Computer Organization and Architecture

UNIT-I

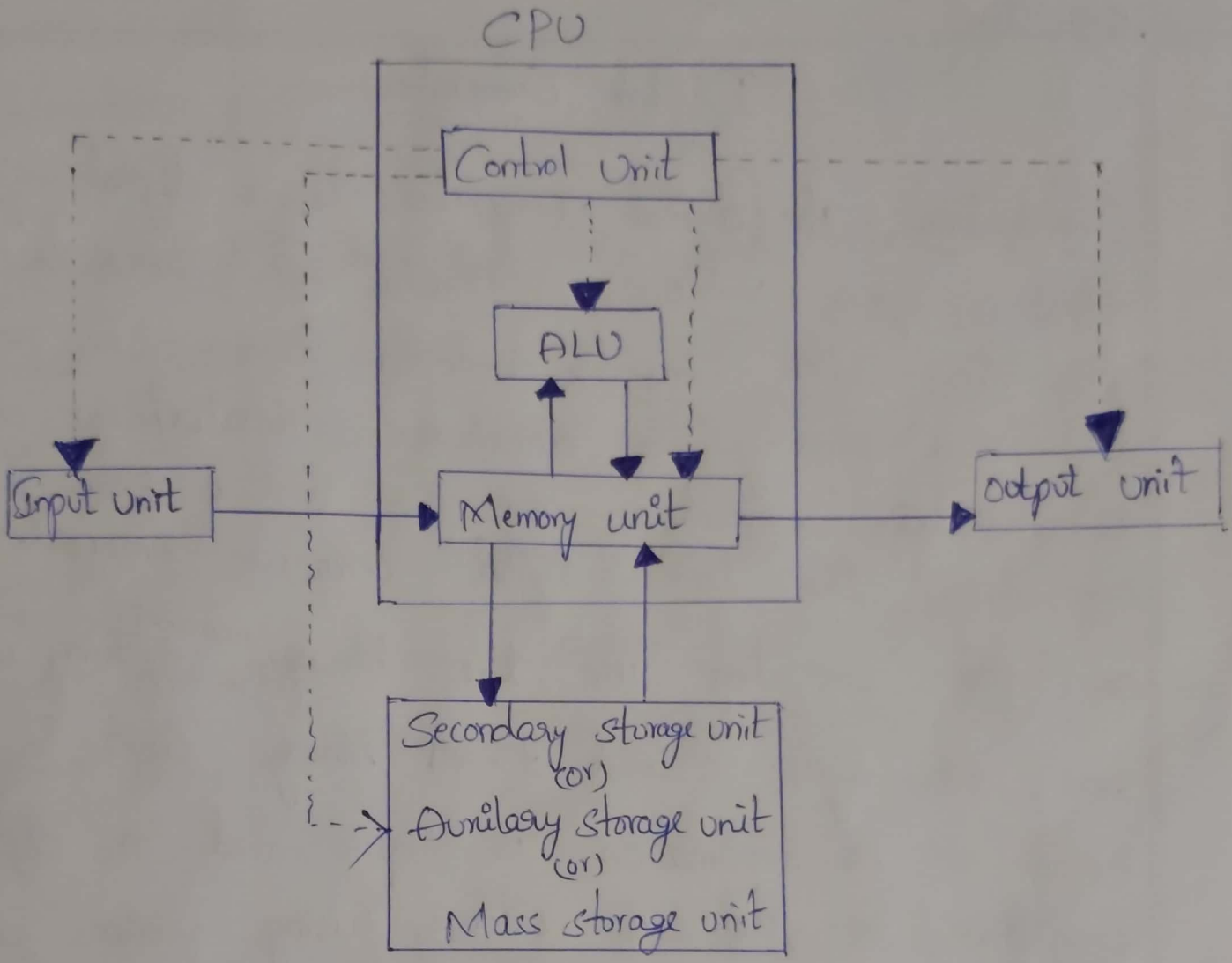
CSE: II-I

Digital Computers

Introduction :- The digital computer is a digital system that perform various computational tasks. The first electronic digital computer, developed in the late 1940s, was used primarily for numerical computations and the discrete elements were the digits. From this application the term digital computer emerged.

→ digital computers use the binary number system, which has two digits: 0 and 1. A binary digit is called a bit. Information is represented in digital computers in groups of bits. By using various coding techniques, groups of bits can be made to represent not only binary numbers but also other discrete symbols, such as decimal digits or letters of the alphabets.

Block diagram of Digital Computer



Input unit :- Input unit used to process of entering data and programs in to the computer system, the input unit takes data from us to the computer in an organized manner for processing.

CPU :- The Central processing unit (CPU) takes data and instructions from the storage unit and makes all sorts of calculations based on the instructions given and the type of data provided.

Memory unit :- All the data and instructions are stored here before and after processing. Intermediate results of processing are also stored here.

Control unit :- It controls the all operations like input, processing and output are performed by control unit. It takes care of step by step processing of all operations inside the computer.

Output unit :- An output device is a part of computer hardware equipment which converts information into human-readable form. It can be text, graphics, tactile, audio and video.

Computer Organization :- Computer Organization refers to the level of abstraction above the digital logic level, but below the Operating System. It mainly concerned with the structure and behaviour of a computer system as seen by the user.

→ It tells us how exactly all the units in the system are arranged and interconnected

→ An organization is done on the basis of architecture.

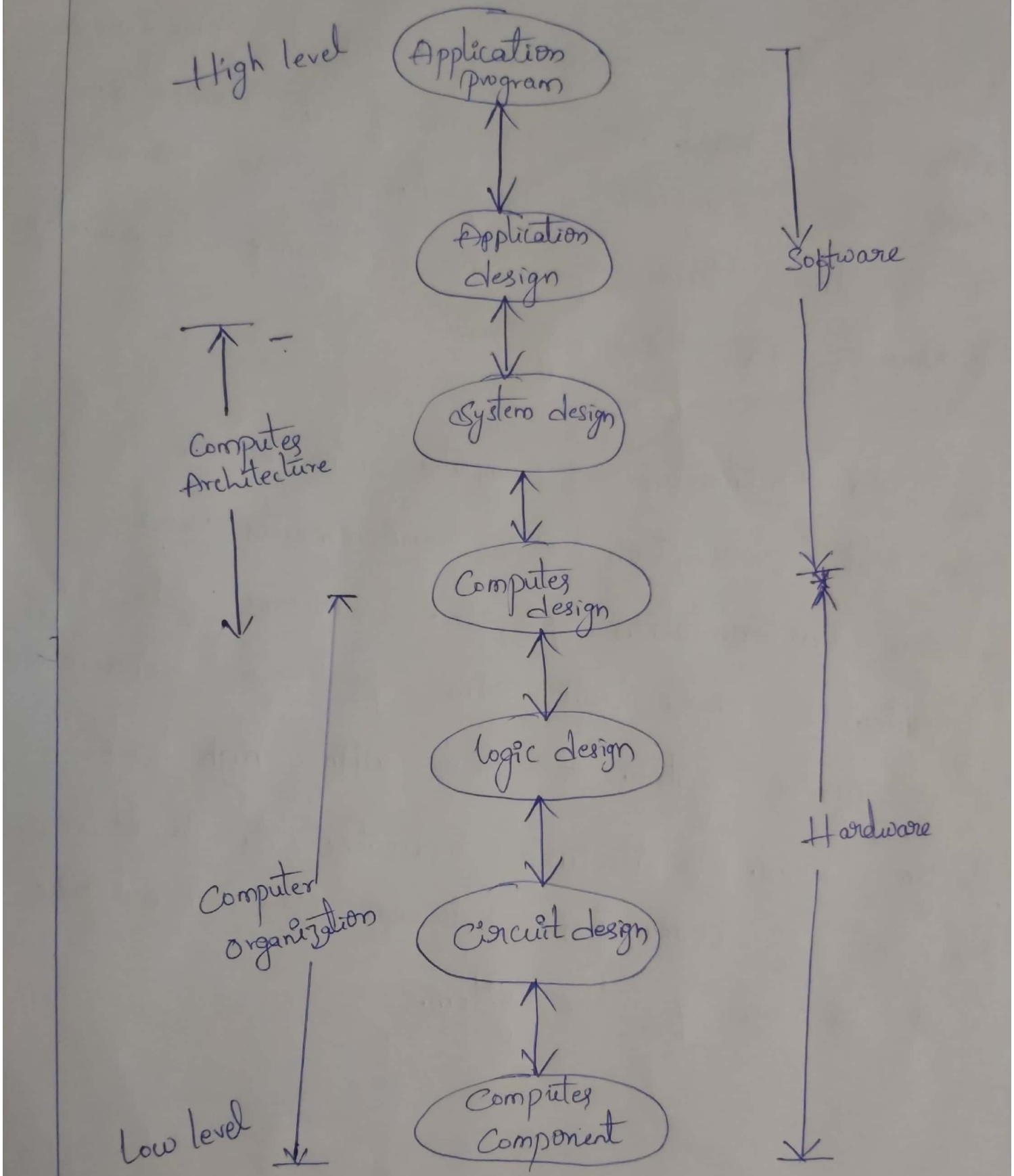
→ Computer organization deals with the low level design issues. which involves physical components such as circuit design, Adders, signals and peripherals.

→ the main objective of this subject to understand the overall basic computer hardware structure, including the peripheral devices.

Computer design and Computer Architecture

Computer design:- Computer design is concerned with the hardware design of a computer. Once the Computer specifications are formulated, it is the task of the designer to develop hardware for the system.

Computer Architecture:- Computer Architecture is a functional description of requirements and design implementation for the various parts of Computer. It deals with the functional behaviour of computer system. It deals with high level design issue. Architecture indicates its hardware. Architecture coordinates between the hardware and software of the system.



Computer System

Register transfer Language and microoperations

Registers are a type of computer memory used to quickly accept, store, and transfer data and instructions that are being used immediately by the CPU. The registers used by the CPU are often termed as processor registers.

→ A processor register may hold an instruction, a storage address or any data.

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

Micro-operations:- A micro operation is an elementary operation performed on the information stored in one or more registers.

→ The result of microoperation may be stored in source or another register.

→ Example of microoperations are load, store, clear, shift, addition, count etc.

→ A sequence of micro operations were performed to complete one operations.

For example

TO add two numbers following micro operation sequence has to be performed.

1. Load first number in register 1
2. Load second number in register 2
3. Perform add micro operation
4. store the result in the destination register 3

How can we write above lengthly description in Symbolic notation.

$R_1 \leftarrow A$

$R_2 \leftarrow B$

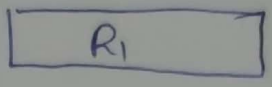
$R_3 \leftarrow R_1 + R_2$

Register transfer Language :-

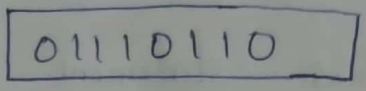
The Symbolic notation used to describe the micro-operation among registers called Register transfer Language.

There are various methods of RTL-

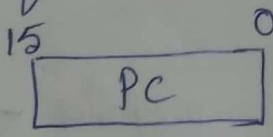
→ General way of representing a register by the name of the register enclosed in a rectangular box.



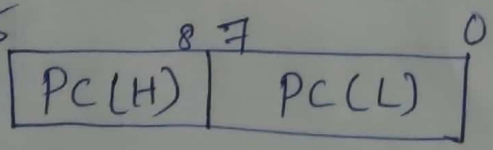
→ Register is numbered in a sequence of 0 to n-1.



→ The numbering of bits in a register can be marked on the top of the box.



→ A 16-bit register PC is divided into 2 parts. Bits (0 to 7) are assigned with lower byte of 16-bit address and bits (8 to 15) are assigned with higher bytes of 16-bit address.



Register transfer :-

The information transformed from one register to another register is represented in symbolic form by replacement operator is called Register transfer.

. Replacement Operator :-

In the statement, $R_2 \leftarrow R_1$, $\leftarrow$ acts as a replacement operator. This statement defines the transfer of content of register R_1 into register R_2.

Basic symbols of RTL :-

( ) → denotes a part of register - R_1 (8-bit) (0-7)

$\leftarrow$ → denotes a transfer of information - $R_2 \leftarrow R_1$

, → specify two micro-operations of register transfer. - $R_1 \leftarrow R_2$, $R_2 \leftarrow R_1$

: → denotes Conditional Operations - $P: R_2 \leftarrow R_1$ if $P = 1$

Register transfer Operations

The operation performed on the data stored in the registers are referred to as register transfer operations.

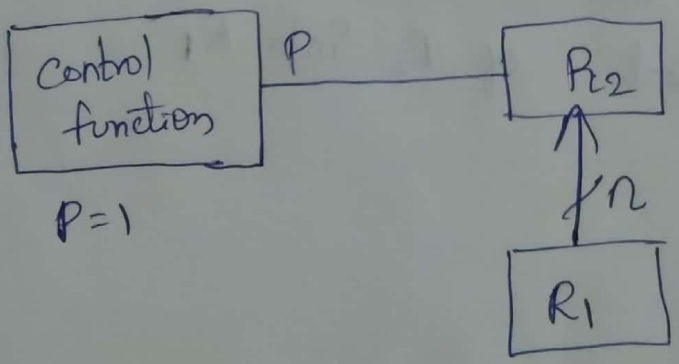
There are different types of register transfer operations:

1. Simple transfer - $R_2 \leftarrow R_1$

The content of R_1 are copied into R_2 without affecting the content of R_1. It is an unconditional type of transfer operation.

2. Conditional transfer :-

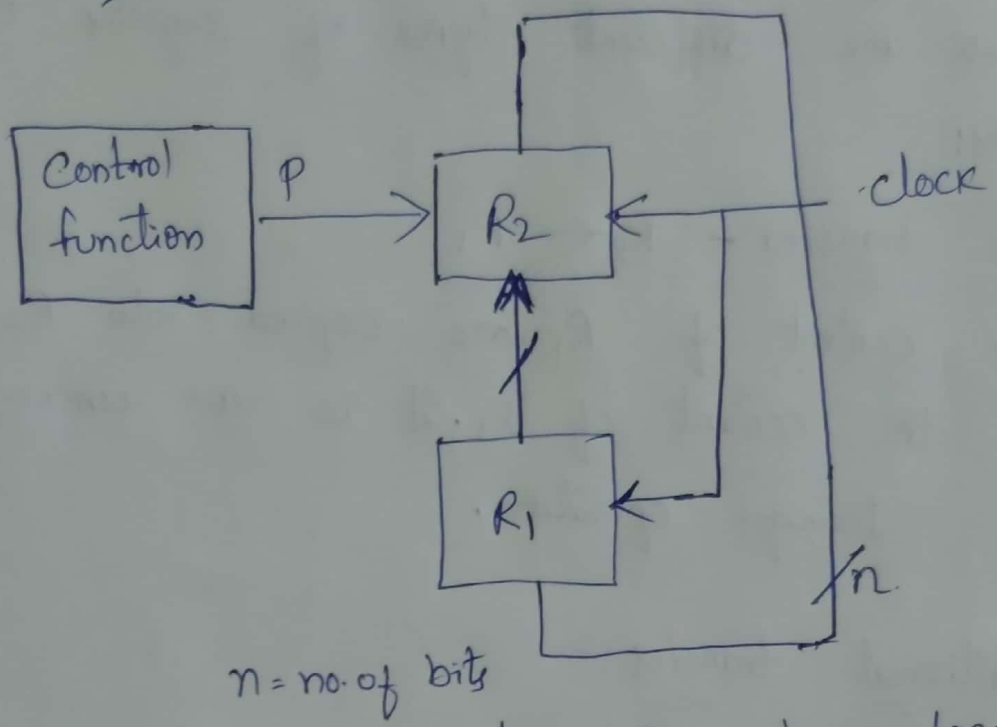
$P : R_2 \leftarrow R_1$



It indicates that if P=1, then the content of R_1 is transferred to R_2. It is a unidirectional operation.

3. Simultaneous Operations :-

If 2 or more operations were to occur simultaneously then they were seperated with comma(,)



If the control fuition P=1, then load the content of R_1 into R_2 and at the same clock load the content of R_2 into R_1

Bus and Memory transfers :-

A typical digital computer has many registers and paths must be provided to transfer information from one register to another. The number of wires will be excessive if seperate lines are used between each register and all other registers in the system.

→ A more efficient scheme for transferring information between registers in a multiple-register configuration is a common bus system.

→ A bus structure consist of a set of common lines, one for each bit of a register through which binary information is transferred one at a time.

→ Here we will see two ways to construct the bus.

a) By using multiplexers

b) By using three state buffer.

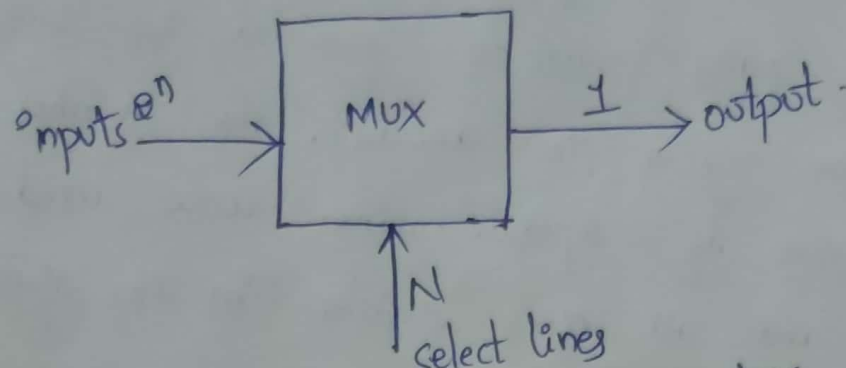
a) By using Multiplexers :- A Multiplexer is a digital switch that has multiple inputs and a single output.

→ The select lines determine which input is connected the output.

→ Multiplexer is denoted as MUX

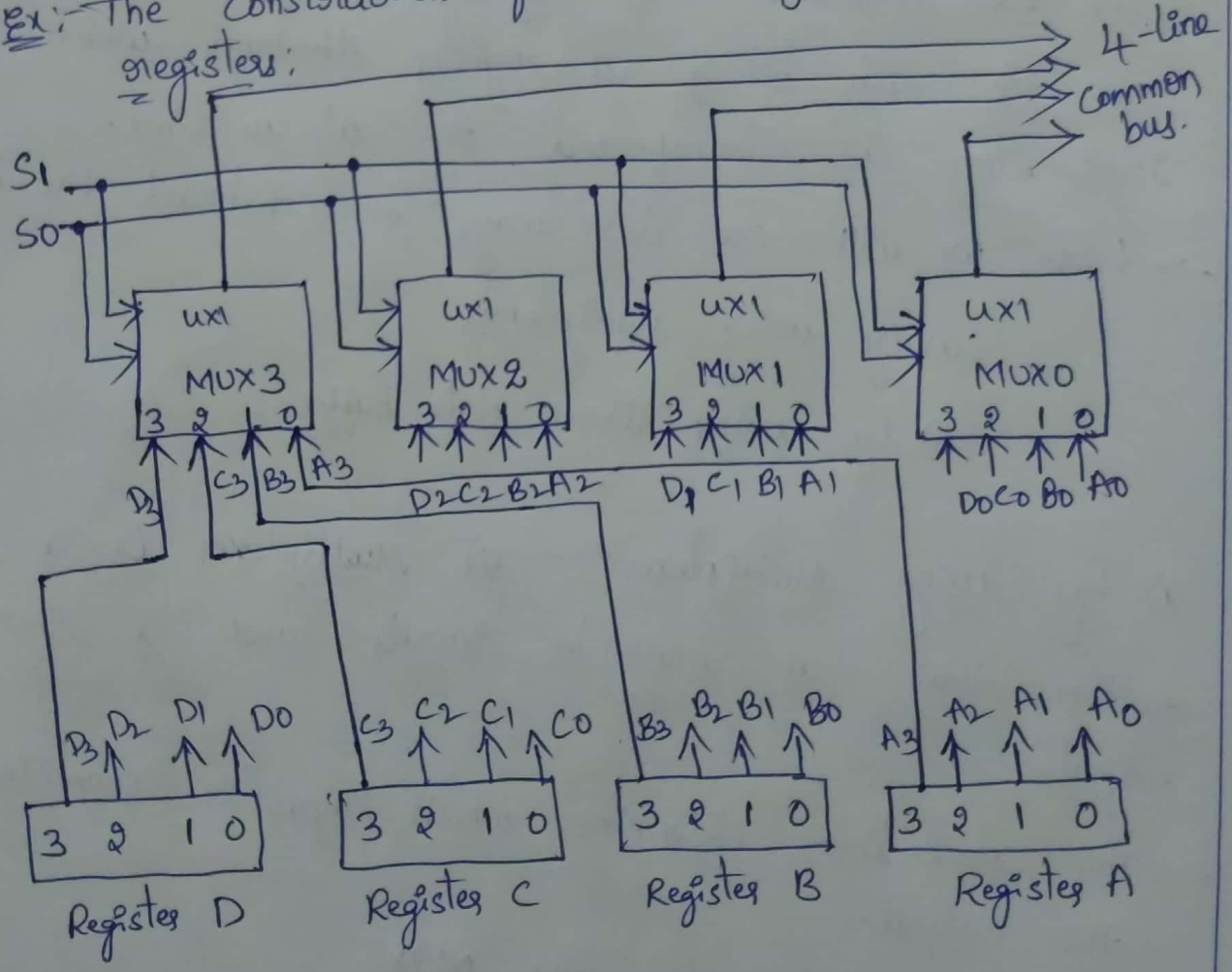
→ MUX Types :- 2 to 1 (1 select line)
4 to 1 (2 select lines)
8 to 1 (3 select lines)
16 to 1 (4 select lines)

Multiplexer block diagram



One way of constructing a common bus system is with multiplexers. The multiplexers select the source register whose binary information is then placed on the bus.

Ex:- The construction of a bus system for four registers:



Function Table for Bus.

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

→ Description of 4-Register bus :-

→ Each register has four bits, numbered 0 to 3. The bus consist of four 4×1 multiplexers each having four data inputs, 0 to 3, and two (selected lines) selection inputs, S_1 and S_0. In order not to complicate the diagram with 16 lines crossing each other, we use labels to show the connections from the outputs of the registers to the inputs of the multiplexers.

→ S_1 and S_0 selection lines should be applied to all MUX0 MUX1, MUX2, MUX3.

→ each MUX receives input from all registers. for example MUX0 receives input from output 0 bit of all the registers. i.e Register A '0' bit is A_0, Register B '0' bit is B_0, Register C '0' bit is C_0 and Register D '0' bit output is D_0.

and similarly MUX1 receives the inputs from '1' bit of four registers and remaining 2 mux will receive the bits from 2 bit & 3 bit of all registers.

→ The two selection lines S_1 and S_0 are connected to the selection inputs of all four multiplexers.

→ When $S_1S_0 = 00$, the 0 data inputs of all 4 multiplexers are selected and applied to the outputs that form the bus. This causes the bus lines to receive the content of Register A

→ Similarly Register B is selected if $S_1S_0 = 01$ and so on.

→ The transfer of information from a bus into one of many destinations registers can be accomplished by connecting the bus lines to the inputs of all destination registers and activating the load control

→ The Symbolic statement for bus transfer is

$$BUS \leftarrow C, \quad R_1 \leftarrow BUS$$

The above syntax represents the content of register C is placed on the bus, and the content of bus is loaded into register R_1 by activating its load control input.

=

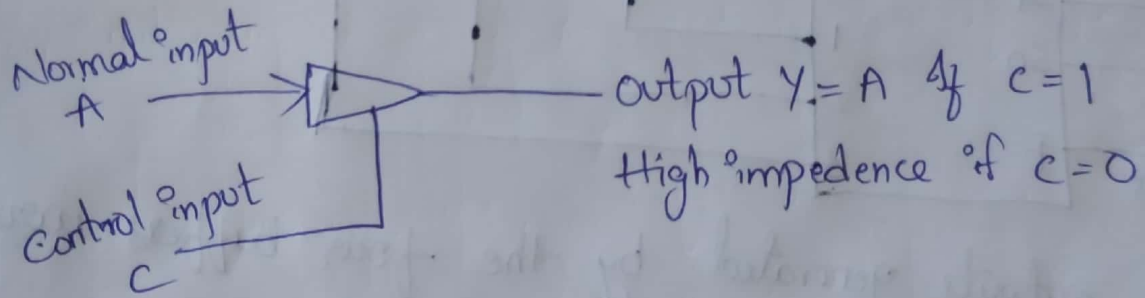
Bus transfer using three state buffer :-

→ A bus system can also be constructed using three-state-buffer. A three state buffer is a digital circuit that has three gates,

1. logic 1
2. logic 0
3. the third gate exhibits a high-impedence state.

→ In high-impedence state the circuit behaves as a open circuit, and the output transfer is blocked.

→ Therefore a three state buffer is a combinational circuit that acts like a simple switch.

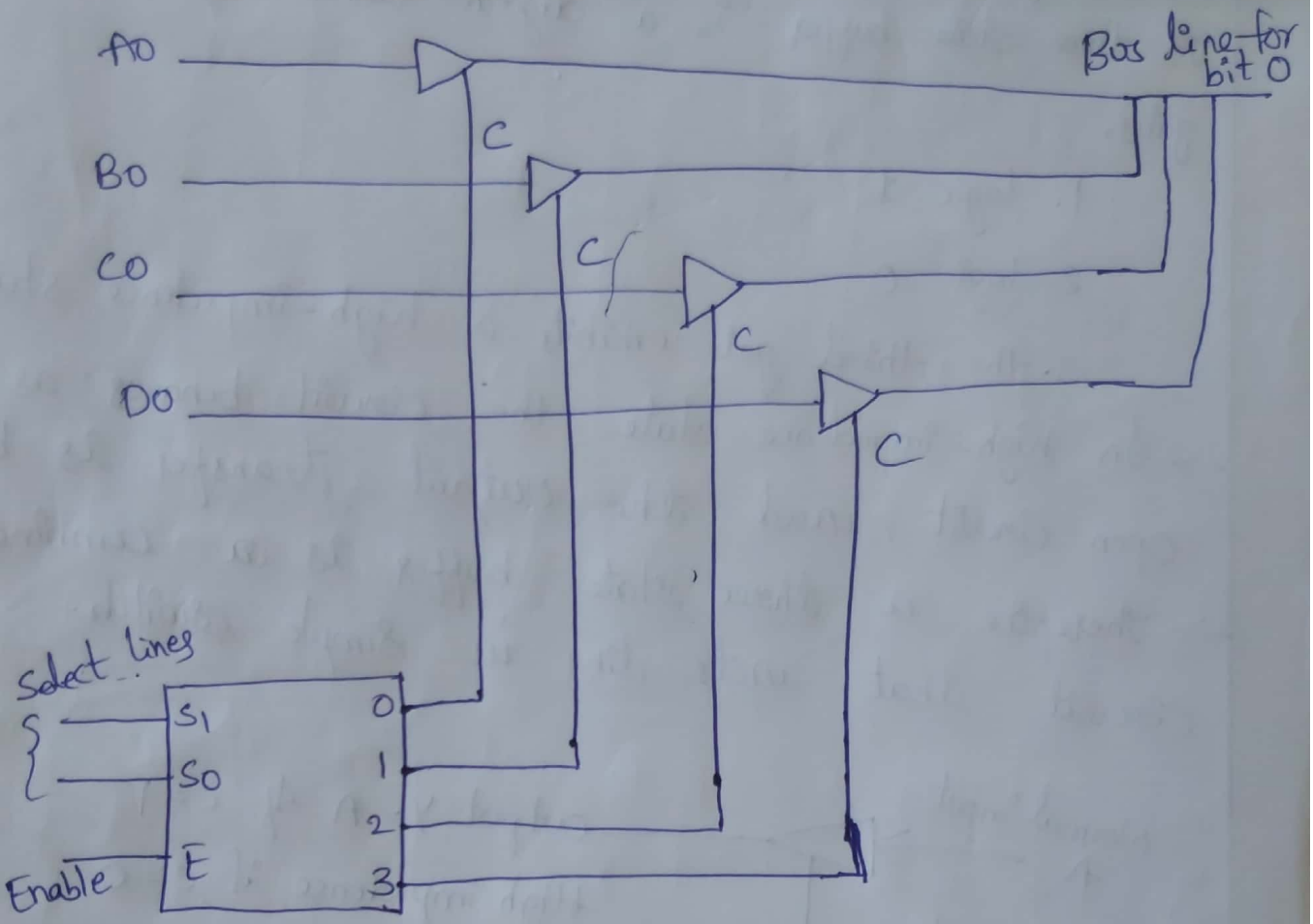


Graphic Symbols for three-state buffers

→ If control input C = 1 then it passes the input signal to the output.

→ If C = 0 then the electrical current will be blocked. this case is called high-impedence state.

Bus transfer using three state buffer



→ The outputs generated by the four buffers are connected to form a single bus line.

→ Only one buffer can be in active state at a given point of time.

→ The control inputs to the buffers determine which of the four normal inputs will communicate with the bus line.

→ A 2*4 decoder ensures that no more than one control input is active at any given point of time.

→ A_0, B_0, C_0, D_0 are the 0^{th} bits of registers A, B, C, D respectively.

→ S_1, S_0 are the selector lines used to give the control signal.

S_1	S_0	control input
0	0	0 is selected
0	1	1 is selected
1	0	2 is selected
1	1	3 is selected.

Memory transfer

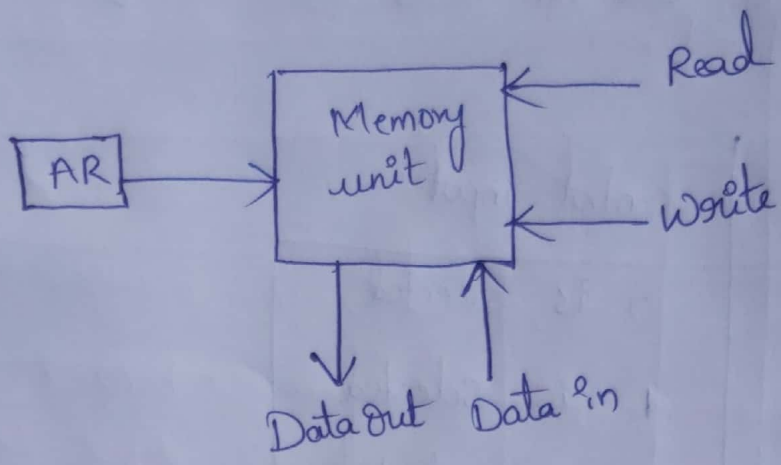
The transfer of information from a memory word to the outside environment is called a read operation.

→ The transfer of a new information to be stored into the memory is called a write operation.

→ A memory word will be symbolized by the letter M.

→ We must specify the address of memory word while writting the memory transfer operations.

→ The address register is designated by AR and the data register is by DR.



→ A read operation can be stated as:

Read : DR ← M[AR]

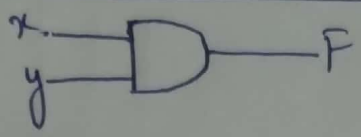
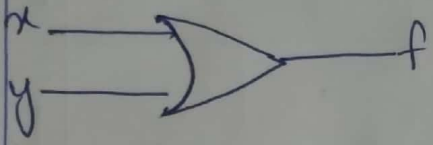
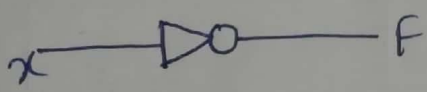
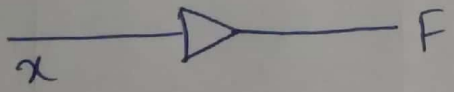
The above statements indicates that a transfer of information into the data register [DR] from the memory word [M] selected by the address register (AR).

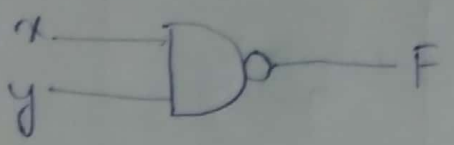
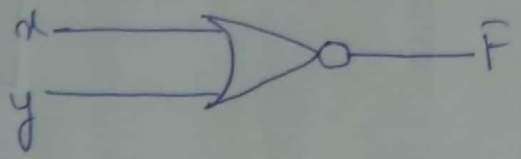
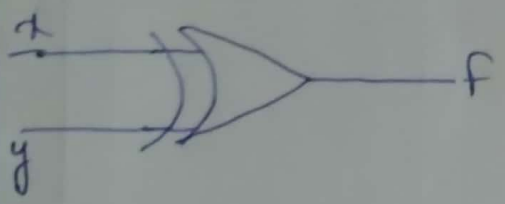
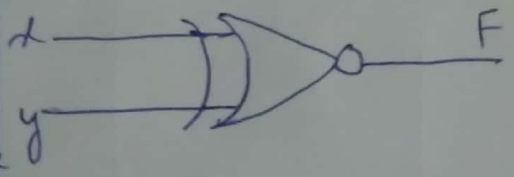
→ The Write operation can be stated as:

Write : M[AR] ← R_1

The above statement describes that a transfer of information from R_1 into the memory word (M) selected by the address register (AR).

Logic gates :-

	Name	Graphic Symbol	Algebraic equation	Truth table
1	AND		F = x * y	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
2	OR		f = x + y	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
3	Inverter (or) NOT		F = x'	x | F 0 | 1 1 | 0
4	Buffer		F = x	x | F 0 | 0 1 | 1

	Name	Graphic Symbol	Algebraic function	Truth table
5	Nand		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
6	NOR		$F = (x+y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
7.	Exclusive - OR (XOR)		$f = xy' + x'y$ (or) $\Rightarrow x \oplus y$	x y | f 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
8	Exclusive - NOR (or) Equivalence		$F = xy + x'y'$ (or) $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Addition of two binary numbers

Eg: $R_1 = 10010$

$R_2 = 11010$

$R_3 \leftarrow R_1 + R_2$

```
  00100  → Carry value.
  10010
+ 11010
-------
 |1|01100  → Sum value
```

MSB (most significant bit)

Addition of 4 bit binary number.

Eg. $R_1 = 1101$

$R_2 = 1011$

$R_3 \leftarrow R_1 + R_2$

```
  1110  → Carry value.
  1101
+ 1011
------
 |1|1000  → Sum value.
```

MSB

Sum value = 1000

Carry value = 1110.

Arithmetic Micro Operations :-

Arithmetic micro-operations perform arithmetic operation on numerical data stored in registers.

→ The basic arithmetic micro-operations are

- → Addition
- → Subtraction
- → Increment
- → Decrement
- → shift.

	Symbolic designation	Description
1	$R_3 \leftarrow R_1 + R_2$	Contents of R_1 plus R_2 transferred to R_3
2	$R_3 \leftarrow R_1 - R_2$	contents of R_1 minus R_2 transferred to R_3
3	$R_2 \leftarrow \bar{R}_2$	Complements the content of R_2 (1's complement)
4	$R_2 \leftarrow \bar{R}_2 + 1$	2's complement the content of R_2 (negate).
5	$R_3 \leftarrow R_1 + \bar{R}_2 + 1$	R_1 plus the 2's complement of R_2 (subtraction)
6	$R_1 \leftarrow R_1 + 1$	Increment the contents of R_1 by one
7	$R_1 \leftarrow R_1 - 1$	Decrement the contents of R_1 by one.

Addition :-

Binary Adder

To implement the add micro operation with hardware, we need the registers that hold the data and the digital component that performs the arithemetic addition.

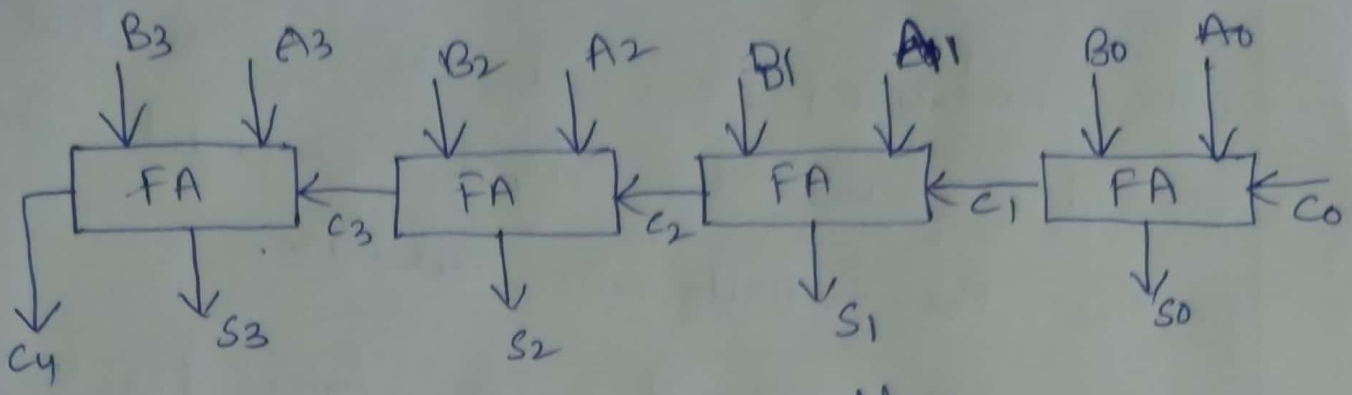
→ The digital circuit that forms the arithmetic sum of two bits and a previous carry is called a full-adder.

→ The digital circuit that generates the arithmetic sum of two binary numbers of any length is called binary adder.

→ The binary adder is constructed with full adder circuits connected in cascade.

→ The output carry from one full-adder connected to the input carry of the next full-adder.

→ The interconnections of four full adders (FA) to provide a 4-bit binary adder.



4- bit binary adder

→ The augend bits of A and the addend bits of B are designated by subscript numbers from right to left, with subscript '0' denoting the low order bit.

→ The carries are connected in a chain through the full adders.

→ The input carry to the binary adder is C_0 and the output carry is C_4.

→ The S outputs of the full-adders generate the required sum bits.

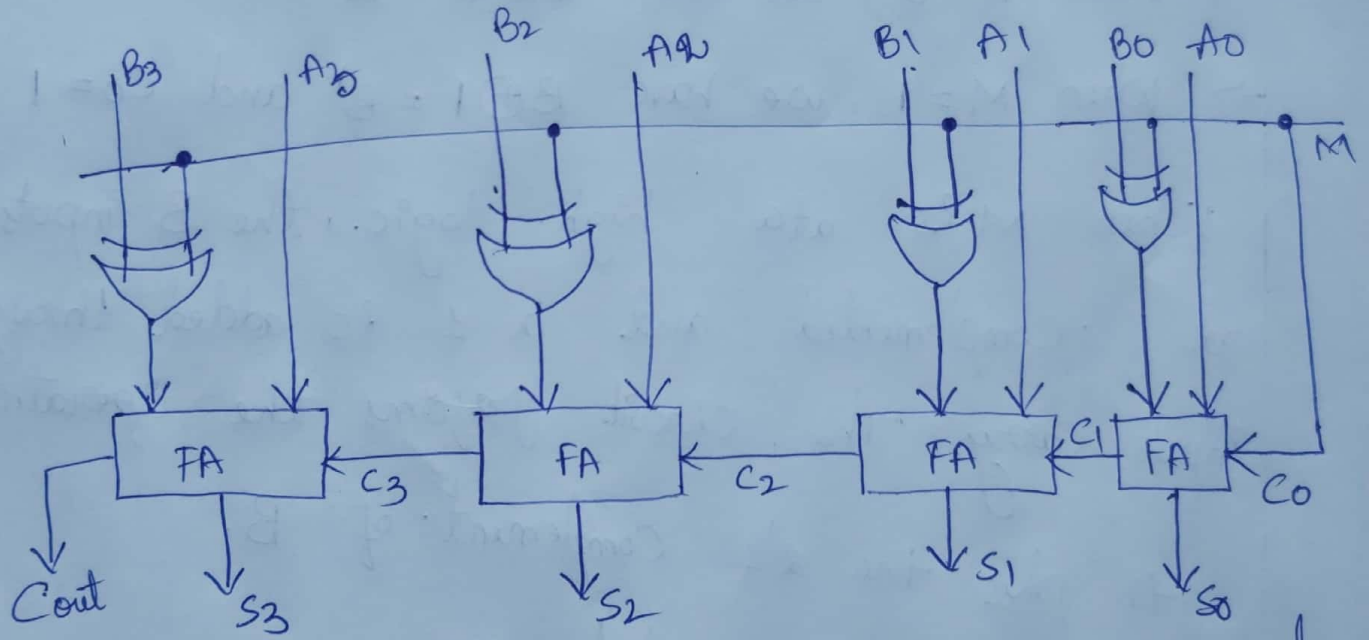
→ An n-bit binary adder requires n full-adders.

4-bit binary Adder-Subtractor :-

In digital circuits, A Binary-Subtractor is one which is capable of both addition and subtraction of binary numbers in one circuit itself.

→ The operation being performed depends upon the binary value the control holds.

→ This circuit requires Exor gate, Binary addition and subtraction, full adder.



→ Lets Consider two 4-bit binary numbers A and B as inputs to the digital circuit for the operation.

A_0 A_1 A_2 A_3 for A

B_0 B_1 B_2 B_3 for B

→ The circuit consist of 4 full adders since we are performing operation on 4-bit numbers.

→ There is a mode operation M that holds a binary value of either 0 or 1 which determines that the operation being carried out is addition or subtraction.

→ When M=0 the circuit is adder and when M=1 the circuit becomes a subtractor.

→ When M=0, we have $B \oplus 0 = B$.

→ When M=1, we have $B \oplus 1 = B'$ and $C_0 = 1$

When M is at a high logic, the B inputs are complemented and a 1 is added through input carry. The circuit performs the operation A plus the 2's complement of B.

→ It can be represented as

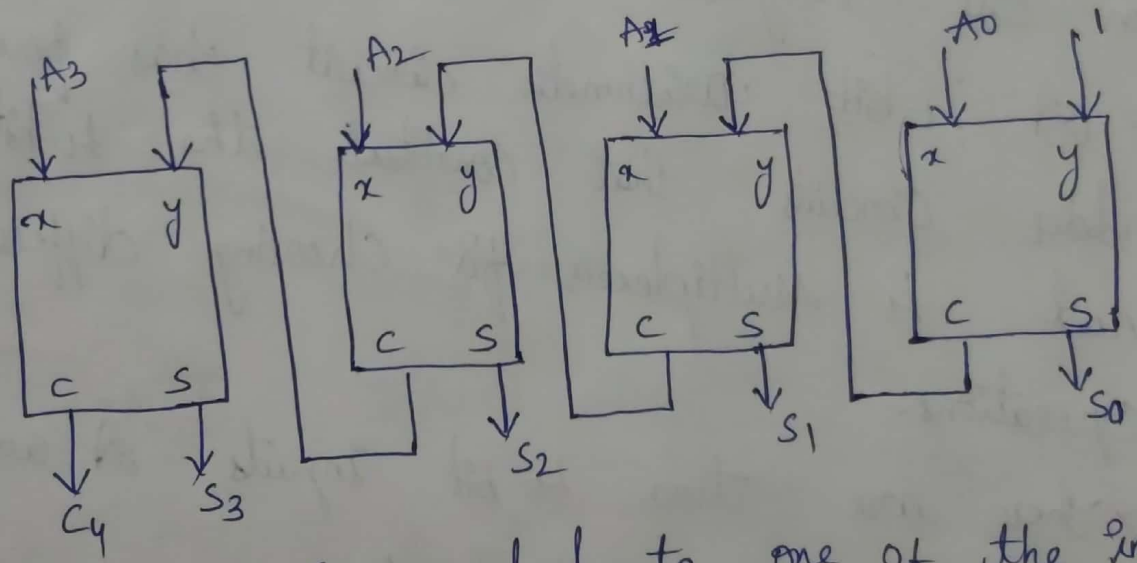
$$R \leftarrow A + \overline{B} + 1$$

Binary Incrementer:-

The increment micro-operation adds one binary value to the value of binary variables stored in a register.

→ For instance a 4-bit register has a binary value 0101, when incremented by one the value becomes 0110.

→ A 4-bit Combinational circuit incrementer can be represented by the following block diagram.



→ A logic-1 is applied to one of the inputs of least significant half-adder, and the other input is connected to the least significant bit of the number to be incremented.

→ The output Carry from one half-adder is connected to one of the inputs of the next-higher-order half-adder.

* The binary incrementer circuit receives the four bits from A_0 through A_3, adds one to it, and generates the incremented output in S_0 through S_3.

→ The output carry C_4 will be 1 only after incrementing binary 1111

Arithmetic Circuit :-

→ The arithmetic microoperations can be implemented in one composite arithmetic circuit.

→ A 4-bit arithmetic circuit has four full-adder circuits that constitute the 4-bit adder and 4- multiplexers for choosing different operations.

→ There are two 4-bit inputs A and B and a 4 bit output D.

→ The 4 inputs from A go directly to the X inputs of the binary adder.

→ Each of output of multiplexer is go to the y inputs of the binary adder.

→ The input Carry C_{in} goes to the Carry input of the full adder in the least significant position. The other Carries are Connected from one stage to the next.

→ Each of four inputs (B_0 B_1 B_2 B_3 / 0, 1, 2, 3) from B are Connected to the data inputs (0) of the multiplexers.

→ The Multiplexer data inputs (1) receive the complements of B.

→ The other two data inputs (2, 3) are Connected to logic-0 and logic-1

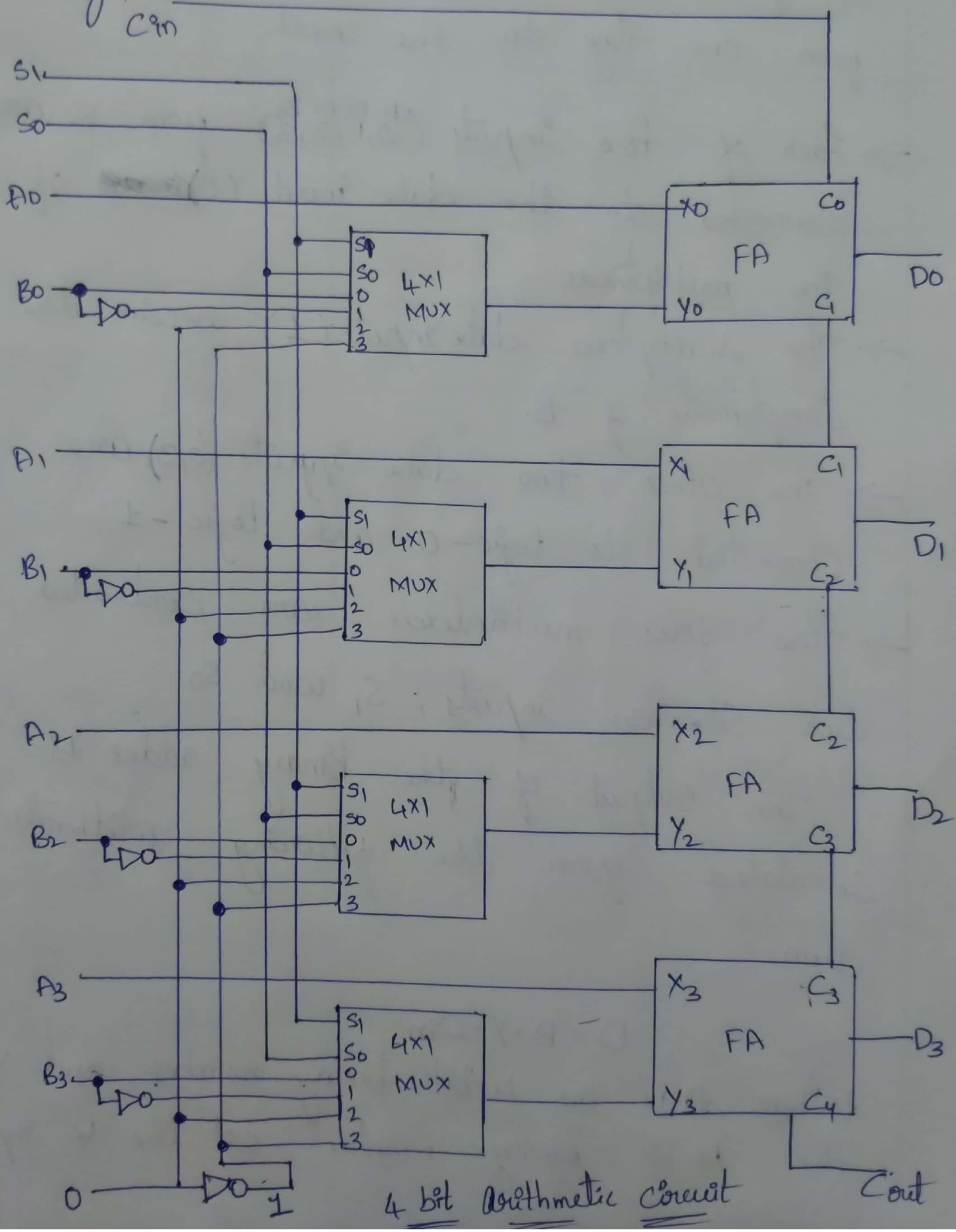
→ The four multiplexers are controlled by two Selection inputs; S_1 and S_0

The output of the binary adder is calculated from the following arithmetic sum:

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

Where A is the 4-bit binary number and Y is the 4-bit binary number and the Y inputs

of the binary adder. Cin is the input Carry., which can be equal to 0 & 1



4 bit Arithmetic Circuit

Arithmetic Circuit function table

Select S_1	Select S_0	Select C_{in}	Input Y	Output $D = A + Y + C_{in}$	Microoperation
0	0	0	B	$D = A + B$	Add
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

When $S_1S_0 = 00$, the value of B is applied to the Y inputs of the adder. if $C_{in} = 0$, the output $D = A + B$, if $C_{in} = 1$, output $D = A + B + 1$. both cases perform addition operation with(or) without adding the input carry.

→ When $S_1S_0 = 01$, the complement of B is applied to the Y inputs of the adder, if $C_{in} = 1$. then output $D = A + \bar{B} + 1$, which is equivalent to a subtraction of A - B.

→ When $S_1S_0 = 10$, all 0's are inserted into the Y inputs. The output is $D = A + 0 + C_{in}$ i.e $D = A + C_{in}$

when $C_{in} = 0$, then output $D = A + 0$ i.e $D = A$.

when $C_{in} = 1$, then output $D = A + 1$

→ The first case we have a direct transfer from input A to output D., in the second case the value of A is incremented by 1.

→ When $S_1S_0 = 11$, all 1's are inserted into the Y inputs of the adder to produce the decrement operation $D = A - 1$, when $C_{in} = 0$, This is because a number with all 1's is equal to the 2'complement of 1 (the 2's complement of binary 0001 is 1111)

→ When $C_{in} = 1$ then $D = A - 1 + 1 \Rightarrow A$, which causes a direct transfer from input A to output D.

* The microoperation $D = A$ is generated twice, so there are only seven distinct microoperations in the arithmetic circuit

Logic Microoperations

Logic microoperations specify binary operations for strings of bits stored in registers. These operations consider each bit of the register seperately and treat them as binary variables. For example the exclusive-OR microoperation with the contents of two registers R_1 and R_2.

$$P: R_1 \leftarrow R_1 \oplus R_2$$

Eg:

```
  1010   Contents of R1
  1100   contents of R2
 ------
  0110   Content of R1 after P=1
```

→ Special symbols will be adopted for the logic microoperations OR, AND, and Complement. to represent them from the corresponding symbols used to express boolean functions.

→ The Symbol $\vee$ will be used to denote an OR microoperation.

→ The Symbol $\wedge$ to denote an AND operation.

→ The complement microoperations is the same as the 1's complement and uses a bar on top of the symbol

List of logic microoperations :-

There are 16 different logic operations that can be performed with two binary variables

→ They can be determined from all possible truth tables obtained with two binary variables each of the 16 columns F0 through F15 represents a truth table.

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

The 16 boolean functions of two variables x and y are expressed in an algebraic form in the first column.

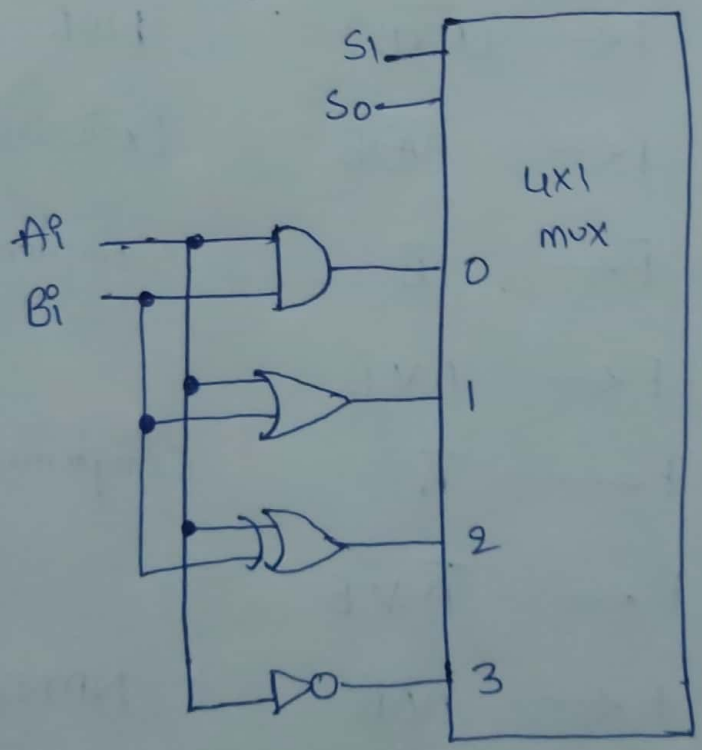
	Boolean function	Microoperation	Name
1	$F_0 = 0$	$F \leftarrow 0$	Clear
2	$F_1 = xy$	$F \leftarrow A \wedge B$	AND
3	$F_2 = xy'$	$F \leftarrow A \wedge \bar{B}$	
4	$F_3 = x$	$F \leftarrow A$	Transfer A
5	$F_4 = x'y$	$F \leftarrow \bar{A} \wedge B$	
6	$F_5 = y$	$F \leftarrow B$	Transfer B
7	$F_6 = x \oplus y$	$F \leftarrow A \oplus B$	Exclusive-OR
8	$F_7 = x + y$	$F \leftarrow A \vee B$	OR
9	$F_8 = (x+y)'$	$F \leftarrow \overline{(A \vee B)}$	NOR
10	$F_9 = (x \oplus y)'$	$F \leftarrow \overline{A \oplus B}$	Exclusive-NOR
11	$F_{10} = y'$	$F \leftarrow \bar{B}$	Complement B
12	$F_{11} = x + y'$	$F \leftarrow A \vee \bar{B}$	
13	$F_{12} = x'$	$F \leftarrow \bar{A}$	Complement A
14	$F_{13} = x' + y$	$F \leftarrow \bar{A} \vee B$	
15	$F_{14} = (xy)'$	$F \leftarrow \overline{A \wedge B}$	NAND
16	$F_{15} = 1$	$F \leftarrow$ all 1's	Set to all 1's

Hardware Implementation :-

The hardware implementation of logic micro-operations requires that logic gates be inserted for each bit or pair of bits.

→ Although there are 16 logic microoperations, most computers use only four - AND, OR, XOR (exclusive OR), and Complement from which all others can be derived.

→ Logic microoperations are very useful for manipulating individual bits or a portion of a word stored in register.



One stage of logic circuit

S_1	S_0	O/P	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

Function table

Shift microoperations

Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic and other data processing operations. The contents of a register can be shifted to the left or the right. At the same time that the bits are shifted.

→ during a shift left operation the serial input transfer's a bit into a bit into the rightmost position

→ During a shift right operation the serial input transfers a bit into the leftmost position.

→ there are three types of shifts:

- → Logical shift
- → Circular shift
- → Arithmetic shift.

Logical shift :- A logical shift is one of that transfers '0' through the serial input. We will adopt the symbols shl and shr for logical shift-left and shift right microoperations.

$R_1 \leftarrow shl\ R_1$

$R_2 \leftarrow shr\ R_2$

The two microoperations specify a 1-bit shift to the left of the content of Register R_1 and a 1-bit shift to the right of the content of register R_2. The register symbol must be the same on both sides of the arrow. The bit transferred to the end position through the serial input is assumed to be 0 during a logical shift.

Circular shift :- The circular shift also known as rotate operation circulates the bits of the register around the two ends without loss of information.

→ This is accomplished by connecting the serial input of the shift register to its serial input. We will use the symbols cil and cir for the circular shift left and right respectively.

$R_1 \leftarrow$ cil R_1

$R_2 \leftarrow$ cir R_2

Arithmetic shift :- An arithmetic shift is a micro-operation that shifts a signed binary number to the left & right.

→ An arithmetic shift-left multiplies a signed binary number by 2.

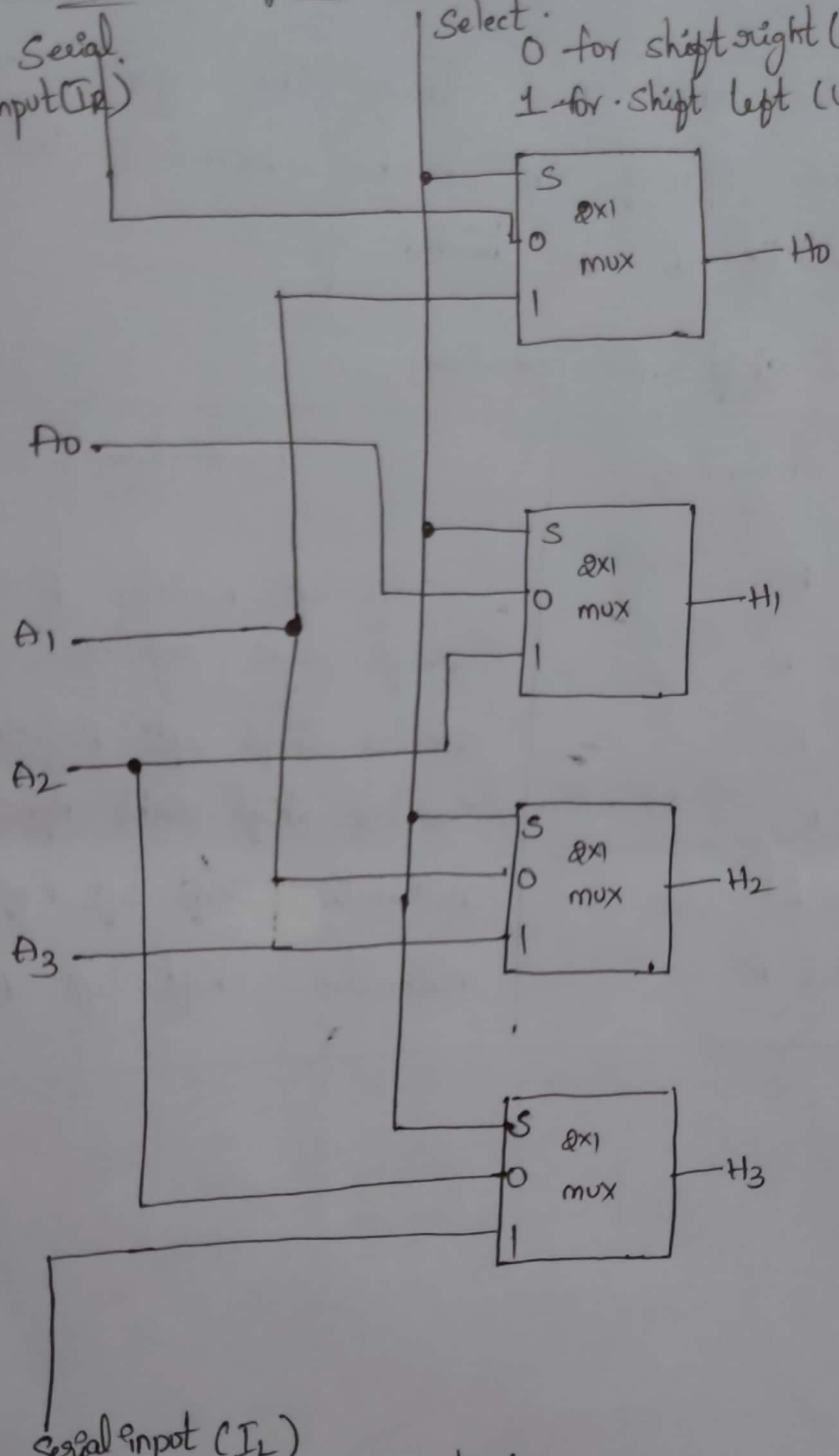
→ An arithmetic shift right divides the number by 2.

→ Arithmetic shift must leave the sign bit unchanged because the sign of the number remains the same when it multiplied & divided by 2.

shift micro operations

Symbolic designation	Description
R ← shl R	shift-left Register R
R ← shr R	shift-right register R
R ← cil R	circular shift-left Register R
R ← cir R	circular shift-right Register R
R ← ashl R	arithmetic shift-left Reg.
R ← ashr R	arithmetic shift-left R

Hardware implementation



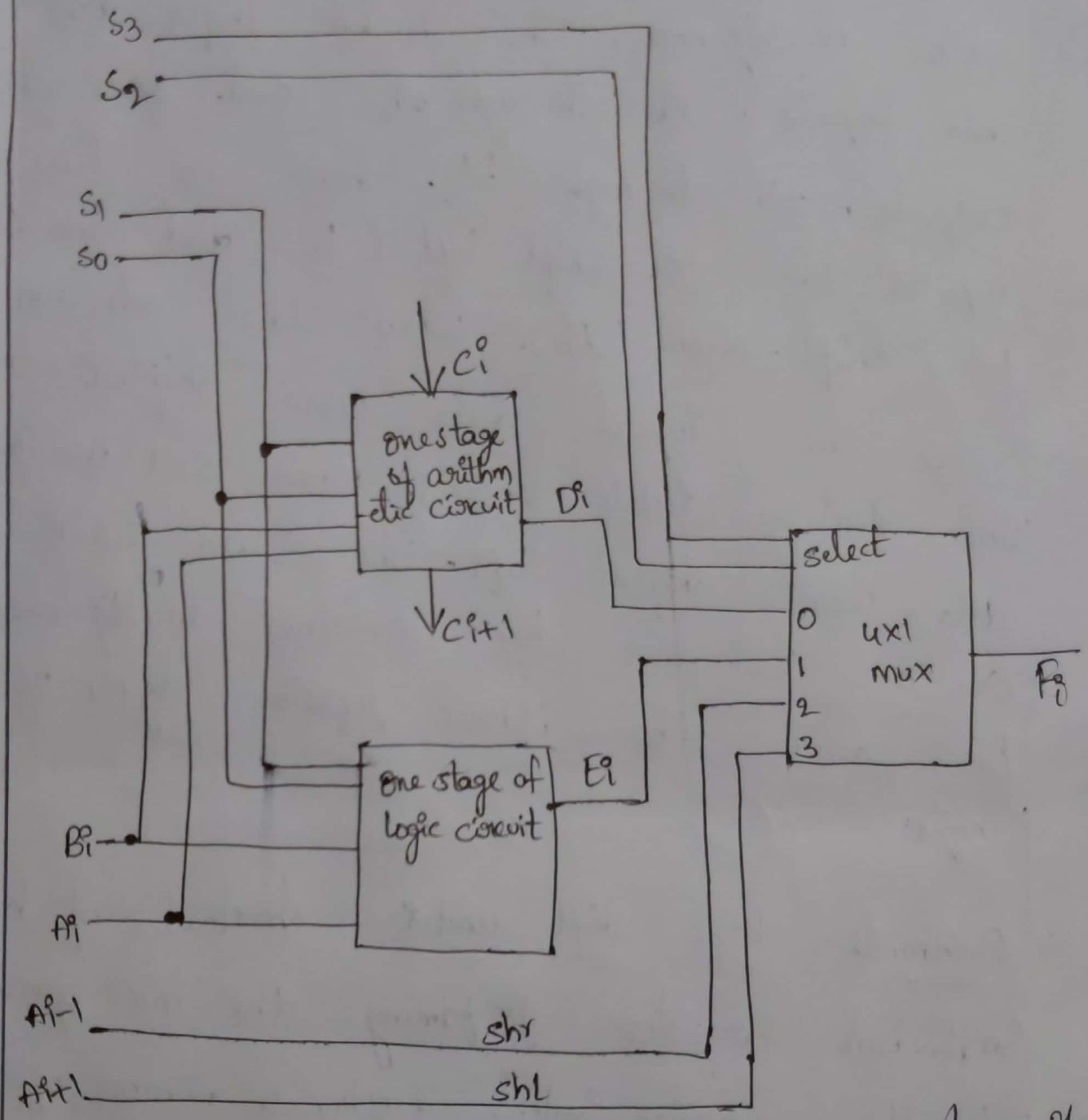
Function table

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

A Combinational Circuit shifter can be constructed with multiplexers. The 4-bit shifter 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 the other for shift right (I_R). When the selection input $S=0$ the input data are shifted right. The function table shows. When $S=1$ the input data are shifted left. The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

* Arithmetic Logic shift unit*- Instead of having individual registers performing the microoperations directly, Computer Systems employ a number of storage registers connected to a common operational unit called arithmetic logic unit, abbreviated ALU.

one stage of arithmetic logic shift unit



→ the above diagram represents the one stage of arithmetic logic shift unit. the subscript 'i' describes a typical stage. Inputs Ai and Bi are applied to both the arithmetic and logic units. A particular micro operation is selected with Inputs S_1 and S_0.

→ A 4x1 multiplexer at the output choosen between an arithmetic circuit output D_i and a logic output E_i. The data in the multiplexer are selected with inputs S_3 and S_2. The other two data inputs to the multiplexer receive inputs A_{i-1} for the shift right operation and A_{i+1} for the shift left operation.

→ The output carry C_{i+1} of a given arithmetic stage must be connected to the input carry C_i of the next stage in sequence.

→ The circuit whose one stage is specified in provides eight arithmetic operations, four logic operations and two shift operations.

→ the first eight are arithmetic operations and are select with $S_3 S_2 = 00$. The next four are logic operations and are selected with $S_3 S_2 = 01$.

→ The last two operations are shift operations and are selected with $S_3 S_2 = 10$ and 11.

The oth

Function table for Arithmetic logic shift unit

operation select :-

S_3	S_2	S_1	S_0	C_{in}	Operation	function
0	0	0	0	0	F=A	Transfer A
0	0	0	0	1	f=A+1	Increment A
0	0	0	1	0	F=A+B	Addition
0	0	0	1	1	F=A+$\bar{B}$+1	Add with carry
0	0	1	0	0	F = A+$\bar{B}$	Sub with borrow
0	0	1	0	1	F = A+$\bar{B}$+1	Subtraction
0	0	1	1	0	F = A−1	Decrement A
0	0	1	1	1	F = A	Transfer A
0	1	0	0	X	F = A∩B	AND
0	1	0	1	X	F = A∨B	OR
0	1	1	0	X	F = A⊕B	XOR
0	1	1	1	X	F = $\bar{A}$	Complement A
1	0	X	X	X	F = shr A	shift right A into F
1	1	X	X	X	F = shl A	shift left A into F

Basic Computer Organization and Design

Instruction codes :-

→ A digital computer is capable of executing several microoperations.

→ An instruction code is a group of bits that instructs the computer to perform a specific operation.

Program:- A program is a set of instructions that specify the operations, operands, and the sequence by which processing has to be done.

→ An instruction code is a group of bits that instructs the computer to perform specific operation. it is usually divided into two parts, each having its own particular interpretation.

- → Operation code (opcode)
- → Operands.

The most basic part of an instruction code is its operation part.

→ The operation code (opcode) of an instruction is a group of bits that define each operation such as add, subtract, multiply, shift and complement.

→ It must consist of atleast n bits for a given 2^n distinct operations.

→ Suppose we are having 64 (2^6) operation. Then the length of opcode will be 6.

→ The operation part of instruction code specifies the operation to be performed.

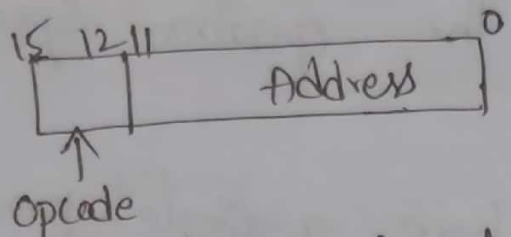
→ There are many ways to arrange the binary code of instructions, and each computer has its own particular instruction code format.

Stored program organization

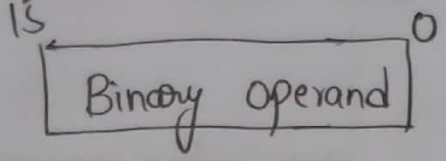
The simplest way to organize a computer is to have one processor register and an instruction code format with two parts.

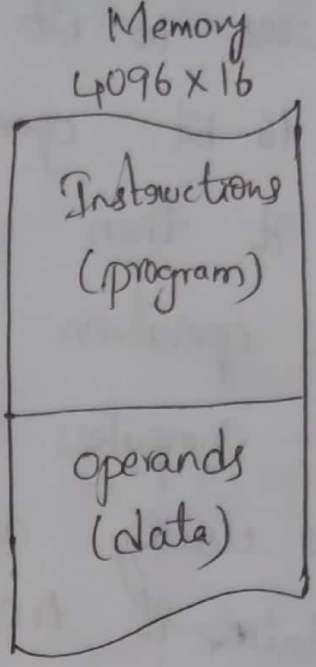
→ The first part specifies the operation to be performed and the second part specifies an address.

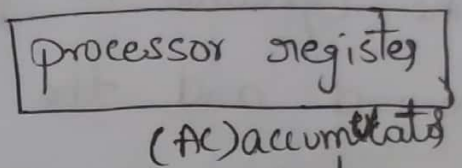
→ The memory address tells the ~~computer~~ control where to find a operand in memory.



Instruction format







→ Instructions are stored in one section of memory and data in another

→ For a memory unit with 4096 words we need 12 bits to specify an address since $2^{12} = 4096$.

→ If we store each instruction code in one 16-bit memory word, we have available four bits for the operation code (opcode) to specify one out of 16 possible operations, and 12 bits to specify the address of an operand.

→ The control reads a 16 bit instruction from the program portion of memory.

→ It uses 16 bit address part of instruction to read 16-bit operand from the data portion of memory, it then executes the operation specified by the operation code.

→ The computers that have a single-processor registers usually assign to it the name accumulator and label it AC.

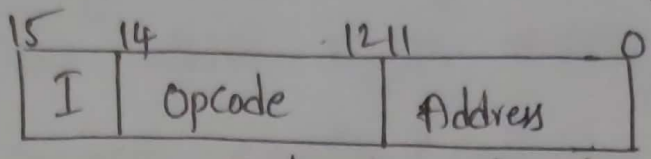
→ The operation is performed with the memory operand and the content of AC.

Immediate Operand :- Sometimes address bits of an instruction code is used to hold actual operand when second part of an instruction code specifies an operand, the instruction is said to have an immediate operand.

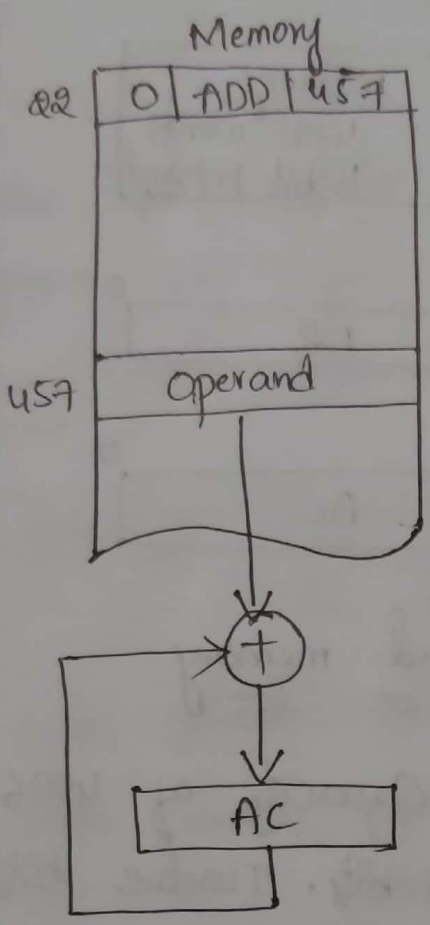
Direct address :- When second part of an instruction specifies the address of an operand, the instruction is said to have a direct address.

Indirect address :
When the bits of the second part of the instruction designates an address of a memory word in which the address of the operand is found.

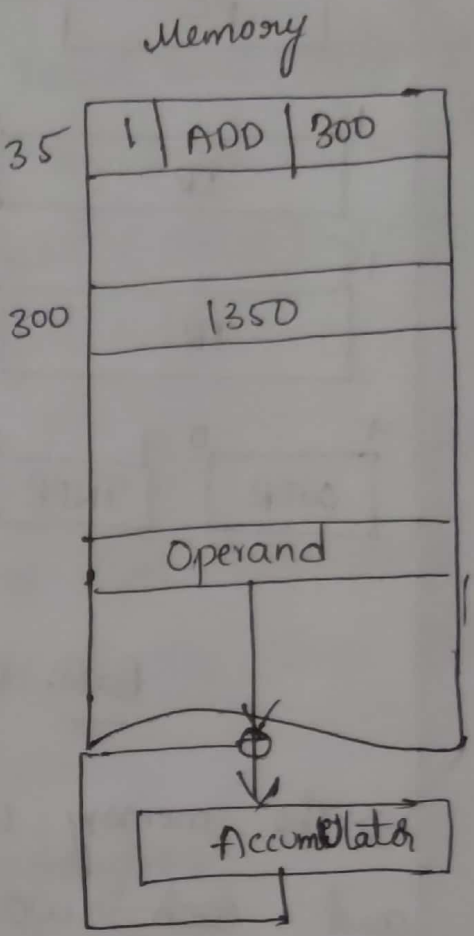
Direct and Indirect addressing example



(a) Instruction format



(b) Direct address



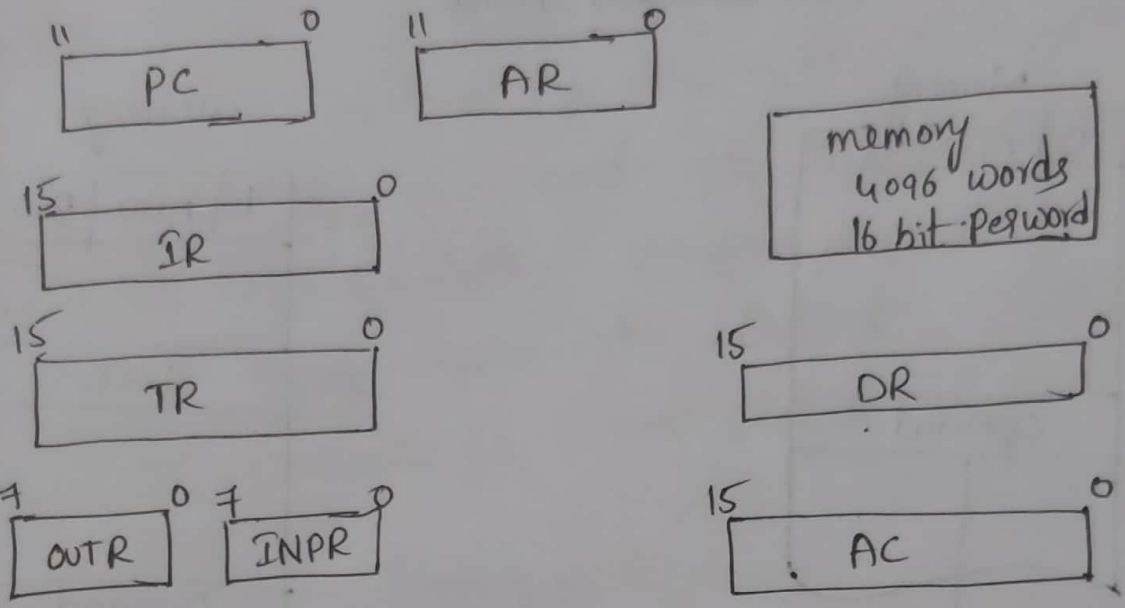
(c) Indirect address.

Computer registers :-

The basic Computer has 8 registers.

- → Data register
- → Address register
- → Accumulator
- → Instruction register
- → Program Counter
- → Temporary register

→ Input register
→ Output register.



Basic Computer registers and memory

→ The memory unit has a capacity of 4096 words and each word contains 16 bits. Twelve bits of an instruction word were needed to specify the address of an operand.

→ The DR (data register) holds the operand read from memory.

→ The accumulator (AC) register is a general purpose processing register. The instruction read from memory is placed in the instruction register (IR).

→ The temporary register (TR) is used for holding temporary data during the processing.

→ The memory address register (AR) has 12 bits since this is the width of a memory address.

→ The program counter (PC) also has 12 bits and it holds the address of the next instruction.

→ TWO registers are used for input and output. The input register (INPR) receives an 8-bit character from an input device. The output (OUTR) holds an 8-bit character for an output device.

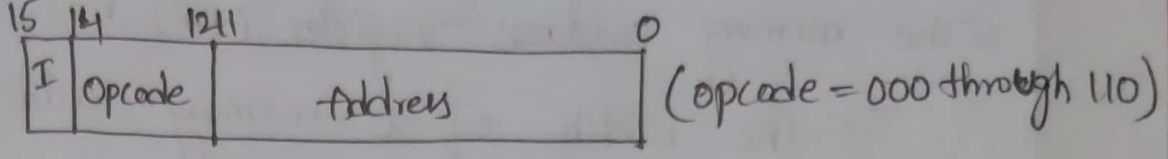
Computer instructions :-

The basic computer has three instruction code formats. Each format has 16 bits. The three instruction formats are

1. Memory-reference instruction
2. Register-reference instructions
3. Input-output reference instructions.

1. Memory-reference instruction :-

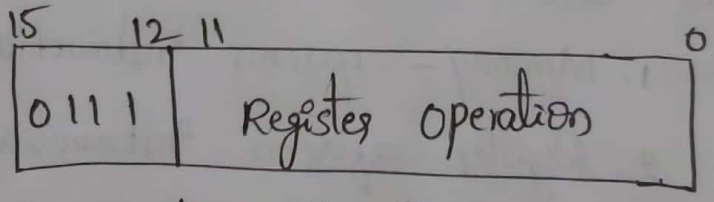
Memory reference instruction uses 12 bits to specify an address and one bit to specify the addressing mode I. I is equal to 0 for direct address and to 1 for indirect address.



(a) Memory - reference instruction

2. Register - reference instruction format :-

The register - reference instructions are recognized by the operation code 111 with a 0 in the leftmost bit (bit 15) of the instruction. A register - reference instruction specifies an operation on & a test of the AC register. An operand from memory is not needed, the other 12 bits are used to specify the operation & test to be executed.



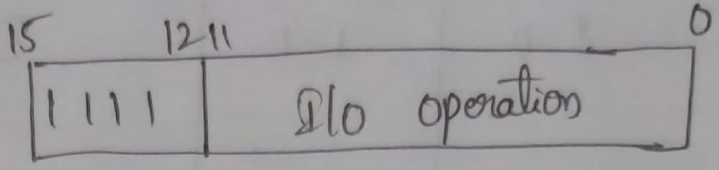
opcode = 111, I = 0

(b) Register - reference instruction

(3) Input - Output Operation instructions :-

Input-Output instructions does not need a reference memory and is recognized by the operation code 111 with 1 in the leftmost bit of the instruction.

→ The remaining 12 bits are used to specify the type of input-output operation or test performed.



opcode := 111, I = 1

(c). Input-Output Instruction

The type of instruction is recognized by the computer control from the four bits in positions 12 through 15 of the instruction. If the three opcode bits in positions 12 through 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.

→ If the 3-bit opcode is equal to 111, the 15 bit is 0, the instruction is a register-reference type. If the 15 bit is 1, the instruction is an input-output type.

Basic Computer Instructions

Symbol	Hexadecimal code I=0	Hexadecimal code I=1	Description
AND	0XXX	8XXX	AND memory word to AC
ADD	1XXX	9XXX	ADD memory word to AC
LDA	2XXX	AXXX	LOAD memory word to AC
STA	3XXX	BXXX	Store content of AC in memory
BUN	4XXX	CXXX	Branch unconditionally
BSA	5XXX	DXXX	Branch and Save return address
ISZ	6XXX	EXXX	Increment and Skip if Zero.

CLA	7800	Clear AC
CLE	7400	Clear E
CMA	7200	Complement AC
CME	7100	Complement E
CIR	7080	Circulate right AC and E
CIL	7040	Circulate left AC and E
INC	7020	Increment AC
SPA	7010	Skip next instruction if AC positive
SNA	7008	Skip next instruction if AC negative
SZE	7002	Skip next instruction if E_0
HLT	7001	Halt computer.

Symbol	Hexadecimal code	Description
INP	F800	Input character to AC
OUT	F400	Output character from AC
SKI	F200	Skip on input flag.
SKO	F100	Skip on output flag
ION	F080	Interrupt On
IOF	F040	Interrupt off

→ A memory reference instruction has an address part of 12 bits. The address part is denoted by three x's and stand for the three hexadecimal digits being equivalent to 12 bits address.

→ The last bit of the instruction is designated by the symbol I.

→ When I=0, the last four bits of an instruction have a hexadecimal digit equivalent from 0 to 6 since the last bit is 0.

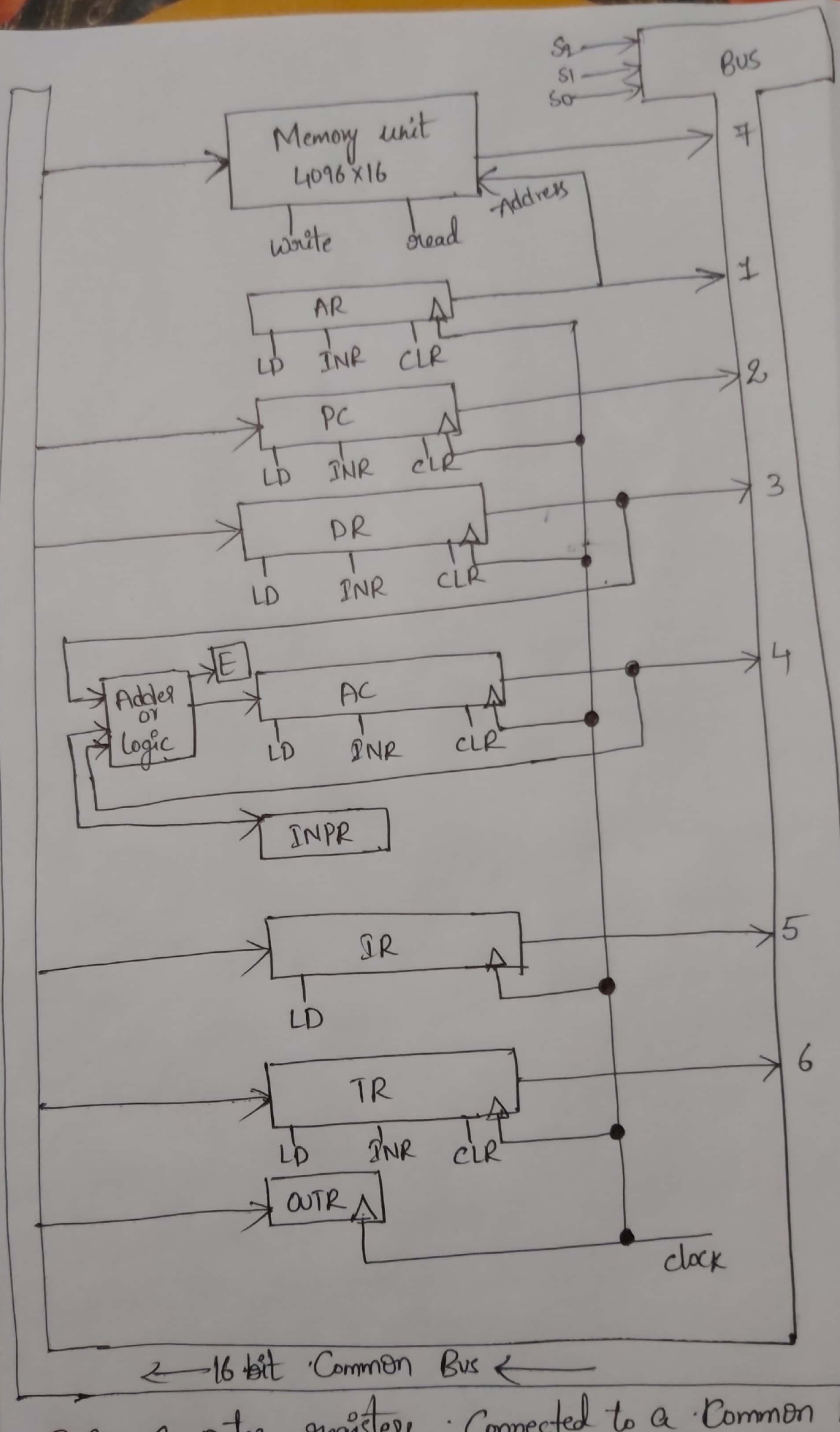
→ When I=1, the hexadecimal digit equivalent of the last four bits of the instruction ranges from 8 to E since the last bit is 1.

→ Register reference instructions use 16 bits to specify an operation. The leftmost four bits are always 0111, which is equivalent to hexadecimal 7. The other three hexadecimal digits give the binary equivalent of the remaining 12 bits.

→ The input-output instructions also use all 16 bits to specify an operation. The last four bits are always 1111, equivalent to hexadecimal F.

Common bus System (Computer registers).

The basic computer has eight registers, a memory unit, and a control unit. Paths must be provided to transfer information from one register to another and between memory and registers. The number of wires will be excessive if connections are made between the outputs of each register and the inputs of the other registers. A more efficient scheme for transferring information in a system with many registers is to use a common bus.



Basic Computer registers Connected to a Common bus

Connections:- The output of all the registers except the OUTR (Output register) are connected to the common bus. The output selected depends upon the binary value of variables S_2, S_1 and S_0. The lines from common bus are connected to the inputs of the registers and memory.

→ A register receives the information from the bus when its LD (load) input is activated. while in case of memory the write input must be enabled to receive the information. The contents of memory are placed onto the bus when its Read input is activated.

Various registers:-

4 registers DR, AC, IR and TR have 16 bits and 2 registers AR and PC have 12 bits. The INPR and Output have 8 bits are each. The INPR receives characters from input device and delivers it to the AC while the output register receives character from AC and transfers it to the output device. 5 registers have 3 control inputs. LD (load), INR (increment) and CLR (clear). These type of registers are similar to a binary counter.

Adder and logic Circuit :-

The adder and logic circuit provides the 16. inputs of AC. This circuit has 3 sets of inputs. One set comes from the outputs of AC which implements register micro operations. The other set comes from DR (data register) which are used to perform arithmetic logic ~~unit~~ micro operations. The result of these operations is sent to AC while the end around carry is stored in E. as The third set is from INPR.

Instruction set completeness (Computer instructions)

A computer should have a set of instructions so that the user can construct machine language programs to evaluate any gution that is known to be computable.

→ The set of instructions are said to be complete if the computer includes a sufficient number of instructions in each of the following categories.

1. Arithmetic, logical and shift instructions.
2. Instructions for moving information to and from memory and processor registers.
3. Program control instructions together with instructions that check status conditions.
4. Input and output instructions.

→ Arithmetic, logic and shift instructions provide computational capabilities for processing the type of data the user may wish to employ.

→ A huge amount of binary information is stored in the memory unit, but all computations are done in processor registers. Therefore, one must possess the capability of moving information between these two units.

→ Program control instructions such as branch instructions are used change the sequence in which the program is executed.

→ Input and output instructions act as an interface between the computer and the user. Programs and data must be transferred into memory, and the results of computations must be transferred back to the user.

Timing & Control :-

Timing and control units are present in CPU. Control unit mainly useful to generate the timing and control signals. Control unit coordinates between CPU, main memory, Input and Output devices. the control unit is classified into two major categories

1. Hardwired control
2. Microprogrammed control.

Hardwired control :-

The hardwired control organization, the control logic is implemented with gates, flip-flops, decoders, and other digital circuits

→ The following image shows the block diagram of a Hardwired control organizations.

→ A Hard-wired control consists of two decoders, a sequence counter, and a number of logic gates.

→ An instruction fetched from the memory unit is placed in the instruction register (IR)

→ The component of an instruction register includes; I bit, the operation code, and bits 0 through 11.

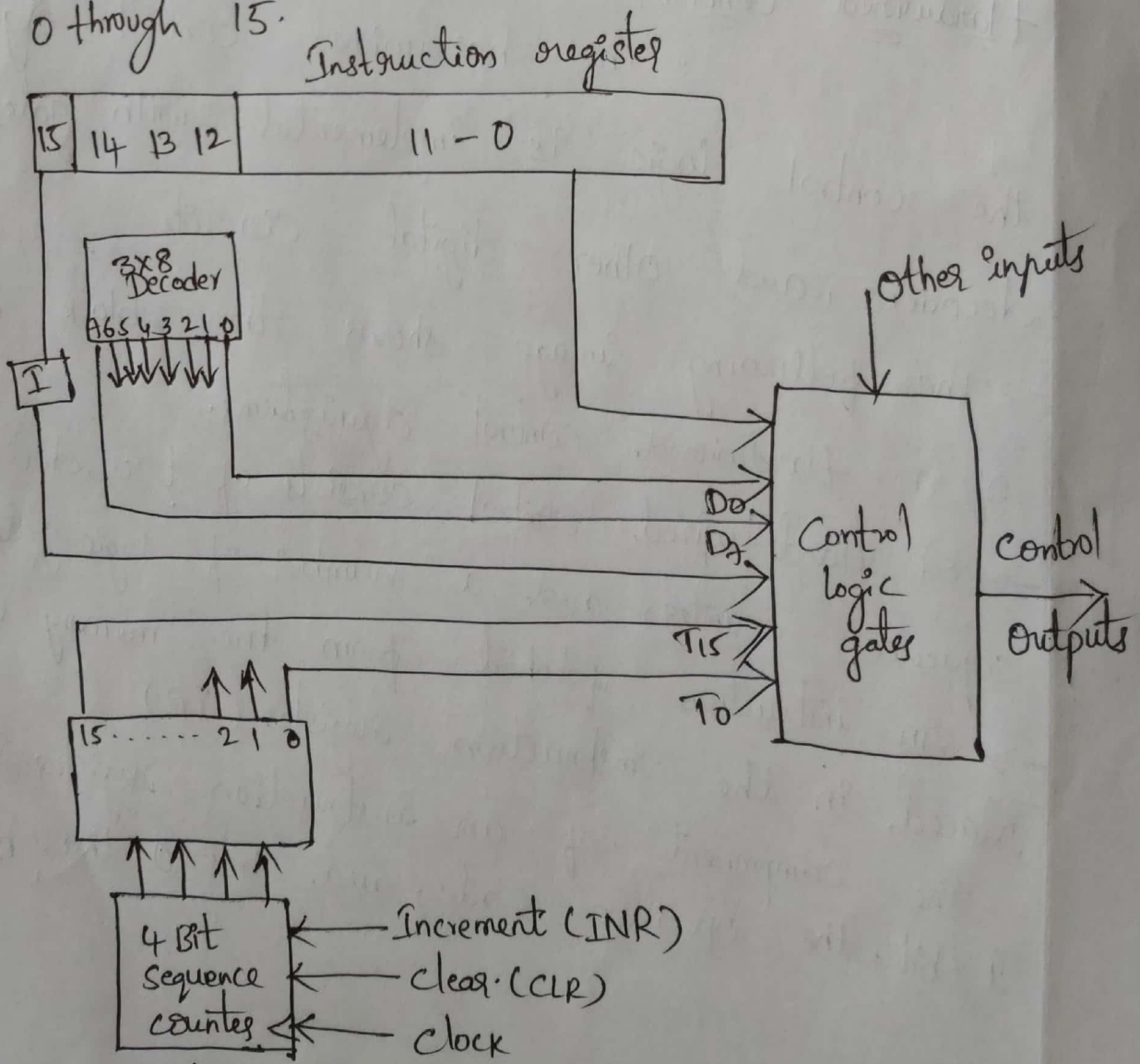
→ The operation code in bits 12 through 14 are coded with a 3x8 decoder.

→ The outputs of the decoder are designated by the symbols D0 through D7.

→ The operation code at bit 15 is transferred to a flip-flop designated by the symbol I.

→ The operation codes from bits 0 through 11 are applied to the control logic gates.

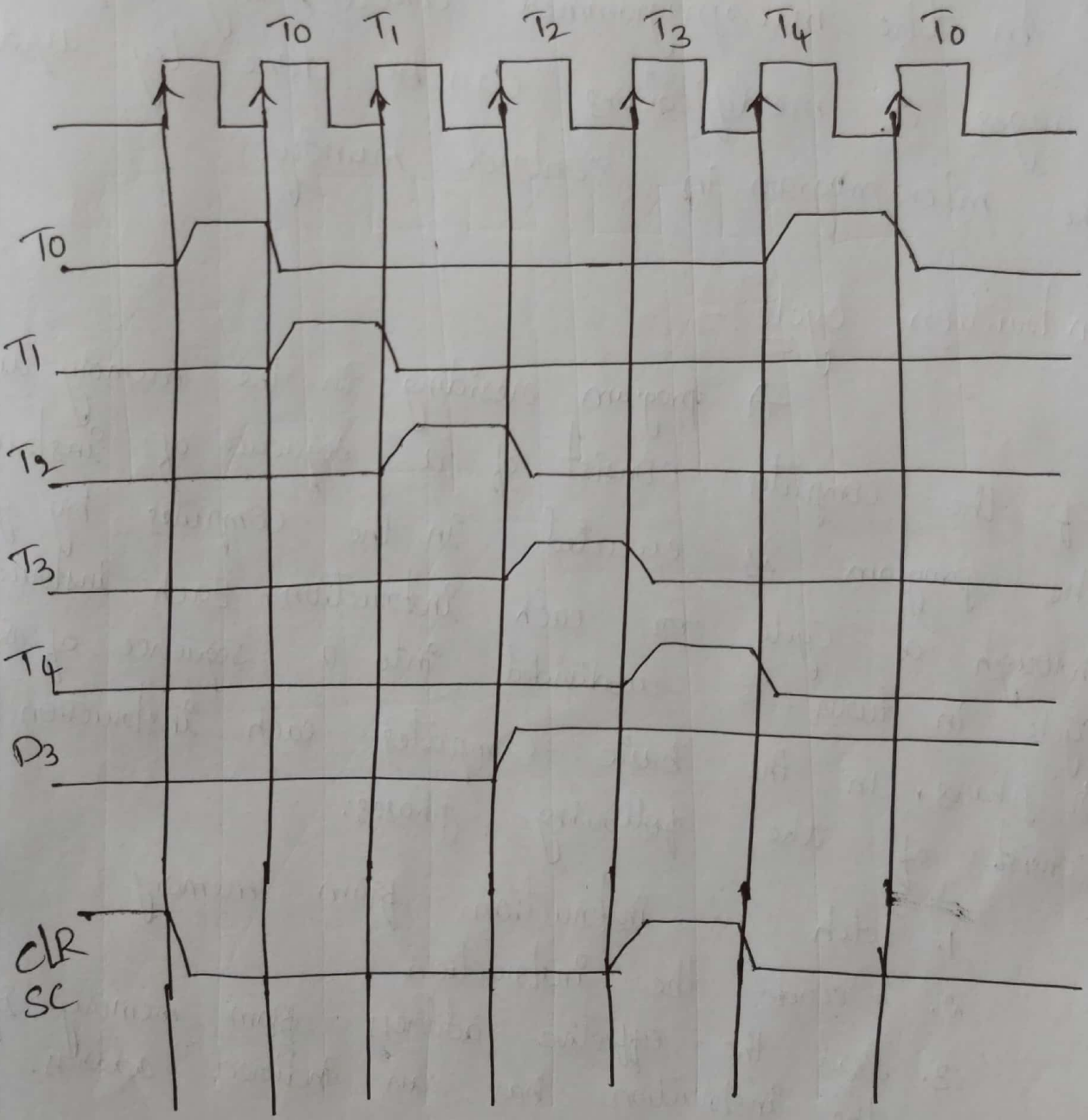
→ The sequence counter (SC) can count in binary from 0 through 15.



Control unit of a basic computer

As an example, consider the case where SC is incremented to provide timing signals T_0, T_1, T_2, T_3 & T_4 in sequence. At time T_4, SC is cleared to 0 if decoder output D_3 is active. This expressed symbolically by the statement.

$$D_3T_4: SC \leftarrow 0$$



example of control timing signal.

2. Microprogrammed control Organization:-

In the microprogrammed Organization, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations

→ In the microprogrammed control, any required changes & modifications can be done by updating the microprogram in control memory.

Instruction Cycle:-

A program residing in the memory unit of the computer consist of a sequence of instructions. The program is executed in the computer by going through a cycle for each instruction. Each instruction cycle in turn is subdivided into a sequence of subcycles or phases. 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

The completion of step 4, the control goes back to step 1 to fetch decode, and execute the next instruction. This process continous indefinitely unless a 'HALT' instruction is encountered.

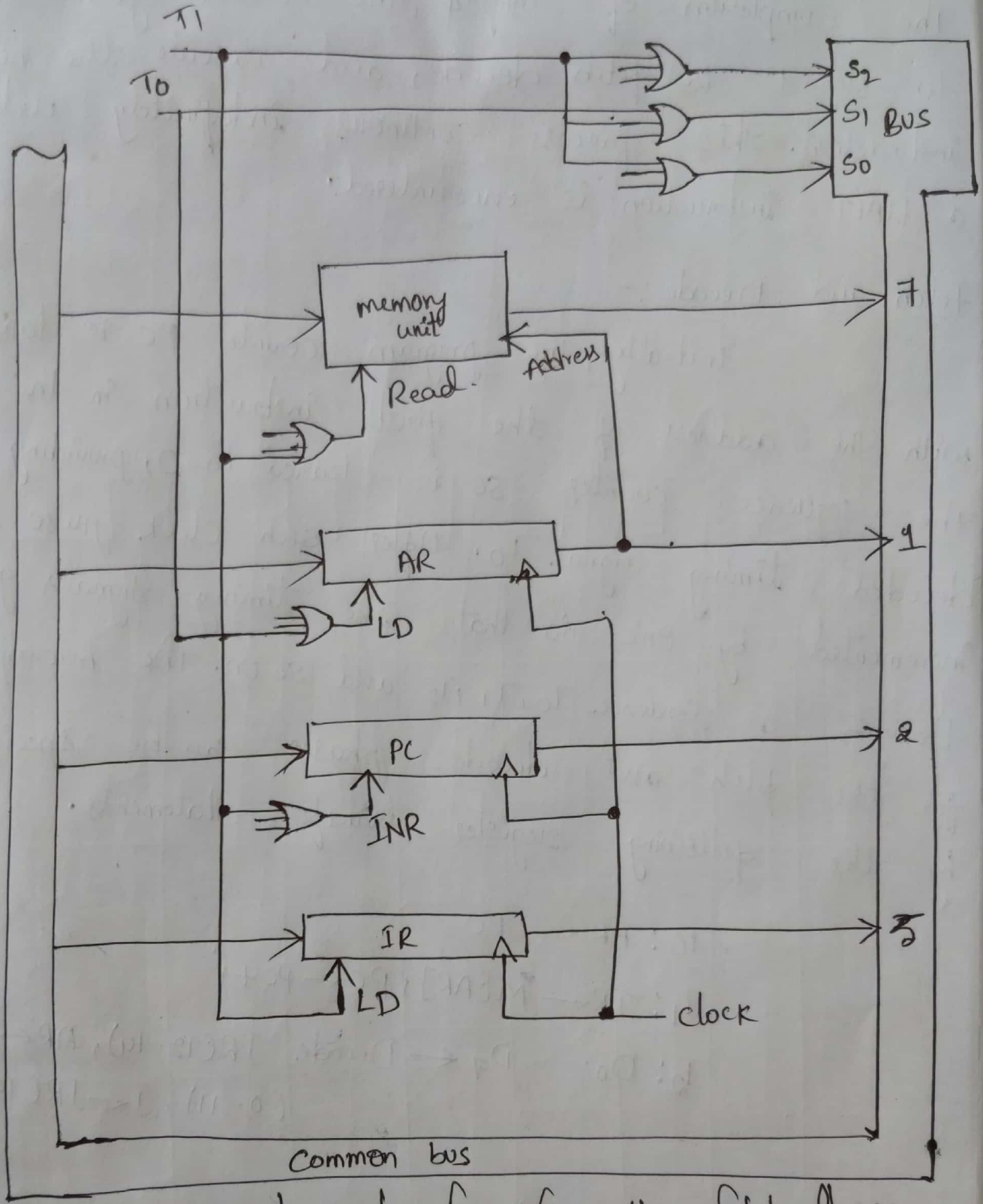
Fetch and Decode :-

Initially, the program counter PC is loaded with the address of the first instruction in the program. The 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 the timing signals 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 the following register transfer statements.

T_0 : AR ← PC

T_1 : IR ← M[AR], PC ← PC + 1

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



Register transfers for the fetch phase

The first two register statements are implemented in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal T0.

1. Place the content of PC onto bus by making the bus selection inputs $S_2S_1S_0$ equal to 010.
2. Transfer the content of bus to AR by enabling the LD input of AR.

The next clock transition initiates the transfer from PC to AR since $T_0 = 1$. In order to implement the second statement

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

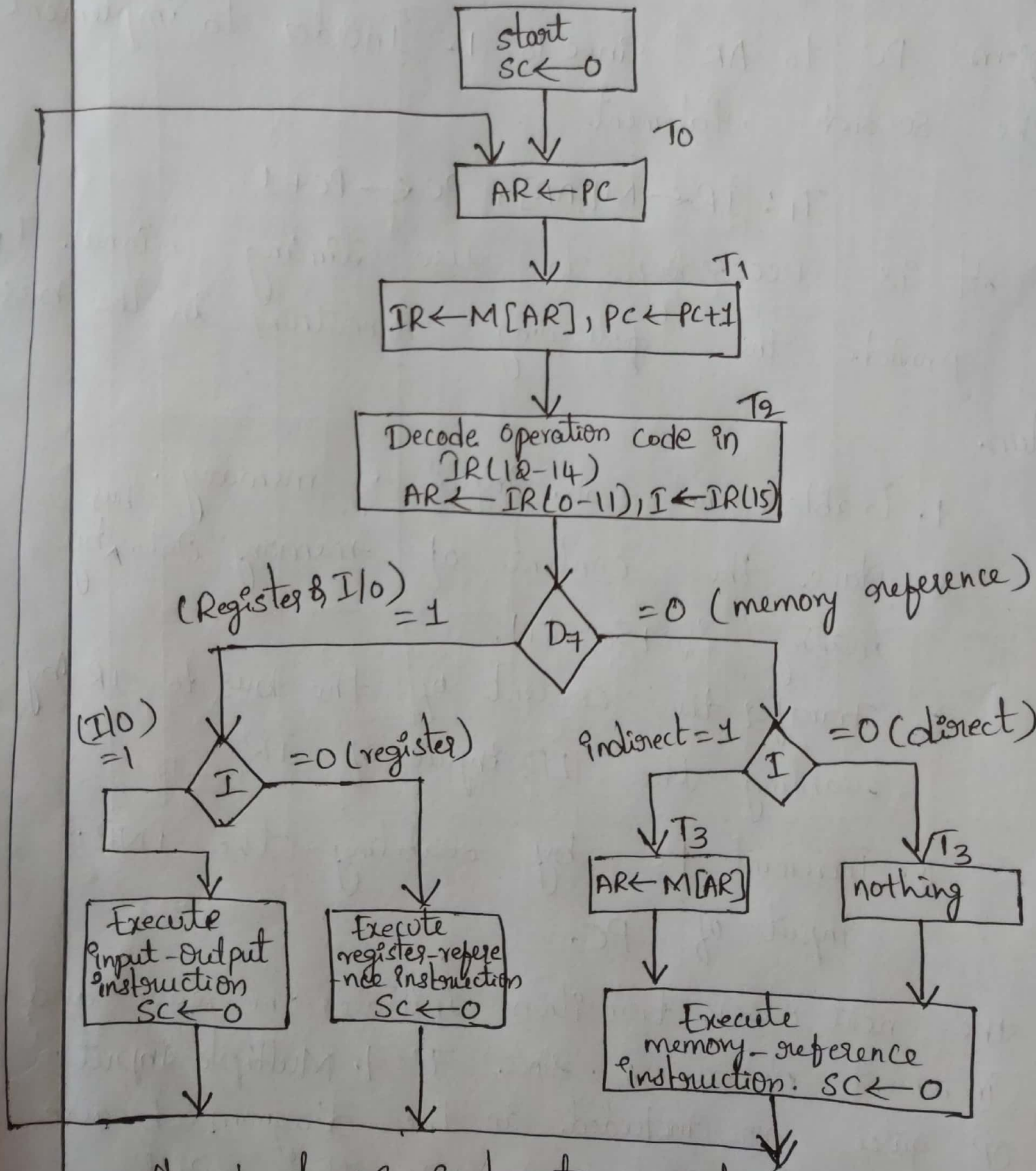
it is necessary to use timing signal T_1 to provide the following connections in the bus system.

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

The next clock transition initiates the read and increment operations. Since $T_1 = 1$. Multiple input OR gates are included in the diagram because there are other control functions that will initiate similar operations.

Determine the type of instruction :-

The timing signal that is active after the decoding is T_3. During the time T_3, the control units determine the type of instruction that was just read from memory.



flowchart for instruction cycle

Decoder output D_7 is equal to 1 if the operation code is equal to binary 111. we determine that if $D_7 = 1$, the instruction must be a register-reference & input-output type. If $D_7 = 0$, the operation code must be one of other seven values 000 through 110, specifying a memory-reference instruction

→ If $D_7 = 0$ and $I = 1$, we have a memory-reference instruction with an indirect address. It is then necessary to read the effective address from memory.

→ The microoperation for the indirect address condition can be symbolized by the register transfer statement.

$$AR \leftarrow M[AR]$$

→ The three instruction types are subdivided into four seperate paths. The selected operation is activated with the clock transition associated with timing signal T_3. This can be symbolized as follows.

$D_7' I T_3$: $AR \leftarrow M[AR]$

$D_7' I' T_3$: Nothing

$D_7 I' T_3$: Execute a register-reference instruction

Register reference instructions :-

Register reference instructions are recognized by the control when $D_7 = 1$ and $I = 0$. These instructions use bits 0 through 11 of the instruction code to specify 12 instructions. These 12 bits are available in IR(0-11). They were also transferred to AR during time T_2.

Execution of Register-reference instructions

$D_7 I' T_3 = r$ (Common to all register-reference instructions)
$IR(i) = B_i$ (bit in (0-11) that specifies the operation).

	$r: SC \leftarrow 0$	clear SC
CLA	$rB_{11}: AC \leftarrow 0$	Clear AC
CLE	$rB_{10}: E \leftarrow 0$	Clear E
CMA	$rB_9: AC \leftarrow \overline{AC}$	Complement AC
CME	$rB_8: E \leftarrow \overline{E}$	Complement E
CIR	$rB_7: AC \leftarrow shr\ AC,$ $AC(15) \leftarrow E,$ $E \leftarrow AC(0)$	Circulate right
CIL	$rB_6: AC \leftarrow shl\ AC, AC(0) \leftarrow E,$ $E \leftarrow AC(15)$	circulate left
INC	$rB_5: AC \leftarrow AC + 1$	Increment AC
SPA	$rB_4:$ if $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$	Skip if positive
SNA	$rB_3:$ if $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$	Skip if negative

SZA	rB_2: if (AC=0) then (PC ← PC+1)	Skip if AC zero
SZE	rB_1: if (E=0) then (PC ← PC+1)	Skip if E zero
HLT	rB_0: S ← 0 (S is a start-stop flip-flop)	Halt computer.

There are a total 12 register reference instruction in computer architecture that are in use and they are listed in table.

→ Each instruction has its own functionality and all these register reference instructions operate on accumulator only.

Functionality of all register reference instructions :-

CLA :- It clears the accumulator and in result it puts a zero in the accumulator.

CLE : Clears the extended accumulator (E) and to place a zero in it.

CMA: It is used to complement the value in the accumulator (AC)

CME : Complements the extended accumulator (E)

CIR : It is used to circulate the accumulator rightwards.

CIL: Circulates the accumulator leftwards.

INC: It is used to increment the value in the accumulator (AC)

SPA: It states that skip to any other location that is specified if the value in the accumulator is positive.

SNA: Skips to some other location if the accumulator has a negative value.

SZA: is used to skip to some other location or instruction if the result stored is zero in the accumulator (A)

SZE:- is used to skip to some other memory location or instruction, if the result acquired is zero in the extended accumulator (A)

HLT :- Halts the execution of the program.

=

Memory reference instructions :-

The seven memory reference instructions are decoded output D_i for $i = 0,1,2,3,4,5$ and 6 from the operation decoder that belongs to each instruction is in the address register AR and was placed there during timing signal T_2 when $I = 0$, & during timing signal T_3 when $I = 1$. The execution of the memory-reference instructions starts with timing signal T_4.

Memory reference instructions

Symbol	operation decoder	Symbolic description
AND	D_0	$AC \leftarrow AC \wedge M[AR]$
ADD	D_1	$AC \leftarrow AC + M[AR]$, $E \leftarrow Cout$
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	If $M[AR] + 1 = 0$ then $PC \leftarrow PC + 1$

AND to AC :-

→ Performs the AND logic operations on pairs of bits in AC and the memory word specified by the effective address.

→ Two timing signals are needed.

- In T_1 transferring Operand from memory into DR
- In T_5 transferring result of AND logic operation between the contents of DR and AC.
- In T_3 SC is cleared to 0 and control is transferred to T_0 to start a new instruction cycle.

Example :-

D_0T_4: $DR \leftarrow M[AR]$

D_0T_5: $AC \leftarrow AC \wedge DR$, $SC \leftarrow 0$

ADD to AC :-

→ Adds the contents of memory word specified by the effective address to the value of AC

→ Sum is transferred into AC and the output carry Cout is transferred to the E (extended accumulator) flipflop.

D_1T_4: DR ← M[AR]

D_1T_5: AC ← AC+DR, E ← Cout, SC ← 0

LDA: Load to AC :-

→ Transfers the memory word specified by the effective address to AC.

→ Necessary to read the memory word into DR first and transfer the contents of DR into AC

→ there is no direct path from bus into AC

→ to maintain one clock cycle as well.

D_2T_4: DR ← M[AR]

D_2T_5: AC ← DR, SC ← 0

STA: Store AC :-

→ Stores the content of AC into the memory word specified by the effective address.

→ the output of AC is applied to the bus and the data input of memory is connected to the bus.

D_3T_4: M[AR] ← AC, SC ← 0

BUN :- Branch unconditionally :-

→ PC is incremented at time T_1 to prepare it for the address of the next instruction in the program sequence.

→ BUN transfers the program to the instruction specified by the effective address.

→ Allows the programmer to specify an instruction out of sequence and we say that the program branches (jumps) unconditionally

D_4T_4 : PC ← AR SC ← 0 (resetting SC transfers control to T_4)

BSA : Branch and save return address :-

→ useful for branching to a portion of the program called a subroutine or procedure.

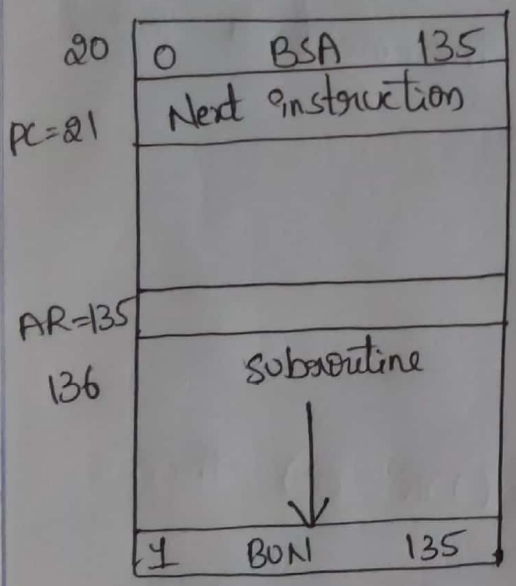
→ When executed, it stores the address of the next instruction in sequence (which is available in PC) into a memory location specified by the effective address.

→ (Effective address + 1) is then transferred to PC to save as the address of the first instruction in the subroutine.

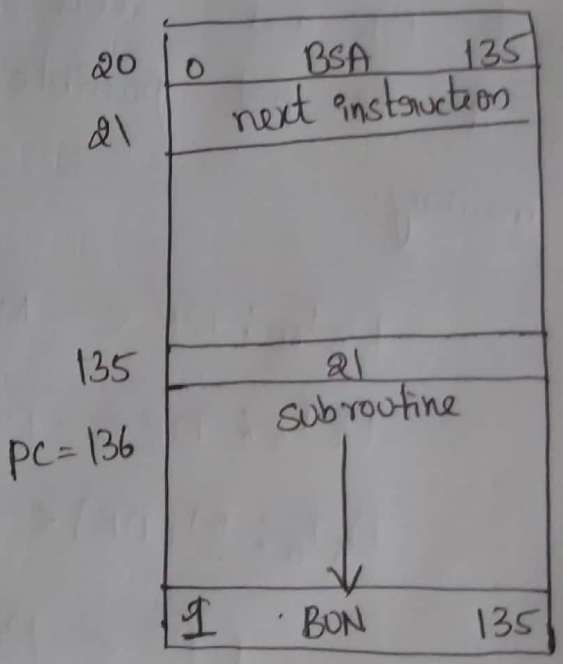
→ The return to the original program is accomplished by the BUN instruction placed at the end of the subroutine.

D_5T_4: M[AR] ← PC, AR ← AR+1

D_5T_5: PC ← AR, SC ← 0



(a) Memory, PC, and AR at time T_4.



(b) Memory and PC after execution

Example of BSA instruction execution

ISZ: Increment and Skip if Zero:-

→ Increments the word specified by effective address.

→ If the incremented value is equal to 0, PC is incremented by 1.

→ When a negative number (in 2's complement) stored in memory word is repeated incremented by 1 it eventually reaches zero.

→ It is necessary to read the word into DR, increment DR and store the word back into memory since it is not possible to increment a word inside the memory.

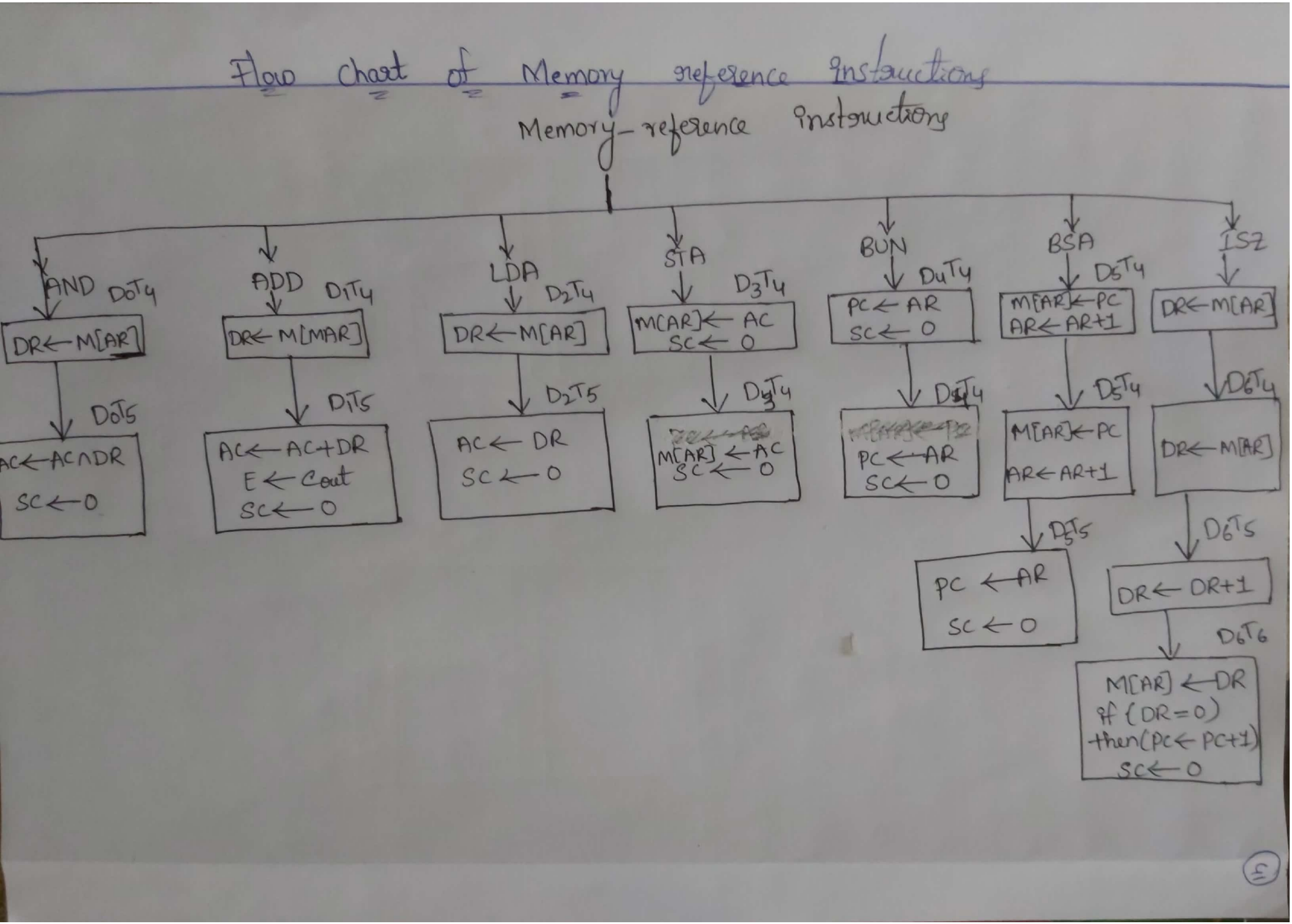
D_6T_4 : DR ← M[AR]

D_6T_5 : DR ← DR+1

D_6T_6 : M[AR] ← DR, if (DR = 0) then

(PC ← PC+1), SC ← 0

Flow chart of Memory reference instructions



Program Interrupt :-

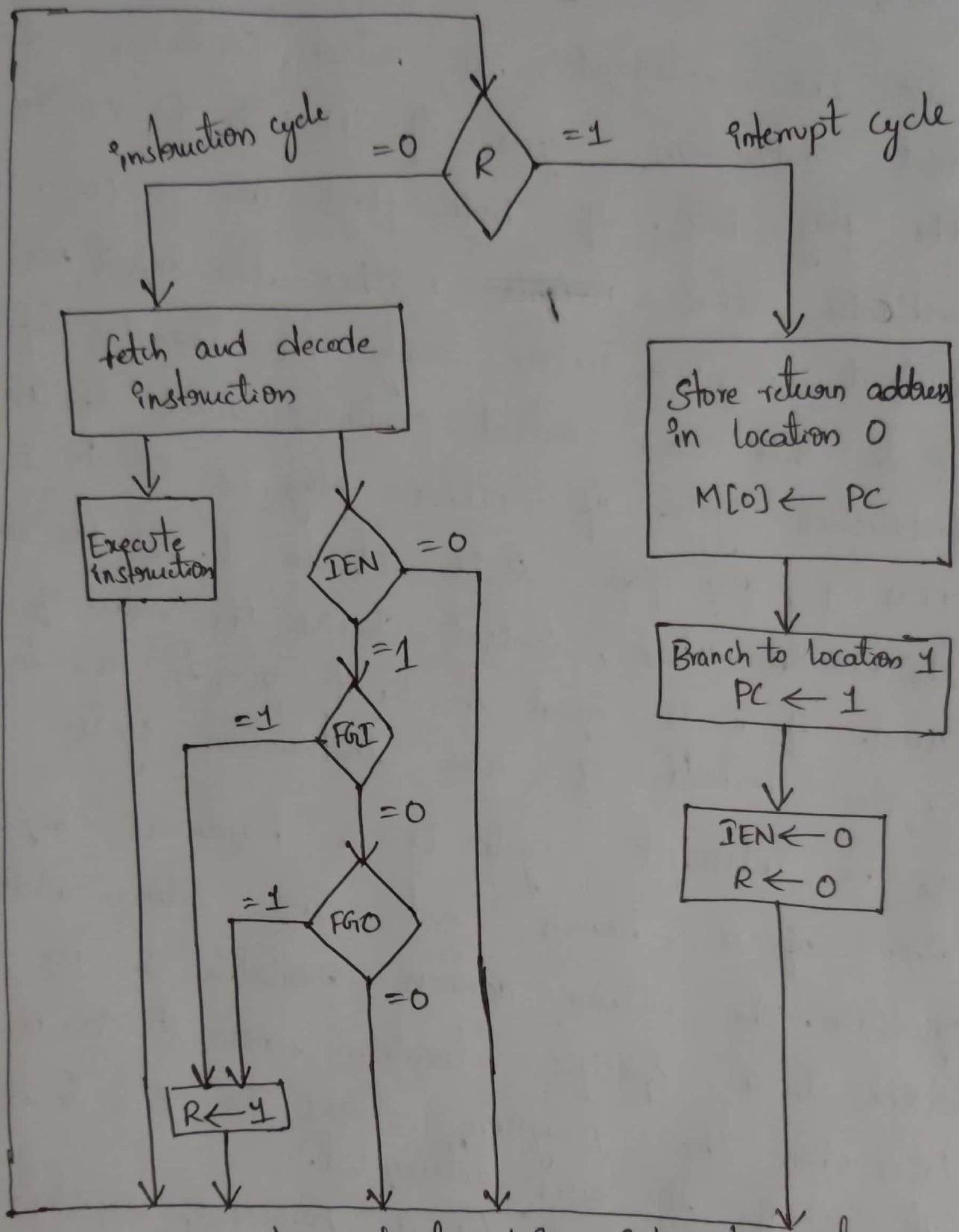
An interrupt is a special signal which needs to be executed immediately whenever the cpu receives interrupt signal, then CPU stops the execution of current running program and CPU control will be shifted to the interrupt related program.

→ Once cpu has completed the execution of interrupt related program. CPU control will shifted to the original program. Now CPU will execute the remaining instructions of the currently running program.

→ The interrupt enable flipflop. IEN can be set and cleared with two instructions. When IEN is cleared to 0 (with the IOF instruction), the flags cannot interrupt the computer. When IEN is set to 1 (with ION instruction), the computer can be interrupted

Flow chart for interrupt cycle :-

An interrupt flipflop R is included in the computer. When R=0, the computer goes through an instruction cycle. Where as R=1 the computer goes through an interrupt cycle.



Flow chart for interrupt cycle.

During the execute phase of instruction cycle IEN is checked by the control. If it is 0, it

indicates that the programmer does not want to use the interrupt, so control continues with the next instruction cycle. If IEN is 1 control checks the flag bits. If both flags are zero (0), it indicates that neither the input nor the output registers are ready for transfer information.

→ In this case, control continues with the next instruction cycle. If either flag is set to 1 while IEN = 1, flip-flop R is set to 1. At the end of the execute phase, control checks the value of R, and if it is equal to 1, it goes to an interrupt cycle instead of an instruction cycle.

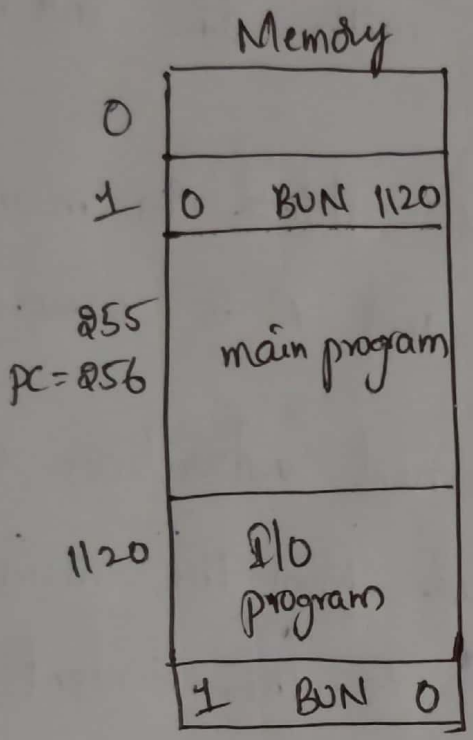
→ The interrupt cycle is a hardware implementation of a branch and save return address operation. The return address available in PC is stored in a specific location where it can be found later when the program returns to the instruction at which it was interrupted.

→ This location may be a processor register, a memory stack, or a specific memory location.

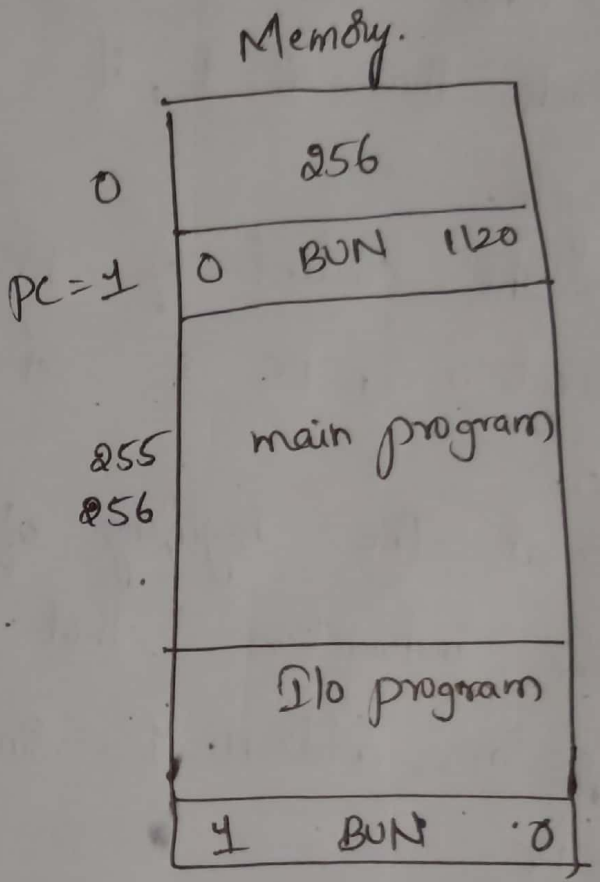
→ Here we choose the memory location at address 0 as the place for storing the return address,

control then inserts 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.

Demonstration of the Interrupt Cycle :-



(a) Before interrupt



(b) After interrupt cycle.

→ The above example that shows that what happens during the interrupt cycle is shown.

→ Suppose that an interrupt occurs and R is set to 1 while the control is executed the instruction at address 255.

→ At this time, the return address 256 is in PC. The programmer has previously placed an input-output service program in memory starting from address 1120 and a BUN 1120 instruction at address 1.

→ When control reaches timing signal T0 and finds that R = 1, it proceeds with the interrupt cycle.

→ The content of PC (256) is stored in memory location 0, PC is set to 1, and R is cleared to 0.

→ At the beginning of the next instruction cycle, the instruction that is read from the memory is, in address 1. Since this is the content of PC.

→ The branch instruction at addrss 1 causes the program to transfer to the input-output service program at addrss 1120.

→ This program checks the flags, determine which flag is set, and then transfers the required input & output information.

Once this is done, the instruction ION is executed to set IEN to 1, and the program returns to the location where it was interrupted.

→ Finally the instruction returns the computes to the original place in the main program is a branch indirect instructions with an address part of 0. this instruction is placed at the end of the I/O service program.

→ after this instruction is read from memory during the fetch phase, control goes to the indirect phase (because I = 1) to read the effective address.

→ The effective addres is in location 0 and is return address that was stored there during the previous interrupt cycle.

→ The execution of the indirect BUN instruction results in placing into PC the return address from location 0.

the interrupt cycle initiated after the last execute phase if the interrupt flip-flop R is equal to 1. This flip-flop is set to 1 if IEN=1 and either FGI or FGO are equal to 1. This can happen with any clock transition except when timing signals T_0, T_1 or T_2 are active.

→ The condition for the setting flip-flop R to 1 can be expressed with the following register transfer statement.

$$T_0 T_1' T_2' (IEN)(FGI + FGO) : R \leftarrow 1$$

The Symbol + between FGI and FGO in the control function designates a logic OR operation. This is ANDed with IEN and $T_0' T_1' T_2'$

→ This can be done with the following sequence of microoperations.

RT_0: AR ← 0, TR ← PC

RT_1: M[AR] ← TR, PC ← 0

RT_2: PC ← PC + 1, IEN ← 0, R ← 0, SC ← 0

During the first timing signal AR is cleared to 0, and the content of PC is transferred to the

temporary register TR. with the second timing signal, the return address is stored in memory at location 0 and PC is cleared to 0. The third timing signal increments PC to 1, clears IEN and R, and control goes back to T_0 by clearing SC to 0. The beginning of the next instruction cycle has the condition $R'T_0$ and the content of PC is equal to 1. The control then goes through an instruction cycle that fetches and executes the BUN instruction in location 1.

* UNIT - 1 Completed *