Unit - 3

Basic Computer Organization :→

1> Register Transfer Language →

Digital system design is an interconnection of digital hardware modules which are constructed from digital components such as :→ registers, decoders, arithmetic elements and control logic.

(a) Micro operation → It is an elementary operation performed on the information stored in one or more registers.
for eg. shift, count, clear, load etc

The internal hardware organization of a digital computer has following specifications :→

1. The set of register it contains and their function
2. The sequence of micro operations performed on the binary information stored in the registers.
3. The control that initiates the sequence of micro operations.

(b) Register Transfer Language :→

The symbolic notation used to describe the micro operation transfers among registers is

called a register transfer Language.

(C) Register Transfer :→

Capital letters are used to denote a computer register . For eg :-

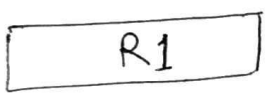
MAR : It is the register that holds an address for the memory unit or called memory address register.

PC : Program Counter

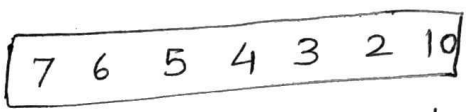
IR : Instruction register

R1 : processor register

The block dig representation of a register →



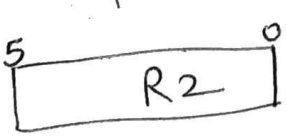
(a) Register R



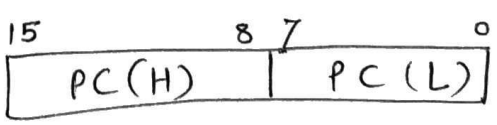
(b) Showing individual bits

The most common way to represent a register is by a rectangular box shown in fig (a) The individual flip-flops in an n-bit register are numbered in sequence from 0 through n-1 from right most position.

Similarly a 16-bit register can be marked on top of the box shown in fig (c) & (d) below.



(c)



(d) Division in two parts

Bits 0 → 7 are assigned the symbol L (low byte)
Bits 8 → 15 are assigned " " H (high byte)
And 16-bit register named as PC. (Program counter)

Register Transfer → It is the information transfer from one register to another in symbolic form by means of a replacement operator. eg.

$$R2 \leftarrow R1$$

denotes a transfer of the content of register R1 into register R2.
The contents of source register (R1) does not change.

(d) Control function :→

If we want the transfer to occur only under a predetermined control condition, like

If (P = 1) then (R2 ← R1)

Here P is a control signal.
Sometimes it can be seperated by a control function that is equal to 1 or 0. like

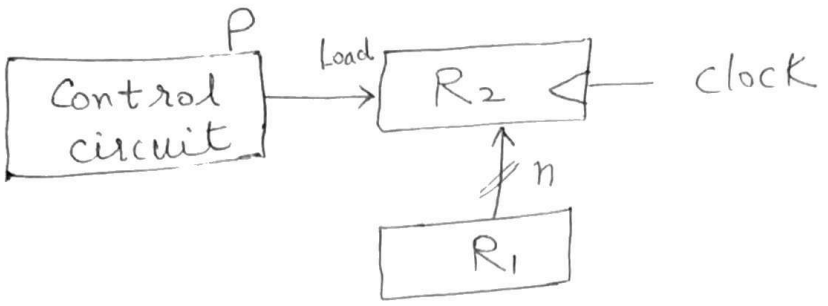
P : R2 ← R1

colon is used to terminate the control condition and transfer operation if P = 1

Register Transfer operation is depicted with block dig →

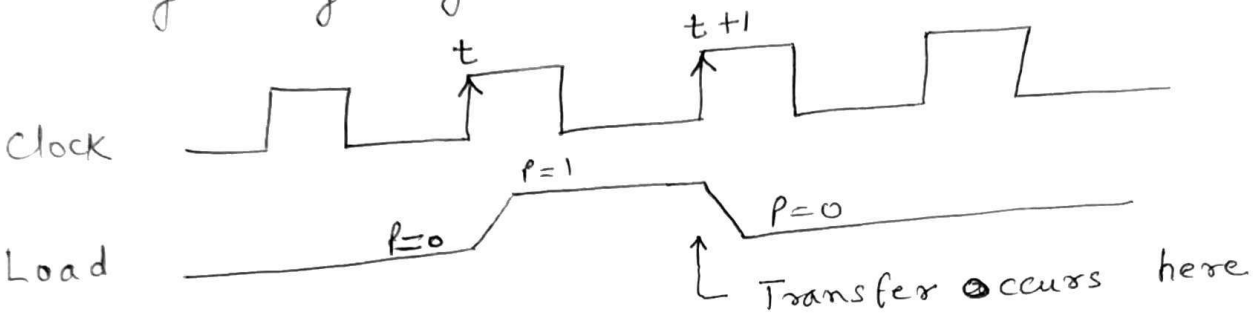
Block dig

Transfer from R1 to R2 when p=1



'n' o/p's of register R_1 are connected to the 'n' i/p's of register R_2

Using Timing dig →



P is activated in the control section by the rising edge of a clock pulse at time 't'. The data i/p's of R_2 are then loaded into the register in parallel

Basic Symbols for Register Transfer are given:→

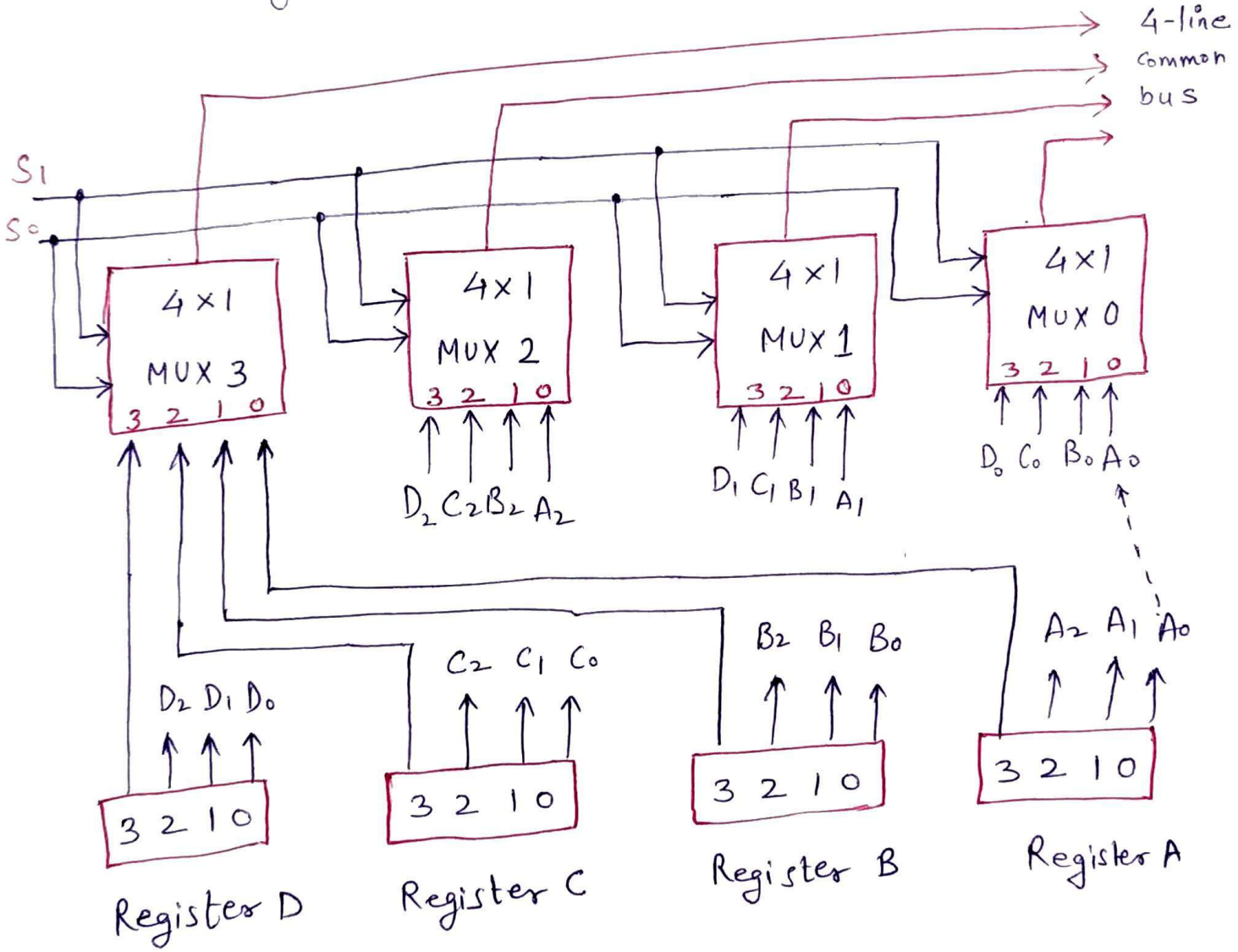
Symbol	Description	Examples
Letters (numerals)	Denotes a register	MAR, R_2
Paranthesis ( )	Denotes a part of a register.	R2(0-7), R2(L)
Arrow ←	Denotes transfer of data.	R2 ← R1
Comma ,	Seperates two micro operations	R2 ← R1, R1 ← R2

Bus & Memory Transfers →

A digital computer has many registers and paths provided so that transfer of information can be done from one register to another, with a common bus system. (CBS)

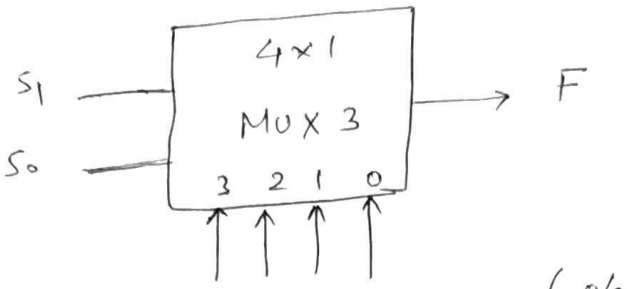
CBS structure has a set of common lines, one for each bit of a register

The design of a bus system for 4 registers →

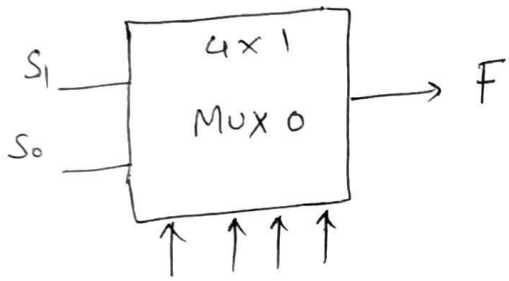


(1.) Each register has 4 bits with number 0, 1, 2, 3 and two select i/p S_1 & S_0.

(2.) The 16 o/p's of registers are connected to the i/p of the multiplexers. for eg Mux 3



D_3 C_3 B_3 A_3 ← (o/p s of registers A, B, C, D last bits)



D_0 C_0 B_0 A_0 ← (o/ps of register A, B, C, D 1st bit of registers)

(3.) Here MUX 0 multiplexes the four '0' bits of registers (A, B, C, D) and so on.

Select lines S_1 & S_0 choose the four bits of a register

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

when $S_1 S_0 = 00$, the 0 data i/p's of all four MUX

are selected and applied to the o/p s that form the bus. The bus lines receive the content of register A

Hence a common bus for 8 registers of 16 bits each requires 16 MUX (one for each line in the bus). with 3 select lines (S_2 S_1 S_0) in each MUX.

Three-state Bus Buffers :→

A bus system can be constructed with 3-state gates instead of multiplexers.

Three-state gate is a digital ckt that exhibits three states :→ (1) logic 1, (2) logic 0 (3) high-impedance state.

High-impedance state behaves like an open ckt which means, o/p is disconnected. Symbol is shown fig-1

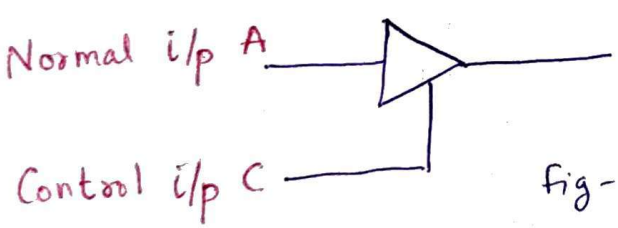


fig-1

o/p Y = A if C = 1

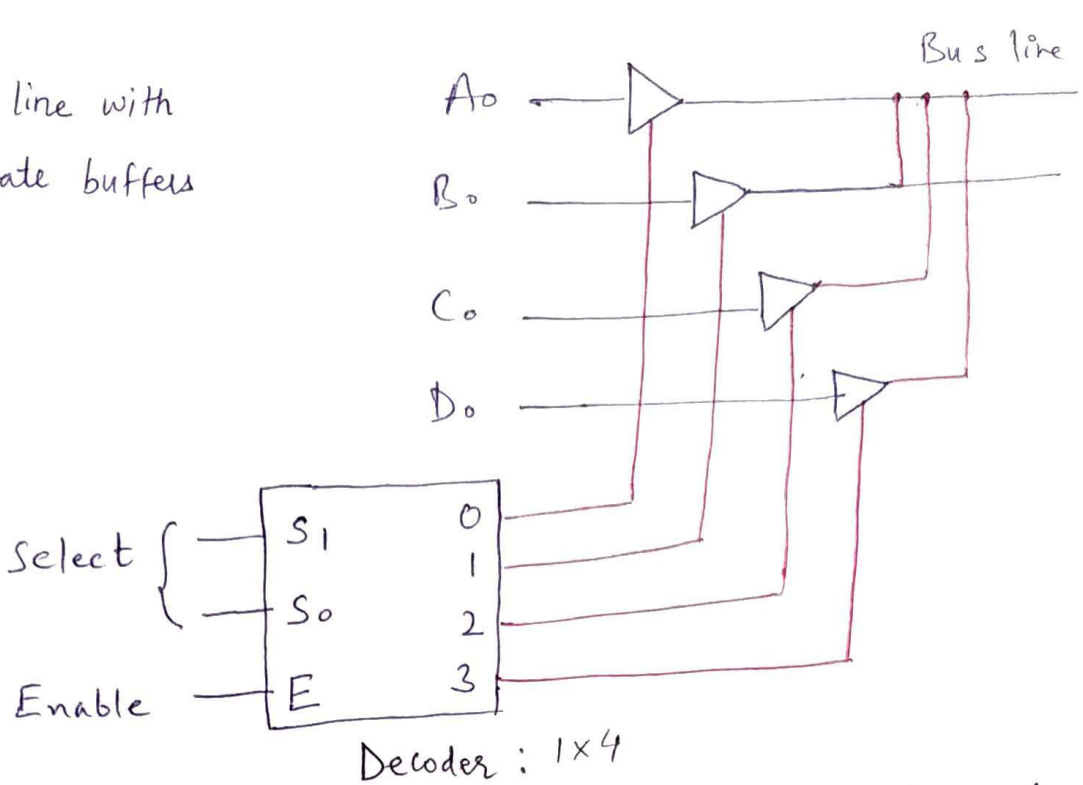
High impedance if C = 0

Construction of a bus system with 3 state buffers is shown in fig. 2.

If i/p = 1, then o/p is enabled & gate behaves like a conventional buffer (ie. o/p Y = A same as i/p)

If i/p = 0, then o/p is disabled and gate goes to a high impedance state.

Bus line with 3-state buffers



Decoder : 1×4

The o/p of four buffers are connected together to form a single bus line. Only one buffer will be in active state at any given time. So a decoder controls this condition

→ When Enable = 0, all the 4 o/p's are 0 the bus line is in a high-impedance state.

→ When Enable = 1. only one of the state buffers will be active.

Memory Transfer →

Transfer of information from a memory word to the outside environment is called a Read operation

Transfer of new information to be stored into the memory is called a write operation.

Symbol for memory word is letter M.

Eg → Consider a memory unit that receives the address from a register, called Address Register Symbol : AR. Let DR denotes, data transfered to another register called Data Register.

The Read operation is given as : →

Read : DR ← M[AR]

* It means transfer of information into DR from the memory word M selected by the address in AR

Write operation is →

Write : M[AR] ← R1

This will transfer information from R1 register into the memory word M selected by the address in AR.

Arithmetic Micro operations : →

A micro operation is an elementary operation performed with the data stored in registers. Micro operations are classified into 4 categories: →

1. Register transfer micro operations transfer binary information from one register to another.
2. Arithmetic micro operations perform arithmetic operation on numeric data stored in registers.
3. Logic micro operations perform bit manipulation operations on non numeric data stored in registers.

4. Shift microoperations perform shift operations on data stored in registers.

The Register Transfer micro operation does not change the information content while arithmetic Logic & shift microoperations change the information content during transfer.

eg. Consider arithmetic micro operation →

$$R3 \leftarrow R1 + R2$$

It specifies an add microoperation and contents of register R1 are added to the contents of Register R_2 and the sum (R1+R2) is transferred to Register R3

Subtract Microoperation :→

is implemented through complement & addition

$R1 - R2$

$$R3 \leftarrow R1 + \overline{R2} + 1 \quad = R1 - R2$$

($\overline{R2} + 1$: 2's complement)

where $\overline{R2}$ is 1's compliment of R2.

Add 1 to 1's compliment ($\overline{R2}$) results into 2's compl.

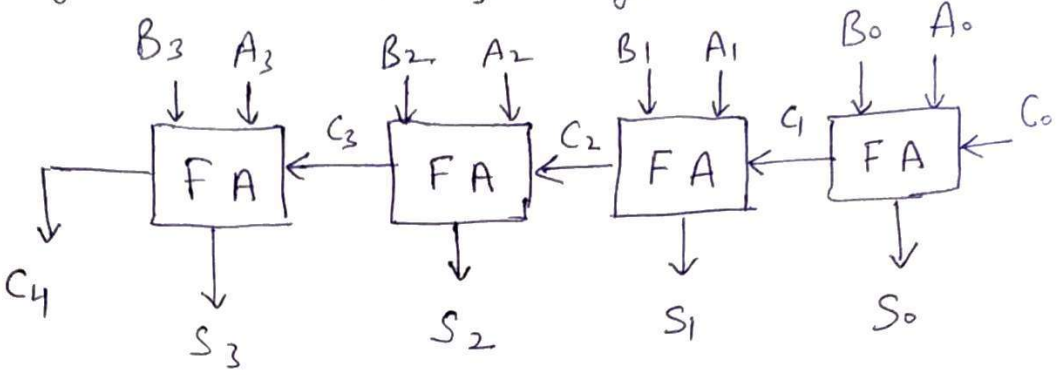
Hence, Subtraction of $R1 - R2 = R1 + (2\text{'s compl. } R2)$

Increment microoperation :→ $R1 \leftarrow R1 + 1$

indicates the increment of 1 in the contents of R1 Register

Decrement :→ $R1 \leftarrow R1 - 1$

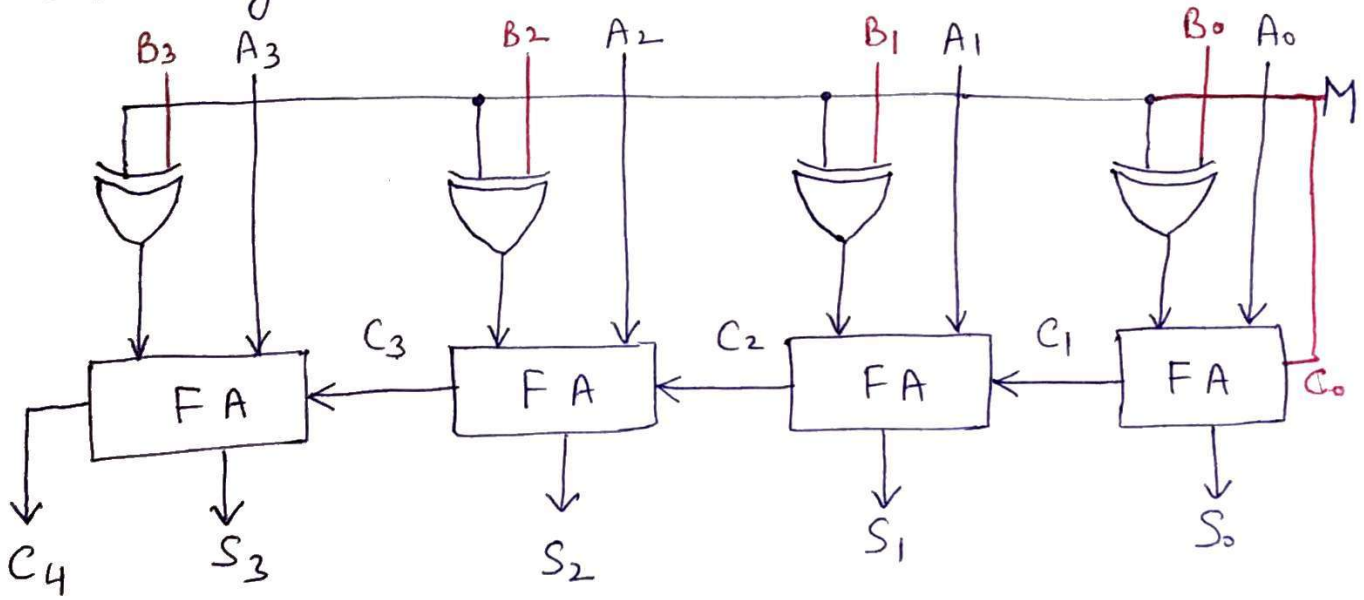
Binary Adder :→ It generates the arithmetic sum of two binary numbers of any length.



Here register R1 may be used to store the n-bit data for A i/ps and R2 for B i/p's. Sum of A & B can be stored in third Register R3.

Binary Adder - Subtractor :→

The addition & subtraction operations can be combined into one common ckt using an Ex-OR gate with each full-adder, shown in fig→



If M = 0 ckt behaves an adder

M = 1 " " as a subtractor

When $M=0$, each XOR gate i/p $0 \oplus B = B$ & the ckt performs $A + B$

When $M=1$, XOR gate i/p is $1 \oplus B = \overline{B}$

and $C_0 = 1$, ckt performs $A + \overline{B} + 1 = A - B$

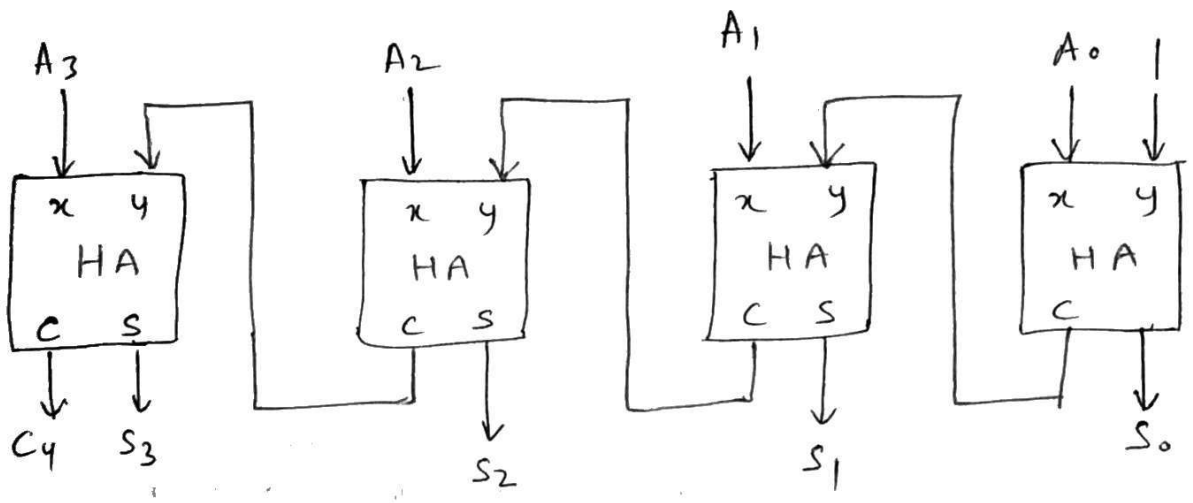
$\overline{B} + 1$ is 2's compliment of B.

Binary Incrementer :→

If a 4bit register has a binary value 0110. Add 1

$$\begin{array}{r} 0110 \\ +\ 1 \\ \hline 0111 \end{array}$$

The dig of 4-bit incrementer ckt can be designed with half-adder (HA)



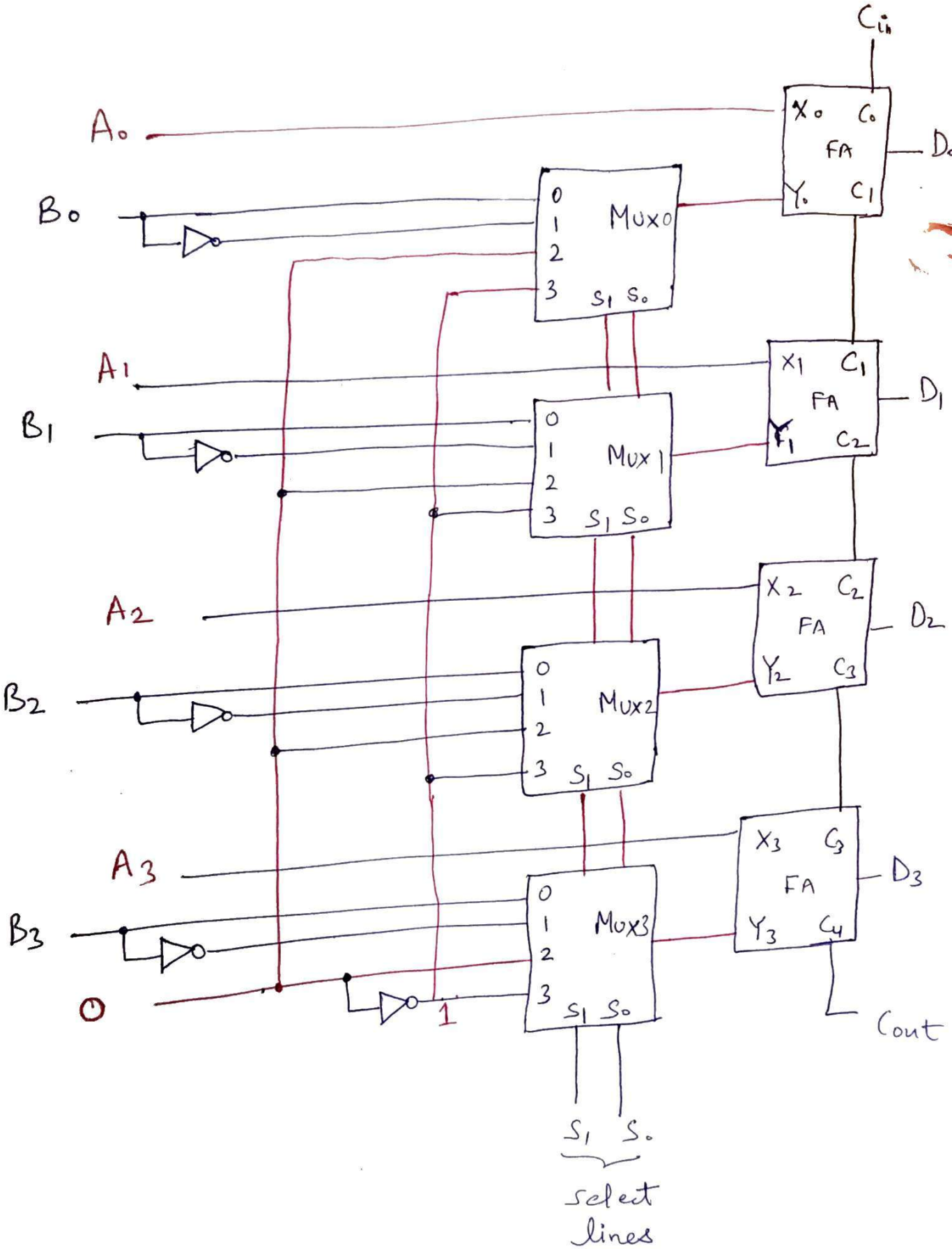
$A_3\ A_2\ A_1\ A_0$ ← Register R_1 (data of A)

$+\ 1$

$S_3\ S_2\ S_1\ S_0$

o/p carry C_4 will be 1 only after $S_3 S_2 S_1 S_0 = 0000$ and $A_3 A_2 A_1 A_0$ is 1111.

Arithmetic circuit :→ using FA and MU a 4-bit arithmetic ckt is shown in fig →



This ckt has 4 bit adder & 4 Mux for choosing different operations. Addition is performed by :→

C_3 C_2 C_1 C_0 carry

A_3 A_2 A_1 A_0 ← 4bit i/p A

\+ B_3 B_2 B_1 B_0 ← 4bit i/p B

D_3 D_2 D_1 D_0 o/p sum as D

One Mux is selected from these 4 Mux with $S_1 S_0$ select lines eg. if $S_1 S_0 = 00$, Mux 0 is selected Inputs of B are connected with Mux 0 as B_0, B_1, B_2, B_3

The output D of the binary adder is :→

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

where A is the 4-bit binary no. at the X inputs and Y is the 4-bit binary no. at the Y inputs

It is possible to generate arithmetic operations from this ckt, like addition, subtraction, complement 2's compliment, increment given below :→

$R3 \leftarrow R1 + R2$ ← addition

$R3 \leftarrow R1 - R2$ ← subtraction

$R2 \leftarrow \overline{R2}$ ← compliment

$R2 \leftarrow \overline{R2} + 1$ ← 2's compliment

$R3 \leftarrow R1 + \overline{R2} + 1$ ← subtraction using 2's compliment

$R1 \leftarrow R1 + 1$ ← increment

$R1 \leftarrow R1 - 1$ ← decrement

Eight Arithmetic operations from the ckt can be ...

Select			input	o/p	Micro operation
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + \bar{B} + 1$	Add with carry
0	1	0	$\bar{B}$	$D = A + \bar{B}$	subtract with borrow
0	1	1	$\bar{B}$	$D = A + \bar{B} + 1$	subtract
1	0	0	0	$D = A$	Transfer A
1	0	1	0	$D = A + 1$	Increment A
1	1	0	1	$D = A - 1$	Decrement A
1	1	1	1	$D = A$	Transfer A

When $S_1 S_0 = 00$, the value of B is applied to the Y inputs of the adder. If $C_{in} = 0$, o/p $D = A + B$. If $C_{in} = 1$, o/p $D = A + B + 1$, both perform add microoperation with / without adding the i/p carry.

Logic Microoperations

The XOR microoperation with the contents of two registers R1 & R2 is symbolized by the statement

P: $R1 \leftarrow R1 \oplus R2$

Eg. if 1010 is content of R1

$\oplus$ 1100 is " " R2

0110 is " " R1 after P=1

Contents of Register R1 is result of XOR.

Special symbols → symbol $\vee$ denotes OR microoperation

Symbol $\wedge$ denotes AND micro operation.

When '+' occurs between registers it is addition

* If '+' occurs in a control function it is OR operation

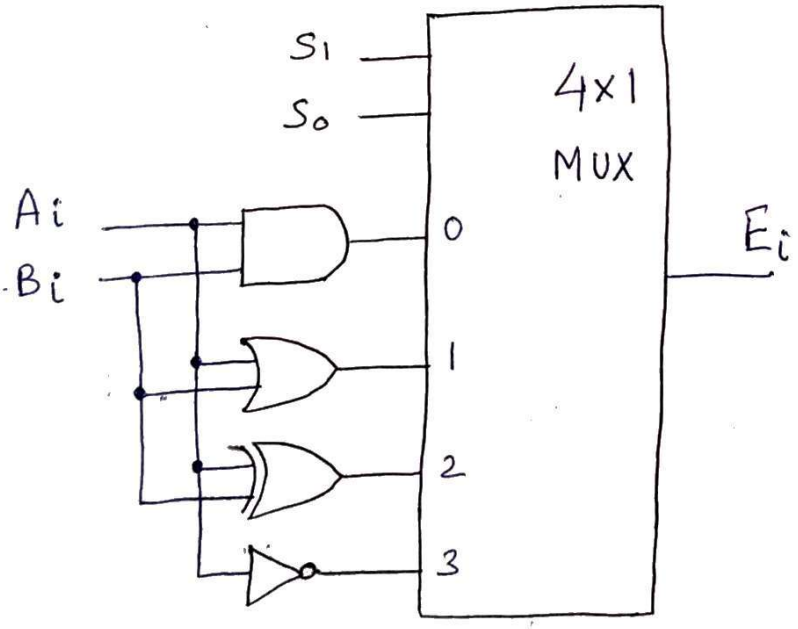
Eg. $P + Q : R1 \leftarrow R2 + R3 \;,\; R4 \leftarrow R5 \vee R6$

($+$ in $P+Q$: OR; $+$ in $R2+R3$: Add; $\vee$ in $R5 \vee R6$: OR)

Hardware Implementation

Most computers use only four logic micro operations AND, OR, XOR and compliment :

Dig. of a one stage circuit to generate four basic logic microoperations is shown :→



The o/ps of the gates are applied to the data i/ps of the MUX. For a logic ckt with n-bits this dig will be repeated n times for $i = 0, 1, 2 \ldots n-1$

Function Table →

S_1	S_0	output	operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	XOR
1	1	$E = \overline{A}$	Complement

Shift Microoperations :→

It is used for serial data transfer. The contents of a register can be shifted to the left/right. During a shift-right operations the serial i/p transfers a bit into the leftmost position.

3 types of shifts are :→ logical, circular, arithmetic.

(1) $R1 \leftarrow shl\ R1$ — shl – shift-left
$R2 \leftarrow shr\ R2$ — shr – shift-right

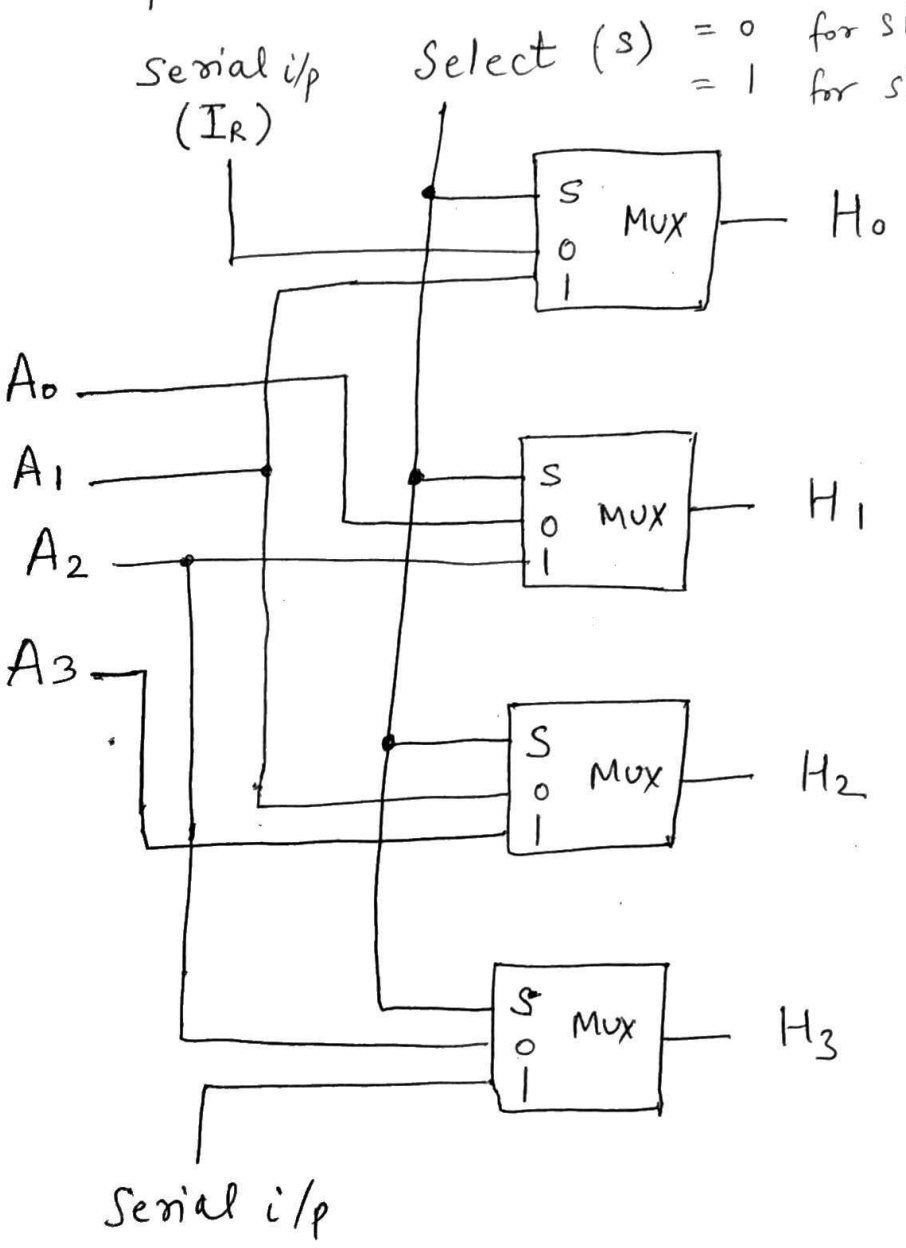
(2) $R1 \leftarrow cil\ R1$ — cil – circular shift left register R1
$R2 \leftarrow cir\ R2$ — cir – cir. shift right

(3) $R1 \leftarrow ashl\ R1$ — ashl – arithmetic shift left
$R2 \leftarrow ashl\ R2$ — ashl – arith. shift right

Hardware can be implemented with a combinational ckt shifter as shown in fig →

The 4-bit shifter has four data i/p's, A_0 through A_3 and four data o/p's, H_0 — H_3. There are 2 serial i/p's : one for shift left (I_L) and

the other for shift right (I_L). When the selection i/p $S=0$, the i/p data are shifted right (In dig downwards) when $S=1$, i/p data are shifted left (upwards in dig) A shifter with n-data i/p's and o/p's requires n multiplexers.



Four-bit combinational circular shifter

Function Table

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

Chapter - 5 Basic Computer Organization and Design (Morris Mano)

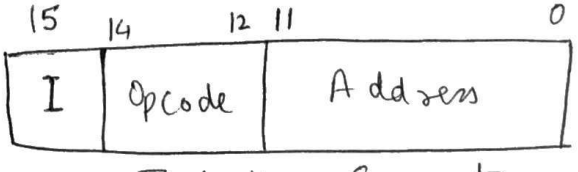
Program → It is a set of instructions that specify the operations, operands and the sequence by which processing has to occur.

Instruction code :→ is a group of bits that instruct the computer to perform a specific operation. The basic part of instruction is operation code (op code). It is a group of bits that define such operations as add, subtract, multiply, shift and complement.

Stored Program Organization :→

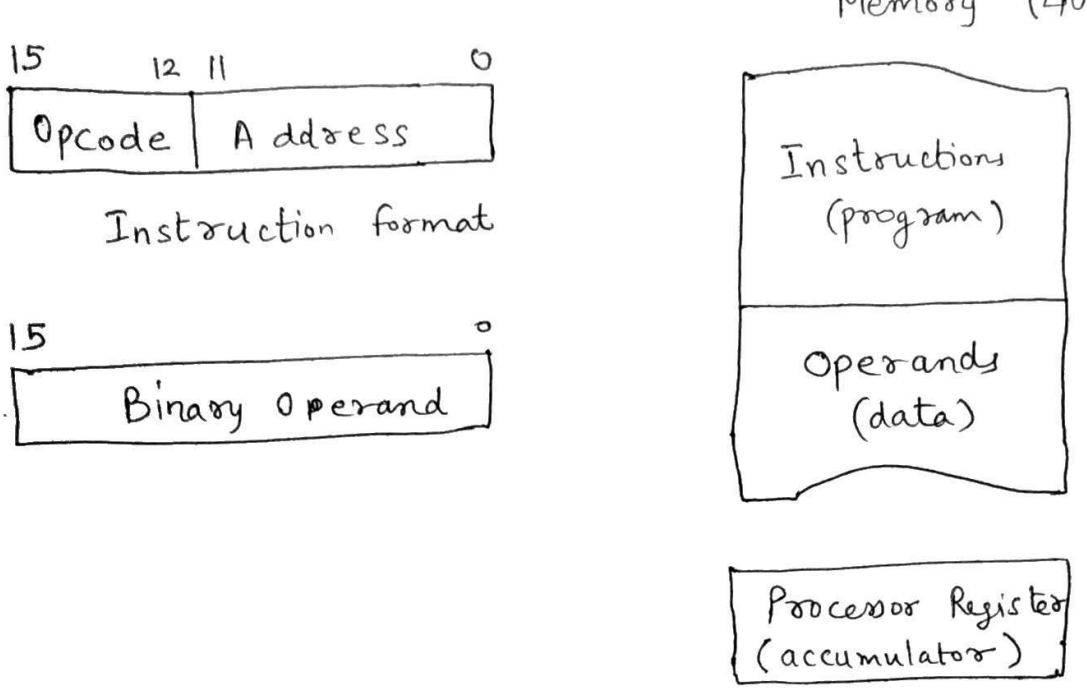
Computer is organized with one processor register (accumulator) and an instruction code format 1st part (Process Register) specifies the operation to be performed, 2nd part instruction code specifies an address.

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 togther with the data stored in the process register.



Instruction format

Fig-1 Stored Program Organization

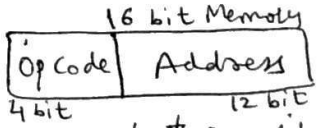


Memory has a one section of instructions to be stored and another section of data to be stored.

For a memory unit with 4096 words we need 2^{12} ie 12 bits to specify an address

for eg for a memory word with 16-bit :→

4 bits are used by opcode and 12 bits to specify an address of an operand.

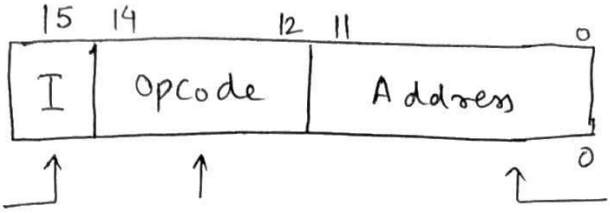


Accumulator (AC) : Single-processor register is called an accumulator or AC.

operations such as clear AC, complement AC, increment AC does not need an operand from memory.

Direct & Indirect Address →

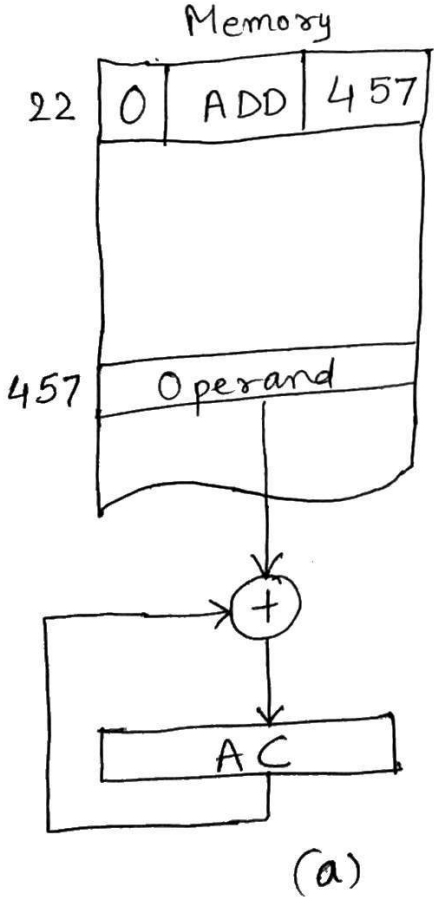
Consider the instruction code format shown in fig →



I = 0 for a direct address mode

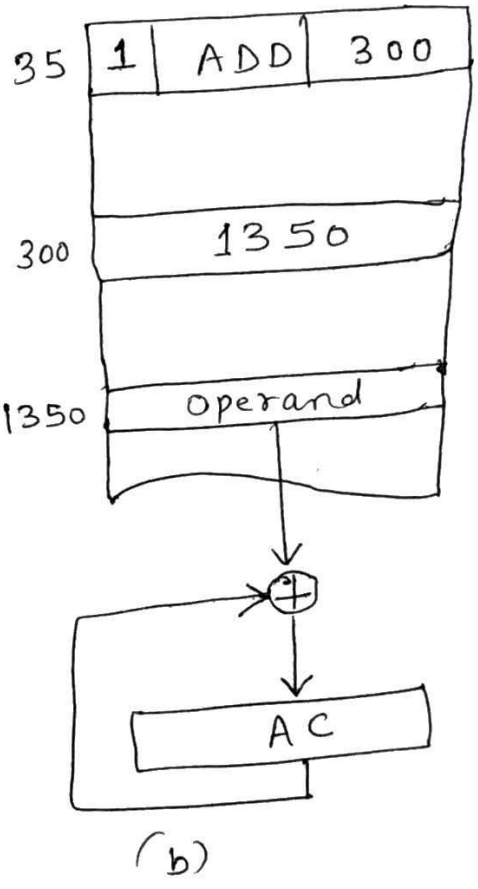
I = 1 for an indirect address mode

Ex : direct address mode:



(a)

Indirect address mode:



(b)

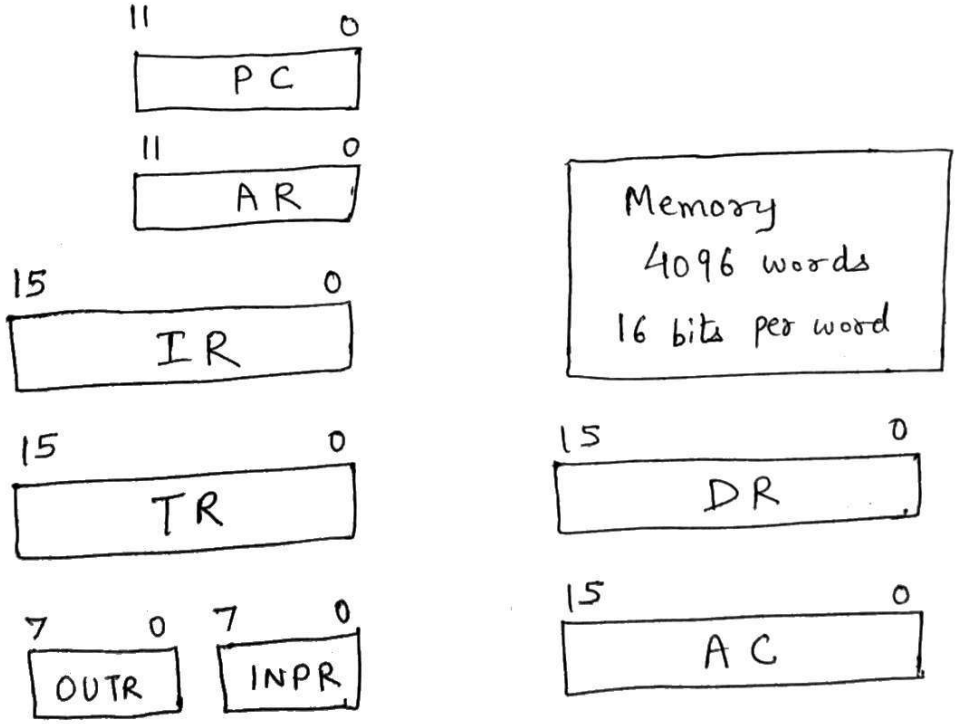
In fig (a) I = 0 it is a direct address instruction. Opcode specifies ADD instruction. The control finds the operand in memory at address 457 & adds it to the content of AC.

In fig b. I = 1, it is an indirect address instruction. The address part is the binary equivalent of 300. The control goes to address 300 to find the address of the operand. ie 1350. The operand found in address 1350 is then added to the content of AC.

Effective address :→ fig(a) 457, it is first reference to read the address of the operand in direct instruction fig(b). 1350

Computer Registers →

The computer needs processor registers for manipulating data and a register for holding a memory address. The basic computer register are shown →

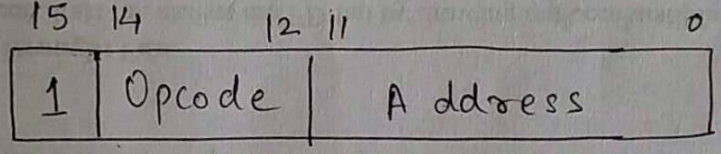


DR – Data Register, 16 bits, holds memory operand
AR – Address Register, 12 bits, holds address for memory

AC - Accumulato Reg. , 16 bits , Processor Register
IR - Instruction Reg. , 16 bits , holds instruction code
PC - Program counter , 12 bits , holds address of instruction
TR - Temporary register , 16 bits , holds temporary data
INPR - Input reg. , 8 bits holds i/p character
OUTR - o/p reg. , 8 bits holds o/p character

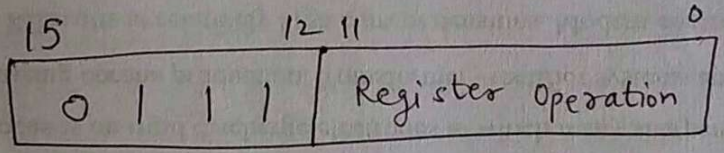
Computer Instructions

3 instruction code formats each of 16 bits are



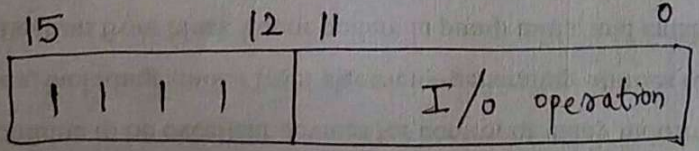
(Opcode = 000 through 110)

(a) Memory - reference instruction



(Opcode = 111, I=0)

(b) Register - reference instruction



(Opcode = 111, I=1)

(c) Input - output instruction

(a) A memory - reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I = 0 for direct address and I = 1 Indirect

(b) Register reference instructions are recognized by the operation code 111 with a 0 in the left most bit (bit 15) of the instruction while remaining 12 bits are used to specify the type of i/p - o/p operation

Type of instruction is recognized by the computer control from the 4 bits ie 12 – 15. If the 3 opcode bits ie 12 – 14 are not equal to 111, the instruction is a memory reference type and the bit in position 15 is taken as the addressing mode I.

Instructions are listed in table :→

Symbol	I=0	I=1	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
BUN			Branch unconditionally
BSA			Branch & save return address
ISZ			Increment and skip if zero.
CLA			clear AC
CLE			clear E
CMA			Complement AC
INC			Increment AC
HLT			Halt computer
INP			Input character to AC
OUT			output character to AC.

Timing and Control

A master clock generator controls the timing for all registers in the basic computer.
The clock pulses do not change the state of a register unless the register is enabled by a control signal.
There are 2 major types of control organization:
(1) Hardwired control (2) Microprogrammed control

→ In Hardwired control, logic is implemented with gates, F/F's, decoders, digital clcts.

→ In microprogrammed organization, the control information is stored in a control memory.

Block dig of a control unit

It has 2 decoders, a sequence counter, a no. of control logic gates. An instruction read from memory is placed in the Instruction Register (IR). Position of IR register in the common bus system is shown in block dig.

IR consists of 3 parts: I bit, Opcode, bits (0–11)
15 bit (12–14) bits

The Opcode in bits 12–14 are decoded with a 3 x 8 decoder with o/p s $D_0 - D_7$
Bit 15 of the instruction is transfered to a F/F
Bits 0–11 are applied to the control logic gates.

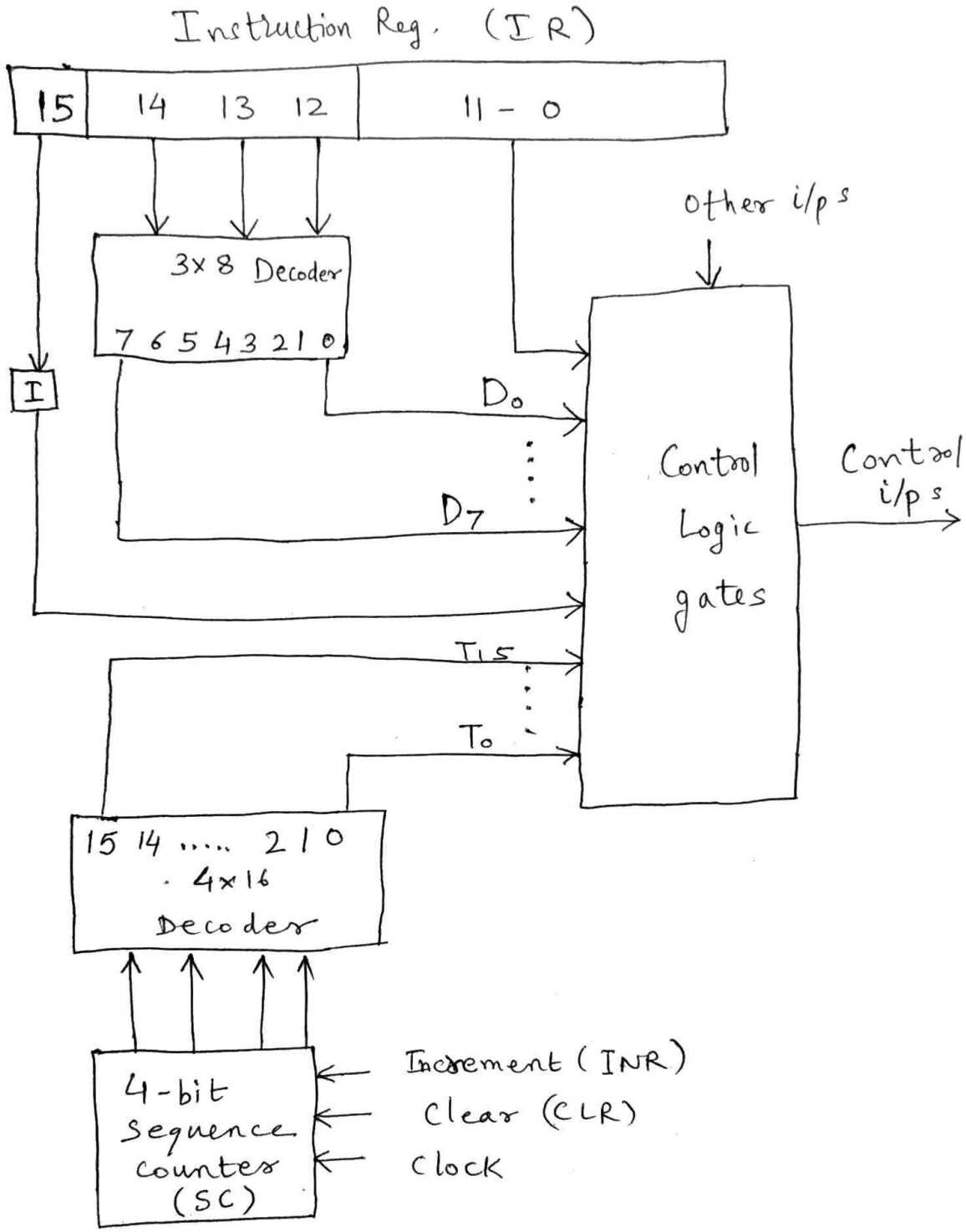
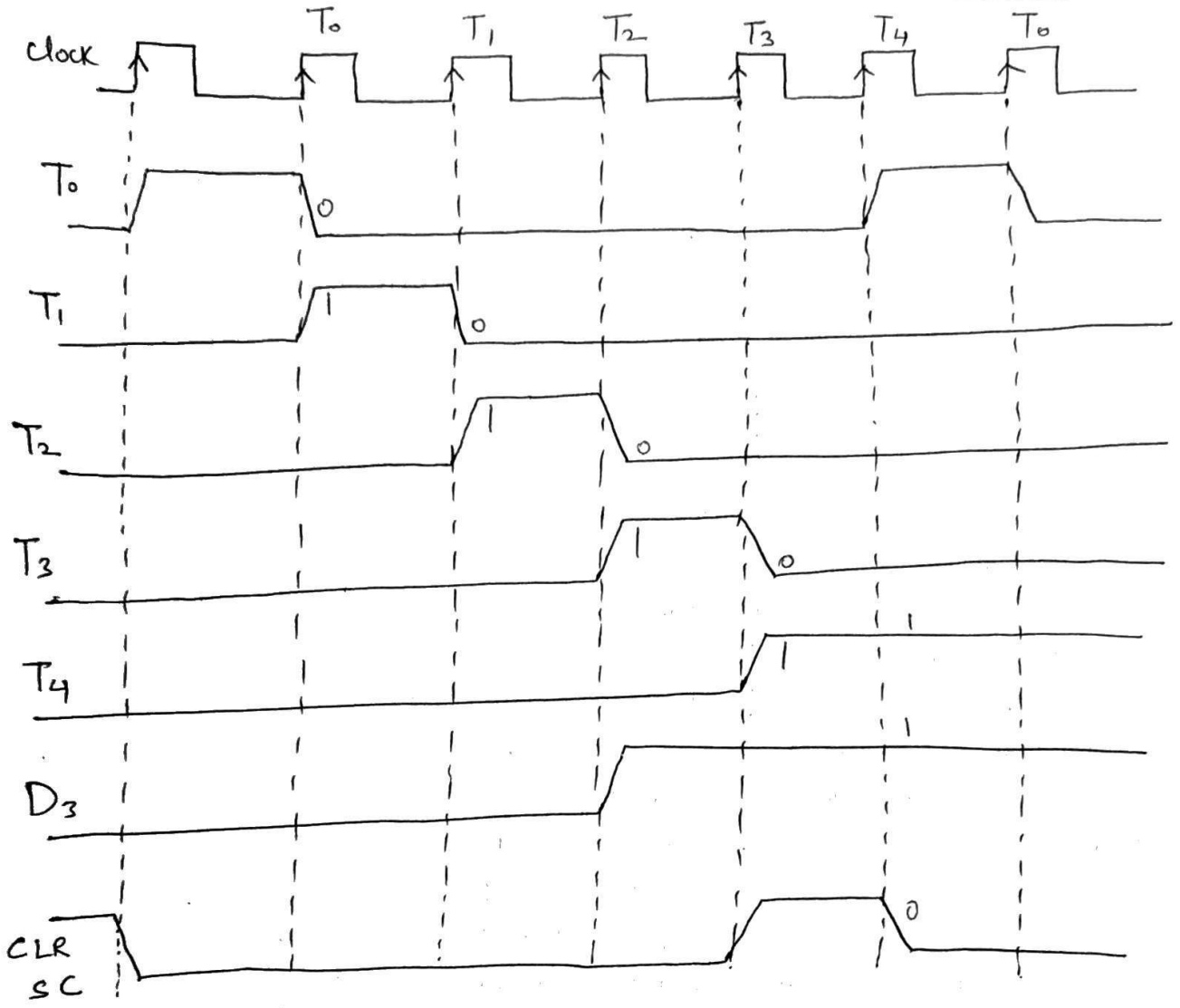


Fig.A Control unit of basic computer

4-bit sequence counter (SC) can count in binary (0 - 15) o/p's of the counter are decoded into 16 timing signals $T_0 - T_{15}$.

SC is incremented to provide timing signals T_0, T_1, T_2, T_3 and T_4 in sequence.

Timing dig for relationship of the control signals →



At time T_4, SC is cleared to 0 if decoder o/p D_3 is active. This is expressed by statement

$$D_3 T_4 : SC \leftarrow 0$$

Initially, CLR i/p of SC is active and 1st +ve transition of the clock clears SC to 0. This activates the timing signal T_0. Registers connected to T_0 will be triggered & SC is incremented for every +ve clock. Hence sequence of timing signals are produced ie. T_0, T_1, T_2, T_3, T_4 and so on.

If timing signal is not cleared, it will continue with $T_5, T_6 \ldots\ldots T_{15}$ and back to T_0.

operation $D_3 T_4 = 1$ is implemented with AND gate.

Instruction Cycle

In the basic computer each instruction cycle consists of the following phases:→

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.

1) Fetch and Decode

Initially PC (Prog. Counter) is loaded with the address of the 1st instruction in the prog. SC is cleared to 0 with signal timing T_0, After each clock pulse, SC is incremented by 1, so sequence $T_0, T_1, T_2 \ldots\ldots$ goes on.

Microoperations for Fetch & decode phase can be explained with register transfer :→

T_0 : $AR \leftarrow AC$

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

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

During clock transition T_0 : Transfer of address from PC to AR, because only AR is connected to the address i/p's of memory.

At T_1: Instruction Read from Memory is then placed in the register IR (instruction Reg.)

Also PC is incremented by 1

At T_2: Operation code in IR is decoded, the indirect bit is transferred to F/F I and the address part of the instruction is transfered to AR, ie $AR \leftarrow IR(0-11)$

Also SC is incremented after each clock pulse, T_0, T_1, T_2

-In fig → Register Transfer is implemented at T_0, T_1 →

(1) Place the content of PC onto the bus by making the bus selection i/p $S_2 S_1 S_0 = 010$ due to T_0 is connected with S_1

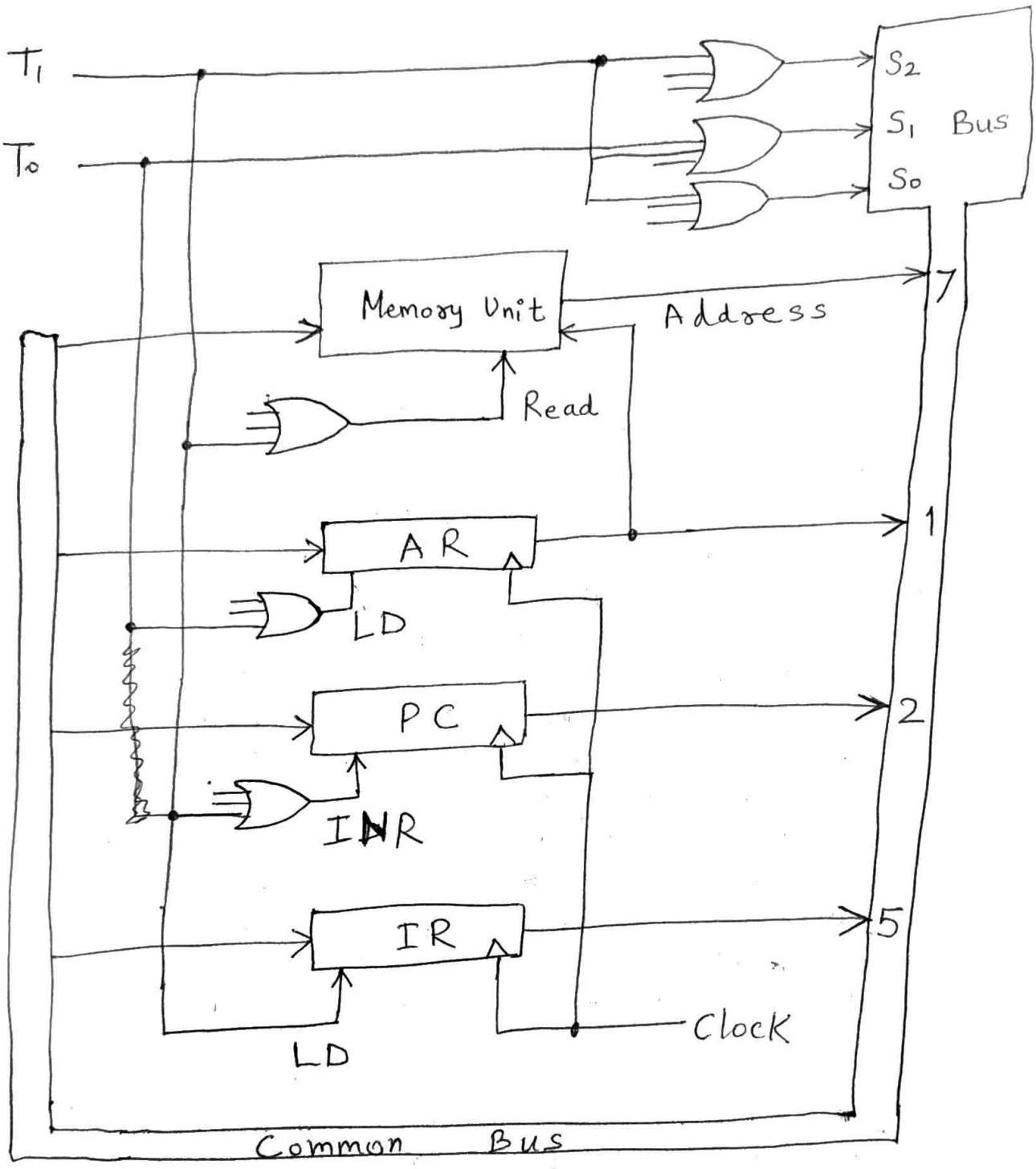
(2) Transfer the content of the bus to AR by enabling the LD i/p of AR

The next clock initiates the transfer from PC to AR, since $T_0 = 1$. Second statement is implemented ie.

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

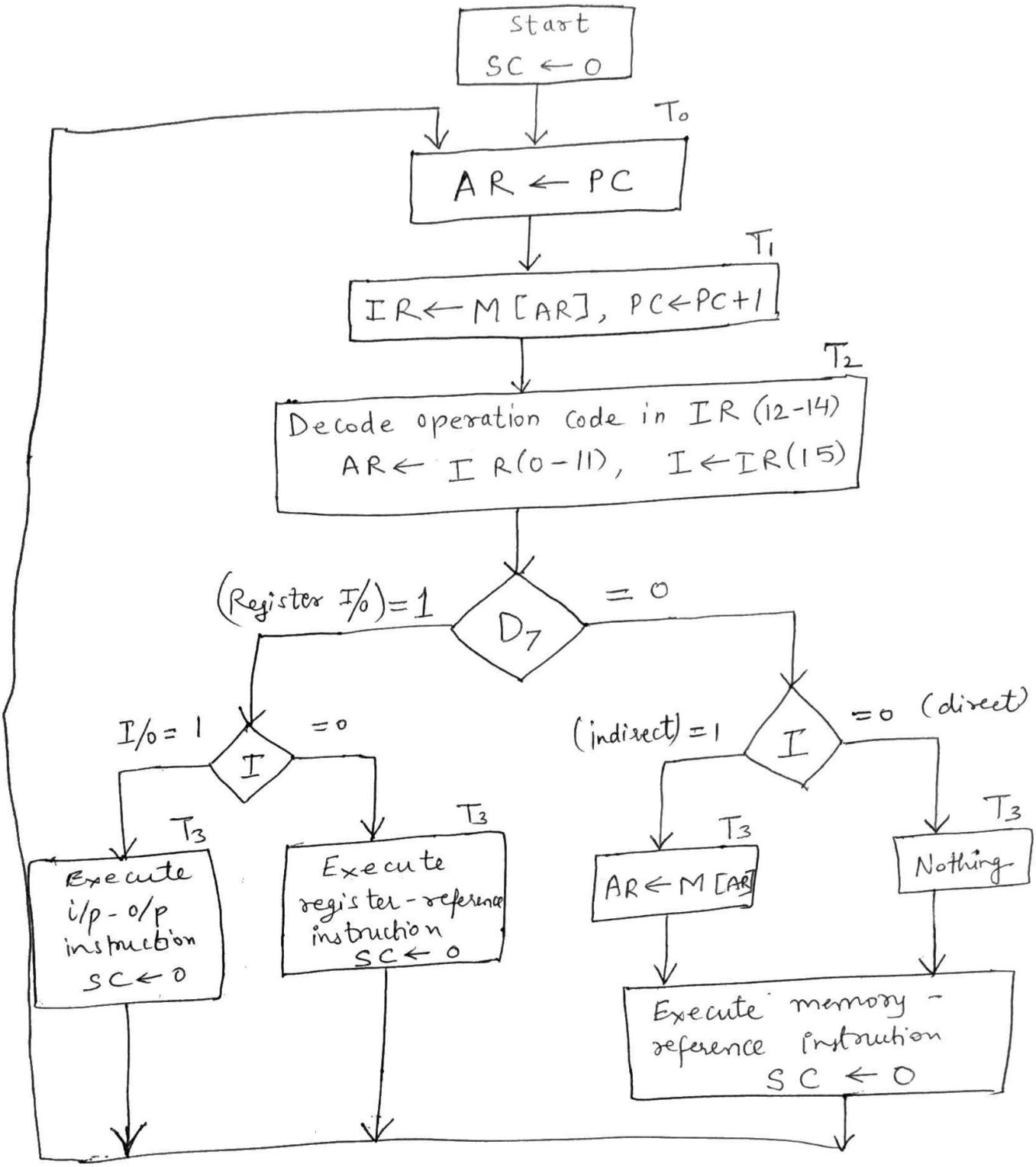
timing signal T_1 provides the following connections in the bus system →

Fig Register Transfers for the Fetch phase



1. Enable the read i/p of memory
2. Place the content of memory onto the bus by making $S_2 S_1 S_0 = 111$
3. Transfer the content of the bus to IR by enabling the LD i/p of IR.
4. Increment PC by enabling the INR i/p of PC.

Flow chart for instruction cycle :→



If $D_7 = 0$ & $I = 1$, we have a memory-reference instruction with an indirect address, register transfer occurs

If $D_7 = 1$ & $I = 0$, register-reference instruction is recognized by the control

List of Register-Reference Instructions.

CLA	→	clear AC
CLE	→	clear E
CMA	→	Complement AC
CME	—	Complement E
CIR	—	Circulate right
CIL	—	Circulate left
INC	—	Increment AC
SPA	—	Skip if +ve
SNA	—	Skip if −ve
SZA	—	Skip if AC = 0
SZE	—	Skip if E = 0
HLT	—	Halt computer

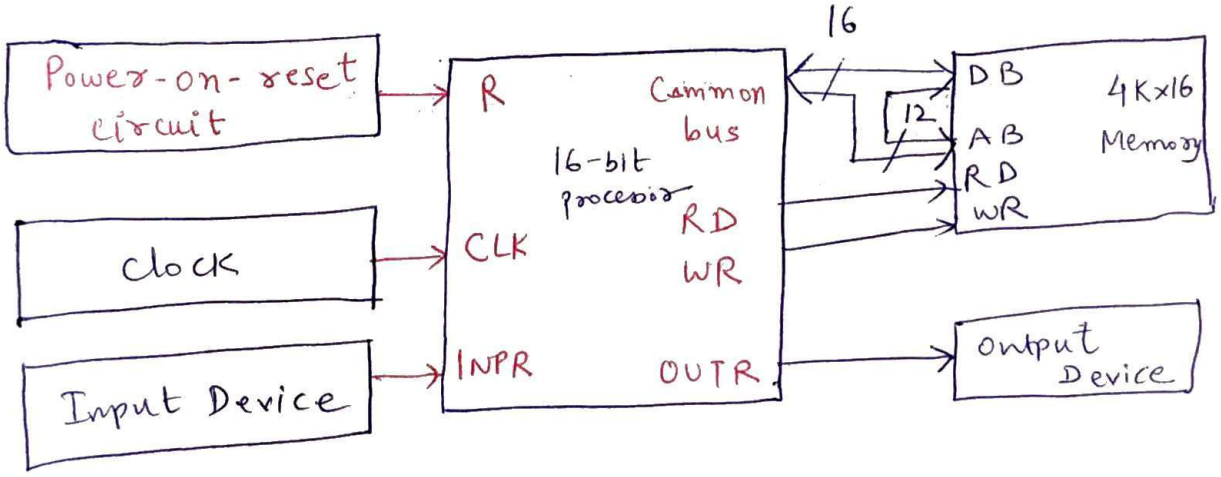
Memory-Reference Instructions

	operation decoder	
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$
LDA	D_2	$AC \leftarrow M[AR]$
STA	D_3	$M[AR] \leftarrow AC$
BUN	D_4	$PC \leftarrow AR$
BSA	D_5	$M[AR] \leftarrow PC$, $PC \leftarrow AR+1$
ISZ	D_6	$M[AR] \leftarrow M[AR]+1$

Design of Basic Computer :→

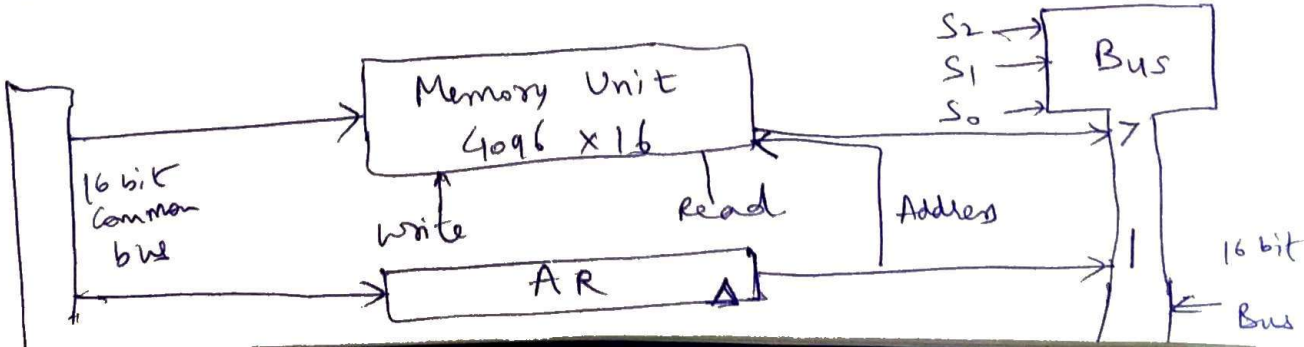
The basic computer consists of the following hardware components

1. A memory unit with 4096 words of 16 bits each.
2. Nine registers ; AP, PC, DR, AC, IR, TR, OUTR, INPR, SC
3. Seven flip-flops : I, S, E, R, IEN, FGI and FGO.
4. Two decoders : 3 × 8 operation decoder
   4 × 16 timing decoder
5. A 16-bit common bus
6. Control logic gates
7. Adder & Logic ckt connected to the i/p of AC.

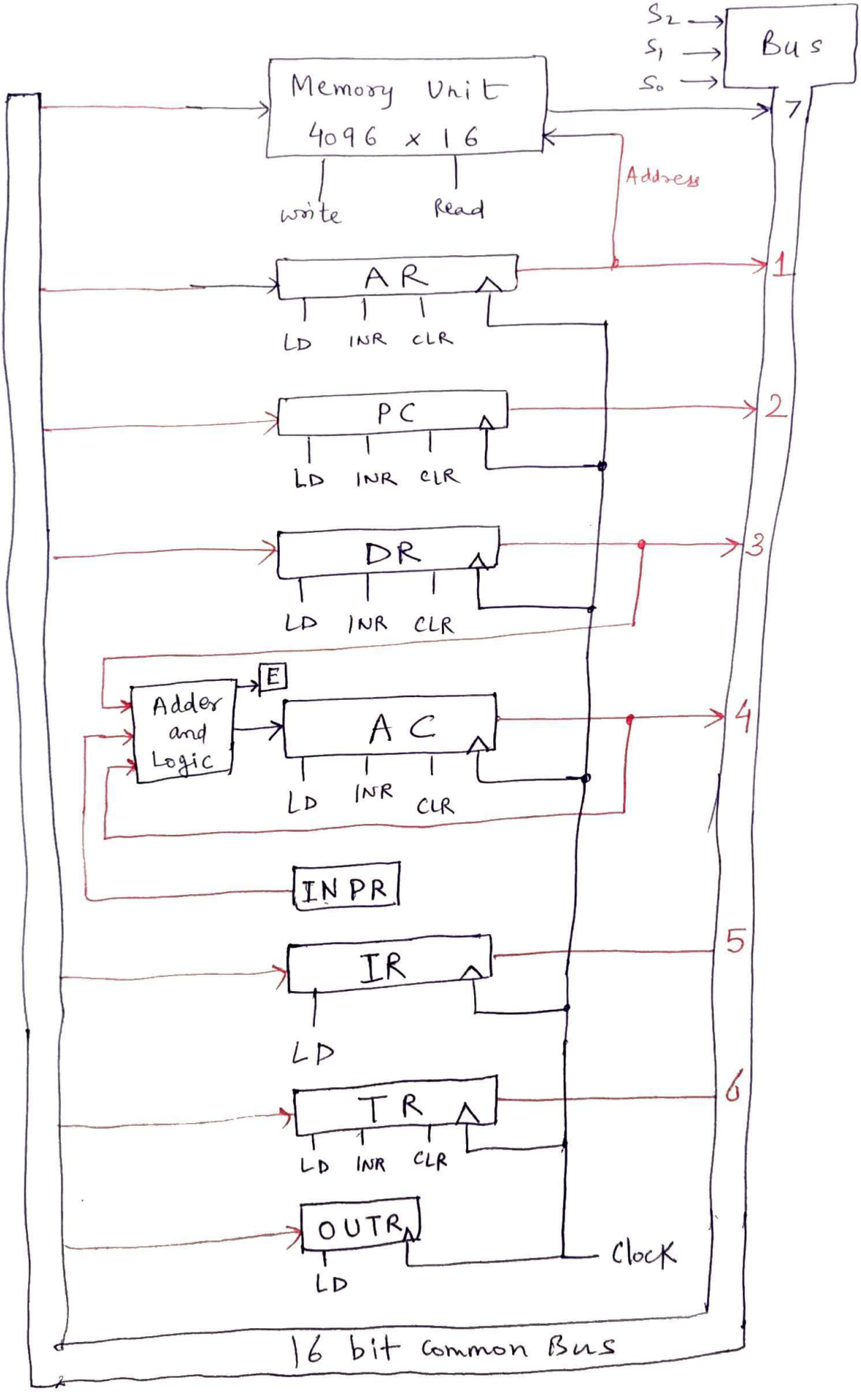


Control of Common Bus :→

The 16-bit common bus shown earlier :→



Connection of the registers & memory of the basic computer to a common bus system is shown :→



Encoder for bus selection ckt is given →

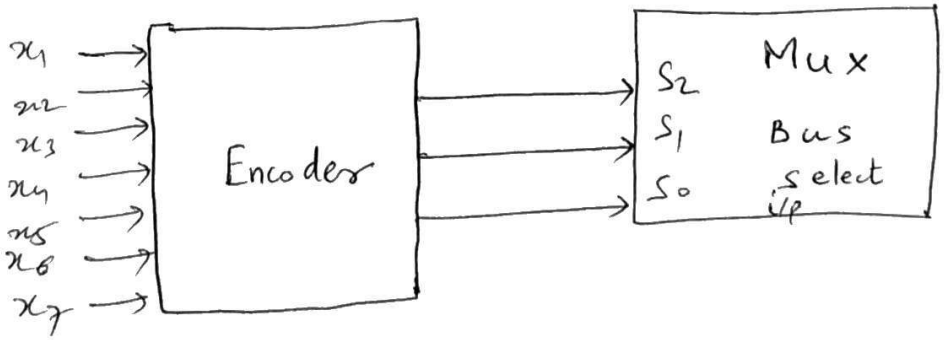
Inputs							Outputs			Register selected for bus
x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	
0	0	0	0	0	0	0	0	0	0	None
1	0	0	0	0	0	0	0	0	1	AR
0	1	0	0	0	0	0	0	1	0	PC
0	0	1	0	0	0	0	0	1	1	DR
0	0	0	1	0	0	0	1	0	0	AC
0	0	0	0	1	0	0	1	0	1	IR
0	0	0	0	0	1	0	1	1	0	TR
0	0	0	0	0	0	1	1	1	1	Memory

Let $x_1 = 1$, the value of $S_2 S_1 S_0 = 001$, the o/p AR will be selected for the bus.

Boolean functions for the encoder are :→

$$S_0 = x_1 + x_3 + x_5 + x_7$$
$$S_1 = x_2 + x_3 + x_6 + x_7$$
$$S_2 = x_4 + x_5 + x_6 + x_7$$

The encoder is placed at the inputs of the bus selection logic is shown below :→



Chapter – 6 Programming

Machine Language :→ A program is a list of instructions or statements for directing the computer to perform a required task.

Computers can execute programs only when they are represented internally in binary

Machine Language :→ is a binary program for a computer to understand.

Assembler :→ Users employ symbols (letters, numerals, characters) for the operation parts address etc of the instruction code. It is to be translated into binary code by assembler.

Assembly Language program :→

The symbolic program translated into binary coded instruction by the assembler is called assembly language prog.

High Level Language :→

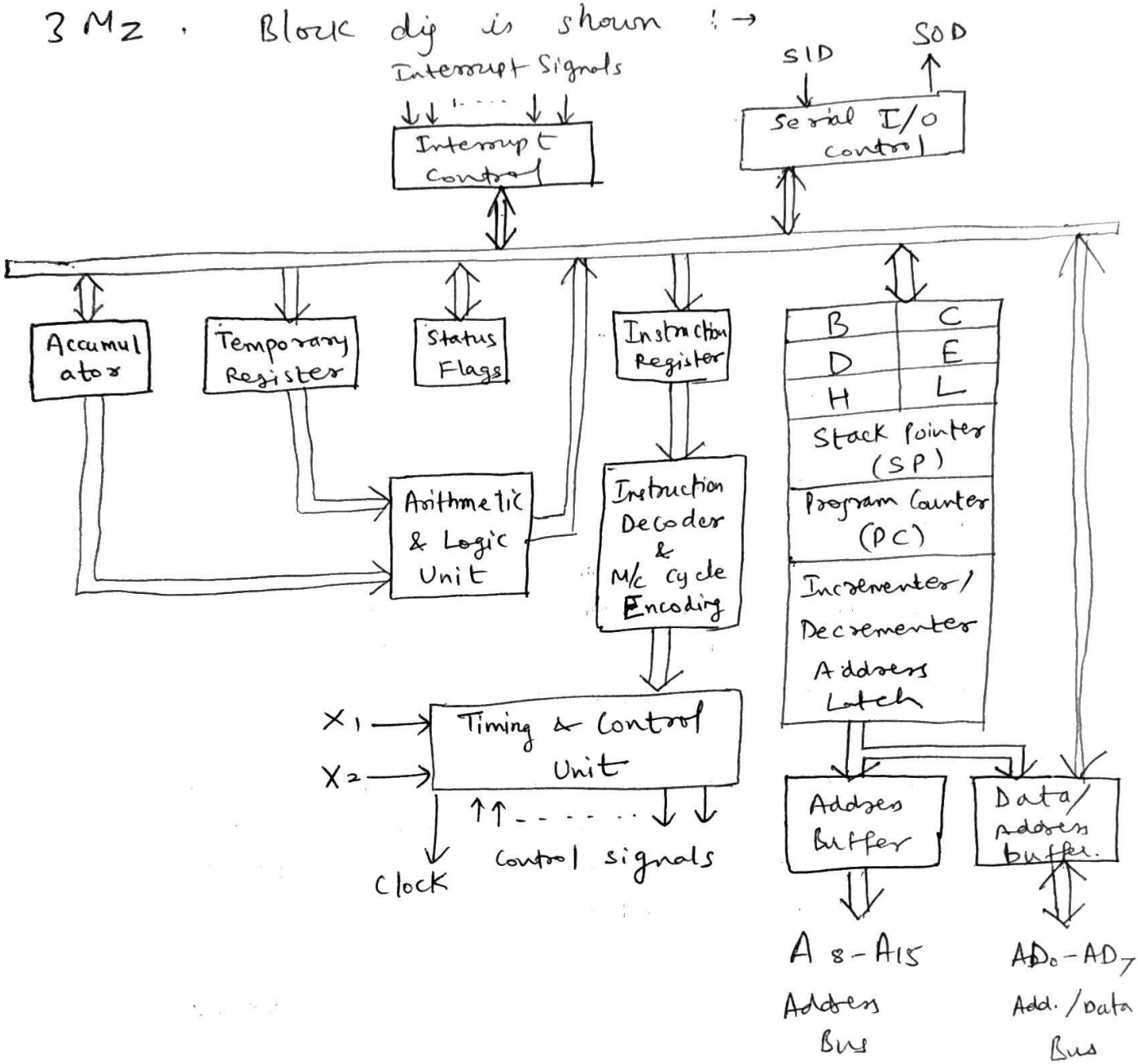
It is a sequence of statements in a form that people prefer to think in solving a problem.

Fortran is a high-level programming Language

Compiler :→ It is a program that translates a high level language into binary code

Assembly Language Programming with 8085 :→

8085 is an 8 bit microprocessor, It is a 40 pin Ic, operates on +5 V dc supply, Clock is about 3 Mz. Block dig is shown :→



Description of Block dig ⇒

(1) ALU → Arithmetic & Logic Unit performs the addition, subtraction, Logical AND, OR, EXOR, complement (NOT), Increment (Add 1), Decrement Left shift, Rotate left - right, clear etc.

(2) Timing & Control Unit → It is a part CPU, to generate timing & control signals which are necessary for the execution of instructions.
It controls data flow b/w CPU and peripherals.
Control Unit of CPU acts as a brain of the computer system.

(3) Registers :→ Registers are used by the microprocessor for temporary storage and manipulation of data and instructions. 8085 registers are :-

(i) One 8 bit Accumulator (ACC) ie register A.
(ii) Six 8-bit general purpose registers B, C, D, E, H and L.
(iii) One 16-bit stack Pointer, SP
(iv) One 16 bit program counter, PC
(v) Instruction Register
(vi) Temporary Register

(i) Accumulator → It is denoted register 'A' or accumulator to hold one of the operands of an arithmetic or logic operation.
It serves as one i/p to the A.L.U. Final result of an arithmetic or logical operation is placed in Accumulator.

(ii) General purpose Registers :→ Six 8 bit registers are B, C, D, E, H and L to hold 16-bit data a combination of two 8 bit registers are used. called as register-pair such as :→
B-C, D E, and H-L.
H-L pair is used as a memory pointer

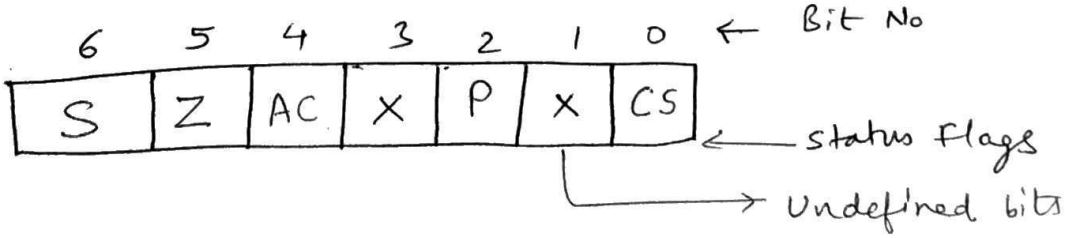
(iii) Prog. Counter (PC) → It is a 16 bit special-purpose register to hold the memory address of the next instruction to be executed. It keeps track of memory address and increments the content of PC during the execution of an instruction.

(iv) Instruction Register (IR) :→ It holds the opcode (operation code) of the instruction to be executed

(v) Temporary Register :→
It is an 8-bit register associated with ALU to hold data during an arithmetic / logical operation.

(vi) Flags :→ 8085 up has 5 f/fs to serve as status flags. f/fs are set/reset

(a) Carry flags (CS)
(b) Parity flag (P)
(c) Auxiliary Carry flag (AC)
(d) Zero flag (Z)
(e) Sign flag (S)



After the execution of an arithmetic instruction if a carry is produced, then CS = 1 (set to 1) otherwise CS = 0.
In case of addition of two 8-bit nos, if sum is larger than 8-bits, a carry is produced. and CS = 1

Similarly in case of subtraction, if borrow occurs, then CS = 1.

Parity Flag, P = 1 (set to 1), if the result of an arithmetic / logical operation contains even no. of 1's. If result contains odd no. of 1's, P = 0 (reset to 0)

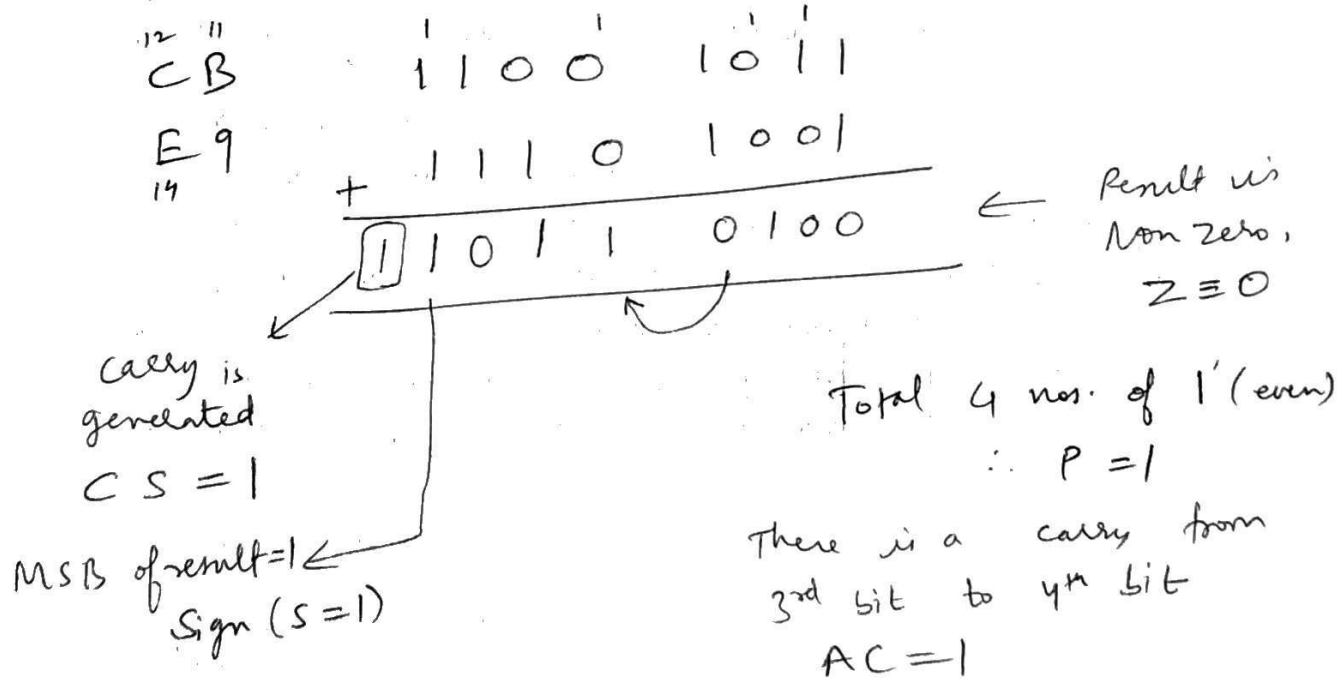
Auxiliary Carry: (AC) holds carry out of the bit no. 3 to the bit no. 4 resulting from the execution of an arithmetic operation.

Zero Flag :→ (Z). If the result of an arithmetic or logical operation is 0, then Z = 1 otherwise Z=0

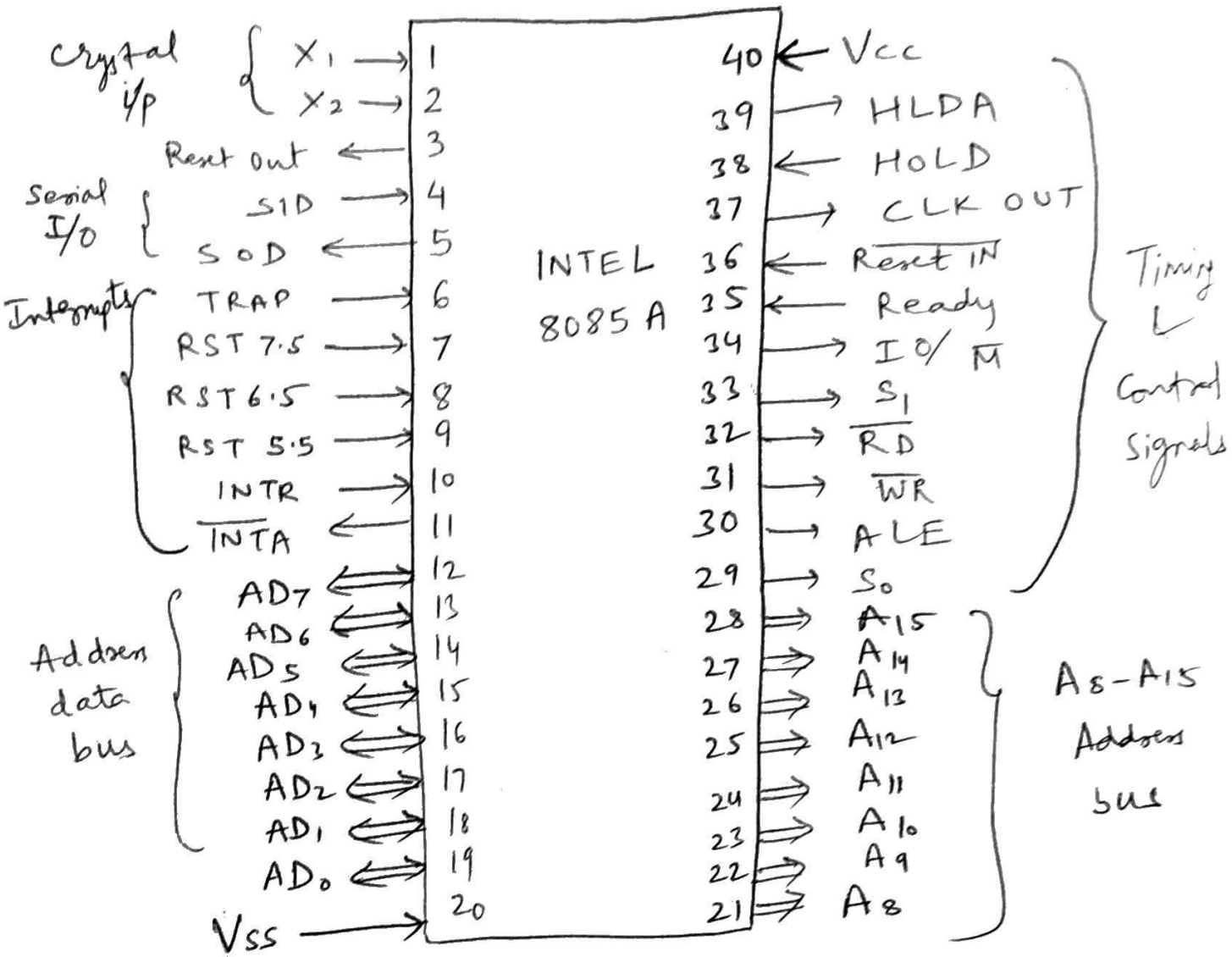
Sign Flags: (S): If the result of an arithmetic and logical operation is -ve, then S = 1, otherwise S = 0.

A B C D E F
10 11 12 13 14 15

Example: , ADD CB , E9



8085 pin configuration :→



A8 – A15 (output) :→ These are Add. bus & used for the MSB of the memory address.

AD0 – AD7 (I/O) :→ These are time multiplexed address/data bus. They are used for the least LSB 8 bits of the memory address during the 1st clock cycle of a machine cycle.

ALE → (O/P) : It is address Latch enable It goes high during 1st clock cycle of machine cycle

and enables the lower 8 bits of the address to be latched either into the memory or external latch.

$\overline{RD}$ (output) :→ when Up ~~sends~~ reads data from a memory location of o/p device, it is called READ operation. It is a low signal operation or logic '0'

$\overline{WR}$ (o/p): when Up sends data to a memory location or o/p device, it is called WRITE operation

HLDA :→ HOLD aknowledge signal sent out by the Up after recieving the HOLD signal. The microprocessor takes over the buses.

INTR :→ when Interrupt signal goes high Up suspends the execution of its instructions.

RST 5.5 ; 6.5, 7.5 and TRAP and $\overline{INTA}$ are the 5 interrupts.

Highest Priority has TRAP & lowest (INTR).

X_1, X_2 are the connected to crystal oscillators to produce a clock signal.

V_{SS} = ground reference

V_{CC} = +5 volts

SID — Data line for serial i/p. It is loaded into the 7 bit of Accumulator when RIM instruction is executed.

SID → Data line for serial o/p.

SIM instruction is executed.

READY → Signal sent by an I/p device to the µp before µp perfoms data transfer operation. When READY is high, means that i/p / o/p device is ready to recieve / send data.

S_0, S_1 are status signals.

S_1	S_0	operations
0	0	Halt
0	1	write
1	0	Read.
1	1	Fetch.

Intel 8085 Instructions →

(1) MOV A,B :→ Move the content of register B to A

Addressing Modes :→

(1) Direct addressing Modes (2) Register addressing (3) Register indirect addressing (4) Immediate addressing mode

(1) Direct Addressing :→ the address of the operand (data) is given in the instruction format.

eg STA 2400H → It stores the content of the accumulator in memory location 2400 H.

Here 2400H is the memory address where data is to be stored.

(2) Register Addressing :→ Here operand (data) is in one of the general purpose registers. The opcode specifies the address of the registers in addition to the operation to be performed.

eg. MOV A, B

It means move the content of register B to A

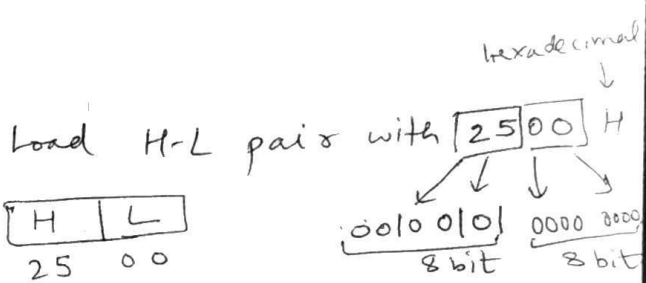
eg. ADD B

It means to Add the content of register B to the corentent of register A.

(3) Register Indirect Addressing :→

Here the address of the operand (data) is specified by a register pair.

eg. LXI H, 2500H Load H-L pair with 2500 H



ADD M Add the content of the memory location whose address is in H-L pair (ie 2500H) to the content of accumulator

HLT Halt program

(4) Immediate Addressing :→

Here the operand is specified with in the instruction itself.

eg. MVI A, 05 Move 05 in register A

eg. ADI 06 Add 06 to the content of the Accumulator

Instructions 8085 µP

(A) Data Transfer : Move Data

(i) MOV A, B Moves the content of register B to A

(ii) MOV B, M Move the content of memory to register B.

eg. LXIH, 200H Load H-L pair by 2000H (hexadecimal value) (H L)

MOV B, M Move the content of the memory location 2000H to register B.

HLT Halt.

Here instruction LXI H, 200H loads H-L pair with 2000H which is address of a memory location, then contents of memory location 2000H is moved to register B.

(iii) MOV M, C Moves the content of register C to the memory location whose address is in H-L pair

(iv) MVI A, 05 Moves data 05 in register A.

(v) MVI M, 08 Moves 08 to the memory location whose address is in H-L pair.

(vi) LXI H → for loading data in H-L pair

LXI B → for " " in B-C pair

(vii) LDA 2400H Loads the content of the memory location 2400H into the accumulator.

(viii) STA 2000H, store the content of accumulator in the memory location 2000H

(ix) LHLD 2500H Load H-L pair direct

LHLD 2500H will load the content of the memory location 2500H into register L. The content of the memory location 2501H is loaded into register H.

SHLD store HL pair direct, content of L is stored 2500H & register H in

LDAX B will load the content of the memory location (whose address is in the B-C pair), into the accumulator.
also used for DE pair

STAX D will store the content of the accumulator in the memory location whose address is in DE pair.
also used for BC pairs.

XCHG Exchange the contents of HL with DE pair

Arithmetic Instructions

ADD B → Adds the content of register B to the content of accumulator and sum is placed in accumulator.

ADD M The content of the memory location addressed in HL pair is added to the content of the accumulator & sum is placed in accu.

ADC R The content of register R and carry Status are added to the content of the accumulator.

ADC M Add memory with carry to accumulator

DAD H Add register pair to HL pair

SUB R Subtract register from Accumulator

SUB M Subtract ~~register~~ memory content from Accumulator

SBB R Subtract Register from Accumulator with Borrow

SBB M Subtract memory from accumulator with Borrow

INR R Increment register content

INR M Increment memory content

DCR R Decrement register Register R,

DCR M " memory content

INX HL Increment register pair

DCX DE Decrement " "

DAA → Decimal adjust accumulator

Logical Instructions

ANA R → Content of register R is anded with accumulator content

ANA M → Content of memory is anded with accumulator.

ANI 05 → AND immediate data with accumulator.

ORA R → Content of register R is ORed with accumulator content

ORA M → " memory " " "

ORI 05 → OR immediate data with accumulator

XRA R → ExOR contents of Register with accumulator

XRA M → " " memory " "

XRI 05 → ExOR immediate data with Accumulator

CMA → Complement the Accumulator contents

CMC → " the carry status

STC → Set carry status

CMP R → Compare the Register R contents with Accumulator contents

CMP M → Compare memory contents with Accumulator.

CPI 05 → Compare immediate data with accumulator.

RLC → Rotate accumulator Left (contents)

RRC → Rotate accumulator contents right

RAL → Rotate accumulator contents left through carry.

RAR → " " right " "

Branch Instructions

Conditional | Unconditional

JMP LABEL

JMP (ADDRESS) : Unconditional Jump instruction specified by the address.

JZ (LABEL) : Conditional Jump ; if the result is zero.

JNZ (LABEL) : Jump if the result is not zero.

JC (LABEL) Jump if there is a carry

JNC (LABEL) Jump if there is no carry

JPE (LABEL) Jump if even parity

JPO (LABEL) " " odd parity

CALL <Label> Unconditional call ; Call the subroutine identified by the address

RET Return from subroutine. (End subroutine)

RST n Restart

n = 0 → 7

PUSH HL Push the content of register pair to Stack

POP HL Copy the contents of the stack into the specified register pair

HLT Halt the program execution

SIM Set Interrupt Mask.

RIM Read Interrupt Mask.

NOP No operation.

Assembly Language Programs

① W.A.P to place 05 in register B

Mnemonics	operands	comments
MVI	B, 05	→ Gets 05 in register B
HLT		→ Stop

② W.A.P. get 05 in regist A, then move it to register B

MVI	A, 05	→ Gets 05 in register A
MOV	B, A	→ Transfer 05 from register A to B
HLT		→ stop

③ W.A.P. to load the content of the memory location FC50H

Memory Add	Mnemonics	Operands	
FC00	LDA	FC50	Get the content of the memory location FC50H into Accumulator
FC03	MOV	B, A	Moves the content of Reg. A to B.
FC04	HLT		Stop.

④ W.A.P. to Add 8-bit Numbers, Sum 8-bit
add 49H and 56H, Use memory location 2501H for 49H and 2502H for 56, store sum in 2503H location

8 bit no = 56 → 0101 0110
49 → 0100 1001
8 bit

Prog →

Memory Add.	Mnemonics	operands	Comments
2000	LXI	H, 2501H	Get Add. of 1st no 49H in HL pair
:			
2003	MOV	A, M	MOV 1st no. 49H in Accumulator
2004	INX	H	Increment content of HL pair
2005	ADD	M	Add 1st and 2nd nos.
2006	STA	2503H	store sum in 2503H
2009	HLT		Stop.

INX H increases the content of H-L pair by one ie 2501 + 1 = 2502

Result : 2503 — 9F H

```
  49 H
+ 56 H
-------
 9 15 H
  9F H
```

---

⑤ WAP to find 1's complement of an 8-bit number

eg. NO. = 96 H , binary form = 1001 0110 (9 6)

One's complement = 0110 1001 = 69H (6 9)

Memory Address	Opcode	Operands	Comments
2000	LDA	2501 H	Get data in accumulator
2003	CMA		Take its complement
2004	STA	2502 H	Store the result in address 2502H
2007	HLT		Halt.

Data 1:

2501 — 96 H

Result : 2502 — 69H

Data 2 :-

2501 — E4 H

1110 0100

1's complement 0001 1011

1 B

Result : 2502 — 1B

---

⑥ W.A.P to find 2's complements of an 8-bit No.

eg. 96 = 1001 0110 (9) (6)

1's complement = 0110 1001 = 69 (6) (9)

Increment by 1 → + 0000 0001

Result : 2's complement → 0110 1010 = 6A (6) (A)

Address	opcode	operands	Comments
2000	LDA	2501 H	Get data = 96 H in the accumulator
2003	CMA		Take its 1's complement
2004	INR	A	Take 2's complement by adding 1
2005	STA	2502 H	Store result = 6A H in location 2502 H
2008	HLT		Stop

Data : 2501 H → 96 H
Result : 2502 H — 69 H

Data : 2501 H — E4 H
Result : 2502 H — 1C H

⑦ W.A.P to subtract two 8-bit Numbers :→

eg. 1st No = 49 H
2nd No = − 32 H
Subtract = 17 H

2000	LXI	H, 2501 H	Get Address of 1st No in H-L pair
2003	MOV	A, M	Move 1st No = 49 H in accumulator
2004	INX	H	Increase the content of HL pair from 2501 to 2502 H
2005	SUB	M	Subtract 1st no. − 2nd no
2006	INX	H	Content of HL pair becomes 2503 H
2007	MOV	M, A	Store result in 2503 H
2008	HLT		Halt/stop.

```
MVI A, 20H
MVI B, 10H
SUB B
STA 2502H
HLT
```

```
 20
-10
 10
```

(8) W.A.P. to Multiply 8 bit nos :→

```
2000  MVI  A, 00H
      MVI  B, 05H
      MVI  C, 06H
LOOP: ADD  B
      DCR  C
      JNZ  LOOP
      HLT
```

5 × 5 = 10 → A

10 × 5 = 15 → 0A

15 × 5 = 20 → 25

20 × 5 = 25 → 31

25 × 5 = 30

Chapter-7 → Microprogrammed Control

This chapter contains (1) Control Memory (2) Address Sequencing Micoprogram Example (4) Design of Control Unit

The major functional parts in a digital computer are CPU, Memory, Input-Output.

The main digital hardware functional units of CPU are :→ Control Unit, Arithmetic & logic Unit (ALU) and registers.

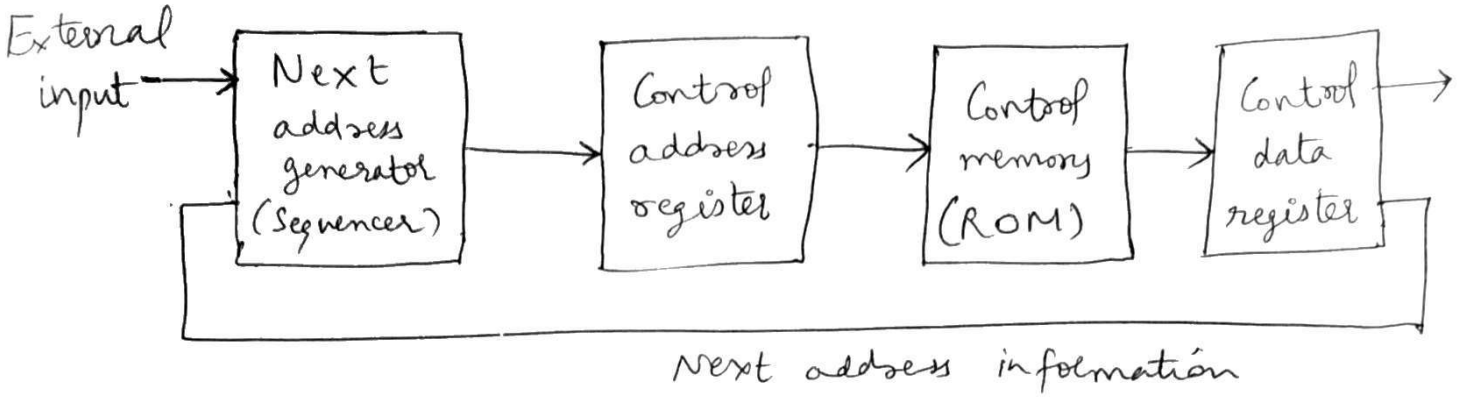
Two Methods of implementing Control Unit are :→
(a) Hardwired control (b) microprogrammed control

Hardwired control designed by fixed instructions, fixed logic blocks, arrays, encoders, decoders. When the control signals are generated by hardware using conventional logic design techniques the control unit is said to be hardwired.

Microprogramming is a 2nd alternative for designing the control unit of a digital computer.

Control Word :→ It is a string of 1's & 0's to represent the control variables at any given time. Each word in control memory contains with in it a micro instruction. A sequence of micro instructions constitutes a microprogram.

Block dij of microprogrammed control unit is shown in fig →



The control memory is assumed to be a ROM, within which all control information is permanently stored.

→ The control memory address register specifies the address of the microinstruction and the control data register holds the microinstruction read from memory.

Once these operations are executed, the control must determine the next address.

Sequencer :→ The next address generator is some times called a microprogram sequencer, as it determines the address sequence that is read from control memory.

Functions of a microprogram sequencer, are→incrementing
→ the control address register by one
→ loading into the control address register from control memory
→ transferring an external address
→ loading an initial address to start the control operations.

Address Sequencing :→

Micro instructions are stored in control memory in groups, with each group specifying a routine.

Each computer instruction has its own microprogram routine in control memory to generate microoperations that execute the instruction.

Steps for control to undergo for the address sequencing :→

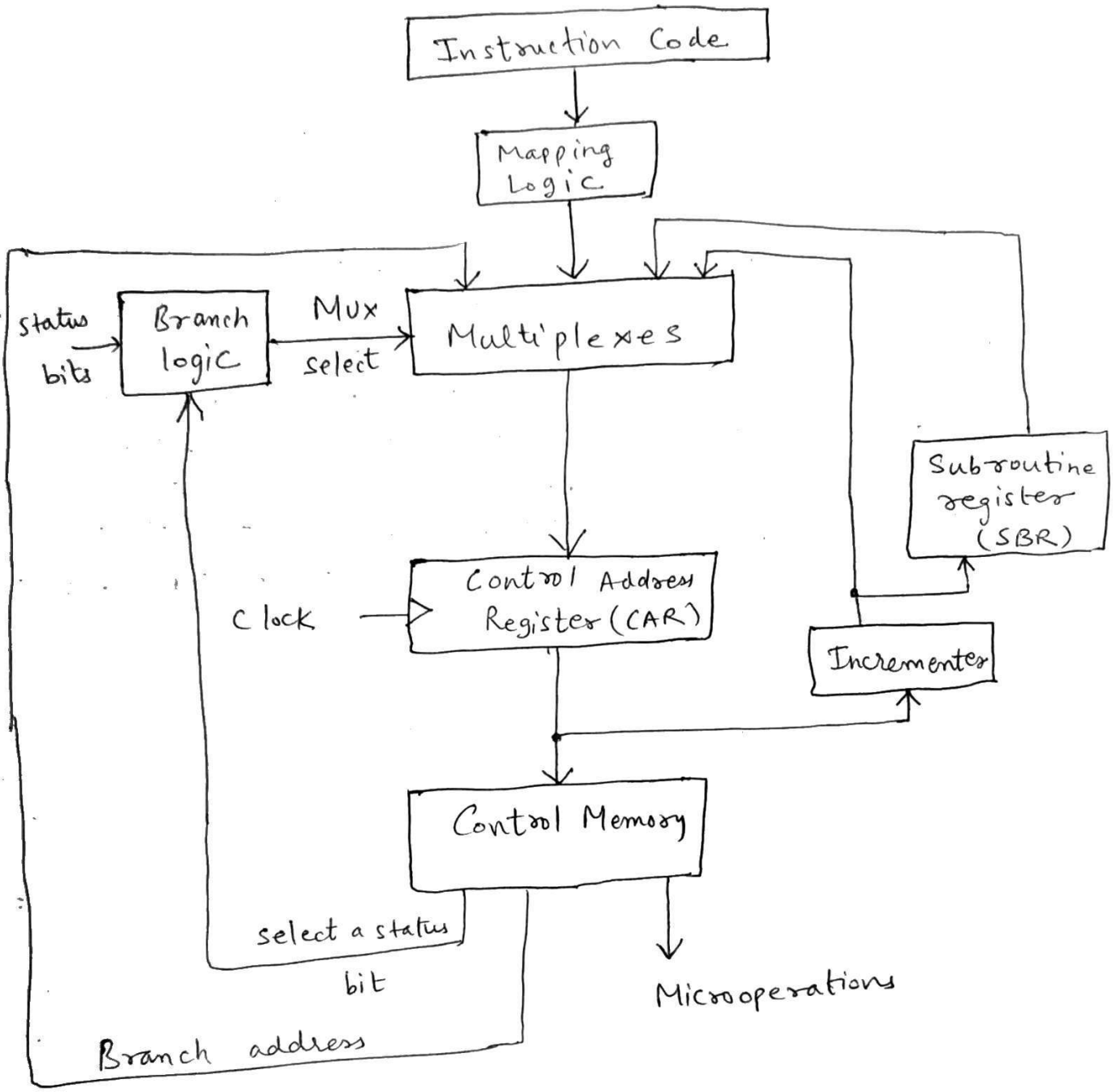
1) An initial address is loaded into the control address register when power is turned on. This address of the 1st micro instruction that activates the instruction fetch routine.
The fetch routine may be sequenced by incrementing the control address register

Summary of the address sequencing :→

1) Incrementing of the control addres register
2) Unconditional branch or conditional branch, depending on status bit conditions
3) A mapping process from the bits of the instruction to an address for control memory.
4) A facility for subroutine call & return.

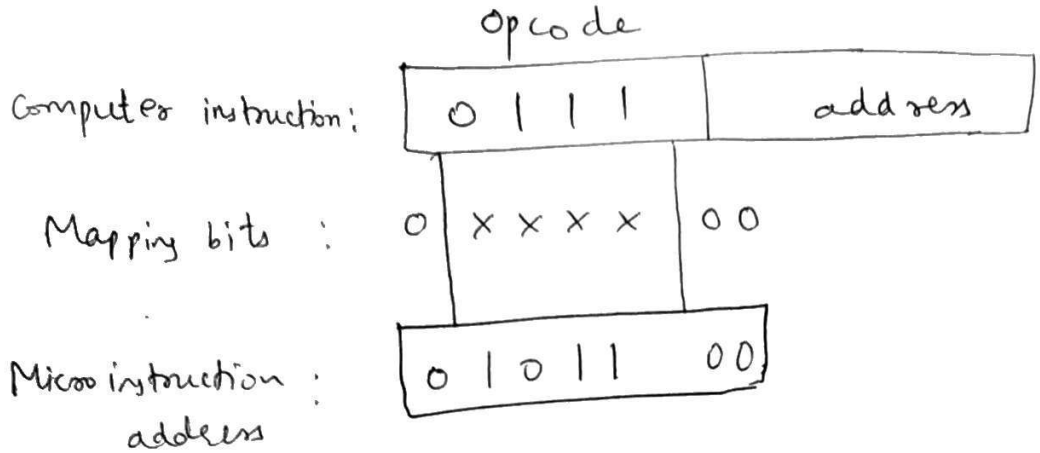
The block dig of a control memory and hardware needed for selecting the next micro instruction addr show four different paths from which the CAS (control Address Register) recieves address →

Block diy



Conditional branching :→ The branch logic in above fig provides decision-making capabilities. 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 a number, the mode bits of an instruction, and i/p / o/p status conditions.

Mapping of Instruction :→ Consider a simple instruction format →



An operation code has 4 bits to specify 16 instructions
A control memory has 128 words, requiring an address of seven bits.

One simple mapping process converts the 4bit Opeeation code to a 7-bit address for control memory. This mapping consists of placing a 0 in the MSB of the address, transferring the 4 opcode bits & clears the two LSB bits

If the routine needs more than 4 instructions it can use addresses 1000000 —— 1111111

Mapping concept provides flexibility for adding instructions for control memory as needed.

Subroutine :→ It may be called from any point with in the main body of the micro program. Subroutines are programs that used by other routines to accomplish a particular task.

Symbolic Microinstructions :→

A symbolic microprogram can be translated into its binary equivalent by means of an assembler.

Each line of the assembly language microprogram defines a symbolic microinstruction.

Each Symbolic microinstruction is divided into 5 fields :→ label, microoperations, CD, BR, and AD.

pseudoinstruction ORG is used to define the origin (first Address) of a microprogram routine.

Symbolic Microprogram :→

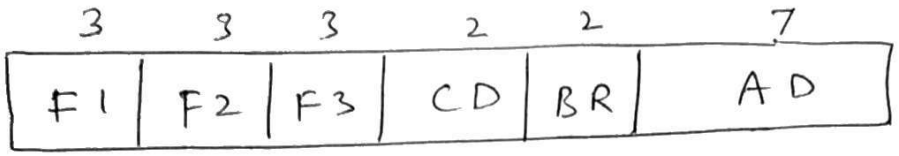
It is a convenient form for writing microprograms in a way that people can read and understand.

Symbolic microprogram must be translated to binary either by means of an assembler program.

eg.

micro Routine	Adress Decimal	Adress Binary	Binary micro instruction F1	F2	F3	CD	BR	AD
ADD	0	0000000	000	000	000	01	01	1000011
	1	0000001	100	000	000	00	00	0000010
	2	0000010						
	3	0000011						
BRANCH	4							
	7	0000111						
STORE	8	0001000	000	000	000	01	01	1000011
FETCH								

Microinstruction code format (20 bits) is written as



F1, F2, F3 are the 3 fields to specify micro operations for the computer. CD field selects status bit conditions. BR field specifies the type or branch to be used. AD field contains a branch address.

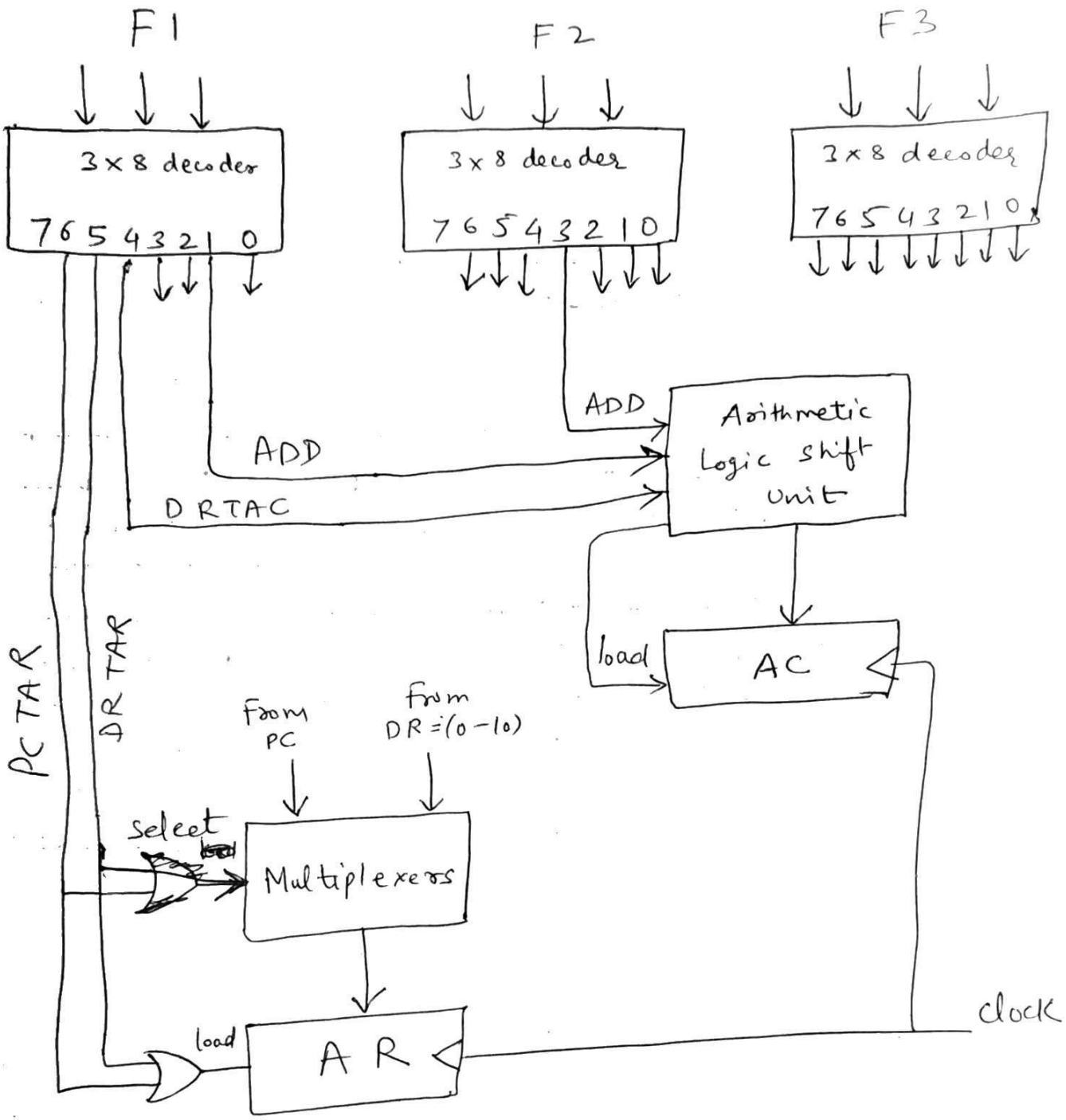
It is of 7 bits.

Control memory has $128 = 2^7$ words.

F1	F2	F3	CD	BR
000	000	000	00	00
001	001	001	01	01
010	.		10	10
!	!	!	11	11
111	111	111		

Design of Control Unit :→

Each of the 3 fields of micro instruction presently (F1, F2, F3) in the o/p of control memory are decoded with a 3x8 decoder to give 8 o/p's. Block dig in fig. (next page) shows o/p 5 and 6 of decoder F1 are connected to the load i/p of AR (Add. register) so that when either one of these o/p's is active, information from the multiplexers is transfered to AR.



The Mux select the information fom DR when o/p 5 is active and from PC when o/p 5 is inactive. The transfer into AR occurs with a clock pulse only when o/p 5 or 6 of the decoder are active.

DR to AC = DRTAC,

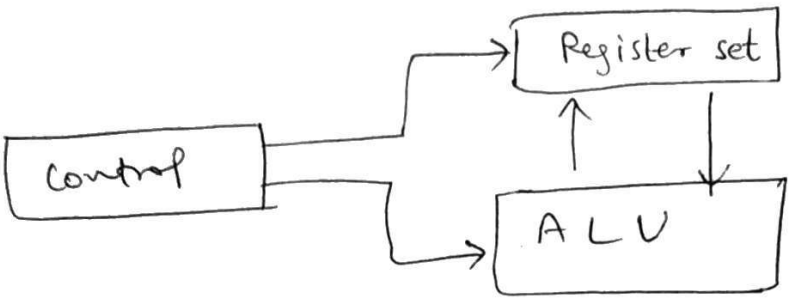
PC to AR = PCTAR,

DR to AR = DRTAC

Chapter – 8 CPU (central processing Unit)

The 3 major parts of CPU are :→

1 Control 2 ALU , 3 Register set



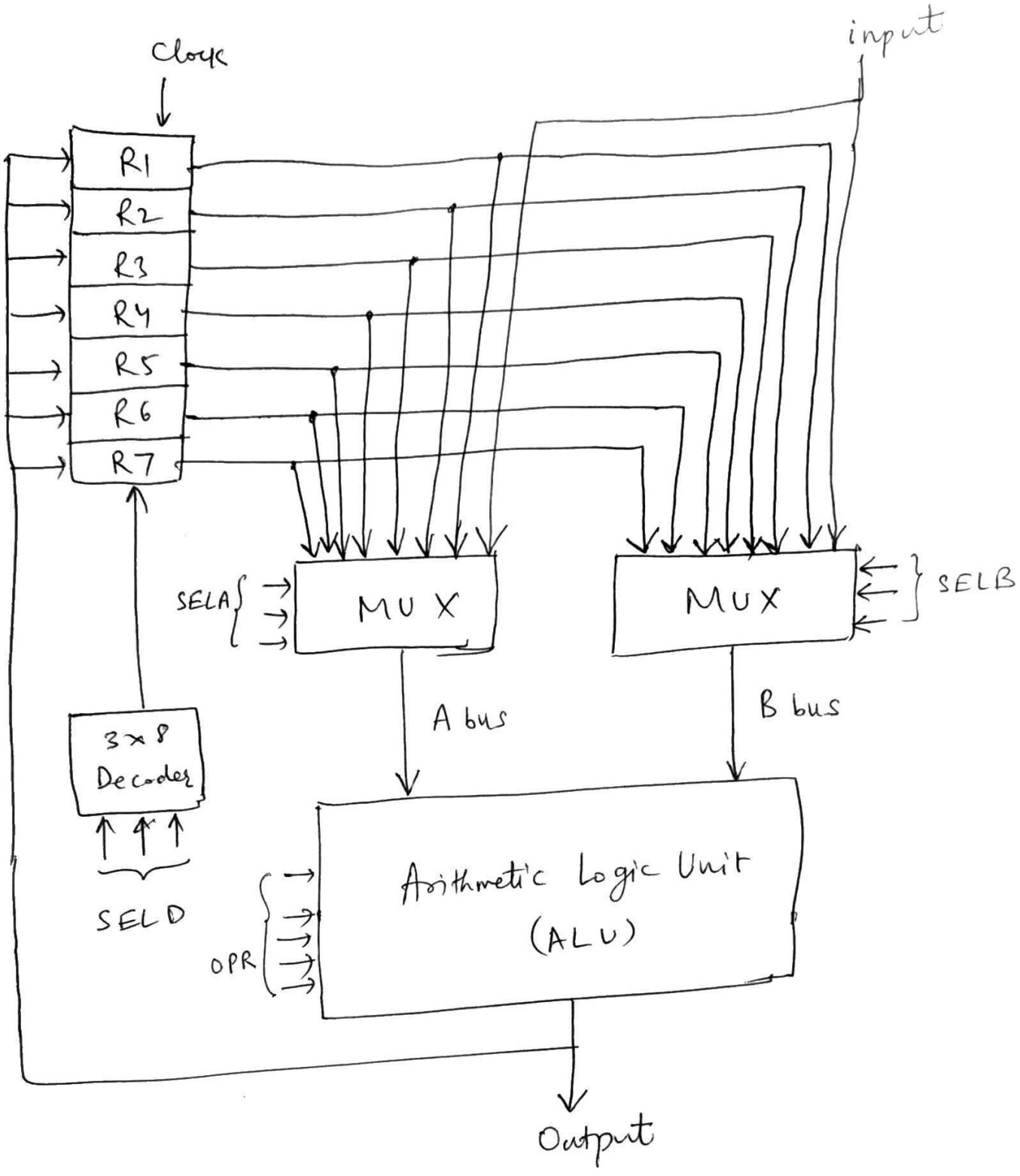
CPU block dig →

Computer architecture includes instruction formats addressing modes; instruction set , CPU registers. and RISC (Reduced instruction set computer) CISC (Complex instruction set computer).

General Register Organization →

A bus organization for 7 CPU registers in shown in fig. O/p of each register is connected to the 2 Mux to form the two buses A & B The selection lines in each mux select one register or the i/p data for the particular bus. The result of the microperation is available for O/p data and also goes into the i/p's of all the registers.

The registers is selected by a decoder 3 x 8



To perform the operation of $R1 \leftarrow R2 + R3$ the control must provide the binary selection variables to the following selector i/ps ⇒

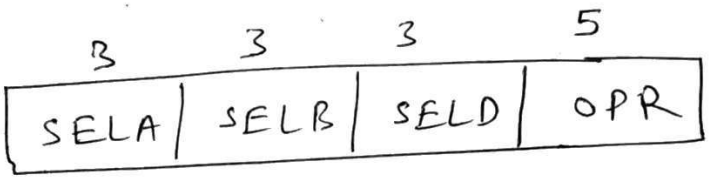
1. MUX A selector (SELA): to place the content of R2 into bus A.
2. MUX B selector (SELB): to place the content of R3 into bus B.

3. ALU operation selector (OPR): to provide the arithmetic addition A+B.

4. Decoder destination selector (SELD): to transfer the content of the out-put bus into R1

Control Word :→

There are 14 binary selection i/ps in the unit and their combined value specifies a control word as shown below :→



= 14 bits

Then encoding of the register selections is given

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

Stack Organization :→

Stack is a storage device that stores information in such a manner that the item stored last is the first item retrieved. ie Last in First Out (LIFO)

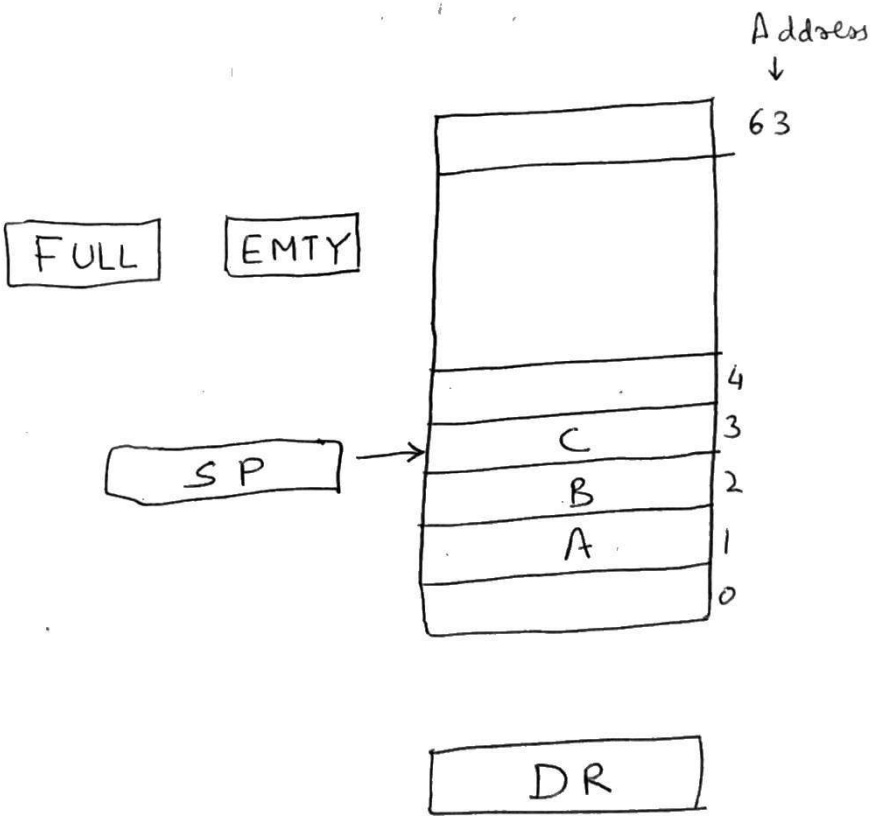
The register that holds the address for the stack is called a stack pointer (SP). Its value always points at the top item in the stack

Two operations of a stack are :→

(i) The insertion of items called as push (or push down)

(ii) The deletion of items called as POP (or pop-up)

Register stack → It can be organized as a collection of a finite number of memory words or registers. A 64-word register stack organization



The stack pointer register SP contains a binary number whose value = address of the word that is currently on top of the stack.

For eg. 3 items are placed in the stack : A, B & C.

If item C is on top of the stack, now SP = 3

To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the content of SP.

Now item B comes on top of the stack and SP = 2

In a 64 word stack, SP contains 6 bits ie $2^6 = 64$. In binary 63 = 111111

SP has only 6 bits, it can not exceed 63.

Incrementing SP by 1, ie 111111 + 1 = 1000000

So SP show 000000 6 bits only.

1 bit register FULL is set to 1, when the stack is full, And the 1-bit register EMTY is set to 1 when the stack is empty of items

DR is the data register that holds the binary data which written / read out of the stack.

Initially, SP = 0, EMTY = 1 & FULL = 0

Push operation is implemented with micro operations :→

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

Similarly, if a new item is deleted from the stack by POP operation with following micro operations :-

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

SP ← SP − 1 Decrement stack pointer

If (SP = 0) then (EMTY ← 1) Check if stack is empty

FULL ← 0 Mark the stack not full

Memory stack :→

The implementation of a stack in the CPU is done by assigning a portion of memory to a stack operation and using a processor register as a SP, fig →

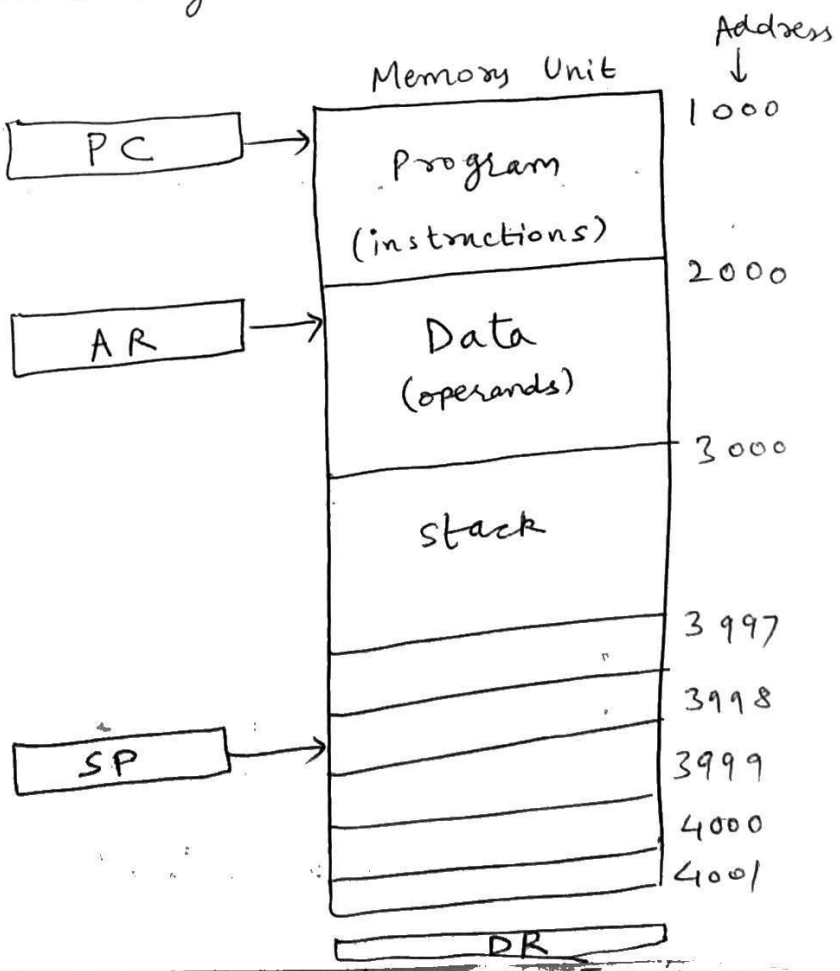


Fig shows a computer memory partitioned into 3 segments Program, Data and stack.

PC points at the address of the next instruction in the program. AR (Address Register) points at an array of data. SP points at the top of the stack. PC is used to fetch phase to read an instruction.

AR is used to execute phase to read an operand

SP will push/pop items from the stack.

1st item is stored in the stack at address 4000, 2nd item at address 3999 & last address can be used for stack at 3000.

Push :→ New item is inserted with push operation as follows :→

SP ← SP − 1 SP is decrement

M[SP] ← DR memory write operation inserts word from DR.

POP :→ New item is deleted with pop operation :→

DR ← M[SP] Read the top item

SP ← SP + 1 SP is incremented.

Reverse Polish Notation :→ (RPN)

Polish mathematician Lukasiewicz showed that arithmetic expressions can be represented in Prefix Notation, by placing the operators before the operands.

Reverse Polish Notation (post fix notation) places the operator after the operands. eg.

A + B	Infix Notation
+ AB	Prefix "
AB +	Post fix. (Reverse Polish Notation)

RPN form is suitable for stack manipulation

For eg. given arithmetic expression is

A * B + C * D

In RPN, AB * CD * +

All arithmetic operations like Division, multiplication first than addition & subtraction operations

Computer scans from left to right in this form:→

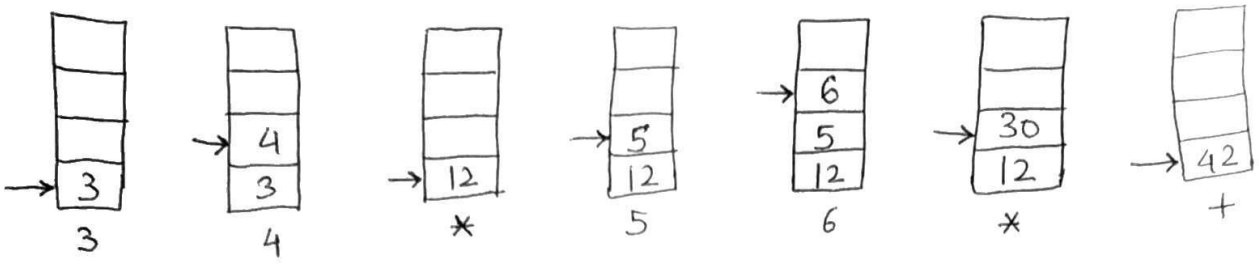
(A * B) (C * D) +

Stack operations in RPN :→

Consider an eg. (3 * 4) + (5 * 6)

In RPN form = 34 * 56 * +

Stack operations are explained in fig →



Each box represents one stack operation and Arrow (→) always points to the top of the stack. 1st no. 3 is pushed into the stack, then no. 4.

Instruction formats :→

Three - Address instruction formats eg.

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

Program in Assembly Language with 3 registers is →

```
ADD   R1 , A , B
ADD   R2 , C , D
MUL   X , R1 , R2
```

denotes R1 ← M[A] + M[B]

denotes R2 ← M[C] + M[D]

M[X] ← R1 * R2

Here M[A] denotes the operand at memory addres A

Two Address Instructions :→

```
MOV   R1 , A
ADD   R1 , B
MOV   R2 , C
ADD   R2 , D
MUL   R1 , R2
MOV   X , R1
```

R1 ← M[A]

R1 ← R1 + M[B]

R2 ← M[C]

R2 ← R2 + M[D]

R1 ← R1 * R2

M[X] ← R1

One-Address Instruction:→

One-address instructions use an implied accumulator (AC) register for all data manipulation. eg $X = (A+B) * (C+D)$

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

where T is the address of a temporary memory location required for storing the intermediate result.

Zero-Address Instruction:→

PUSH & POP instructions will be used →

PUSH	A	top of stack ← A
PUSH	B	top of stack ← B
ADD		// ← (A+B)
PUSH	C	// ← C
PUSH	D	// ← D
ADD		// ← (C+D)
MUL		// ← (C+D) * (A+B)
POP	X	M[X] ← TOS (Top of stack)

Addressing Modes :→

Instruction cycle has 3 phases :→

1 Fetch the instruction from memory
2 Decode the instruction
3 Execute the instruction

(1) **Implied Mode** :→ In this mode the operands are specified implicitly in the definition of the instruction eg. CMA complements the data in accumulator. There is no register / memory ie implied mode.

(2) Immediate Mode — MVI , A , 05H

(3) Register Addressing Mode — ADD R1, R2 — R1←R2+R1

(4) Register Indirect Mode

(5) Direct Addres Mode — ADD R2 , 3 — R2←[R2]+3

(6) Indirect Address Mode

(7) Relative Address Mode

(8) Indexed Addressing Mode

(9) Base Register Addressing Mode

(10) Auto - Increment // — INR A

(11) Auto - Decrement // — DCR A

Types of Interrupts :→

1> External 2> Internal 3> Software

External interrupts come from i/p - o/p devices, from a timing device. (1) Maskable (2) Non maskable

Maskable → Hardware interrupts that can be delayed when highest priority interrupt has occured

Non maskable → That cannot be delayed

(2) Software interrupts are generated from internal devices and programs. RST 5.5, RST 6.5, RST 7.5,

(3) Internal interrupts are synchronous with program while external interrupts are asynchronous. Internal interrupts arise from erroneous use of an instruction. It is also called as Trap.

eg:. Register Overflow, Stack Overflow etc.

Reduced Instruction set Computer (RISC)

A computer with a large no. of instructions is classified as CISC, Complex instruction set computer

Characteristics of CISC architecture :→

1. A large number of instructions - typically (100-250)
2. Some instructions that perform specialized tasks and are used infrequently.
3. A large variety of addressing modes - (5-20) different modes.
4. Variable - length instruction formats
5. Instruction that manipulate operands in memory

RISC characteristics :→

Concept is to reduce execution time by simplifying the instruction set of computer

1. Relatively few instructions.
2. " few addressing modes.
3. Memory access limited to load & store instructions.
4. All operations done with in the registers of the CPU.
5. fixed length, easily decoded instruction format
6. Single - cycle instruction execution
7. Hardwired rather than microprogammed control.

Chapter - 10 Computer Arithmetic

We need to develop the various arithmetic algorithms for addition, subtraction, multiplication, divisions etc

Addition & subtraction with signed-magnitude data

There are 8 different conditions when two nos. A & B are added / subtracted. When signs of A & B are identical / different → add the two magnitudes and attach the sign of A to the result.

When the signs of A & B are different / identical Compare the magnitudes and subtract the smaller number from the larger.

Choose the sign of the result to be same as A if $A > B$. or the complement of the sign of A if $A < B$.

If the two magnitudes are equal, subtract B from A & make the sign of result +ve.

Operation	Add Magnitudes	Subtract Magnitudes		
		when A > B	when A < B	A = B
(+A) + (+B)	+(A+B)			
→ (+A) + (−B) ✓		+(A−B)	−(B−A)	+(A−B)
(−A) + (+B) ✓		−(A−B)	+(B−A)	+(A−B)
(−A) + (−B)	−(A+B)			
→ (+A) − (+B) ✓		+(A−B)	−(B−A)	+(A−B)
→ (+A) − (−B)	+(A+B)			
→ (−A) − (+B)	−(A+B)			
(−A) − (−B) ✓		−(A−B)	+(B−A)	+(A−B)

Hardware Implementation: →

Let A & B are the two registers that hold the magnitudes of the numbers.

As & Bs be the two F/Fs that hold the signs

Result of operation is transfered to 3rd register.

Accumulator register is formed by A & As together.

A parallel-adder is needed to to perform A+B.

To establish, if A > B, A = B, A < B, Comparator is needed.

Parellel-subtractor circuits are needed for A−B & B−A

Subtraction is done by adding A to 2's complement of B

O/p carry is transfered to F/F E.

Add overflow F/F (AVF) holds the overflow bit when A & B are added.

$A + (\overline{B} + 1) = 2$'s complement method.

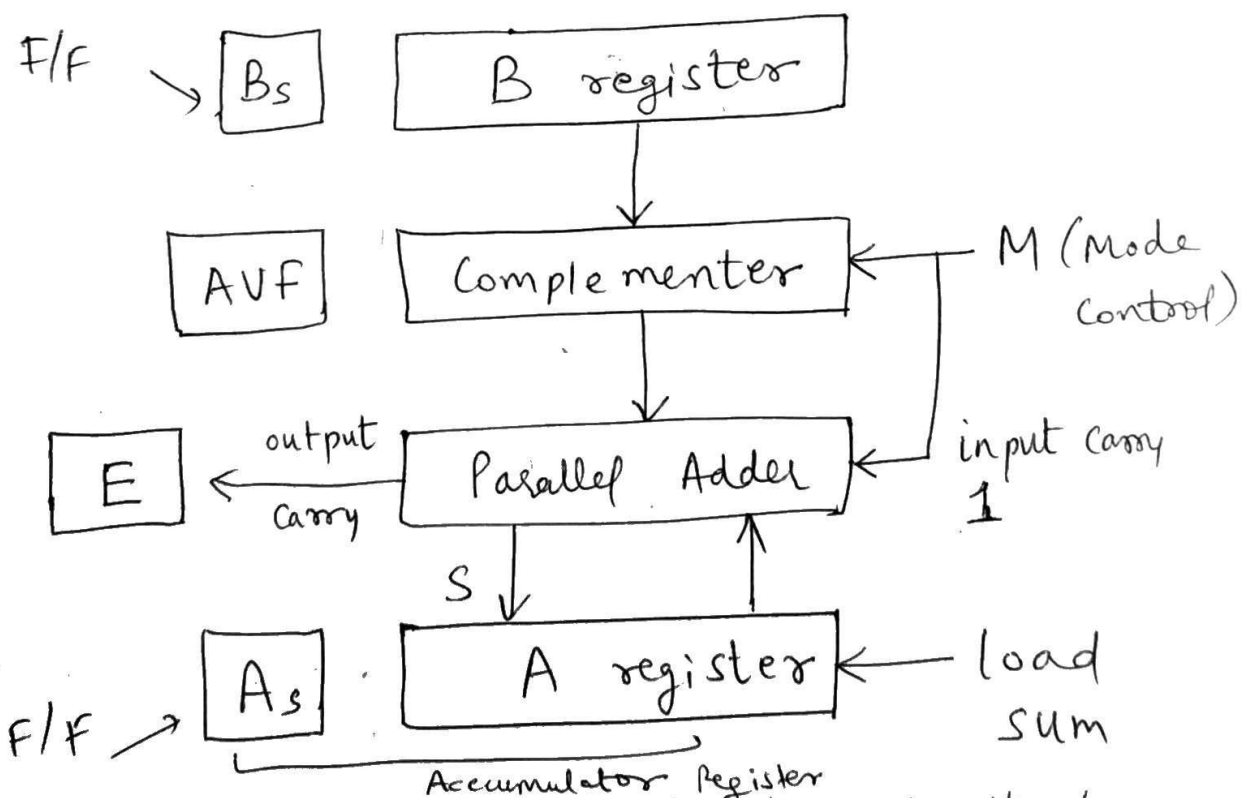
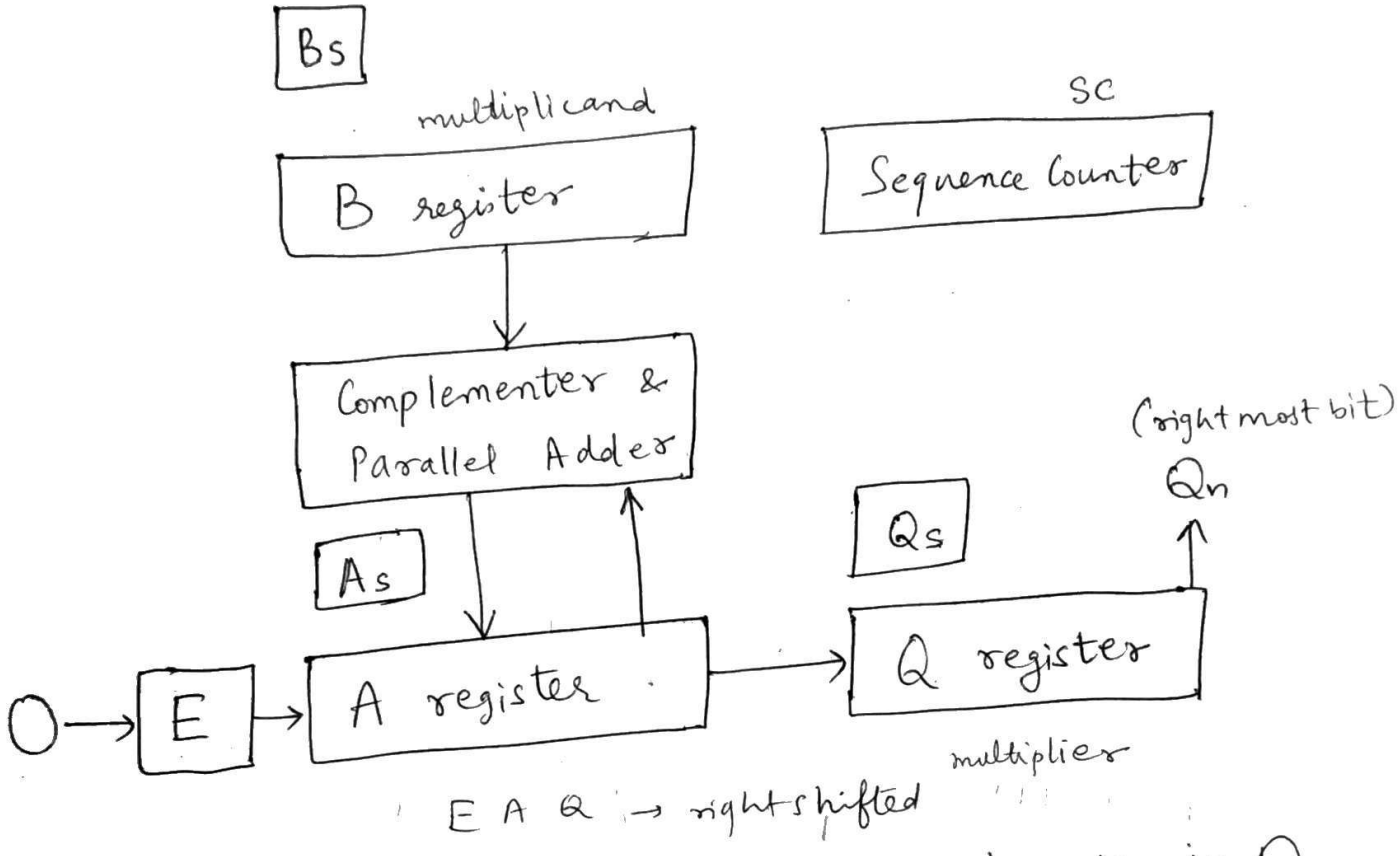


fig → Hardware design for sign-magnitude Addition/Subtraction

Multiplication Algorithms :→

multiplication of two fixed-point binary numbers in signed-magnitude representation. The sign of the product is +ve if multiplicand & multiplier are alike. otherwise product sign is −ve. Design →



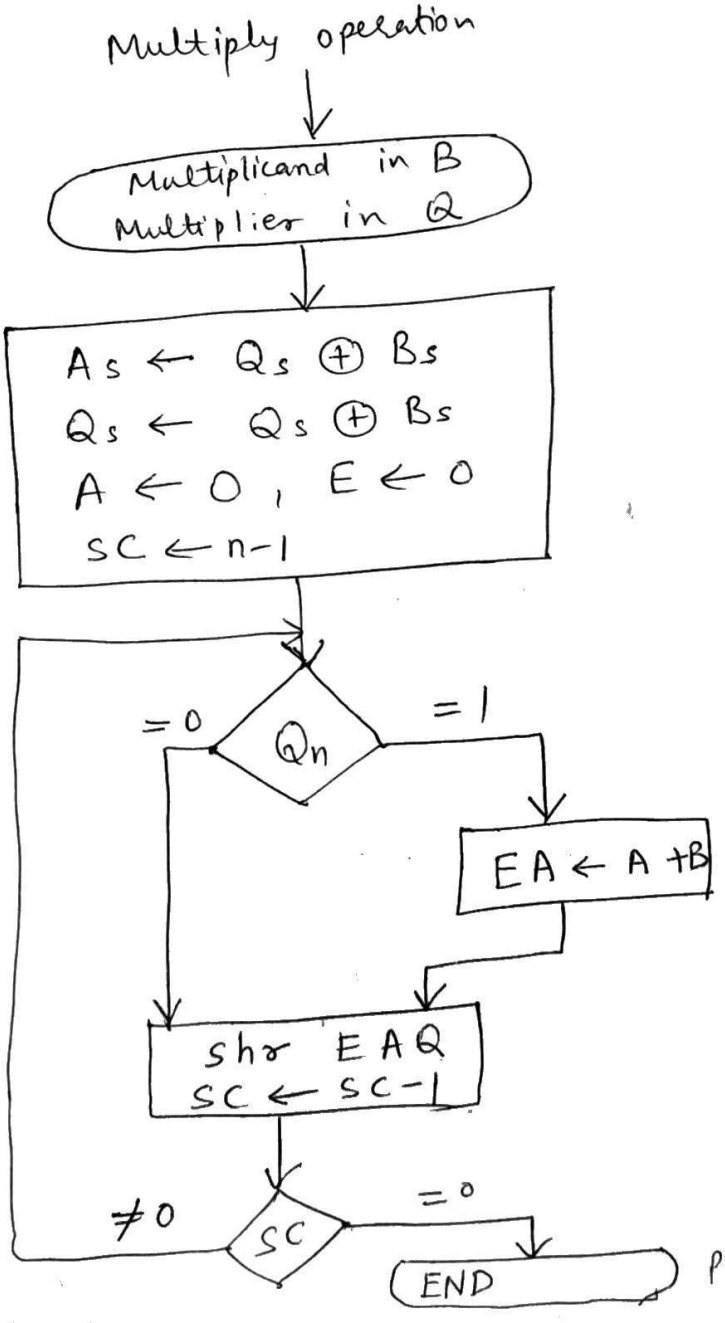
→ Multiplier is stored Q register & its sign in Qs
→ Sequence Counter (SC) is initially set to a number = the no. of bits in the multiplier
→ Decrement counter by 1 after products.
→ When the SC = 0, Product is complete & stops process
→ Register B has multiplicand, & Q has multiplier.
→ The sum of A + B form a partial product which is transferred to the EA register.
→ Both partial product & multiplier are shifted to the right & denoted by EAQ

eg. 23
× 19
437

```
  10111      B   Multiplicand
× 10011      Q   Multiplier
---------
  10111
  10111
  00000
 00000   +
 10111
---------
 110110101      Product
---------
```

If the multiplier bit is a 1, multiplicand is copied down otherwise, zeros are copied down.

Flow chart →



Booth Multiplication Algorithm :→

Booth algorithm gives a procedure for multiplying binary integers in signed-2's complement representation.
Strings of 0's in the multiplier require no addition but just shifting.
A string of 1's in the multiplier from bit weight 2^K to weight 2^m can be treated as $2^{K+1} - 2^m$.

Ex: binary number 0 0 1 1 1 0 (+14)
has a string of 1's from 2^3 to 2^1 (K=3, m=1)

The no. can be represented :→

$$2^{K+1} - 2^m = 2^4 - 2^1 = 16 - 2 = 14$$

∴ M is the multiplicand and 14 is multiplier,

Then $M \times 14 = M[2^4 - 2^1]$

Product $= M \times 2^4 - M \times 2^1$

∴ Product can be obtained by shifting the binary multiplicand M, 4 times to the left and subtracting M shifted left 1 time.

Hardware implementation of Booth Algorithm is shown in fig with registers :→
Registers A, B and Q are renamed as AC, BR, QR respectively. Q_n designates the least significant bit of the multiplier in register QR
An extra F/F Q_{n+1} is appended to QR to facilitate a double bit inspection of the multiplier.

Hardware of Booth Algo →

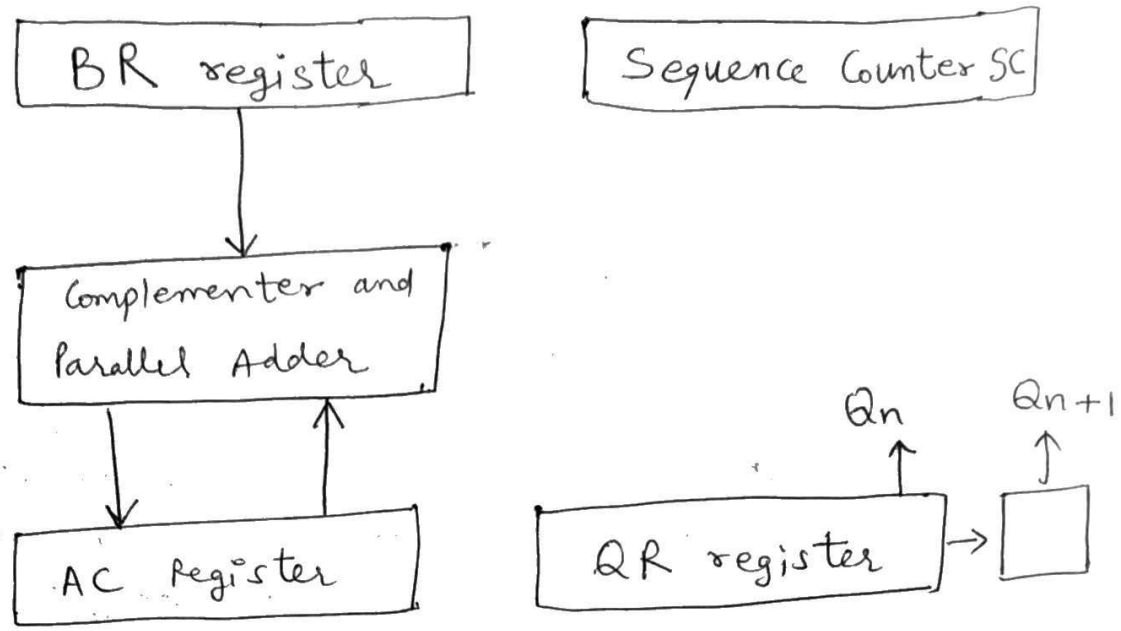


fig 1

Flow chart is shown in fig 2 →

Register AC and bit Q_{n+1} are initially cleared to 0.

$SC \leftarrow n$, n is the no. of bits in the multiplier.

Two bits of the multiplier are inspected ie.

Q_n & $Q_{n+1} = 10$, It means the 1st bit 1 in a string of 1's has been encountered (occur)

∴ subtraction of the multiplicand will be done from the partial product in AC. ie

$$AC \leftarrow AC + \overline{BR} + 1$$

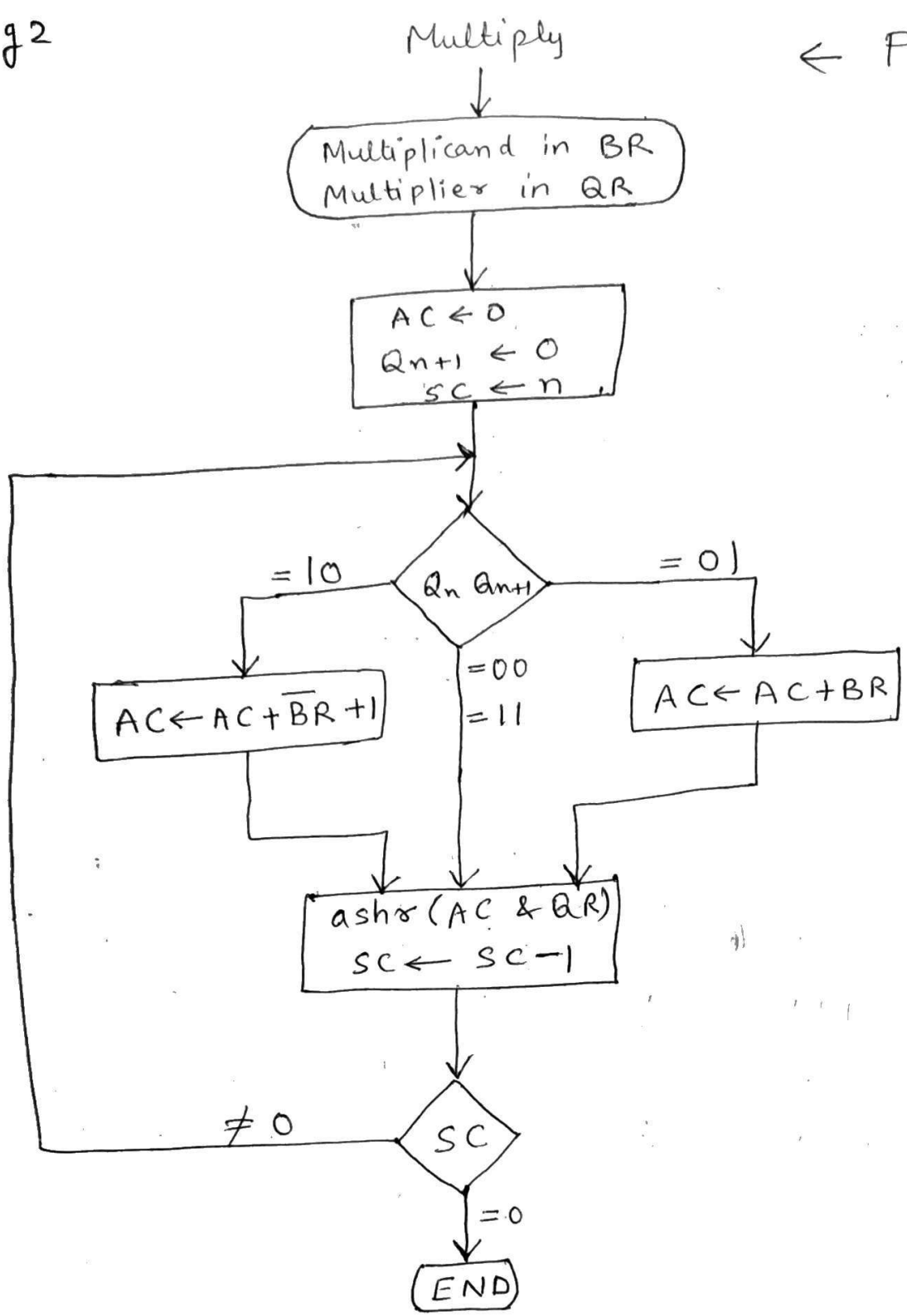
2's complement addition

If the two bits $Q_n\, Q_{n+1} = 01$, then the 1st 0 in a string of 0's has been encountered (occured)

∴ Addition of the multiplicand to the partial product in AC is done. $AC \leftarrow AC + BR$

If two bits are equal, partial product doesn't change $Q_n\, Q_{n+1} = 00$ or 11

fig 2



← Flow chart of Booths Algorithm

Then partial product will be shifted right, including multiplier using (ashr) arithmetic shift right operation. ie ashr AC & QR will shift right Decrement the Sequence Counter, SC ← SC − 1
Loop is repeated 'n' times.

Example of Multiplication with Booth Algorithm →

If $n = 5$. Multiply $(-9) \times (-13) = +117$

Multiplier in QR = -ve , Multiplicand in BR = -ve.

While Product is of 10 bits in AC & QR is +ve.

$Q_n\ Q_{n+1}$ $BR = -9$ Multiplicand. Let $QR = -13$ multiplier

Determine 2's compliment of -9 & -13.

$-9 = 01001$ in binary form

1's compl. $(-9) = 10110$,

2' Comp $(-9) =$ $\begin{array}{r} +1 \\ \hline 10111 \end{array}$ BR

$QR = -13 = 01101$

1's compl. $= 10010$

$\begin{array}{r} +1 \\ \hline \end{array}$

2's compl. $= 10011$ QR

Array Multiplier :→

During multiplication, checking of bits of the multiplier one at a time & forming partial products is a sequential operation. It needs add & shift micro operations.

The multiplication of 2 binary nos. can be done with one micro operation by means of a combinational ckt that forms the product bits all at once.

Example 2 bits by 2 bits multiplication →

Multiplicand → $b_1\ b_0$

Multiplier → $a_1\ a_0$

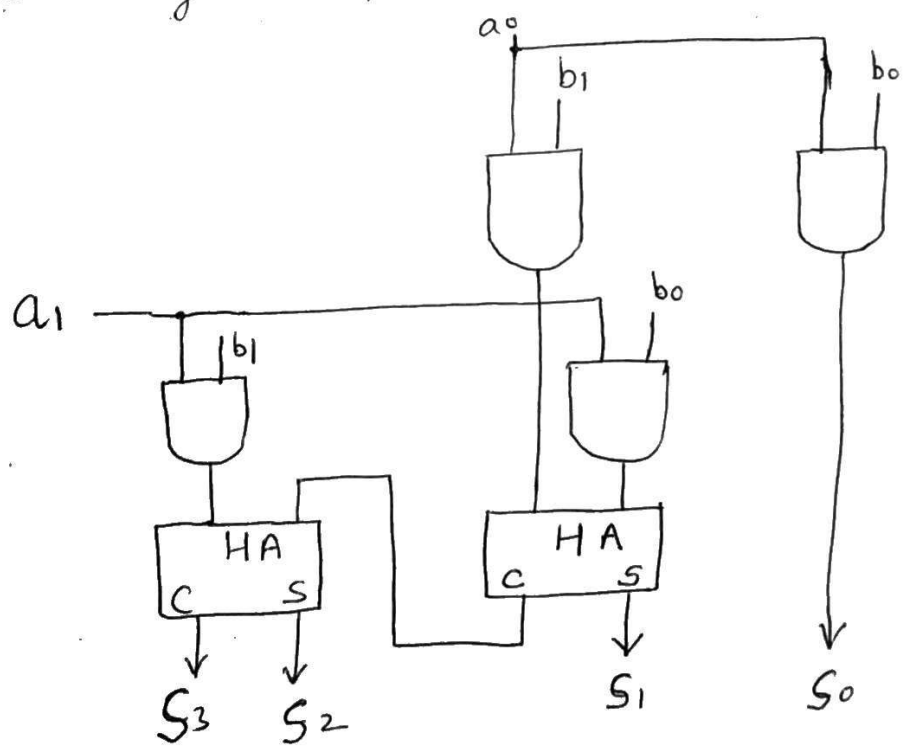
$\times$

$a_0b_1 \quad a_0b_0$ ← first partial product = AND operation

$a_1b_1 \quad a_1b_0$ ← 2nd partial product & shift left one position

product → $S_3\ S_2\ S_1\ S_0$

This AND gate implementation is shown in fig →



$a_0\ b_0$

$1 \times 1 = 1$

$1 \times 0 = 0$

$0 \times 1 = 0$

$0 \times 0 = 0$

Fig 1→ 2 bit by 2 bit Array Multiplier ckt

4 bit by 3 bit Array Multiplier

Increasing the no. of bits more than 2, needs full-adders to produce sum.

Multiplicand $b_3\ b_2\ b_1\ b_0$

Multiplier $\times\ a_2\ a_1\ a_0$

$a_0(b_3\ b_2\ b_1\ b_0)$ ← 1st partial product with a_0

Augend → $a_0b_3\ a_0b_2\ a_0b_1\ a_0b_0$

Addend → $a_1b_3\ a_1b_2\ a_1b_1\ a_1b_0\ \times$

$a_2b_3\ a_2b_2\ a_2b_1\ a_2b_0\ \times\ \times$

$S_6\ S_5\ S_4\ S_3\ S_2\ S_1\ S_0$