memory breaks programs into fixed size blocks called pages.

-> The OS fits as much as it can run the instructions in those pages.