27/8/24
15

- **Shift Micro-operations in computer Architecture**

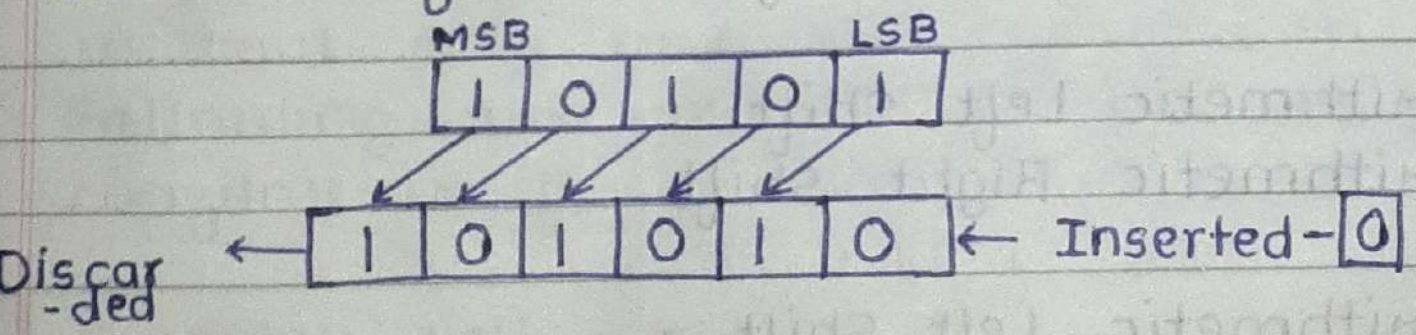
⇒ Shift micro-operations are those micro-operations that are used for the serial transfer of information. These are also used in conjunction with arithmetic micro-operation logic micro-operation, and other data-processing operations. There are three types of shift micro-operations:

- **Logical Shift:-**

⇒ It transfers the 0 through the serial input. We use the symbols '<<' for the logical left shift and '>>' for the logical right shift

- **Logical Left Shift:-**

⇒ In this shift, one position moves each bit to the left one by one. The Empty least significant bit (LSB) is filled with zero (i.e, the serial input), and the most significant bit (MSB) is rejected. It is denoted by the double left arrow key. (<<)

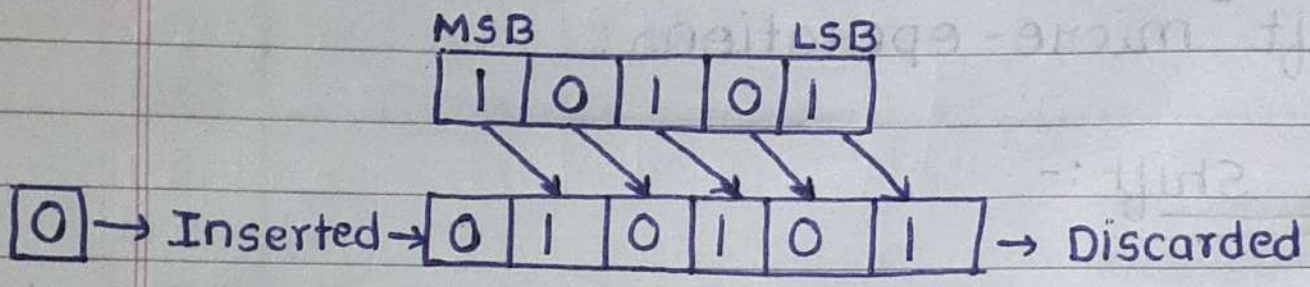


NOTE ⇒ Every time we shift a number towards the left by 1 bit it multiplies

that number by 2.

- **Logical Right Shift :-**

⇒ In this shift, each bit moves to the right one by one and LSB is rejected and the empty MSB is filled with zero. It is denoted by (>>).



NOTE ⇒ Every time we shift a number towards the right by 1 bit it divides that number by 2.

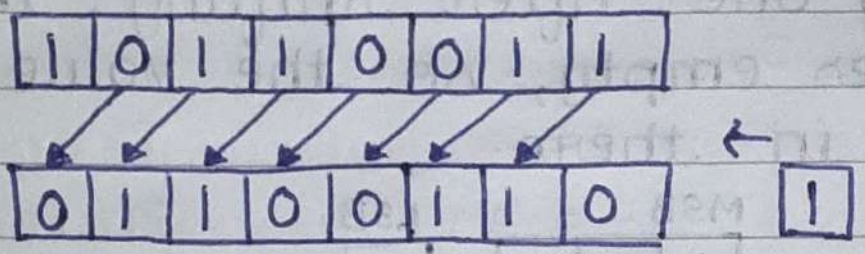
- **Arithmetic Shift :-**

The arithmetic shift micro-operation moves the signed binary number either to the left or to the right position. Following are the two ways to perform the arithmetic shift.

1. Arithmetic Left Shift
2. Arithmetic Right Shift

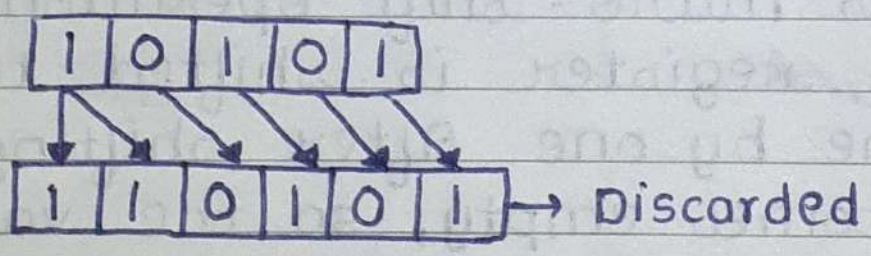
- **Arithmetic Left Shift :-**

In this shift, each bit is moved to the left one by one. The (LSB) is filled with zero and (MSB) is rejected. Same as the Left Logical Shift.



- **Arithmetic Right Shift:-**

⇒ In this shift, each bit is moved to the right one by one and (LSB) bit is rejected and the empty (MSB) is filled with the value of the previous MSB.



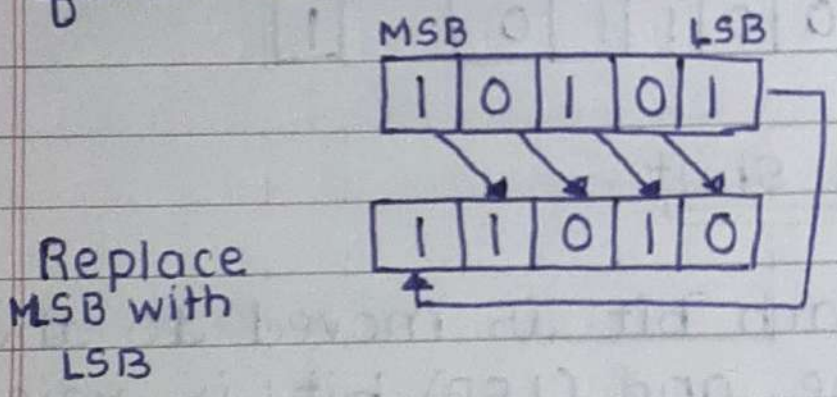
- **Circular Shift:-**

⇒ The circular shift circulates the bits in the sequence of the register around both ends without any loss of information.
Following are the two ways to perform circular shift.

1. Circular Left Shift
2. Circular Right Shift

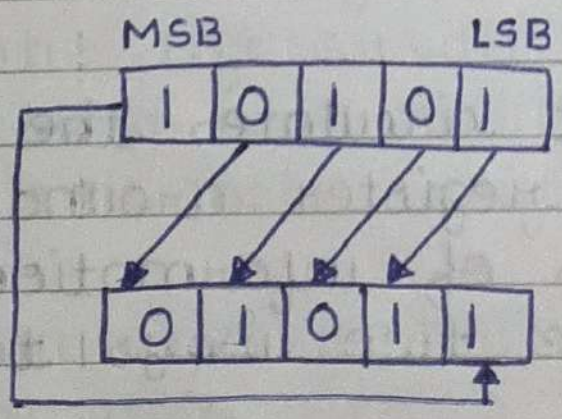
- **Circular Right shift:-**

⇒ In this micro-shift operation each bit in the register is shifted to the right one by one. After shifting, the MSB becomes empty, so the value of LSB is filled in there.



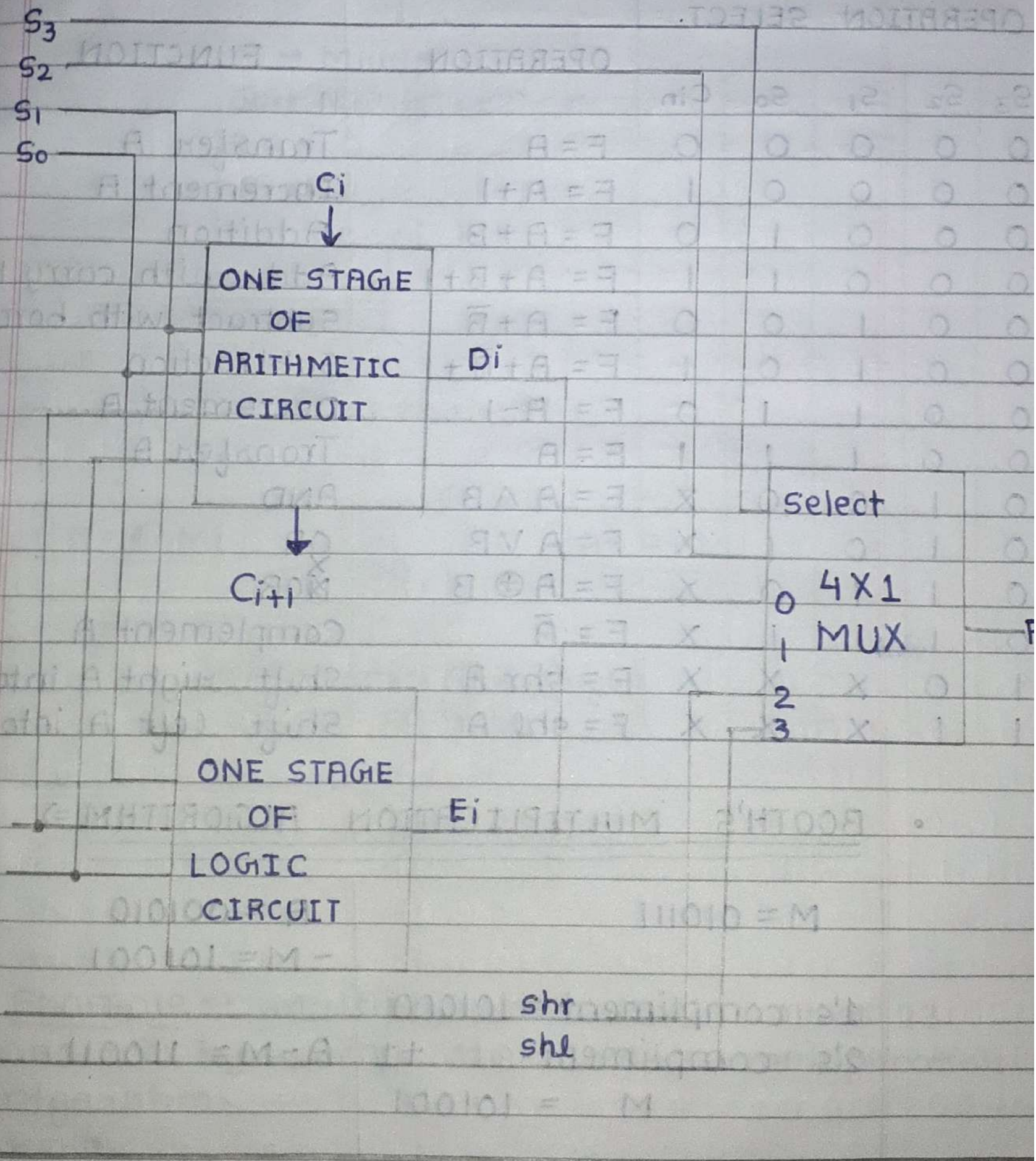
- **Circular Left shift:-**

⇒ In this micro-shift operation each bit in the register is shifted to the left one by one. After shifting, the LSB becomes empty, so the value of MSB is filled in there.



Replace LSB with MSB.

- ONE STAGE OF ARITHMETIC LOGIC SHIFT UNIT =>



- **FUNCTION TABLE FOR ARITHMETIC LOGIC SHIFT UNIT =>**

OPERATION SELECT					OPERATION	FUNCTION
S_3	S_2	S_1	S_0	Cin		
0	0	0	0	0	F = A	Transfer A
0	0	0	0	1	F = A + 1	Increment A
0	0	0	1	0	F = A + B	Addition
0	0	0	1	1	F = A + B + 1	Add with carry 1
0	0	1	0	0	F = A + $\bar{B}$	Subtract with borrow
0	0	1	0	1	F = A + $\bar{B}$ + 1	Subtraction
0	0	1	1	0	F = A - 1	Decrement A
0	0	1	1	1	F = A	Transfer A
0	1	0	0	X	F = A $\wedge$ B	AND
0	1	0	1	X	F = A $\vee$ B	OR
0	1	1	0	X	F = A $\oplus$ B	XOR
0	1	1	1	X	F = $\bar{A}$	Complement A
1	0	X	X	X	F = shr A	Shift right A into F
1	1	X	X	X	F = shl A	Shift left A into F

- **BOOTH'S MULTIPLICATION ALGORITHM =>**

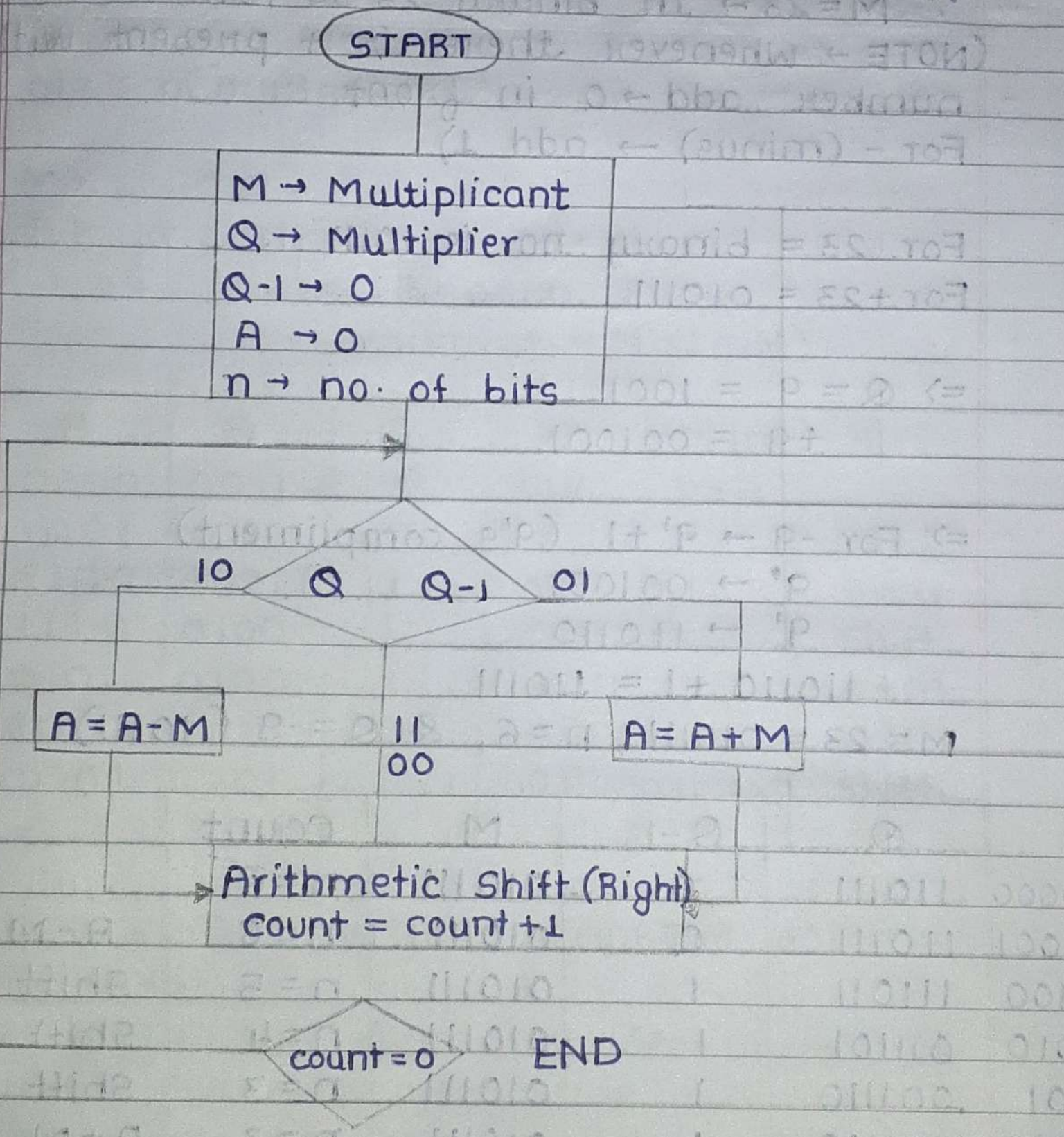
M = 010111

1's compliment = 101000
2's compliment = +1
M = 101001

A = 001010
-M = 101001

A - M = 110011

- **FLOWCHART ⇒**



- **Example ⇒** Multiply the two number 23 and -9 by using the Booth's multiplication algorithm.

⇒ Here

M → multiplicant, is 23

M = 23

Q = -9

∵ M = 23 → In binary 23 is 10111

(NOTE → Whenever there is + present with number add → 0 in front.
For - (minus) → add 1)

For 23 = binary no. is 10111
For +23 = 010111

⇒ Q = 9 = 1001
+9 = 001001

⇒ For -9 → 9' +1 (9's compliment)
9' → 001001
9' → 110110
110110 +1 = 110111

M = 23 = 010111, n = 6, & Q = -9 (110111)

A	Q	Q-1	M	Count	
000000	110111	0	010111	n = 6	
101001	110111	0	010111	n = 6	A - M
110100	111011	1	010111	n = 5	Shift
111010	011101	1	010111	n = 4	Shift
111101	001110	1	010111	n = 3	Shift
010100	001110	1	010111	n = 3	A + M
001010	000111	0	010111	n = 2	Shift
110011	000111	0	010111	n = 2	A - M
111001	10011	1	010111	n = 1	Shift
111100	110001	1	010111	n = 0	Shift.

Q. Multiply 7 x 3 by using booth multiplication algorithm.

=> Q = 3 (Multiplier)
M = 7 (Multiplicant)

M = 7 = 0111
Q = 3 = 0011
n = 4, hence A = 0000
Also, M(2' compliment) → M' + 1 = -M)

A	Q	Q-1	M	Count	
0000	0011	0	0111	n = 4	
1001	011	0	0111		A - M
1100	1001	1	0111	n = n-1 = 3	Shift
1110	0100	1	0111	n = n-1 = 2	Shift
0101	0100	1	0111		A + M
0010	1010	0	0111	n = n-1 = 1	Shift
0001	0101	0	0111	n = n-1 = 0	Shift

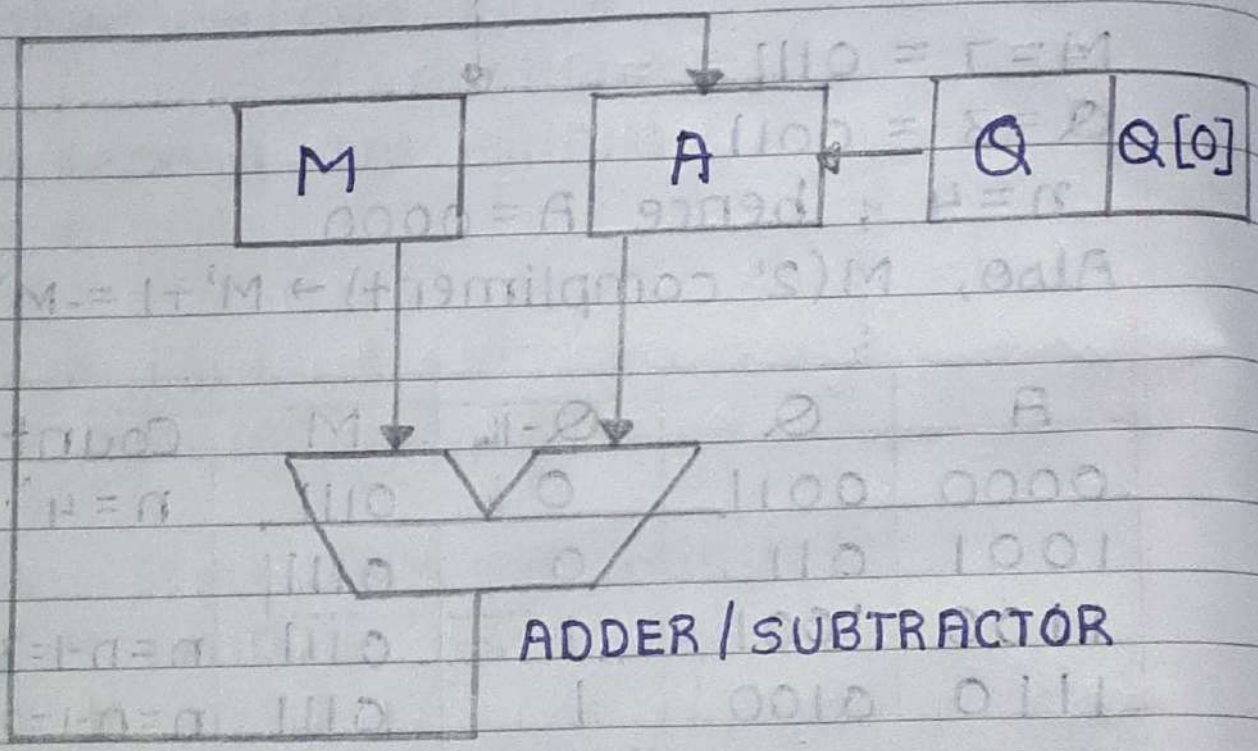
- RESTORING DIVISION ALGORITM FOR UNSIGNED INTEGER =>

=> A division algorithm provides a quotient and a remainder when we decide two number. They are generally of two types slow and fast algorithm.

- SLOW ALGORITM AND FAST ALGORITHM =>

=> Slow division algorithm are restoring, non-restoring, non-performing restoring, SRT algorithm and under fast comes Newton-Raphson and Goldschmidt. In this article,

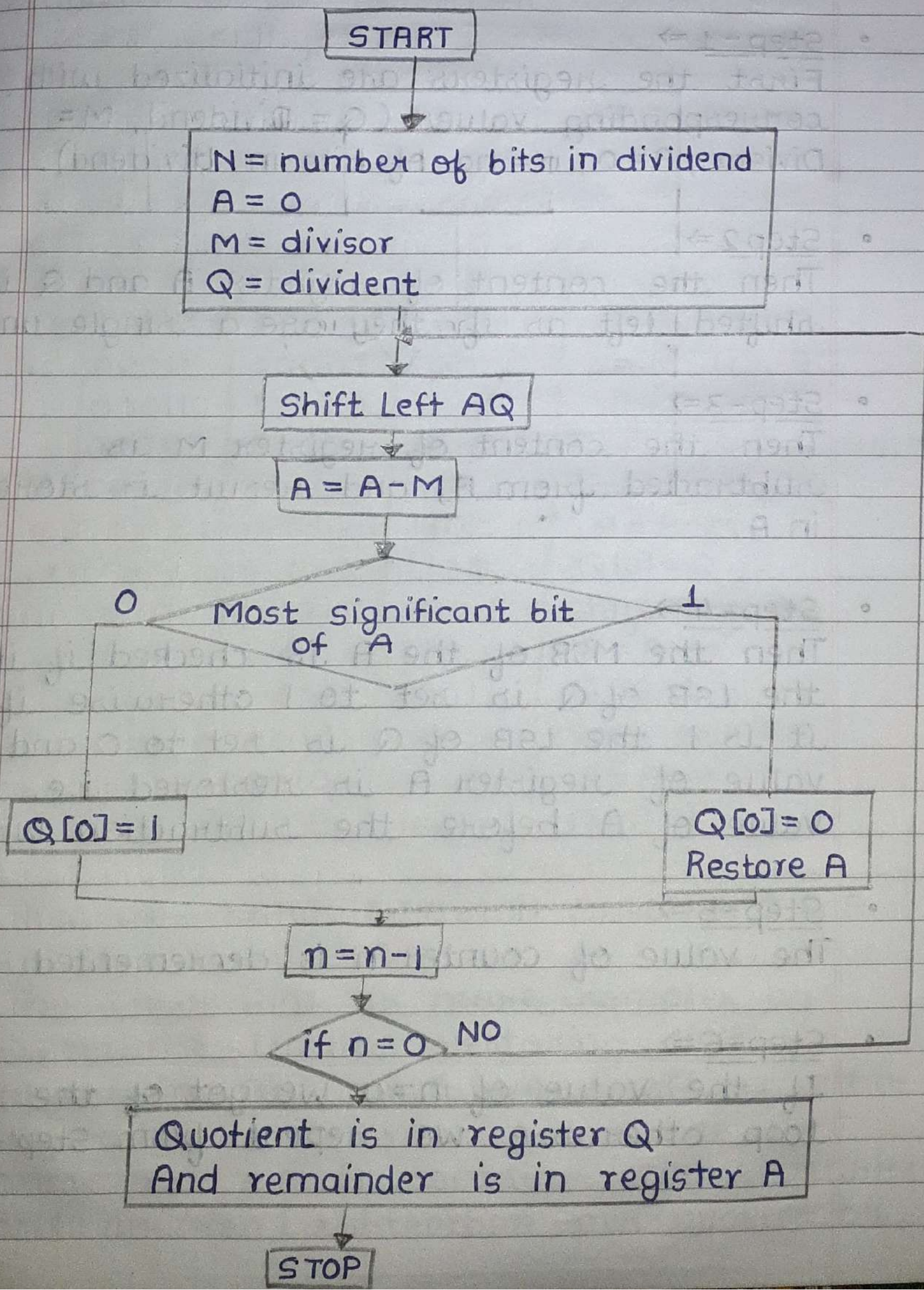
will be performing restoring algorithm for unsigned integer. Restoring term is due to fact that value of register A is restored after each iteration.



Here, register Q contain quotient and register A contain remainder. Here n-bit dividend is loaded in Q and divisor is loaded in M. Value of Register is initially kept 0 and this is the register whose value is restored during iteration due to which it is named Restoring.

⇒ Remember to restore the value of A most significant bit of A is 1. As that register Q contain the quotient i.e, 3 and register A contain remainder 2.

- **FLOWCHART =>**


START
N = number of bits in dividend
A = 0
M = divisor
Q = divident
Shift Left AQ
A = A-M
Most significant bit of A
0
1
Q[0] = 1
Q[0] = 0
Restore A
n = n-1
if n = 0
NO
Quotient is in register Q
And remainder is in register A
STOP

- **STEP INVOLVED:-**

- **Step-1 ⇒**
First the registers are initialized with corresponding values (Q = Dividend, M = Divisor, A = 0, n = no. of bits in dividend).

- **Step-2 ⇒**
Then the content of register A and Q is shifted left as if they are a single unit.

- **Step-3 ⇒**
Then the content of register M is subtracted from A, and result is stored in A.

- **Step-4 ⇒**
Then the MSB of the A is checked if it is 0 the LSB of Q is set to 1 otherwise if it is 1 the LSB of Q is set to 0 and value of register A is restored i.e. the value of A before the subtraction with M.

- **Step-5 ⇒**
The value of counter n is decremented.

- **Step-6 ⇒**
If the value of n = 0 we get of the loop otherwise we repeat from step-2.

- **Step-7 ⇒**
Finally, the register Q contain the quotient and A contain remainder.

★ **Example ⇒**
Perform Division Restoring Algorithm.
Dividend = 11
Divisor = 3

⇒

n	M	A	Q	Operation
4	00011	00000	1011	Initialize
	00011	00001	011_	shift left AQ
	00011	11110	011_	A=A-M
	00011	00001	0110	Q[0]=0 AND restore A
3	00011	00101	110_	shift left AQ
	00011	11111	110_	A=A-M
	00011	00010	1100	Q[0]=0
2	00011	00101	100_	shift left AQ
	00011	00010	100_	A=A-M
	00011	00010	1001	Q[0]=1
1	00011	00010	001_	shift left AQ
	00011	00010	001_	A=A-M
	00011	00010	0011	Q[0]=1

- **NON-RESTORING DIVISION ALGORITHM FOR UNSIGNED INTEGER ⇒**

⇒ This algorithm is more complex as compared to the restoring division algorithm. But when we implement this algorithm in hardware, it has an advantage, i.e. it contains only 1 decision and addition / subtraction per quotient bit.

After performing the subtraction operation, there will not be any restoring steps. Due to this, the numbers of operations basically cutdown up to half. Because of the less operation, the execution of this algorithm basically fast. It performs simple operations such as addition, subtraction. In this we will use the sign bit of register A. 0 is the starting value / bit of register A.

- STEP INVOLVED =>

1. STEP-1 => The corresponding value will be initialized to the registers, i.e register A=0, register M will contain Divisor, register Q will contain Dividend, and N is used to specify the no. of bits in dividend.

2. STEP-2 => In this step, we will check the sign bit of A.

3. STEP-3 => In this bit of register A is 1, then shift the value of AQ left, and perform A = A+M. If this bit is 0; then shift value of AQ into left and perform A = A+-M. That means in case of 0, the 2's complement of M is added into register A and result is stored into A.

4. STEP-4 ⇒ Now, we will check the sign bit of A again.

5. STEP-5 ⇒ If this bit of register A is 1, then Q[0] will become 0. If this bit is 0, then Q[0] will become 1. Here Q[0] indicates the LSB of Q.

6. STEP-6 ⇒ After that the value of N will be decremented. Here N is used as a counter.

7. STEP-7 ⇒

If the value of N=0, then we will go to the next step. otherwise, we have to go step 2 again.

8. STEP-8 ⇒

A = A+M, if sign bit of register A is 1.

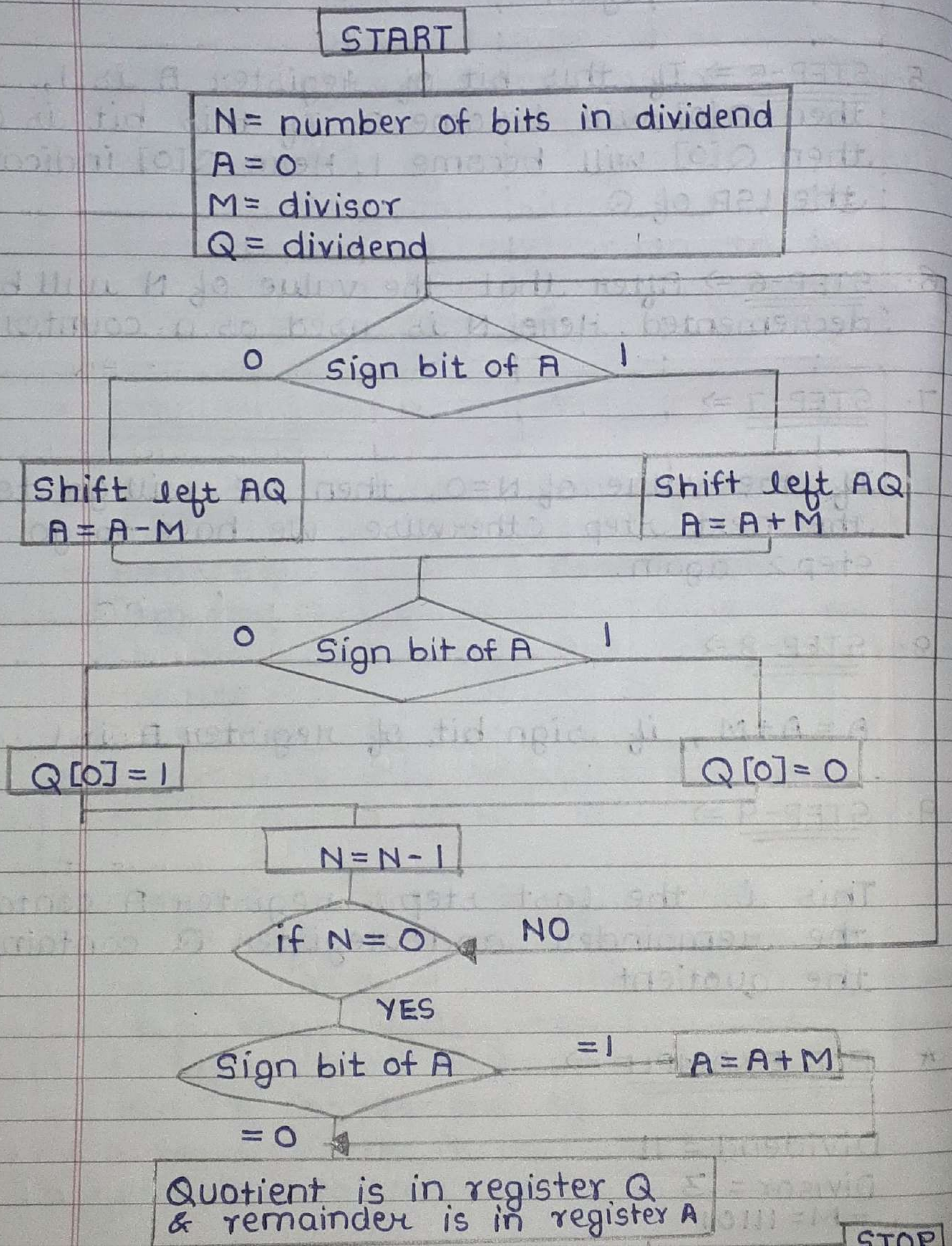
9. STEP-9 ⇒

This is the last step, register A contain the remainder and register Q contains the quotient.

* FOR EXAMPLE ⇒

Dividend = 11
Divisor = 3
−M = 11101

- **FLOWCHART =>**


START
N= number of bits in dividend
A = 0
M = divisor
Q = dividend
Sign bit of A
0
1
Shift left AQ
A = A-M
Shift left AQ
A = A+M
Sign bit of A
0
1
Q[0] = 1
Q[0] = 0
N = N - 1
if N = 0
NO
YES
Sign bit of A
=1
A = A+M
=0
Quotient is in register Q
& remainder is in register A
STOP

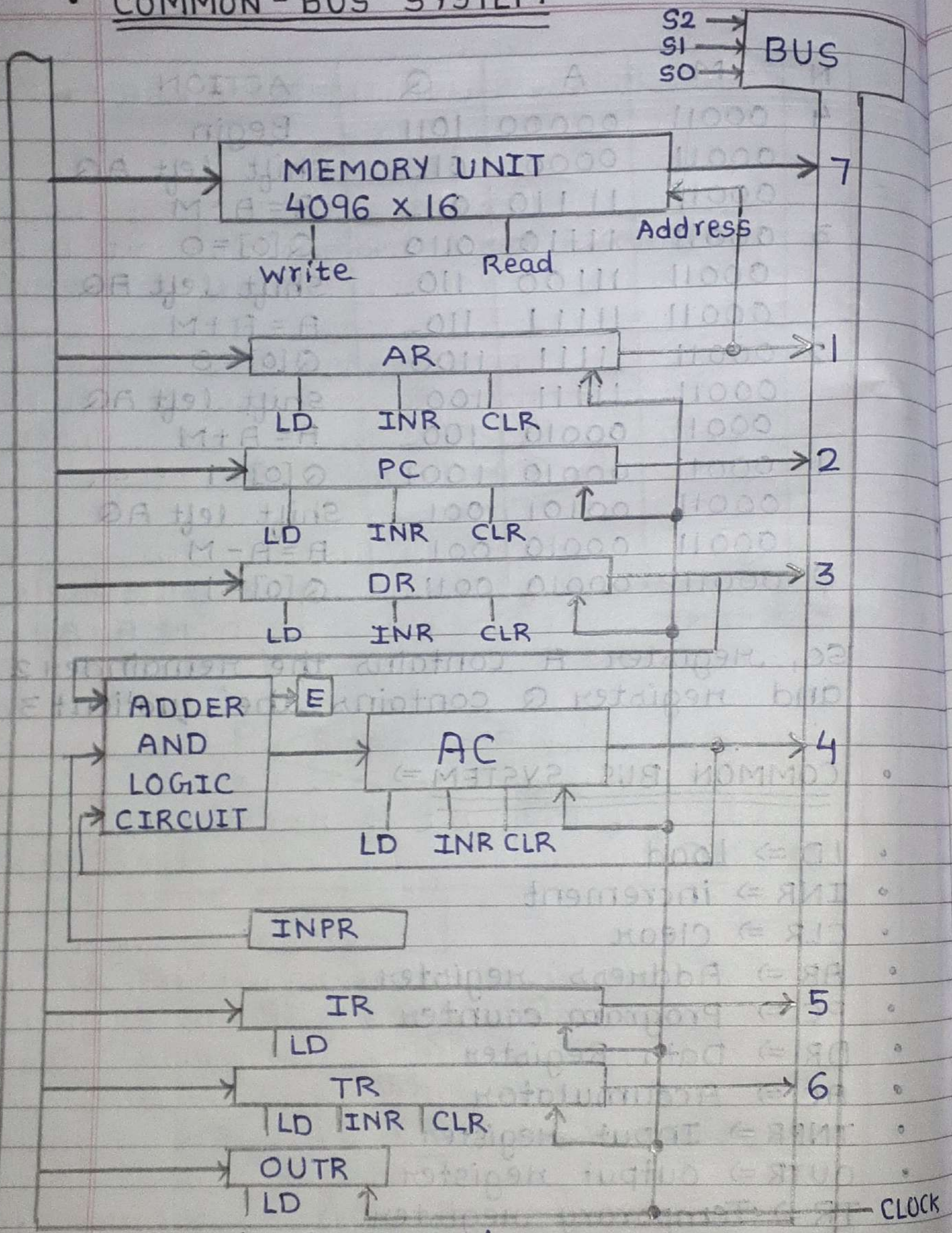
N	M	A	Q	ACTION
4	00011	00000	1011	Begin
	00011	00001	011_	shift left AQ
	00011	11110	011_	A = A - M
3	00011	11110	0110	Q[0] = 0
	00011	11100	110_	Shift left AQ
	00011	11111	110_	A = A + M
2	00011	11111	1100	Q[0] = 0
	00011	11111	100_	shift left AQ
	00011	00010	100_	A = A + M
1	00011	00010	1001	Q[0] = 1
	00011	00101	001_	Shift left AQ
	00011	00010	001_	A = A - M
0	00011	00010	0011	Q[0] = 1

So, register A contains the remainder 2, and register Q contains the quotient 3.

- **COMMON BUS SYSTEM =>**

- LD => Load
- INR => increment
- CLR => Clear
- AR => Address register
- PC => Program counter
- DR => Data Register.
- AC => Accumulator
- INPR => Input register.
- OUTR => output register.
- TR => Temporary register
- IR => Instructions register.

• COMMON-BUS SYSTEM



• INSTRUCTION CYCLE ⇒

A program residing in the memory unit of the computer consists of a sequence of instructions. In the basic computer each instruction cycle consists of four phases.

1. Fetch an instruction from memory.
2. Decode instruction
3. Read the effective address.
4. Execute the instruction.

For the fetch and decode phases can be specified by the following register transfer statements.

T_0 : AR ← PC

T_1 : IR ← M[AR],

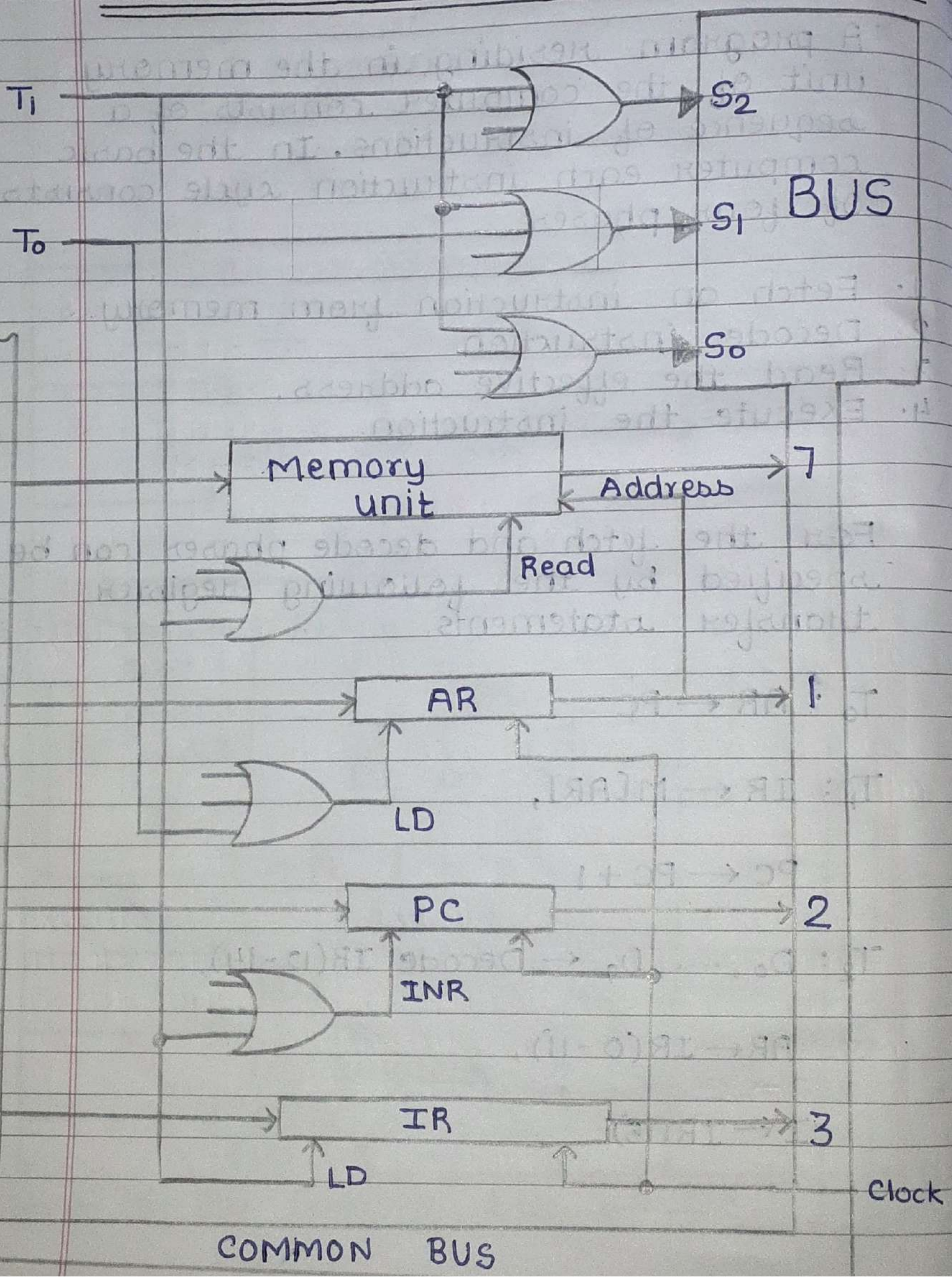
PC ← PC + 1

T_2 : $D_0, \cdots, D_7$ ← Decode IR(12-14),

AR ← IR(0-11),

I ← IR(15)

- ## REGISTER TRANSFERS FOR THE FETCH PHASE



⇒ Figure shows how the first two register transfer are implemented in the bus system. To provide the data path for the transfer of PC to AR, we must apply timing signal T_0 to achieve following connection.

1. Place the content of PC onto the bus by making the bus selection inputs S_2, S_1, S_0 equal to 010.
2. Transfer the content of the bus to AR by enabling the LD input of AR.

- The transfer from PC to AR, since $T_0 = 1$. In order to implement the second statement.

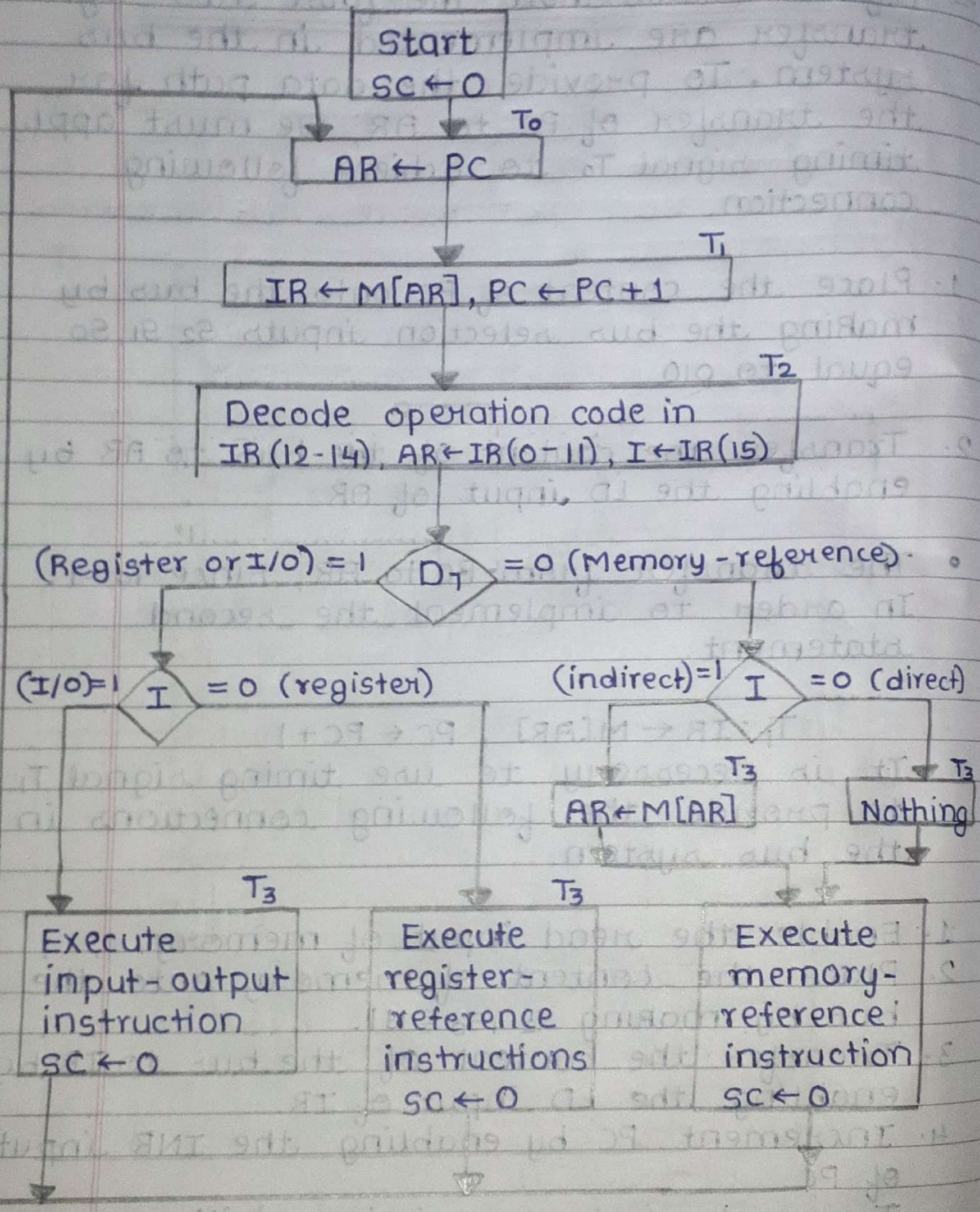
$$T_1 : IR \leftarrow M[AR],\ PC \leftarrow PC + 1$$

It is necessary to use timing signal T_1 to provide the following connections in the bus system.

1. Enable the read input of memory.
2. Place the content of memory onto the bus by making $S_2S_1S_0 = 111$.
3. Transfer the content of the bus to IR by enabling the LD input of IR.
4. Increment PC by enabling the INR input of PC.

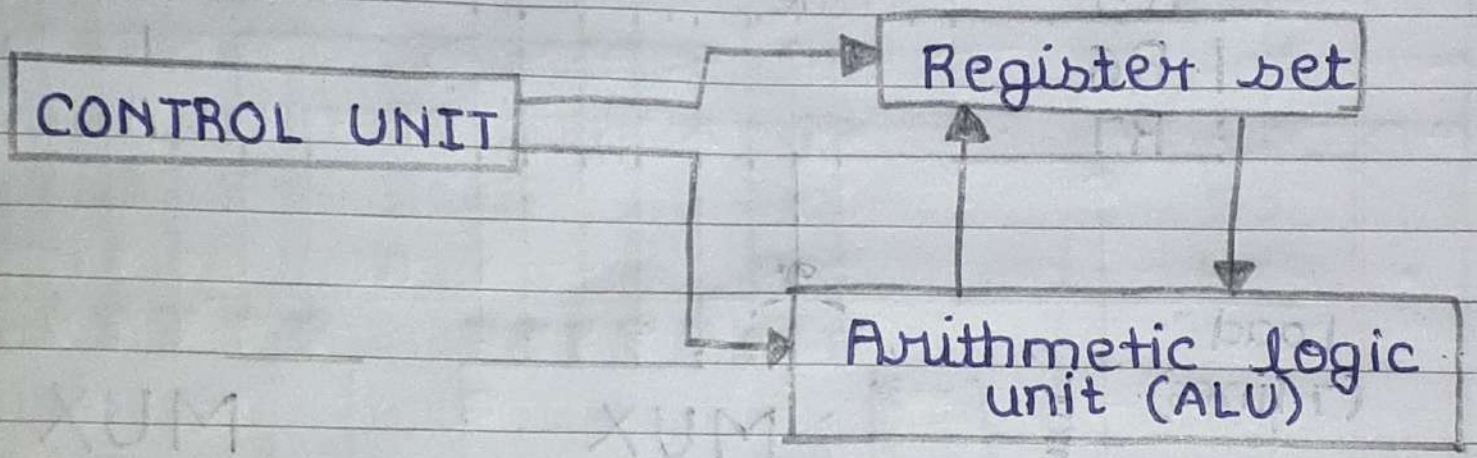
The next clock transistion initiates the read & increment operations since $T_1 = 1$.

- ## FLOWCHART FOR INSTRUCTION CYCLE



$D_7'IT_3$: $AR \leftarrow M[AR]$
$D_7'I'T_3$: Nothing
$D_7I'T_3$: Execute a register-reference instruction
D_7IT_3 : Execute an Input-output instruction.

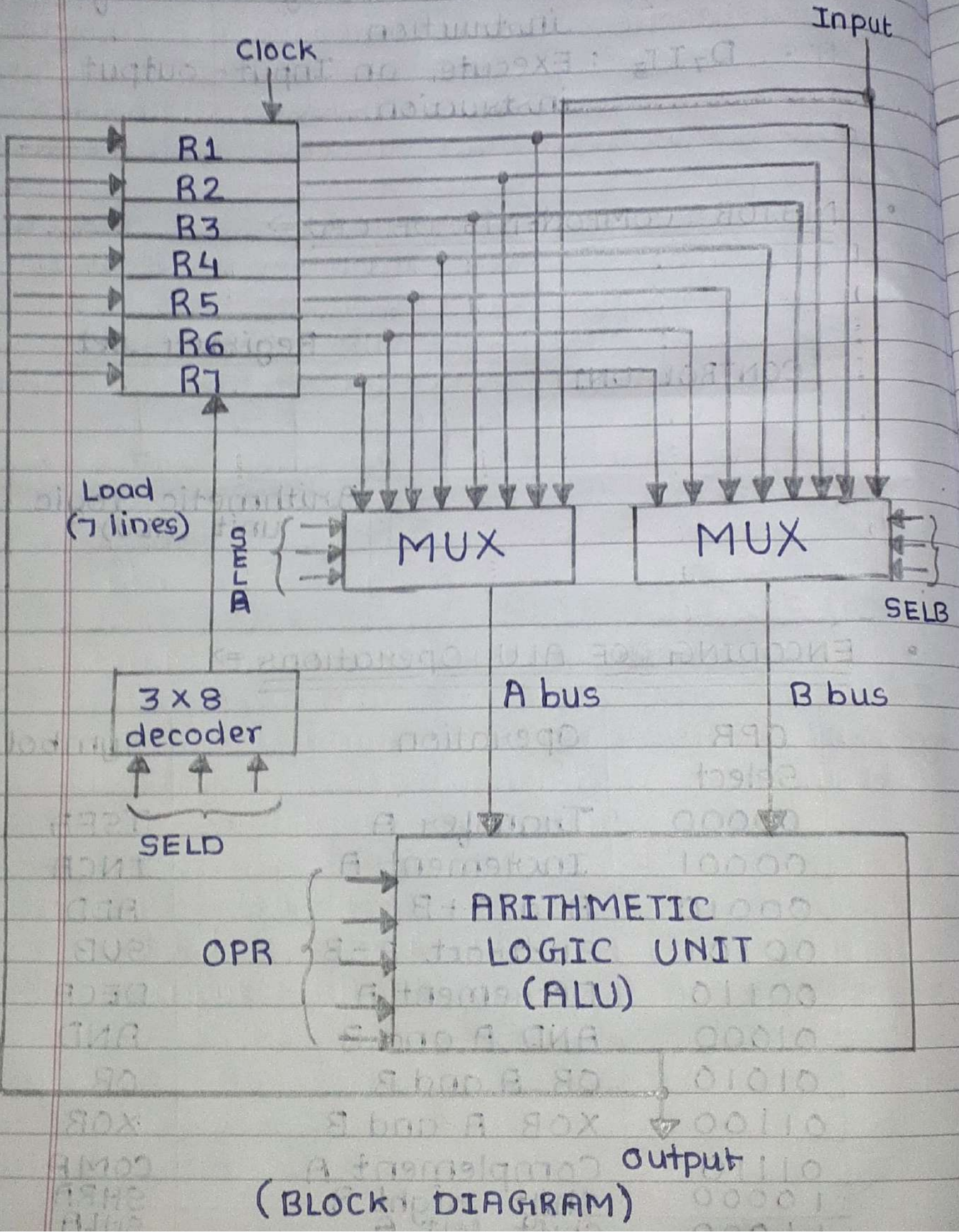
- **MAJOR COMPONENTS OF CPU =>**



- **ENCODING OF ALU Operations =>**

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

- REGISTER SET WITH COMMON ALU.



(BLOCK DIAGRAM)

3	3	3	5
SELA	SELB	SELD	OPR

CONTROL WORD

- ENCODING OF REGISTERS SELECTION FIELDS

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

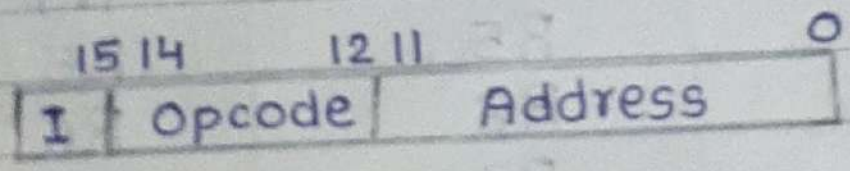
⇒ To perform, operation → R1 ← R2 + R3 the control must provide binary selection variables to the following selector inputs

1. MUX A selector (SELA): To place the content of R2 into bus A.
2. MUX B selector (SELB): To place the content of R3 into bus B.
3. ALU operation selector (OPR): to provide arithmetic addition A+B.
4. Decoder destination selector (SELD): to transfer the content of the output bus into R1.

- Examples of Micro-operations for CPU

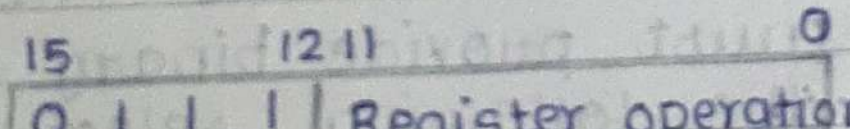
Micro-operations	Symbolic Designation SELA	SELB	SELD	OPR	CONTROL WORD
R1 ← R2 - R3	R2	R3	R1	SUB	010 011 001 00101
R4 ← R4 ∨ R5	R4	R5	R4	OR	100 101 100 01010
R6 ← R6 + 1	R6	—	R6	INCA	110 000 110 00001
R7 ← R1	R1	—	R7	TSFA	001 000 111 00000
Output ← R2	R2	—	None	TSFA	010 000 000 00000
Output ← input	Input	—	None	TSFA	000 000 000 00000
R4 ← shl R4	R4	—	R4	SHLA	100 000 100 11000
R5 ← 0	R5	R5	R5	XOR	101 101 101 01100

- Basic computer instruction formats



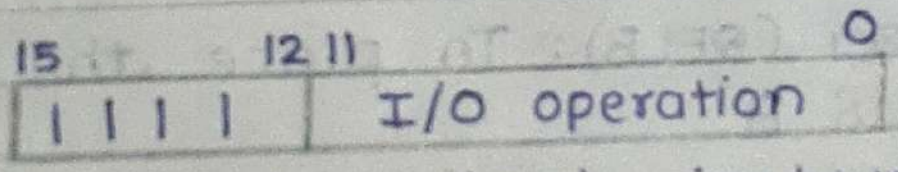
(opcode = 000 through 110)

(a) Memory - reference instruction.



(opcode = 111, I = 0)

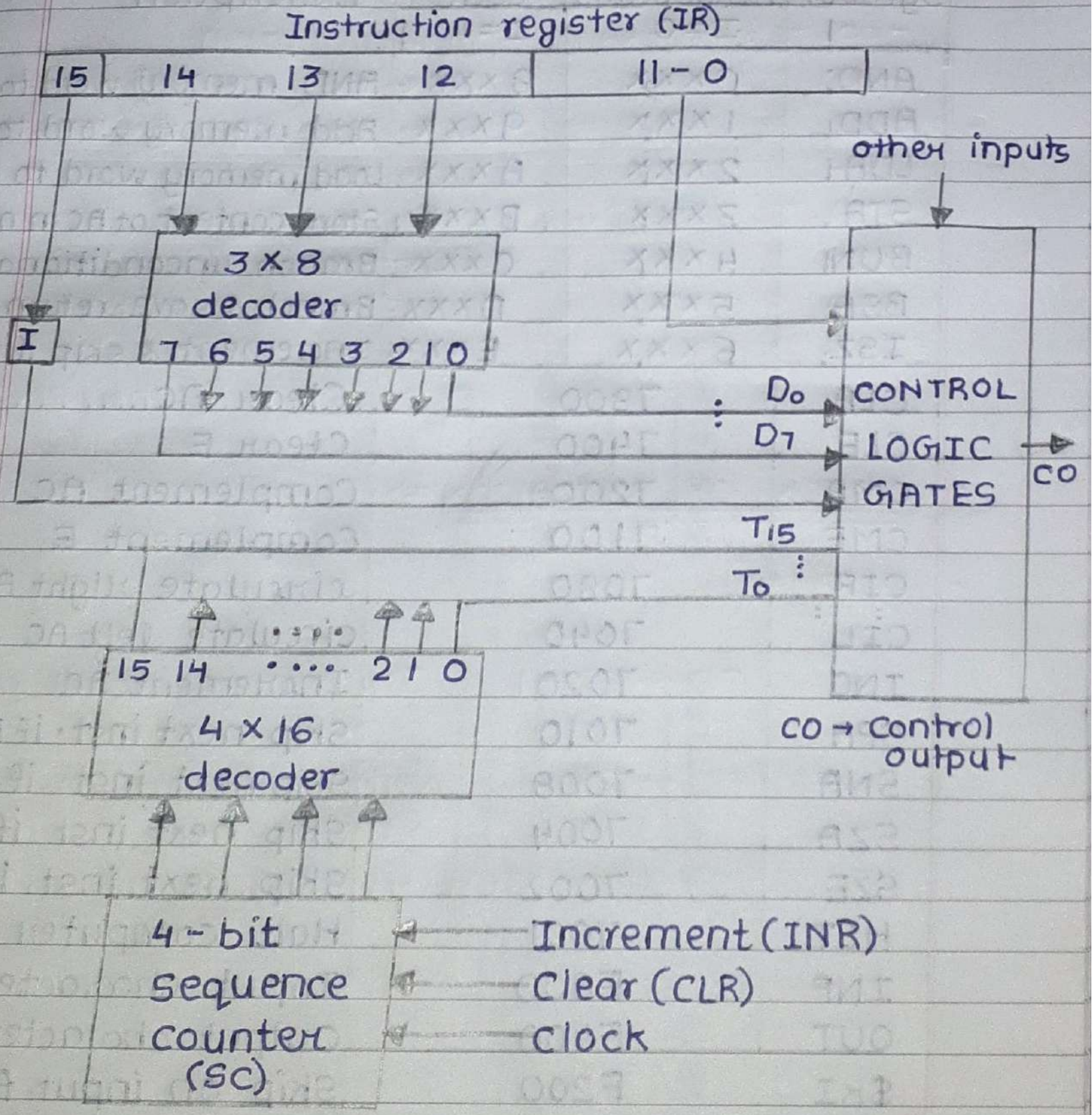
(b) Register - reference instruction



(opcode = 111, I = 1)

(C) Input - output instruction

- **CONTROL UNIT OF BASIC COMPUTER**



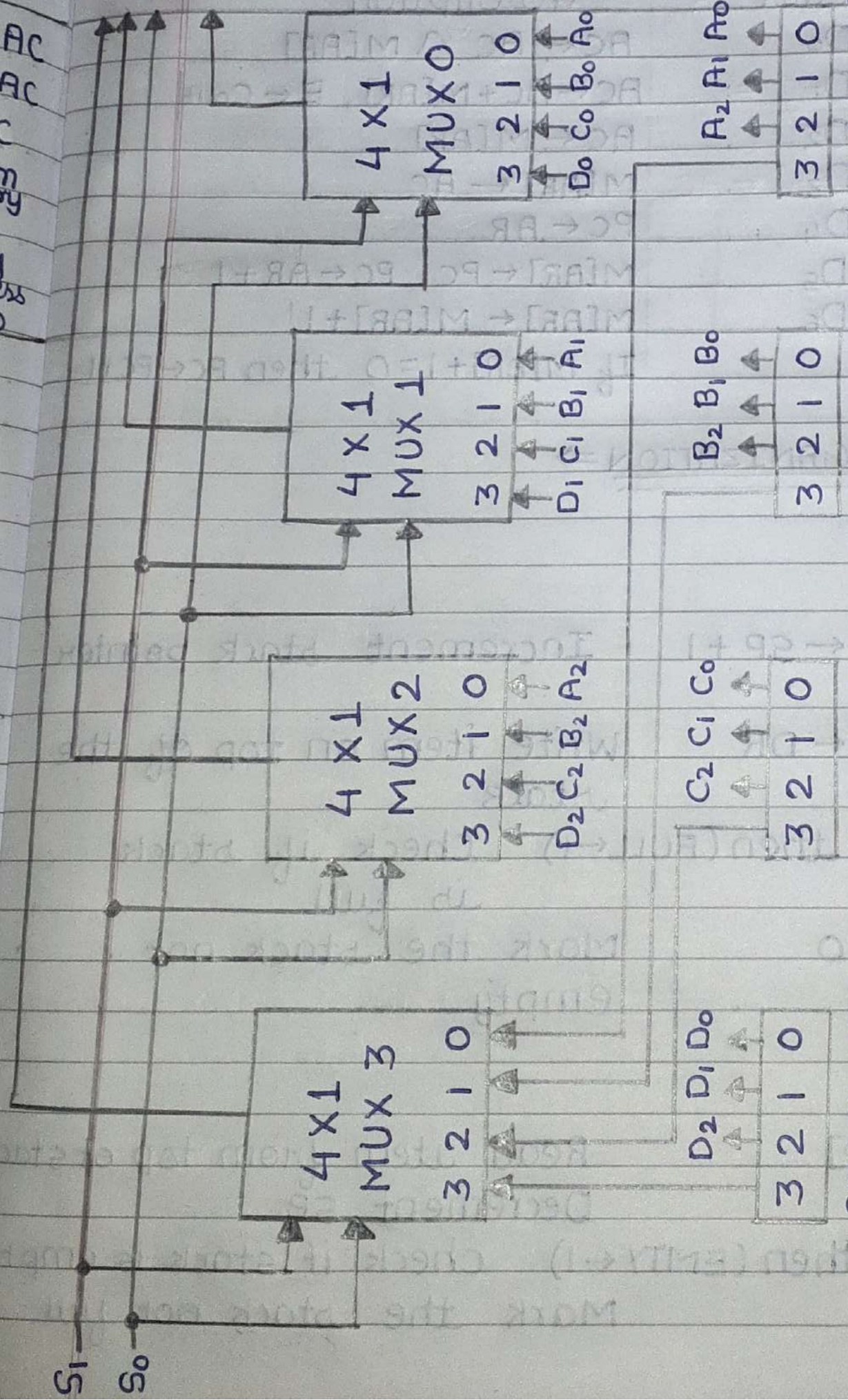
Date 10/9/24
Page 42

- **BASIC COMPUTER INSTRUCTIONS =>**

Symbol	Hexadecimal code I=0	I=1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch & save return address
ISZ	6xxx	Exxx	Increment & skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Complement E
CIR	7080		circulate right AC & E
CIL	7040		circulate left AC & E
INC	7020		Increment AC
SPA	7010		Skip next inst. if AC (+)
SNA	7008		Skip next inst. if AC (-)
SZA	7004		Skip next inst. if AC (0)
SZE	7002		Skip next inst. if E(0)
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

4 line common BUS

BUS SYSTEM FOR FOUR REGISTERS =>



- **MEMORY - REFERENCE INSTRUCTIONS =>**

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

- **STACK ORGANIZATION =>**

- **PUSH =>**

SP ← SP + 1 Increment stack pointer

M[SP] ← DR Write item on top of the stack

IF (SP = 0) then (FULL ← 1) Check if stack is full

EMTY ← 0 Mark the stack not empty

- **POP =>**

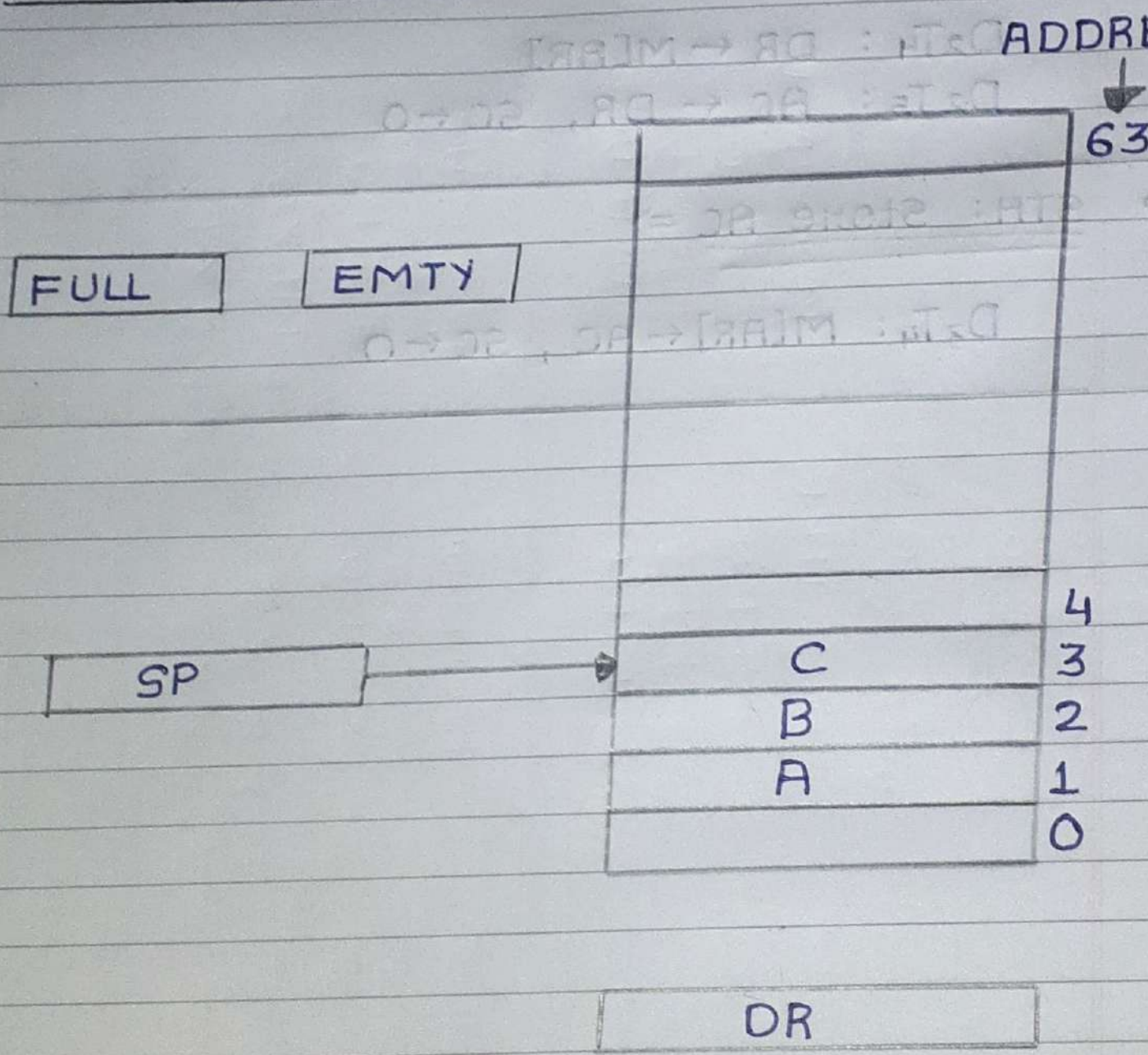
DR ← M[SP] Read item from top of stack

SP ← SP - 1 Decrement SP

IF (SP = 0) then (EMTY ← 1) check if stack is empty

FULL ← 0 Mark the stack not full

- **BLOCK DIAGRAM OF A 64-WORD STACK =>**



- **AND to AC =>**

D_0T_4 : $DR \leftarrow M[AR]$
D_0T_5 : $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

- **ADD to AC =>**

D_1T_4 : $DR \leftarrow M[AR]$
D_1T_5 : $AC \leftarrow AC + DR$, $E \leftarrow Cout$, $SC \leftarrow 0$

- **LDA : Load to AC ⇒**

D_2T_4 : $DR \leftarrow M[AR]$

D_2T_5 : $AC \leftarrow DR$, $SC \leftarrow 0$

- **STA : Store AC ⇒**

D_3T_4 : $M[AR] \leftarrow AC$, $SC \leftarrow 0$