"CA + Computer Organization + Computer Design. 26 Oct

MODULE-I

BY DEEPAK GAUR

Architecture:- "Computer Architecture is concerned with the structure and behaviour of various functional modules of the computer and how they interact to provide the processing needs of the user".

Book: Computer System Architecture, Morris Mano PHI

Computer Organization:- "Computer Organization is concerned with the way the hardware components are connected together to form a computer system."

Computer Design:- "Computer design is concerned with the development of the hardware for the computer taking into consideration a given set of specifications."

Course Contents:-

1) Register Transfer Languages.
2) Basic Computer Organization & Design.
3) Central Processing Unit
4) Memory & Infrasystem communication & I/O Organization.
5) Pipelining, Vector Processing & Multiprocessor.

* The computer architecture is concerned with the digital system.

* Digital System:- Digital System is an interconnection of digital hardware modules that accomplish a specific information task. Digital System are best defined by

the registers they contain & the operations that are performed on the data stored in them.

* Microoperation :→ "The operations that are performed on the data stored in registers are called microoperation. or "A microoperation is elementry operation performed on the information stored in one or more registers.

Examples of Microoperations are :-

① Shift, count, clear and load.

* A counter with parallel load is capable of performing the microoperations increament & load.

* A bidirectional shift register is capable of performing the shift right and shift left microoperations.

* Register Transfer Language :→

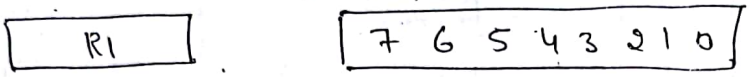
" Symbolic notation describe the microoperations transfers among register is called the register transfer language.

The term register transfer implies the availability of hardware logic circuits ~~and~~ that can perform a stated microoperation and transfer the result of microoperation to same or another registers.

Information transfer from one register to another in symbolic form by means of a replacement operator.

$R2 \leftarrow R1$ denotes transfer of contents of register R1 into register R2.

* Representation of Register in block diagram:→



Register R Individual Bits.

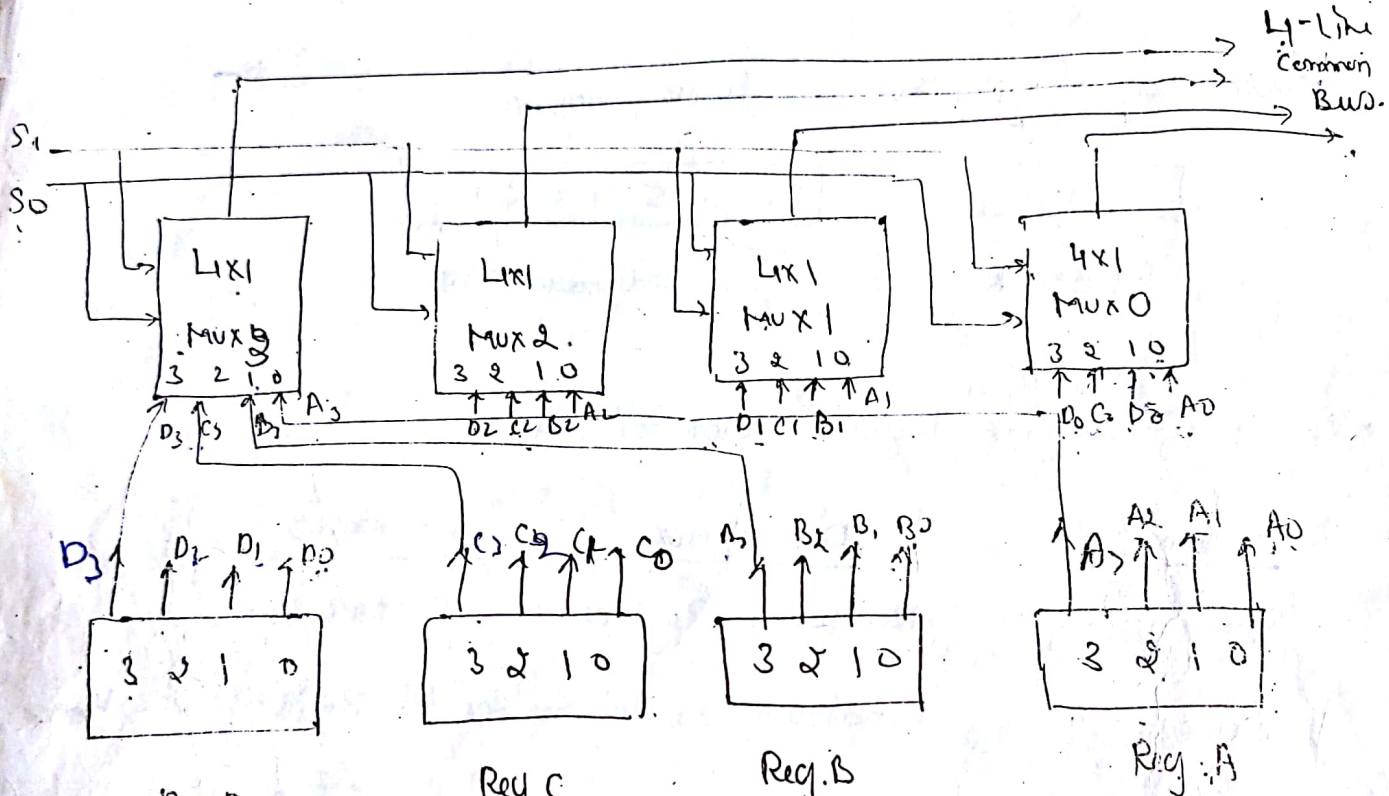
* Basic Symbols for Register Transfer :→

Symbols:	Description	Examples.
Letters	Denotes a Register.	MAR, R2
Parantheis ()	Denotes a part of register.	R2(0-7), R2(L)
Arrows (←)	Denotes transfer of Into	R2 ← R1
Comma,	Seperates two microoperations	R2 ← R1, R1 ← R2

Bus & Memory Transfers :→

* Common Bus System. "A bus structure consists of a set of common lines, one for each bit of register, through which binary information is transferred at a time". Control register determines which register is selected by bus during each particular register transfer.

One way of constructing a common bus system is with multiplexers. The multiplexers selects the source register whose binary information is then placed on the bus. Construction of a bus system for four registers is shown as:-



Function Table for Bus:-

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

Memory Transfer :-

"A memory unit is a collection of Storage cells together with associated circuits needed to transfer information in and out of storage". Memory stores ~~binary~~ binary information in groups of bits called "words".

Types of memories used in computer :-

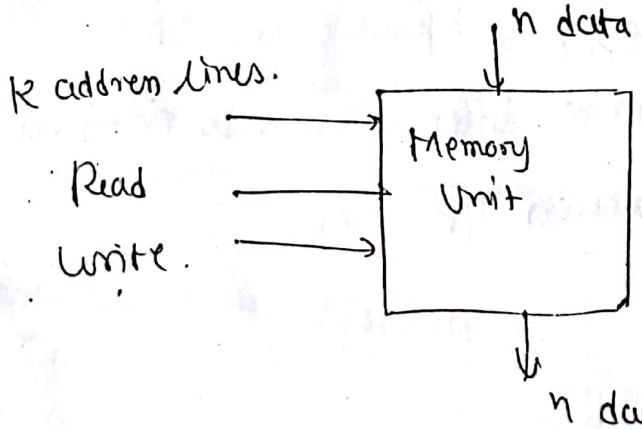
① RANDOM ACCESS MEMORY (RAM) :-

In random access memory the memory cells can be accessed for information transfer from any desired memory location. i.e. the process of locating a word in memory is the same and requires an equal amount of time no matter where the cells are located physically in memory, thus the name "Random Access".

* Operations on RAM :-

(A) Write And (B) Read.

Write Signal Specify a transfer-in operation and write specify the transfer out signal.



* Steps that must be taken for the purpose of transferring a new word to be stored into memory are as follows:-

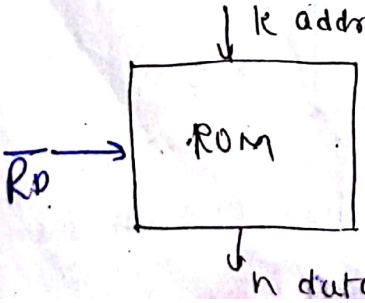
1) Apply Binary address of desired word into address lines.
2) Apply data bits that must be stored in memory into the data input lines.
3) Activate the write input.

* Steps that must be taken for purpose of transferring a stored word out of memory are as follows:-

(1) Apply binary address of desired word into the address line.
(2) Activate the read input.

(9) READ ONLY MEMORY:

ROM is a memory unit that performs the read operation only, it does not have write capability. This means binary information stored in ROM is made permanent during hardware production of unit & can not be altered by writting different words into it.



Arithmetic Microoperations :→

"A microoperation is an elementary operation performed with data stored in registers." Following are four categories of microoperations :-

1. Register Transfer microoperations transfer binary info. from one register to another.
2. Arithmetic microoperation perform arithmetic operations on numeric data stored in registers.
3. Logic microoperation perform bit manipulation operations on non-numeric data stored in registers.
4. Shift microoperations perform shift operations on data stored in registers.

* Difference B/W Register Transfer microoperation & other three microoperations :-

" Register Transfer microoperation does not change the information content when the binary info. moves from source register to destination register. But other three types of microoperation changes the information content during the transfer.

Basic Arithmetic Operations :→ Basic arithmetic microoperations.

are, as follows:-

① addition ② Subtraction ③ Increment ④ Decrement and ⑤ Shift.

① Addition :-

$R3 \leftarrow R1 + R2$

States the addition microoperations. It specify that contents of register R1 and R2 are added and then result is stored in register R3.

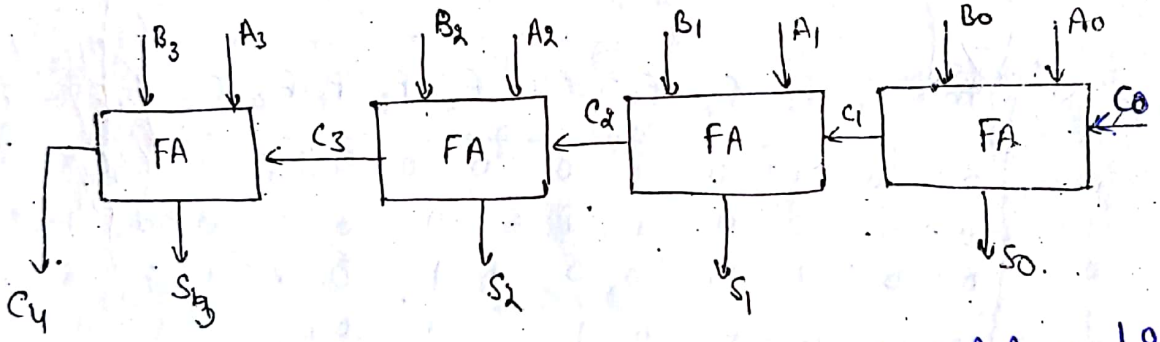
Table of Arithmetic micro operations:-

Symbolic Representation	Description
$R3 \leftarrow R1+R2$	content of R1 plus R2 transfer to R3.
$R3 \leftarrow R1-R2$	" " R1 minus R2 " " R3.
$R2 \leftarrow \overline{R2}$	complement content of R2 (1's complement)
$R2 \leftarrow \overline{R2}+1$	2's complement the contents of R2. (negat
$R3 \leftarrow R1+(\overline{R2}+1)$	R1 plus the 2's complement of R2 (Subtracti
$R1 \leftarrow R1+1$	Increment Content of R1 by 1.
$R1 \leftarrow R1+1$	Decrement " " R1 by 1.

Binary Adder :→

To implement the add micro operations with hardware, we needed the registers that hold the data and digital ~~computers~~ component that performs the arithmetic addition. The digital circuit that perform the arithmetic sum of two ~~bits etc~~ binary numbers of any length is called a binary adder. The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full adder.

Binary adder is constructed with full adder circuits connected in cascade, with the output carry from one full adder connected to the input carry of the next full adder.



[4-bit Binary Adder]

$A_3A_2A_1A_0 = 1010$

$B_3B_2B_1B_0 = 0101$

Logic MICROOPERATIONS →

Logic microoperation specify binary operations for strings of bits stored in registers. These operations consider each bit of register separately and treat them as binary variables.

P: $R1 \leftarrow R1 \oplus R2 = R1\bar{R}2 + \bar{R}1R2$

* Symbols used for logic microoperations:-

OR operations	Symbols.
OR	$\vee$
AND	$\wedge$
XOR	

Complement, is also logical microoperation.

* List of logic microoperations :-

There are 16 different logic operations that can be performed with two binary variables. These operations can be determined from all possible truth tables obtained with two binary variables as shown:-

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

[Truth table for 16 Functions of Two Variables]

* 16 - Boolean functions of two variables x and y are expressed in algebric form as well as logic microoperations derived from these functions by replacing variable 'x' & 'y' are as follows:-

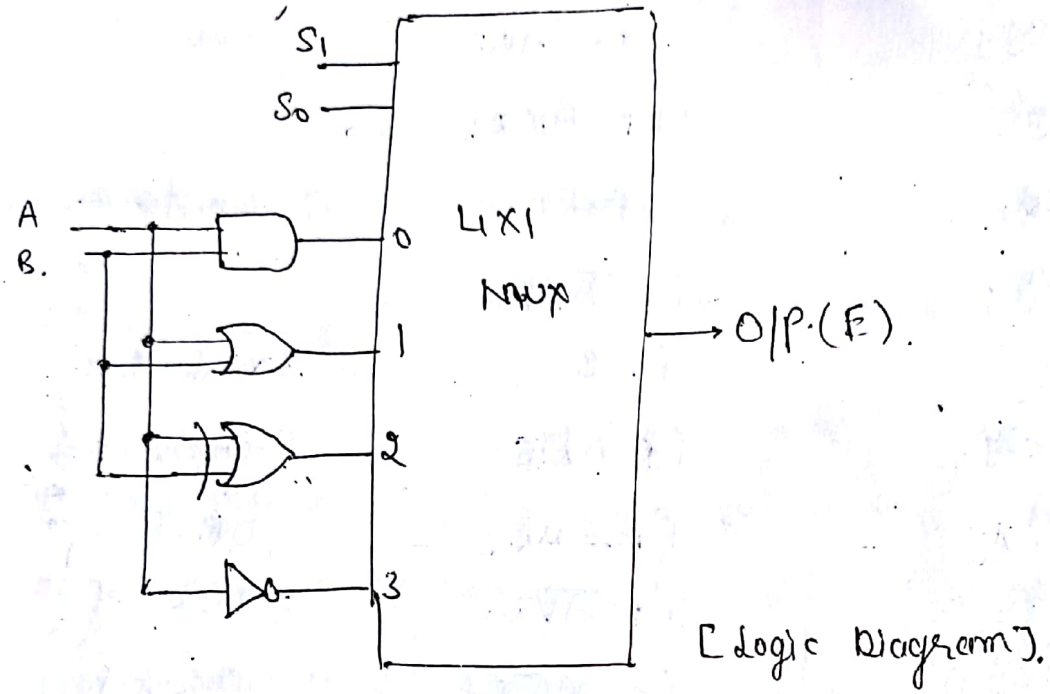
Boolean Function	microoperation	Name.
$F_0 = 0$	$F \leftarrow 0$	Clear
$F_1 = xy$	$F \leftarrow A \wedge B$	AND.
$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
$F_3 = x$	$F \leftarrow A$	Transfer A
$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
$F_5 = y$	$F \leftarrow B$	Transfer B
$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
$F_7 = x + y$	$F \leftarrow A \vee B$	OR
$F_8 = (x+y)'$	$F \leftarrow \overline{A \vee B}$	NOR
$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive NOR
$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B
$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
$F_{12} = x'$	$F \leftarrow \bar{A}$	
$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware Implementation:

Hardware Implementation of logic microoperations require that logic gates be inserted for each bit or pair of bits in the registers to perform the required logic functions.

Following figure shows one stage of

the circuit that generates the four basic logic microoperations. It contains four gates and a multiplexer.



[Logic Diagram].

Table for selecting output :→

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

(Functional Table of above Circuit)

Shift Microoperations:→

Shift microoperation are used for serial transfer of data. They are also used in conjunction with arithmetic, logic and other data processing operations. The content of a register can be shifted to the left or right. ~~At the same time~~ During a shift right operation the serial input transfers a bit into the ~~left~~ right most position. and during shift left operation the serial input transfer a bit into the ~~right~~ left most position.

* There are three types of shifts:→.

(1) logical shift.:→

A logical shift is one that transfer 0 through the serial input.

R1 ← shl R1 → specify 1 bit ~~left~~ shift to left of
R2 ← shr R2 the content of register R1.

(2) circular shift:→

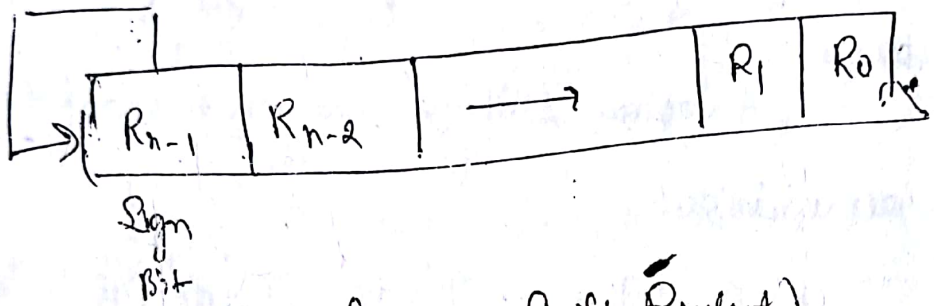
Circular shift circulates the bits of the register around the two ends without loss of information. This is ~~connected~~ accomplished by connecting the serial o/p of shift register to its serial input.

R ← cil R circulate shift left register R
R ← cir R " " Right " R.

e.g. 1110

(3) Arithmetic shift:- An arithmetic shift is a micro operation

that shifts a signed binary number to the left or right. An arithmetic shift left multiplies a signed binary number by 2. An arithmetic shift right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of number remains the same when it is multiplied or divided by 2.



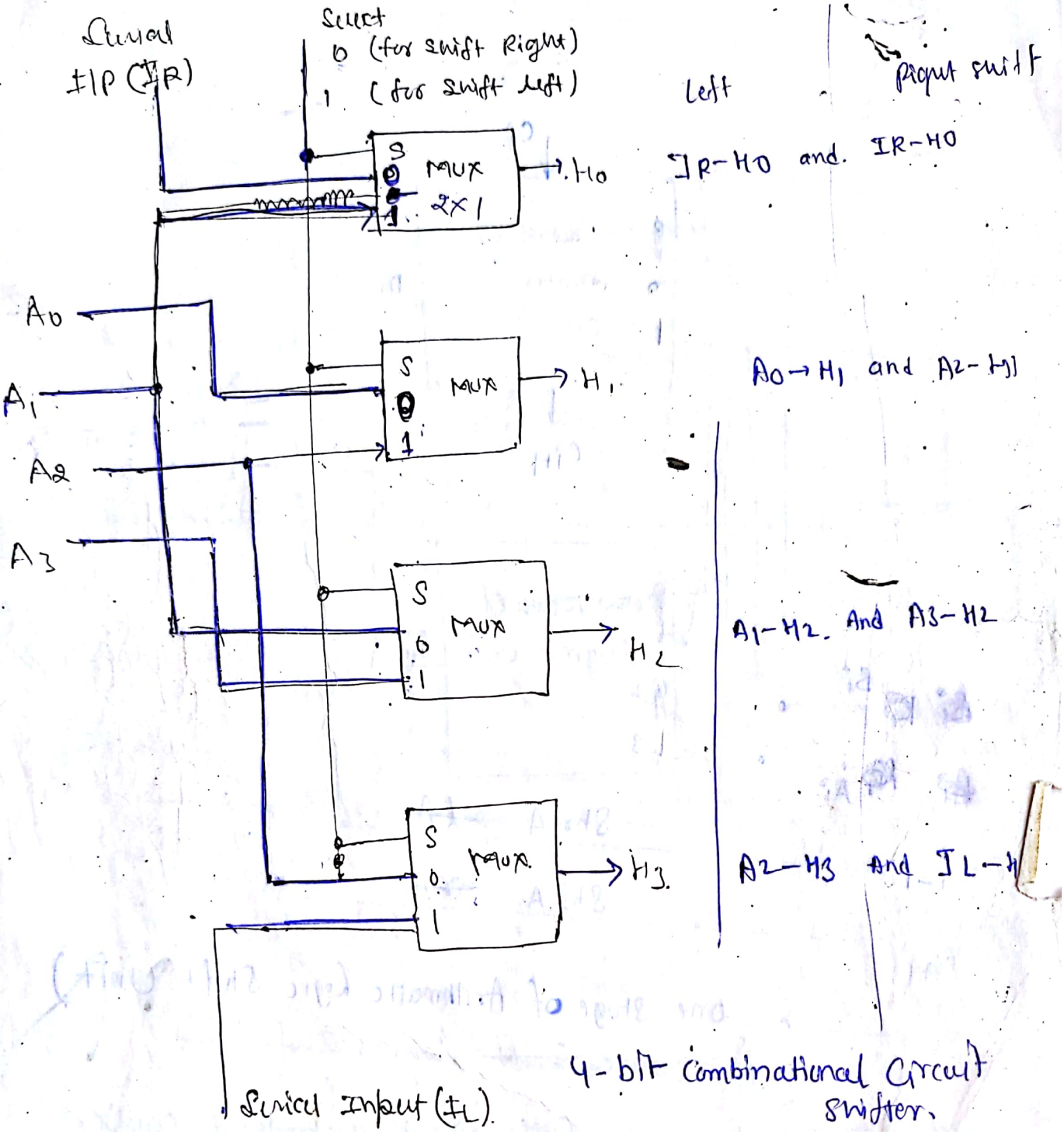
(Arithmetic Shift Right)

Arithmetic shift left inserts 0 into R_0 and shifts all other bits to left. Initial bit of R_{n-1} is lost and replaced by bit from R_{n-2}.

Hardware Implementation →

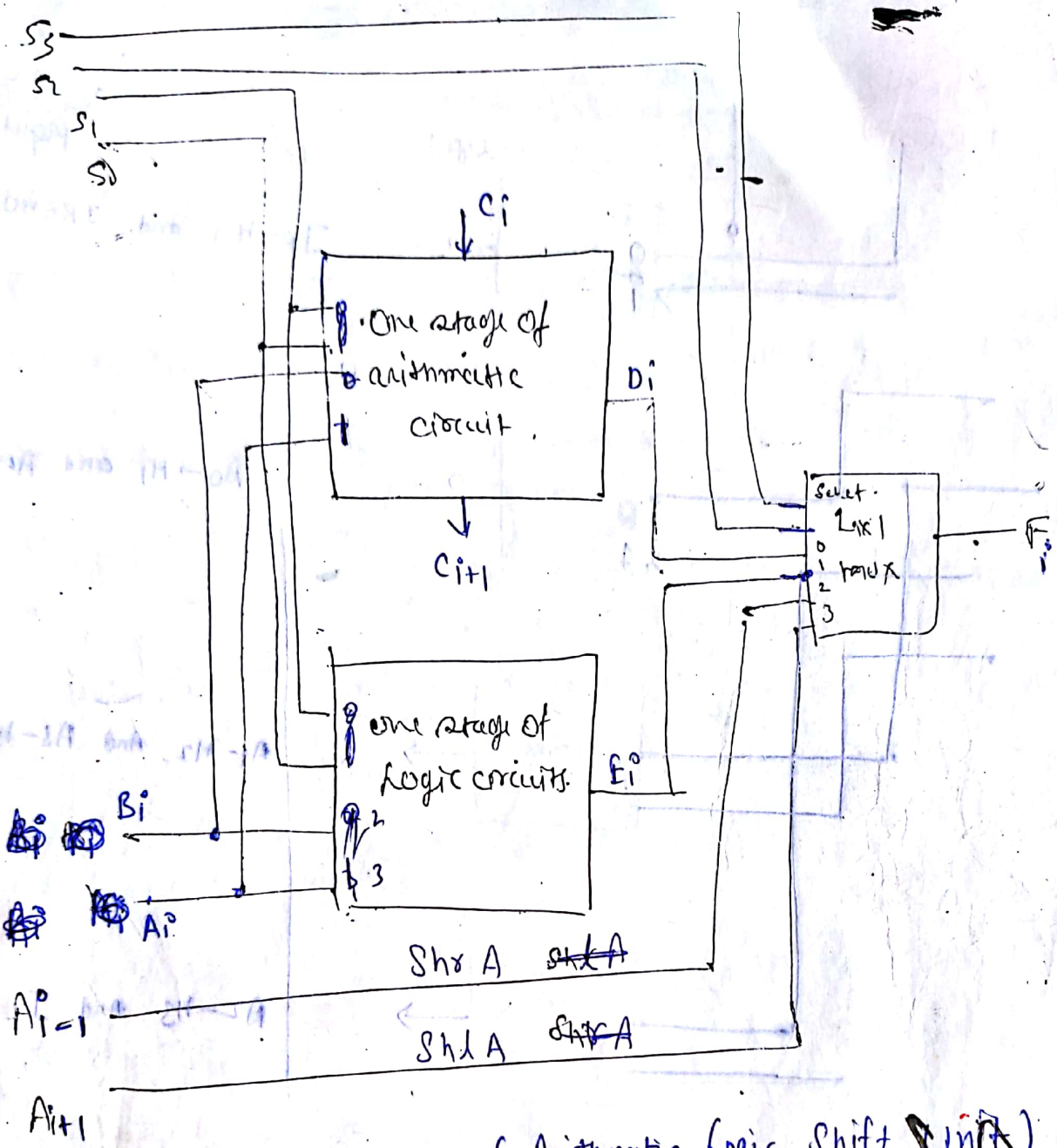
Implementation of shift micro-operations is done by combinational circuit shifter. A combinational circuit shifter can be constructed with multiplexers. The 4-bit shifters has four data inputs, A_0 through A_3 and four data outputs.

H_0 through H_3. There are two serial inputs, one for shift left (I_L) and another for shift right (I_R)



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

* Arithmetic Logic Shift Unit :→



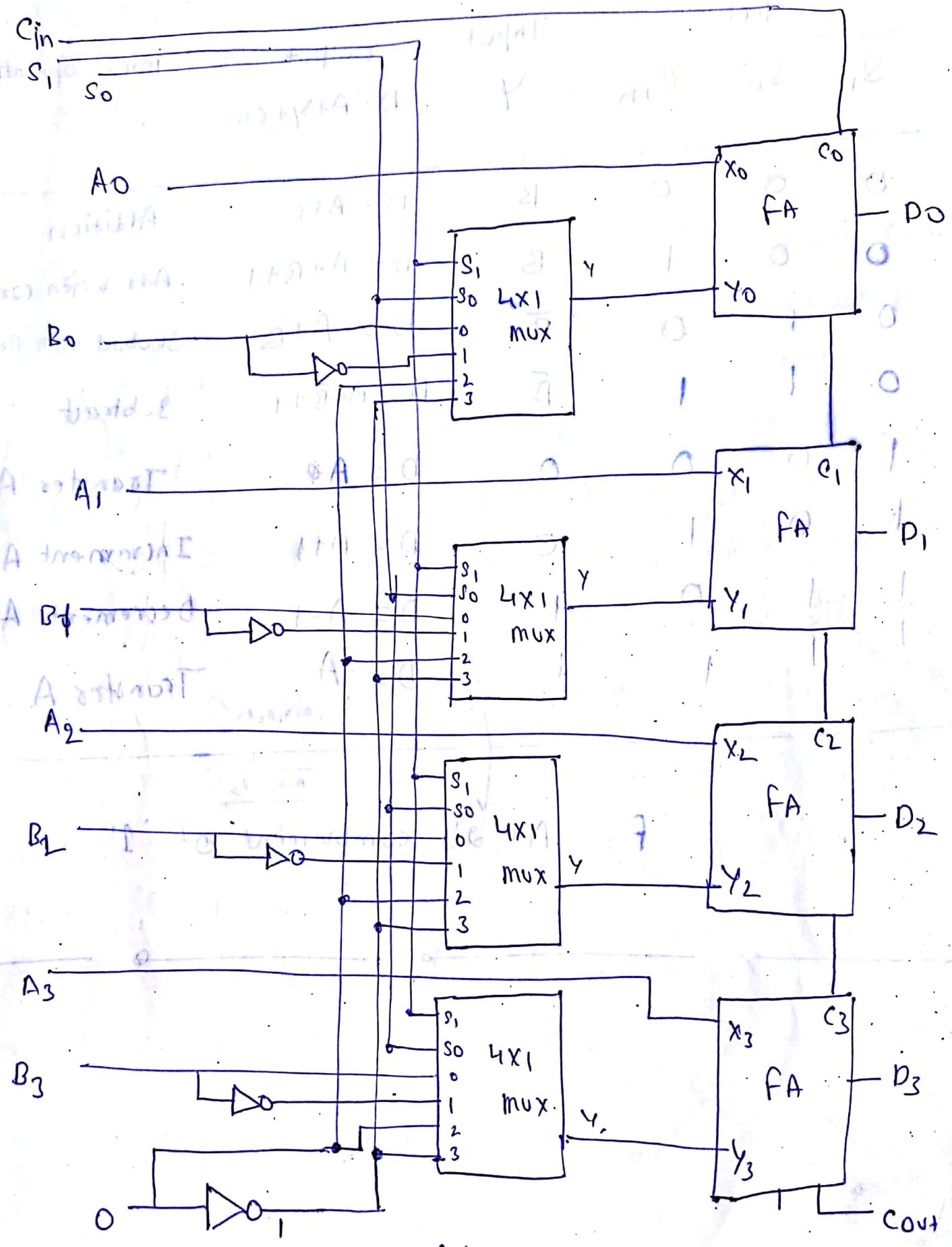
One stage of Arithmetic Logic Shift Unit

(Arithmetic ~~Shift~~ Logic Shift Unit)

Arithmetic, Logic and Shift circuits introduced earlier, can be combined into one ALU with common selection variable(s).

- Draw the 4-bit binary Decrementor circuit diagram.
- Draw block diagram for xy2 : R1 ← R2, R2 ← R1

4-Bit Arithmatic circuit →



[4-Bit Arithmatic circuit]

Arithmatic circuit function table

Select			Input	Output	micro-operation
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	
0	0	0	B	$D = A + B$	Addition
0	0	1	B	$D = A + B + 1$	Add with carry
0	1	0	$\overline{B}$	$D = A + \overline{B}$	Subtract with Borrow
0	1	1	$\overline{B}$	$D = A + \overline{B} + 1$	Subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A.

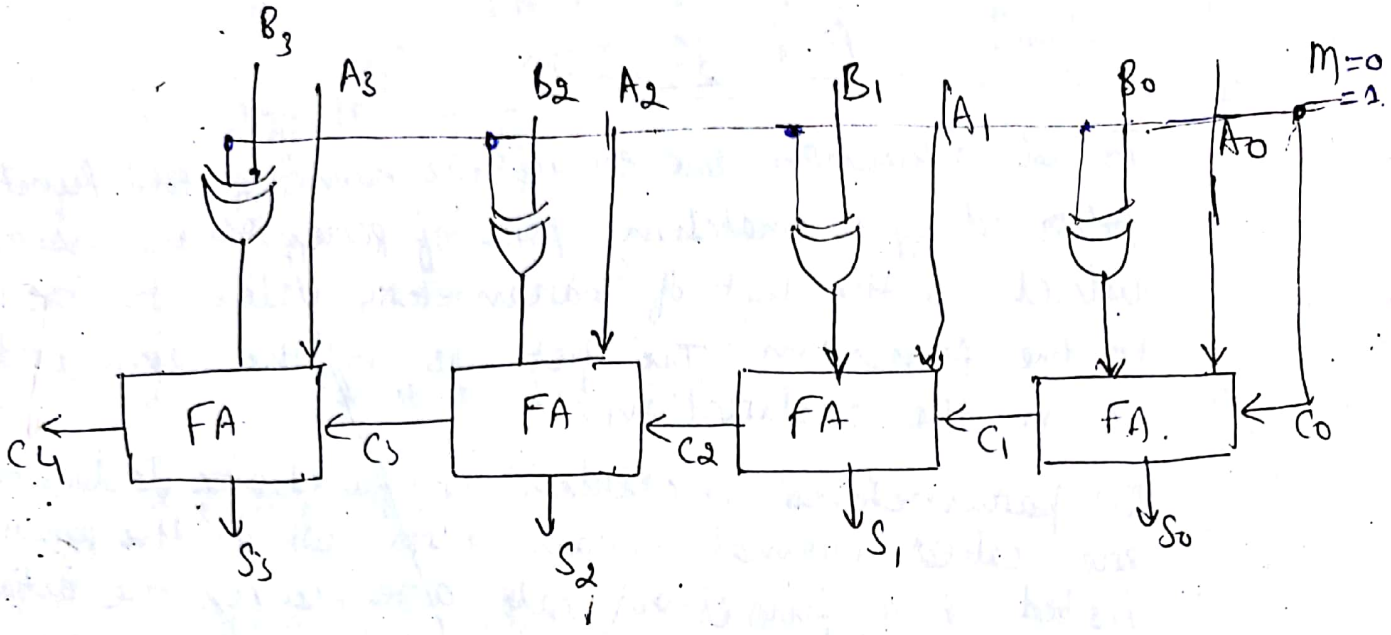
$F = A$ + 2's complement of '1'.

Function table for Arithmatic Logic Shift unit

	Operation Select						
	S_3	S_2	S_1	S_0	Cin	Operation	function
1)	0	0	0	0	0	$F = A+B$	Addition
2)	0	0	0	0	1	$F = A+B+1$	Adder with carry
3)	0	0	0	1	0	$F = A+\overline{B}$	Subtract with borrow
4)	0	0	0	1	1	$F = A+\overline{B}+1$	Subtract
5)	0	0	1	0	0	$F = A$	Transfer 'A'
6)	0	0	1	0	1	$F = A+1$	Increment 'A'
7)	0	0	1	1	0	$F = A-1$	Decrement 'A'
8)	0	0	1	1	1	$F = A$	Transfer 'A'
9)	0	1	0	0	x	$F = A \cap B$	AND
10)	0	1	0	1	x	$F = A \cup B$	OR
11)	0	1	1	0	x	$F = A \oplus B$	XOR
12)	0	1	1	1	x	$F = \overline{A}$	Complement 'A'
13)	1	0	x	x	x	$F = shr\ A$	Shift right 'A'
14)	1	1	x	x	x	$F = shl\ A$	Shift left 'A'

2's complement of binary no. by 4-bit adder-Subtractor.

Subtraction of 2 binary no. by 4-bit Subtractor



4-bit adder-Subtractor.

✓ M=0, circuit is adder,

✓ when M=1, circuit is subtractor.

A = 0101 => A0=1, A1=0, A2=1, A3=0

B = 1010 => B0=0, B1=1, B2=0, B3=1.

B = B = 1010, $\bar{B}$ = 0101 A = 0101 2's

0101
0101
1010

A0 = 1
B0 = 0
S0 = 1 C0 = 1

A1 = 0
B1 = 0
C0 = 1, S0 = 1,

A2 = 1
B2 = 1
C1 = 1, C1 = 1 S1 = 1

A3 = 0
B3 = 0
C2 = 1 = 1 S2 = 1

B ⊕ M
B in. ⊕ $\bar{B}$ in
0 ⊕ $\bar{B}$ = $\bar{B}$

Statement 1-n: is the body of the function. It can be single instructions or a block of instru

CALLING A FUNCTION

To call a function, one enters the name of the functio followed by a matching pair of parenthesis inside. which is the list of parameters values to be pa to the function. The list is in the same orde as in the declaration.

The parameters included in functions declaration are called formal parameters while the parame listed in a function call are called the actual parameters.

The function calling mechanism matches the formal and actual parameters simply by order and pay no attention to the identific names.

for example:

float volume (float, float, float);

the function call statement can be used:

volume (l, b, h);

where l, b, h are float variables.

```
int x=5, y=3, z;
    z=additional (x,y);
```

In this case we call the function adding passing value of x & y; that means 5 and 3 resp, not the variables themselves.

The function addition is declared thus.

int addition (int a, int b);

$A = 1010$
$B = 0101$

$m = 0$, circuit is adder.

$B \oplus m = B\bar{m} + \bar{B}m = B$, $C_0 = 0$.

$A = 1010$
$B = 0101$

$S_0 = 1$, $S_1 = 1$, $S_2 = 1$, $S_3 = 1$.

1010
$2^3, 2^1, 2^0$
$8 + 2 = 10$
$0101 = 5$.

(2) $m = 1$, circuit is subtracter.

$B \oplus m = B\bar{m} + \bar{B}m = \bar{B}$. $C_0 = 1$,

$A = 1010$
$B = 0101$

$S_0 = \bar{B}_0 + C_0 = 0 + 1 = 1$, $C_1 = 0$.
$S_1 = \bar{B}_1 + C_1 = 1 + 0 = 1$,

2's complement $= 1011$.

$$\begin{array}{r} 1010 \\ 1011 \\ \hline +0101 \end{array} = 0101 = 5 \text{ Ans}$$

ADDER — $A_3A_2A_1A_0$

$A = 1010$
$B = 0101$
$B_3B_2B_1B_0$

$A_0 + B_0 = 0 + 1 = 1$.
$A_1 + B_1 = 1 + 0 = 1$.
$A_2 + B_2 = 0 + 1 = 1$

$0\bar{1} + \bar{0}1 = 1$.
$0\bar{0} + 0\bar{0} = 1$

$0 \oplus 0 =$
$0\bar{0} + \bar{0}0$.
$0 + 1 = 1$. $= 0$.

XOR

0 1	0
1 0	0
1 1	1
0 0	1

XOR.

00	$= 01 + 10 = 0$
11	$= 0$
10	$= 1$
01	$= 1$

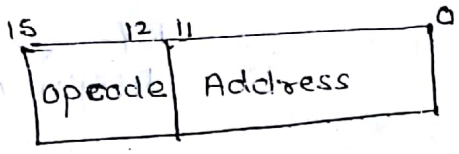
Instruction Codes:→

An instruction code is a group of bits that instruct the computer to perform a specific operation. The most basic part of an instruction code is its operation part. The operation code of an instruction is a group of bits that define such operations like add, substract, multiply & compement. The number of bits required for an operation code is an instruction depends on the total number of operation available in computer.

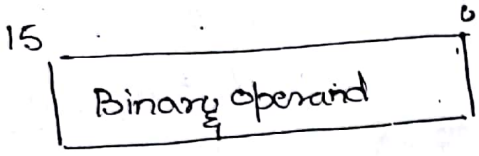
The operation part of an instruction code specifies the operation to be performed. This operation must be performed on some data stored in processor registers or in memory. An instruction code therefore must specify not only the operations but also the registers or the memory where the operands are to be found.

Stored Program Organization:→ The simplest way to organize a computer is to have one processor register and an instruction code format with two parts. the first part specify the operation to be performed and the second specify an address of operand. The memory address tells the control where to find an operand in memory. This operand is read from memory and used as the data to be operated on together with the data stored in the processor register. following figure shows this type of organization.

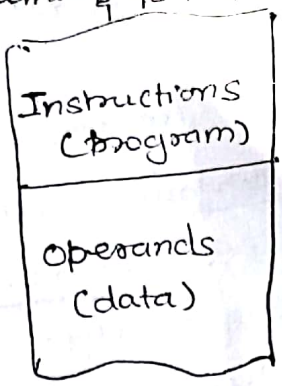
- Processor Register,
- Instruction code format → operation & operand
- memory address tells the computer control where operand in memory.



[Instruction format]



Memory 4096 x 16



[Stored program Organization]

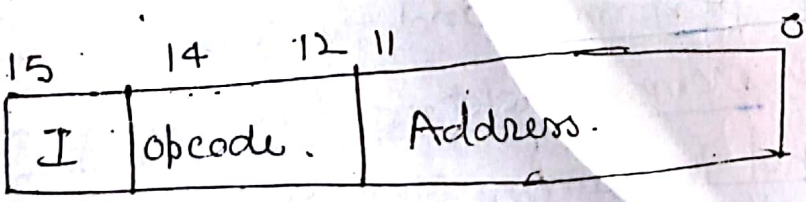
* Indirect Address:- Immediate and Direct address

Sometimes we use the address bits of an instruction code not as an address but as the actual operand when second part of an instruction code specifies an operand instruction is said to be Immediate operand.

When the second part specifies the address of operand, the instruction is said to be direct address.

In ~~this~~ indirect address the bits in the second part of the instruction designate an address of memory word in which address of operand is found.

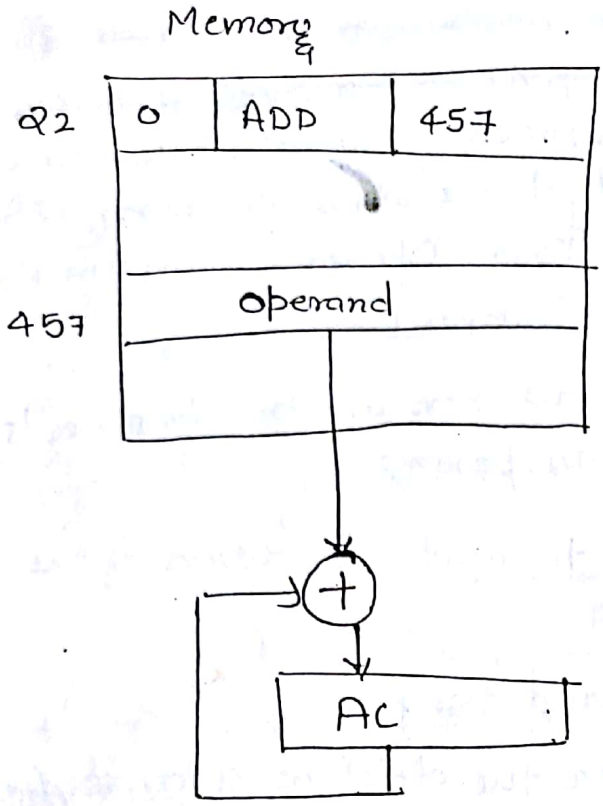
Following figure show demonstration of direct and indirect address.



(a) Instruction formate

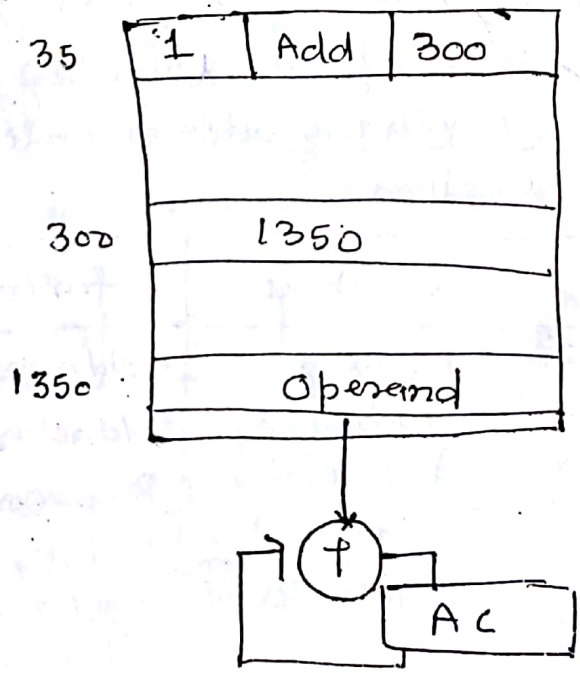
Above figure shows the 3-bits operation code, a 12-bit address and indirect mode bit specify by I. the mode bit is '0' for direct address and 1 for an indirect address

A direct address instruction is shown as:



This direct address instruction is ~~space~~ placed in address 22 in memory. The 'I' bit is '0', So instruction in recognize as a direct address instruction. The opcode specifies an Add instruction and address part is equavalent to 457. control finds the operand in memory at address 457 and add is the Content of Accomolator.

An indirect address instruction is shown is:→



Above figure has mode bit '1', so it is recognized as an indirect address (~~as in memory~~ ~~The 'I' bit is '0', so instruction's recognized as a direct address instruction. The opcode specifies an Add instruction and address part is equivalent to 457~~) instruction. Address part is binary equivalent to 300. Control goes to address '300' to find the address of operand. Address of operand in this case is '1350'. Operand found in Address '1350' and is then based to accumulator.

So indirect address instructions needs two references to memory to fetch an operand.

(1) The first reference is needed to read the address of the operand and.

(2) Second reference is for operand itself.

So, '457 and '1350' are the effective address of direct & indirect address instructions respectively.

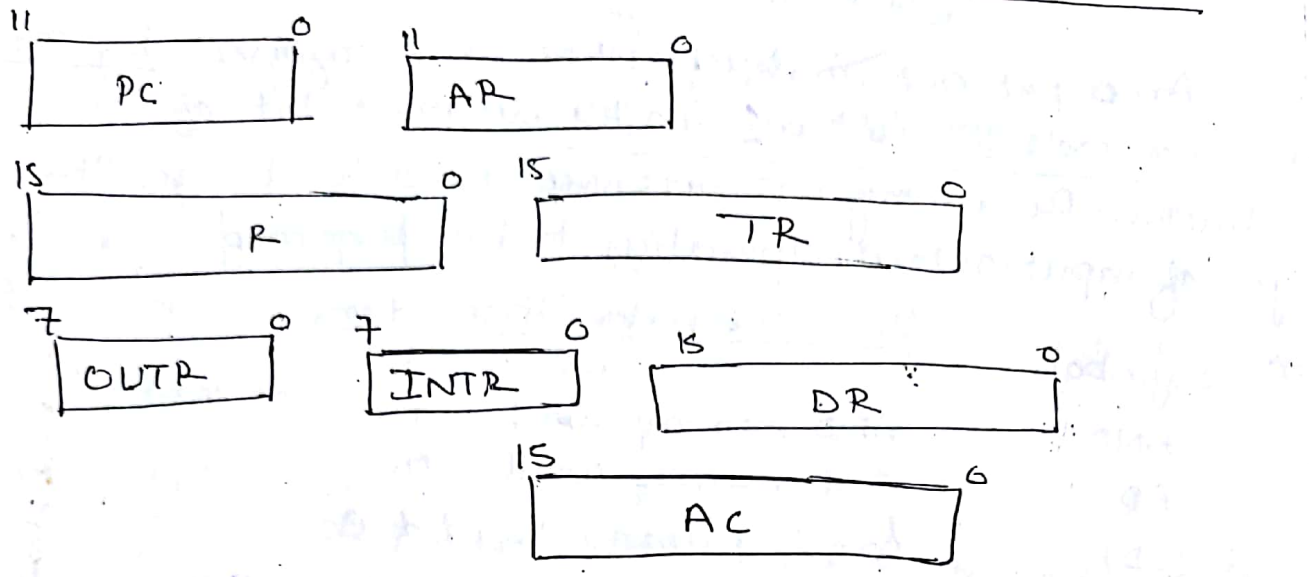
---

nputer Registers:

Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. Control reads an instruction from a specific address in memory and executes it. It then continue by reading the next instruction in sequence and executes it and so on. So Computer needs processors registors for manipulating data and addr. register for holding a memory address. following is list of registers for Basic computers:-

Reg symbol	No of Bits	Ref. Name	function
DR	16	Data Reg.	Hold memory operand
AR	12	Address Reg.	Hold address for memory
AC	16	Accumulator	Processor Reg.
IR	16	Instruction Reg	Hold instruction code
PC	12	Program Counter	Hold address of Instruction.

TR	16	Temporary Reg.	Hold Temporary Data
INPR	8	Input Reg.	Hold Input Character
OUTR	8	Output Reg.	Hold output Character.

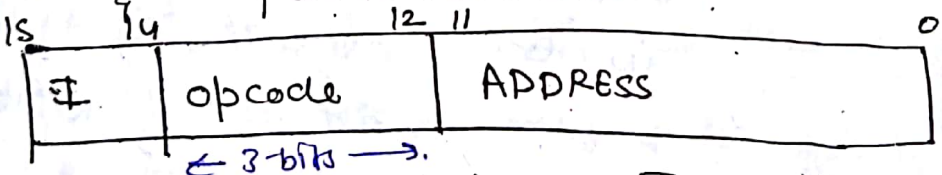


Computer Instructions:→

Basic Computer has three instruction code formats :- Each format has 16 bits. The operation code part of the instruction contains three bits and the remaining 13-bits depends on the operation code encounter.

Three type of instructions are:-

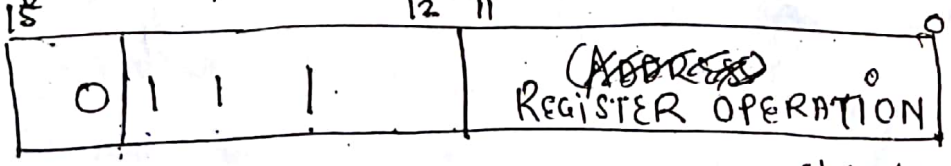
(a) memory - Reference Instruction



(opcode = 000 through 110) i.e. 0→6

Memory - Reference Instruction.

(b) Register- Reference Instruction:→



111=7

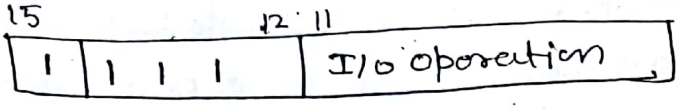
Opcode = 111, I=0

4+4.

(7).

A register Reference instructions are recognized by operation code 111 with the '0' in the left most bit. of the instruction.

c) Input-output Instruction:→



opcode = 111, I = 1

Input-Output Instruction.

An input-output instruction is recognized by the operation code 111 with a 1 in the leftmost bit of the instruction. Remaining 12 bits are used to specify the type of input-output operation to be performed.

* Basic Computer Instructions.

Symbol	Description.
AND	AND memory word to AC
ADD	ADD memory word to AC
LDA	Load memory word to AC
STA	Store content of AC in memory.
ISZ	Increment & skip if zero.

CLA	Clear AC
CMA	Complement AC
CIR	Circulate Right AC
CIL	Circulate left AC
INC	Increment AC
SPA	Skip next instruction if AC positive
SNA	Skip next instruction if AC Negative
SZA	skip next instruction if AC is zero
HLT	Halt computer

Symbol	Description
INP	Input character to AC
OUT	Output Character from AC.
SKI	Skip on input flag
SKO	Skip on output flag.
ION	Interrupt on.
IOF	Interrupt off.

~~Timing & control :-~~

Instruction Cycle :-

A program storing the memory unit of the computer consists of a sequences of instructions. The program is executed in the computer by going through a cycle for each instruction. Following phases are in instruction cycle :-

1) Fetch an instruction from memory.
2) Decode the instruction.
3) Read the effective address from memory. if ~~the instruction has an indirect address~~.
4) Execute the instruction.

After executing all these 4 steps, control goes back to step 1 to fetch, decode and execute the next instruction. This process continues indefinately ~~until~~ a HALT instruction is encountered.
Unless

(*) Fetch & Decode :-

Initially program counter is loaded with the address of the first instruction in the program. Sequence counter SC is cleared to 0, providing a decoded timing signal T_0. After each clock pulse, SC is incremented by one, so that timing signal go through a sequence T_0, T_1, T_2 and so on. The microoperations for the fetch and decode phases can be specified by following register transfer statements :-

T_0 : $AR \leftarrow PC$

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

T_2 : $D_0, \ldots D_7 \leftarrow$ Decode IR (12-14),

$AR \leftarrow IR$ (0-11), $I \leftarrow IR(15)$.

Since AR is connected to the address inputs of memory it is necessary to transfer the address from PC to AR. Instruction read from memory is then placed in the instruction register (IR) with clock transmission associated with signal T_1. At the same time PC is incremented by one to prepare it for the address of next instruction in the program.

AT time T_2, the operation code in IR is decoded, indirect bit is transferred to flip-flop

flip flop I, and address part of instruction is transferred to AR. Sequence counter (SC) is incremented after each clock pulse to produce sequence T_0, T_1 and T_2.

So first two register transfer statements are implemented in bus system as shown in diagram fig 1. To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection:-

(1) Place the content of PC onto the bus by making the bus selection inputs $S_2S_1S_0$ equal to 010 (2)

(2) Transfer the content of the bus to AR by enabling the LD input of AR.

Secondly for implementing second statement, it is necessary to use timing signal T_1 to provide following connections in bus system:-

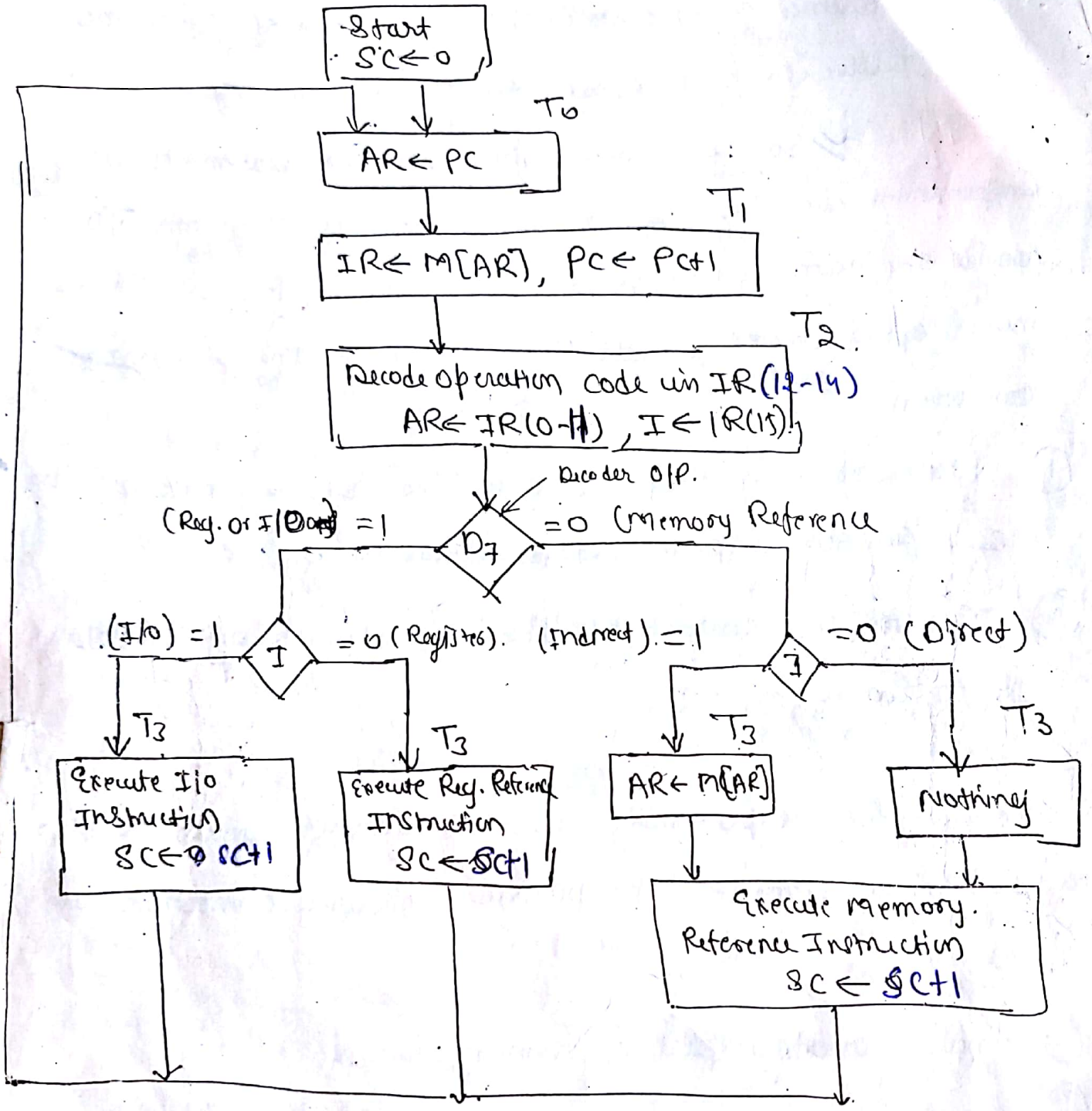
(1) Enable read input of memory.

(2) Place the content of memory onto the bus by making $S_2S_1S_0$ = 111 (7)

(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.

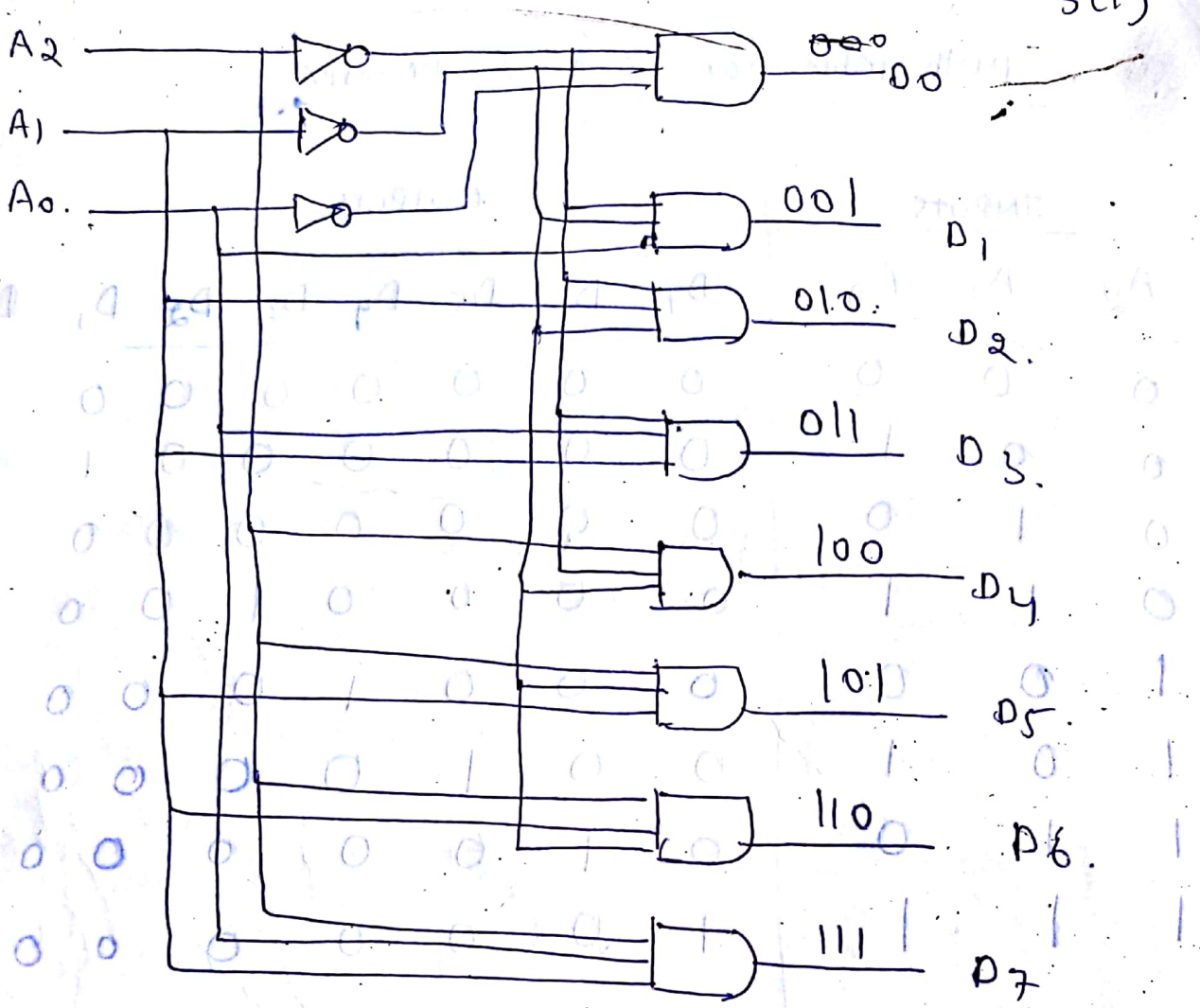
Determine the type of instruction →:



[Flow chart for Instruction Cycle)

In this figure the timing signal that is active after the decoding is T_3. During time T_3, control unit determines the type of instruction that was just read from memory.

5(i)



3-to-8 Decoder.

If $D_7 = 1$, opcode = 111

$D_7 = 0$, opcode = 000 through 110.

A decoder is a circuit that changes a code into a set of signals. It is called decoder because it does the reverse of encoding.

Truth Table for 3-to-8 Decoder

INPUTS			OUTPUTS							
A_2	A_1	A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

When $D_7 = 1$, OPCODE Bits = 111

$D_7 = 0$ OPCODE Bits = 000 - 110

0 - 6.

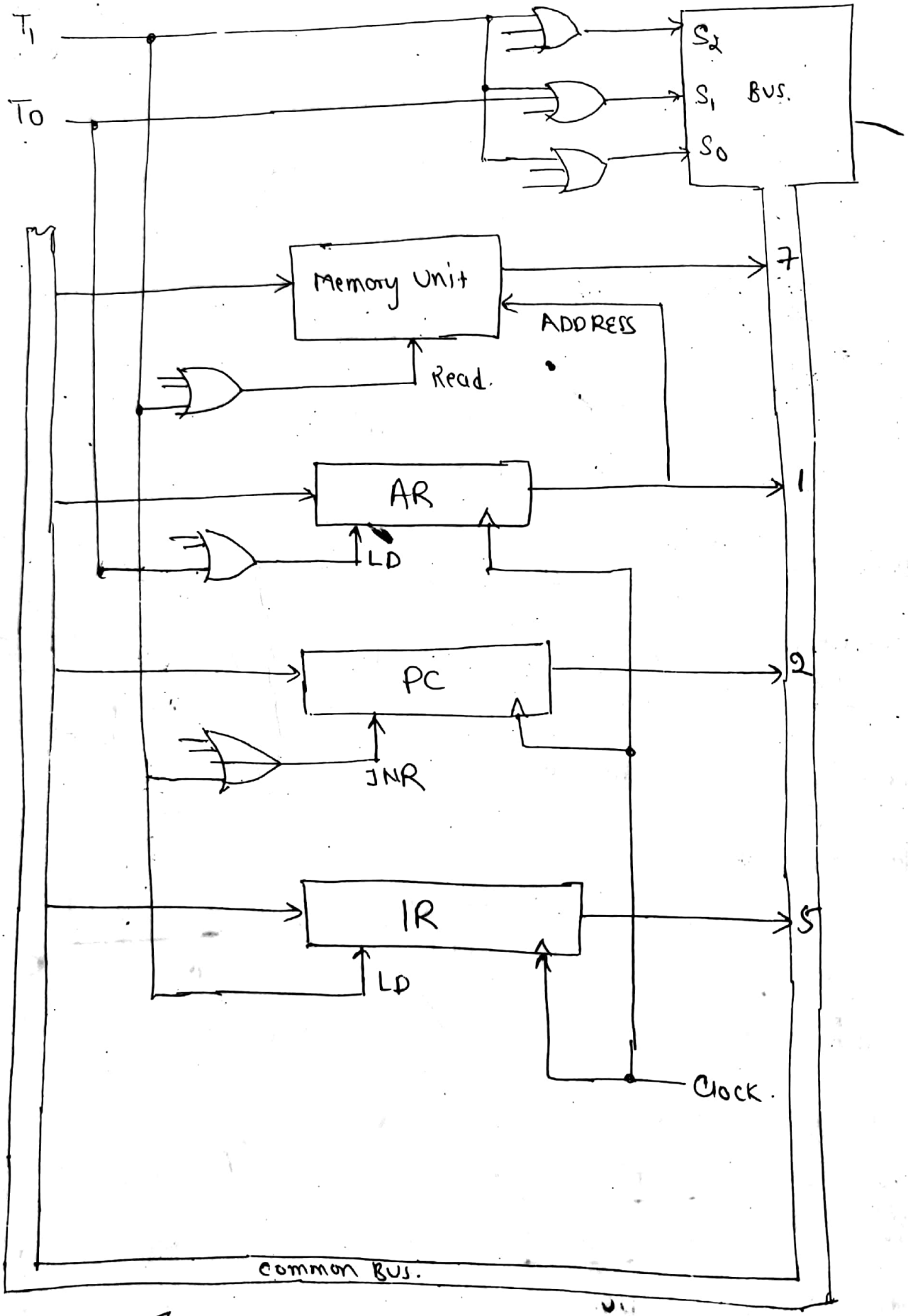


fig 1. [Register Transfer for fetch Phase]

In above flow chart when decoder o/p D_7 is equal to 1, the instruction must be a register-reference or I/p output type. If $D_7=0$, that specify a memory-reference instruction. Control then checks the value of first bit of the instruction which is now available in flip-flop I. If $D_7=0$ and $I=1$, we have memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory. And operation performed is as:-

$$AR \leftarrow M[AR].$$

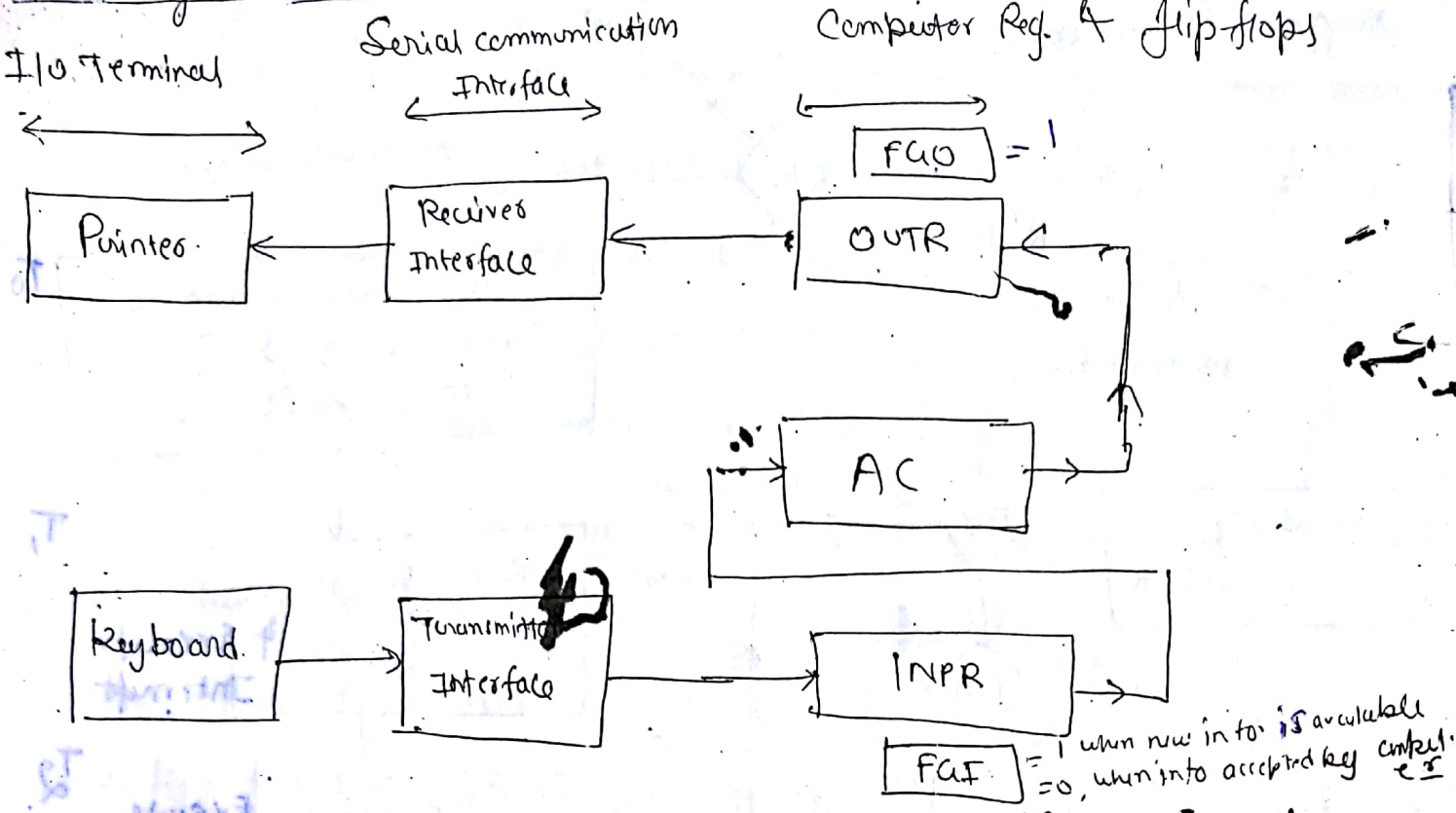
When $D_7=1$, register or I/O reference instruction is selected.

Input-Output and Interrupt

I/P O/P configuration:-

In I/o configuration, terminal sends and receives serial information. Each quantity of information has eight bit of alphanumeric code. Serial information from the keyboard is shifted into the Input register INPR. Similarly serial information for the printer is stored in OUTR. These two interface communicate with a communication interface serially and with the AC in parallel.

I/O configuration is shown as:-

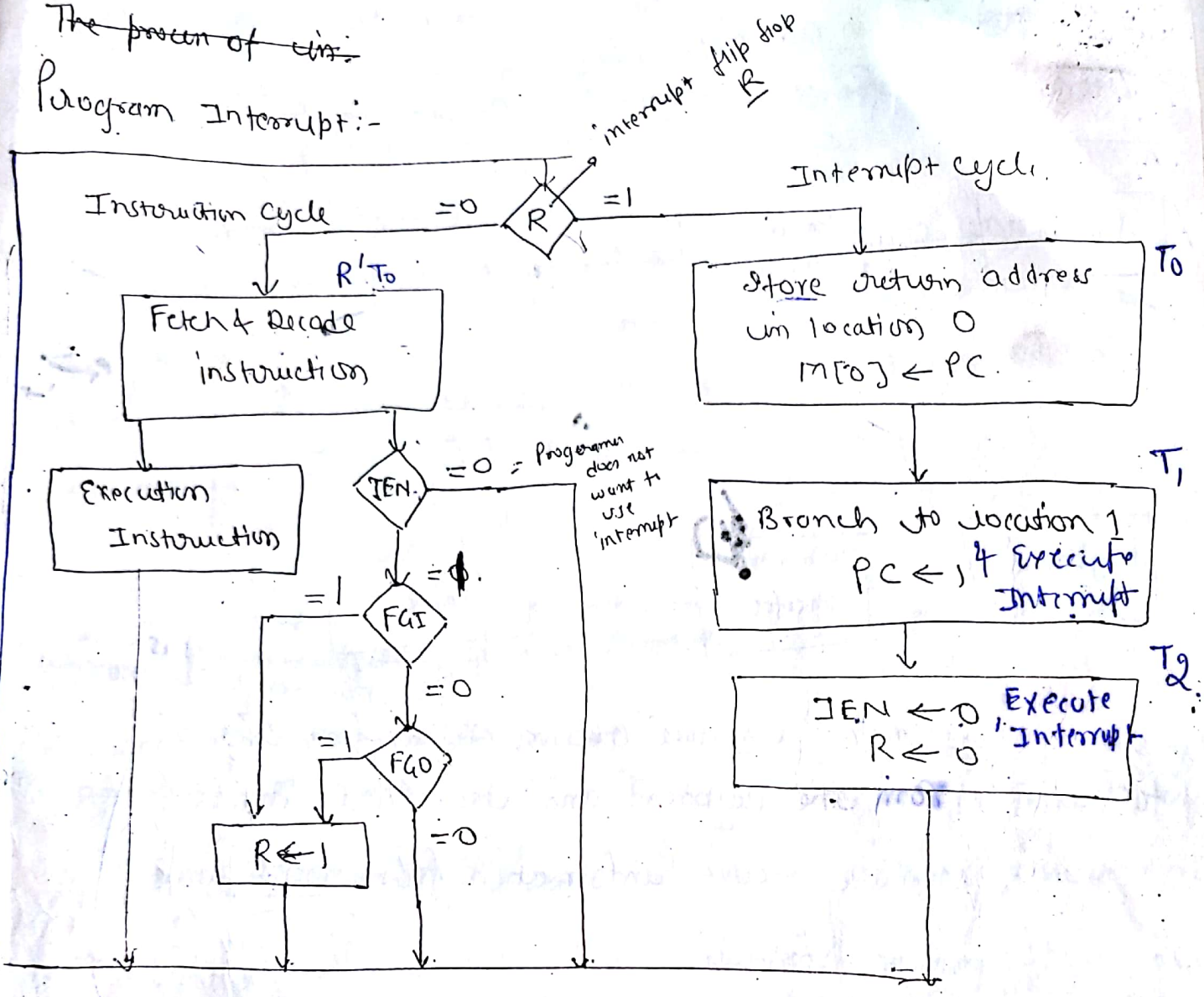


The transmitter interface receive ~~signal from~~ serial information from the keyboard and transmits it to INPR. The receiver interface receive information from OUTR and send it to printer serially.

The input register INPR consists of 8-bits and hold an alphanumeric input information. The 1-bit input flag FGI is a control flip flop. The flag bit is set to 1 when new information is available in the input device and is cleared to '0'. When information is accepted by computer. Flag is needed to synchronize timing rate difference b/w the input device and the computer.

~~The process of this~~

Program Interrupt:-



[Flow chart for interrupt cycle]

An interrupt flip-flop 'R' is included in the computer. When R=0, computer goes through an instruction cycle. During execution phase of instruction cycle IEN is checked by control. If it is '0' it means programmer does not want to use the interrupt. So control continues to next instruction. If IEN=1, control checks both flags bits. If

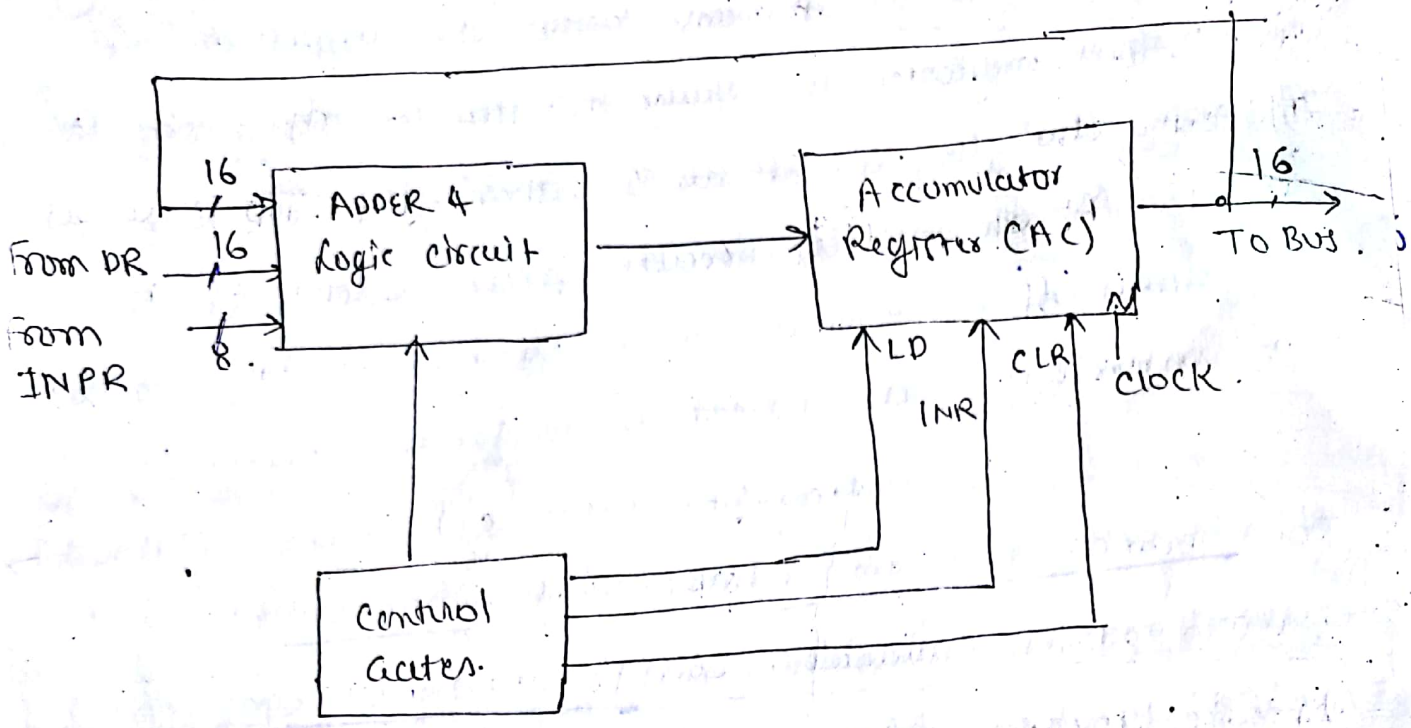
both flags bits are '0' it means neither the input nor the output registers are ready for transfer of information If either the flag is set to 1. then flip flop R is set to 1. At the end of execute phase control checks the value of 'R' and if it is equal to 1. It goes to an interrupt cycle instead of instruction cycle.

Interrupt cycle is a hardware implementation of a branch and save return address operation. The return address available in PC is stored in some specific location where it can be found later when the program returns to instruction at which it was interrupted. This location may be a memory stack, or a specific memory location. Control then insert the address 1 into PC and clears IEN and 'R' so that no more interruptions can occur until the interrupt request from the flag has been serviced.

Design of Accumulator logic circuit.

The circuit associated with the AC register are shown as below:-

P.T.O.



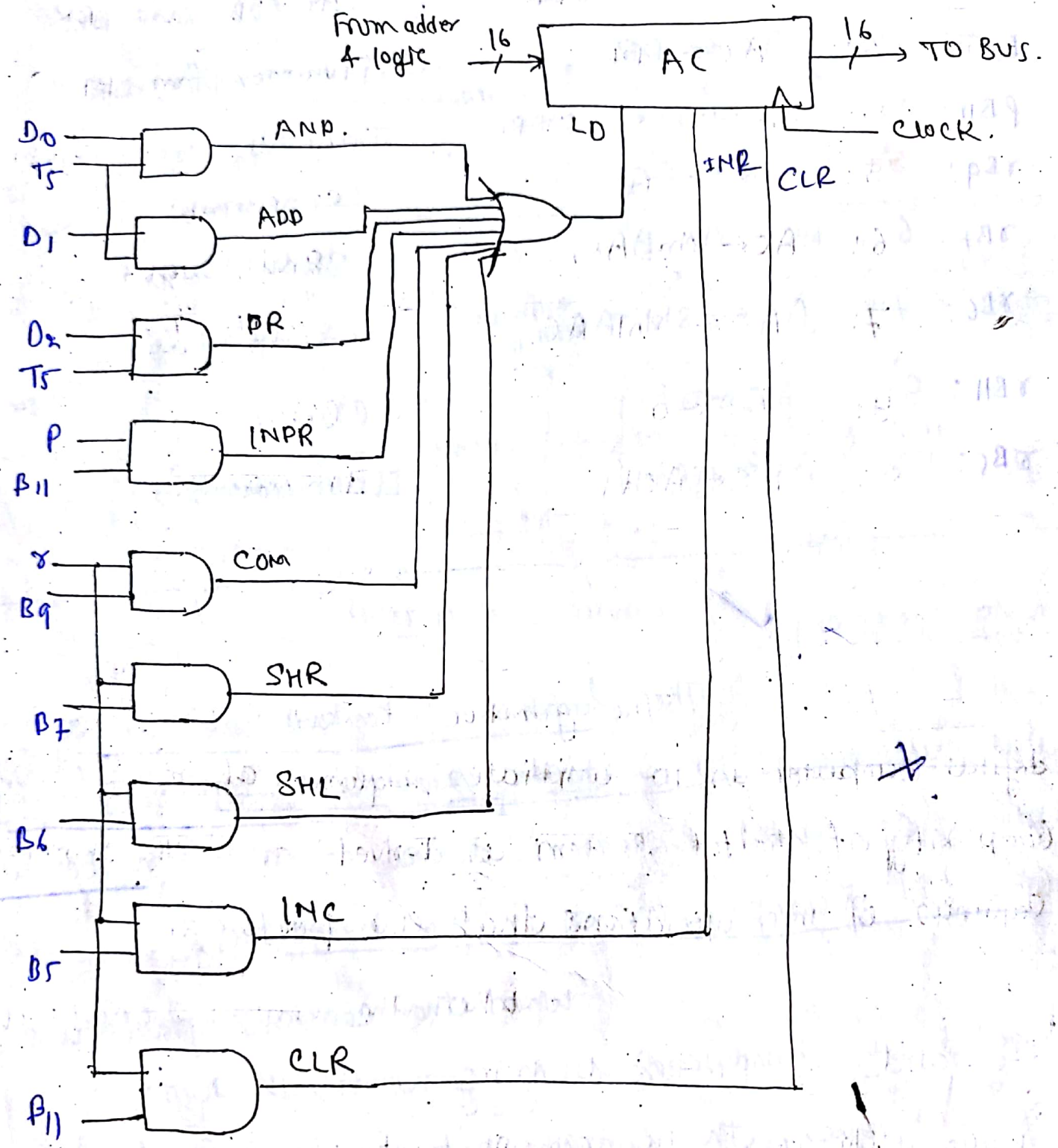
[circuit anociated with AC]

The adder and logic circuit has three set of inputs. one set of 16 inputs comes from the output of AC. Another set of 16 input comes from the Data Reg. and a third set of 8-input comes from the input register INPR.

The outputs of the adder and logic circuit provide the data inputs for the register. Also it is necessary to include logic gates to control the LD, INR and CLR in register and for controlling the operation of adder and logic circuit. & the AC Register.

[Transmission of data]

Gate structure for controlling the LD, INR and CLR of accumulator



And ~~follows~~ above circuit is designed from the following register transfer languages:-

D_0T_5 :	1 :	$AC \leftarrow AC \wedge DR$	AC AND with DR
D_1T_5 :	2 :	$AC \leftarrow AC + DR$	AC ADD with DR.
D_2T_5	3 :	$AC \leftarrow DR$	Transfer from DR.
pB_{11} :	4 :	$AC(0-7) \leftarrow INPR$	Transister from INPR
rB_9 :	5 :	$AC \leftarrow \overline{AC}$	complement
rB_7 :	6 :	$AC \leftarrow shr\ AC,$	Shift right
rB_6 :	7 :	$AC \leftarrow shl\ AC.$	Shift left.
rB_{11} :	8 :	$AC \leftarrow 0$	Clear
rB_5 :	9 :	$AC \leftarrow AC + 1$	Increament.

Control memory ✓

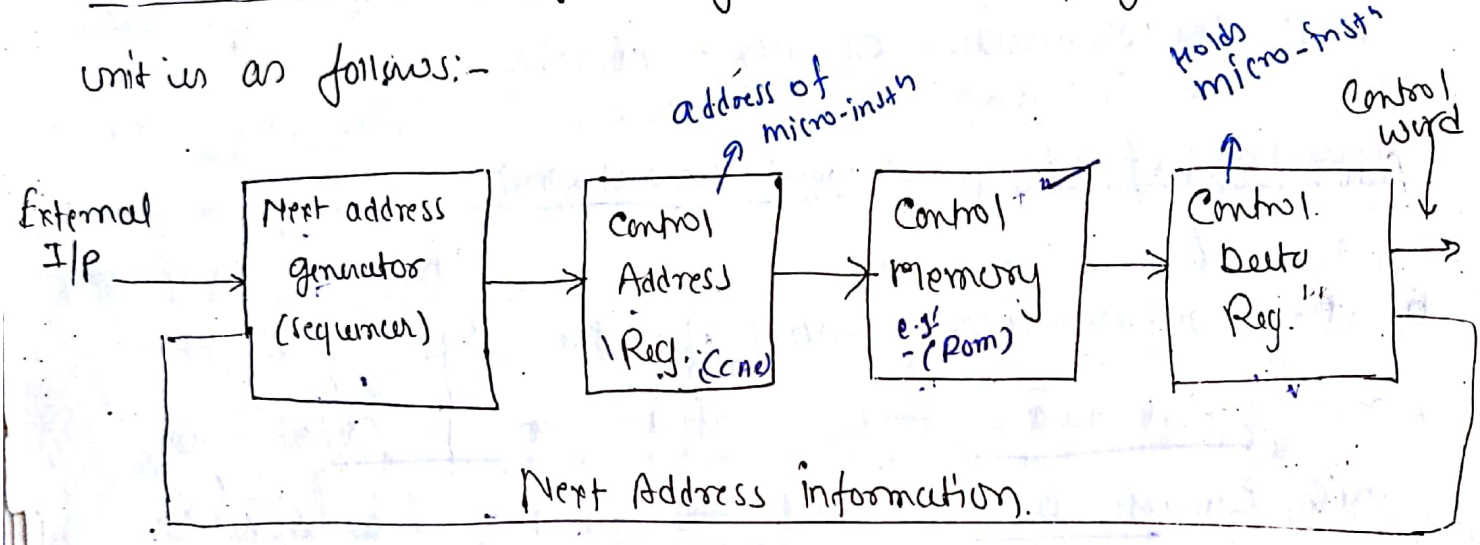
The function of control unit in a digital computer is to initiate sequences of micro operations. Complexity of digital system is derived from the number of sequences of microoperations that are performed.

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired.

The control unit initiates a series of sequential steps of microoperations. During any given

time, certain microoperations are to be initiated, while others remains the idle. The control variable at any given time can be represented by a string of 1's and 0's called control word.

Microprogrammed control unit → A control unit whose binary control variables are stored in memory is called a microprogrammed control unit. and block diagram of microprogrammed control unit is as follows:-



* Control memory is assumed to be a ROM, within which all control information is permanently stored.
* Control memory address register specify the address of micro-instruction. and control data register holds microinstructions read from memory. The microinstruction contains a control word that specifies one or more microoperation for data processor. Once these operations are ~~determined~~ executed. Control must determine next address. The location of the next microoperation may be or next in sequence.

or it may be located somewhere else in control memory.

The function of control data register is to hold the present microoperation while the next address is computed and read from memory. The data register is sometimes called a pipeline register, because it allows the execution of microoperations specify by control word simultaneously with the generation of next microinstruction.

Advantage of microprogrammed control unit:

The main advantage of the microprogrammed control is the fact that once the hardware configuration is established, there should be no need for further hardware or wiring changes. If we want to establish a different control sequence for the system, all we need to do is specify a different set of microoperations for control memory. The hardware configuration should not be changed for different operations; the only thing that must be changed is microprogram residing in control memory.

Address Sequencing →

Address Sequencing is the procen of specifing the address sequence for the microoperations. To appreciate the address sequencing in a microprogram control unit, let us anume the following steps that the control must undergo during execution of single computer instruction. ;-

(1) An initial address is loaded into the control addren (CAR) register when power is turned on in the computer. This address is usualey the address of the first microoperation that activates the instruction fetch routine. At the end of fetch routine, the instruction is in the instruction register of computer.

(2) The control memory next must go through the routine that determines the effective addren of operands. A machine instruction may have bits that specify various addressing modes, such as indirect address and index registers. The effective address computation routine in control memory can be reached through a branch microinstruction, which is conditioned on status of mode bits of instruction. when effective ess computation routine is completed, the

address register.

(3) The next step is to generate the microoperations that execute the instruction fetched from memory. Each instructions has its own microprogram routine stored in a given location of control memory. The transformation from the instruction code bits to an address in control memory where routine is located is referred to as a mapping process.

instruction code bit → address in control memory

"A mapping procedure is a rule that transforms the instruction code into a control memory address."

Once the required routine is reached, the microinstructions that execute the instructions may be sequenced by incrementing the control address register, (CAR).

(4) When the execution of the instruction is completed, control must return to fetch routine. This is ~~executed~~ accomplished by executing an unconditional branch microinstructions to the first address of fetch routine.

So in brief the address sequencing capabilities required in control memory are:-

(a) Increment the control address register.

(b) Unconditional branch or conditional branch, depending

are Status bit conditions.

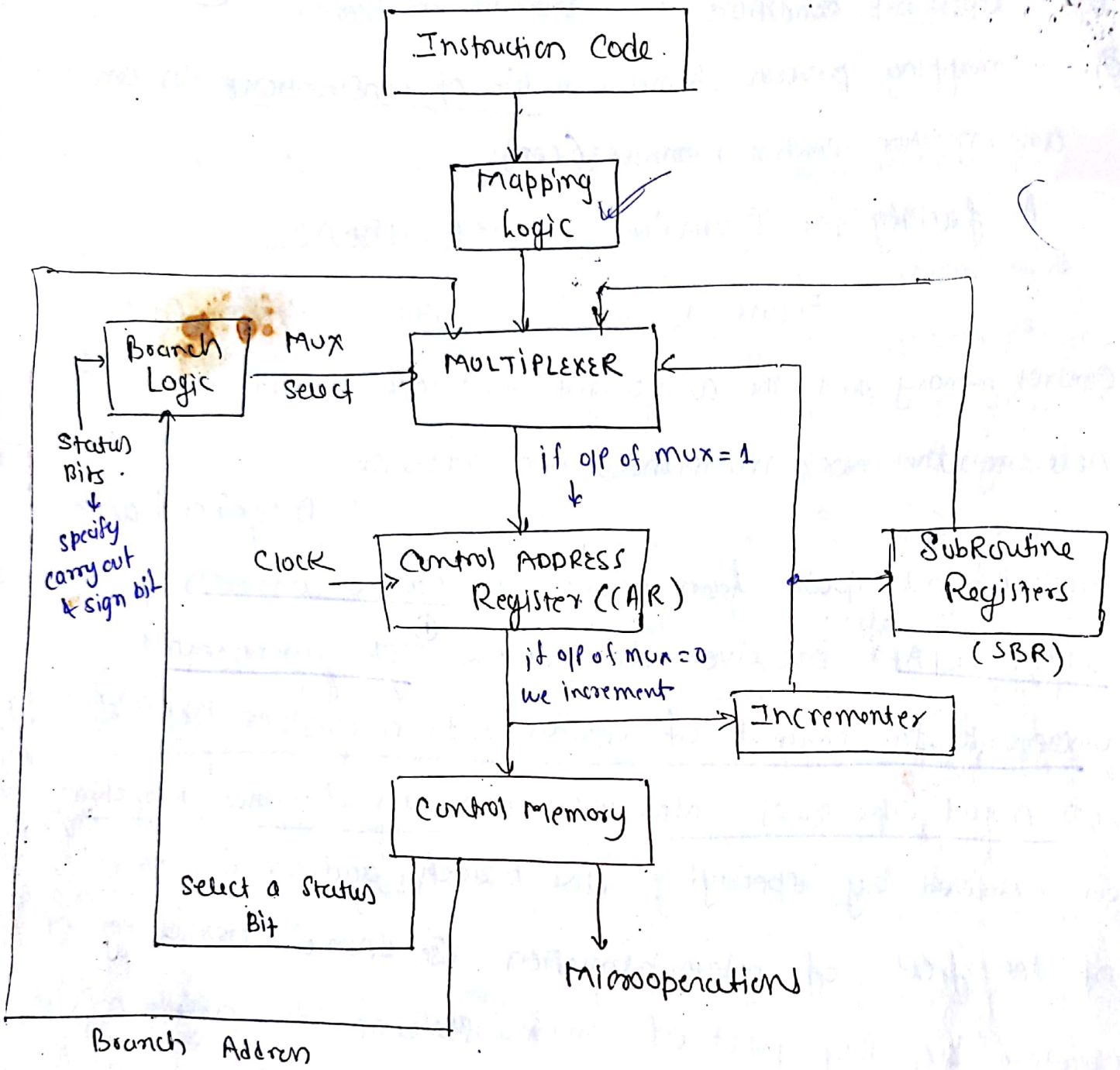
(c) A mapping procen from the bits of instructions to an addren for control memory (CAR)

(d) A facility for Subroutine call and oreturn.

Figure (a) Shows the block diagram of a control memory and the anociated hardware needed for selecting the next microinstruction addreses.

Diagram has four different paths from which the control addrem oregister (CAR) oreceives the address. (1) The incrementer increments the content of control addren oregister by one, to reect the next microinstruction in sequence. Branching is achieved by specifying the branch addrem in one of the fields of microinstruction. Conditional branching is obtained by using part of microinstruction to select a specific Status bit in order to determine its conditions.

(2) Branch logic provides decision-making capabilities in the control unit. The Status conditions are special bits in the System that provide parameter information such as the carry out of an adder, the sign bit of number etc.



[Figure-a [selection of address for control memory].]

The branch logic hardware may be implemented in a variety of ways. The simplest way is to test the specified condition and branch to the indicated address if condition is met; otherwise address is incremented.

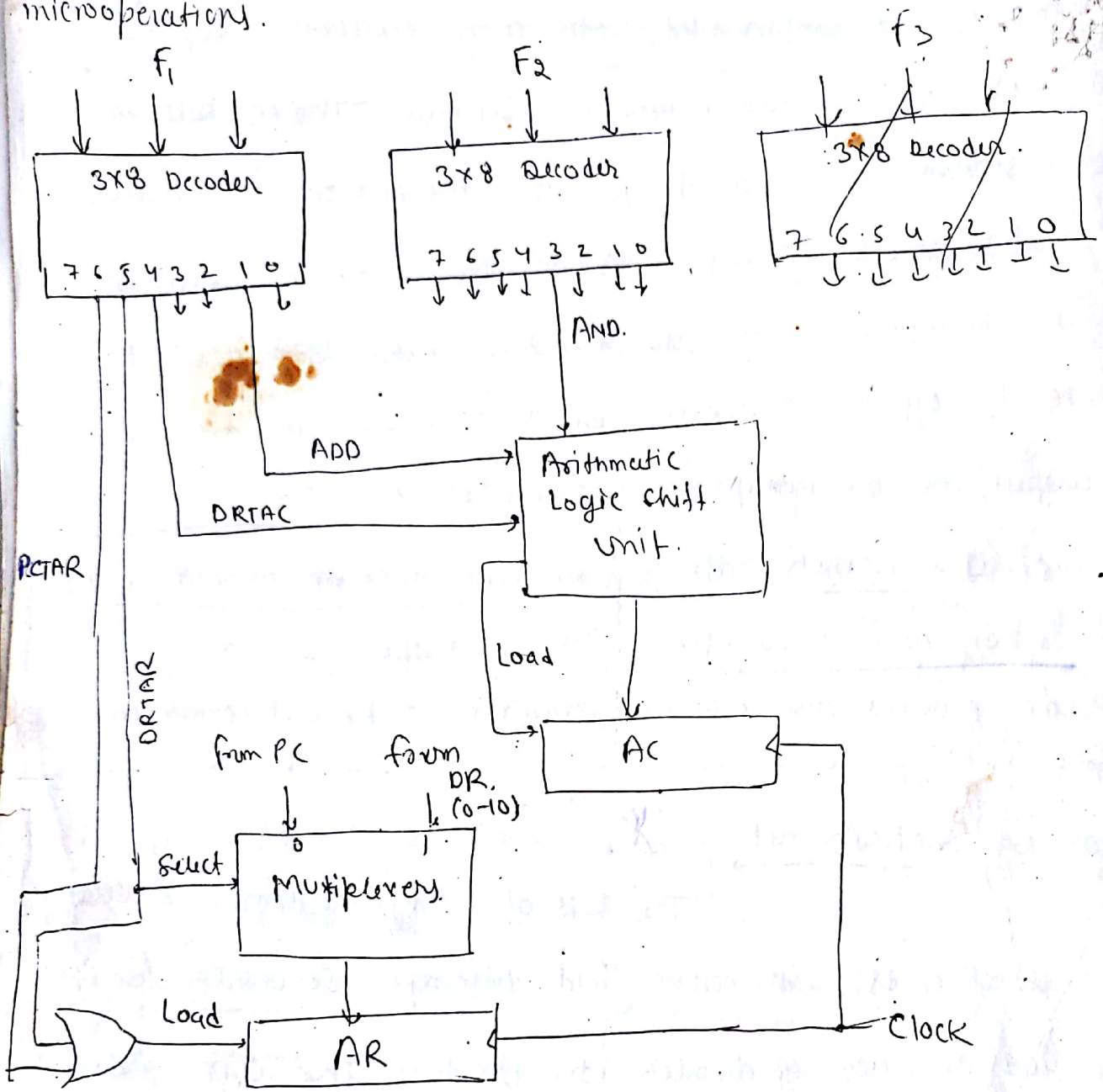
This can be implemented with a multiplexer. Suppose that there are 8-status bits in system. Three bits in microinstruction are used to specify any one of 8 status bit configurations. 3-bits provides the selection variable for the multiplexer. If the selected status bit is in the state 1, o/p of multiplexer is 1, otherwise it is 0. A 1 output in the multiplexer generates a control signal to transfer the branch address from the microinstruction into the control address register. A '0' output in the multiplexer causes the address register to be incremented.

Design of control unit

The bits of microinstructions are usually divided into fields, with each field defining a separate function. Each field requires a decoder to produce the corresponding control signals. Various fields encountered in instruction format provides control bits to initiate microoperation in the system.

Figure shows 2-decoders and some of the connections that must be made from their outputs. Each of the three fields of microinstruction presently available in output of control memory are decoded with a 3x8 decoder to provide 8-outputs. Each of these outputs must be connected to proper circuit to initiate corresponding

microoperations.



(Recoding of microperation fields).

So in above diagram when F1 = 5 (101), next clock pulse transition transfers contents of DR to AR. Similarly when F1 = 6 (110), there is transfer from PC to AR. ~~when~~ outputs 5 and 6 of decoder F1 are connected to load input of AR so that when either one of these outputs is active, information from

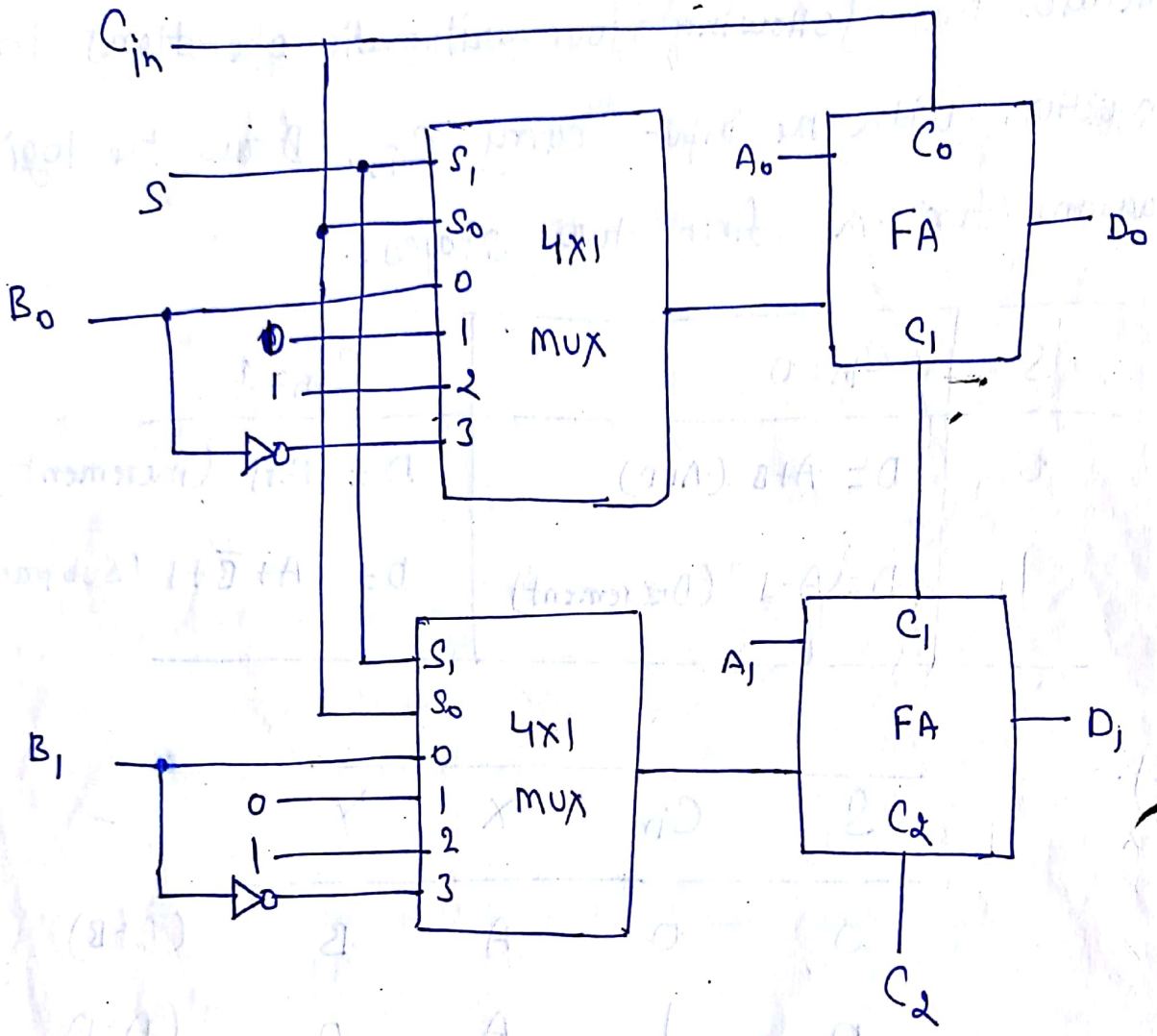
2. Design an arithmatic circuit with one selection variable S and two n-bit data inputs A and B. The circuit generates the following four arithmatic operations in conjuction with the input carry C_{in}. Draw the logic diagram for the first two stages:-

S	$C_{in}=0$	$C_{in}=1$
0	D = A+B (ADD)	D = A+1 (increment)
1	D = A-1 (Decrement)	D = A+$\overline{B}$+1 (Subtract)

Sol:-

S	C_{in}	X	Y	
0	0	A	B	(A+B)
0	1	A	0	(A+1)
1	0	A	1	A-1
1	1	A	$\overline{B}$	A+$\overline{B}$

P.T.O.



[Circuit Diagram]

[Adding any number to carry is increment]

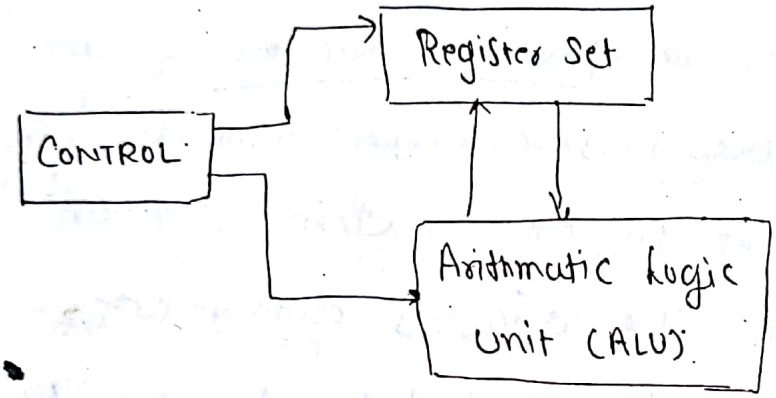
While Adding all '1's to any number is decrement.

Original (1)

MODULE-3 Central Processing Unit"

Central Processing Unit.

The part of the computers that performs the bulk of data processing operations is called the central processing unit and referred to as CPU. The CPU is made of 3- major parts shown as below:-



[Major Component of CPU].

① Register set stores the intermediate data used during the execution of instructions.

② The arithmetic logic unit (ALU) perform the require microoperation for executing the instructions.

③ The control unit supervise the transfer of information among the registers and instruct the ALU as to which operation to perform.

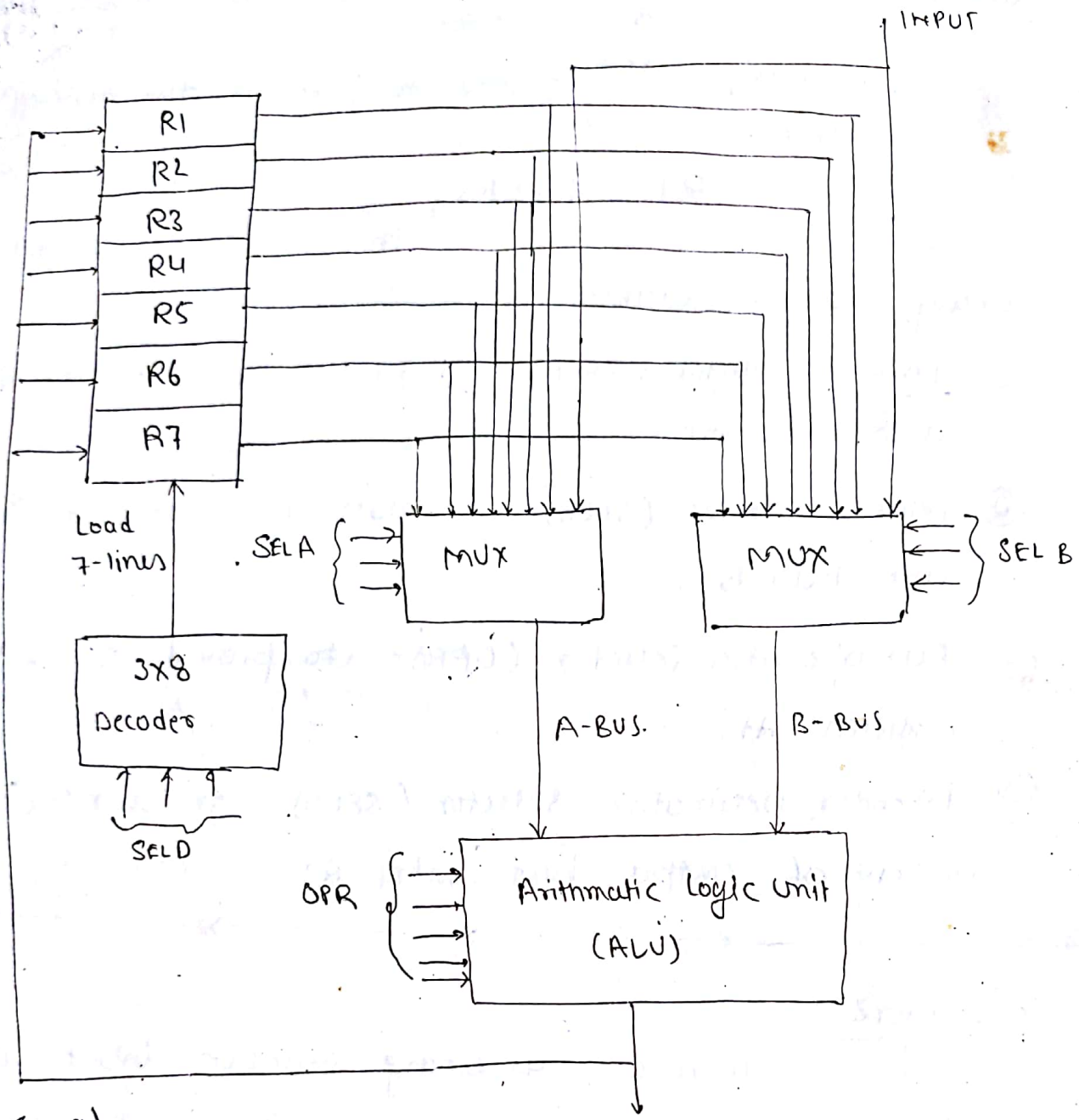
CPU performs a variety of functions dictated by the type of instructions that user are machine incorporated in computer.

General Register Organization

In the programming environment, the memory locations are needed for storing pointers, counters, return addresses, temporary results and partial products during various microoperations. But memory access is most time consuming process in computer, so it is more convenient and more efficient to store these intermediate values in processor registers. When a large number of registers are included in the CPU, it is most efficient to connect them through a common bus system. The registers communicate with each other not only for direct data transfer, but also while performing various microoperations.

A bus organization for 7-CPU registers is shown in fig (a).

The output of each register is connected to two multiplexers (MUX) to form the two buses A and B. The selection line in each multiplexer select one register or the input data for the particular bus. The A and B buses form the input to a common arithmetic logic unit (ALU). Operation selected in the ALU determines the arithmetic or logic microoperation that is to be performed.



(Fig-a) [BLOCK DIAGRAM]

The result of the microoperation is available for o/p data and also goes into the input of all registers. The register that receives the information from the output bus is selected by decoder. The decoder activates one of the register load input, thus providing a transfer path between the data in the

Output bus and the inputs of selected destination register.

e.g. For example if we have to perform the following operation:-

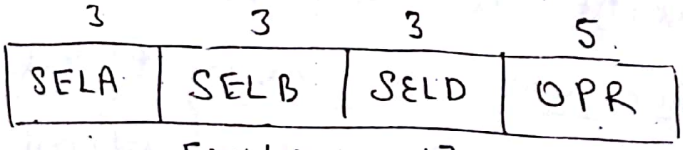
$$R1 \leftarrow R2 + R3$$

then control must provide binary selection variables to 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 the arithmetic addition A+B.
4. Decoder Destination Selector (SELD): to transfer the content of output bus into R1.

Control word:-

There are 14- binary selection inputs in the unit, and their combined value specifies a control word. The 14-bit control word is as follows:-



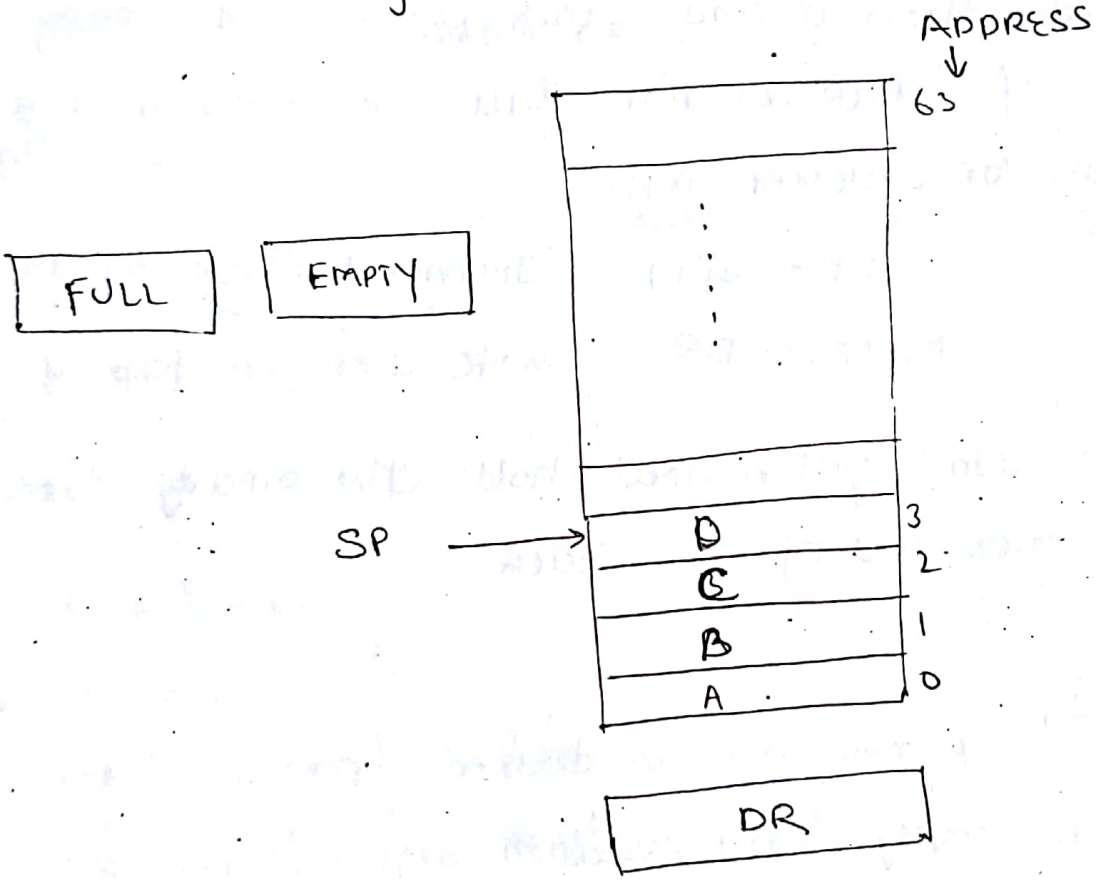
[control word].

It consists of 4- fields. Three fields contain three bits each and one field has 5- bits.

Stack organization →:

A stack is a storage device that stores information in such a manner that the item stored last is the first item retrived. So, Stack is last-in first-out (LIFO) device. The register that holds the address for the stack is called a stack pointer (SP), because its value always points to the top item of the stack.

Block diagram for 64-word stack is as:-



Basic operation of a stack → There are basic 2-operation of Stack:-

① **Push or insertion operation:-** The operation of insertion of new item is ~~thought~~ called Push operation, because it can be thought of the result of pushing a

new item on top.

(3) Pop or Deletion :- The operation of deletion is called pop-up because it can be thought of as the result of removing one item so that stack pops-up.

PUSH OPERATION :-

Initially SP is cleared to '0', EMPTY is set to 1, and FULL is cleared to '0'. So that SP points to the word at address '0' and Stack is marked empty and not full. If Stack is not full i.e. FULL=0, a new item can be inserted as:-

SP ← SP+1	Increment stack pointer.
M[SP] ← DR	write item on top of stack.

DR is data register that holds the binary data to be written or read out of the stack.

Pop operation :-

A new item is deleted from the stack if stack is not empty, and operation sequences are as:-

DR ← M[SP]	Read item from top of stack.
SP ← SP-1	Decrement stack pointer
If (SP=0) then EMPTY ← 1	Check if stack is empty.
FULL ← 0	Mark the stack not full.

Reverse Polish Notation →

A stack organization is very effective for evaluating arithmetic expressions. Common arithmetic operations are written in infix notation. in which operator written between operands. There are three type of notations for arithmatic operation as follows:-

① Infix notation:-

In infix notation operator is placed between the operands. e.g. $(A*B)+(C*D)$ A+B

② Prefix notation:-

In prefix notation, operator is placed before the operands. eg.

A+B infix notation

+AB Prefix notation.

③ Postfix notation:- In postfix notation, operator is placed after the operands.

e.g. A+B infix notation

AB+ postfix notation.

Evaluation of arithmatic operation by using stack. →

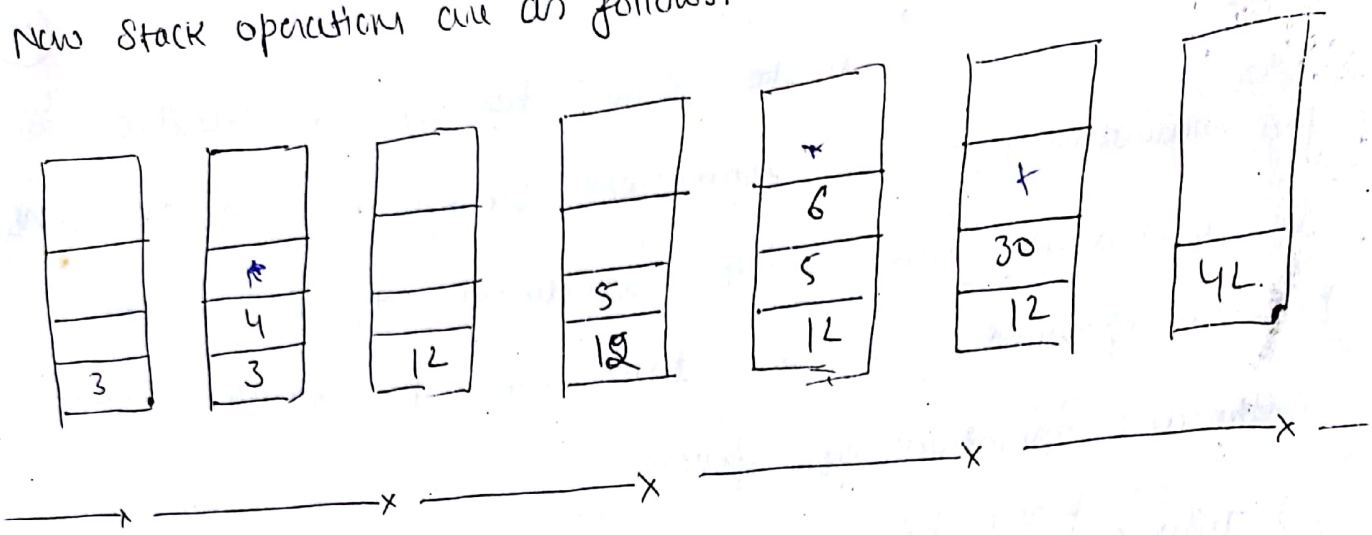
Suppose we have the arithmatic expression:-

$(3*4)+(5*6)$.

2*3+ 6/2*12

calculate postfix notation i.e. 34*56*+

Now stack operations are as follows:-



Instruction Formats:- An instruction format has the following fields:-

1. An operation code field that specifies the operation to be performed.
2. An address field that designates a memory address or a processor register.
3. A mode field that specifies the way the operand or the effective address is determined,

and diagramatically shown as:-

MODE BIT	Operation code	ADDRESS Field.

[Instruction format].

Type of instructions:- Following are the type of instructions:-

① Three- Address Instructions →

Computer with three-address instruction formats can use each address field to specify either a processor register or a memory operand. Following program evaluates X = (A*B) * (C+D) as:-

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

The advantage of three address format is that it results in short programs when evaluating arithmetic expressions. The disadvantage is that binary-coded instruction requires too many bits to specify three addresses.

② Two-Address Instructions :-

Two-address instructions are the most common in commercial computers. Each address field can specify either a processor register or a memory word. Program to generate X = (A+B) * (C+D) is as:-

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

③ One-Address Instructions →:

One address instructions use an accumulator (AC) register for all data manipulation. For multiplication and division there is need for a second register.

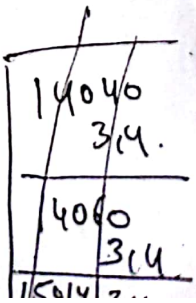
Program to evaluate X = (A+B) * (C+D) is as:-

LOAD	A	AC ← M[A]
ADD	B	AC ← AC + M[B]
STORE	T	M[T] ← AC
LOAD	C	AC ← M[C]
ADD	D	AC ← AC + M[D]
MUL	T	AC ← AC * M[T]
STORE	X	M[X] ← AC.

④ Zero-address Instructions →: A stack organized computer does not use an address field for the instructions ADD and MUL. The PUSH and POP instructions, however, need an address field to specify the operand that communicate with Stack. Program to evaluate X = (A+B) * (C+D) is as follows:-

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

TOS = Top of Stack



ADDRESSING MODES:

The addressing mode specifies a rule for interpeting or modifying the address field of the instruction. The way the operands are chosen during program execution is dependent on the addressing mode of instruction. Computer uses addressing mode due to following main reasons:-

① To give programming versatility easibility. to the user by providing such facilities as pointer to memory, counter for loop control, indexing of data and program relocation.

② To reduce the number of bits in the addressing field of the instruction.

Different Type of addressing modes :-

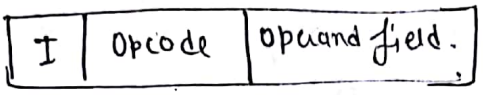
① Implied mode / Direct addren mode

In implied mode, the operands are specified implicitly in the definition of the instruction. e.g. the instruction "Complement accumulater" is an implied mode. because operand in accumulator register is implied in definition of instruction. Another example in zero-addren instructions in stack-organized computer are implied-mode instruction since operands are implied to be on top of stack.

② Immediate mode :- / Indirect addren mode.

In this mode the operand is

Specified in the instruction itself. In other words, immediate mode instruction has an operand field rather than an address field. The operand field contains the actual operand to be used in conjuction with the operation specified in instruction. Immediate mode instructions are useful for initializing registers to a constant value.



(3) Register Indirect mode:

In this mode, ~~operands are in registers~~ instruction specify a register in the CPU whose contents give the address of operand in memory. In other words selected register contain the address of operand rather than operand itself. e.g. LXI H, 2500H - Load H-L Pair with 2500H
MOV A, M. → move content of memory location whose address in H-L pair (2500) to AC.

(4) Auto incremented or autodecremented mode:

(4) Register mode:

In this mode, operands are in registers that reside within CPU. Particular register is selected from a register field in the instruction. e.g. MOV A B.
78, ADD B.

(5) Auto increment or autodecrement mode:

This is similar to register indirect mode except that the register is incremented or decremented after its value is used to access memory. When the address ~~is~~ fixed.

the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table.

(6) Direct address mode → In direct address mode, the effective address is equal to the address part of the instruction. The operand reside in memory and its address is given directly by address field of instruction.

(7) Indirect address mode → In this mode, address field of instruction gives the address where the effective address is stored in memory. Control fetches the instruction from memory and uses its address part to access memory again to read the effective address.

In a few addressing modes requires that the address field of instruction be added to content of a specific register in CPU. The effective address in these mode is calculated as:-

Effective Address = address part of instruction + content of

(8) Relative Address mode: PC + address part of instruction. In this mode content of Program counter is added to address part of instruction in order to obtain the effective address. The address part (is usually) of instruction is usually a signed

number which can be either positive or negative. When this number is added to content of program counter, the result produces an effective address whose position in memory is relative to address of the next instruction.

e.g. Program counter = 825

Address part of instruction = 25.

Effective address computation for relative address mode is = 825 + 25 = 850.

So, effective address is 24 memory location forward from address of the next instruction.

(9) Index Addressing mode.

In index addressing mode content of index register is added to address part of instruction to obtain the effective address. The index register is a special CPU register that contains an index value. Address field of instruction defines the begining address of data array in memory.

(10) Base Register Addressing mode

In base register addressing mode content of base register is added to the address part of instruction to obtain effective address. This is similar to index addressing mode except that the register is now called base register instead of index register.

POP

Program Control - PC

Instructions are always stored in successive memory locations. When processed in the CPU, the instructions are fetched from consecutive memory locations and executed. Each time an instruction is fetch from memory, the program counter is incremented so that it contains the address of next instruction in sequence. After the execution of data transfer or data manipulation instruction, control returns to fetch cycle with the program counter containing the address of the instruction next in sequence. A program control instructions specify condition for altering the content of program counter, while data transfer and data manipulation instruction specify conditions for data processing operations.

Following are some typical Program control instructions are listed as below:-

Branch	BR
Jump	Jmp
Skip	SKP
Call	CALL
Return	RET
Compare	CMP.
Test	TST.

Reduced Instruction Set Computer (RISC) & CISC

CISC → In the early days, computer had small and simple instruction sets, forced mainly by the need to minimized the hardware used to implement them. Many computers have instruction sets that include more than 100 and sometimes even more than 250 instructions. A computer with a large number of instructions is classified as a complex Instruction set computer.

RISC → It is recommended that computers use fewer instructions with simple construct so they can be executed much faster within the CPU without having to use memory as often. This type of computers is classified as a reduced instruction set computer or RISC.

CISC characteristics

1. A large number of instructions - typically from 100 to 250 instructions.
2. Some instructions that perform specialized tasks and are used infrequently.
3. A large variety of addressing modes - typically 5 to 20 different mode.

(4) variable length instruction formats.

(5) Instruction that manipulate operands in memory.

RISC characteristics

(1) Relatively few instructions.

(2) Relatively few addressing modes.

(3) Memory access limited to load and store instructions.

(4) All operations done within the registers of the CPU.

(5) ~~Fixed~~ Fixed - length, easily decoded instruction format.

(6) Single-cycle instruction execution.

(7) ~~Hard~~

(8) RISC is ~~hardwired~~ Hardwired rather than ~~microprogrammed~~ microprogrammed control.

MULTIPLICATION ALGORITHMS

Multiplication of two fixed-point numbers in signed-magnitude representation is done with following example:-

```
23        10111   → multiplicand
19      x 10011   → multiplier
        -------
          10111
         10111x
        00000xx
       00000xxx
      10111xxxx
      --------
437   110110101  → Result.
      --------
```