Unit 2: Arithmetic and logic Unit

Binary Arithmetic op^n on Unsigned and Signed Binary No.

(1) Addition op^n on Unsigned No.

Rules

(a) $0 + 0 = 0$

(b) $0 + 1 = 1$

(c) $1 + 0 = 1$

(d) $1 + 1 = 0$ carry → 1

(e) $1 + 1 + 1 = 1$ carry → 1

Eg:

13 + 12

13 : 1101

12 : 1100

```
  1101
+ 1100
-------
 11001
```

11001 ⇒ 25

Signed Number: There are three ways in which signed numbers can be represented in Binary form:

(a) Sign Magnitude

(b) 1^s complement

(c) 2^s complement

(a) Sign Magnitude: When a signed binary number is represented in signed magnitude, the left most bit is sign bit which tell you whether the number is positive or negative 0 - 0 is for (+)ve and 1 is for (-)ve and remaining bits are the

magnitude bits.

Eg: +12

⇒ [0] [1100]

Sign bit magnitude bit

+ ⇒ 0 (Sign bit)
- ⇒ 1 (Sign bit)

-12

⇒ [1] [1100]

Sign bit Magnitude bit

(b) 1's complement ° 1's complement of a binary number are important because they permit the representation of negative numbers. It is simply changing all 1s to 0s and all 0s to 1s.

Eg°

0	1	0	0	0	1	0	1	0	1
↓1's	↓1's	↓1's	↓1's	↓1's	↓1's	↓1's	↓1's	↓1's	↓1's
⇒ 1	0	1	1	1	0	1	0	1	0

(c) 2's complement ° The 2's complement of a binary Number is found by adding 1 to LSB of the 1's complement.

2's complement = 1's complement + 1

Eg° 1's complement ° 1 0 1 1 1 0 1 0 1 0

+ 1

2's complement ° 1 0 1 1 1 0 1 0 1 1

Ans.

Page No.
Date:

Addition and Subtraction on Signed Number

⇒ There are four case to perform on subtraction as well addition.

Case 1: (A - B)

↓

is same as

(+A) + (-B)

Case 2: (+A) + (+B)

Case 3: (-A) + (+B)

Case 4: (-A) + (-B)

Eg: Case 1: (A - B) ⇒ (+A) + (-B)

• 12 - 8

Step 1: Consider it to this form

(+12) + (-8)

Step 2: Write the Binary NO.

+12 = 0 1100

sign bit ↳ Magnitude

Step 3: (-8) So write the binary of (+8)

+8 = 0 1000

sign magnitude.

Change it to 1's complement

= 1 0 1 1 1

Now Add 1 to it i.e. calculate it's 2's complement

1 0 1 1 1

+ 1

1 0 1 1 1 1 0 0 0

Sign bit → magnitude

Page No.
Date:

ଏଥିରେ, $-8 = 11000$

Step 4 : Add both binary numbers

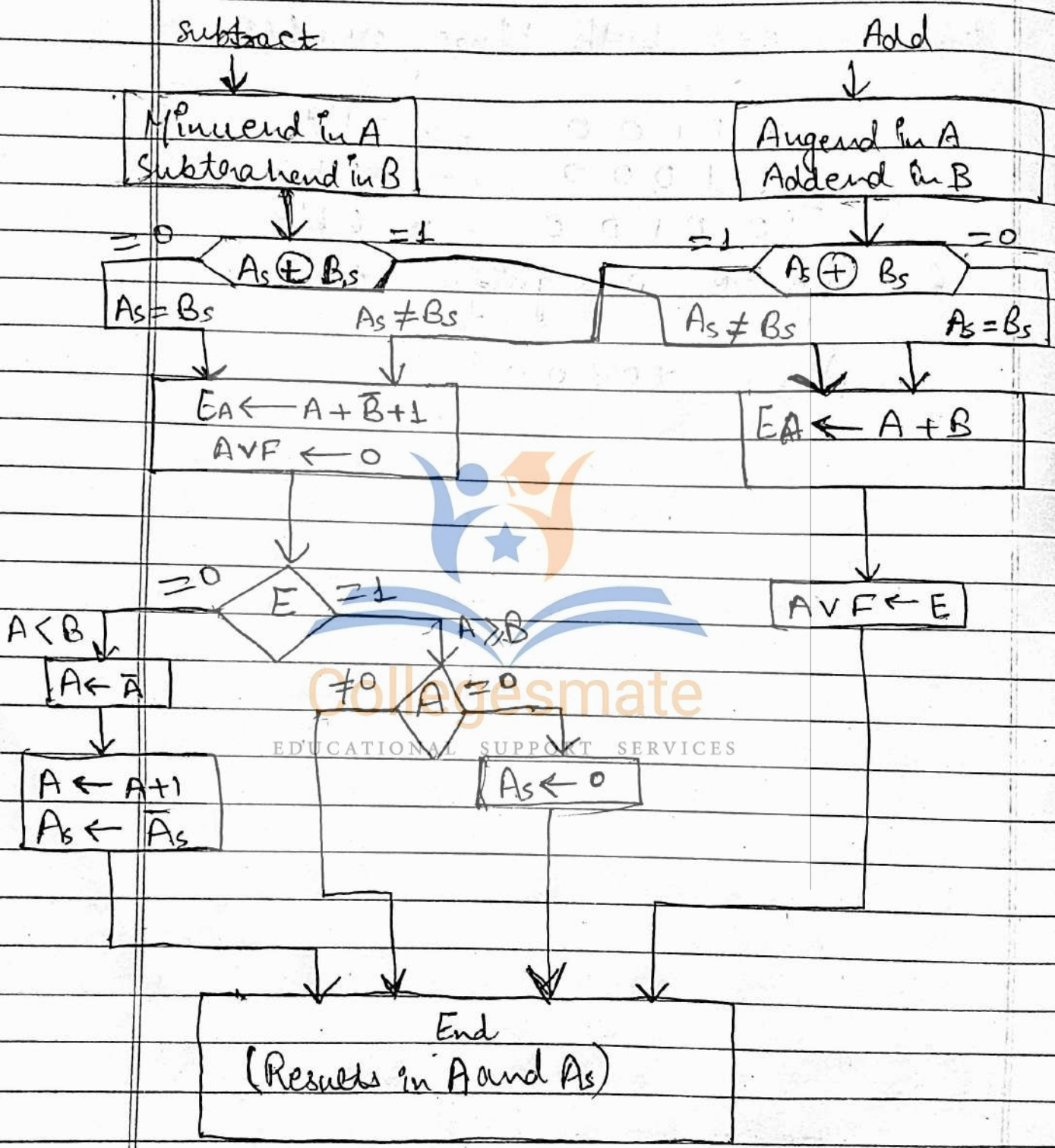
$$\begin{array}{rl} 0\ 1\ 1\ 0\ 0 & \rightarrow 5 \text{ bit} \\ +\ 1\ 1\ 0\ 0\ 0 & \rightarrow 5 \text{ bit} \\ \hline 1\ 0\ 0\ 1\ 0\ 0 & \rightarrow 6 \text{ bit} \end{array}$$

(So ignore insignificant bits)

Ans : 00100

Page No.
Date:

Flowchart for addition and subtract operation



Page No.
Date:

Eg: Perform the of" 6 - 7

Step 1: Write the Binary of Both number

6 → 1 1 0

7 → 1 1 1

Step 2: Write the sign bit.

$A_s \leftarrow 0$

$B_s \leftarrow 0$

Step 3: Perform the subtraction using flow chart

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

So $A_s = B_s$

$E_A \leftarrow A + \bar{B} + 1$

Step 4:

$A \leftarrow 0110$

$B \leftarrow 0111$

$\bar{B} \leftarrow 1000$

$+1$

$\bar{B}+1 \rightarrow 1001$

$A: 0110$

$\bar{B}+1: +1001$

1111

Step 5: $E_A \leftarrow 1111$ No carry

Step 6: $E = 0 \Rightarrow A \leftarrow \bar{A}$

Step 7: 1's Complement of A value

$A \leftarrow 1111$

$A \leftarrow 0000$

Step 8°: add 1 to A

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

Step 9°: Complement of sign bit (A)

$A \leftarrow \bar{A_s}$

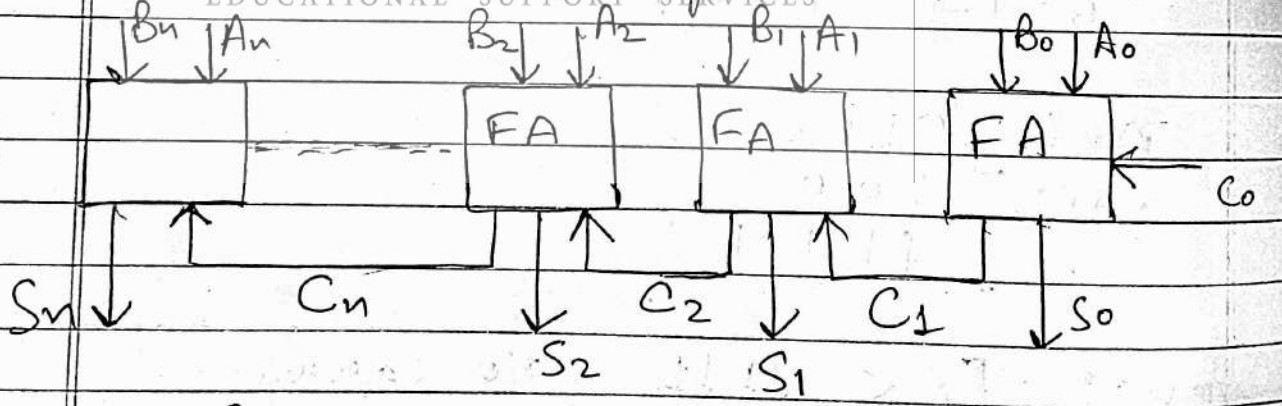
$1 \leftarrow 0$

(1) 0 0 0 1

sign magnitude

Ans: 1 0 0 0 1

Look ahead Carry Adder



n bit adder repeat carry adder

Page No.
Date:

Unit 1 Questions

Ques. A machine has 24 bit instruction format. It has 32 registers and each of which is 32 bit long. It needs to support 49 instructions. Each instruction has 2 register operands and one immediate operand. If the immediate operand is signed (means -ve.) integer, the minimum value of immediate operand is

A) -64 (B) -128 (C) -256 (D) -32

=> | opcode | operand |

Add R_1 R_2 50

24 Bit

opcode	Reg1	Reg2	Immediate operand
6	5	5	8

$24-(6+5+5) = 8$

$2^5 = 32$

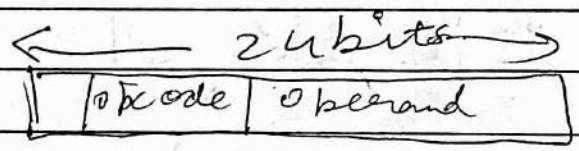
$2^6 = 64$

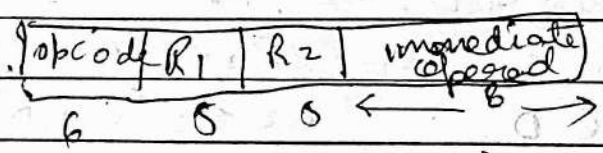
0 — 255

256

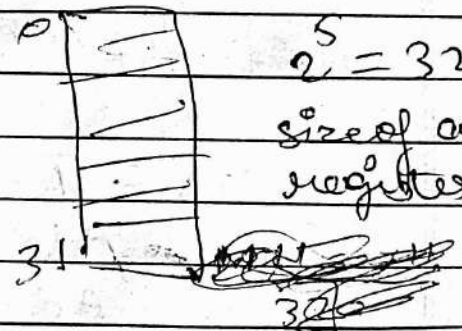
(-128 to 127)

Ans. -128





$24-(6+5+5) = 24-16 = 8$



$2^5 = 32$

size of address registers = 5

$2^6 = 64$

Ans. -128

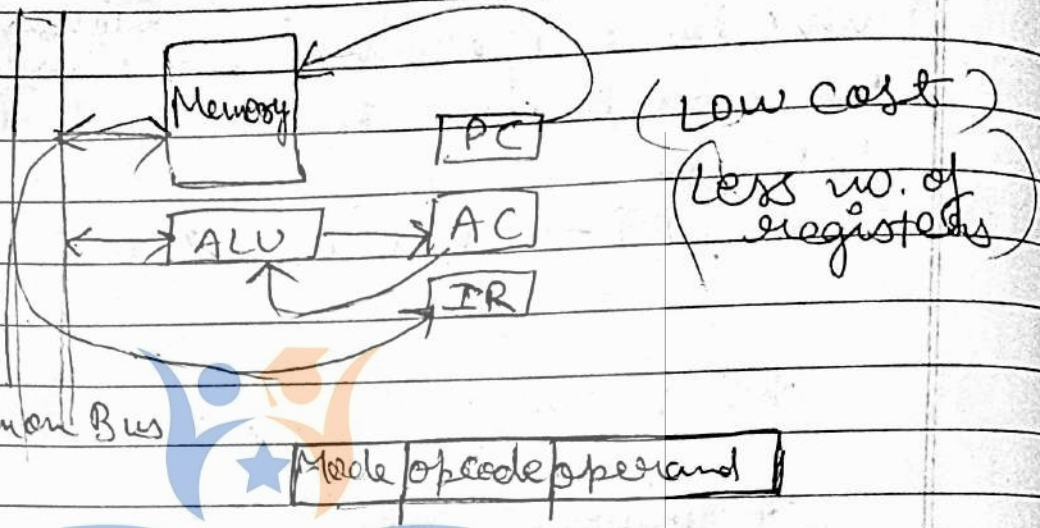
$2^8 = 256$

-128 to 127

Types of CPU Organisation

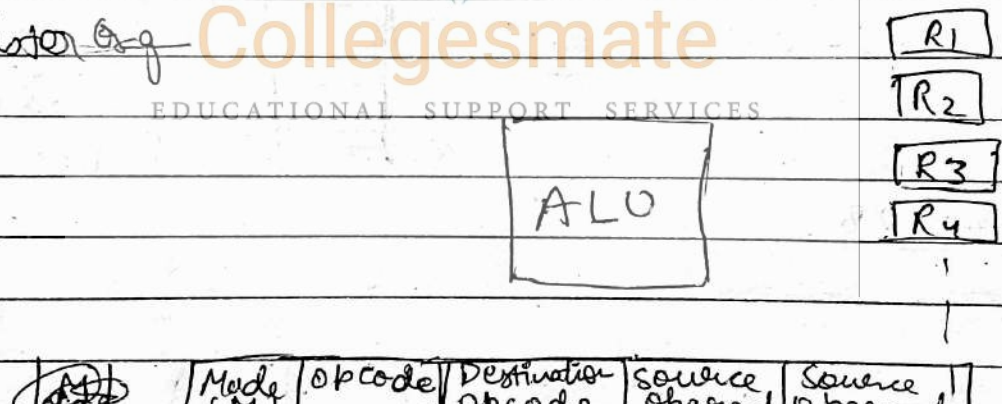
1) Single Accumulator organisation — Single Address instruction
2) General Register Organisation — 2 address / 3 address
3) Stack Organisation

1) Single Accu Org



(Low cost)
(less no. of registers)

2) General Register Org



$X = (A+B) * (C+D)$

~~Add R~~

Add R_1, A, B — $R_1 \leftarrow (A)+(B)$

Add R_2, C, D — $R_2 \leftarrow (C+D)$

MUL X, R_1, R_2 — $M[X] \leftarrow R_1 * R_2$

L:1.17 L:1.18

Page No.
Date:

3) Register Stack Organisation (LIFO)

flip flop → FULL Empty

Address

63

PUSH

SP ← SP+1

M[SP] ← DR

If (SP=0) then (Full ← 1)

Empty ← 0

64 word Reg.

3
2
1
SP → 0

POP

DR ← M[SP]

SP = SP - 1

If (SP=0) then (Empty ← 1)

Full ← 0

Data Register (a)

Addressing Modes

1) Implied mode
2) Immediate Mode
3) Register Mode
4) Register Indirect Mode
5) Auto increment Mode
   Auto decrement Mode
6) Direct Addressing Mode
7) Indirect addressing mode
8) Relative address Mode
9) Indexed Addressing mode
10) Base Register addressing mode

Implied mode / Implicit mode

↳ operand is specified implicitly in the definition of instruction.

↳ Used for zero address and one address instructions

Eg: * INC A
    * CRC
    * CL A
    * Add

opcode	operand

↳ Reduce size of instruction

Immediate

↳ operand is directly provided as constant.

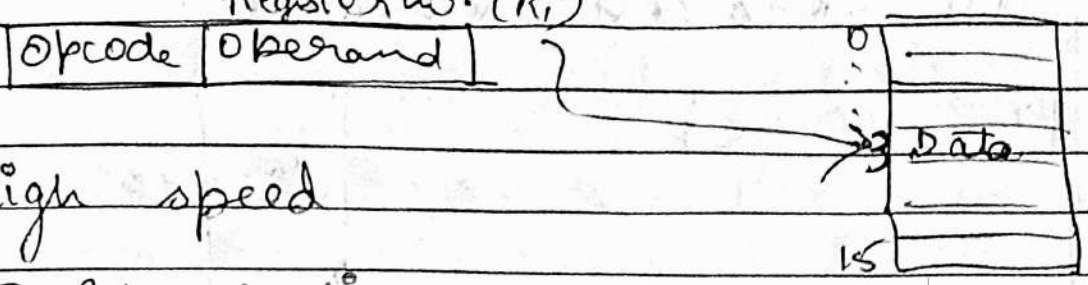
↳ No computation required to calculate effective Address.

opcode	operand

Add R1, #3

Register mode

↳ Operand is present in the register
↳ Register no. is written in instruction.



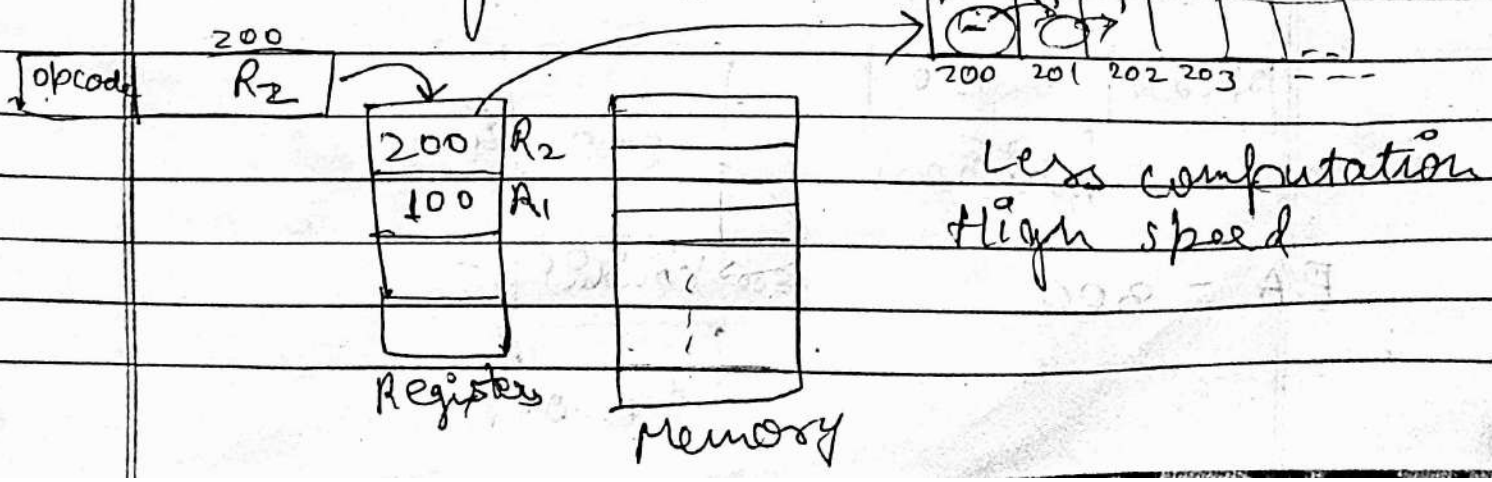
↳ High speed

Register Indirect Mode

↳ Register contains address of operand rather than operand itself.

Auto increment or Auto decrement

↳ Special case of Register indirect addressing mode.



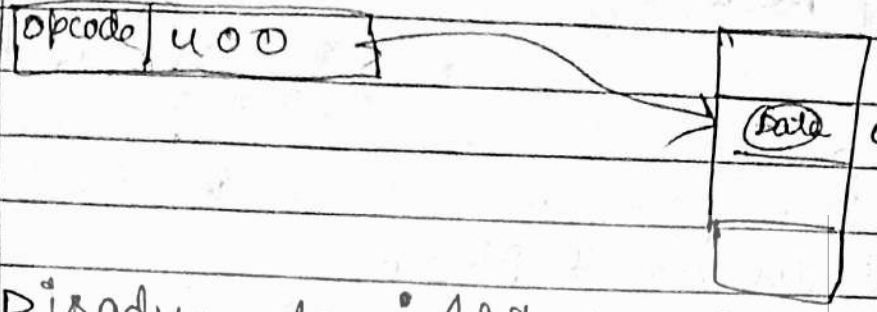
Less computation
High speed

Page No.
Date:

Imp

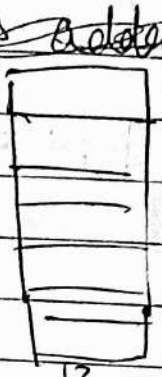
Direct Addressing / Absolute addressing mode

↳ Actual address is given in instruction.
↳ Use to access variables.



Disadvantage: If instruction size is fixed

Memory address cannot exceed 12 bit

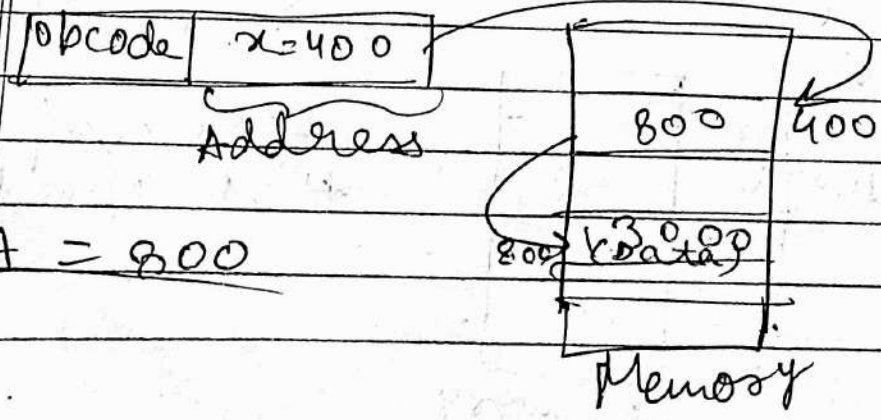


2^{12} = 0 to 4095 locations

Imp

Indirect addressing mode

↳ Used to implement pointers and passing parameters
↳ 2 memory access required.

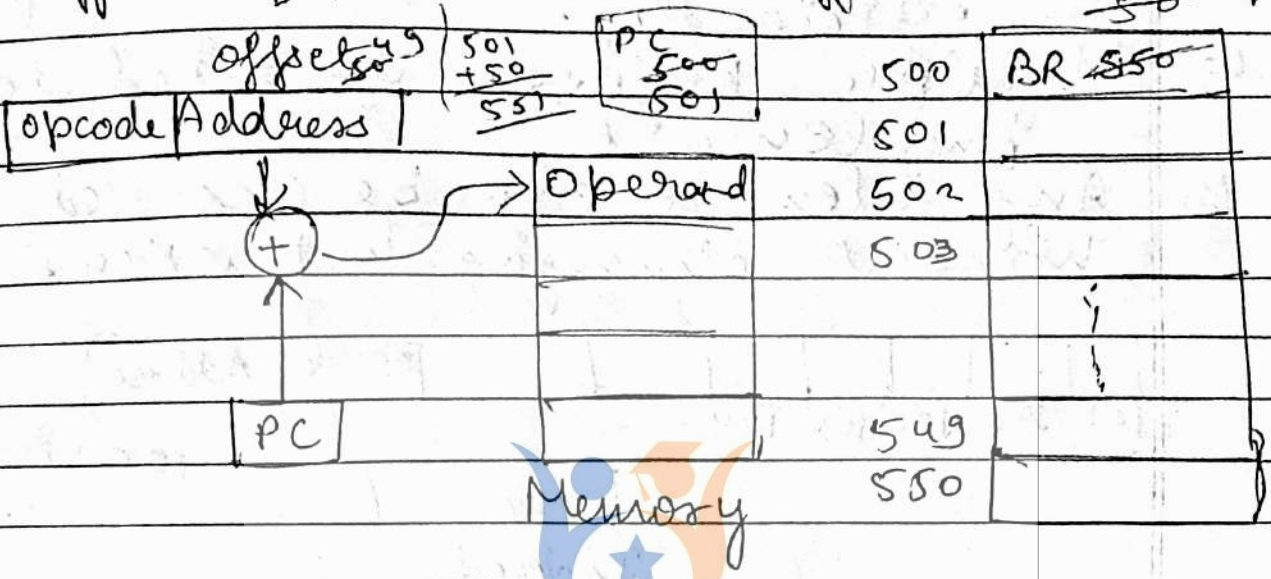


EA = 800

Page No.
Date:

Relative Addressing mode

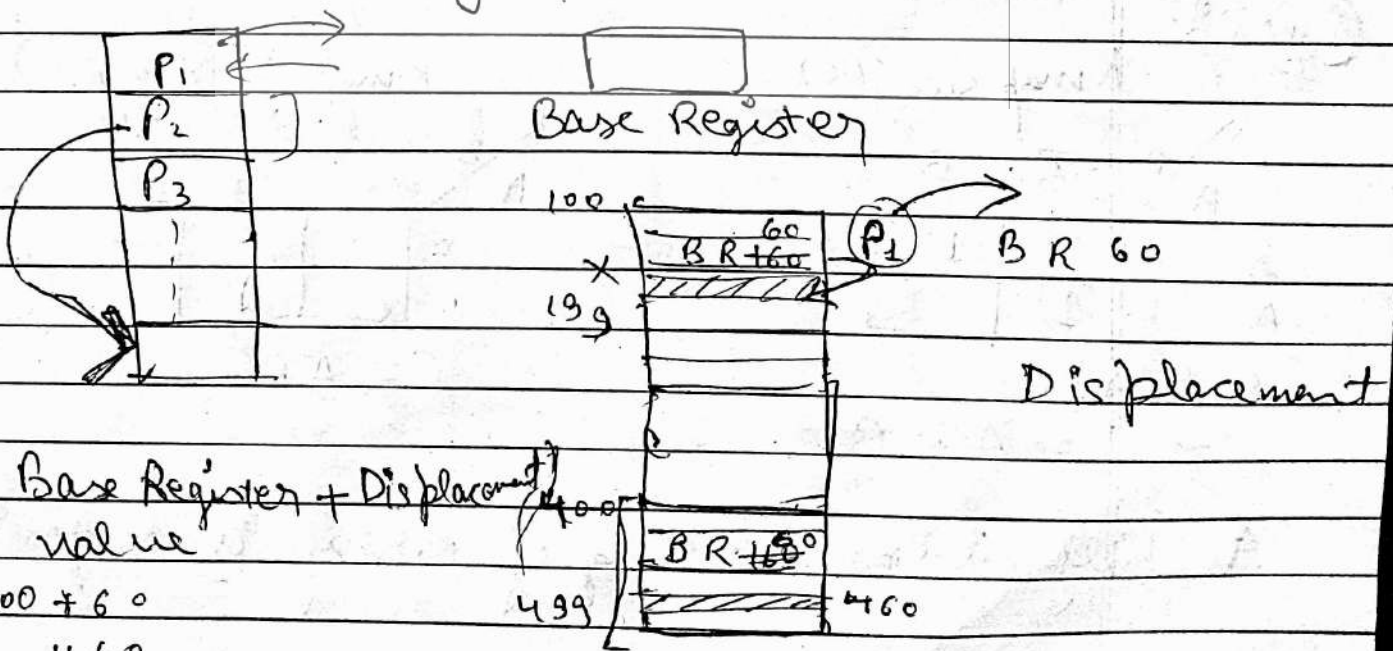
Effective Address = PC + offset



Used in instructions like branch, jump.

Base Register Addressing Mode

↳ Used in Program Relocation



EA = Base Register + Displacement value

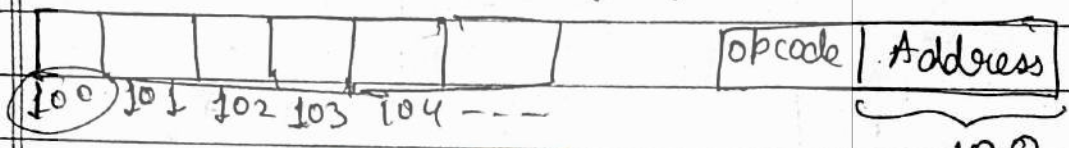
= 400 + 60

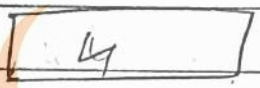
= 460

Page No.
Date:

Indexed Addressing Mode

↳ Use to access or implement array efficiently.
↳ Multiple Registers required to implement
↳ Any element can be accessed without changing instruction.



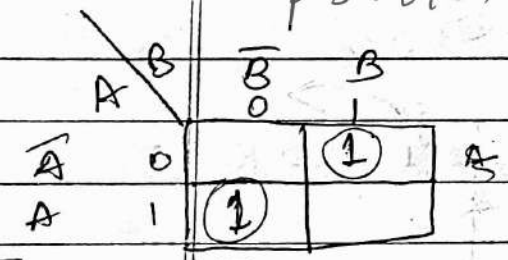


Indexed Register

EA = Base address + Index Register value
= 100 + 4
= 104

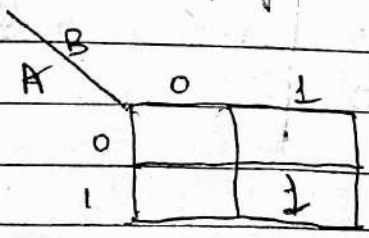
Ques 2

Kmap sum(S)



$S = \bar{A}B + A\bar{B}$
$\Rightarrow S = A \oplus B$

Kmap for carry(C)



$C = AB$

In half adder adding of carry is not possible.

Page No.	
Date:	

Unit 2: Arithmetic and Logic Unit

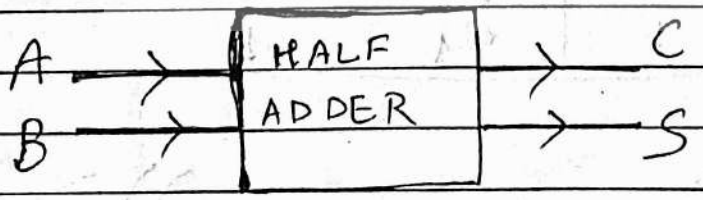
Syllabus: Look ahead carries adders Multiplication: Signed operand Multiplication, Booths algorithm and array multiplier. Division and logic operations. Floating point arithmetic operation, Arithmetic & logic unit design. IEEE Standard for Floating Point Numbers.

Half Adder (combinational logic circuit designed to add two single bit numbers)

0	0	1	1
+ 0	+ 1	+ 0	+ 1
0	1	1	10 → Sum
			Carry

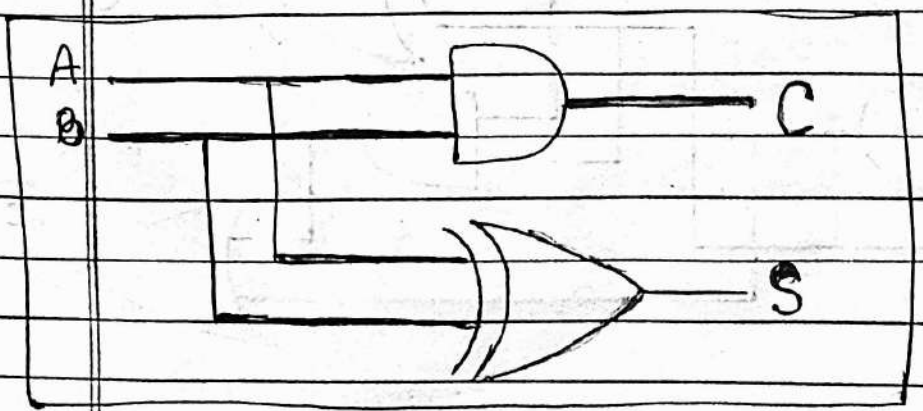
(2 input 2 outputs i.e. Sum & carry)

add 2 { digit (0/1), bits (0/1), decimal No. (any number)



Truth table

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



Half Adder Circuit

Full Adder

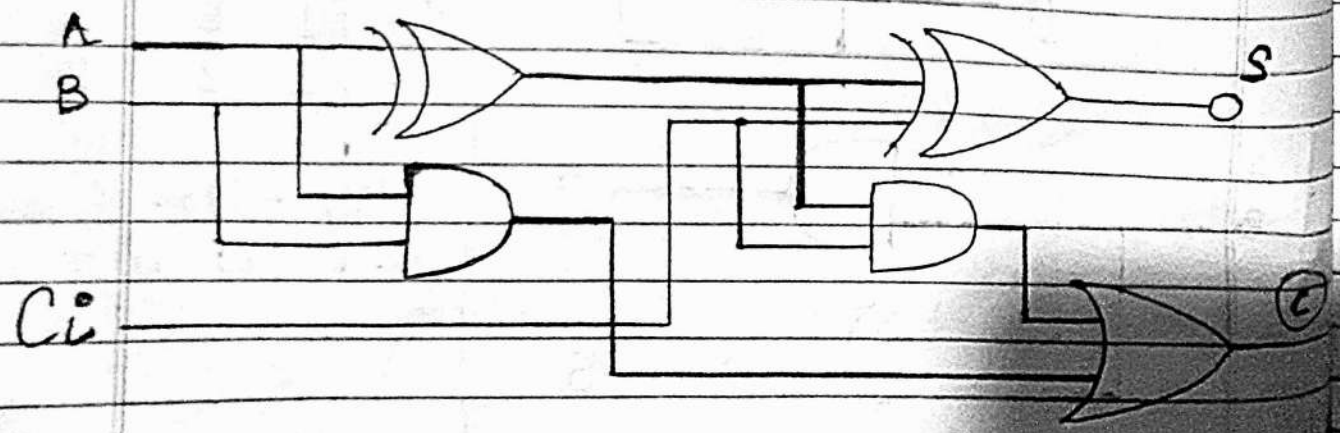
3-bit

```
  0        0        1
+ 0      + 1      + 1
---      ---      ---
  0        1       10  → Sum
                    ↳ Carry out
```

```
  1  1
  1  0  0  1      ← Carry input
  0  0  1  1
---------------
  1  1  0  0
```

Truth Table

A	B	Ci	S	Co
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

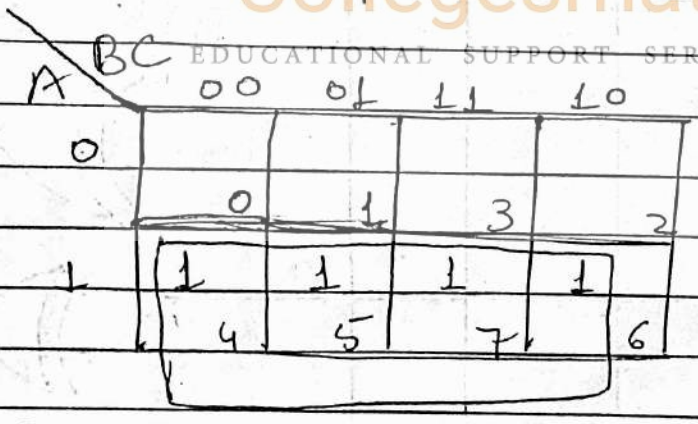


Page No.
Date:

Binary to Gray Code

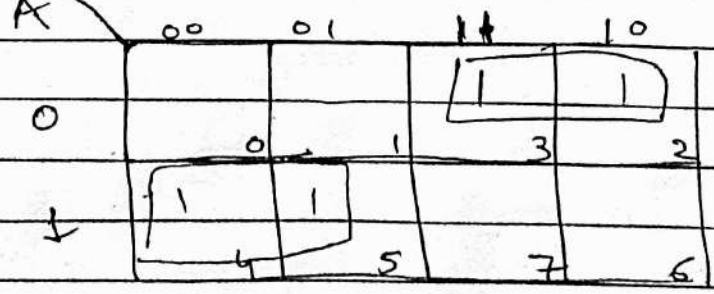
Decimal	Binary A	B	C	Gray D	E	F
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

Step 1: First bit as it is put.
Step 2: Next two bit and previous bit also include to operate XOR operation.



$D = A$

Kmap for E



$E = A\bar{B} + \bar{A}B$
$= A \oplus B$

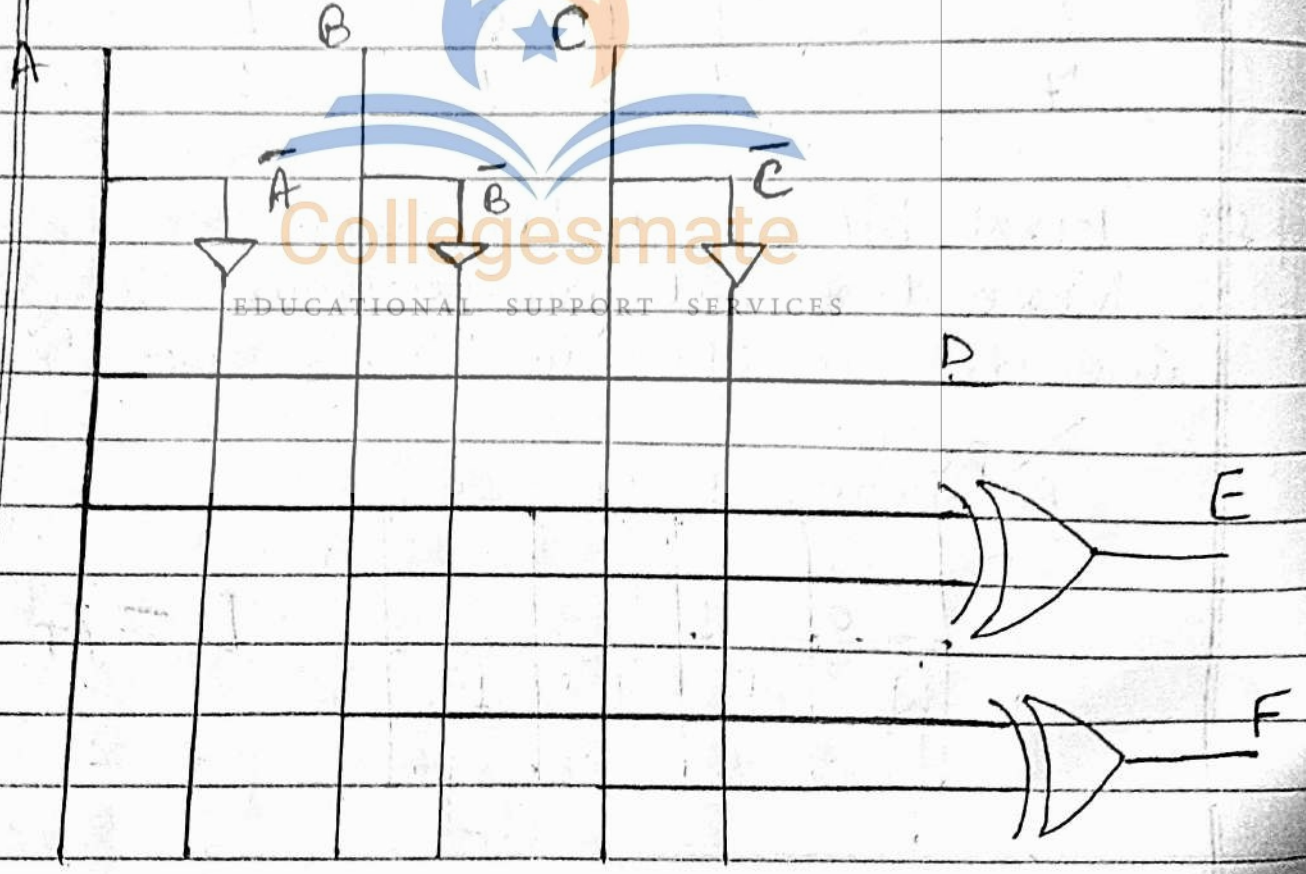
Page No.
Date:

KMap for F

A \ BC	00	01	11	10
0	0	1	3	2
1	4	5	7	6

(cells 1 and 5 are grouped; cells 2 and 6 are grouped)

$$F = \bar{B}C + B\bar{C} \quad \Rightarrow F = B \oplus C$$



Page No.
Date:

Gray Code to Binary Code

If given a gray code 1 0 1 1 0 1 1 0 1 to convert into Binary code.

Gray code			Binary Code		
A	B	C	B_1	B_2	B_3
0	0	0	0	0	0
0	0	1	0	0	1
0	1	1	0	1	0
0	1	0	0	1	1
1	1	0	1	0	0
1	1	1	1	0	1
1	0	1	1	1	0

Decoder: Decoder means decode the Input (n) and provide 2^n output

$n = 3$ (Input)

$2^n = 2^3 = 8$ (Output)

Page No.
Date:

3 x 8 Decoder Truth Table

E	A	B	C	D_7 (2^7)	D_6 (2^6)	D_5 (2^5)	D_4 (2^4)	D_3 (2^3)	D_2 (2^2)	D_1 (2^1)	D_0 (2^0)
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0
0	x	x	x	0	0	0	0	0	0	0	0

$D_7 = A\,B\,C$

$D_6 = A\,B\,\bar{C}$

$D_5 = A\,\bar{B}\,C$

$D_4 = A\,\bar{B}\,\bar{C}$

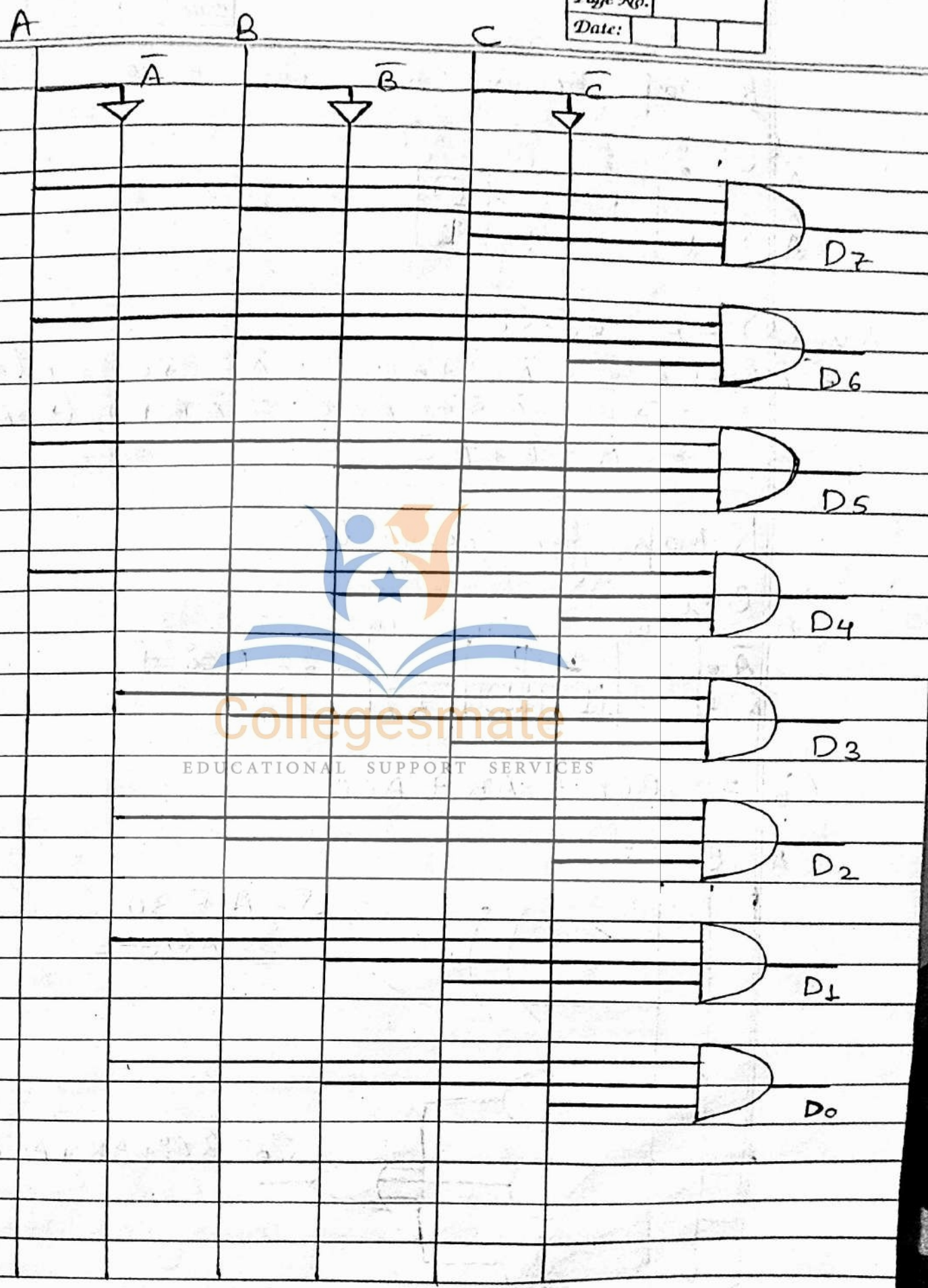
$D_3 = \bar{A}\,B\,C$

$D_2 = \bar{A}\,B\,\bar{C}$

$D_1 = \bar{A}\,\bar{B}\,C$

$D_0 = \bar{A}\,\bar{B}\,C$

Page No.
Date:



Page No.
Date:

Full Adder

K Map for Sum (S)

A \ BCi	$\bar{B}\bar{C}_i$ 00	$\bar{B}C_i$ 01	BC_i 11	$B\bar{C}_i$ 10
$\bar{A}$ 0		1		1
A 1	1		1	

$$Sum(S) = A \oplus B \oplus C_i$$

$$= \bar{A}BC + A\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + ABC = \bar{A}B(C+\bar{C}) + \bar{B}(\bar{A}C + A\bar{C})$$

$$= \bar{A}B + \bar{A}\bar{B}\bar{C} + AB\bar{C} = \bar{A}B + \bar{B}(\bar{A}\bar{C} + A\bar{C})$$

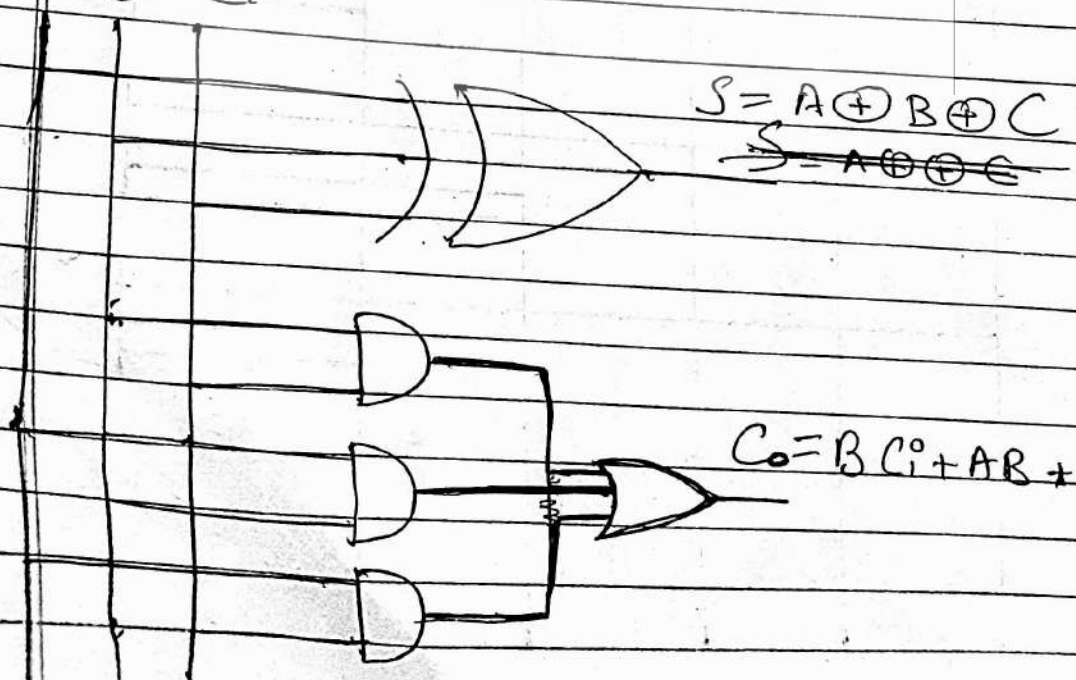
$$= \bar{A}(B + \bar{B}\bar{C})$$

K Map for Carry (C_o)

A \ BCi	00	01	11	10
$\bar{A}$ 0		1		
A 1	1	1		1

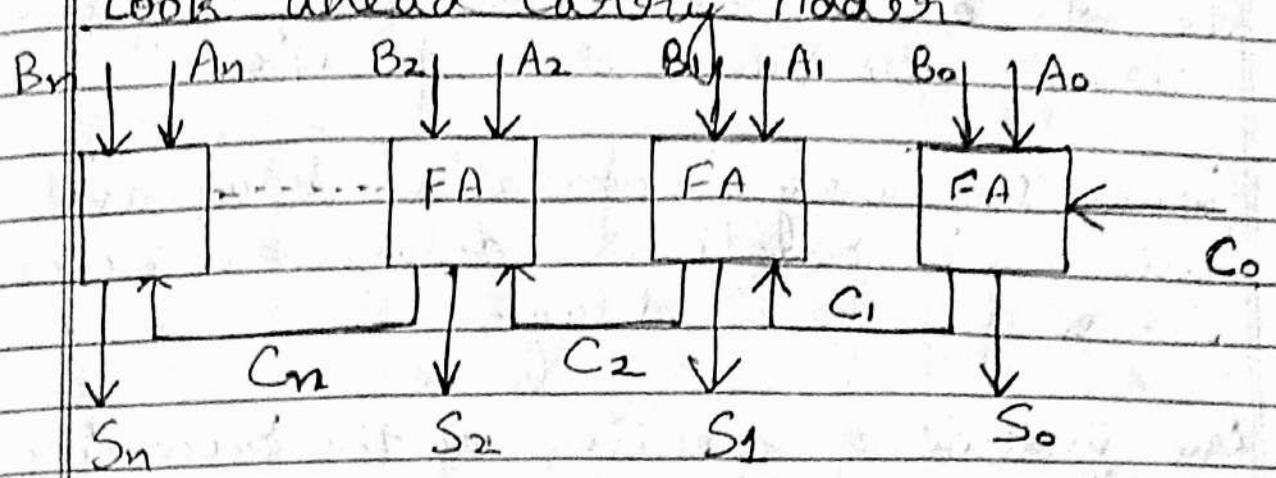
$$C_o = BC_i + AB + AC_i$$

A B C_i



Page No.
Date:

Look ahead Carry Adder



n bit adder repeat carry Adder discuss in the last section is implemented using Full adder stage in which the carry output of each full adder stage is connected to the carry Input of the next higher order stage.

The sum and output of any stage can not be produced until the input carry occurs this lead to time delay in the addition process. This delay is known as propagation delay.

Eg:

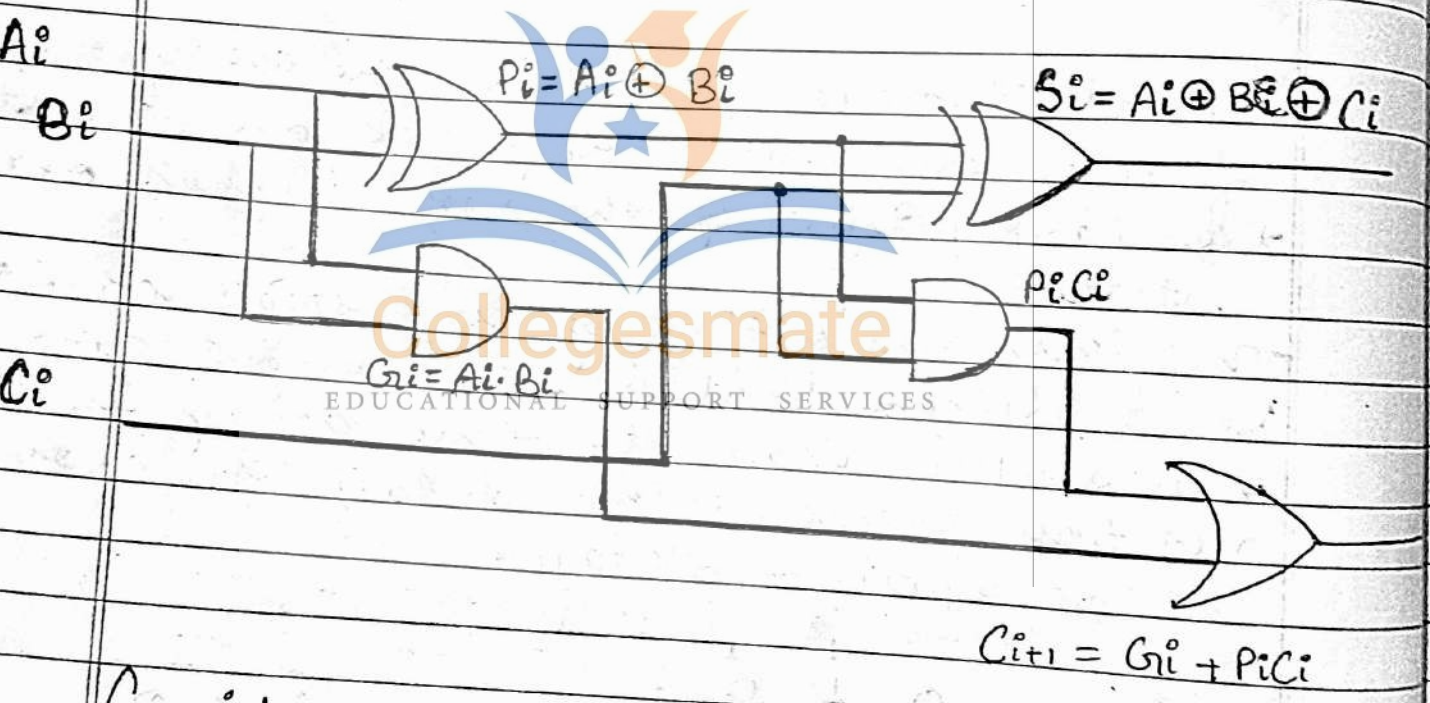
C_4	C_3	C_2	C_1	
	1	1		
0	1	0	1	
0	0	1	1	$C_0 \leftarrow 0$
1	0	0	0	

Addition of the LSB position produce a carry in to the second position. This carry added to the second position produce a carry into the third position. The later carry when added

to the bit of the third position produce carry into the last position.

Generally carry propagation delay and sum propagation delay are increase in terms of gate delay.

One method of speeding up this process by eliminating interstage carry delay is called look-ahead carry adder.



Consider a circuit of Full adder we define two function carry generate and carry propagate.

$$P_i = A_i \oplus B_i$$

$$G_i = A_i B_i$$

The output Sum and Carry can be expressed as:

$S_i = P_i \oplus C_i$

$C_{i+1} = G_i + P_i C_i$

G_i is called carry generate and it produce on carry when both A_i and B_i are one regardless of the input carry. P_i is called carry propagate because it is termed associated with the propagation of the carry form G_i to C_{i+1}.

$$\boxed{C_{i+1} = G_i + P_i C_i}$$

$i = 0$

$$\boxed{C_1 = G_0 + P_0 C_0}$$

$i = 1$

$$C_2 = G_1 + P_1 C_1$$
$$= G_1 + P_1 (G_0 + P_0 C_0)$$

$$\boxed{C_2 = G_1 + P_1 G_0 + P_1 P_0 C_0}$$

$i = 2$

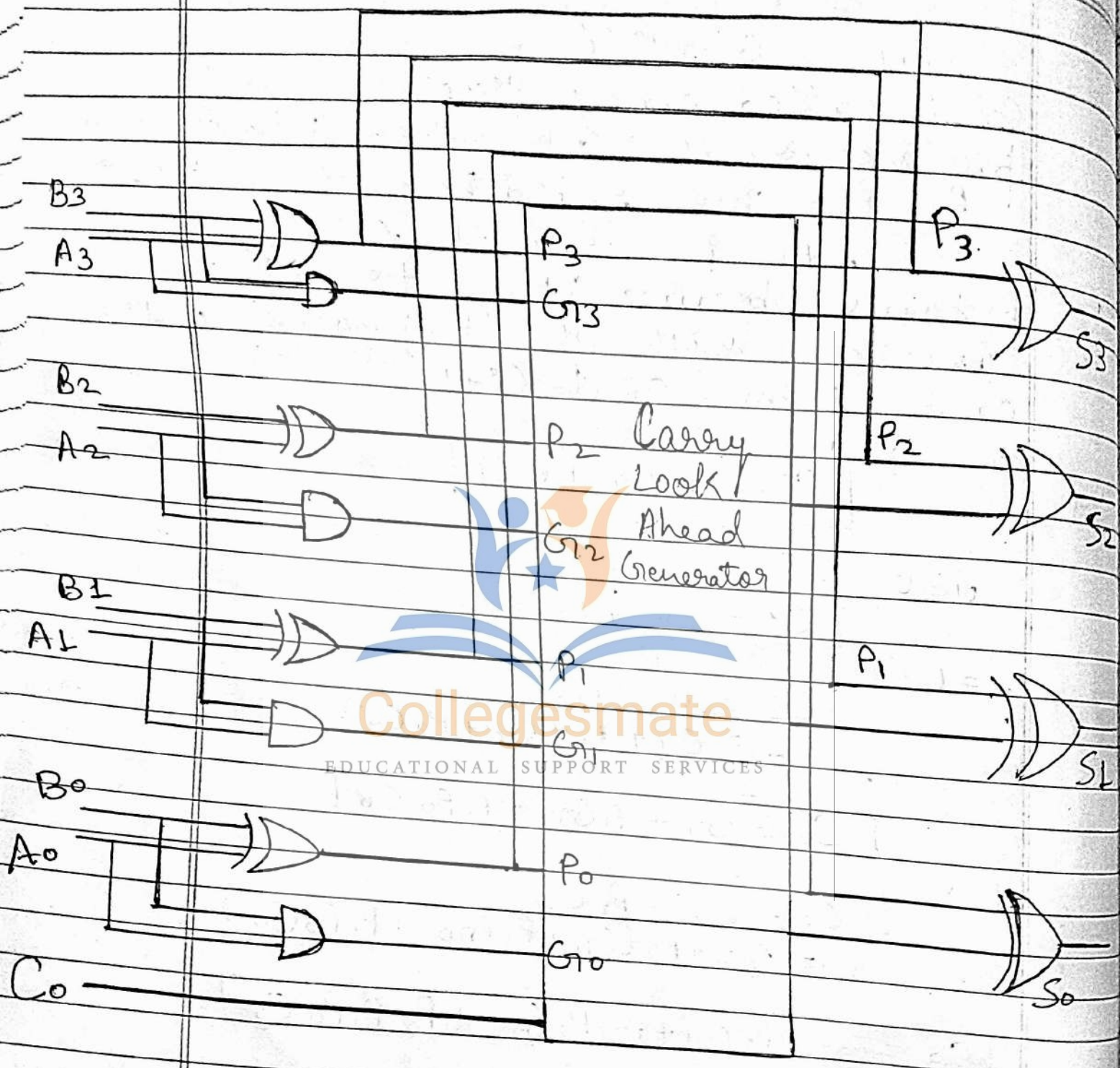
$$C_3 = G_2 + P_2 C_2$$
$$= G_2 + P_2 (G_1 + P_1 G_0 + P_1 P_0 C_0)$$

$$\boxed{C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0}$$

$i = 3$

$$C_4 = G_3 + P_3 C_3$$
$$C_4 = G_3 + P_3 (G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_0)$$

$$\boxed{C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_0}$$



Page No.
Date:

Binary Multiplication

Binary multiplication follow some rules:

$0 * 0 = 0$

$0 * 1 = 0$

$1 * 0 = 0$

$1 * 1 = 1$

Eg: Perform multiplication 12×3

Step1: Write a Binary of 12 (8 bit representation)

0 0 0 0 1 1 0 0

Step2: Write the binary of 3 (8 bit representation)

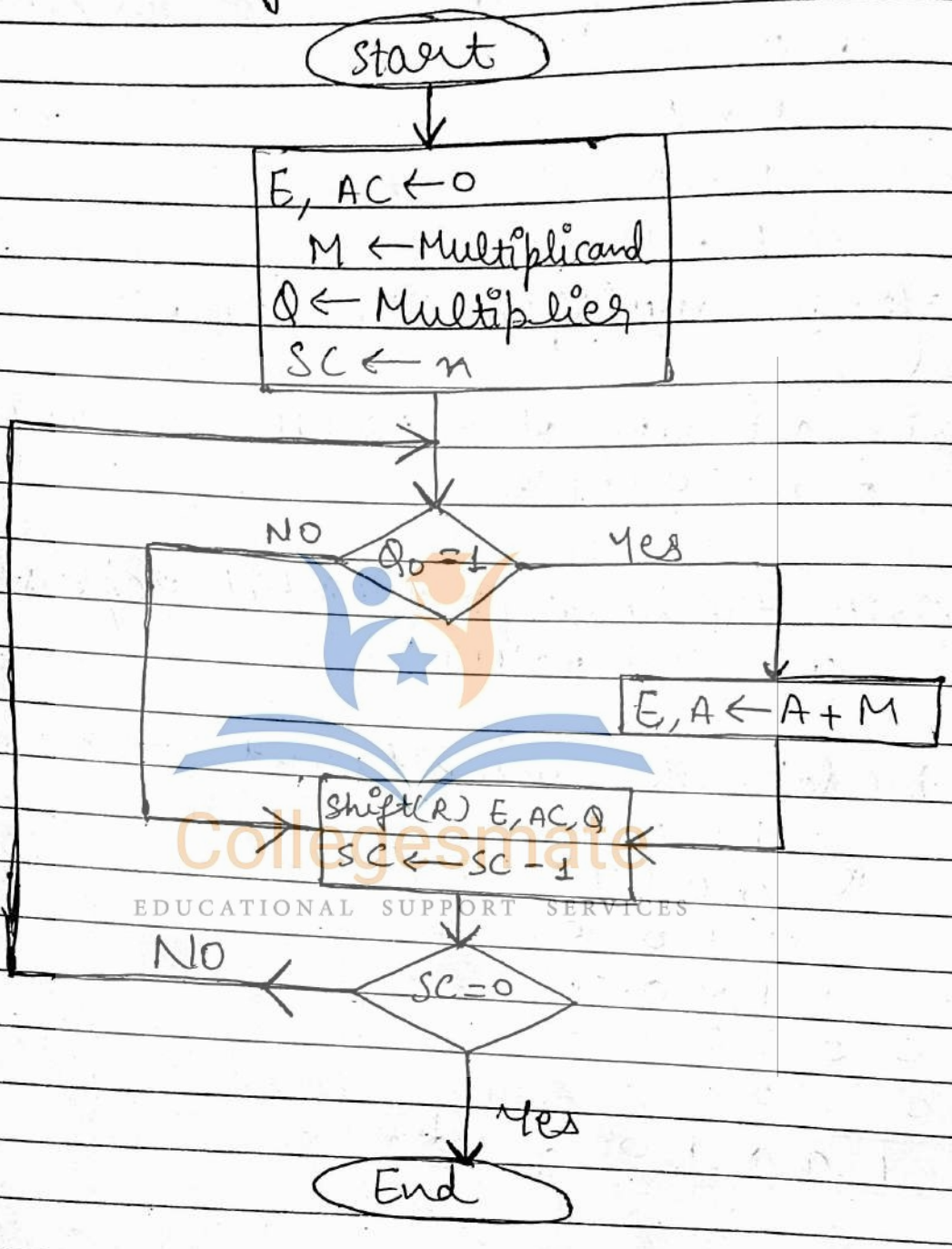
0 0 0 0 0 0 1 1

Step3: Neglet the insignificant zeros.

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

Ans

Flowchart of Unsigned Binary Multiplication



Eg: 3 x 4 to perform multiplication

Sol: Step 1: Write the Binary of decimal no.

3 → 0011 → M → Multiplicand

4 → 0100 → Q → Multiplier

3 × 4
↓ ↓
M Q

Page No.	
Date:	

Step 2: Write down the Table.

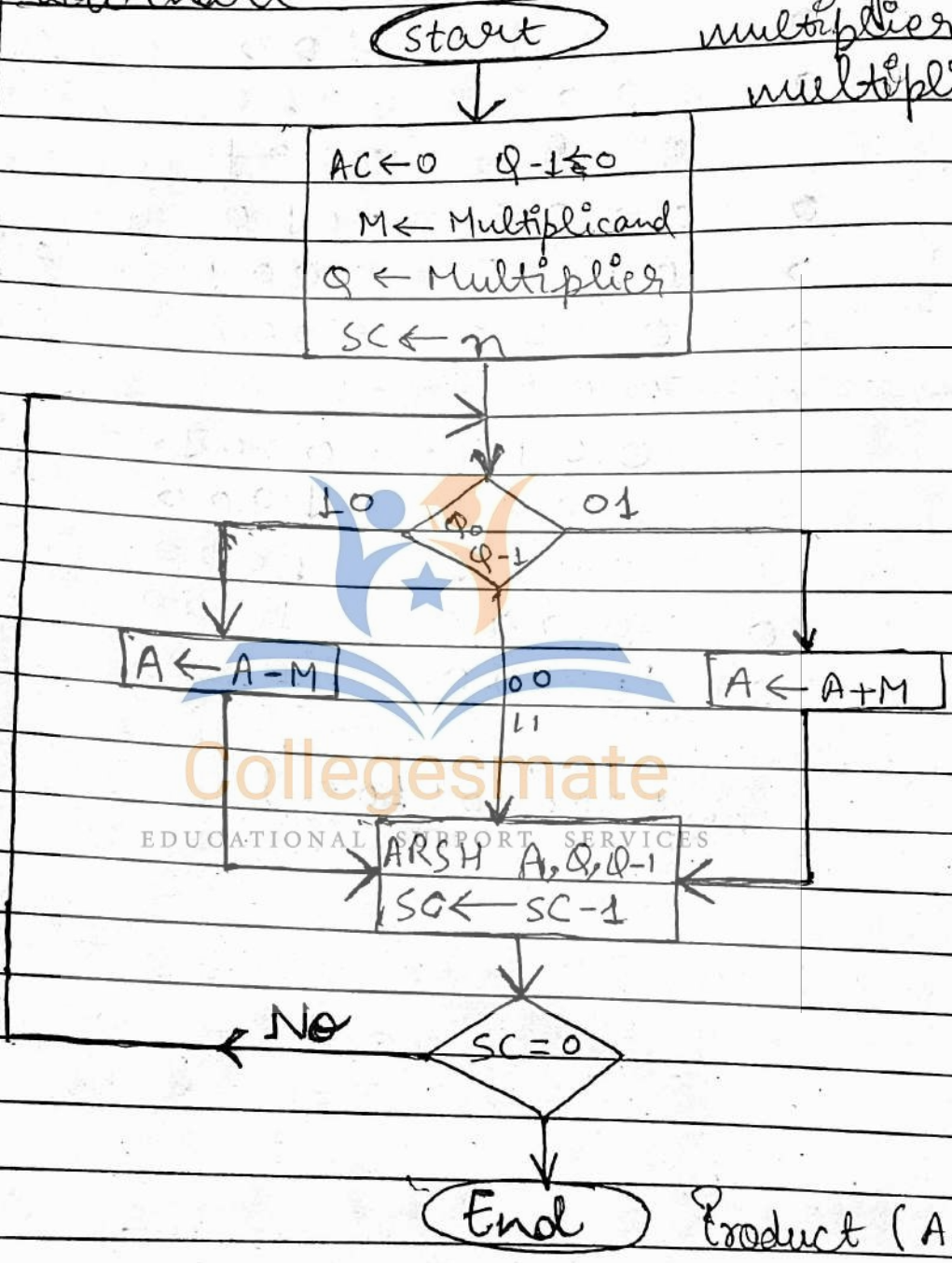
Count (SC)	E	A	Q	Steps
4	0	0000	0100	Initial
4	0	0000	0010	Shift right (E, AC, Q)
~~3~~	~~0~~	~~0000~~	~~0001~~	
3	0	0000	0010	SC ← SC − 1
3	0	0000	0001	Shift right (E, AC, Q)
2	0	0000	0001	SC ← SC − 1
~~2~~	~~0~~	~~0000~~		
2	0	0011	0001	EA ← A + M
2	0	0001	1000	Shift right (E, AC, Q)
1	0	0001	1000	SC ← SC − 1
1	0	0000	1100	Shift right (E, AC, Q)
0	0	0000	1100	SC ← SC − 1

Ans: 00001100

Page No.
Date:

Binary Signed Number Multiplication using Booth's Algorithm

Flow chart



max
SC = no. of bits in multiplier or multiplicand

Product (A.Q)

where

ARSH : Arithmetic Right Shift

Page No.
Date:

Four Cases are used

(a) Both (+ve) no. i.e. $(+A) * (+B)$
(b) Both (-ve) no. i.e. $(-A) * (-B)$
(c) First no. (+ve) and second no. (-ve) i.e. $(+A) * (-B)$
(d) First no. (-ve) and second (+ve) i.e. $(-A) * (+B)$

Case 1: $(+5) * (+4)$ using Booth's algorithm.

Step 1: Create a Table format

SC	A	Q	Q-1	Steps

Step 2: Write the Binary of Both Number.

$(+5) = 101$
$(+4) = 100$

Step 3: Provided a sign bits of both number and also calculated two's complement of Both no.

$(+5) = 0101$ $\quad$ $1010 + 1 = 1011$ (-5)
$(+4) = 0100$ $\quad$ $1011 + 1 = 1100$ (-4)

Step 4: First value Represent of M (Multiplicand)

$M = 0101$ $(+5)$

Second value represent of Q (Multiplier)

$Q = 0100$ $(+4)$

Step 5: Put all values in the Table.

Step 6:

SC	AC	Q	Q-1	Steps
100	0000	0100	0	Initial

Page No.
Date:

Step 7: SC means number of bits are used in multiplicand and multiplier. So, 4 bits are used SC value = 4 (100)

Step 8: Write down the initial value.

Step 9: check the value Q0 Q-1 if found 00. So next op^{n} to be perform shifting - if found 01 to be perform A ← A+M, if found 10 to be performed A ← A-M

SC	AC	Q	Q-1	Steps
4	0000	0100	0	Initial
4	0000	0010	0	Arithmetic shift
3	0000	0010	0	SC ← SC-1
3	0000	0001	0	Arithmetic shift
2	0000	0001	0	SC ← SC-1
2	1101	0001	0	A ← A-M
1	1101	1000	1	Arithmetic shift
1	10010	1000	1	~~SC ← SC-1~~ ~~A ← A+M~~
1	10010	1000	1	~~A ← A+M~~
1	0010	1000	1	A ← A+M
1	0001	0100	0	A ← A+M
0	0001	0100	0	Arithmetic shift

Step 10: Final value is SC = 0 then stop the process and Resultant of (A.Q)

Page No.
Date:

Step 11: AC = 0001 Q = 0100

Ans:

0 0 0 1 0 1 0 0 (+20)

sign bit

Case 2: One (+ve) and second (-ve)

Eg: (+5) * (-4)

⇒ M = +5 = 0101

Q = +4 = 0100

M = 0101

1's compl = 1010

+1

2's complement = 1011

M(-5) = 1011

Q = +4 = 0100

1's comp = 1011

+1

2's comp: 1100

Q = -4 = 1100

SC	AC	Q	Q-1	Steps
0100	0000	1100	0	Initial
0011	0000	0110	0	ARSH, SC←SC-1
0010	0000	0011	0	ARSH, SC←SC-1
	+1011			A ← A-M
	1011			
0010	1011	0011	0	A ← A-M
0001	1101	1001	1	ARSH, SC←SC-1
0000	1110	1100	1	ARSH, SC←SC-1

A.Q

Ans: 1 1 1 0 1 1 0 0 ⇒ 2's complement Result

sign bit ~~1 1 0 0~~ 1 1 0 1 1 0 0

0 0 1 0 0 1 1

+1

0 0 1 0 1.0 0 ⇒ (-20) Ans

Page No.
Date:

Case 3: $(-5) * (+4)$

Soln:

$M = -5 = 1011$

$Q = +4 = 0100$

$M = +5 = 0101$

$1's = 1010$

$+1$

$2's = 1011$

SC	AC	Q	Q-1	Steps
4	0000	0100	0	Initial
3	0000	0010	0	ARSH, SC←SC-1
2	0000	0001	0	ARSH, SC←SC-1
	0101			
2	0101	0001	0	A←A-M
2	0101	0001	0	A←A-M
1	0010	1000	1	ARSH, SC←SC-1
	1011			~~A←A+M~~
1	1101	1000	1	AC←A+M
0	1110	1100	0	ARSH, SC←SC-1

A.Q: 1 1 1 0 1 1 0 0 — 2^s complement

sign

1's: 0 0 1 0 0 1 1

+1

2's: 1 0 0 1 0 1 0 0 = (−20)

Page No.
Date:

Case 4: (-5) * (-4)

M = -5 = 1011 M = +5 = 0101

Q = -4 = 1100 Q = -4 = 0100

SC	AC	Q	Q-1	Steps
4	0000	1100	0	Initial
3	0000	0110	0	ARSH, SC ← SC-1
2	0000	0011	0	ARSH, SC ← SC-1
	+0101			A ← A - M
	0101			
2	0101	0011	0	A ← A - M
1	0010	1001	1	ARSH, SC ← SC-1
0	0001	0100	1	ARSH, SC ← SC-1
	+1011			~~A ← A+M~~
0	1100	0		

Ans: A.Q: 00010100

Sign bit

Binary Array Multiplier

An array multiplier is a digital combinational circuit used for multiplying two binary numbers by employing an array of full adders and half adders.

A : A_1 A_0

B : b_1 b_0

A_1 A_0

× B_1 B_0

A_1B_0 A_0B_0

A_1B_1 A_0B_1

A_1B_1 $A_1B_0+A_0B_1$ A_0B_0

Multiplicand ← 1 0 1 1 × 1 1 0 1 → Multiplier

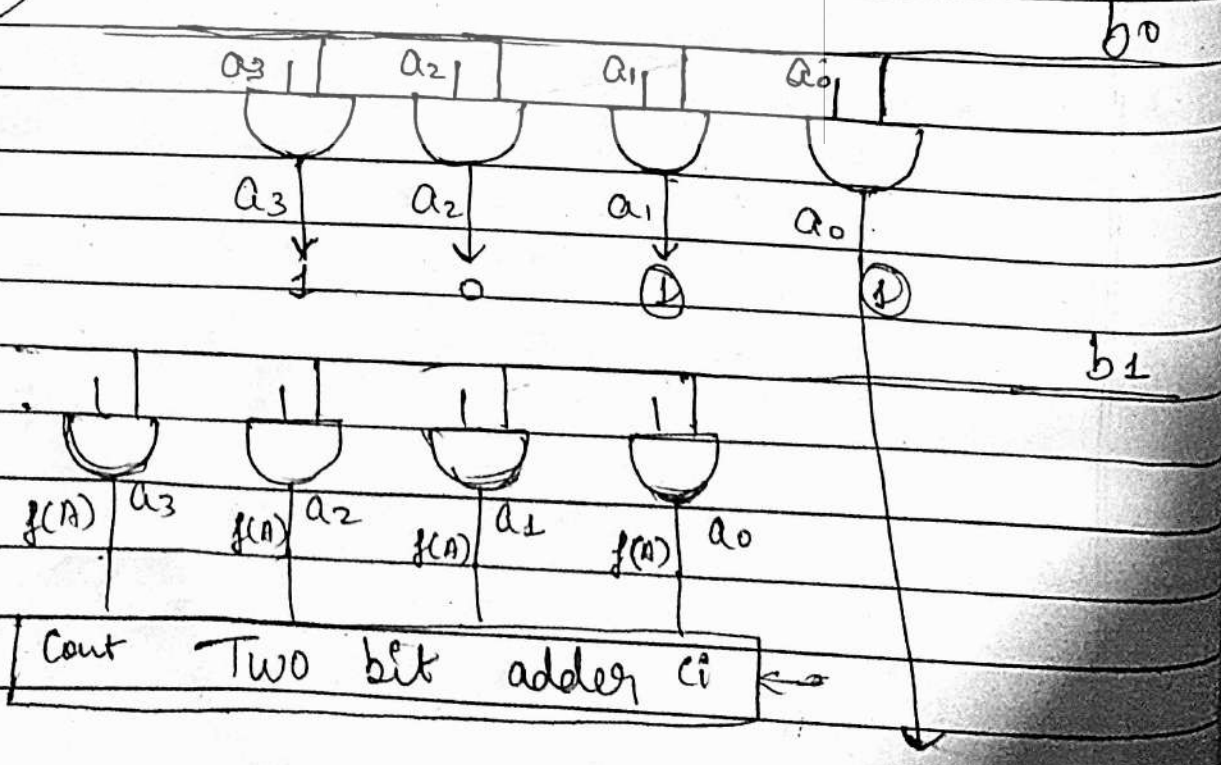
a_3 a_2 a_1 a_0

1 0 1 1

① AND

1 0 1 1

1 level circuit



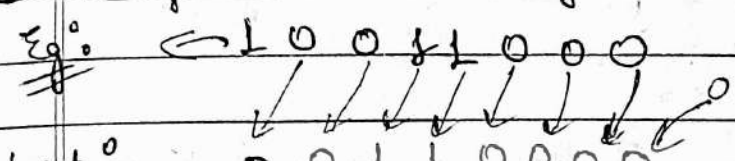
Shift Microoperation

Shift micro operation are those that shift the binary digits in a register either leftwards or rightwards.

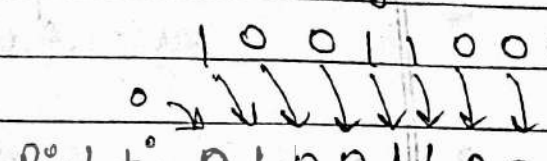
Categories of shift microoperation

① Logical

Eg:



Left :



Right :

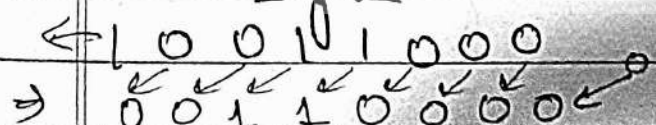
② Circular shift microoperation

Left

Right

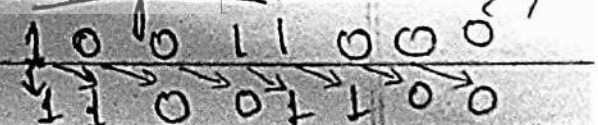
③ Arithmetic shift microoperation

Left



(Same as logical shift left)

Right



Date:

Floating Point Number addition

Algorithm

1. Select a floating point number and then compare their exponent.
2. Choose the highest exponent and align the mantissa.
3. Perform addition of the two numbers.

FPN Representation

$$N = \pm M * r^{\pm E}$$

r: radix / Base

Eg: $X = 0.9504 \times 10^3$

$Y = 0.8200 \times 10^2$

Exponent of

$X = 3$

$Y = 2$

Exponent of x > Exponent of y

$\therefore x > y$

$X = 0.9504 \times 10^3$

$Y = 0.8200 \times 10^2 = 0.0\,8200 \times 10^3$

$X + Y = 0.9504 \times 10^3 + 0.0\,8200 \times 10^3$

= add the Mantissa

$= (0.9504 + 0.0\,8200) \times 10^3$

$= 1.03240 \times 10^3$

Floating Point Multiplication

Page No.
Date:

Step 1: Check for zero
Step 2: Add the exponents and subtract 127 (bias).
Step 3: Multiply the mantissa and determine of sign of the resultant.
Step 4: Normalize the resulting value if necessary.

Eg:

1	00000111	1000...000	$-1.5 \times 2^{7-127}$
S	E	M	

0	11100000	1000...0000	$1.5 \times 2^{224-127}$

add exponent:

00000111 + 11100000
= 11100111 (231)

add the exponent 231 − 127
subtract bias 11100111 − 01111111
= 01101000 (104)

Multiply mantissa
1.1000--- × 1.1000--- = 10.01000---

Normalize
$10.01000\text{---} \times 2^{104} = 1.001000 \times 2^{105}$

Ans

1	01101001	001000
S		M

Page No.
Date:

Division

Step 1: If the dividend is zero report result as 0.

Step 2: If the divider is zero report result as ∞.

Step 3: Subtract exponent and bias.

Step 4: Divide the mentisa and determine sign of the resultant

Step 5: Normalize the result.