Integer Representation

i) Sign magnitude
ii) 1's complement
iii) 2's complement

Represent using sign magnitude ; 1's complement and 2's complement for the following :-

+13 and -13

	+13	-13
sign magnitude	00001101	10001101
1's complement	00001101	11110010
2's complement	00001101	11110011

-11 in 8-bit

	+11	-11
sign magnitude	00001011	10001011
1's complement	00001011	1111 0100
2's complement	00001011	1111 0101

+0 and -0 in 4-bit

	+0	-0
sign magnitude	0000	1000
1's complement	0000	1111
2's complement	0000	0000

That why modern digital computers use 2's complement for the negative numbers operations.

Scanned by CamScanner

Overflow

i) $\begin{matrix} - \\ + \end{matrix}$ or $\begin{matrix} + \\ - \end{matrix}$ } There is no chance of overflow

ii) $\begin{matrix} - \\ - \end{matrix}$ or $\begin{matrix} + \\ + \end{matrix}$ } There is chance of overflow

Define overflow. Explain with example. What hardware is required to detect overflow.

Eg :- a) Assume 4-bit

i) For unsigned

$$\begin{array}{rcl} 9 & \Rightarrow & 1001 \\ +\ 11 & \Rightarrow & \underline{1011} \\ & & 10100 \end{array}$$

The addition of two 4-bit numbers generate sum in 5 bit. So, overflow occurred.

ii) For signed

$$\begin{array}{r} -5 \\ -7 \\ \hline \end{array}$$

$+5 \Rightarrow 0101$
$-5 \Rightarrow 1011$
$+7 \Rightarrow 0111$
$-7 \Rightarrow 1001$

$$\begin{array}{rccccc} & 1 & 0 & & & \\ & 1 & 0 & 1 & 1 & \\ + & 1 & 0 & 0 & 1 & \\ \hline 1 & 0 & 1 & 0 & 0 & \end{array}$$

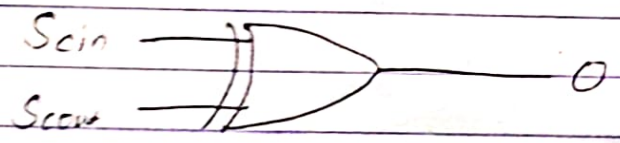
Overflow occurs when carry into sign bit $\neq$ carry out from the sign bit.

Scanned by CamScanner

Here is truth for carry into sign and carry out from sign bit.

S_{cin}	S_{cout}	Overflow
0	1	1
1	0	1
0	0	0
1	1	0

Analysing above truth, the hardware to detect overflow is X-OR gate 2.)



b) Assume 5 - bit

i) For unsigned

$$\begin{array}{rcl} 20 & \Rightarrow & 10100 \\ +15 & \Rightarrow & \underline{01111} \\ & & 100011 \end{array}$$

The addition of two 5 bit number generates the sum in 6-bit. So, overflow occurred.

ii) For signed

let the signed no. be -15 and -2;

+15 ⇒ 01111 and -2 ⇒ 00010

-15 ⇒ 10001 -2 ⇒ 11110

Here,

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

Since, the carry into sign bit is not equal to the carry out from sign bit, hence, overflow occurred.

Fixed Point Representation

The position of decimal point or binary point is always fixed.

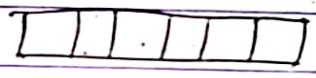
e.g. 99.09

110.10 etc

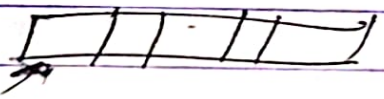
+ve => 0 in MSB

-ve => 1 in MSB

However, there is no physical representation of decimal or binary point. We only assume its presence.



← We assume extreme right position for binary point to store integer (non-fraction)



We assume extreme left for binary point to store fractional number.

Scanned by CamScanner

3) Decimal Fixed Point Representation

$$\begin{array}{r} +\ 650 \\ +\ (-375) \\ \hline ? \end{array}$$

10's complement of 375;

$\Rightarrow r^n - N$

$\Rightarrow 10^3 - 375$

$\Rightarrow 1000 - 375$

$\Rightarrow 625$

+ 650 ⇒	0000 (1)	0110	0101	0000
− 625 ⇒	1001	0110	0010	0101
	1010	1100	1111	0101
	0110	0110		
	10000	00010		

$\Rightarrow +275$

4) Floating Point Representation

Expressed in the form of $m \times r^e$

where

m = mantissa

r = Base of radix

e = exponential (determines the position of decimal / binary point)

eg:- $9009 = 9.009 \times 10^3$

$= 0.9009 \times 10^4$

Scanned by CamScanner

Normalized floating point representation

It can provide highest possible precision

Unnormalized	Normalized
0.001	0.1×10^{-2}
9.0003	0.9003×10^{1}

Two standard

i) IEEE

ii) ANSI

Floating point numbers can be represented in ANSI 32 bit standard as follows :-

$$\underbrace{S}_{\text{sign}}\underbrace{EEEEEEEE}_{\text{exponent (2's complement form)}}$$

Mantissa

eg i) $1.001 \Rightarrow 0.1001 \times 2^{1}$

$\Rightarrow$ 00000001 10010000 00000000 00000000

ii) $0.00011 \Rightarrow 0.11 \times 10^{-3}$

$\Rightarrow$ 01111101 11000000 00000000 00000000

iii) $-10.01 \Rightarrow -0.1001 \times 10^{2}$

$\Rightarrow$ 100000010 10010000 00000000 00000000

Scanned by CamScanner

7) Binary Code

i) Gray - Code

ii) Excess -3

iii) BCD

iv) 84-2-1

v) ASCII

vi) EBCDIC

Self complementing code

	8	4	-2	-1	Excess-3
0	0	0	0	0	0011
1	0	1	1	1	0100
2	0	1	1	0	0101
3	0	1	0	1	0110
4	0	1	0	0	0111
5	1	0	1	1	1000
6	1	0	1	0	1001
7	1	0	0	1	1010
8	1	0	0	0	1011
9	1	1	1	1	1100

8) Error Detection code

→ Parity (Even or Odd).

Scanned by CamScanner

Even Parity Generator

3- bit message

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

Circuit

$A \oplus B \oplus C$

Odd Parity Generator

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

$P_o = C \odot (A \oplus B)$

$= C \odot x$

$x = (A \oplus B)$

Scanned by CamScanner

2. Register Transfer & Microoperations

Microoperation

The operation performed on the data stored in registers is called microoperation. It is elementary operation performed on data stored in registers. eg load, store, clear, add, subtract, shift etc.

Register Transfer Language

The symbolic notation used to describe the microoperation transfer among registers is called a register transfer language.

eg:- $R_1 = R_2 + R_3$ $\Rightarrow$ $R_1 \leftarrow R_2 + R_3$

microoperation RTL

Register transfer

eg:- $R_1 \leftarrow R_2$

Basic Symbol for register transfer

Symbol	Description	Example
letter & numbers	Denotes a register	R_1, R_2, MAR etc
Parenthesis ( ),	Denote a part of register	$R2(0-7)$, $R2(L)$, $R(H)$
Arrow $\leftarrow$	Denote transfer of microoperation	$R_2 \leftarrow R_1$
Comma ,	Seperate two microoperations	$R_2 \leftarrow R_1$, $R_3 \leftarrow R_4$

eg:- If $q = 1$ then $(R_1 \leftarrow R_2 + R_3)$ or $q = 0$

$q : R_1 \leftarrow R_2 + R_3$ $\quad$ $\bar{q} : R_1 \leftarrow R_2 + R_3$

control function

Scanned by CamScanner

5.) Bus and Memory Transfer

i) Common Bus

Common Bus can be constructed either by using MUX or three state buffer.

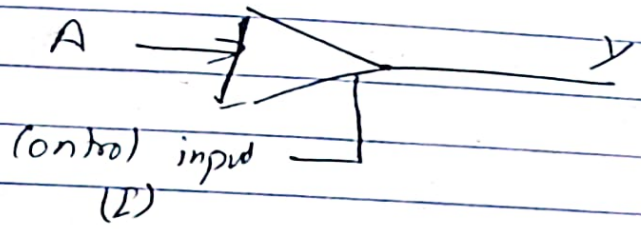
a) Common bus system with Multiplexer

* Function table

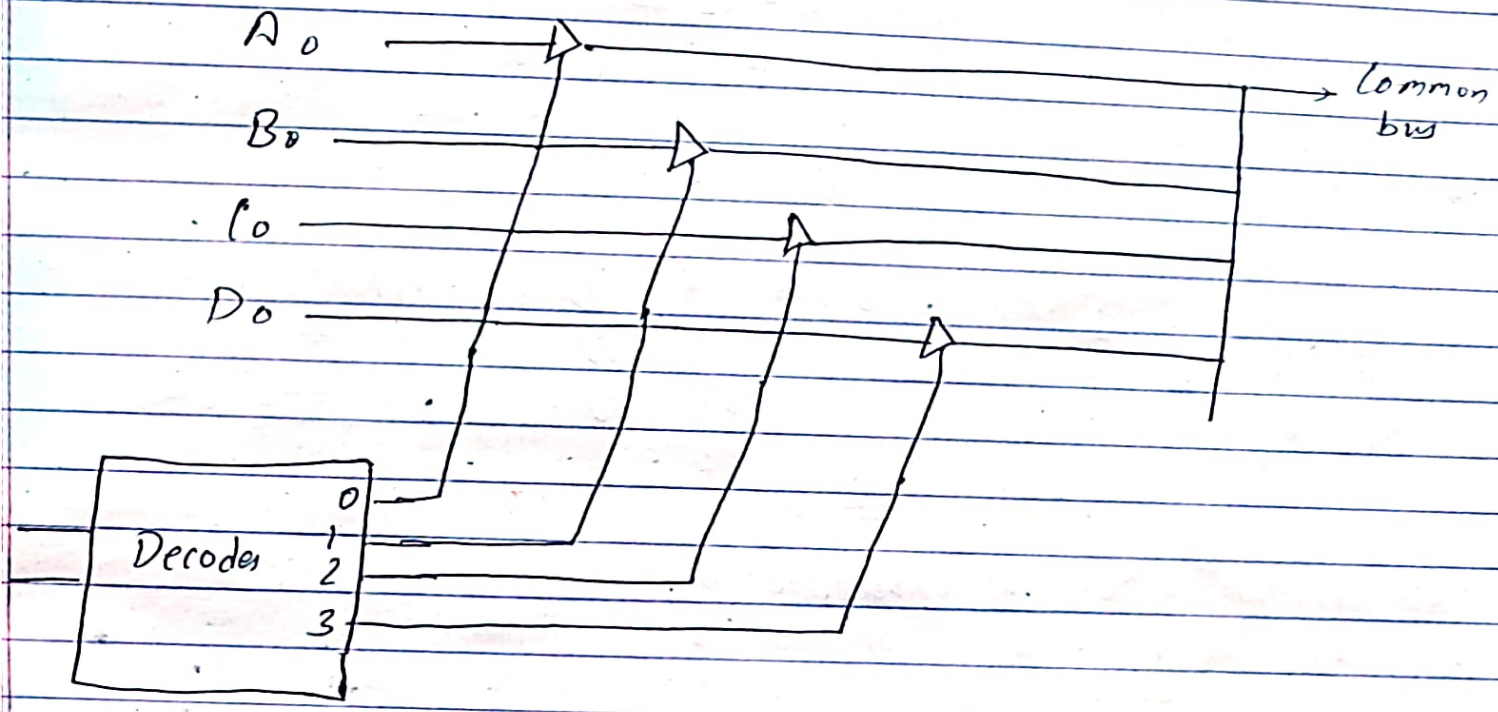
Selection		Register
0	0	A
0	1	B
1	0	C
1	1	D

Scanned by CamScanner

Common bus system with three state buffes



If $I = 1$; $Y = A$
If $I = 0$; $Y =$ higher impedence state



S_0	S_1	Decoder o/p
0	0	0
0	1	1
1	0	2
1	1	3

Scanned by CamScanner

6) Memory Transfer

Two operations

a) Memory read

Memory word

Read : $DR \leftarrow M[AR]$

b) Memory write

Write : $M(AR) \leftarrow R_1$

7) Types of Microoperation

i) Arithmetic Microoperation eg : Add, Sub etc

ii) Logical Microoperation eg : AND, OR etc

iii) Register Transfer " eg : $R_1 \leftarrow R_2$

iv) Shift " eg : $R \leftarrow ShlR$

8) Arithmetic Microoperation

a) $R \leftarrow A+B$ // addition

b) $R \leftarrow A+B+1$ // addition with carry

c) $R \leftarrow A+\bar{B}$ // subtraction with borrow

d) $R \leftarrow A+\bar{B}+1$ // subtraction using 2's complement

e) $R \leftarrow A-B$ // subtraction

f) $R \leftarrow A+1$ // increment

g) $R \leftarrow A-1$ // decrement

Scanned by CamScanner

Binary Adder

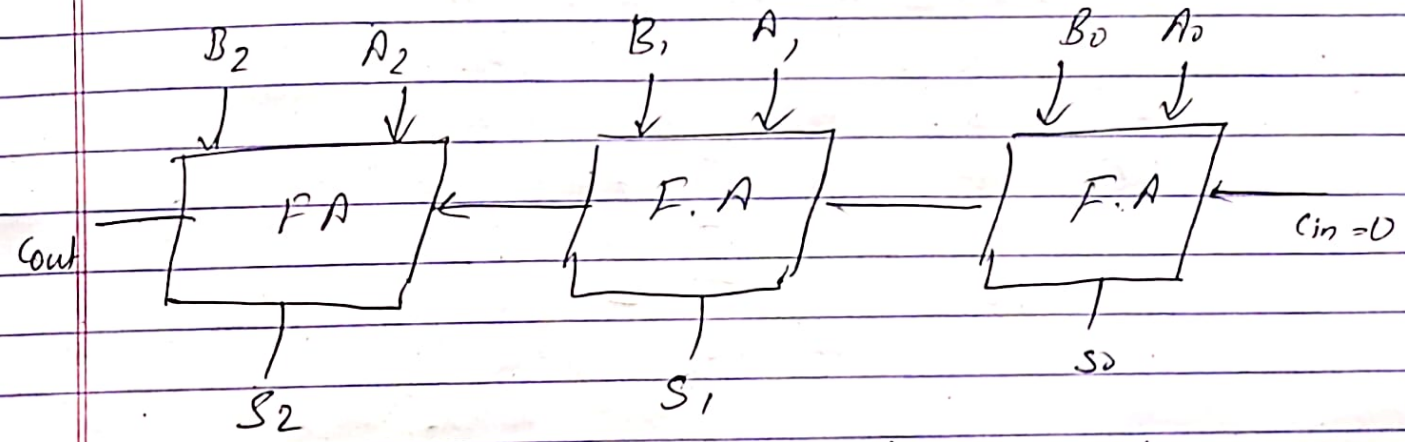


fig :- 3-bit parallel binary adder

Binary Subtractor

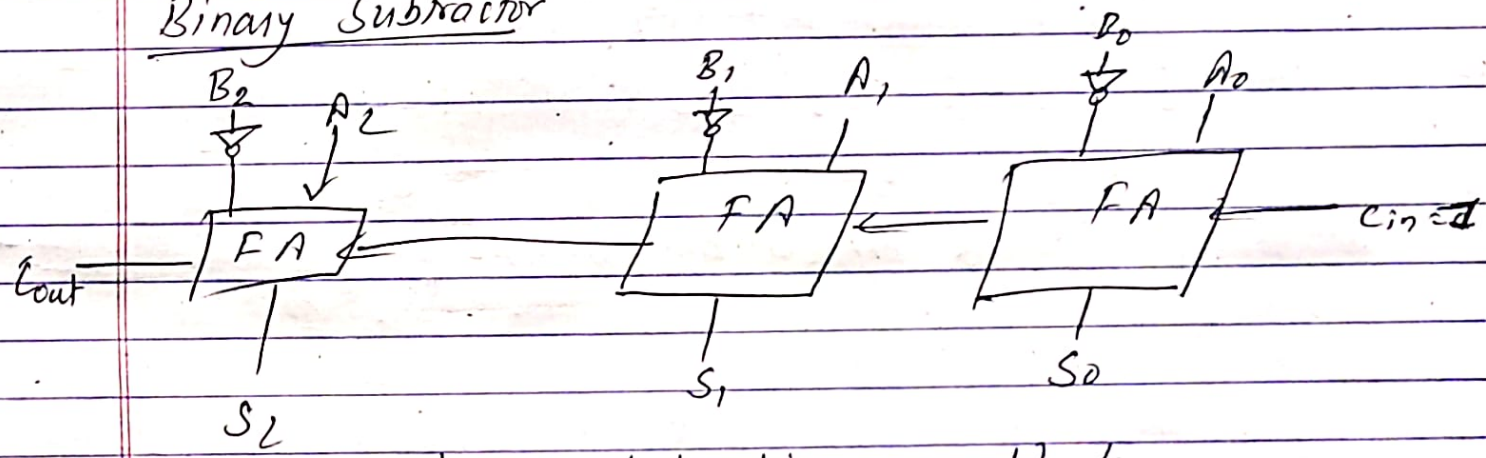


fig: 3-bit binary subtractor

Adder - Subtractor

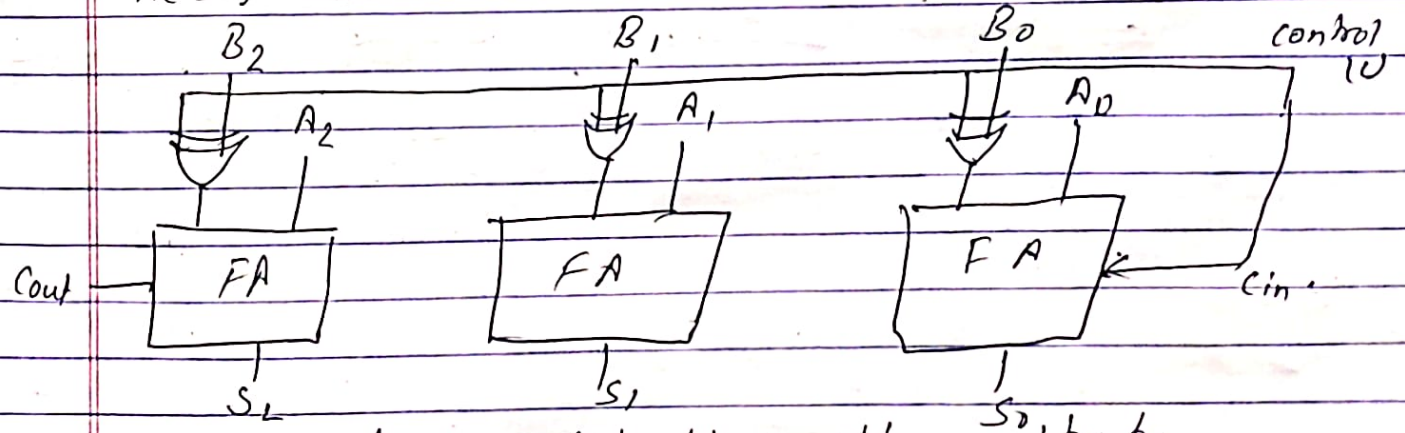


fig :- 3 bit binary adder - subtractor.

Scanned by CamScanner

If $c=0$: $S = A+B$ (Addition)
$c=1$: $S = A+\bar{B}+1$ (Subtraction)

C	B	o/p	
0	0	0	+
0	1	1	
1	0	1	−
1	1	0	

Binary incrementer

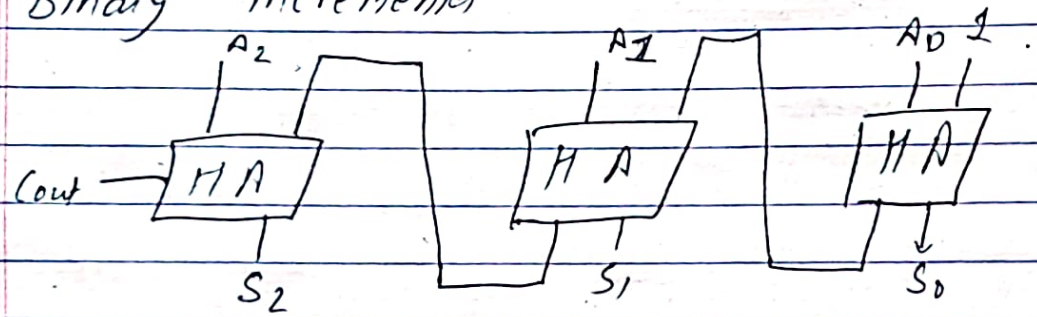
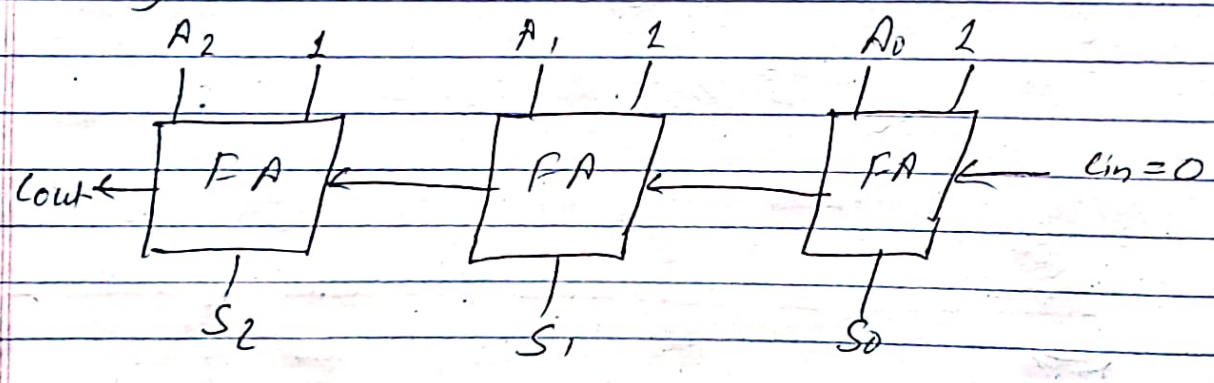


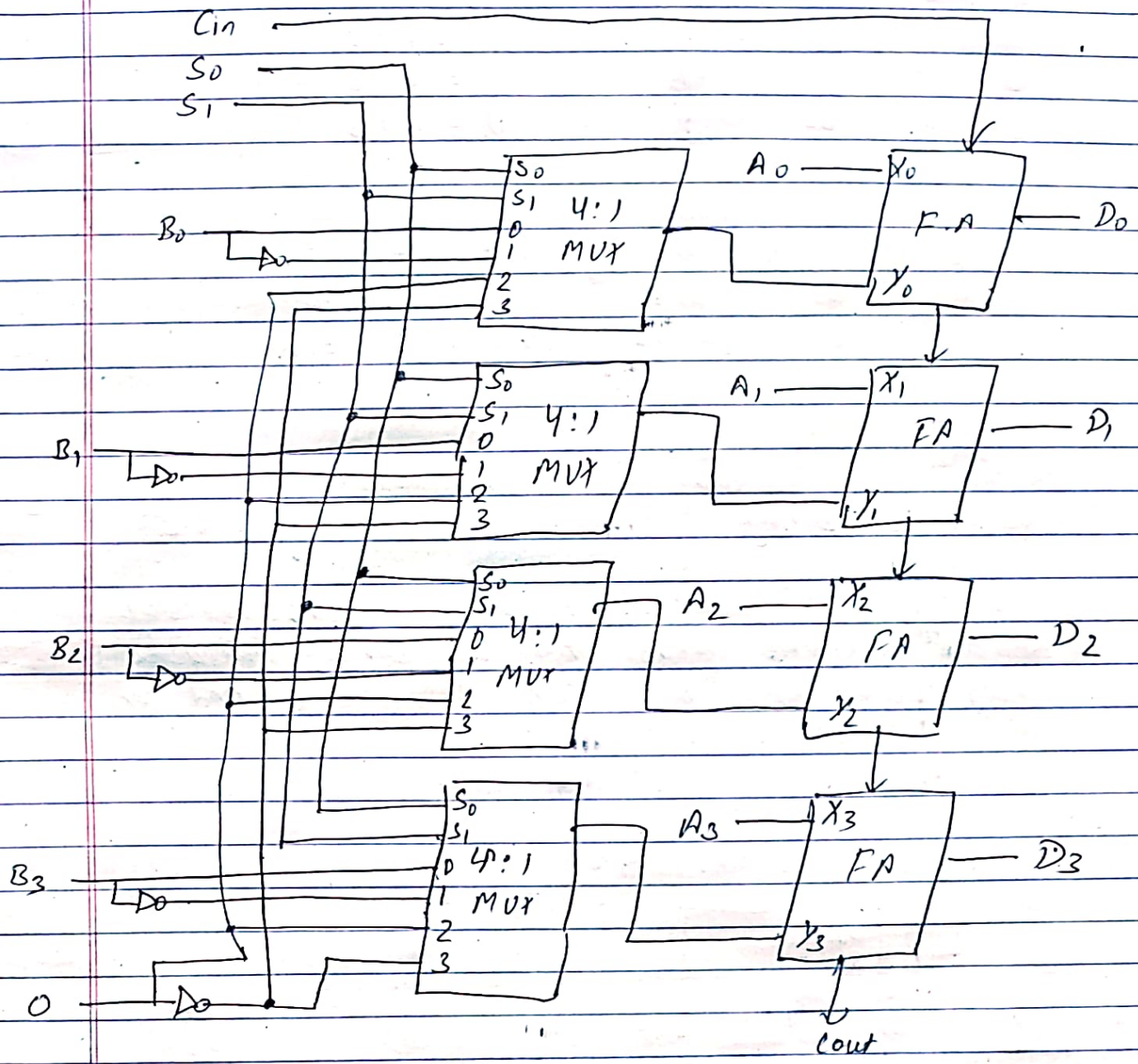
fig :- 3-bit binary incrementer

Binary decrementer



Scanned by CamScanner

4-bit arithmetic circuit



MUX

S_1	S_0	o/p
0	0	0
0	1	1
1	0	2
1	1	3

Scanned by CamScanner

Function Table

S_1	S_0	C_{in}	Y	
0	0	0	B	$D = X + Y + C_{in}$
0	0	1	B	$D = A + B$
0	1	0	$\bar{B}$	$D = A + B + 1$
0	1	1	$\bar{B}$	$D = A + \bar{B}$
1	0	0	0	$D = A + \bar{B} + 1$
1	0	1	0	$D = A$
1	1	0	all 1's	$D = A + 1$
1	1	1	all 1's	$D = A - 1 + 1 = A$

Logic Microperation

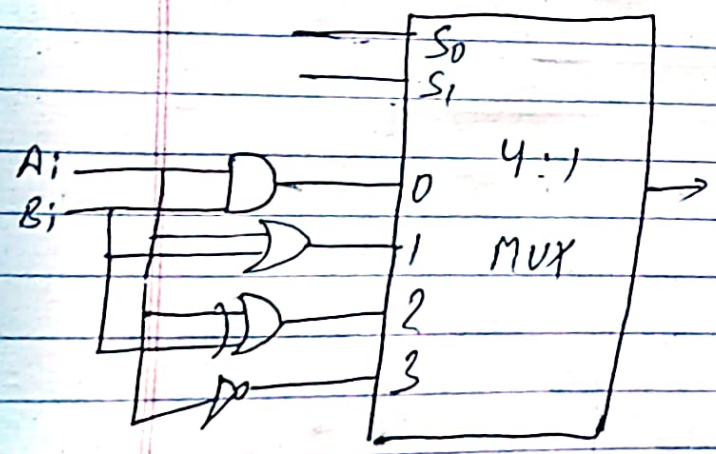
i) Specifies binary operation for string of bit stored in register.

ii) These operation consider each bit of the register seperately and treats them as binary variables.

iii) Very useful for bit manipulation data and for making logical decision.

eg : AND, OR, XOR etc.

Hardware implementation



Function table

S_0	S_1	o/p
0	0	$A \wedge B$
0	1	$A \vee B$
1	0	$A \oplus B$
1	1	$\bar{A}$

Scanned by CamScanner

V.V I

Q.) Explain the application of logical microoperation with example

Useful for manipulating individual bits or a portion of a word stored in a register
They can be used to change bit values, delete a group of bit or insert new bit value into a register.

Some application

Assume :-

processor register (A) = 1010
operand register (B) = 0011

a) Selective-set operation
Sets to 1 the bits in register A where there are corresponding 1's in register B.

eg :- A (before) : 1010
B . 0011
A (after) 1011 (Microoperation OR)

b) Selective complement
Complement bits in A where there are corresponding 1's in B.

eg. A (before) : 1010
B : 0011
A (after) : 1001 (Microoperation X-OR)

Scanned by CamScanner

c) Selective clear

Clears to 0 the bits in A only where there is corresponding 1's in B

$$\begin{array}{rl} A = & 1010 \\ B = & \underline{0011} \\ & 1000 \end{array}$$ (Microoperation $A \wedge \overline{B}$)

d) Mask Operation

bits of A are cleared only where there are corresponding 0's in B

eg :- $$\begin{array}{rl} A = & 1010 \\ B = & \underline{0011} \\ & 0010 \end{array}$$ (Microoperation AND)

e) Clear Operation

Compares the word in A and B and produces an all 0's result if the two number are equal

eg :- $$\begin{array}{rl} A = & 1010 \\ B = & \underline{1010} \\ & 0000 \end{array}$$ (XOR)

f) Insert Operation

→ Masking (AND) + ORING

$$\begin{array}{rrr} A = & 1001 & 0111 \\ & \underline{1111} & \underline{0000} \\ & 1001 \ \cancel{0000} & 0000 \\ & \underline{0000} & \underline{1100} \\ & 1001 & 1100 \end{array}$$

Scanned by CamScanner

Shift Microoperation

→ Used for serial transfer of data
→ Also used in conjuction with asithmetic, logic & other data processing operations.
→ The content of a register can be shifted to the left or right.

Types

* Logical Shift
→ Transfer 0 through the serial inputs
Micro operation

i) $R_1 \leftarrow shl\ R_1$ eg:- R_1 (before) : 100011
R_1 (after) : 0 0 0 1 1 1 0 (out)

ii) $R_1 \leftarrow shr\ R_1$ eg:- R_1 (before) : 100011
R_1 (after) : 0 10001 (out)

* Circular Shift (aka rotate operation).
→ Circulates the bit of the register around the two ends without loss of information.
Microoperation :

i) $R_1 \leftarrow cil\ R_1$ Eg : R_1 (before) : 100010 R_1 (after) : 00010
$R_2 \leftarrow cir\ R_1$ Eg :- R_2 (before) : 001111 R_1 (after) : 100111

Scanned by CamScanner

Arithmetic shift

→ Shifts a signed binary number to right or left.

→ An arithmetic shift-left multiplies a signed binary number by 2. So, there is always chance of overflow.

Overflow can be detected

as $V_S = R_{n-1} \oplus R_{n-2}$

R_{n-1}	R_{n-2}	.....	R_0

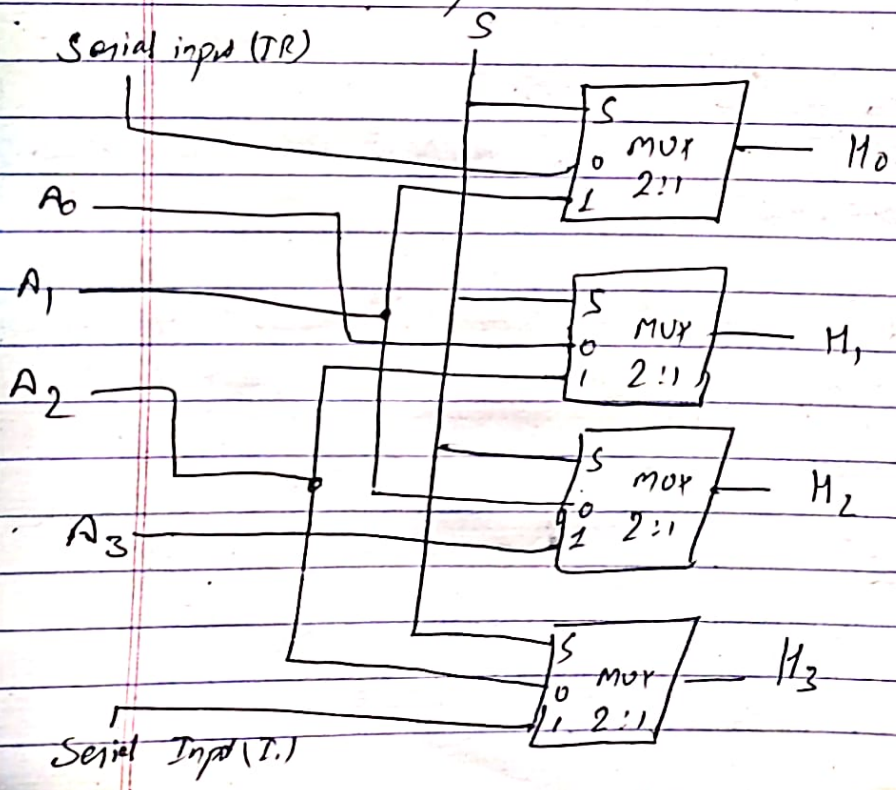
→ An arithmetic shift-right divides a signed binary number by 2 (no chance of overflow)

Microoperation :-

i) $R \leftarrow$ ashr R eg 1001101 → 1100110

ii) $R \leftarrow$ ashl R eg:- 0011001 → 0110010

Hardware implementation



Function table

S	H_0	H_1	H_2	H_3
0	IR	A_0	A_1	A_2
1				

Scanned by CamScanner

Imp

Arithmetic Logic Shift Unit

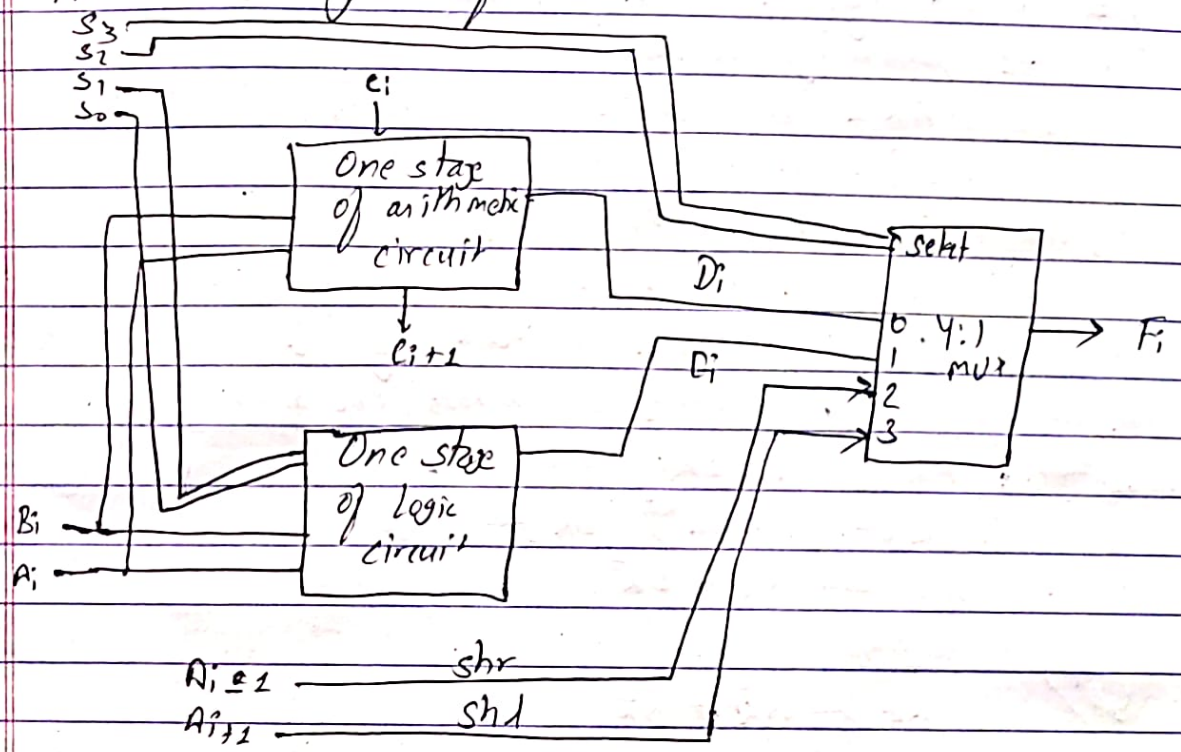


fig: one stage of ALSU

Hardware Implementation

	Control Input S_3	S_2	S_1	S_0	C_i	Output F_i
Arithmetic Micro op	0	0	0	0	0	$F = A+B$
	0	0	0	0	1	$F = A+B+1$
	0	0	0	1	0	$F = A+\bar{B}$
	0	0	0	1	1	$F = A+\bar{B}+1$
	0	0	1	0	0	$F = A$
	0	0	1	0	1	$F = A+1$
	0	0	1	1	0	$F = A-1$
	0	0	1	1	1	$F = A$

	Control Input S_3	S_2	S_1	S_0	C_i	Output F_i
Logic Micro op	0	1	0	0	X	$F = A \wedge B$
	0	1	0	1	X	$F = A \vee B$
	0	1	1	0	X	$F = A \oplus B$
	0	1	1	1	X	$F = \bar{A}$
Shr	1	0	X	X	X	$F = \text{Shr } A$
Shl	1	1	X	X	X	$F = \text{Sh} \downarrow A$

Scanned by CamScanner

The 8-bit registers AR, BR, CR, and DR initially have the following values

AR = 11110010
BR = 11111111
CR = 10111001
DR = 11101010

Determine the 8-bit values in each register after execution of the following sequence of micro-operation

$AR \leftarrow AR + BR$
$CR \leftarrow CR \wedge DR, \quad BR \leftarrow BR + 1$
$AR \leftarrow AR - CR$

Sol/

$AR \leftarrow AR + BR$

```
AR :   11110010
     + 11111111
     ---------
   (1) 11110001
```

AR ⇒ 11110001

$BR \leftarrow BR + 1$

```
BR :   11111111
     +       1
     ---------
   (1) 00000000
```

BR ⇒ 00000000

$CR \leftarrow CR \wedge DR$

```
CR :   10111001
     ∧ 11101010
     ---------
     10101000
```

CR ⇒ 10101000

$AR \leftarrow AR - CR$

```
AR :   11110001
     - 10101000
     ---------
       00101
```

```
AR :   11110001
CR+1 : 01010000
     ---------
   (1) 01001001
```

Scanned by CamScanner

Unit 7

VVI

Computer Arithmetic

Addition & Subtraction

a) Addition & subtraction with signed magnitude data.
b) Addition & subtraction with signed magnitude data.

Addition (Subtraction) algorithm

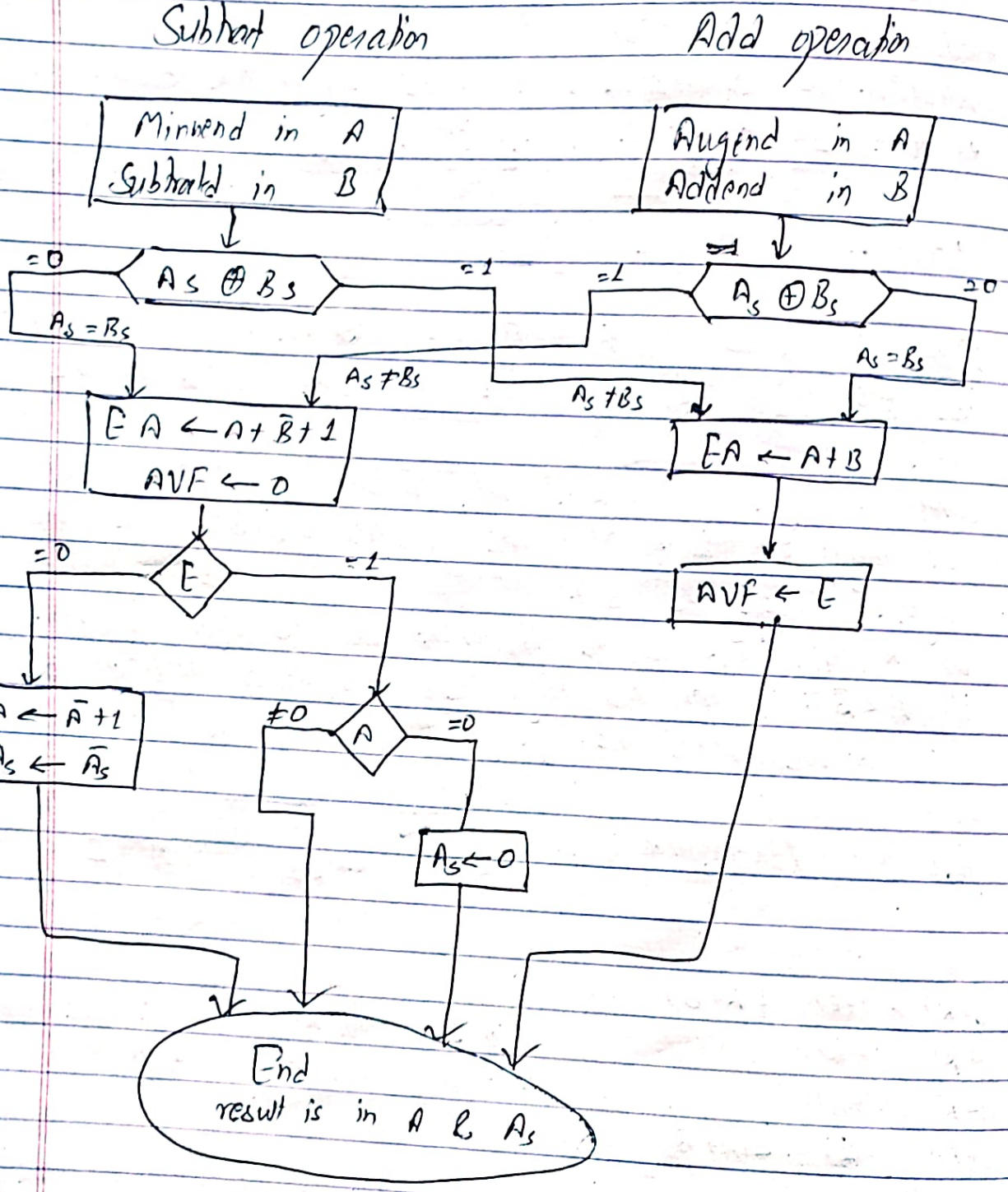
→ When the signs of A & B are identical (different), add the two magnitude & attach the sign of A to the result

→ When the signs of A & B are different (identical), compare the magnitude & subtract the smaller number from the larger. Choose the sign of the result to be the same as A if $A > B$ or the complement of the sign of A if $A < B$. If the two magnitudes are equal, subtract B from A and make the sign of the result positive.

Operations	Add magnitude	Subtract Magnitude		
		$A > B$	$A < B$	$A = B$
$(+A) + (+B)$	$+(A+B)$			
$(+A) + (-B)$		$+(A-B)$	$-(B-A)$	$+(A-B)$
$(-A) + (+B)$		$-(A-B)$	$+(B-A)$	$+(A-B)$
$(-A) + (-B)$	$-(A+B)$			
$(+A) - (+B)$		$+(A-B)$	$-(B-A)$	$+(A-B)$
$(+A) - (-B)$	$+(A+B)$			
$(-A) - (+B)$	$-(A+B)$			
$(-A) - (-B)$		$-(A-B)$	$+(B-A)$	$+(A-B)$

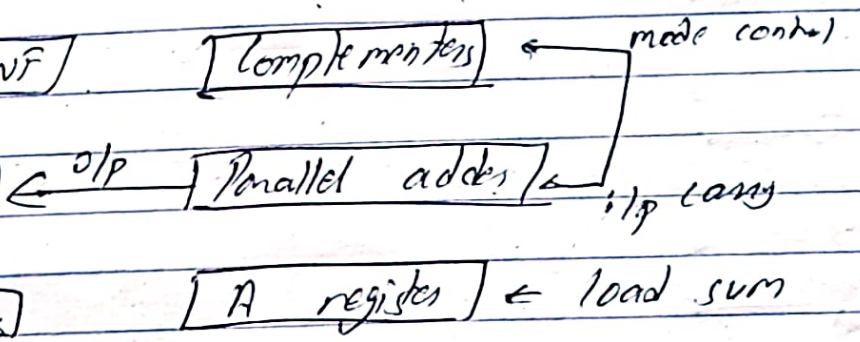
Scanned by CamScanner

Flow chart



Scanned by CamScanner

Hardware implementation



Numerical Example

a) $(+17) + (-19)$

Augend (A) = 17 = 0010001
$A_s = 0$
Addend (B) = 19 = 0010011
$B_s = 1$
$A_s \oplus B_s = 1$
A = 0010001
$\bar{B}+1$ = 1101101
1111110
E = 0
A = 1111110
$A \leftarrow \bar{A}+1$ = 0000010
$A_s \leftarrow \bar{A}_s = 1$
Result : A_s A
1 0000010
$\Rightarrow -2$.

b) $(-17) + (+19)$

Augend (A) = 17 = 0010001
$A_s = 1$
Addend (B) = 19 = 0010011
$B_s = 0$
$A_s \oplus B_s = 1$
A = 0010001
$\bar{B}+1$ = 1101101
1111110
E = 0
A = 1111110
$A \leftarrow \bar{A}+1$ = 0000010
$A_s \leftarrow \bar{A}_s = 0$
Result : A_s A
0 0000010
$\Rightarrow 2$

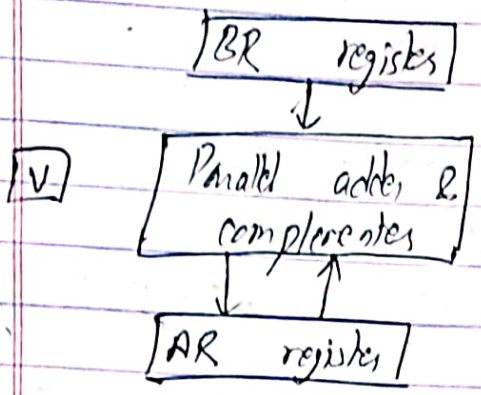
c) $(-17) - (-19)$

Minuend (A) = 0010001
Subtrahend (B) = 0010011
$A_s = 1$ $B_s = 1$
$A_s \oplus B_s = 0$
A = 0010001
$\bar{B}+1$ = 1101101
1111110
E = 0
A = 1111110
$A \leftarrow \bar{A}+1$ = 0000010
$A_s \leftarrow \bar{A}_s = 0$
Result : A_s A
0 0000010
$\Rightarrow 2$

Scanned by CamScanner

b) Addition & Subtraction with signed 2's complement data

Hardware Implementation



Algorithm :-

Subtraction operation

Minuend in AR
Subtrahend in BR

$AR \leftarrow AR + \overline{BR} + 1$
$V \leftarrow$ overflow

End
result in AR

Add operation

Augend in AR
Addend in BR

$AR \leftarrow AR + BR$
$V \leftarrow$ overflow

End
result in AR

Numerical Example

a) (-17)
$-(-14)$

Sol/

Assuming 8-bit representation

$+17 \Rightarrow 00010001$

$AR \Rightarrow -17 \Rightarrow 11101111$

$+14 \Rightarrow 00001110$

$BR \Rightarrow -14 \Rightarrow 11110010$

$AR \Rightarrow 11101111$

$+ \overline{BR} + 1 \Rightarrow 00001110$

11111101

Here, sign bit = 1 so result is in 2's complement form

$\Rightarrow -00000011$

$\Rightarrow -3$

(-17)
$+(-14)$

Sol/

Assuming 8-bit representation

$+17 \Rightarrow 00010001$

$AR \Rightarrow -17 \Rightarrow 11101111$

$+14 \Rightarrow 00001110$

$BR \Rightarrow -14 \Rightarrow 11110010$

$AR \Rightarrow 11101111$

$BR \Rightarrow 11110010$

$\boxed{1}11100001$

Here sign bit = 1 so result is in 2's complement form

$\Rightarrow -00011111$

$\Rightarrow -31$

Scanned by CamScanner

c) $(+15)$
$+(-13)$
So/

Assuming 8 - bit

AR => +15 => 00001111

+13 => 00001101

BR => -13 => 11110011

AR => 0000 1111

BR => 1111 0011

100000010

~~V~~

Sign bit = 0 So.

required result = 00000010

= +2

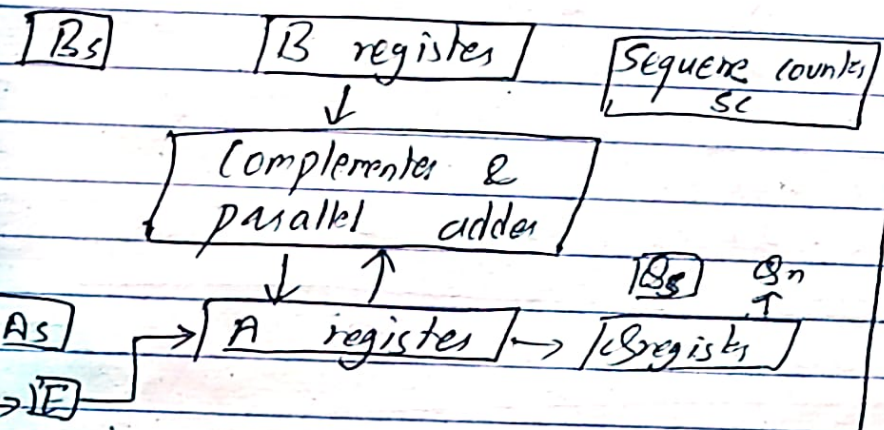
Scanned by CamScanner

Multiplication Algorithm

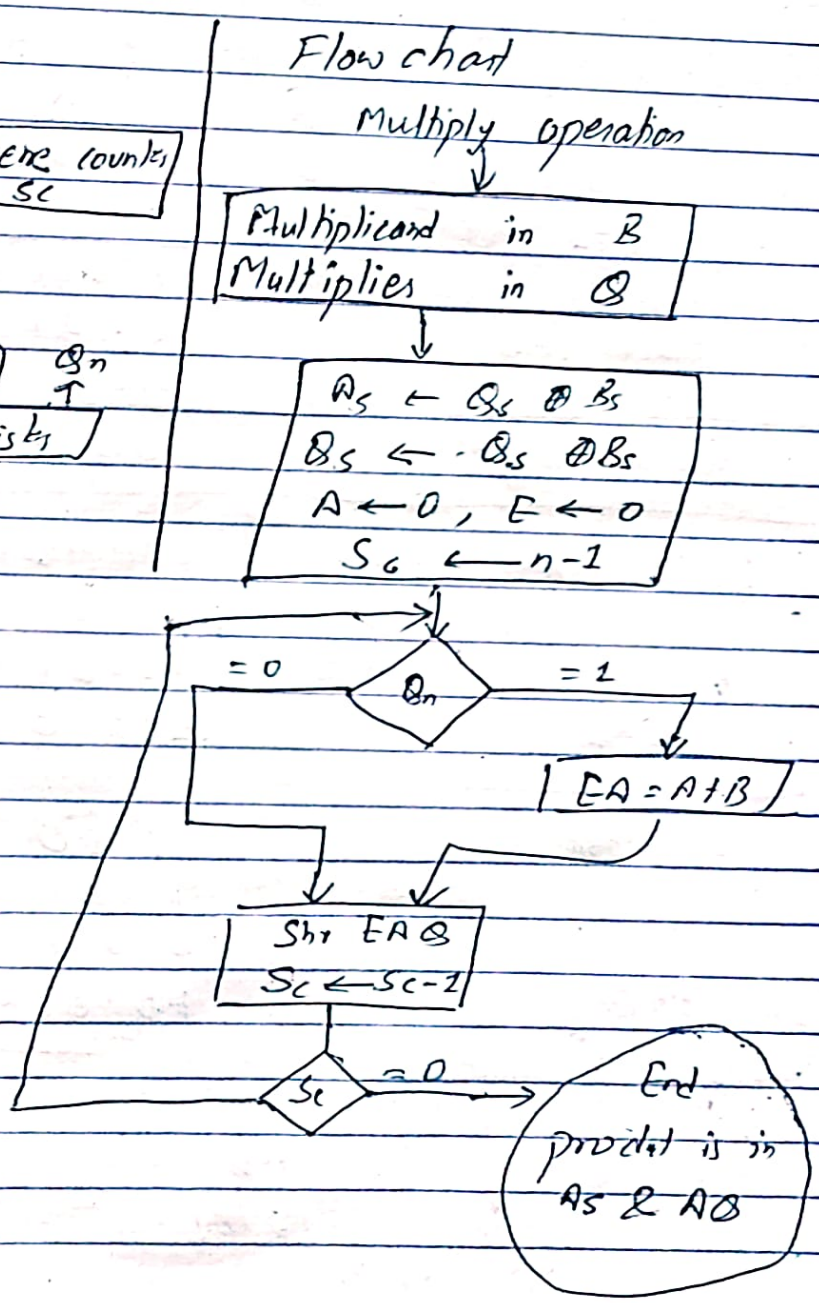
a) Multiplication with signed magnitude data
b) Multiplication with signed signed 2's complement data
(Booths multiplication algorithm)

a) Multiplication with signed magnitude data

Hardware Implementation



Flow chart



Scanned by CamScanner

Numerical Example

a) (-27)
× 13

Multiplicand (B) = 27 = 11011 $B_s = 1$
Multiplier (Q) = 13 = 1101 $\oplus\ Q_s = 0$
$A_s = 1$

Comment	E	A	Q	S_c
Initially,	0	00000	1101	4
$Q_n = 1$, add		11011		
	0	11011		
Shr EAQ		01101	1110	3
	0			
$Q_n = 0$ Shr EAQ		00110	1111	2
$Q_n = 1$, add		00110		
		11011		
	1	00001		
Shr EAQ		10000	1111	1
$Q_n = 1$, add		10000		
		11011		
	1	01011		
Shr EAQ		10101	1111	0

A_s	AQ
1	10101 1111

⇒ -351

14
12

Multiplicand (B) = 14 = 1110
Multiplier (Q) = 12 = 1100

B_s 0
$\oplus$ Q_s 0
$A_s = 0$

mment	E	A	Q	Sc
tially	0	0000	1100	4
0 Sh. EAQ	0	0000	0110	3
0 Sh EAQ	0	0000	0011	2
add		0000 1110 1110		
EAQ	0	0111	0001	1
add		0111 1110 0101		
AQ	1	1010	1000	0

A_s	AQ	
0	1010	1000

$\Rightarrow$ 168

Scanned by CamScanner

c) (-15)
x(-9)

Sol

Multiplicand (B) = -15 = 1111 $B_s = 1$

Multiplier (Q) = -9 = 1001 $\oplus$ $Q_s = 1$

$A_s = 0$

Comment	E	A	Q	S_c
Initially,	0	0000	1001	4
$Q_n = 1$ add		0000 1111 1111		
Shr EAQ	0	0111	1100	3
$Q_n = 0$ Shr EAQ	0	0011	1110	2
$Q_n = 0$ Shr EAQ		0001	1111	1
$Q_n = 1$ add	1	0001 1111 0000		
Shr EAQ		1000	0111	0

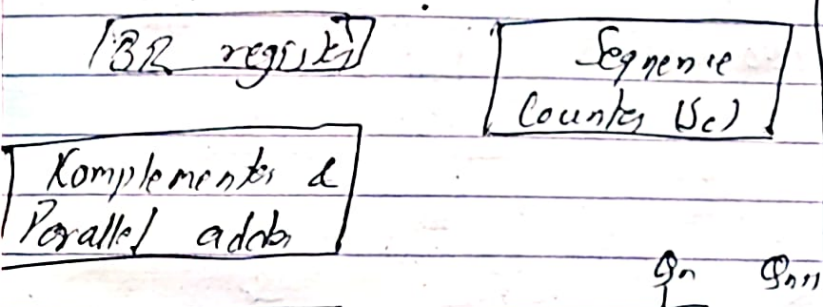
A_s	A	
0	1000	0111

=> -135

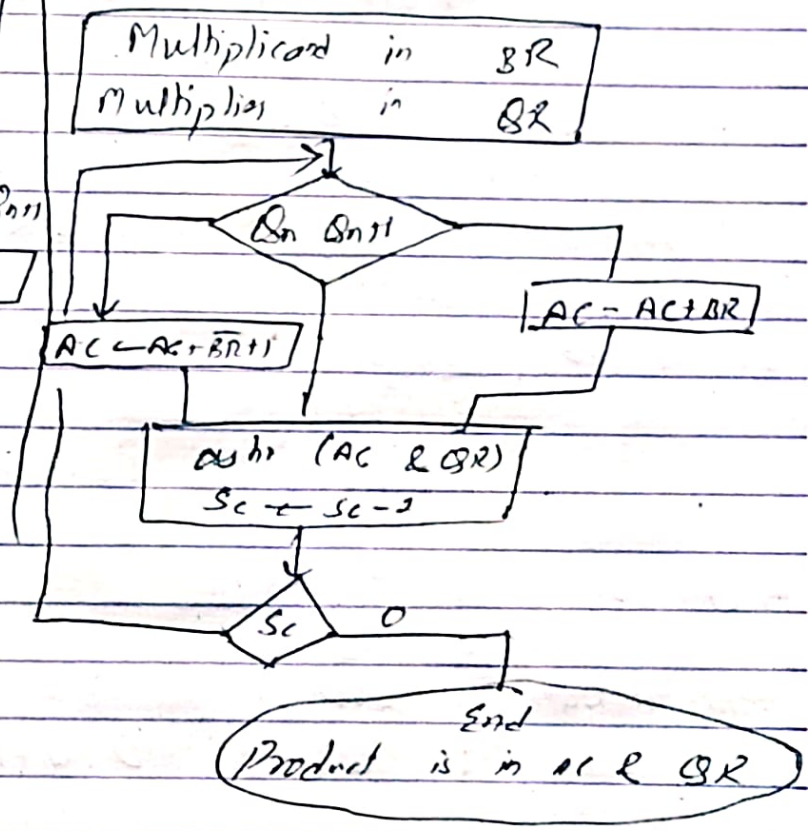
Scanned by CamScanner

Multiplication with signed 2's complement data

Hardware Implementation



Flowchart Multiply



Scanned by CamScanner

N.E ~~A~~ -27

$\times 13$

$+27 = 011011$

Multiplicand $(BR) = 100101$

$\overline{BR} + 1 = 011011$

$QR = 13 = 01101$

~~Result~~

~~AC~~ ~~QR~~

~~110101~~ ~~00001~~

Comment	AC	QR	Q_{n+1}	Sc
Initially,	000000	01101	0	5
$Q_nQ_{n+1} = 10$	011011			
	011011			
	001101	10110	1	4
$Q_nQ_{n+1} = 01$	001101			
	100101			
~~Ashr (AC&QR)~~	110010			
Ashr (AC&QR)	111001	01011	0	3
$Q_nQ_{n+1} = 10$	111001			
	011011			
	010100			
Ashr (AC&QR)	001010	00101	1	2
$Q_nQ_{n+1} = 11$				
Ashr (AC&QR)	000101	00010	1	1
$Q_nQ_{n+1} = 01$	000101			
	100101			
	101010			
Ashr (AC&QR)	110101	00001	0	0

Scanned by CamScanner

result = AC & QR

= 110101 00001

Sign bit = 1, so result is in -ve

in 2's complement of 110101 00001

= - 001010111 11

= - (256 + 64 + 16 + 8 + 4 + 2 + 1)

= -351

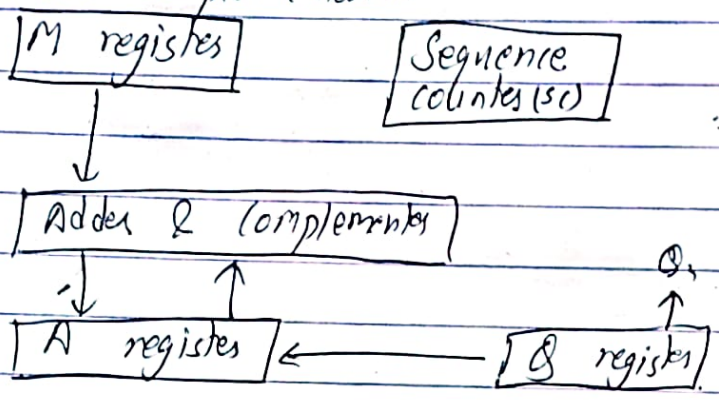
Scanned by CamScanner

Division algorithm
→ Restoring division algorithm
→ Non-restoring division algorithm

Restoring devision algorithm

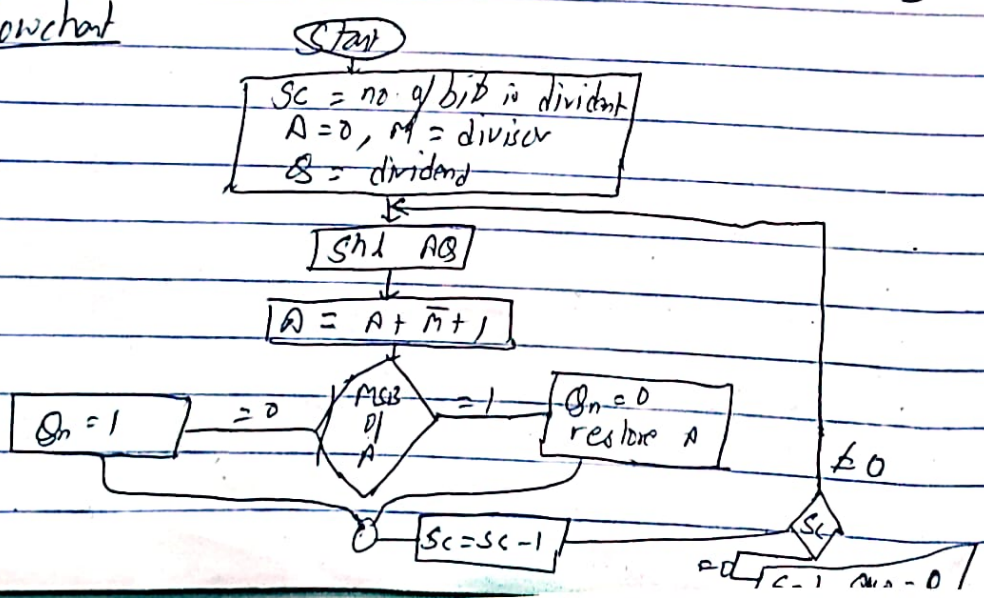
→ It provides a quotient & a remainder, when we divide two numbers.

Hardware Implementation



Registes a contain quotient & registes contain remainder n-bit dividend is loaded in Q & divisor in m value of registes A is initially kept 0 & this is the registes whose value is restored during iteration due to which it is named restoring

Flowchart



Shl A
A = A+
MSB of A =
Q_n = 0, res

Shl AQ
A = A + $\bar{M}$ + 1
MSB of A = 1
Q_n = 0 restore

Shl AQ
A = A + $\bar{M}$ + 1
MSB of A = 0

Shl AQ
A = A + $\bar{M}$ + 1
MSB of A = 1
Q_n = 0, restore A

Scanned by CamScanner

Numerical Example

13/5

Sol

Dividend = Q = 13 = 1101

Divisor = M = 5 = 0101

$\bar{M}+1$ = 1011

Comment	A	Q	Sc
ially	0000	1101	4
AQ	0001	101□	
$\bar{M}+1$	1011		
	1100		
es tore A →	0001	1010	3
	0011	010□	
1	1011		
	1110		
r A	0011	0100	2
	0110	100□	
	1011		
	(1) 0001		
0	0001	1001	1
	0011	001□	
1	1011		
	1110	0010	
A	0011		0

Quotient = Q = 0010 = 2

Remainder = A = 0011 = 3

Scanned by CamScanner

b) 17/3

Sol/ dividend = 17 = 10001 = Q

divisor = 3 = 00011 = M

$\bar{M}+1$ = 11101

Comment	A	Q	Sc
initially	00000	10001	5
Shl AQ	00001	0001□	
$A = A + \bar{M} + 1$	11101		
MSB of A = 1	11110		
$Q_n = 0$ restore A	00001	00010	4
Shl AQ	00010	0010□	
$A = A + \bar{M} + 1$	11101		
MSB of A = 1	11111		
$Q_n = 0$ restore A	00010	00100	3
Shl AQ	00100	0100□	
$A = A + \bar{M} + 1$	11101		
MSB of A = 0	1 00001		
$Q_n = 1$	00001	01001	2
Shl AQ	00010	1001□	
$A = A + \bar{M} + 1$	11101		
MSB of A = 1	11111		
$Q_n = 0$ restore A	00010	10010	1
Shl AQ	00101	0010□	
$A = A + \bar{M} + 1$	11101		
	1 00010	10101	0

A = 00010 = 2

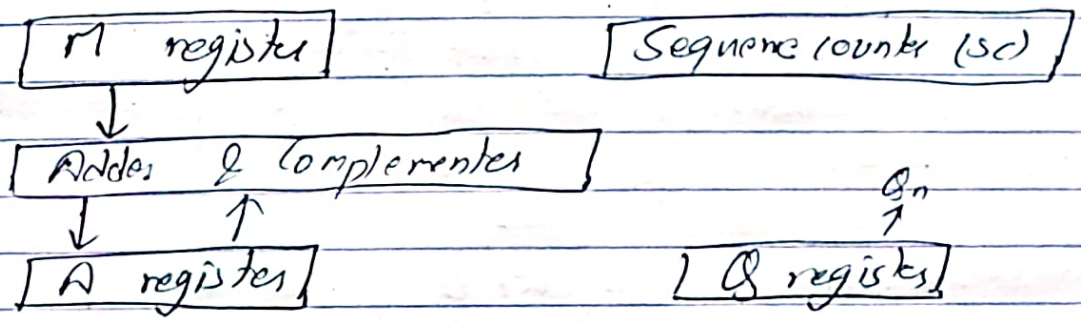
Q = 00101

non - restoring division algorithm

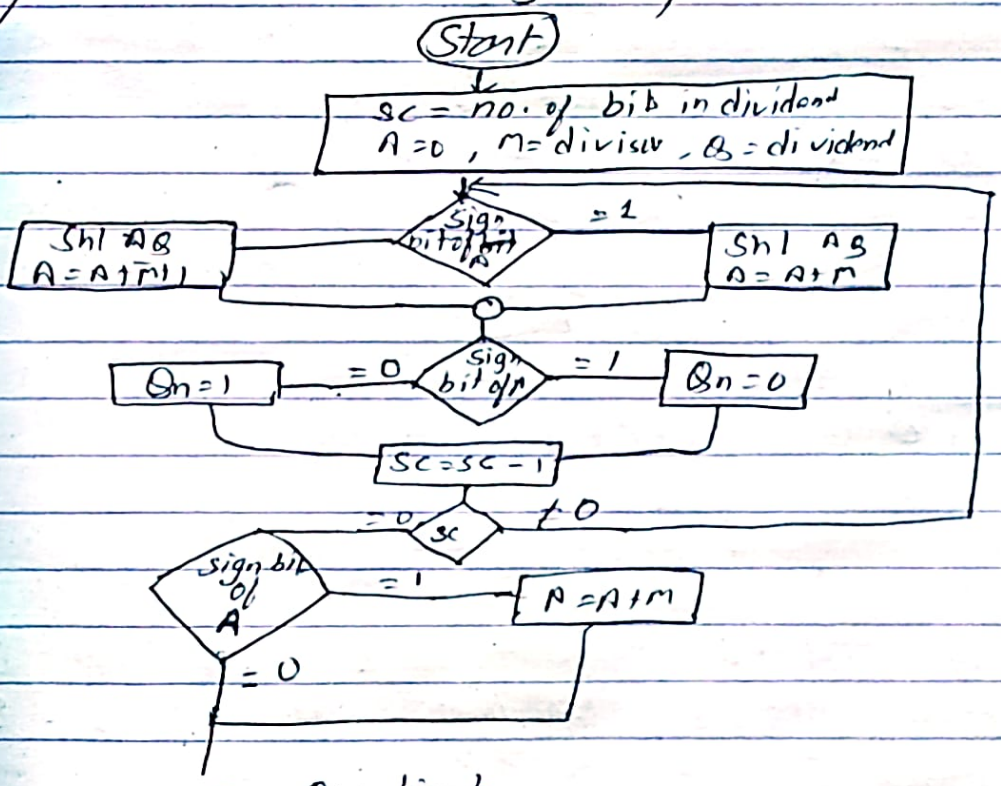
It provides a quotient & a remainder when we divide two numbers.

Improved version of restoring division algorithm

Hardware Implementation



Register a contain Quotient & register A contain remainder n-bit dividend is loaded in Q & divisor in m. value of register A is initially kept 0.



Q = Quotient

A = Remainder

Scanned by CamScanner

a) 13/5

50/

dividend = 13 => 1101 (Q)

divisor = M => 0101

$\bar{M}+1$ = 1011

Comment	A	Q	SC
initially,	0000	1101	4
sign of A = 0	0001	1010	
A = A + $\bar{M}$ + 1	1011		
sign of A = 1	1100	1010	3
sign of A = 1	1001	0101	
A = A + M	0101		
sign of A = 1	1110	0100	2
sign of A = 1	1100	1001	
A = A + M	0101		
sign of A = 0	1 0001	1001	1
sign of A = 0	0011	0011	
A = A + $\bar{M}$ + 1	1011		
sign of A = 1	1110	0010	0

sign of A = 1

A = A + M

1110
0101
1 0011

remainder = A = 0011 = 3

Quotient = Q = 0010 = 2

Unit 8

Input - Output Organization

Peripheral device

→ Provides an efficient mode of communication between the central system & the outside environment.

Types

Input device → Keyboard, mouse etc
Output device → Printer, monitor etc
Input - Output device → Memory

Input Output Interface

→ Provides a method for transferring information between internal storage & external I/O devices

→ Peripheral devices are connected to CPU via a special communication links and its purpose is to resolve difference exists in between CPU & peripheral devices

a) Peripheral devices are electromechanical & electromagnetic where as CPU is purely electronic so signal representation is different ⇒ signal conversion required

b) Data transfer rate mismatched ⇒ synchronization is required

c) The operating modes of peripherals are different from each other & each must be controlled so as not to disturb the operation of other peripherals connected to CPU
⇒ device selection required.

d) Data codes & formats in peripherals differ from the word format in the CPU & memory

∴ function of interface ⇒ synchronization, device selection, signal conversion.

Scanned by CamScanner

I/O Bus & Interface module

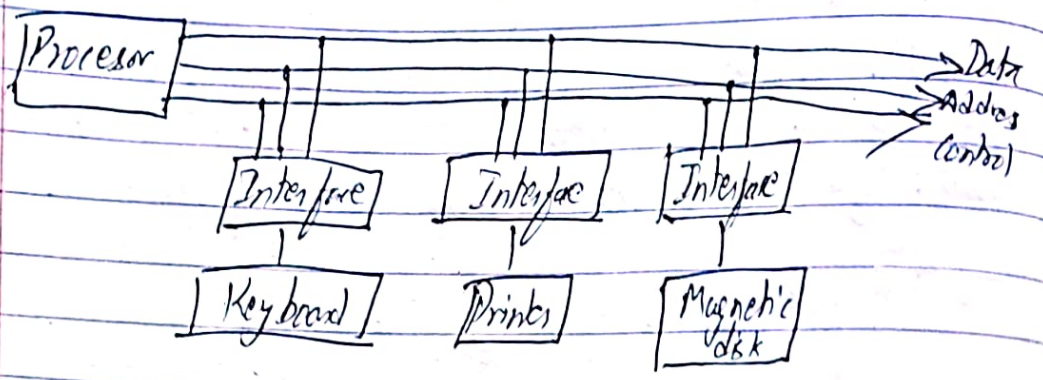


fig:- connection of I/O bus to I/O devices

Function codes or I/O command => issued by CPU and is executed by Interface & peripherals

Types

a) Control command :- issued to activate the peripheral & to inform it what to do.

b) Status command :- used to test various status condition in the interface & peripherals eg:- error during data transfer, completion successfully etc.

c) Data I/O command :- Opposite of data o/p command

I/O Verses Memory Bus

→ Three ways that computer buses can be used to communicate with memory & I/O

a) Use two seperate buses, one ~~of~~ for memory & another for I/O

b) Use one common bus for both memory & I/O but have seperate control lines for each

c) Use one common bus for memory & I/O with

common control line.

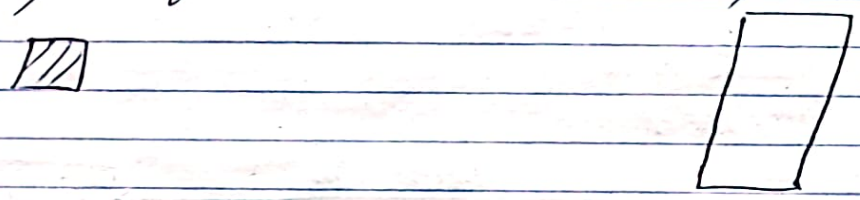
Isolated Versus Memory - mapped I/O

Isolated I/O

→ CPU has distinct input & output, and memory command instruction

→ Different address space for I/O & memory operation

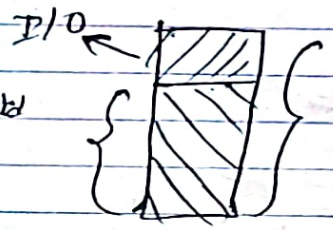
Address space for I/O Address space for memory



Memory - Mapped I/O

No distinct I/O & Memory Instruction

No distinct address space rather some address from ~~same~~ same address space is allocated for I/O & remaining address is used for memory operation.



So here effective address space used for memory is reduced.

Scanned by CamScanner

Asynchronous Data Transfer

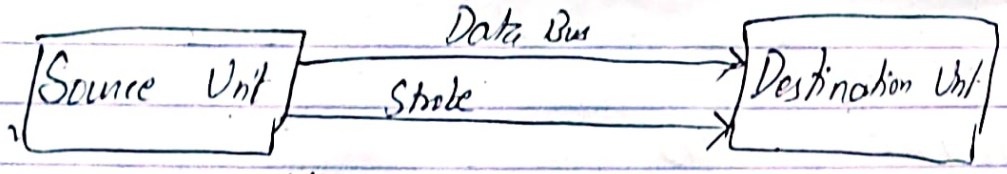
→ Generally communication in between CPU & peripherals is performed asynchronously

→ In this mode of transfer, there is need of control signals to be transmitted between the communicating unit to indicate the time at which data is being transmitted.

→ For this we have two approach :-

1) Strobe method
2) Handshaking method

Strobe Method



a) block diagram

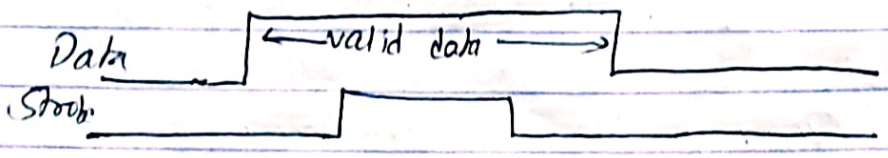
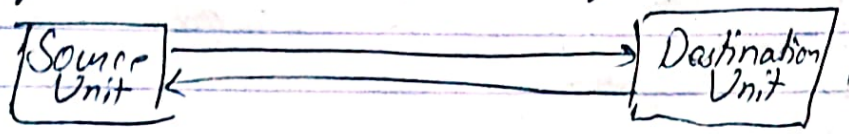


fig :- timining diagram

fig :- source - initiated strobe for data



a) Block diagram

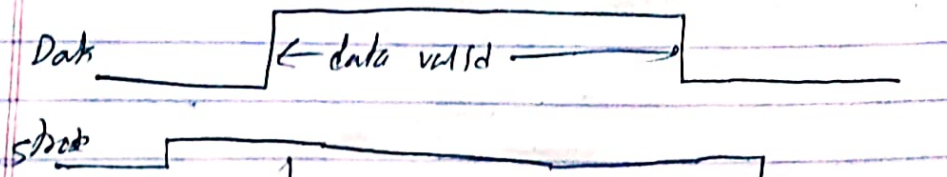


fig :- timing diagram

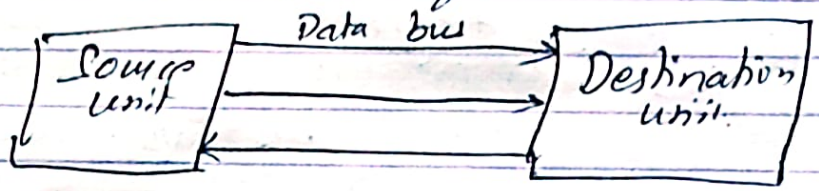
fig :- destination initiated strobe for data transfer

The strobe is a single line that informs the destination until when a valid data word is available in the bus.

Advantage :-
(Source - initiated) => No way of knowing whether the destination unit has actually received whether the data item that was placed in the bus
(Destination - initiated) => No way of knowing whether the source unit has actually placed data on data bus.

Handshaking Method

Solves the limitation of strobe method



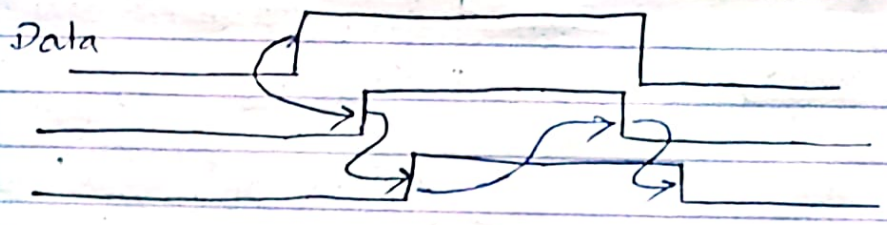


fig:- Timing diagram

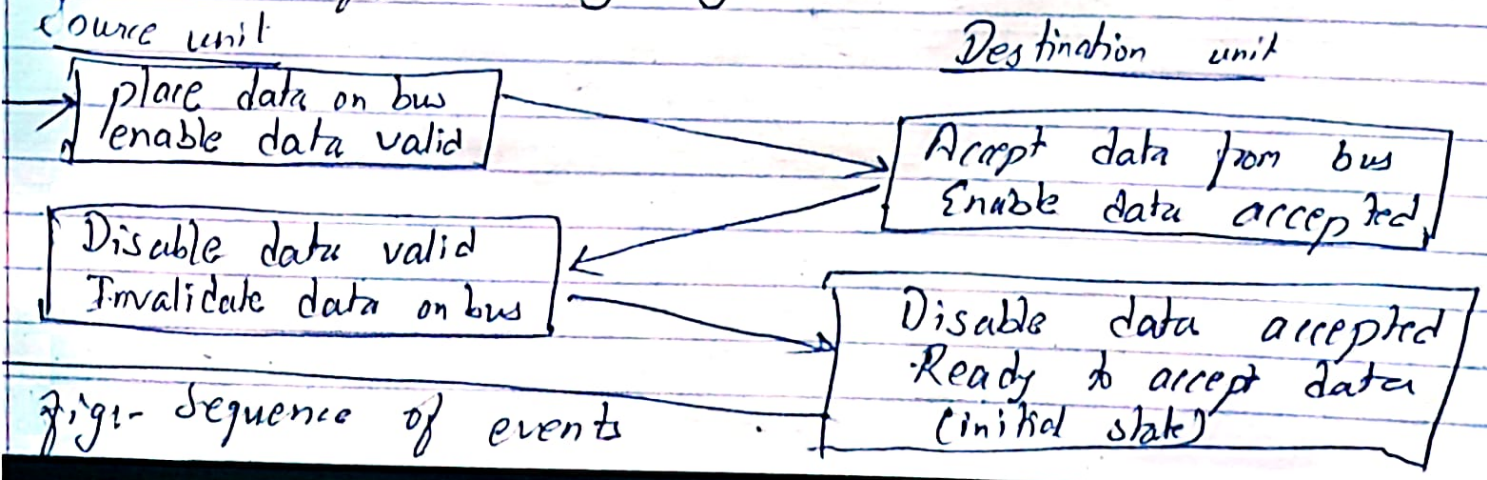


fig:- Sequence of events

Destination initiated transfer using handshaking

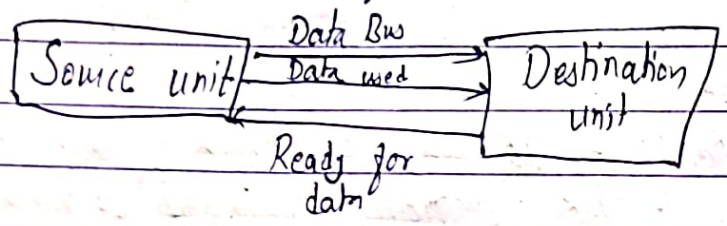


Fig :- Block diagram

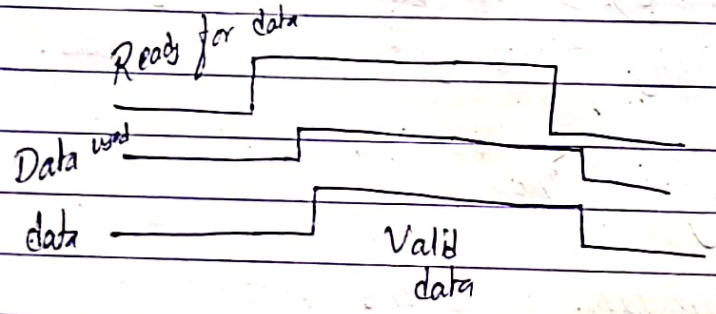


fig :- Timing diagram

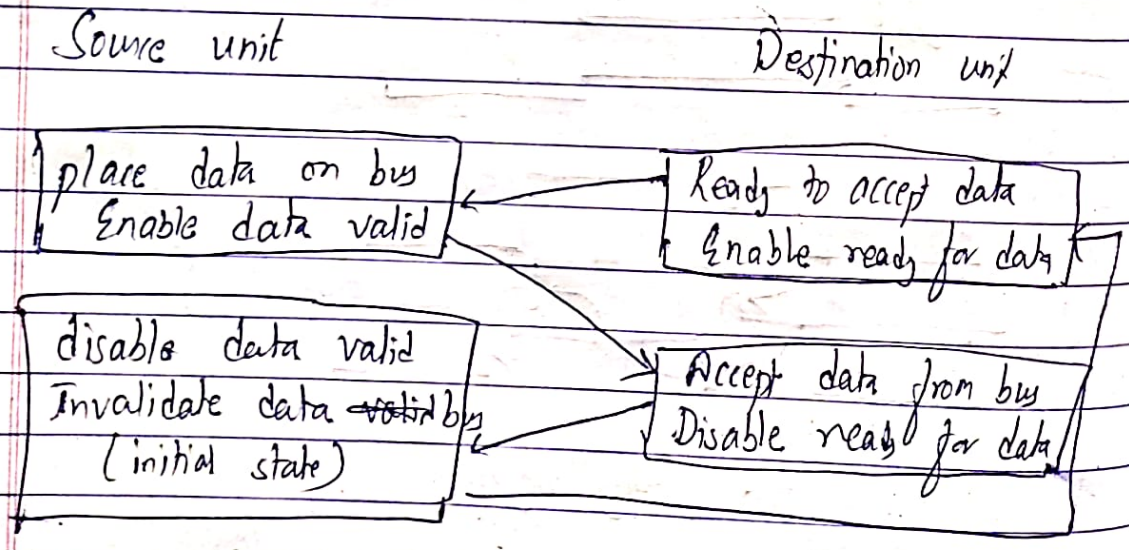
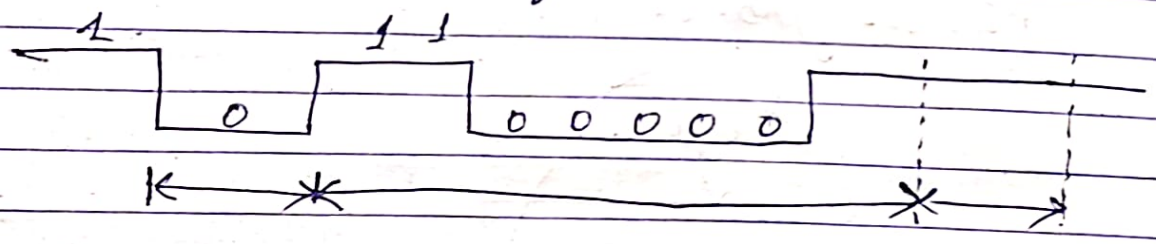


fig :- sequence of events

Asynchronous serial transfer



Eg :- UART (Universal asynchronous receiver Transmitter) is used for asynchronous serial transfer.

Mode of transfer :-

Data transfer to and from peripherals can be handled in one of three possible modes

a) Programmed I/O :
- → Constantly monitor I/O device
- → Inefficient use of CPU

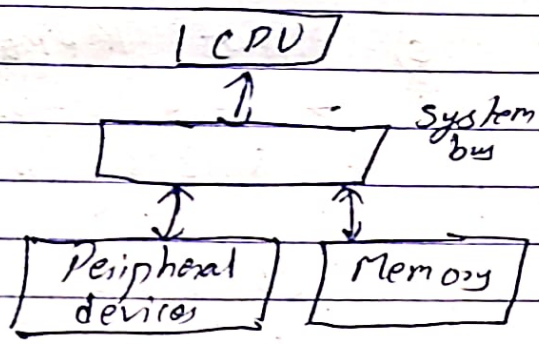


fig : programmed I/O

b) Interrupt
- → CPU doesnot monitor pheripherals
- → I/O devices generates interrupt signal to CPU when it is ready for data transfer
- → One interrupt signal is received by CPU, it stops momentarily, its ongoing task and branch to ISR to serve I/O devices.

Scanned by CamScanner

→ After serving I/O device, CPU resumes its previous job.

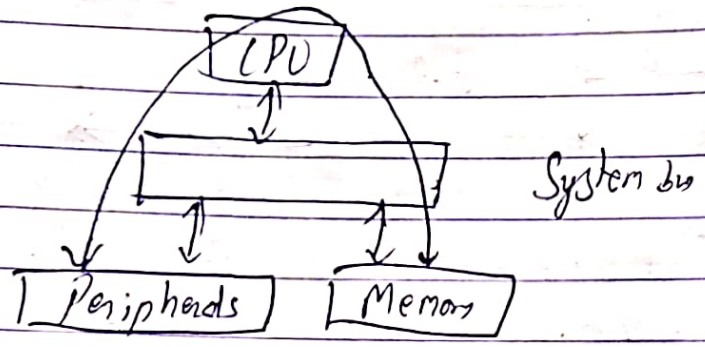


fig:- Interrupt

→ efficient use of CPU

Interrupt address :-
a) Vectored address
b) Non-Vectored address

c) Direct Memory address (DMA)
→ Very useful for bulk data transfer eg data transfer in between memory & harddisk.
→ Data transfer between peripherals & memory is supervised by DMA controller without CPU intervation.
→ CPU remains idle & it can executes instruction that doesnt requires the access of system bus

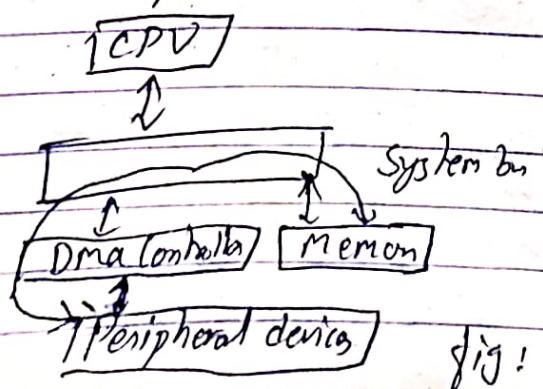


fig: DMA.

Scanned by CamScanner

Priority Interrupt

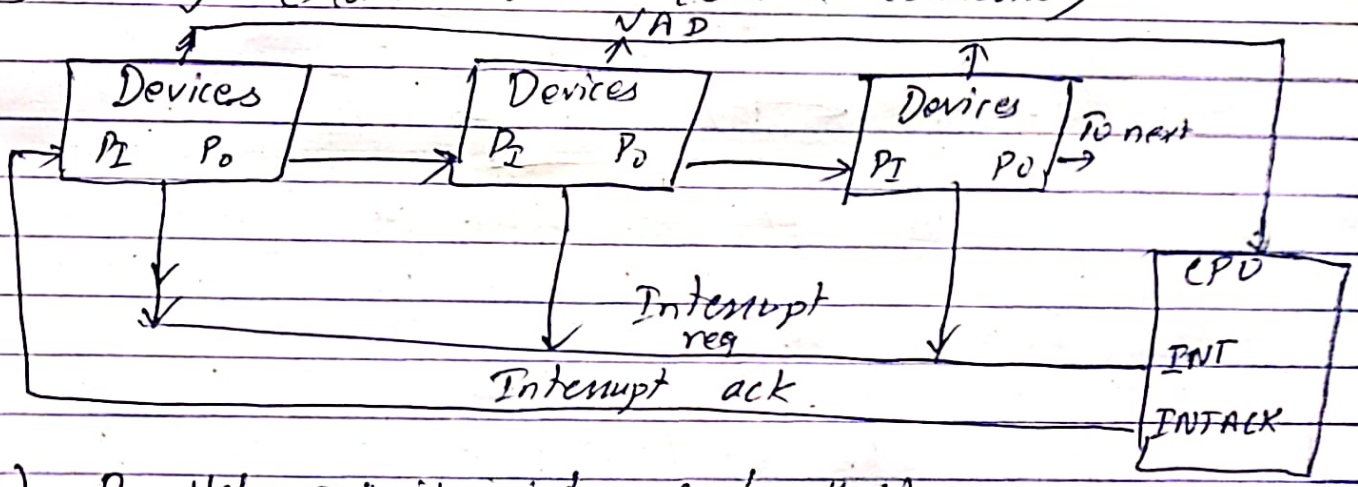
How can CPU handle simultaneous Interrupt from different Interrupt source.

Two approach

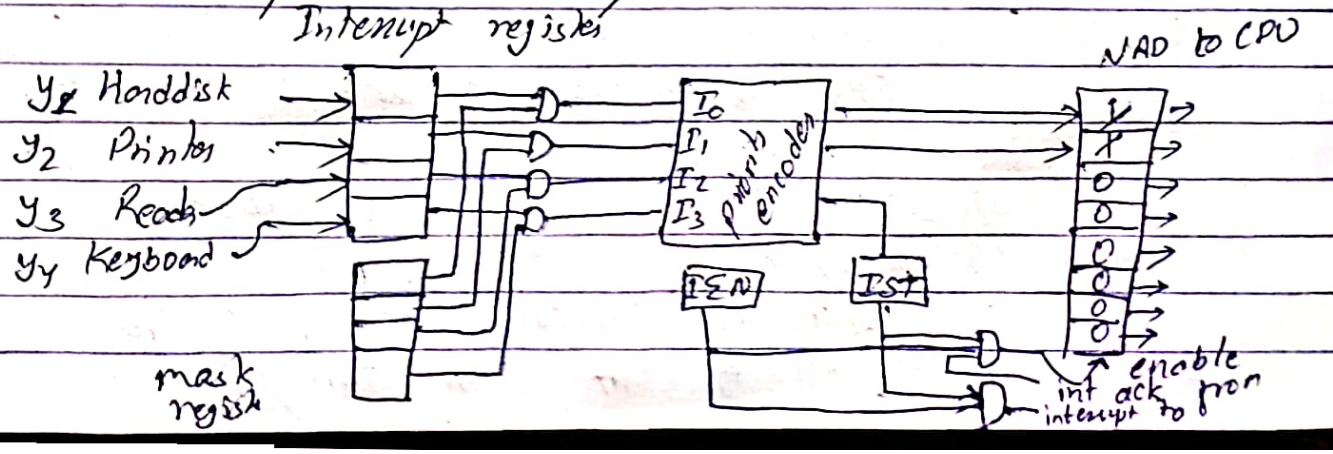
a) Software approach
↳ polling procedure
disadvantage :- CPU // very slow

b) Hardware approach :-
a) Daisy chain method
b) Parallel priority method

a) Daisy Chain method (serial connection)



b) Parallel priority interrupt (method) :-



Scanned by CamScanner

IEN = Interrupt Enable
IST = Interrupt status flip flop

Function table of priority Encoder

I_0	I_1	I_2	I_3	X	Y	IST
0	0	0	0	x	x	0
1	x	x	x	0	0	1
0	1	x	x	0	1	1
0	0	1	x	1	0	1
0	0	0	1	1	1	1

Direct Memory access :

How DMA works ?

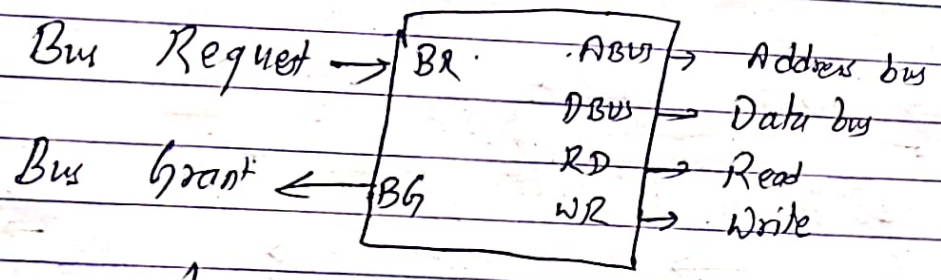


fig :- CPU bus signal for DMA transfer

DMA Transfer mode :

a) Burst transfer
b) Cycle stealing

DMA Controller :

CPU initialized the DMA by sending the following info :-

a) Starting address of memory block
b) Word count
c) Control to specify mode of transfer
d) Control to start DMA transfer

Scanned by CamScanner

DMA controller cont.....

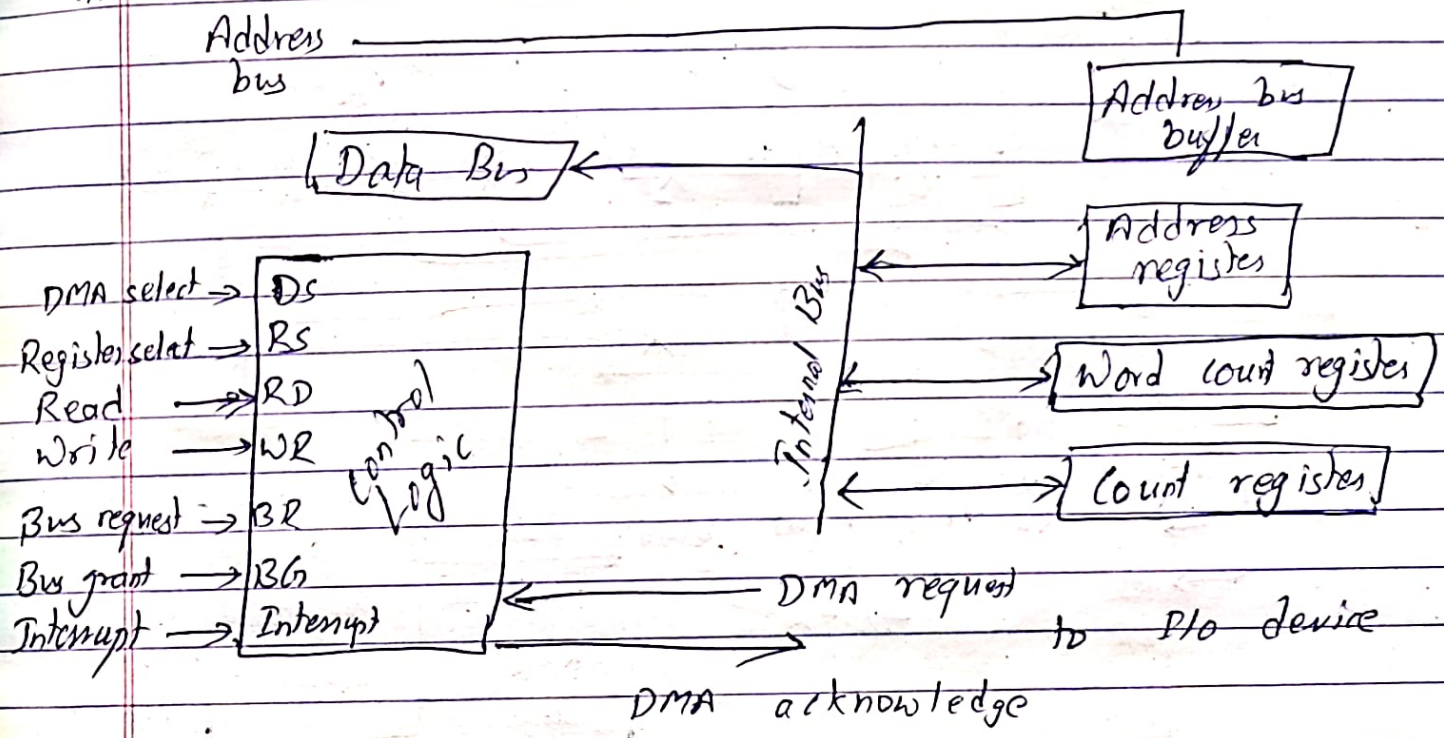
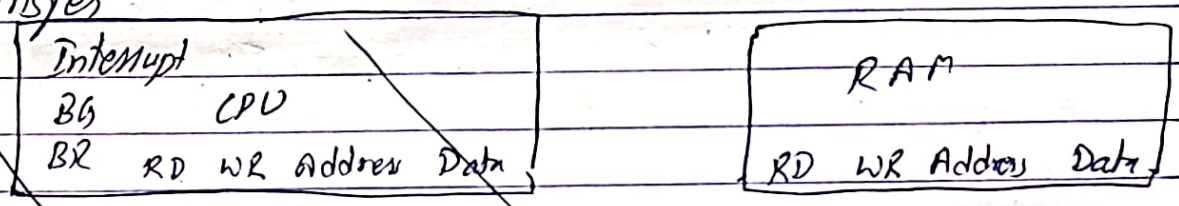


fig :- block diagram of DMA controller

DMA transfer



Scanned by CamScanner

DMA transfer

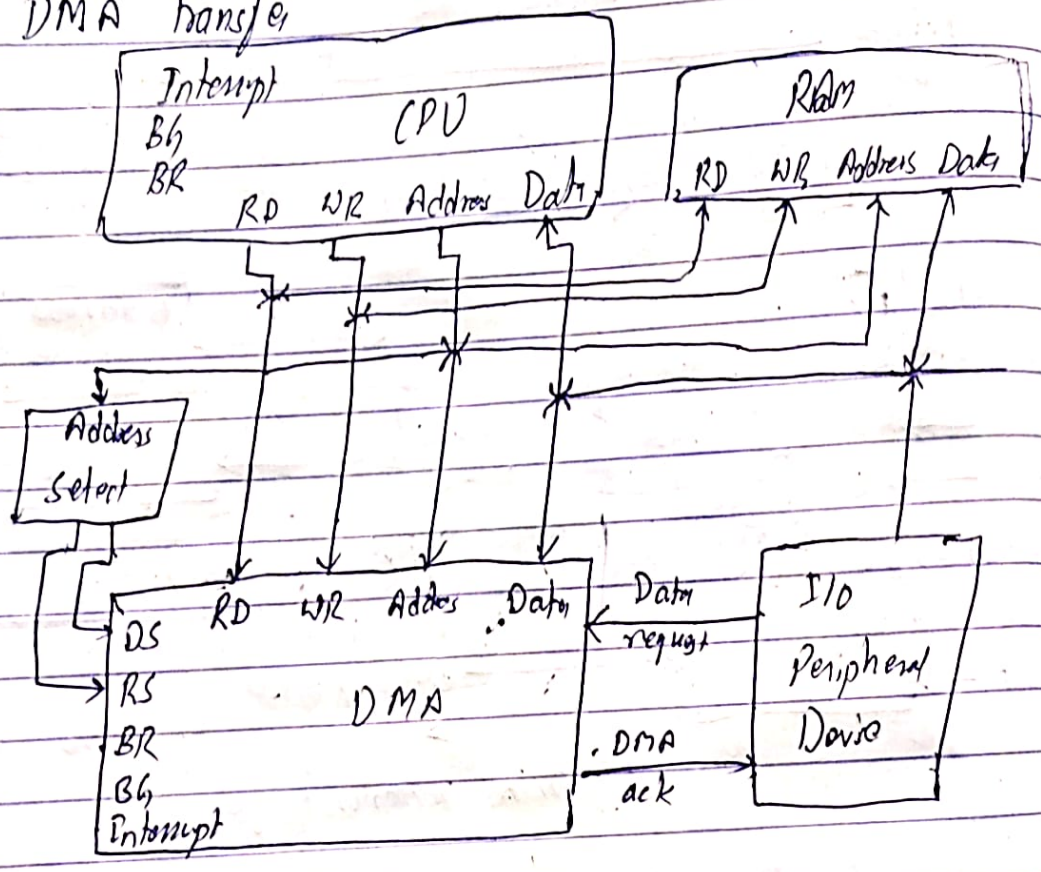


fig: DMA transfer in computer system

Input - Output Processor (IOP)

→ processor with DMA capability that communicates with I/O devices

→ Data communication processor (DSP) ⇒ communicate with remote terminals over telephone & other communication media in a serial fashion.

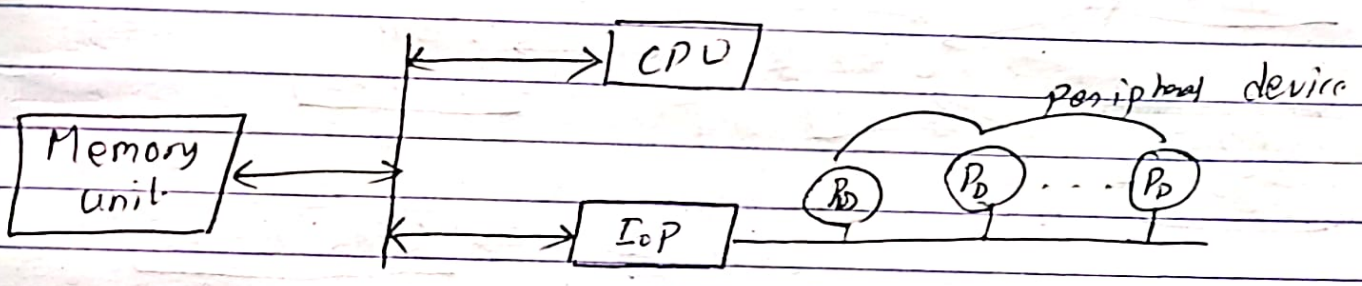
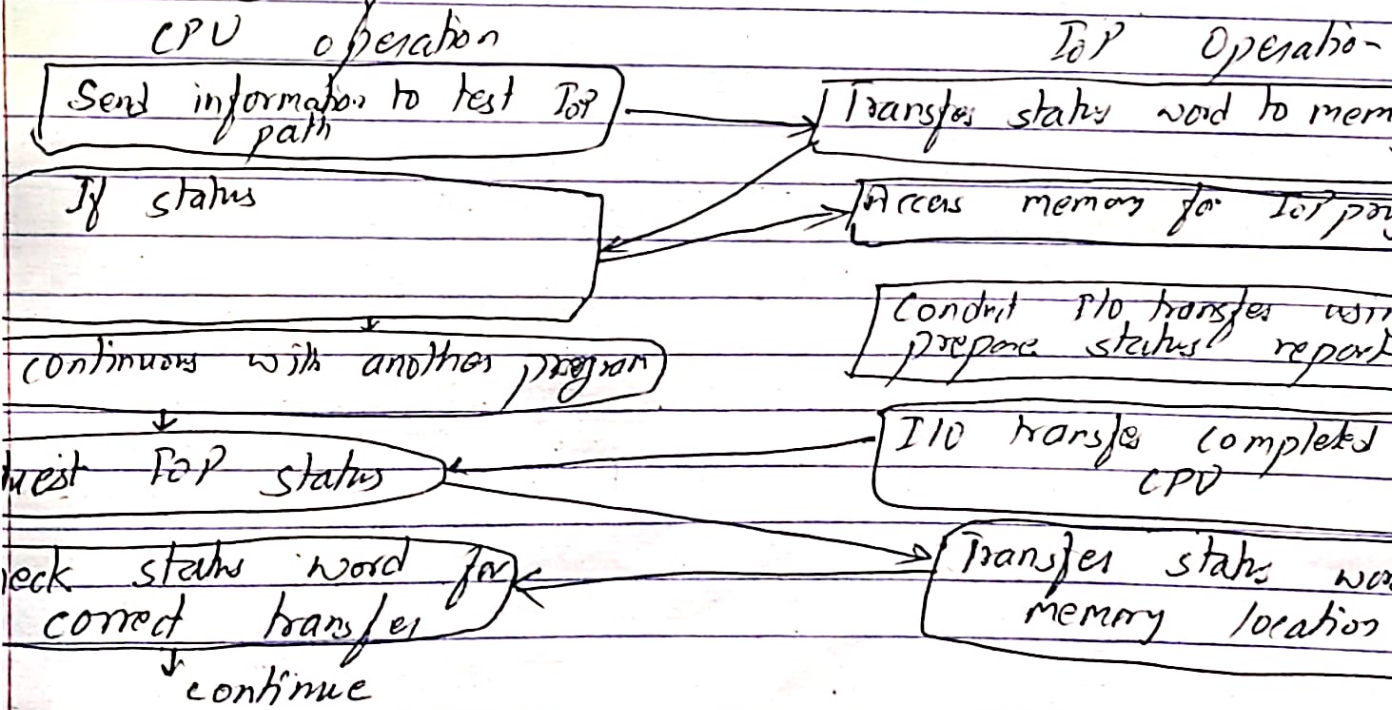


fig :- Block diagram of a computer with I/O processor.

CPU - IOP communication

How CPU communicates with IOP ? Explain with necessary flowchart.



Scanned by CamScanner

Memory organization

Memory hierarchy

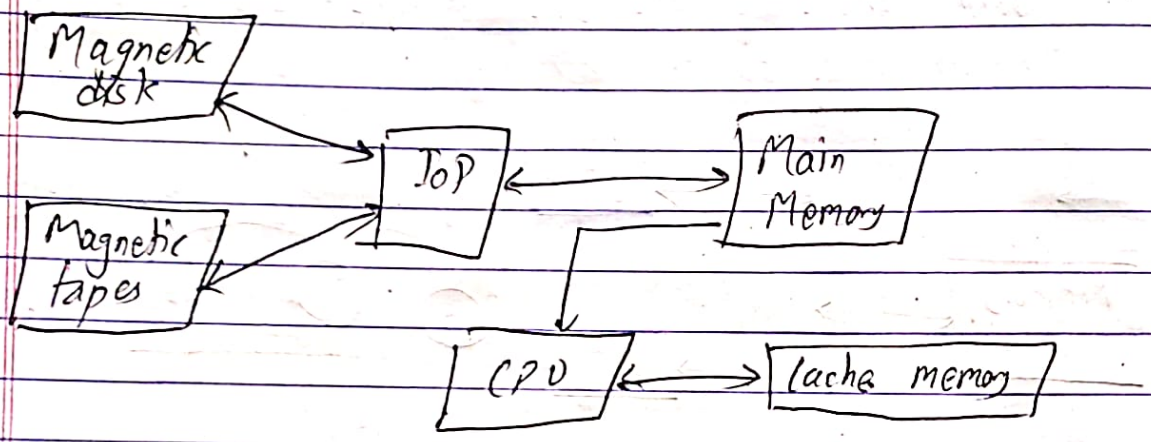


fig:- Memory hierarchy used in Computer System

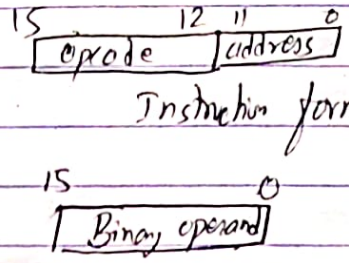
⇒ Goal: To provide maximum possible average access speed while minimizing the overall cost of computer system.

Scanned by CamScanner

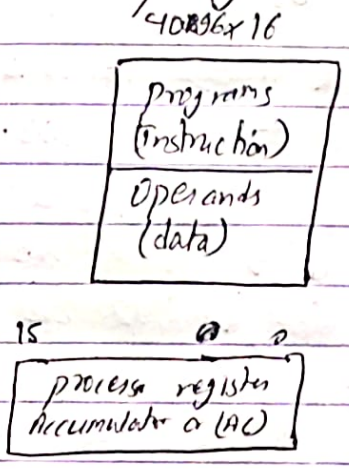
Unit 3

Basic Computer Organization & Design

Basic Computer (Stored program organization)



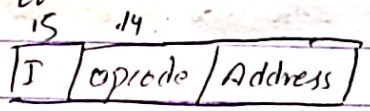
Instruction format



Addressing mode of Basic computer

1) Direct Address :- Second part ~~compute~~ specifies effective address
2) Indirect Address :- second part specifies address of effective address

*) Effective address :- address of operands



a) Instruction formats

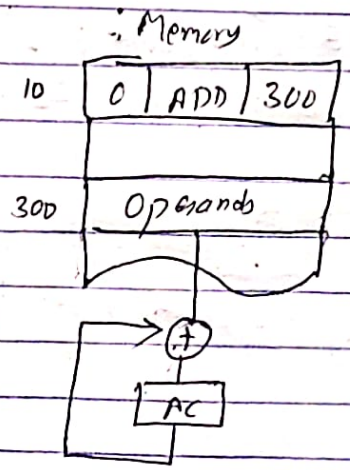


fig:- direct address

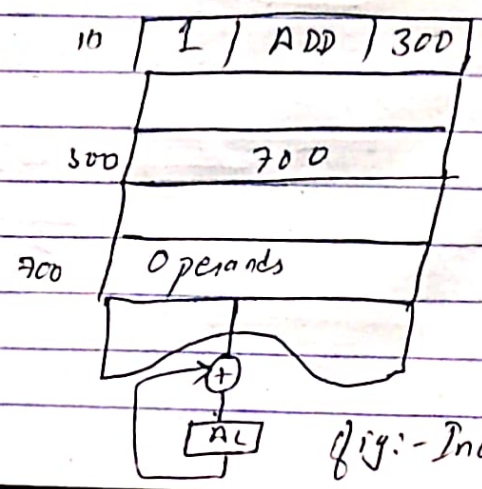


fig:- Indirect address

Scanned by CamScanner

Computer registers

List of available register in basic computer

i) Data register (DR) => 16 bits
ii) Address register (AR) => 12 bits
iii) Accumulator (AC) => 16 bits
iv) Instruction register (IR) => 16 bits
v) Program counter (PC) => 12 bits
vi) Temporary register (TR) => 16 bits
vii) Input register (INPR) = 8 bits
viii) Output register (OUTR) = 8 bits

Instruction format of basic computer

15	14 … 12	11 … 0
I	Opcode	Address

(opcode = 000 through 110)

i) Memory reference instruction

Instruction

AND, ADD, LDA, STA, BUN, BSA, ISZ

15	14 … 12	11 … 0
0	111	register operation

(opcode = 111 & I = 0)

ii) Register reference instruction

Instruction

CLA, CLE, CMA, CME, CIR, CIL, INC, SPA, SNA, SZA, SZE, HLT

15	14 … 12	11 … 0
1	111	I/O operation

opcode = 111 & I = 1

iii) I/O instruction

Instruction :-

INP, OUT, SKI, SKO, ION, IOF

Instruction set completeness

Is instruction of basic computer complete?
Yes, the instruction set of basic computer is complete since it has sufficient number of instruction to perform the following operation.

a) Arithmetic, logic & Shift operation

Arithmetic
- → ADD ⇒ ADD, instruction
- → Substraction ⇒ ADD, CMA, INC
- → Division ⇒ Shifting + substraction
- → Multiplication ⇒ Shifting + addition

logic ⇒ AND + CMA ⇒ NAND

Shift ⇒ CIR & CIL

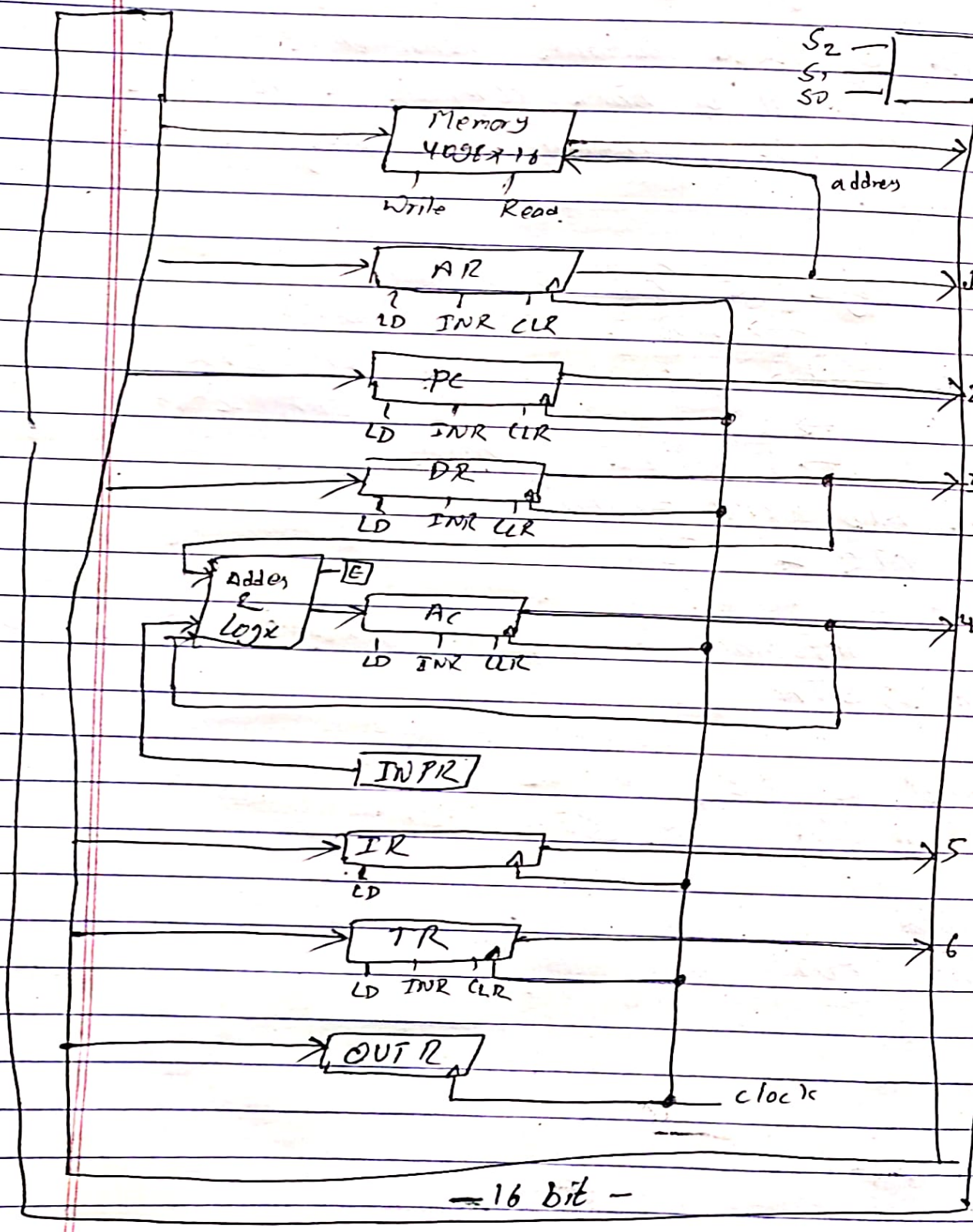
b) For moving information to & from memory & CPU registers
LDA, STA

c) For Branching & testing various condition
BUN, SPA, SNA, SZA, SZE

d) I/O operation
INP, OUT

Scanned by CamScanner

Common bus system



Scanned by CamScanner

Cache Memory

→ Cache memory works based on locality of reference principal.

→ Goal :- To improve the memory access time

→ Performance metrics :-

$$\text{Cache ratio} = \frac{\text{cache hit}}{\text{total reference}}$$

$$\text{Cache miss ratio} = 1 - \text{cache hit ratio} = \frac{\text{cache miss}}{\text{Total reference}}$$

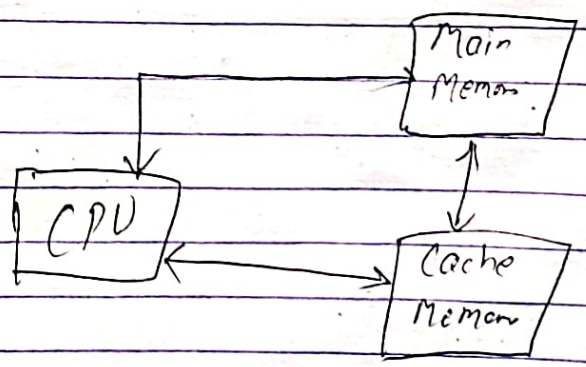
Num :-

i) Find the cache hit ratio and cache miss ratio, if the CPU find memory word in cache 110 times out of 130

ii) The average memory access time of main memory is 1000 ns and the cache memory is 100 ns. The CPU finds reference word in cache 85 times out of 120. Find Average memory access time.

iii) The average memory access time of main memory is 200 ns and cache memory access time is 20 ns. The cache hit ratio is 0.9. Find average memory access time.

Working



Scanned by CamScanner

2) SC

memory access time $(a_m) = 1000$ ns

cache " " $(a_c) = 100$ ns

" hit ration $(H_c) = \frac{85}{120} = 0.708$

" miss " $(M_c) = 1 - \frac{85}{120} = 0.29$

Avrage memory access time $= H_c * a_c + M_c (a_m + a_c)$

$= 0.708 \times 100 + 0.29 (1000 + 100)$

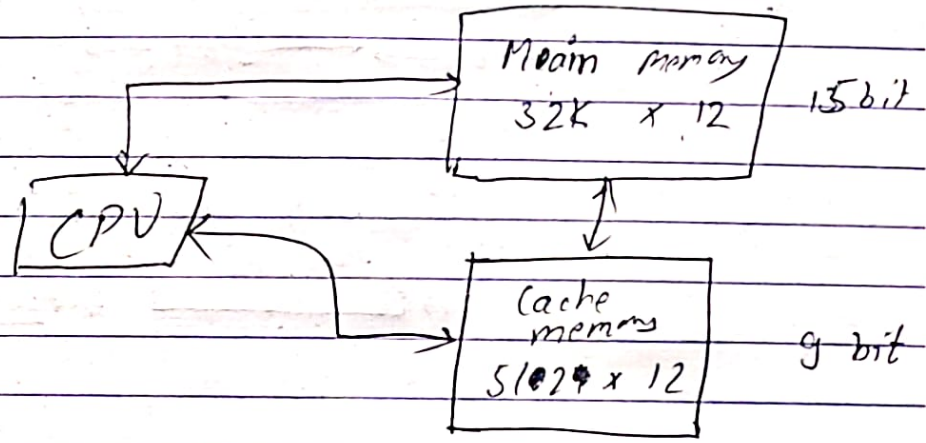
$= 389.8$ ns

Scanned by CamScanner

VVI

Cache mapping

The transformation of data from main memory to cache memory is called cache mapping



eg :-

a) Associative mapping
b) Direct mapping
c) Sort - Associative mapping

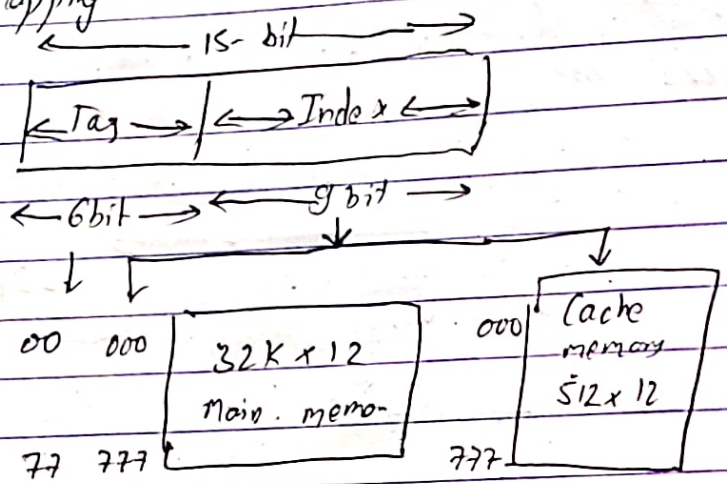
Associative mapping

60000 ← Argument

Address	Data	
00000	0002	0
00100	0030	0
01001	0440	0
70012	0550	0

Scanned by CamScanner

Direct mapping



Main Memory

00000	5000
00001	0010
100001	2400
60000	7007
60001	1005
60002	3004
77777	4004

	Tag	Data
000	00	5000
001		
006		
777		

Set Associative mapping

	Tag 1	Data	Tag 2	Data
000	00	0040	01	0050

Date
Page

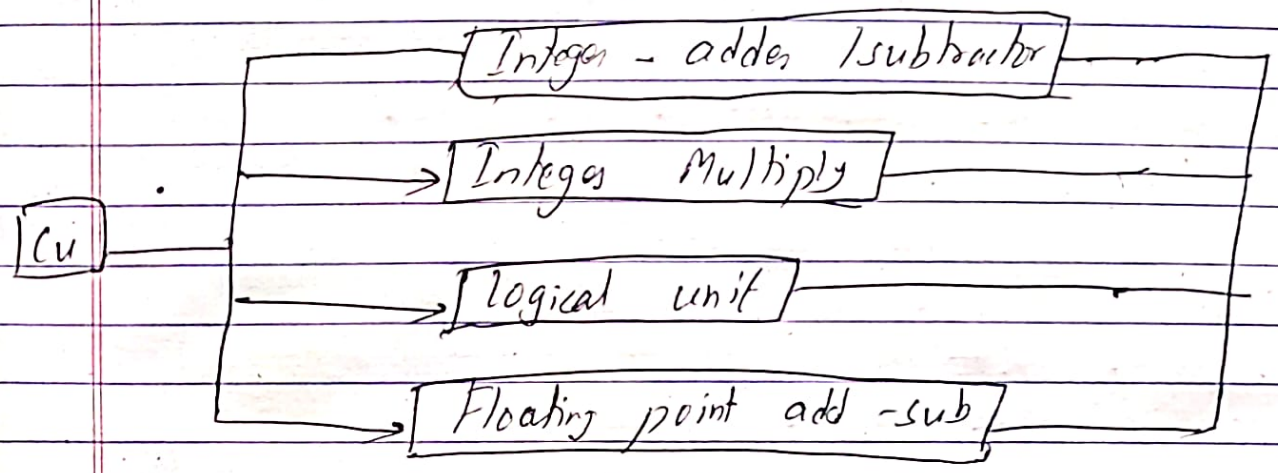
Cache Writing Policy

→ Write through policy
→ Write back policy

Scanned by CamScanner

Pipeline

Parallel processing



Flynn's classification

- SISD
- SIMD
- MISD
- MIMD

Scanned by CamScanner

Basic Comp Organ

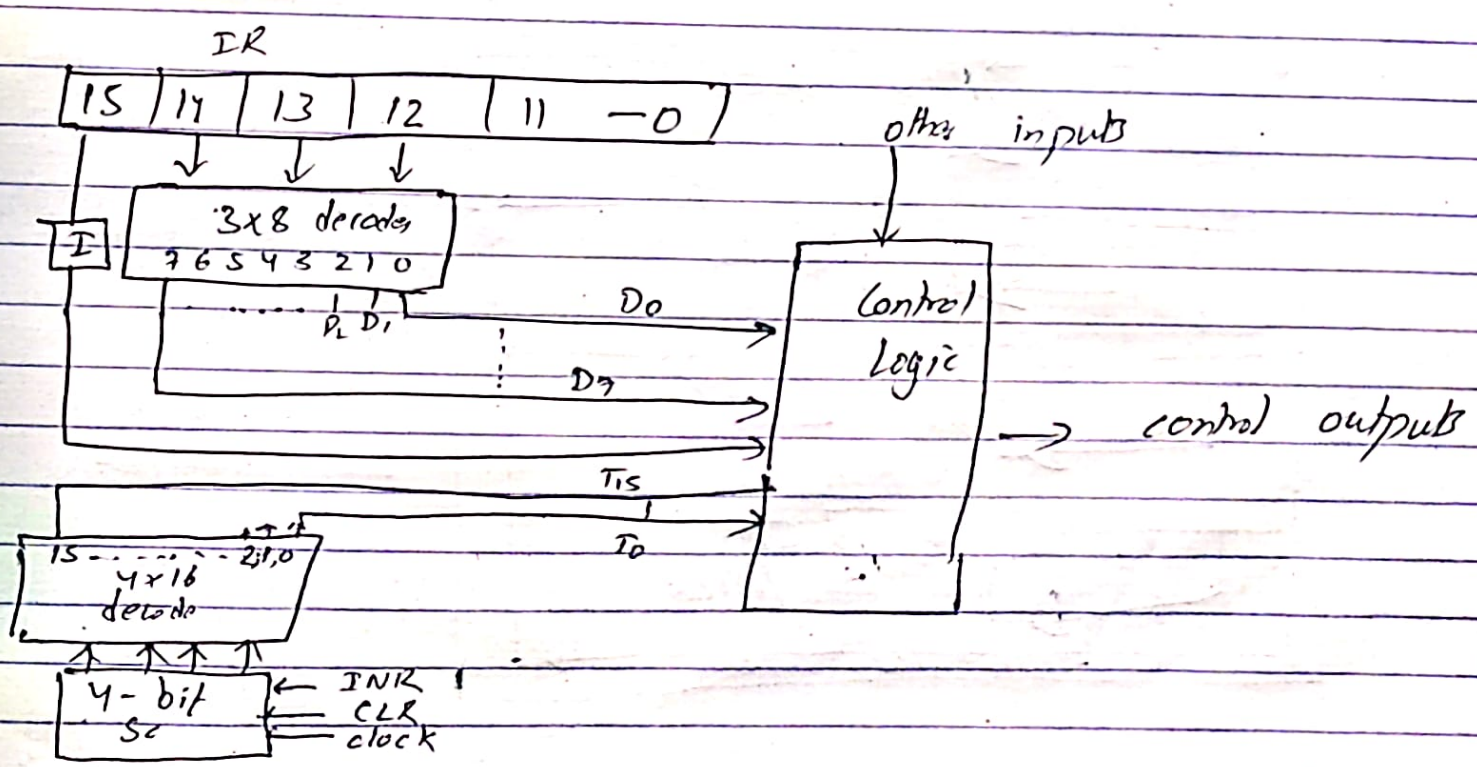
Control Unit

CU

Microprogrammed CU	Hardwired control unit
Control logic is implemented by means of microprogram (software)	→ Control logic is implement using hardware (flip-flops, logic gates etc)
Slow & cheap	→ Fast & expensive
Modification of logic is easy	→ Modification of logic is difficult
Used in general purpose computes	→ Used in microcontrollers

Hardwired control Unit

(How instruction is mapped into control signal? or Explain the block diagram of hardwired control unit?)



Scanned by CamScanner

Instruction cycle

In basic computer it consists of four phases

a) Fetch
b) Decode
c) Effective address calculation in case of indirect addressing mode
d) Execute

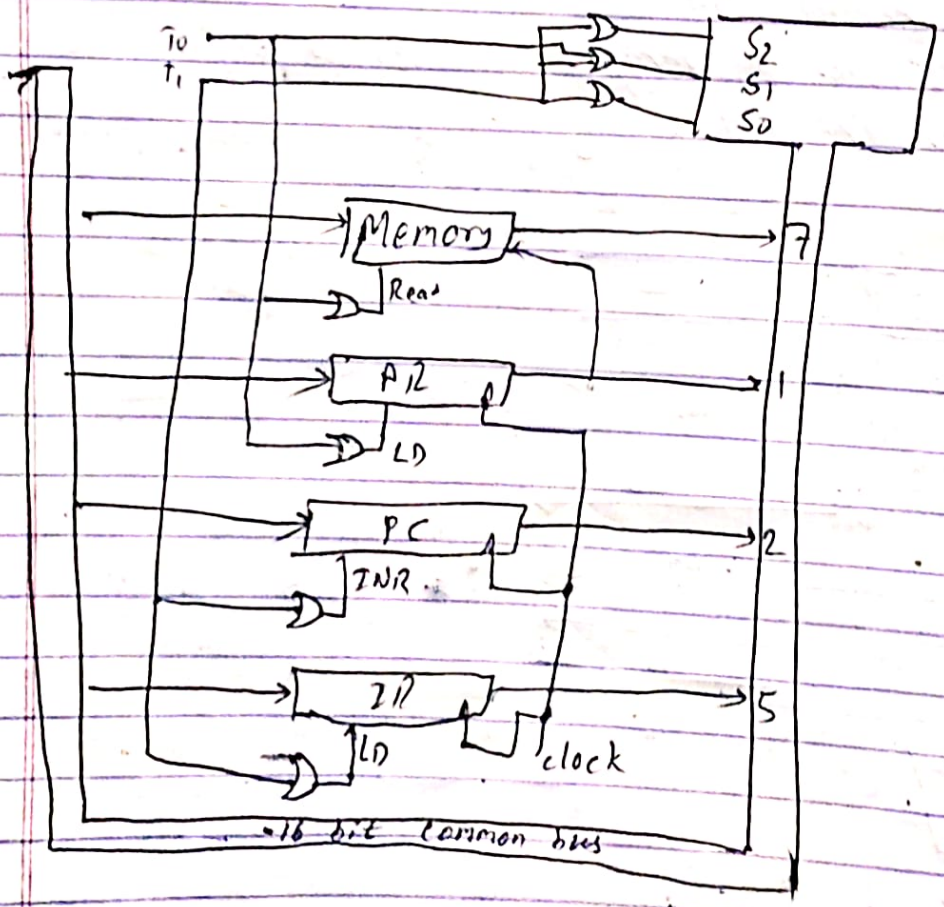
Fetch & Decode Phase

T_0 : $AR \leftarrow PC$

T_1 : $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$ } fetch

T_2 : $D_7 \ldots D_0 \leftarrow$ Decode $IR(12-14)$,
$AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

Registers transfers for fetch phase



Scanned by CamScanner

Determine the type of Instruction

Q.) How Instruction is ~~exp~~ executed in basic computer ? Explain with

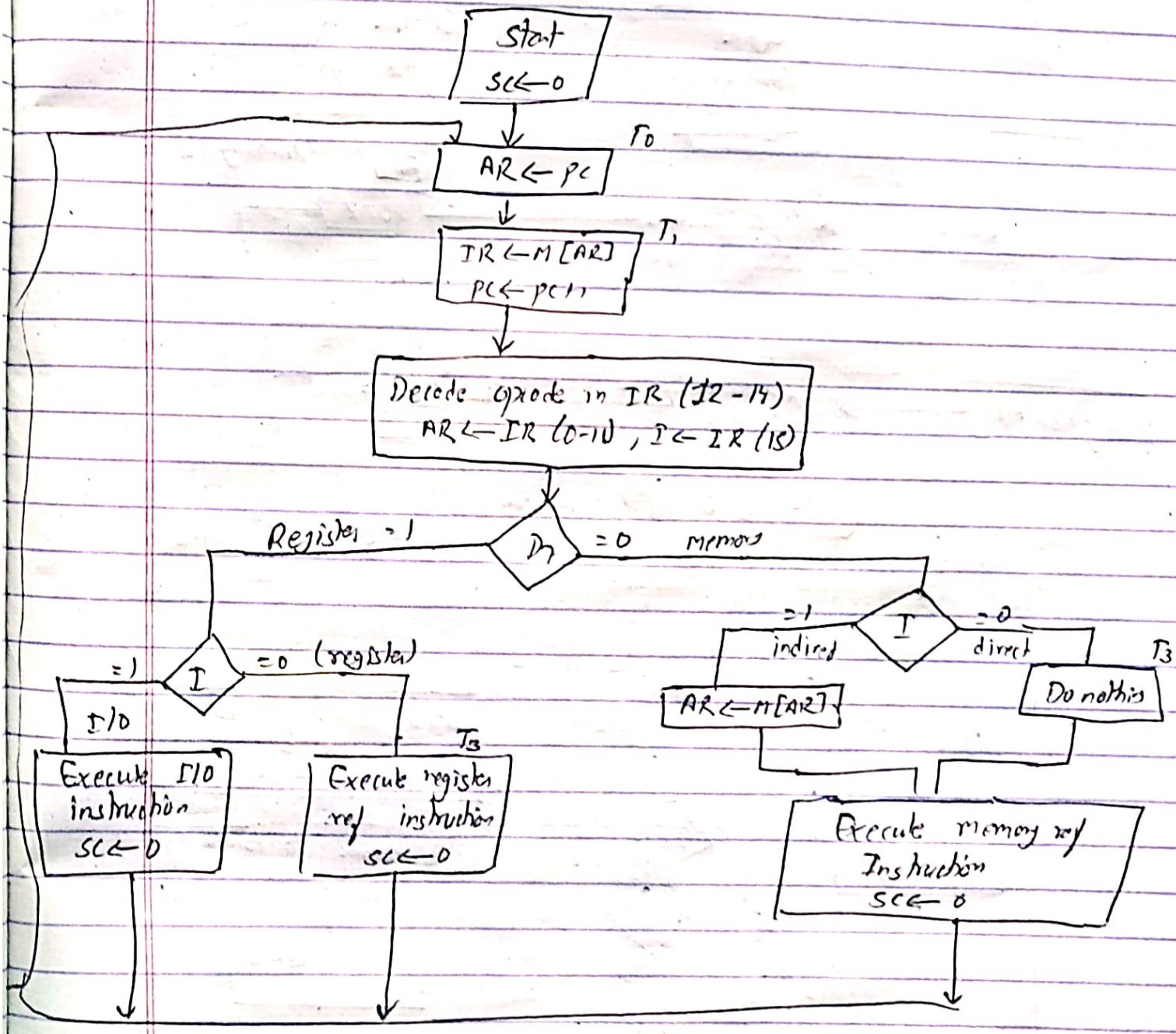


fig :- Instruction cycle

Scanned by CamScanner

Input - Output & Interrupt

Input - Output Configuration (Programmed I/O)

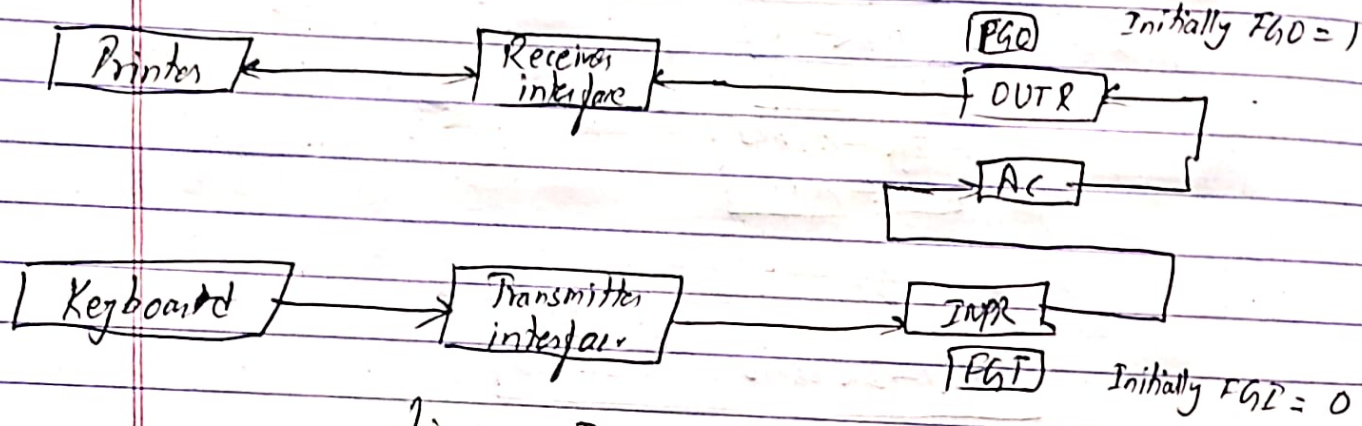
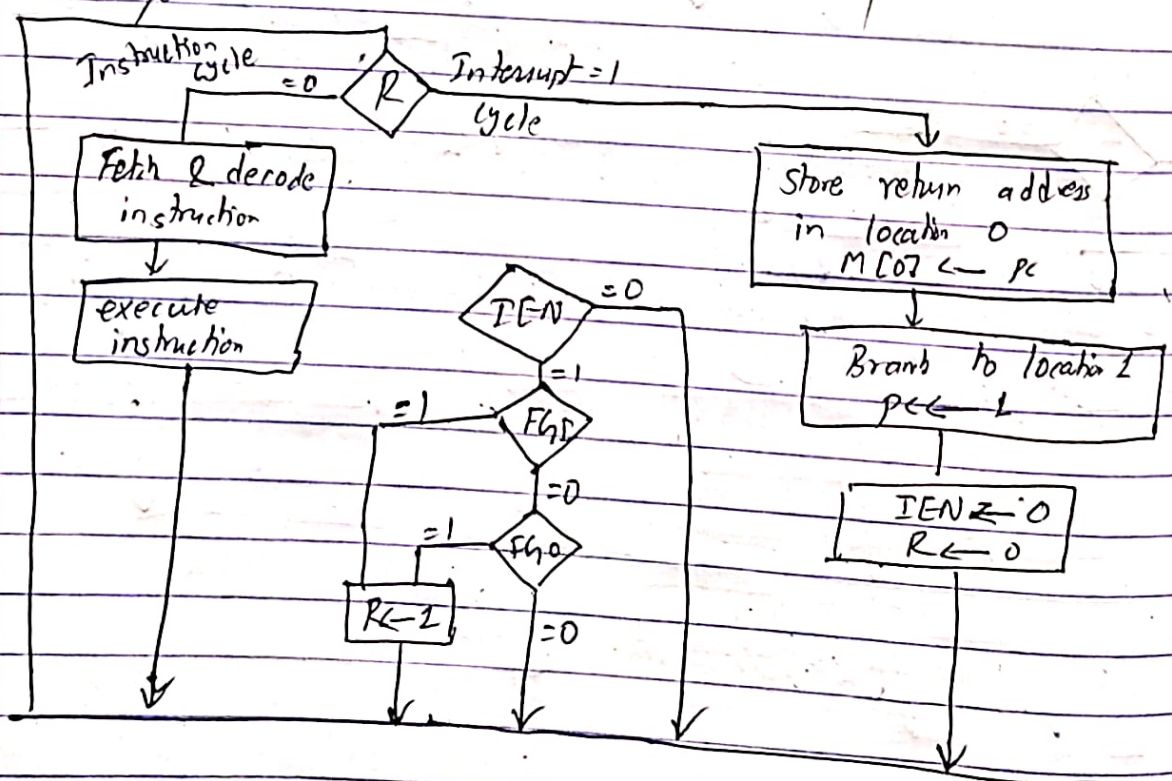


fig :- I-O configuration

Interrupt

$R \leftarrow T_0' T_1' T_2' IEN (FGI + FGO)$



Scanned by CamScanner

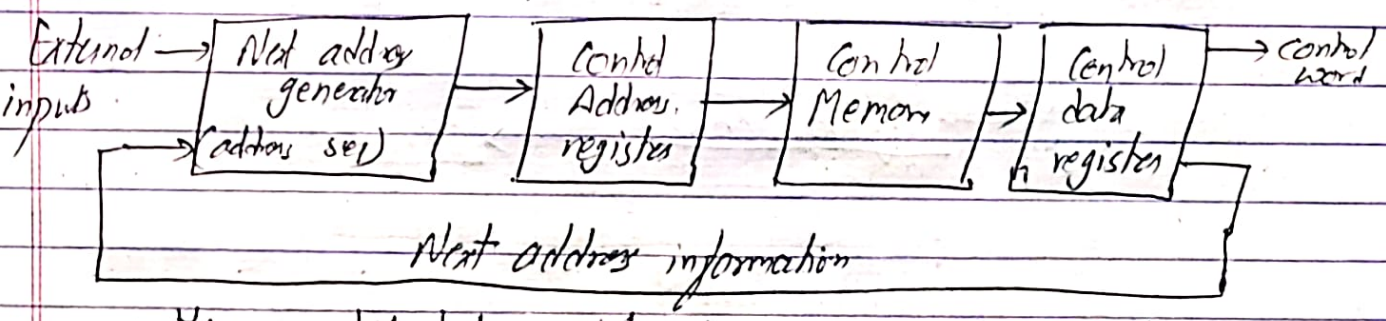
Unit 4

Microprogramed Control Unit

Terminology :-

*) Control Unit
*) Micro instruction
*) Micro program
*) Control Memory
*) Dynamic microprogramming

i) General organization of microprogram control unit.



Here, control data register is called pipeline register

*) Address sequences

→ Address sequencing capabilities required in a control memory are :-

i) Incrementing of the control address register
ii) Unconditional branch or conditional branch depending on status bit conditions
iii) A mapping process from the bits of the instruction to an address for controlling memory
iv) A facility for subroutine call & return.

Scanned by CamScanner

Address Sequencer :-

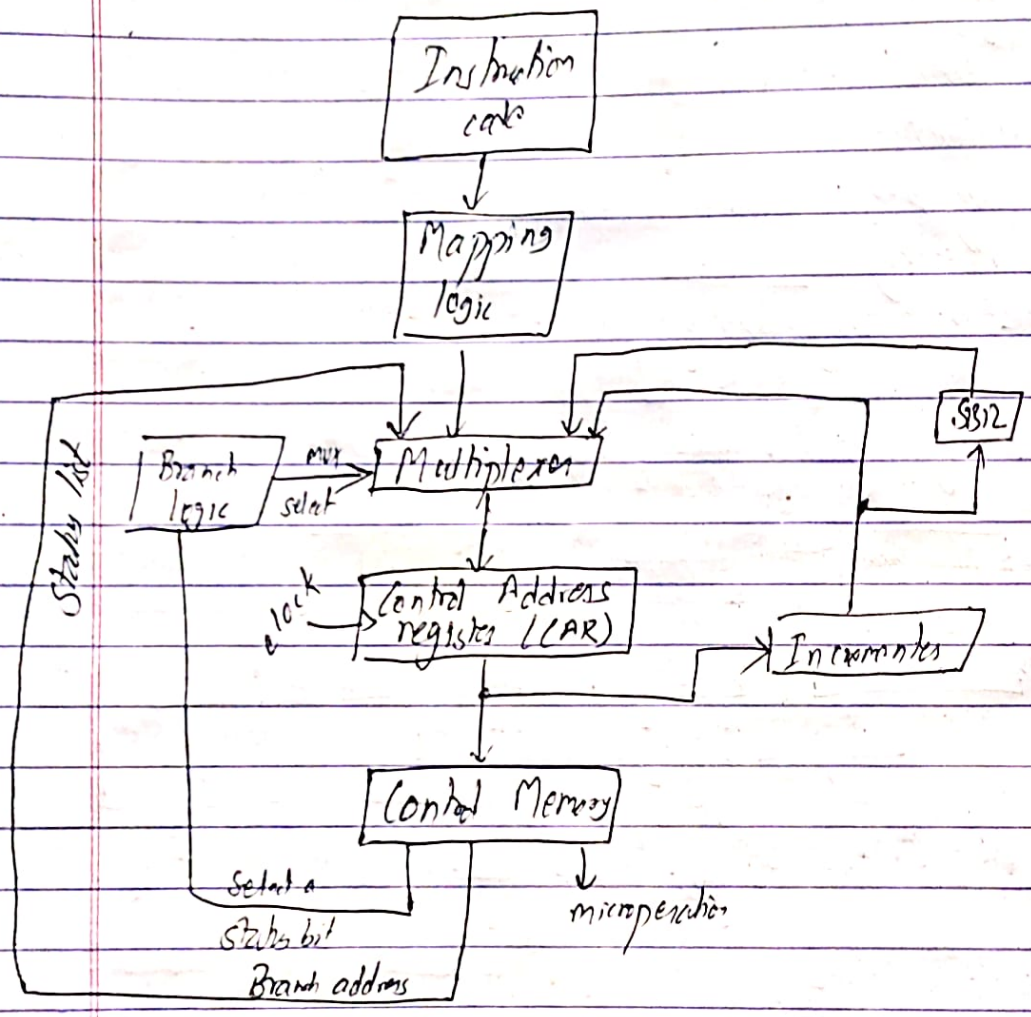


fig :- Selection of address for control memory

eg:- How instruction code is mapped into micro-instruction address :-

Computer instructions Mapping bits

Opcode

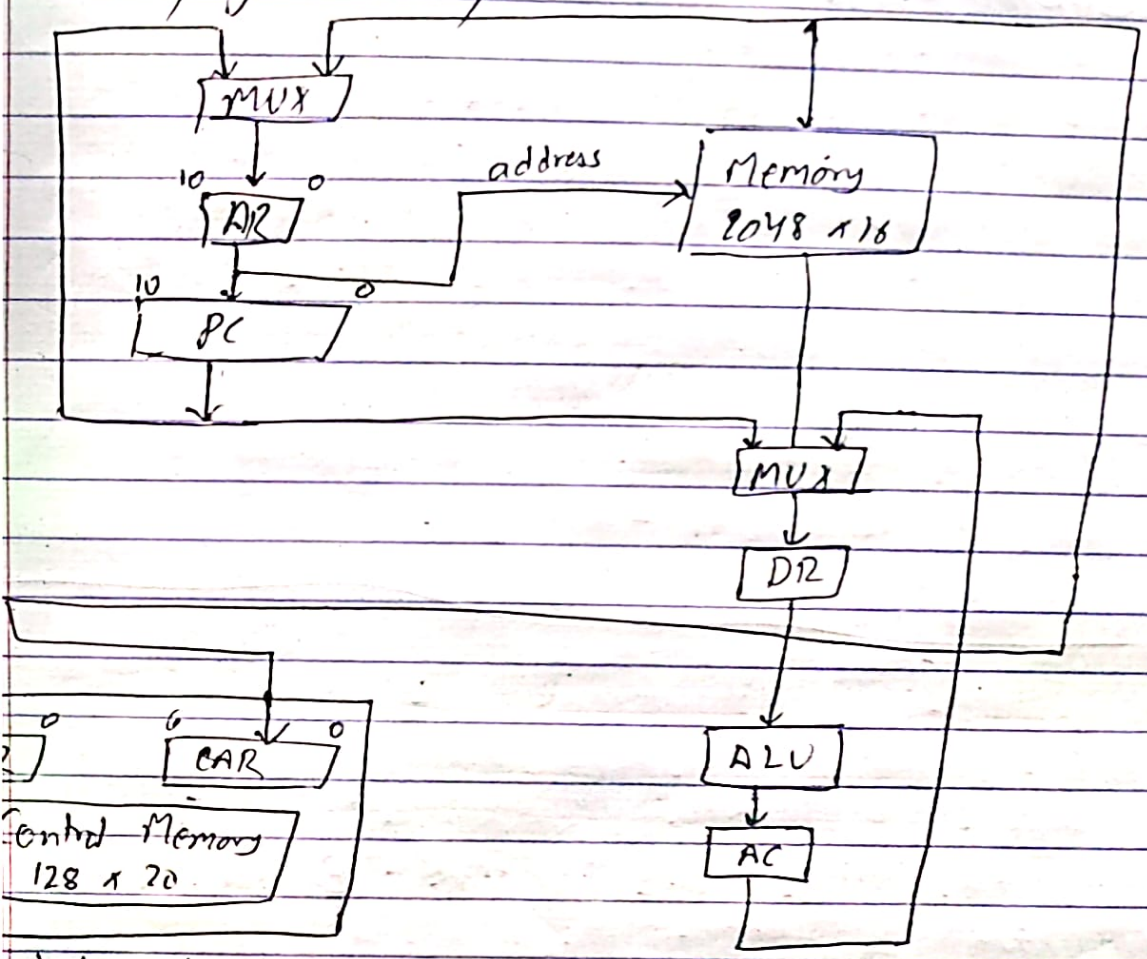
0110	address

0 xxxx , 00

Micro instruction address

0 0110 0 0 0

Microprogram example



ontrol unit

fig:- computer hardware configuration

Microinstruction format

F_1	F_2	F_3	CD	BR	AD
3	3	3	2	2	7

$F_1, F_2, F_3 \Rightarrow$ control word (micro-operation field)

CD : Condition for branching

BR : Branch field

AD : Address field.

fig :- micro-instruction code formed (20 bits)

Scanned by CamScanner

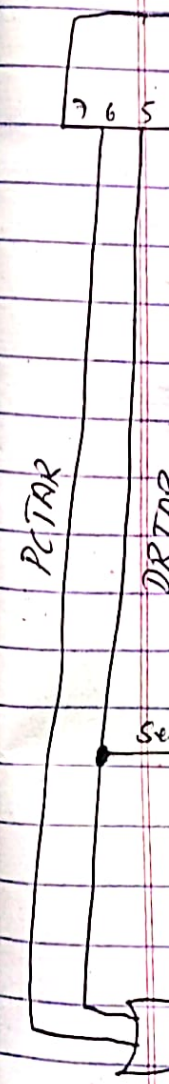
Symbols & Binary code for micro-instruction Fields

F_1	Microoperation
000	None
001	AC
010	AC
011	AC
100	AC
101	AR
110	AR
111	

Assignment:-

1.) How microoperation fields are decoded? Explain with block diagram
2.) Explain the microprogram sequencer with block diagram.
3.) Explain block diagram of address sequencer in details.

1.)



Scanned by CamScanner

How microperation fields are decoded? Explain with block diagram.

The number of control bits that initiate microperations can be reduced by grouping mutually exclusive variables into fields and encoding the k bits in each field to provide 2^k microperations. Each field requires a decoder to produce the corresponding control signals.

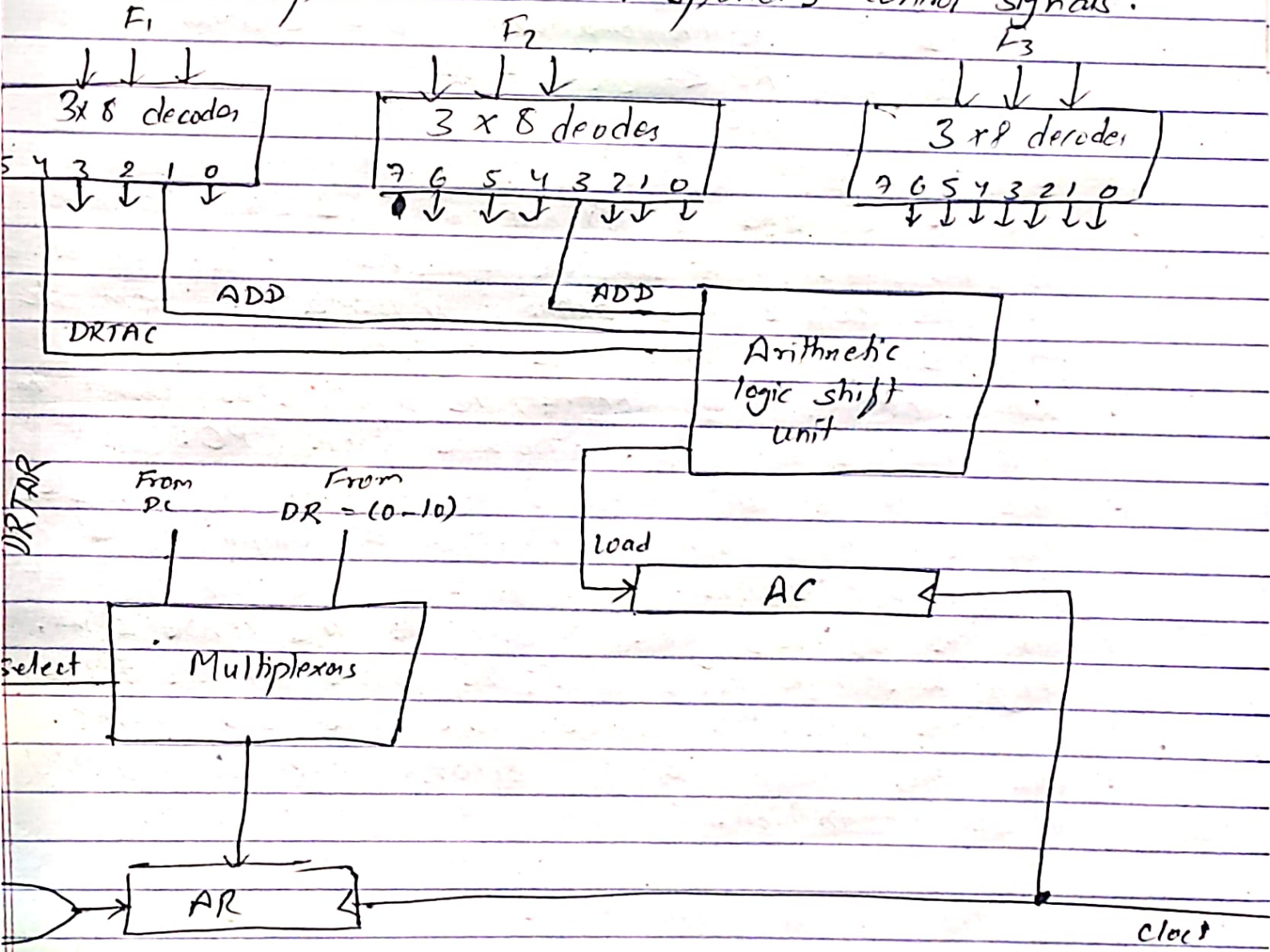


fig:- Decoding of microperation fields

Scanned by CamScanner

- Each of the three fields of the microinstruction presently available in the output of control memory are decoded with a 3x8 decoder to provide eight outputs.
- Each ~~must~~ of these output must be connected to the proper circuit to initiate the corresponding microoperation.
- For eg:-

F1	Microperation	Symbol
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR

- Outputs 5 and 6 of decoder F1 are connected to the load input of AR so that when either one of these outputs are active, information from the multiplexers is transferred to AR.
- When output 5 is active, the multiplexers select the information from DR. otherwise from PC.
- The transfer into AR occurs with a clock pulse transition only when output 5 or output 6 of the decoder are active.
- Instead of using gates to generate the control signals, they will now come from the outputs of the decoders associated with the symbols AND, ADD and DRTAC.
- The other outputs of the decoders must be connected in similar fashion.

Scanned by CamScanner

2. Explain the microprogram sequences with block diagram.

→ The basic component of a microprogramed control unit are the control memory and the circuits that select the next address. The address selection part is called a microprogram sequences.

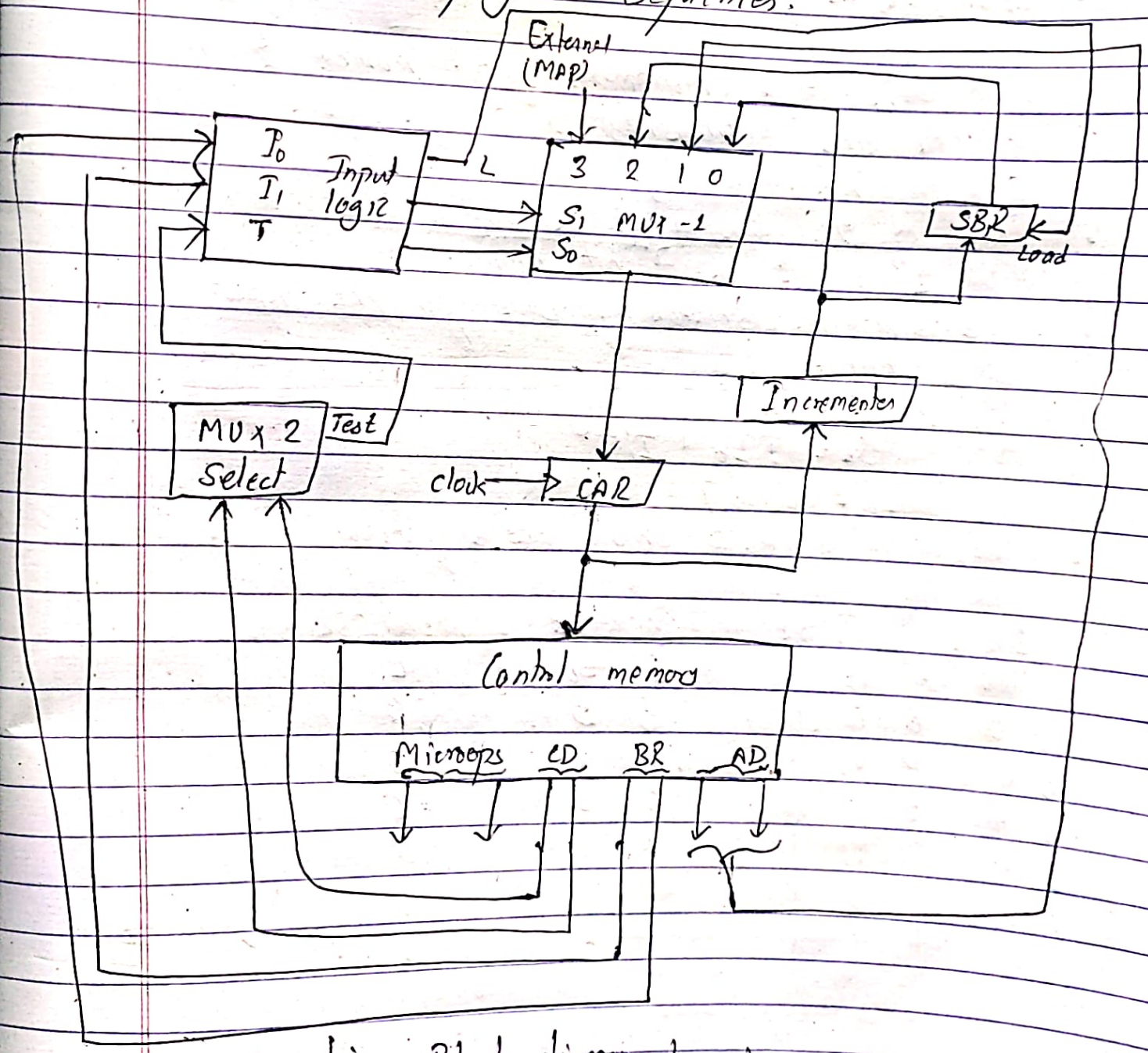


fig:- Block diagram of microprogram sequences.

Scanned by CamScanner

- The input logic has three inputs I_0, I_1 and T, and three outputs S_0, S_1 and L. ~~Variables~~
- Variables S_0 and S_1 select one of the source addresses for CAR. Variable L enables the load input in SBR.
- The binary values of the two selection variables determine the path in the multiplexer.

  For eg:-
  With $S_1, S_0 = 10$ multiplexer input number 2 is selected and establishes a transfer path from SBR to CAR.
- Each of the four inputs as well as the output of MUX 1 contains a 4-bit address.
- The simplified Boolean functions for the input logic circuit is

$$S_1 = I_1$$
$$S_0 = I_1 I_0 + I_1' T$$
$$L = I_1' I_0 T$$

- The truth table to obtain above function is

BR field	Input I_1	I_0	T	MUX 1 S_1	S_0	load SBR L
0 0	0	0	0	0	0	0
0 0	0	0	1	0	1	0
0 1	0	1	0	0	0	0
0 1	0	1	1	1	1	1
1 0	1	0	X	1	0	0
1 1	1	1	X	1	1	0

Scanned by CamScanner

cache ratio = $\frac{\text{cache hit}}{\text{total reference}}$

Cache miss ratio = 1 − cache hit ratio

= $\frac{\text{cache miss}}{\text{total reference}}$

NQ) The average memory access time of main memory is 1000 ns and the cache memory is 100 ns. The CPU finds reference word in cache 85 times out of 120. Find Average memory access time

Sol/

memory access time (a_m) = 1000 ns

cache " " (a_c) = 100 ns

" hit ratio (H_c) = $\frac{85}{120}$ = 0.708

" miss ratio (M_c) = $1 - \frac{85}{120}$ = 0.29

Average memory access time = $H_c * a_c + M_c (a_m + a_c)$

= $0.708 \times 100 + 0.29 (1000 + 100)$

= 389.8 ns

#) Find the cache hit ratio and cache miss ratio, if the CPU finds memory word 110 times out of 130.

Sol/

total reference = 130

cache hit = 110.

Then, cache hit ratio = $\frac{110}{130}$ = 0.8461

cache miss ratio = 1 − 0.8461 = 0.153846

The average memory access time of main memory is 200 ns and cache memory access time is 20 ns. The cache hit ratio is 0.9. Find average memory access time.

Sol/ a_m = 200 ns, a_c = 20 ns, H_c = 0.9, M_c = 1 − 0.9 = 0.1

Average mem access time = $H_c \times a_c + M_c (a_m + a_c)$

= $0.9 \times 20 + 0.1 (200 + 20)$

= 40 ns

Scanned by CamScanner

What is cache mapping ? Explain.

→ The transformation of data from main memory to cache memory is called cache mapping.

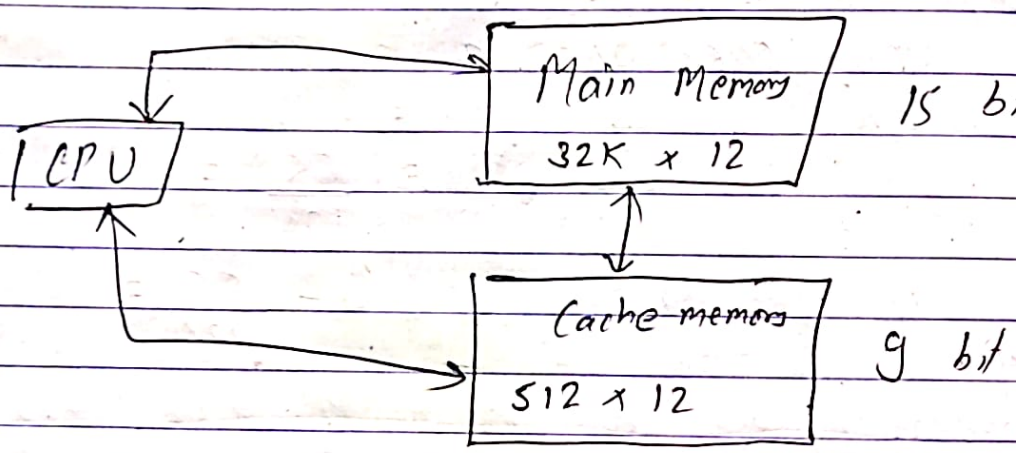
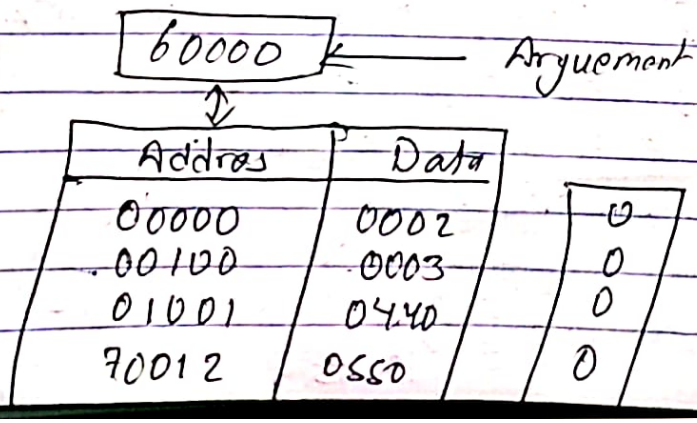


fig:- cache mapping

Types of cache mapping

i) Associate mapping

- The fastest and most flexible cache organization uses an associative memory.
- The associative memory stores both the address and content (data) of memory word.
- This permits any location in cache to store any word from main memory.



Scanned by CamScanner

ii) Direct mapping

- Associative memories are expensive compared to random-access memories because of the added logic associated with each cell.
- The possibility of using a random-access memory for the case is investigated as

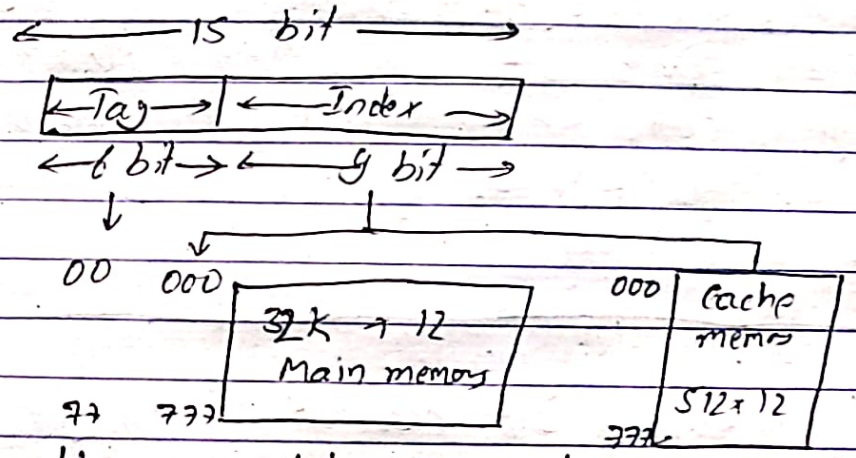
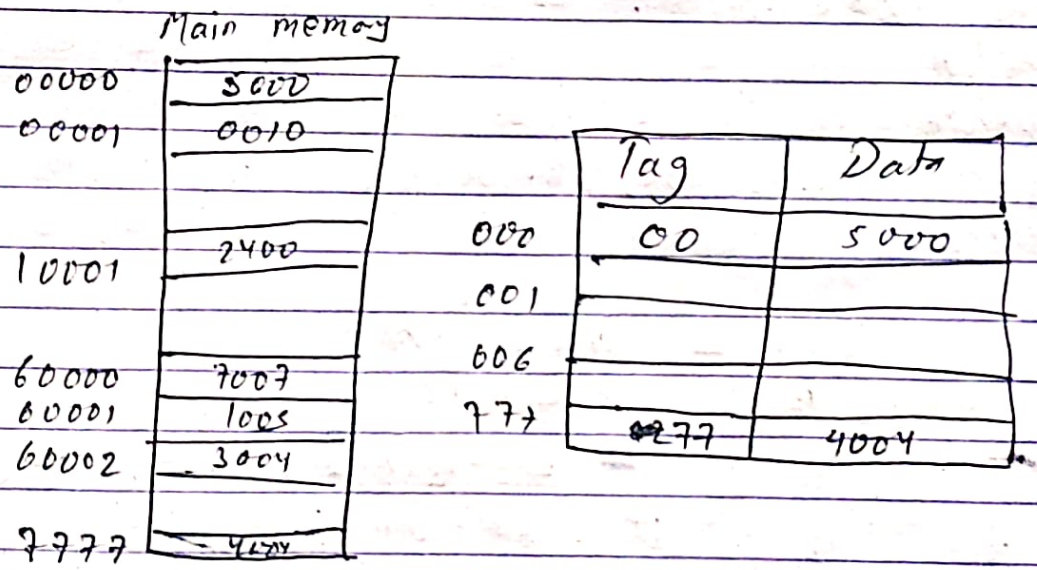


fig :- addressing relationship between main & cach memories

- The CPU address of 15 bit is divided into two fields.
- The nine bits constitute the index field and remaining 6 bits form the tag field.
- Direct mapping cache organization can be illustrated as



Scanned by CamScanner

iii) Set - Associative mapping

- A third type of cache organization, called set-associative mapping, is an improvement over the direct-mapping organization in that each word of cache can store two or more words of memory under the same index address

	Tag 1	Data	Tag 2	Data
000	00	0040	01	0050

fig:- two way set-associative mapping

How instruction is executed in basic computer? Explain with block diagram

→ In basic computer, instruction is executed as following four phase

i) Fetch
ii) Decode
iii) Effective address calculation in case of indirect addressing mode
iv) Execute

Fetch & Decode Phase

T_0 : $AR \leftarrow PC$

T_1 : $IR \leftarrow M[AR]$, $PC \leftarrow PC + 1$ } Fetch

T_2 : $D_7 \ldots D_0 \leftarrow$ Decode $IR(12-14)$;
$AR \leftarrow IR(0-11)$, $I \leftarrow IR(15)$

Scanned by CamScanner

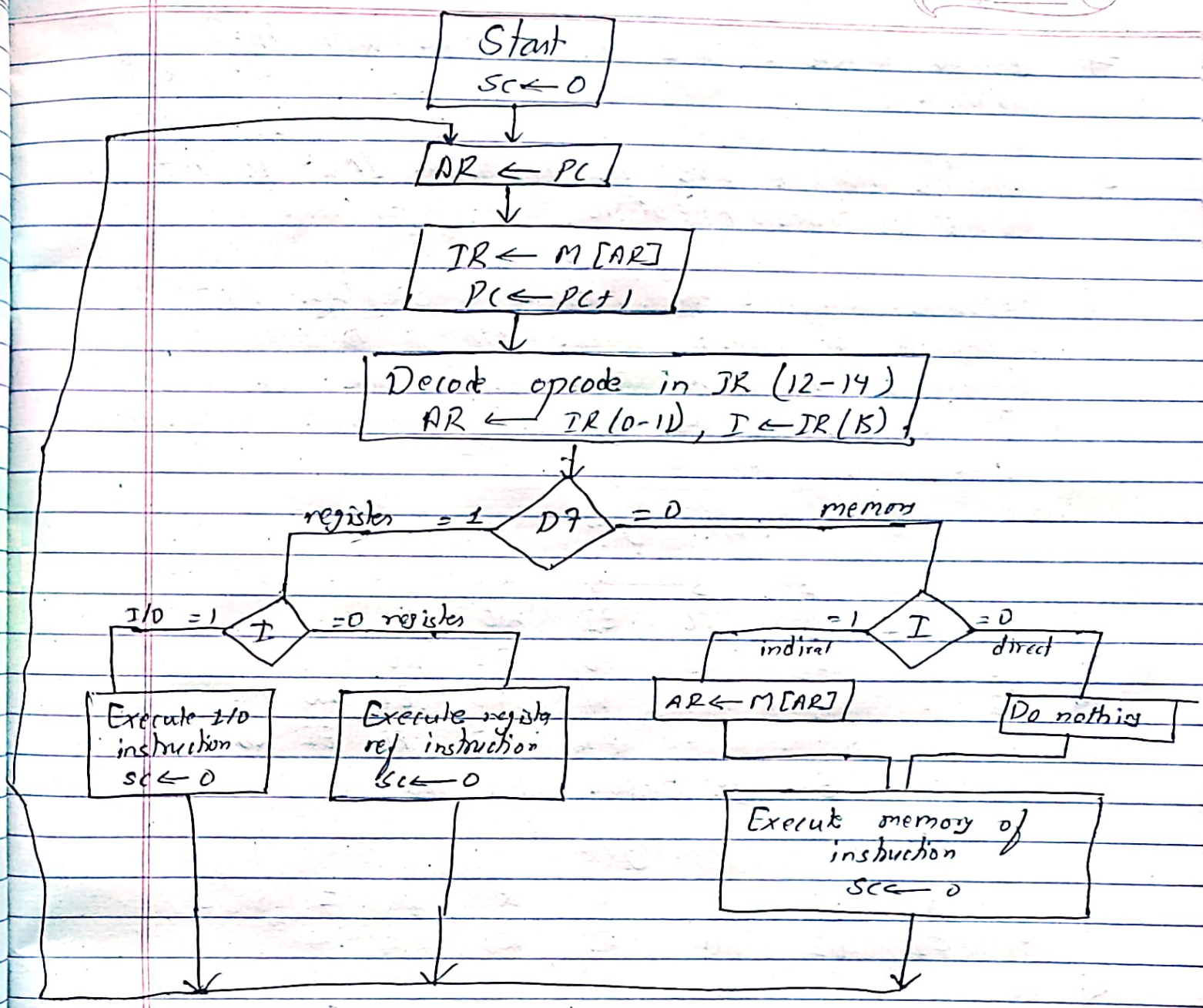


fig :- Instruction cycle.

Scanned by CamScanner

Describe various hazards in pipeline with their solution.

→ Pipeline hazards are the condition that do not permit continued operations.

i) Resource conflicts or hazards (Structural hazard)

- When two or more segment attempt to access the memory at the same time, a resource conflict occurs.
- It can be resolved by using seperate instruction and data memories

clock cycle

Instruction	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			FI	DI	FO	EI	WO		
I4				FI	DI	FO	EI	WO	

fig:- 5-stage pipeline, ideal case

clock cycle

Instruction	1	2	3	4	5	6	7	8	9
I1	FI	DI	FO	EI	WO				
I2		FI	DI	FO	EI	WO			
I3			idle	FI	DI	FO	EI	WO	
I4					FI	DI	FO	EI	WO

fig:- I1 souse operand in memory

figure :- Example of resource Hazard

Scanned by CamScanner

5.) Data Hazards

- The data hazard occurs when there is a conflict in the access of an operand location.
- There are three types of data hazards
  - → Read after write (RAW) or true dependency
    It occurs ~~wh~~ if the read takes place before the write operation is complete.
  - → Write after read (WAR) or anti-dependency
    It occurs if the write operation completes before the read operation takes place
  - → Write after write (WAW) or output dependency
    It occurs ~~wh~~ if the write operations take place in the reverse order of the intended sequence.

Clock cycle

	1	2	3	4	5	6	7	8	9	10
ADD EAX, EBX	FI	DI	FO	EI	WO					
SUB ECX, EAX		FI	DI	Idle		FO	EI	WO		
I3			FI			DI	FO	EI	WO	
I4						FI	DI	FO	EI	WO

Solution of data hazard

- → Hardware interlocks
- → Operand forwarding
- → Delayed load.

Scanned by CamScanner

iii) Branch hazards or control hazards

→ It occurs where the fetch stage cannot know which address the next instruction should be retrieved from. It typically occurs in case of conditional branch that has not yet been resolved.

Solutions for branch instructions

→ Pre-fetch target instruction
→ Branch target instruction
→ Loop buffer
→ Branch prediction
→ Delayed branch

NB)

time taken to process sub-operation = t_p
no. of segment = k
no. of tasks in sequence = n
To complete n task

i) Pipeline system
$(k+n-1)t_p$

ii) Non pipeline system
nkt_p where $t_n = kt_p$
→ nt_n

The speedup ratio = $\frac{nkt_p}{(k+n-1)t_p}$

Suppose, $t = 60ns$ $t = 70ns$, $\boxed{t = 100ns}$, $t = 80ns$
and delay of $t = 10$ ns

For pipeline, clock cycle $= t = 100 + 10$
non-pipeline $t = 60 + 70 + 100 + 80 + 10 = 320ns$

Scanned by CamScanner

A non-pipelined system takes 200 ns to process task. The same task can be processed in a five segment pipeline with a clock cycle of 40 ns. Determine speedup ~~that~~ ratio of the pipeline for 200 tasks. What is the maximum speed up that could be achieved with the pipeline unit over the non-pipelined unit?

Sol^n/

time req^d by non-pipeline system to complete a task $(t_n) = 200$ ns

no. of segment $(k) = 5$

Time req^d by each segment $(t_p) = 40$ ns

no. of task $(n) = 200$

Now

Speed of ratio/st $= \frac{n t_n}{(n-1+k) t_p} = \frac{200 \times 200}{(200-1+5) \times 40} = 4.9$

∴ Pipeline system is 4.9 times faster than non-pipeline system.

Now

Case 2 :- $n = 5$

$(s) = \frac{5 \times 200}{(5-1+5) 40} = 2.777$

Scanned by CamScanner

A non-pipelined system takes 100 ns to process task. The same task can be processed in a five-stage pipeline with a clock cycle of 20 ns. Determine speedup ratio of the pipeline for 100 tasks. What is the theoretical speed up that could be achieved with the pipeline unit over the non-pipelined unit?

Sol^n

Here, $t_n = 100$ ns

$k = 5$

$t_p = 20$ ns

$n = 100$

$$S = \frac{n t_n}{(n-1+k) t_p} = \frac{100 \times 100}{(100-1+5) \times 20} = 4.8076$$

Consider a pipeline system with 10 segment, if each segment requires 10 ns to process sub-operation then find the total time require to completely execute 500 task. Also determine the speed up ratio of pipeline system over equivalent non pipeline system.

Sol^n

Here, $k = 10$

$t_p = 10$ ns

$n = 500$

$t_n = k t_p = 10 \times 10 = 100$

Now

total time for pipeline $= (n-1+k) t_p$

$= (500-1+10) \times 10$

$= 5090$ ns

$$\text{speed of ratio} = \frac{n k t_n}{(n-1+k) t_p} = \frac{500 \times 100}{5090} = 9.823$$

Scanned by CamScanner

Suppose that a pipeline system with 6 segment each requires 10 ns, 30 ns, 40 ns, 5 ns, 15 ns and 50 ns respectively to process sub-operation. If the number of task is 100 then find the total time requires to execute all task by pipeline system. Also determine speed up ratio of pipeline system over equivalent non-pipeline system.

Sol

Here, $t_n = (10 + 30 + 40 + 5 + 15 + 50)\ ns = 150\ ns$

$t_p = 50\ ns$

$n = 100$

$k = 6$

For pipeline system

total time $= (n - 1 + k).t_p$

$= (100 - 1 + 6) \times 50$

$= 5250$

Then

speed up ratio $= \dfrac{n\, t_n}{(n-1+k)t_p} = \dfrac{100 \times 150}{5250} = 2.857$

Scanned by CamScanner

Unit 5

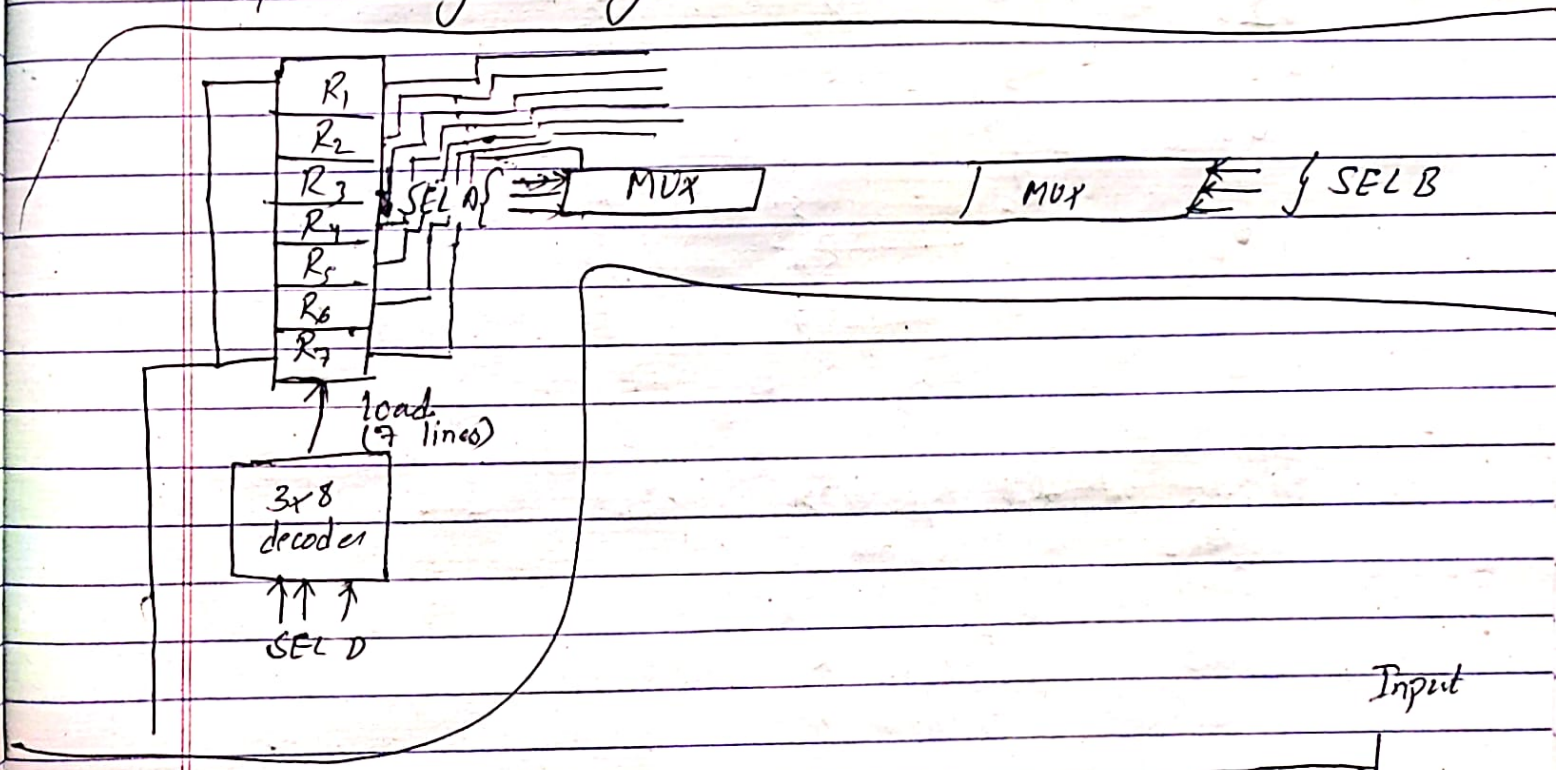
Central Processing Unit

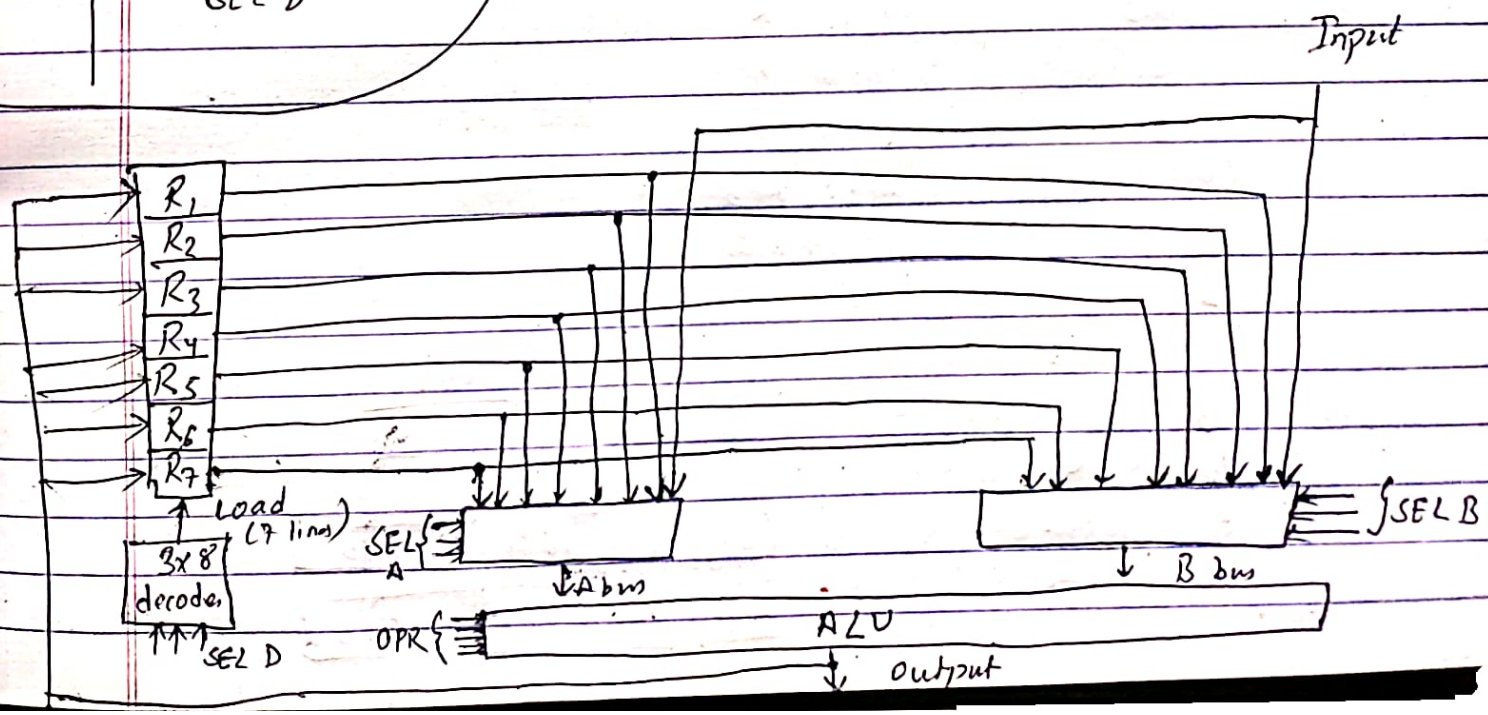
ADD R_1, R_2, R_3;
$R_1 \leftarrow R_2 + R_3$

CPU organization :-

a) Accumulator based (ADD x ; AC → AC + M[x])
b) General registers organization (ADD R_1, R_2 ; $R_1 \leftarrow R_1 + R_2$)
c) Stack organization (ADD, SUB)

General registers organization





Scanned by CamScanner

Control word

3	3	3	5
SEL A	SEL B	SEL D	OPR

Encoding of register selection fields

Binary code	SELA	SELB	SELD
000	Input	Input	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

Encoding of ALU operation

OPR Select	Operation
00000	Transfer A
00001	Increment A
00010	Add A+B
00101	Subtract A-B
00110	Decrement A
⋮	⋮

eg:- write control to execute the following micropersations

a) $R_1 \leftarrow R_2 + R_3$ | 010 | 011 | 001 | 00010 |

b) $R_1 \leftarrow R_1 + 1$ | 001 | 000 | 001 | 00001 |

c) Output $\leftarrow R_2 + R_3$ | 010 | 011 | 000 | 00010 |

d) $R_1 \leftarrow R_1 +$ input | 001 | 000 | 001 | 00010 |

Scanned by CamScanner

$$R_1 \leftarrow R_2 + R_3$$

00010

001 010 011

010	011	001	00010

Stack Organization (LIFO list)

→ Storage device that stores information in such a manner that the item stored last is the first item retrieved.

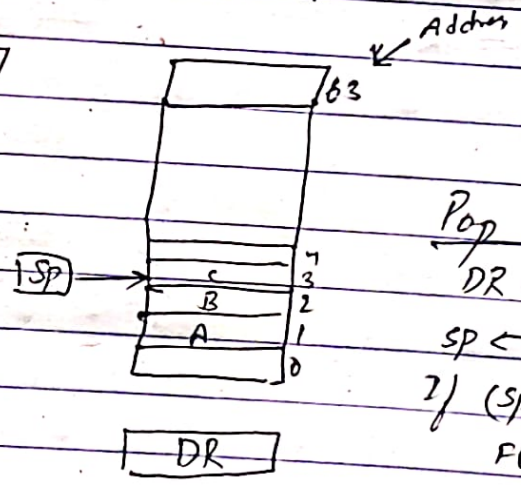
→ The register that holds the address for the stack is called a stack pointer (sp) because its value always points at the top in the stack.

→ Two operation ⓐ Push operation/insertion ⓑ Pop operation/deletion

→ Implementation :- a) Register stack
b) Memory stack

Register stack

FULL EMTY



Push operation :-

$sp \leftarrow sp + 1$

$M[sp] \leftarrow DR$

If $(sp = 0)$ then $(FULL \leftarrow 1)$

$EMTY \leftarrow 0$

Pop operation

$DR \leftarrow M[sp]$

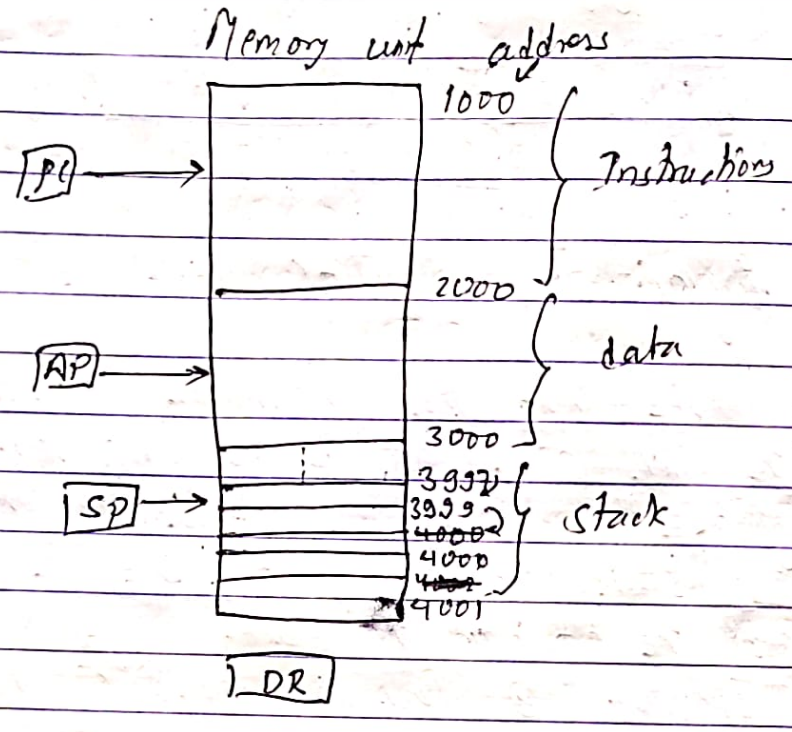
$sp \leftarrow sp - 1$

If $(sp = 0)$ then $(EMTY \leftarrow 1)$

$FULL \leftarrow 0$

Scanned by CamScanner

Memory stack



Push operation

$SP \leftarrow SP - 1$

$M[SP] \leftarrow DR$

Pop operation

$DR \leftarrow M[SP]$

$SP \leftarrow SP + 1$

Scanned by CamScanner

Instruction format

The bits of the instruction are divided into groups called Fields, the most common fields found in the instruction format are :-

a) Op-code field
b) Address field
c) Mode field

On the basis of number of address field found in instruction format, Instruction is categorized in to :-

a) One address instruction [ADD, SUB, DIV, MUL, LOAD, STORE]
   eg :- ADD X , $AC \leftarrow AC + M[X]$
b) Two address instruction [ADD, SUB, DIV, MUL, MOV]
   ADD R_1, R_2 ; $R_1 \leftarrow R_1 + R_2$
c) Three address instruction [ADD, SUB, DIV, MUL]
   ADD R_1, R_2, R_3 ; $R_1 \leftarrow R_2 + R_3$
d) Zero address instruction [PUSH, POP, ADD, SUB, MUL, DIV]
   eg :- ADD : $TOS \leftarrow TOS + TOS$

Scanned by CamScanner

VVI

Evaluate the following Arithmetic expr using zero, one, two & three address instruction :-

a) $y = (A+B) * (C-D)$

b) $y = \dfrac{(A+B) * (C-D)}{(A+D)}$

c) $y = A[(B+C)/(C-D)] \cancel{*(A+B)} + AB$

d) $y = (A+B) / (C-D)$

a) Sol^n

Zero address instruction

$y = AB^{+} CD^{-} *$

PUSH A	;	TOS ← A
PUSH B	;	TOS ← B
ADD	;	TOS ← A+B
PUSH C	;	TOS ← C
PUSH D	;	TOS ← D
SUB	;	TOS ← C−D
MUL	;	TOS ← (A+B) * (C−D)
POP y	;	y ← TOS

One address instruction

LOAD A	;	AC ← M[A]
ADD B	;	AC ← AC + M[B]
STORE T	;	M[T] ← AC
LOAD C	;	AC ← M[C]
SUB D	;	AC ← AC − M[D]
MUL T	;	AC ← AC * M[T]
STORE y	;	M[y] ← AC

Scanned by CamScanner

Two address instruction

```
MOV  R1, A   ;  R1 ← M[A]
ADD  R1, B   ;  R1 ← R1 + M[B]
MOV  R2, C   ;  R2 ← M[C]
SUB  R2, D   ;  R2 ← R2 - M[D]
MUL  R1, R2  ;  R1 ← R1 * R2
MOV  y, R1   ;  M[y] ← R1
```

Three address instruction

```
ADD  R1, A, B  ;  R1 ← M[A] + M[B]
SUB  R2, C, D  ;  R2 ← M[C] + M[D]
MUL  y, R1, R2 ;  y ← R1 * R2
```

d) Sol

Zero

$y = AB^{+}CD^{-}/$

```
PUSH A   ;  TOS ← A
PUSH B   ;  TOS ← B
ADD      ;  TOS ← A+B
PUSH C   ;  TOS ← C
PUSH D   ;  TOS ← D
SUB      ;  TOS ← C-D
DIV      ;  TOS ← (A+B)/(C-D)
POP y    ;  y ← TOS
```

Scanned by CamScanner

One

```
LOAD C      ;   AC ← M[C]
SUB ~~ADD~~ D   ;   AC ← AC − M[D]
STORE T     ;   M[T] ← AC
LOAD A      ;   AC ← M[A]
ADD B       ;   AC ← AC + M[B]
DIV T       ;   AC ← AC / M[T]
STORE y     ;   y ← AC
```

Two

```
MOV  R1, A  ;   R1 ← M[A]
ADD  R1, B  ;   R1 ← R1 + M[B]
MOV  R2, C  ;   R2 ← M[C]
SUB  R2, D  ;   R2 ← R2 − D
DIV  R1, R2 ;   R1 ← R1 / R2
MOV  y, R1  ;   y ← R1
```

Three

```
ADD  R1, A, B   ;   R1 ← M[A] + M[B]
SUB  R2, C, D   ;   R2 ← M[C] − M[D]
DIV  y, R1, R2  ;   y ← R1 / R2
```

Scanned by CamScanner

Addressing Mode

→ Specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

→ Purpose :-

- a) To give programming versality
- b) To reduce the number of bits in the addressing field of the instruction

Types :-

a) Implied mode :- operand are specified implicitly in the defination of the instruction eg: ADD X; AC←AC+M[X]

b) Immediate mode :- Operand is specified in the instruction itself

c) Register mode :- Operands are in registers that reside within the CPU

d) Register indirect : instruct specifies register in the CPU whose content give the address of operand in memory

e) Auto increment or Auto decrement mode :- Similar to the register indirect mode except that the register, is incremented or decremented after (or before) it's value is used to access memory

f) Direct address mode :- address part of instruction specifies EA

g) Indirect address mode :- address part of an instruction specifies the address where EA is found

h) Relative address mode :- EA = Address part of instruction + content of PC.

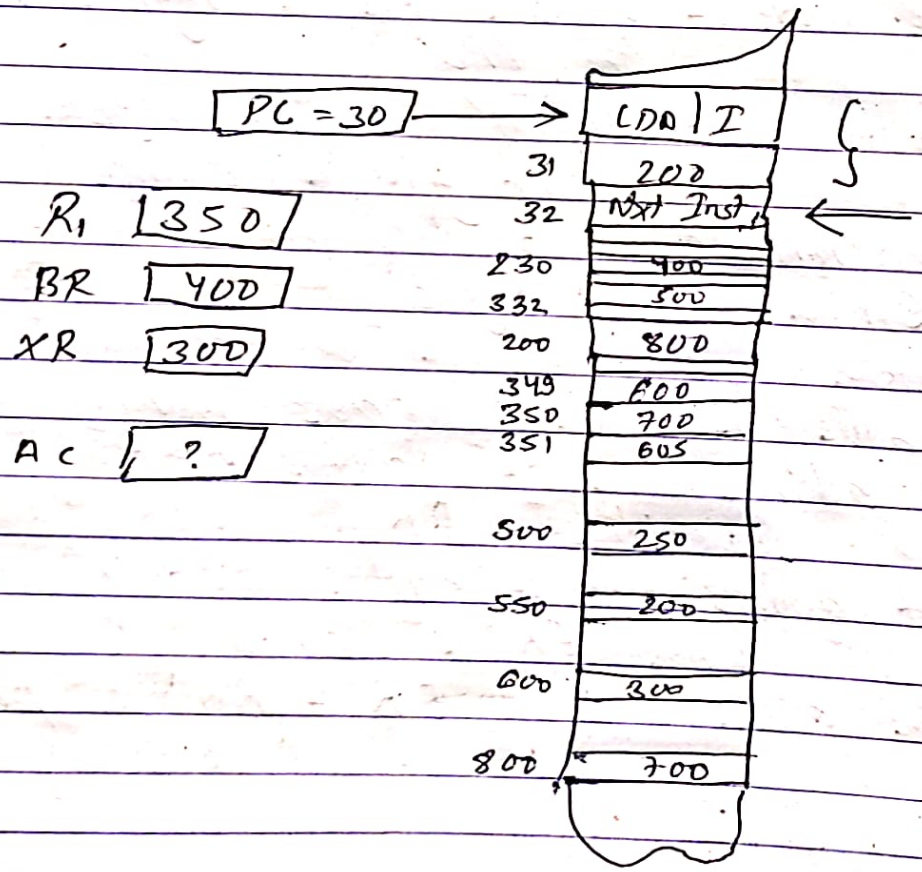
Scanned by CamScanner

i) Indexed addressing mode :-
EA = Address part of instruction + content of indexed register.

ii) Base register addressing mode :-
EA = Address part of instruction + content of Base register.

Numerical example

The data & instruction stored in a memory is shown in fig below. Determine what value will be loaded into AC when the instruction LDA 200 is executed & the addressing mode are :-



Scanned by CamScanner

a) Immediate

Operand = 200 , EA = 31

∴ AC = 200

b) Register direct

Operands = 350 , EA = — , AC = 350

c) Register indirect

EA = 350 , operand = 700 ∴ AC = 700

d) Auto increment

R_1 = 350 , EA = 350 , operand = 700

$\boxed{R_1 = 351}$

e) Auto decrement

R_1 = 350 , R_1 = 349 , EA = 349 , operand = 600

f) PC relative

EA = 200 + 32 = 232

operand = 800

g) Base register add

EA = 200 + 400 = 600

operand = 300

h) Indexed addessing mode

EA = 200 + 300 = 500

operand = 2050

i) Direct

EA = 200

operand = 800

j) Indirect

EA = 800

operand = 700

Scanned by CamScanner

Data transfer & Manipulation

Most computer instructions can be classified into three categories :-

a) Data transfer instructions :-
b) Data manipulation instructions
c) Program control instructions

Data transfer instructions :-

Move data from one place in the computer to another without changing the data content

Name	mnemonic
Load	LD
Store	ST
Move	MOV
Exchange	XCH.
Input	IN
Output	OUT
Push	PUSH
Pop	POP

Data Manipulation Instruction

Perform operations on data & provide the computational capabilities for the computer

a) Arithmetic instructions
b) Logical & bit ~~mang~~ manipulation instructions
c) Shift instructions

Scanned by CamScanner

Arithmetic instructions :-

Name	Mnemonic
Increment	INC
Decrement	DEC
Add	ADD
Subtract	SUB
Multiply	MUL
Divide	DIV
Add with carry	ADDC
Negate	NEG

Logical & Bit manipulation

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
X-OR	XOR
Clear carry	CLR C
Set carry	SET C
Complement carry	COMC
Enable interrupt	EI
Disable interrupt	DI

Shift instruction

Name instructions :-

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

Program Control

Name	Mnemonic
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare	CMP
Test	TST

Status Bit Conditions :- C, S, Z

Scanned by CamScanner

Conditional Branch Instructions :-

Mnemonic	Branch condition	Test condition
BZ	Branch if zero	Z=1
BNZ	" " not zero	Z=0
BC	" " carry	C=1
BNC	" " not carry	C=0
BP	" " plus	S=0
BM	" " minus	S=1

Program Interrupt

Types of interrupts

a) External interrupts
b) Internal interrupts (aka traps)
c) Software interrupts

RISC VS CISC

RISC

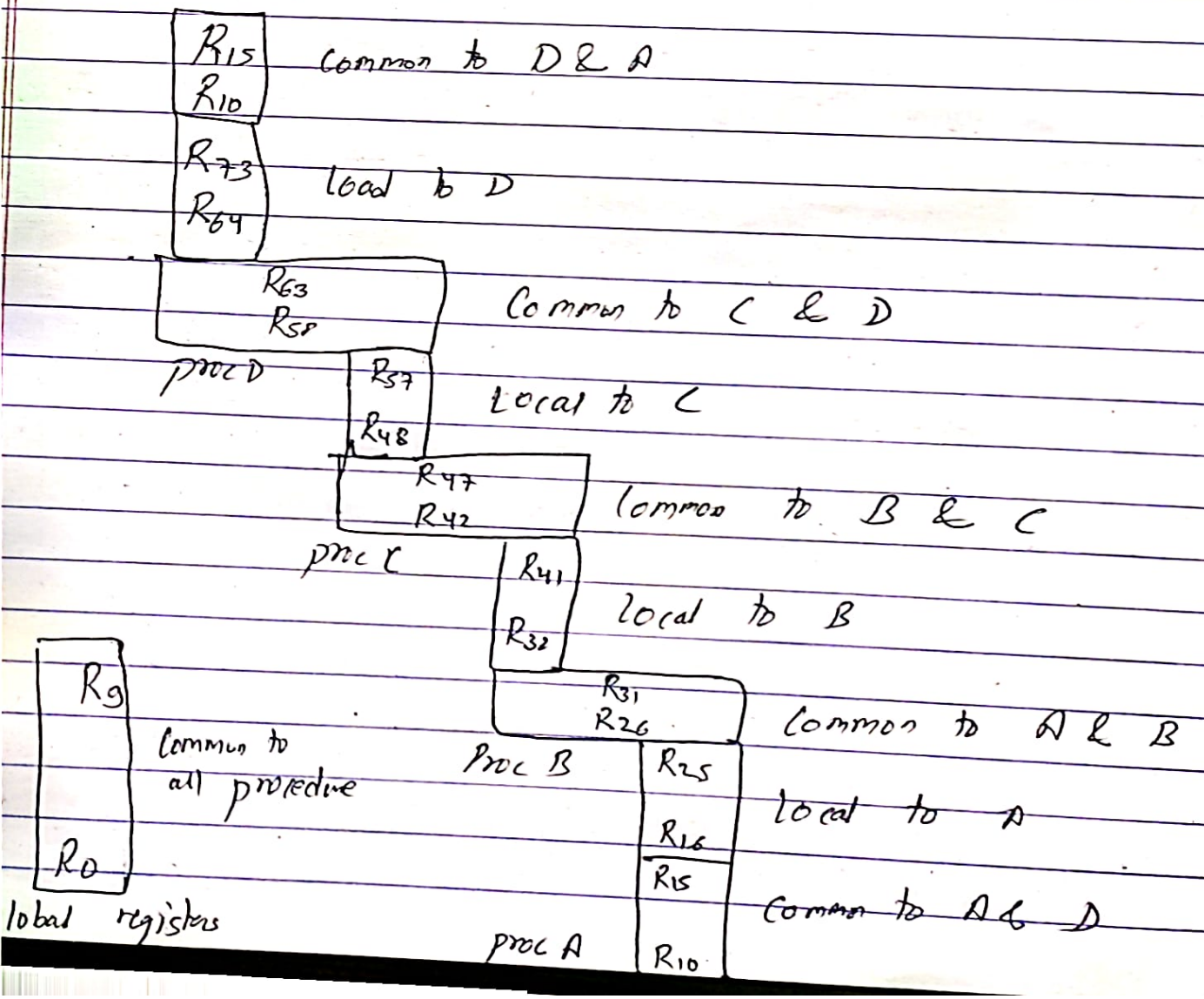
- → Relatively few instruction
- → Relatively few addressing mode
- → Memory access limited to load & store instructions.
- → All operations done within the registers of the CPU
- → Fixed-length, easily decoded instructions
- → single cycle instruction execution
- → Hardwired control unit

Scanned by CamScanner

CISC

- A large number of instructions - typically from 100 to 250 instructions
- Some instructions that perform specialized tasks & are used infrequently
- A large variety of addressing modes - typically from 5 to 20 different mode
- Variable - length instructions format
- Instructions that manipulate operands in memory

Overlapped Register Windows



Scanned by CamScanner