WINNER
Page No.:
Date: / /

Unit 1:
Data Representation

1. Find the 10's complement of $(0.3267)_{10}$

→ 10's complement of 0.3267 is

$= 10^0 - 0.3267$

$= 1 - 0.3267$

$= 0.6733$

*. $(25.639)_{10}$

→ 10's complement of $(25.639)_{10}$ is

$= 10^2 - 25.639$

$= 100 - 25.639$

$= 74.361$

2. Find the 2's complement of $(101100)_2$

→ 2's complement of $(101100)_2$ is

010011

$+1$

010100

*. $(0.0110)_2$.

$N = 0.0110$

$n = 0$

$r = 2$

$r^n - N$

$= 2^0 - 0.0110$

$= 1 - 0.0110$

$= 0.1010$

1.0000

$-\ 0110$

0.1010

3. Subtract 1010100 from 1000100 using 2's complement.

→ 2's complement of 1010100 is 0101011
+1
0101100

Now,

1000100
+ 0101100
1110000

Here, no carry is generated so result is negative.
Again, changing (1110000) = 0001111
+1
0010000

Result = -(0010000)

4. Subtract 72532-3250 using 9's complement.

→ M = 72532
N = 03250

9's complement of 03250 = 10^5-1-03250
= 96749

Now,

72532
+ 96749
169281

discard carry (1)

∴ 72532 - 03250 = 69282

5. Subtract 3250-72532 using 9's complement.

M = 03250

N = 72532

9's complement of 72532 = $10^5 - 1 - 72532$

= 27467

Now,

$$\begin{array}{r} 03250 \\ +27467 \\ \hline 30717 \end{array}$$

Again,

9's complement of 30717 = $10^5 - 1 - 30717$

= 69282

(3250 - 72532) = -69282

6. Subtract 1010100 - 1000100 using 1's complement.

M = 1010100

N = 1000100

1s complement of 1000100 = 0111011

Now,

$$\begin{array}{r} 1010100 \\ 0111011 \\ \hline 10001111 \end{array}$$

carry ← 10001111

Again,

$$\begin{array}{r} 0001111 \\ +1 \\ \hline 0\ 010000 \end{array}$$

∴ 1010100 - 1000100 = 0010000₂

7. Subtract 1000100 - 1010100 using 1's complement.

M = 1000100

N = 1010100

1's complement of 1010100 = 0101011

Now,

$$\begin{array}{r} 1000100 \\ +\,0101011 \\ \hline 1101111 \end{array}$$

Here, no carry is generated so result is negative.

Again,

Changing 1101111 into 1's complement

1101111 = 0010000

∴ 1000100 - 1010100 = -(0010000)

Fixed Point Representation:

Positive integers, including zero, can be represented as unsigned numbers. However, to represent negative integers, we need a notation for negative values. In ordinary arithmetic, a negative number is indicated by a minus sign and a positive number by a plus sign. Because of hardware limitations, computers must represent everything with 1's and 0's, including the sign of a number. As a consequence, we use convention of representing left most bit of a number as a sign bit; 0 for positive and 1 for negative.

In addition to the sign, a number may have a binary (or decimal) point. The position of binary point is needed to represent fractions, integers or mixed integer-fraction numbers. There are two ways of specifying the position of the binary point in a register:

1) by employing the floating point representation
2) by giving binary point to a fixed position.

The fixed-point method assumes that the binary point is always fixed in one position. The two positions most widely used are:

a) A binary point in the extreme left of the register to make the stored number a fraction.
b) A binary point in the extreme right of the register to make the stored number an integer.

Integer Representation:

When an integer binary number is positive, the sign is represented by 0 and the magnitude by a positive binary number. When the number is negative, the sign is represented by 1 but the rest of the number may be represented in one of three possible ways:

1. Signed-magnitude representation
2. Signed-1's complement representation
3. Signed-2's complement representation

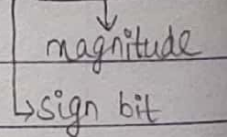
The signed magnitude representation of a negative number consists of the magnitude and a negative sign. In other two representations, the negative number is represented in either 1's or 2's complement of its positive value.

Example:

$+14 = 00001110$

But, for -14, there are three different ways:

- In signed-magnitude representation: 10001110
- In signed-1's complement representation: 11110001
- In signed-2's complement representation: 11110010



Arithmetic Addition :

The addition of two numbers in the signed-magnitude system follows the rules of ordinary arithmetic. If the signs are the same, we add the two magnitudes and give the sum the common sign. If the signs are different, we subtract the smaller magnitude from the larger and give the result the sign of the larger magnitude.

Add the two numbers, including their sign bits, and discard any carry out of the sign (leftmost) bit position. Note that negative numbers must initially be in 2's complement and that if the sum obtained after the addition is negative, it is in 2's complement form.

Examples:

1)

+6	0000 0110
+13	0000 1101
+19	0001 0011

2)

-6	1111 1010 (2's complement of 0000 0110)
+13	0000 1101
+7	0000 0111

3)

+6	0000 0110
-13	1111 0011 (2's complement of 0000 1101)
-7	1111 1001

leftmost sign bit

4) -6 → 1111 1010
-13 → 1111 0011
-19 → 1110 1101

Arithmetic subtraction:

Subtraction of two signed binary numbers is done as: - take the 2's complement of the subtrahend (including the sign bit) and add it to the minuend (including the sign-bit).

The carry out of the sign bit position is discarded.

Subtraction operation can be changed to the addition operation if the sign of the subtrahend is changed.

$(\pm A) - (+B) = (\pm A) + (-B)$
$(\pm A) - (-B) = (\pm A) + (+B)$

Example: $(-6) - (-13) = +7$, in binary with 8-bits this is written as:

$-6 \rightarrow 11111010$
$-13 \rightarrow 11110011$ } 2's complement form

Subtraction is changed to addition by taking 2's complement of the subtrahend (-13) to give $(+13)$

$-6 \rightarrow 11111010$
$+13 \rightarrow 00001101$
$+7 \rightarrow 100000111$

Discarding the end carry, we obtain the correct answer 0000 0111 (+7)

Overflow :-

When two numbers of n-digits each are added and the sum occupies n+1 digits, we say that an overflow occurred. An overflow is a problem in digital computers because the width of the registers is finite. A result that contains n+1 bits cannot be accomodated in a register with a standard length of n bits. For this reason, many computers detect the occurrence of an overflow, and when it occurs, a corresponding flip-flop is set which can then be checked by the user.

An overflow cannot occur after an addition if one number is positive and the other is negative. An overflow may occur if the two numbers added are both positive or both negative. For example: two signed binary numbers +70 and +80 are stored in two 8-bit registers. Since the sum of numbers is 150 exceeds the capacity of the register (since 8-bit register can store values ranging from -128 to +127), hence the overflow occured.

carries: 0 1

+70	0	1000110
+80	0	1010000
+150	1	0010110

carries: 1 0

-70	1	0111010
-80	1	0110000
-150	0	1101010

Note that: The 8-bit result that should have been positive has a negative sign bit and the 8-bit result that should have been negative has a positive sign bit. If, however, the carry out of the sign bit position is taken as the sign bit of the result, the 9-bit answer obtained will be correct. Since, the answer cannot be accommodated within 8-bits, we say that an overflow occurred.

Overflow detection :-

An overflow condition can be detected by observing the carry into the sign bit position and carry out of the sign bit position. If these two carries are not equal, an overflow condition is produced. If the two carries are applied to an exclusive-OR gate, an overflow will be detected when the output of the gate is equal to 1.

Decimal fixed-point representation:

The representation of decimal numbers in a registers is a function of the binary code used to represent a decimal digit. A 4-bit decimal code requires four flip-flops for each decimal digit. The representation of 4385 in BCD requires 16-flip-flops, four flip-flop for each digit. The number will be represented in a register with 16-flip-flop as follows:

$(4385) = (0100\ 0011\ 1000\ 0101)_{BCD}$

While using BCD representation:

Disadvantages:

→ Wastage of memory (viz. binary representation of 4385 uses less bits than its BCD representation)

→ circuits for decimal arithmetic are quite complex.

Advantages:

→ Eliminate the need for conversion to binary and back to decimal (since applications like business data processing requires less computation than I/O of decimal data, hence electronic calculators perform arithmetic operations directly with the decimal data (in binary code)).

Representation of signed decimal numbers in BCD:

It is customary (चलनअनुसारकों) to designate a plus with four 0's and a minus with 1001 (BCD equivalent of 9).

Addition is done by adding all digits, including the sign digits, and discarding the end carry. Obviously, this assumes that all negative numbers are in 10's complement form.

Consider the addition: (+375) + (-240) = +135

$$\begin{array}{rl} & 0\,375\ (0000\ 0011\ 0111\ 0101)_{BCD} \\ + & 9\,760\ (1001\ 0111\ 0110\ 0000)_{BCD} \\ \hline & 0\,135\ (0000\ 0001\ 0011\ 0101)_{BCD} \end{array}$$

equivalent BCD of 0135

10's complement of 240.

Floating-point Representation:

The floating-point representation of a number has two parts:

1) Mantissa: represents a signed, fixed point number. May be a fraction or an integer.

2) Exponent: It designates the position of the decimal (or binary) point.

So, the floating points are represented in computers as the format given below:

Sign	Mantissa	Exponent

For example:

The decimal number +6132.789 is represented in floating-point as:

Fraction	Exponent
+0.6132789	+04

The value of the exponent indicates that the actual position of the decimal point in the fraction. This representation is equivalent to the scientific notation $+0.6132789 \times 10^{+4}$.

Floating point is always interpreted to represent number in the following form:

$$m \times r^e$$

Only the mantissa (m) and the exponent (e) are physically represented in the register (including their signs). The radix (r) and the radix-point position of the mantissa are always assumed.

Example: The binary number +1001.11 is represented with an 8-bit fraction and 6-bit exponent as follows:

Fraction	Exponent
01001110	000100

The fraction has a 0 in the leftmost position to denote positive. The exponent has the equivalent binary number +4. The floating point number is equivalent to

$$m \times 2^e = +(.10011110)_2 \times 2^{+4}$$

Normalization:-

A floating point number is said to be normalized if the most significant digit of the mantissa is non-zero. For eg:- the decimal number 350 is normalized but 0035 is not.

* Two standard forms of floating-point numbers are from the following organizations that decide the standard:
  1) ANSI (American National Standards Institute)
  2) IEEE (Institute of Electrical and Electronic Engineers).

The ANSI 32-bit floating point numbers in byte format with examples are given below:

Byte Format:

Byte 1	Byte 2	Byte 3	Byte 4
SEEEE	. IMMMMMMM	MMMMMMMM	MMMMMMMM

Exponent — Binary point — Mantissa

where; S → sign of the Mantissa
E → Exponent bits in 2's complement
M → Mantissa bits.

Examples:

$13 = 1101 = 0.1101 \times 10^4$

$= 00000100 \quad 11010000 \quad 00000000 \quad 00000000$

$-17 = -10001 = -0.10001 \times 10^5$

$= 10000101 \quad 10001000 \quad 00000000 \quad 00000000$

*. Arithmetic operations with floating point numbers are more complicated than arithmetic operations with fixed-point numbers and their execution takes longer and requires more complex hardware.

Gray code:-

The reflected code or gray code is used to represent digital data converted ..analog data. Gray code changes by only one bit as it sequences from one number to the next.

Decimal	BCD	Gray code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

Table: 4-bit gray code

Note: For gray code; apply XOR -gate on BCD as follows:

BCD : 1010
↓ ↓ ↓ ↓
gray code: 1111

Weighted code (2421):-

2421 is an example of weighted code. In this, corresponding bits are multiplied by the weights indicated and the sum of the weighted bits gives the decimal digit.

Example: 1101 when weighted by the respective digits 2421 gives $2*1 + 4*1 + 2*0 + 1*1 = 7$

Excess-3 codes:-

The excess-3 code is a decimal code that has been used in older computers. This is an unweighted code. Excess-3 code = BCD binary equivalent +3 (0011)

Note: excess-n code is possible adding n to the corresponding BCD equivalent.

Excess-3 gray:-

In ordinary gray code, the transition from 9 back to 0 involves a change of three bits (from 1101 to 0000). To overcome this difficulty, we start from third entry 0010 (as first number) upto the twelfth entry 1010, there by change of only one bit is possible upon transition from 1010 to 0010. Since, code has been shifted up three numbers, it is called the excess-3 gray.

Decimal digit	BCD 8421	2421	Excess-3	Excess-3 gray
0	0000	0000	0011	0010
1	0001	0001	0100	0110
2	0010	0010	0101	0111
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1100
6	0110	1100	1001	1101
7	0111	1101	1010	1111
8	1000	1110	1011	1110
9	1001	1111	1100	1010
Unused bit combi-nations	1010	0101	0000	0000
	1011	0110	0001	0001
	1100	0111	0010	0011
	1101	1000	1101	1000
	1110	1001	1110	1001
	1111	1010	1111	1011

Table: 4 different binary codes for the decimal digit

Error Detection Codes:-

Binary information transmitted through some form of communication medium is subject to external noise that could change bits from 1 to 0 and vice-versa. An error-detection code is a binary code that detects digital errors during transmission. The detected errors cannot be corrected but their presence is indicated.

Parity bit:

An error-correction code is a binary code that detects and corrects single bit error occurred during transmission. The most common error detection code used is the parity bit. A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even. A message of three bits and two possible parity bits is shown in table below:

Message xyz	P(odd)	P(even)
000	1	0
001	0	1
010	0	1
011	1	0
100	0	1
101	1	0
110	1	0
111	0	1

Table :- Parity bit generation

Parity generator:

The circuit that generates the parity bit in the transmission is parity generator. At the sending end, the message (in this case three bits) is applied to a parity generator, where the required parity bit is generated. And the message, including parity bit, is transmitted to its destination.

Message (XYZ)	P (odd)
000	1
001	0
010	0
011	1
100	0
101	1
110	1
111	0

Fig: 3-bit parity generator.

From the table:

$$P = x'y'z' + x'yz + xy'z + xyz'$$
$$= z'(x'y' + xy) + (x'y + xy')z$$

$\because P = z'A' + zA$
$= z \odot A$
$= z \odot x \oplus y$

$[\because A = x'y + xy']$
$[\therefore A' = x'y' + xy]$

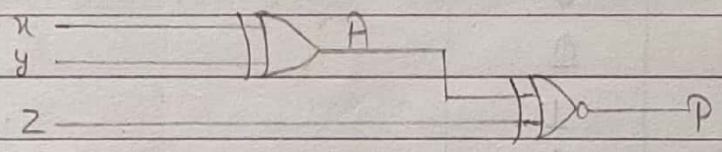


Fig:- Parity generator

Parity checker:

At the receiving end, the 3-bit message and the parity bit are applied to the parity checker circuit to check for possible errors in the transmission. Since the information was transmitted with odd parity, the four bits received must have an odd parity. An error occurs during the transmission if the four bits received have even number of 1's, indicating that one bit has changed in value during transmission.

Message (including parity bit) xyzP	Parity checker
0000	1
0001	0
0010	0
0011	1
0100	0
0101	1
0110	1
0111	0
1000	0
1001	1
1010	1
1011	0
1100	1
1101	0
1110	0
1111	1

From the table:

$$C = x'y'z'P' + x'y'zP + x'yz'P + x'yzP' + xy'z'P + xy'zP' + xyz'P' + xyzP$$

$$= (z'P' + zP)x'y' + (z'P + zP')x'y + (z'P + zP')xy' + (z'P' + zP)xy$$

$$= (z'P' + zP)(x'y' + xy) + (z'P + zP')(x'y + xy')$$

$$= A'B' + AB$$

$$= A \odot B$$

$$= (z'P + zP') \odot (x'y + xy')$$

$$\boxed{C = z \oplus P \odot x \oplus y}$$

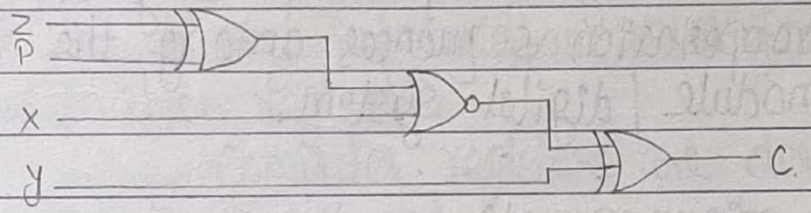


Fig :- Parity checker (4-bit)

Unit 2:
MICROOPERATIONS

one clock pulse mg execute hare operation

Microoperations:-

The operation executed on data stored in a register are called microoperation. A microoperation is an elementary operation performed on the information stored on one or more register. The result of operation may replace the previous binary information of register or may be transferred to another register. Example of microoperation are shift, clear, add, subtract, etc.

The register transfer language can be used to describe the microoperation. A register transfer language is a system for expressing in symbolic form, the microoperation sequences among the register of digital module / digital system.

Types of microoperation:

The microoperation most often encountered in digital computer are classified into four categories:

1. Register transfer microoperations:-
   These microoperation transfer binary information from one register to another.

2. Arithmetic microoperations:-
   These microoperation perform arithmetic

operation on data stored in register.

3. Logic microoperations :-
These microoperation performs bit manipulation operation on data stored in registers.

4. Shift microoperations :-
Shift microoperation perform shift operation on data stored in register.

1. Register transfer microoperations:-
These microoperation transfer binary information from one register to another. This type of microoperation doesnot change the information content when the binary information moves from source register to destination register.

Computer registers are designated by capital letters sometimes followed by numerals. Often name indicates the function. For example; the register that hold an address for the memory unit is usually called memory address register and designated by the name MAR.

Information transfer from one register to another is designated in symbolic form by the means of replacement operator. For example; the statement $R_2 \leftarrow R_1$ denotes the transfer of content of register in R_1 to register R_2.

Normally, we want the transfer occur only under the predefined control condition, this can be specified by control function. A control function is boolean variable that is equal to 1 or 0. The control function is included in the statement as follows:

$P : R_2 \leftarrow R_1$

This symbolise the requirement that transfer operation be executed by h/w only if $P=1$.

The following table summarize the basic symbol for register transfer microoperation:

Symbols	Description	Example
Letters	Denotes register	MAR, R_2
Parenthesis	Denotes the part of register	R_0 (0-7)
Arrow	Denotes the transfer of information	$R_2 \leftarrow R_1$
Comma	Separate two microoperation	$R_2 \leftarrow R_1$, $R_1 \leftarrow R_2$

2. Arithmetic microoperations :-

The arithmetic microoperation perform arithmetic operation on data stored in register. The basic arithmetic operation are addition, subtraction, increment and decrement.

The arithmetic microoperation defined by statement $R_3 \leftarrow R_2 + R_1$ specifies add microoperation.

It states that content of R_1 are added to the content of R_2 and sum is transferred to register R_3. Some typical arithmetic microoperation are listed in the table below:

Symbolic designation	Description
$R_3 \leftarrow R_1 + R_2$	Content of R_1 plus R_2 transferred to R_3
$R_3 \leftarrow R_1 - R_2$	Content of R_1 minus R_2 transferred to R_3
$R_2 \leftarrow \overline{R_2}$	Complement of the content of R_2 (1's complement)
$R_2 \leftarrow \overline{R_2} + 1$	2's complement of the content of R_2 (-ve)
$R_3 \leftarrow R_1 + \overline{R_2} + 1$	Subtraction
$R_1 \leftarrow R_1 + 1$	increment
$R_1 \leftarrow R_1 - 1$	decrement

Binary Adder :-

To implement the add microoperation with hardware, we need the register that hold the data and the digital component that perform arithmetic addition. The digital circuit that generates the arithmetic sum of two binary number of any length is called the binary adder. The binary adder is constructed with full adder circuit connected in cascade with the output carry from one full adder connected to input carry of next full adder. The figure below shows an interconnection of 4 full adders to provide 4-bit binary adder:

P.T.O

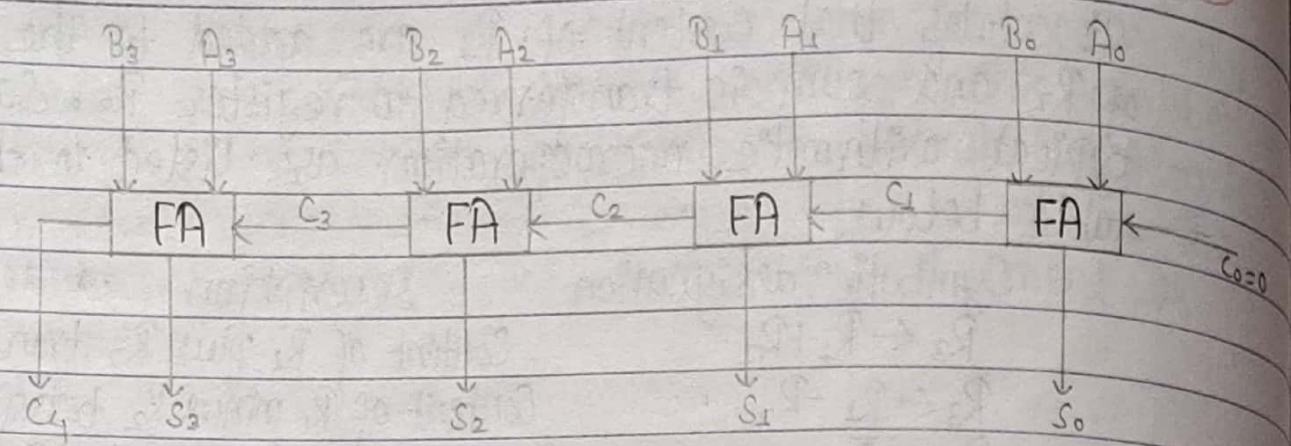


Fig:- 4-bit binary adder

An n-bit binary adder requires n-full adder. The output carry from each full adder is connected to input carry of next higher order full adders.

Binary Subtractor:-

The subtraction A-B can be done by taking the 2's complement of B and adding it to A. It means if we use the inverters to make 1's complement of B by connecting each B_i to an inverter and then add 1 to the least significant bit by setting carry C_0 to 1 of binary adder, then we can make the binary subtractor as a figure below:

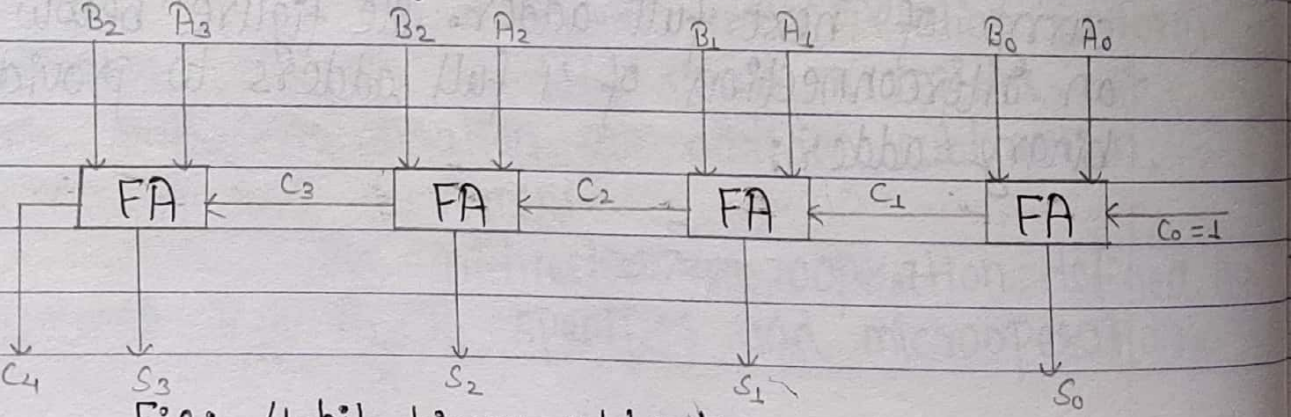


Fig:- 4-bit binary subtractor

Binary adder subtractor:-

The addition and subtraction operation can be combined into one common circuit by including an exclusive-OR gate with each full adder. The 4-bit adder subtractor circuit is shown in figure below:

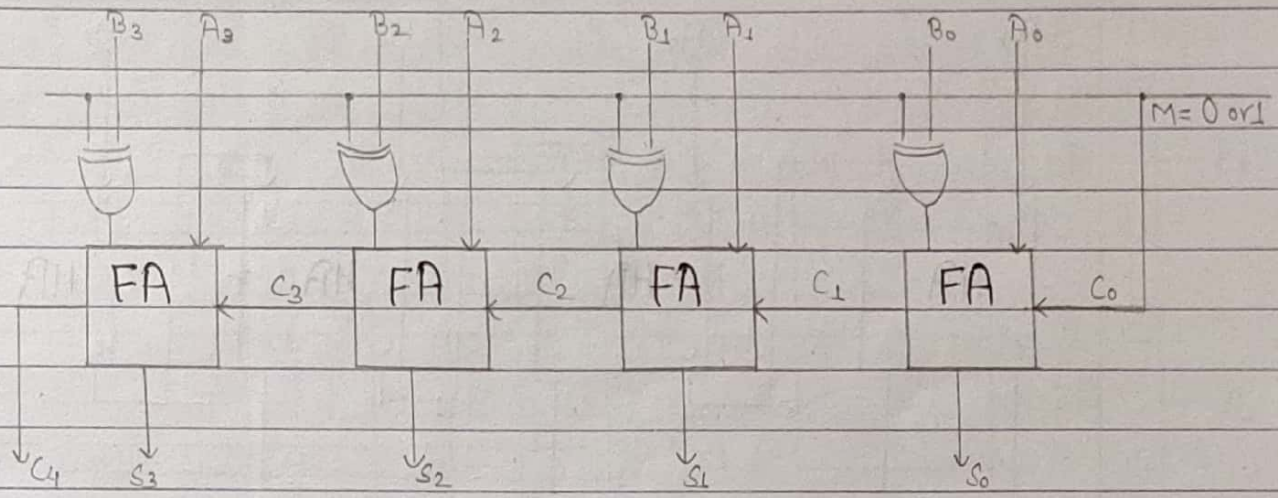


Fig:- 4-bit binary adder subtractor

Here, the mode bit M controls the operation. When M=0, the circuit is an adder and when M=1, the circuit becomes subtractor. When M=0, we have $B \oplus 0 = B$ so full adder receives value of B and input carry is zero. Hence, circuit performs A+B.

When M=1, we have $B \oplus 1 = B'$ and $C_0 = 1$ so the B-input are all complemented and 1 is added through the input carry. Hence, circuit performs A + 2's complement of B i.e. A-B.

Binary incrementer :-

The increment microoperation adds 1 to a number in a register. Increment microoperation can be done with combinational circuit by the means of half adders connected in cascade as in figure below.

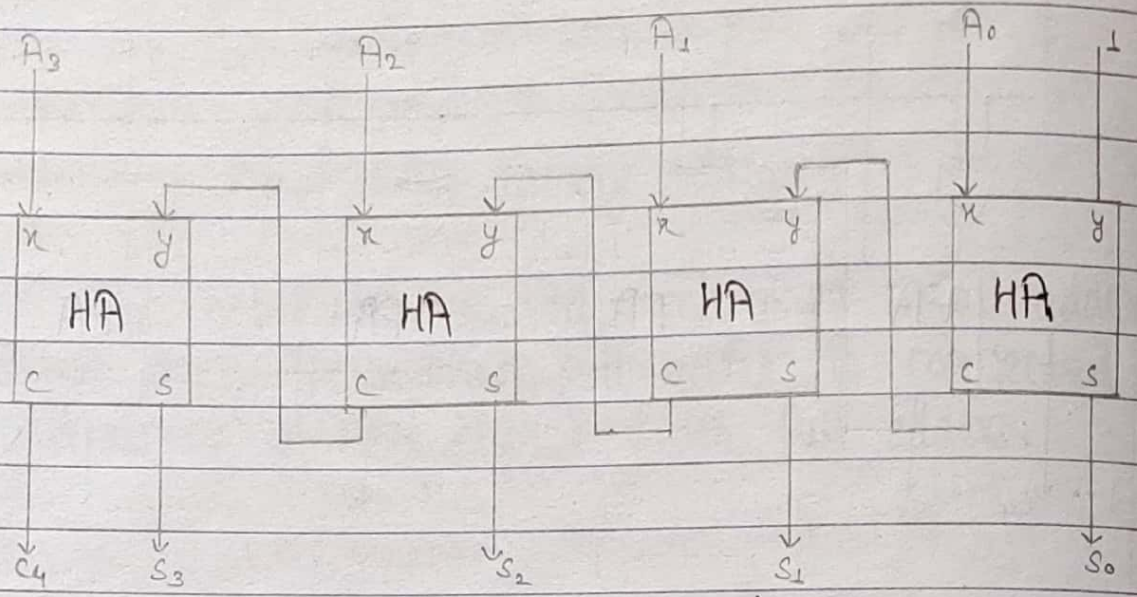


Fig:- 4-bit binary incrementer

This binary incrementer can be extended to an n-bit binary incrementer by extending the diagram to include n-half adders. The least significant half adder must have one input connected to logic 1 and other input receive the no. to be incremented.

Arithmetic circuit :-

The arithmetic microoperations can be implemented in one composite arithmetic circuit. The basic component of arithmetic circuit is parallel adder. By

controlling the data input to the adder, it is possible to obtain different types of arithmetic operations.
The following diagram shows 4-bit arithmetic circuit:

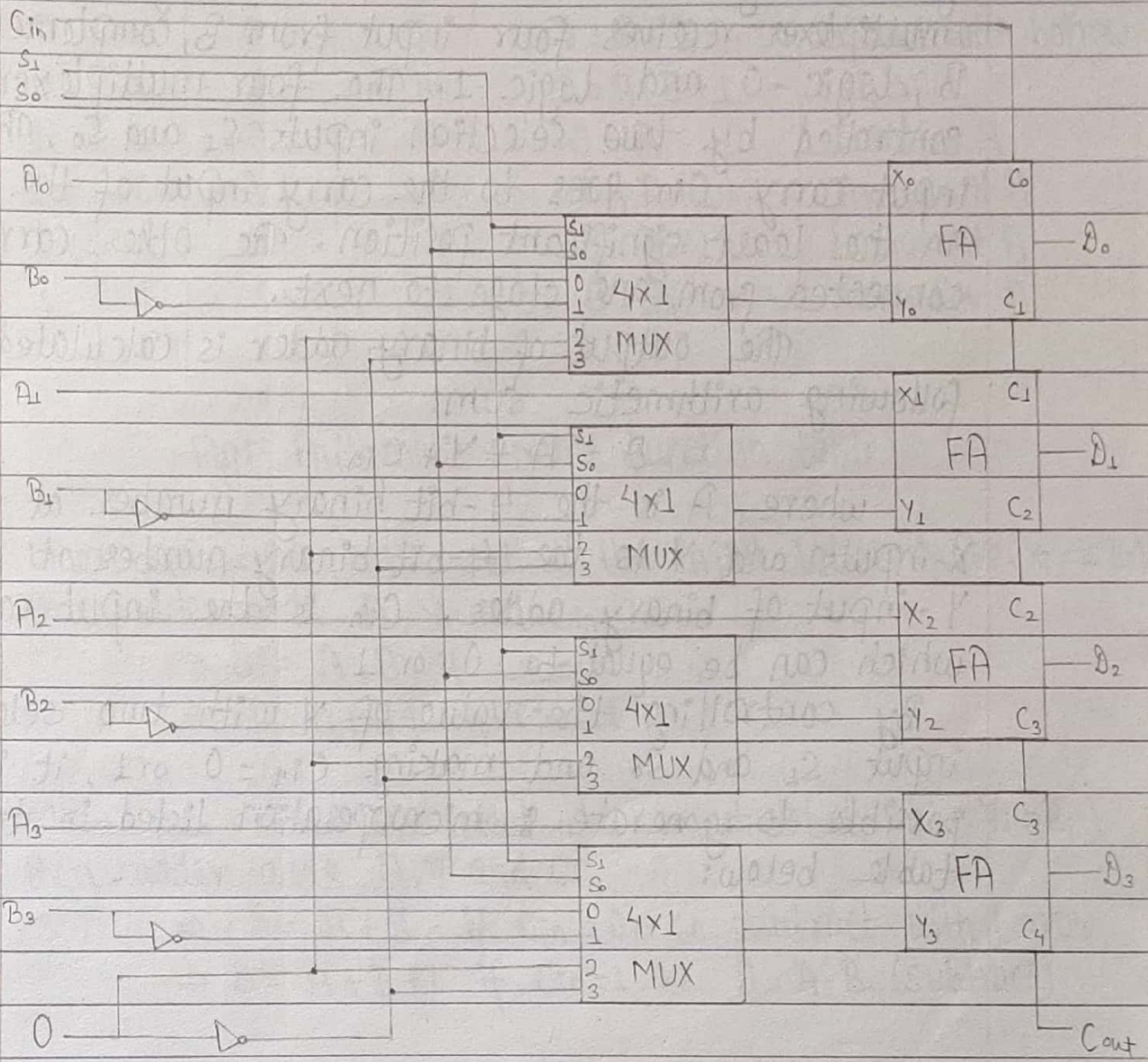


Fig:- 4-bit arithmetic circuit

This arithmetic circuit has four full adder circuit that constitute the 4-bit adder, four multiplexer, 4 bit input A and B and a 4-bit output D. The four input of A goes directly to the X inputs of the binary adder. The multiplexer receives four input from B, complement of B, logic-0 and logic-1. The four multiplexer are controlled by two selection inputs S_1 and S_0. The input carry C_{in} goes to the carry input of the FA in the least significant position. The other carries connected from one stage to next.

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

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

where, A is the 4-bit binary number at the X-inputs and Y is the 4-bit binary number at the Y-input of binary adder. C_{in} is the input carry which can be equal to 0 or 1.

By controlling the value of Y with two selection input S_1 and S_0 and making $C_{in} = 0$ or 1, it is possible to generate 8 microoperation listed in the table below:

Select			input	Output	Microoperation
S_1	S_0	C_{in}	Y	$D = A + Y + C_{in}$	
0	0	0	B	$D = A + B$	Add
0	0	1	B	$D = A + B + 1$	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

Fig:- Arithmetic Circuit function table

→ When $S_1S_0 = 00$, MUX select the input label 0 (i.e. $X_i = <B_i$) then adder adds A, Y and C_{in}.

$\Rightarrow D = A + B$ if $C_{in} = 0$

$\Rightarrow D = A + B + 1$ if $C_{in} = 1$

→ When $S_1S_0 = 01$, MUX select the input label 1 (i.e. $Y_i = \bar{B}_i$) then adder adds A, Y and C_{in}.

$\Rightarrow D = A + \bar{B}$ if $C_{in} = 0$ i.e. subtract with borrow.

$\Rightarrow D = A + \bar{B} + 1$ if $C_{in} = 1$ i.e. $D = A - B$. (subtract)

→ When $S_1S_0 = 10$, MUX select the input label 2 (i.e. Y = 0000)

$\Rightarrow D = A$ if $C_{in} = 0$ i.e. transfer A

$\Rightarrow D = A + 1$ if $C_{in} = 1$ i.e. increment A

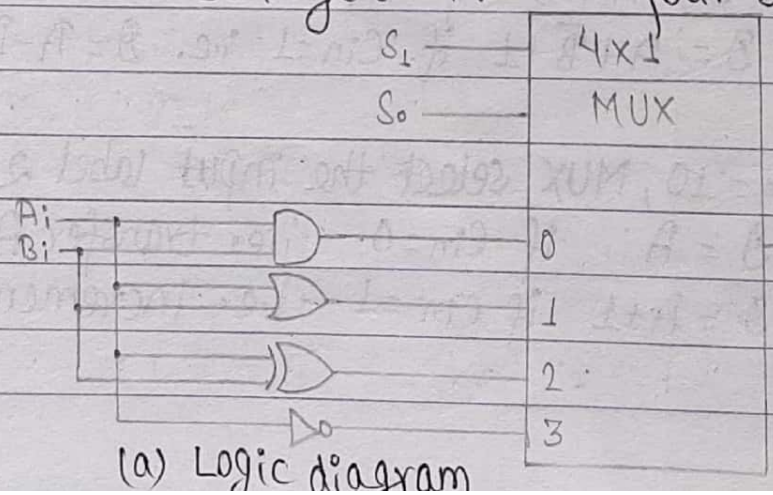
→ When $S_1S_0 = 11$, MUX select the input label 3 (i.e. $X_i = 1111$)
⇒ $B = A-1$ if $C_{in} = 0$ i.e. Decrement A
⇒ $B = A$ if $C_{in} = 1$ i.e. transfer of A

3. Logic microoperations :-

Logic microoperations are bit-wise operation i.e. they work on the individual bits of data. They are very useful for bit manipulation of binary data and for making logical decision. There are, in principle 16 different logic functions that can be defined over two binary input variables. Most of system only implement four of these; they are:
AND ($\wedge$), OR ($\vee$), XOR ($\oplus$), complement / NOT ($^-$).
The other can be created from combination of these.

Hardware implementation of logic microoperation:

The hardware implementation of logic microoperation requires that logic gate be inserted for each bit or pair of bits in the register to perform the required logic function. The figure below shows one stage of a circuit that generates the four basic logic microoperat



(a) Logic diagram

S_1	S_0	Output	Operation
0	0	$E = A \wedge B$	AND
0	1	$E = A \vee B$	OR
1	0	$E = A \oplus B$	X-OR
1	1	$E = \bar{A}$	Complement

(b) Functional table

It consists of four gates and multiplexer. Each of the four logic operation is generated through a gate that performs a required logic. The output of gate are applied to input of multiplexer and it gives direct output according to selection input S_1S_0. The diagram shows one typical stage with subscript 'i'. For a logic circuit with n-bits the diagram must be repeated n-times for $i = 0, 1, 2, \ldots, n-1$.

Application of logic microoperation:

Logic microoperations are very useful for manipulating individual bits or a portion of a word stored in a register. They can be used to change bit values, delete a group of bits or insert a new bit values into a register. The following examples shows how bits of register A are manipulated by logic microoperations as a function of the bits of another register B.

selective set $A \leftarrow A \vee B$

selective complement $A \leftarrow A \oplus B$

selective clear $A \leftarrow A \wedge \bar{B}$

Mask (delete)	$A \leftarrow A \wedge B$
Clear	$A \leftarrow A \oplus B$
Insert	$A \leftarrow (A \wedge B) \vee C$

Selective set:-

The selective set operation sets to 1 the bits in register A where there are corresponding 1's in register B. It does not affect bit position that have 0's in B.

Example:-

```
1010   A (before)
1100   B (logic operand)
----
1110   A after (A ← A ∨ B)
```

selective complement:-

The selective complement operation complements bits in A where there are corresponding 1's in B. It does not affect bit positions that have 0's in B.

Example:-

```
1010   A (before)
1100   B (logic operand
----
0110   A after (A ← A ⊕ B)
```

Selective clear:-

In a selective clear operation, the bit pattern of B is used to clear to certain bits in A. If a bit in B is set to 1, that same position in A gets set to 0, otherwise unchanged. In other word, the selective

clear operation clears to 0 the bits in A only where there are corresponding 1's in B.

Example:-

```
1010   A before
1100   B (logic operand)
0010   A after   (A ← A ∧ B̄)
```

Mask:-

The mask operation is similar to the selective clear operation except that the bits of A are cleared only where there are corresponding 0's in B.

Example:-

```
1010   A before
1100   B (logic operand)
1000   A after   (A ← A ∧ B)
```

Clear:-

The clear operation compares the words in A and B and produces all 0's result if the two numbers are equal. This operation is achieved by an exclusive-OR microoperation. In otherword, in clear operation if the bits in the same position in A and B are same, they are cleared in A, otherwise set in A.

Example:-

```
1010   A before
1100   B (logic operand)
0110   (A ← A ⊕ B)
```

Insert :-

An insert operation is used to introduce a specific bit pattern into A register, leaving the other bit position unchanged.

This is done as:

- Mask operation to clear desired bit position.
- An OR operation to introduce the new bits into the desired position.

Example:-

Suppose we want to introduce 1010 into the low order four bit of A:

```
1101 1000 1011 0001  (original)
1101 1000 1011 1010  (desired)

1101 1000 1011 0001  A before
1111 1111 1111 0000  B (Mask)
-------------------
1101 1000 1011 0001  A after mask (A ← A∨B)
```

Now insert new value.

```
1101 1000 1011 0000  A before
0000 0000 0000 1010  B insert
-------------------
1101 1000 1011 1010  A (Desired) (A ← A∨B)
```

4. Shift microoperations:-

Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic and other data processing operation. The content of register can be shifted to the left or the right. During the shift left operation, the serial input transfer a bit into the rightmost position. During the shift right operation, the serial input transfer a bit into the leftmost position.

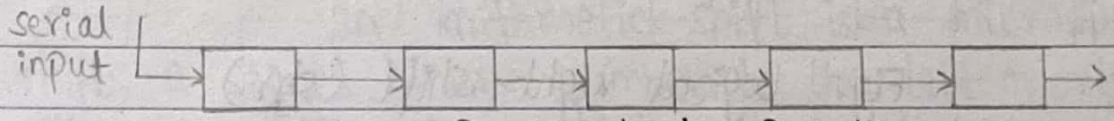


Fig:- Right shift operation

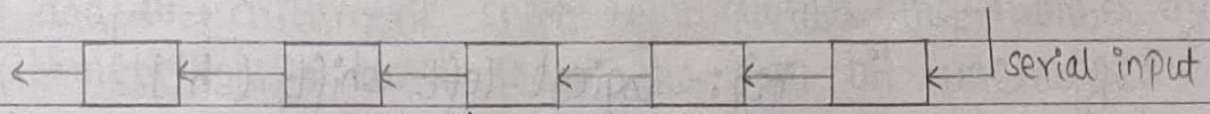


Fig:- Left shift operation.

There are 3 types of shifts:

i) Logical shift
ii) Circular shift
iii) Arithmetic shift.

1) Logical shift:-

A Logical shift is one that transfer 0 through the serial input. In a register transfer language, following notation is used.

shl → for logical shift left
shr → for logical shift right

Example:-

$R_1 \leftarrow shl R_1$

$R_2 \leftarrow shr R_2$

are two microoperations that specify 1-bit shift to the left of the content of register R_1 and 1-bit shift to the right of the content of register R_2. The bit transferred to the end position through the serial input is assumed to be 0 during logical shift.

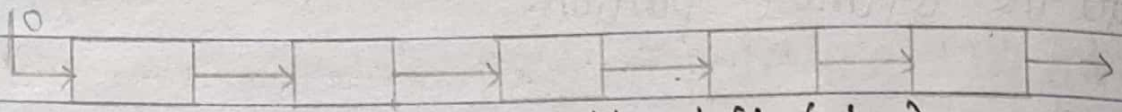


Fig:- Logical right shift (shr)

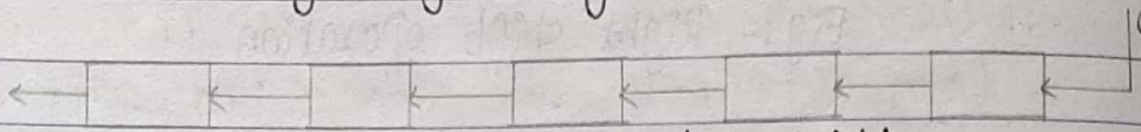


Fig:- Logical left shift (shl)

ii) 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 output of the shift register to its r.serial input.

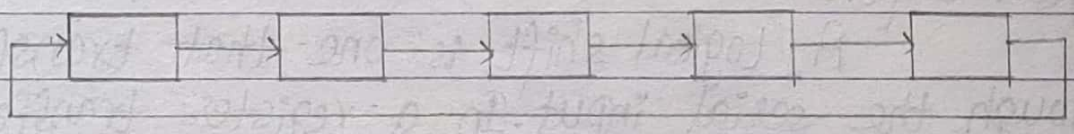


Fig:- Right circular shift

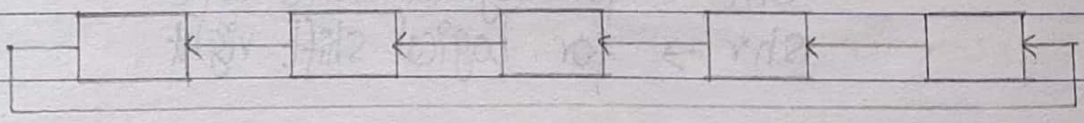


Fig:- Left circular shift

We use,
Cil for circular shift left and,
Cir for circular shift right.

Example:-

$R_2 \leftarrow Cir\ R_2$
$R_3 \leftarrow Cil\ R_3$

(iii) Arithmetic shift :-

An arithmetic shift is a microoperation that shifts a signed binary number to the left or right. An arithmetic shift-left multiplies a signed binary number by 2 and an arithmetic shift right divides the number by 2. Arithmetic shifts must leave the sign bit unchanged because the sign of the number remains the same when it is multiplied or divided by 2. The left most bit in a register holds the sign bit, and the remaining bits hold the number. The sign bit is '0' for positive and '1' for negative. Negative numbers are in 2's complement form.

- Arithmetic shift-right :-

The arithmetic shift right leaves the sign bit unchanged and shift the number including sign bit to the right. Thus, R_{n-1} remains the same, R_{n-2} receives the bit from R_{n-1} and so on for the other bit in the register. The bit in R_0 is lost.

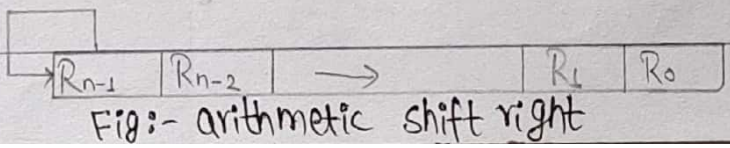


Fig:- arithmetic shift right

- Arithmetic shift-left:-

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

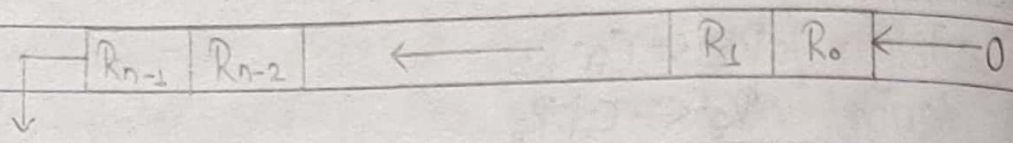


Fig:- arithmetic shift left

Overflow case during arithmetic shift-left:-

If a bit R_{n-1} changes in value after the shift, sign reversal occurs in result. This happens if the multiplication by 2 causes an overflow. Thus left arithmetic shift operation must be checked for the overflow. The overflow occurs after the arithmetic shift if before shift $R_{n-1} \neq R_{n-2}$.

An overflow flip-flop V_s can be used to detect an arithmetic shift-left overflow.

$$V_s = R_{n-1} \oplus R_{n-2}$$

If $V_s = 0$ there is no overflow but if $V_s = 1$ there is an overflow and a sign reversal after the shift.

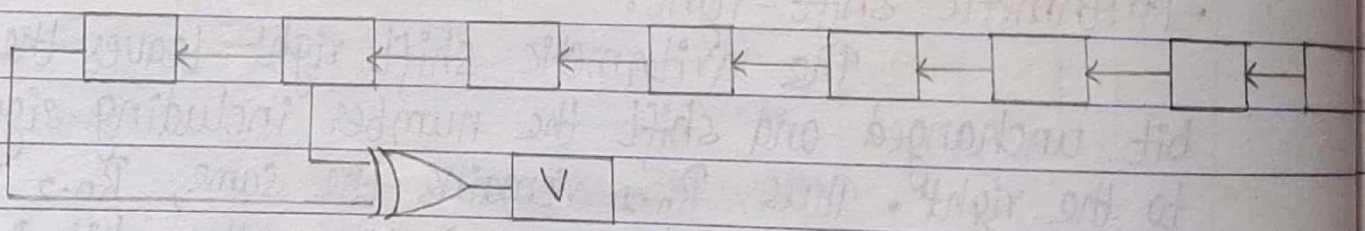


Fig:- overflow detection during arithmetic shift left

The symbolic notation for the shift microoperation is shown in table below:-

Symbolic designation	Description
$R \leftarrow shl\ R$	shift-left register R
$R \leftarrow shr\ R$	shift-Right register R
$R \leftarrow cil\ R$	circular shift-left register R
$R \leftarrow cir\ R$	circular shift-right register R
$R \leftarrow ashl\ R$	arithmetic shift-left register R
$R \leftarrow ashr\ R$	arithmetic shift-right register R.

Hardware implementation of shift microoperation:-

A combinational circuit shifter can be constructed with multiplexer as shown in figure. The 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 other for shift right (I_R). When selection input $S=0$, the input data are shifted right. When $S=1$, the input data are shifted left. A shifter with n data inputs and output requires n multiplexers. The two serial inputs can be controlled by another multiplexer to provide the three possible types of shifts.

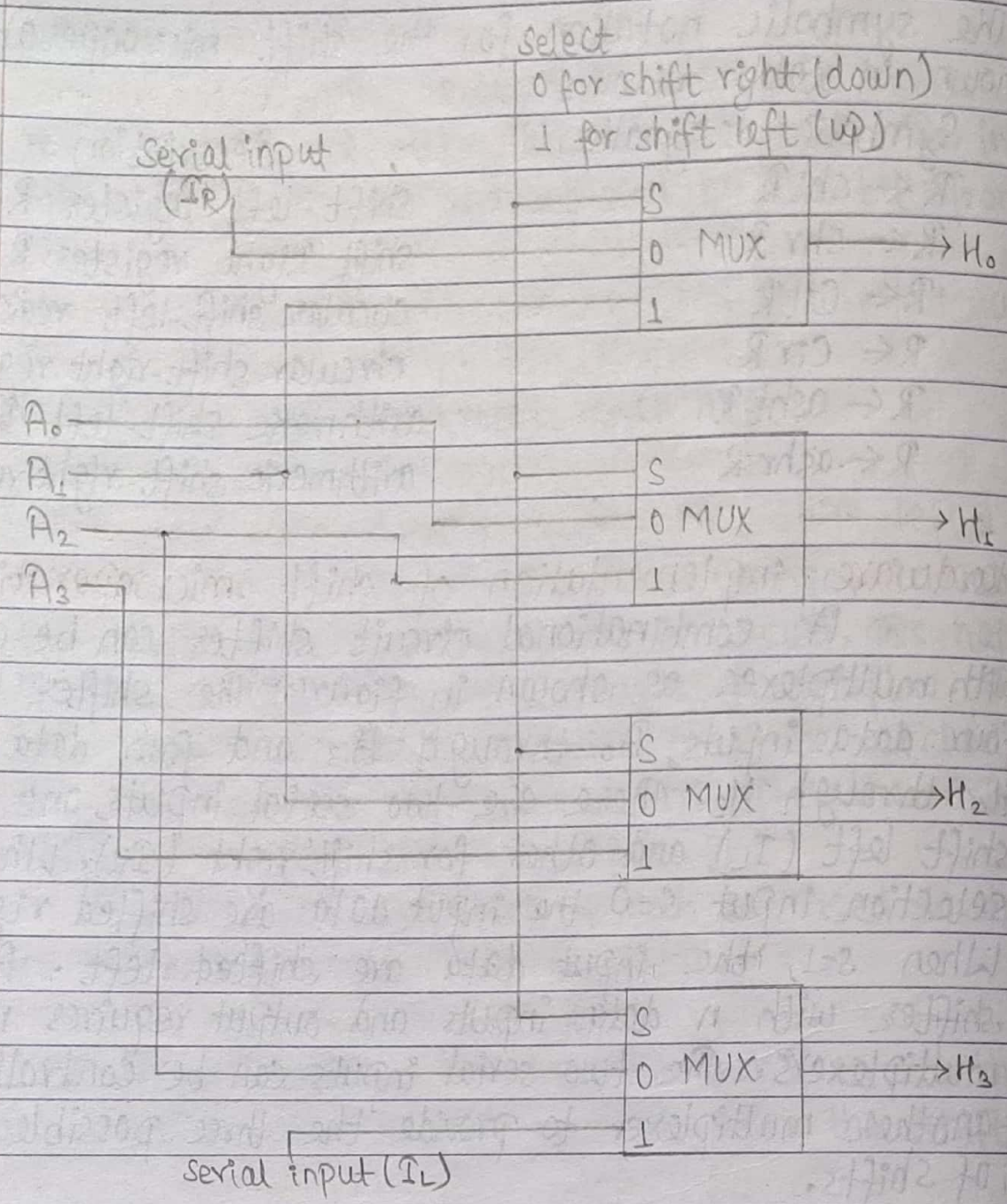


Fig:- 4-bit combinational circuit shifter

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

Arithmetic logic shift unit:-

Instead of having individual registers performing the microoperation directly, computer systems employ a number of storage registers connected to a common operational unit called Arithmetic logic Unit abbreviated as ALU. To perform a microoperation, the content of specified register are placed in the input of common ALU. The ALU perform an operation and the result of operation is then transferred to a destination register. The shift operation are often performed in separate unit but sometimes the shift unit is made part of the overall ALU. The one stage of arithmetic logic shift unit is shown in figure below:

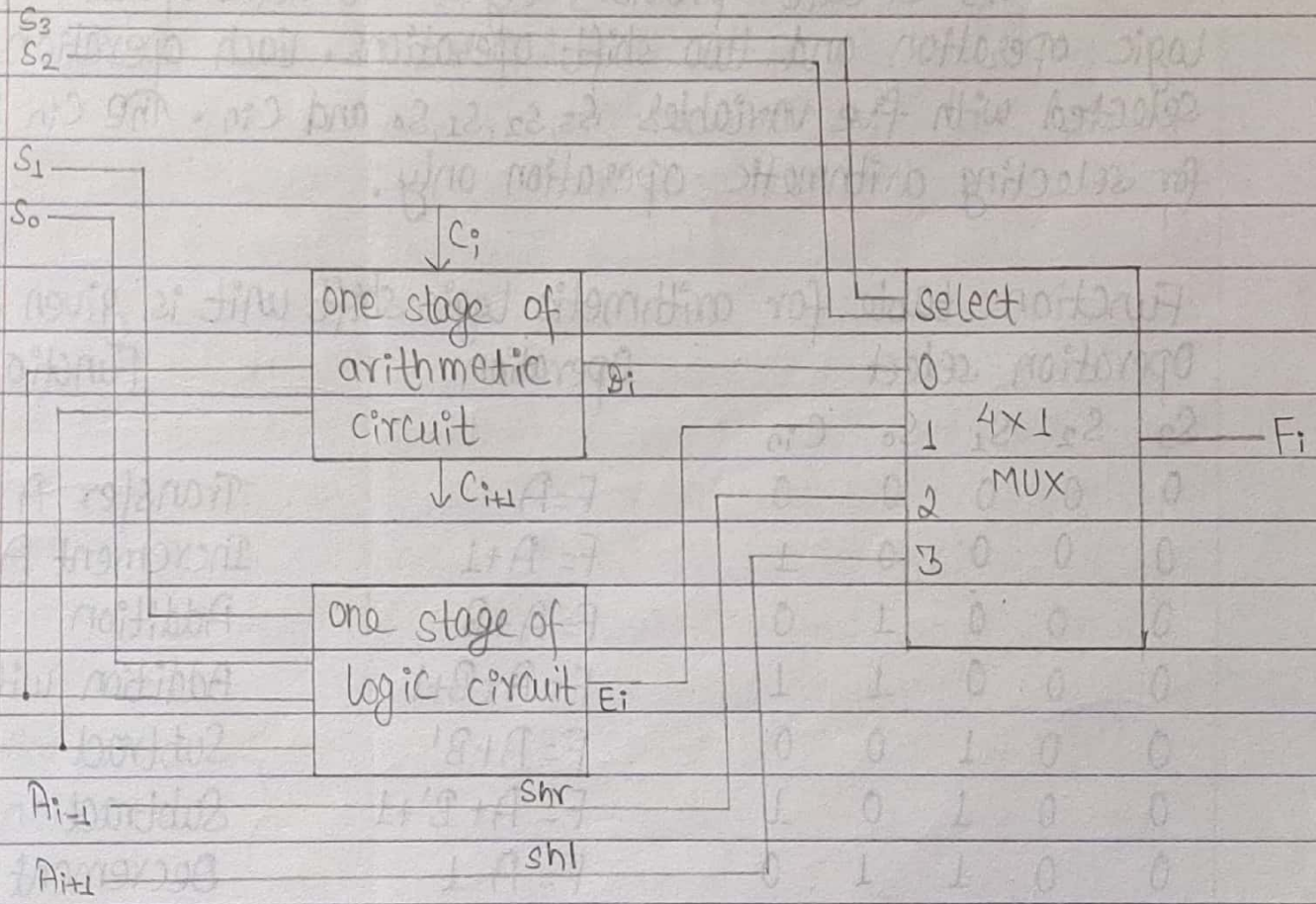


Fig:- one stage of arithmetic logic shift unit.

- A particular microoperation is selected with input S_1 and S_0. A 4x1 multiplexer at the output chooses between an arithmetic output in E_i and logic output in H_i.
- The data in the MUX are selected with input S_3 and S_2. The other two data inputs to the multiplexer receives input A_{i-1} for the shift right operation and A_{i+1} for shift left operation.
- This circuit is repeated n times for n-bit ALU and C_{i+1} of given arithmetic stage must be connected to input carry C_i of next stage in a sequence. The input carry to the first stage is the input carry C_{in} which provides selection variable for arithmetic operation.

This circuit provides eight arithmetic operation, four logic operation and two shift operations. Each operation is selected with five variables S_3, S_2, S_1, S_0 and C_{in}. The C_{in} is used for selecting arithmetic operation only.

Function table for arithmetic logic shift unit is given below:

Operation select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	0	0	0	0	F=A	Transfer A
0	0	0	0	1	F= A+1	Increment A
0	0	0	1	0	F=A+B	Addition
0	0	0	1	1	F=A+B+1	Addition with carry
0	0	1	0	0	F=A+B'	Subtract with borrow
0	0	1	0	1	F= A+B'+1	Subtraction
0	0	1	1	0	F= A-1	Decrement A
0	0	1	1	1	F=A	Transfer A

Operation select					Operation	Function
S_3	S_2	S_1	S_0	C_{in}		
0	1	0	0	x	$F = A \wedge B$	AND
0	1	0	1	x	$F = A \vee B$	OR
0	1	1	0	x	$F = A \oplus B$	XOR
0	1	1	1	x	$F = \bar{A}$	Complement A
1	0	x	x	x	F = shr A	shift right A into F.
1	1	x	x	x	F = shl A	shift left A into F.

Unit 3:

BASIC COMPUTER ORGANIZATION AND DESIGN

Introduction:-

The organization of the computer is defined by its internal registers, the timing and control structure and set of instruction that it uses. The internal organization of a digital system is defined by the sequence of microoperation it perform on data stored in its registers. Every different processor type has its own design (different registers, buses, microoperation, machine instruction, etc). Modern processor is very complex device it contains

- many registers
- multiple arithmetic units for both integers and floating point calculation.
- the ability to pipeline several consecutive instruction for execution speed up.

However, to understand how processor work, we will start with the simplified processor model. M. Moris Mano introduced a simple processor model. He call it as basic computer which is very small compared to commercial computer.

- It has memory of 4096 word in it.
- Each word is 16 bits long.

• $4096 = 2^{12}$ so it takes 12-bit to select a word in memory.

Instruction format of Basic computer :-

A computer instruction is a binary code that specifies the sequence of microoperations for the computer. A computer instruction code is a group of bit that instruct the computer to perform specific operation. It is usually divided into two part:

1) An opcode (operation code):- It specifies the operation of that instruction.

2) An address :- It specifies the register or locations in memory to use for that operation.

In basic computer, since memory contains 4096 ($=2^{12}$) words, we need 12 bit to specify the memory address. In basic computer, bit 15 of the instruction specifies the addressing mode (0 : direct addressing ; 1 : indirect addressing). Since memory word in basic computer are 16-bit long, that leaves 3-bits for opcode of the instruction.

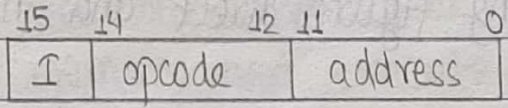


Fig:- instruction format of basic computer

Addressing Mode :-

The address field of an instruction can represent either

1) Direct address:

When the second part of instruction specifies the address of the operand the instruction is said to have direct address. During the direct addressing the bit 15 of the instruction code is 0.

11) Indirect address:

When the second part of instruction specifies the address of a memory word in which address of the operand is found the instruction is to have indirect address. During the indirect addressing bit 15 of instruction code is 1. The indirect address instruction needs two references to memory to fetch an operand. The first reference is needed to read the address of operand, second for the operand itself.

The following figure direct and indirect address:

P.T.O

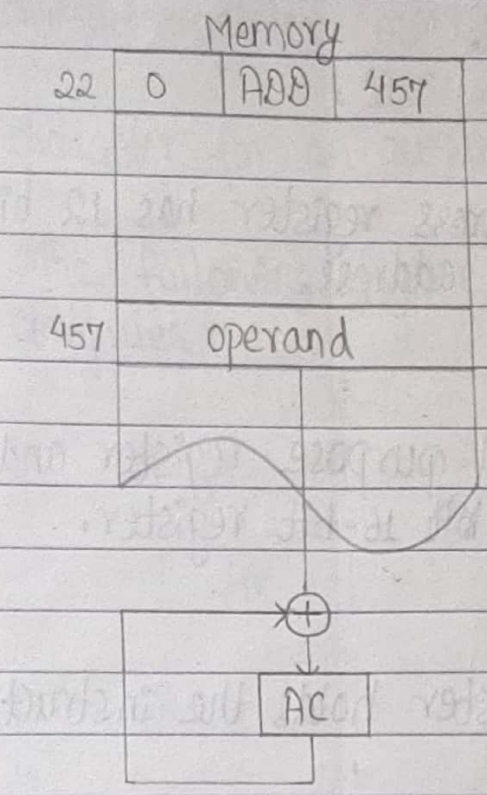


fig:- Direct address

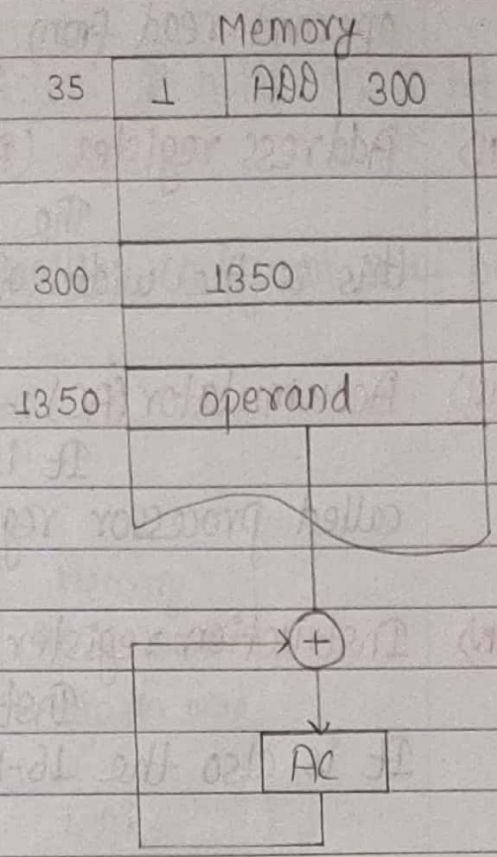


fig:- indirect address

Computer registers:-

Computer instructions are normally stored in consecutive memory locations and are executed sequentially one at a time. The control reads an instruction from specific address in memory and execute it. It then continues by reading the next instruction in sequence and execute it and so on. Thus computer needs register for storing the instruction code after it is read from memory, for manipulating data and for holding the memory address etc.

The following are the register for basic computer:-

1) **Data register (DR):-**

It is the 16 bit register and holds the

operand read from the memory.

II) Address register (AR):-
The memory address register has 12 bit since this is the width of a memory address.

III) Accumulator (AC):-
It is a general purpose register and is also called processor register. It is 16-bit register.

IV) Instruction register (IR):-
Instruction register holds the instruction code. It is also the 16-bit register.

V) Program Counter (PC):-
The program counter has 12 bit and it hold the address of next instruction to be read from the memory after the current instruction is executed.

VI) Temporary register (TR):-
The temporary register is used for holding temporary data during the processing. It is 16-bit register.

VII) Input register (INPR):-
It is an 8-bit register. It receives an 8-bit character from an input device.

viii) Output register (OUTR):-

It is an 8-bit register. It holds an 8-bit character for an output device.

The following figure dipicts the register configuration of basic computer:

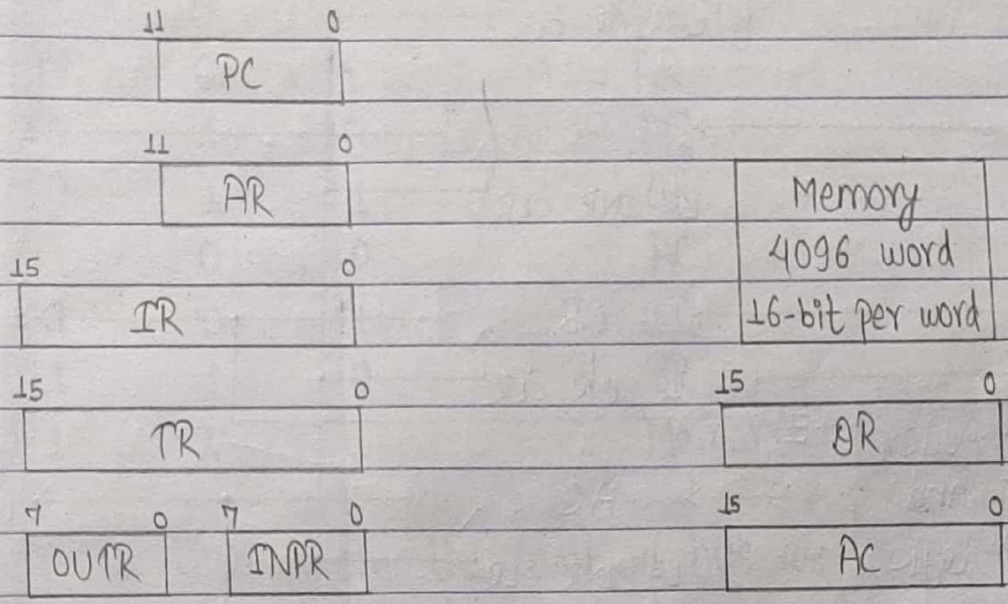


fig:- Basic computer registers and memory.

Common Bus System:-

A more efficient scheme for transferring information in system with many register is to use a common bus. The connection of register and memory of the basic computer to a common bus system is shown in the following figure:-

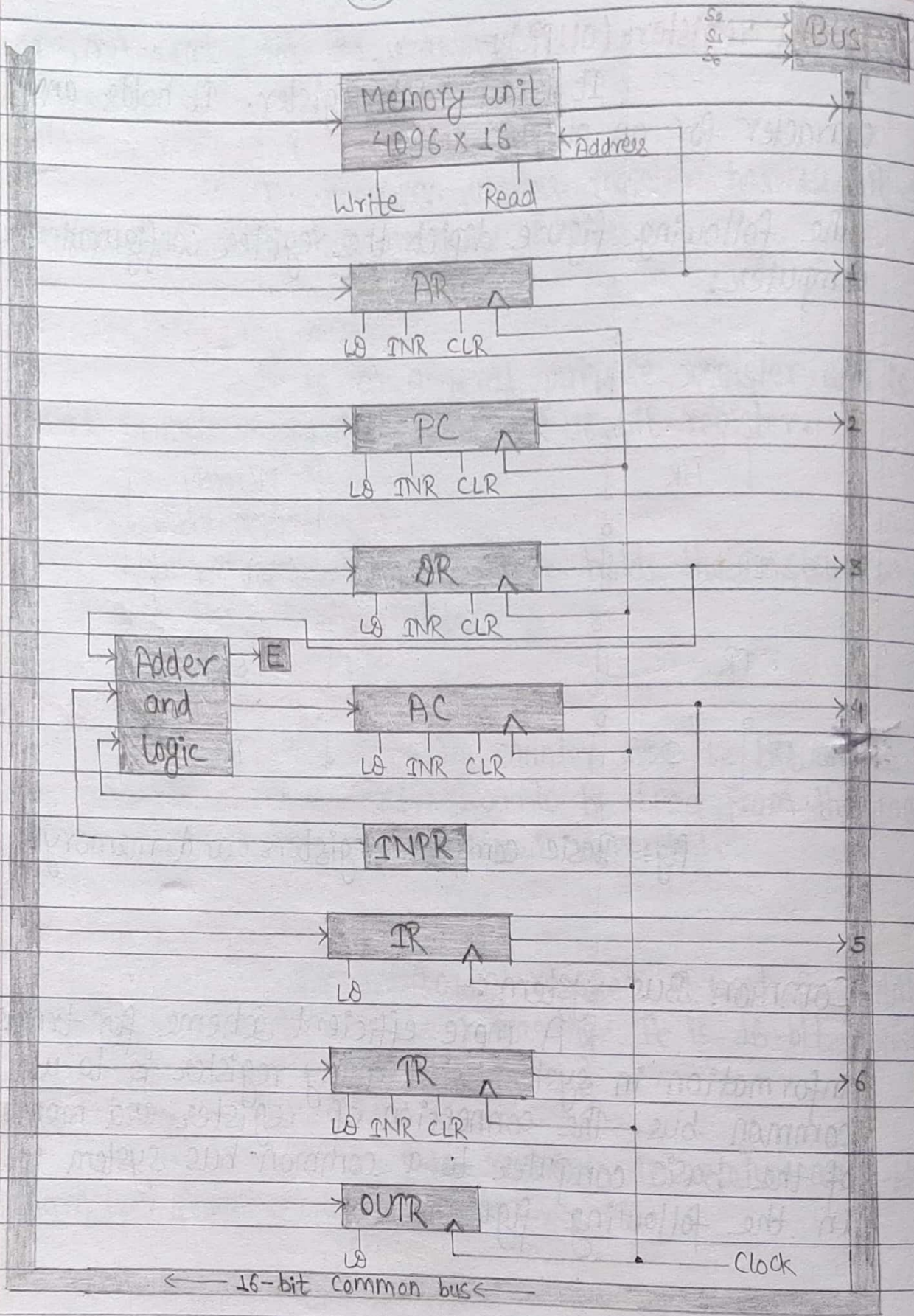


Fig :- Common bus system in Basic computer

In above figure of common bus system, the outputs of six registers and memory are connected to common bus. The three control lines S_2, S_1, S_0 controls which output is selected for the bus lines at any given time.

S_2	S_1	S_0	Register selected
0	0	0	Nothing
0	0	1	AR
0	1	0	PC
0	1	1	DR
1	0	0	AC
1	0	1	IR
1	1	0	TR
1	1	1	Memory

The lines from the common bus are connected to the input of each register and data inputs of memory. The particular register whose (LD) or load input is enabled receives the data from the bus during the next clock pulse transition. The memory receives the contents of the bus when its write input is activated.

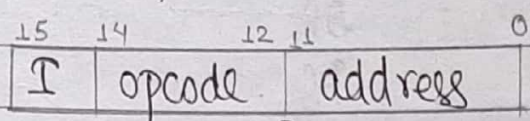
Four registers DR, AC, IR and TR have 16-bits each. Two registers AR and PC have 12 bit each. When the content of AR or PC are applied to 16-bit common bus, the four most significant bits are set to 0's. When AR and PC receive information from the bus only the 12 least significant bits are transferred into register.

The INPR and OUTR have 8-bits each and

communicate with the eight least significant bits in the bus.

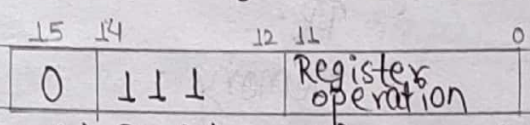
Types of instruction format in Basic Computer:-

The basic computer has three instruction code format as shown in figure below. Each format has 16 bits. The operation code part of the instruction contains three bits and meaning of the remaining 13 bits depends on operation code encountered.



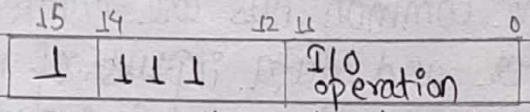
(opcode = 000 through 110)

a) Memory Reference instruction



(opcode = 111, I = 0)

b) Register reference instruction



(opcode = 111, I = 1)

c) Input Output instruction

Fig:- Basic computer instruction format

Types of instruction in Basic Computer :-

1) Memory reference instruction :-

Memory reference instruction uses 12-bit to specify an address and one bit to specify addressing

mode I. I=0 for direct address and 1 for indirect address.
Example:-

AND	AND memory word to AC
ADD	ADD memory word to AC

ii) Register reference instruction:-

These instruction are recognized by operational code 111 with 0 in left most bit of the instruction. A register reference instruction specifies an operation on AC register. An operand from memory is not needed therefore the other 12 bits are used to specify the operation to be executed.

Example:-

CLA	clear AC
CMA	complement AC

iii) I/O instruction:-

The input-output instruction also doesn't need to reference memory and is recognized by operational code 111 with 1 in the leftmost bit of the instruction. The remaining 12 bits are used to specify the type input/output operation performed.

Example:-

INP	input character to AC
OUT	output character from AC.

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

through 14 are not equal to 111, the instruction is memory reference type and the bit in position is taken as the addressing mode I. If a 3-bit opcode is equal to 111, control then inspects the bit in position 15. If this bit is 0, instruction is register reference type. If the bit is 1, the instruction is an input output type.

Basic Computer instruction:

In basic computer, only three bits of the instructions are used for the operation code. It may seem that computer is restricted to a maximum of eight distinct operations. However, since register reference and input-output instruction uses the remaining 12 bit as a part of operation code, total number of instruction can exceed eight. In fact, total number of instructions choosen for the basic computer is equal to 25.

The basic computer instruction are listed in table below:

	Hexadecimal code		
Symbol	I=0	I=1	Description
AND	0xxx	8xxx	AND memory word to AC
ADD	1xxx	9xxx	ADD memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and save return address

Symbol	Hexadecimal code I=0	I=1	Description
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 inst" if AC positive
SNA	7008		Skip next inst" if AC negative
SZA	7004		Skip next inst" if AC zero
SZE	7002		Skip next inst" if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrupt on
IOF	F040		Interrupt off

Table:- Basic Computer Instructions

Instruction Set Completeness:-

A computer should have a set of instructions so that the user can construct machine language programs to evaluate any function that is known to be computable. The set of instruction are said to be complete if it contains a sufficient number of instructions in each of the following categories:-

i) Arithmetic, logical and shift instruction.
ii) Instruction for moving information to and from memory and processor registers.
iii) Program control instruction together with instructions that check status condition.
iv) Input and output instructions.

Is instruction set of basic computer complete?

- The instruction set of basic computer is complete because there is instruction ADD, CMA, INC, can be used to perform addition and subtraction. The circulate instruction, CIR and CIL instruction can be used for arithmetic shifts as well as any other type of shift desired. Addition, subtraction and shifting can be used together to achieve multiplication and division. AND, CMA and CLA can be used to achieve any logical operation. Moving information from memory to AC is accomplished with the LDA instruction. Storing information from AC to memory is accomplished with STA instruction.

The branch instruction BUN, BSA instruction provide the mechanism of program control. INP instruction

BSA → Branch and save return address
BUN → Branch unconditionally

is used to read data from input device and OUT instruction is used to send data from processor to output device.

Instruction Processing and instruction cycle of Basic Computer:

Control Unit :-

The control unit is the part of CPU whose function in a digital computer is to initiate sequence of microoperations. Control unit of a processor translate from machine instructions to the control signals, for the microoperations that implement them. The number of microoperation that are available in a given system is finite. The complexity of digital system is derived from the number of sequence of microoperations that are performed. Two methods of implementing control unit are:-

i) Hardwired control
ii) Microprogrammed control

i) Hardwired control :-

In hardwired control, the control logic is implemented with gates, flip flops, decoders and other digital circuits. It has the advantage that it can be optimized to produce a fast mode of operation. A hardwired control as the name implies, requires changes in wiring among the various component if the design has to be changed or modified.

The design of hardwired control involves the use of fixed instructions, fixed logic blocks, encoders,

decoders, etc. The key characteristics of hard wired control logic are high speed operation, expensive, relatively complex and no flexibility of adding new instructions. Examples of CPUs with hardwired logic control are Intel 8085, Motorola 6802, Zilog 80 and any RISC CPUs.

In short, when the control signal are generated by hardware using conventional logic design technique, the control unit is said to be hardwired.

Microprogrammed Control:-

In microprogrammed control, the control information is stored in a control memory. The control memory is programmed to initiate the required sequence of microoperations. In microprogrammed control, any required changes or modification can be done by updating the microprogram in control memory. Most computer based on CISC architecture have microprogrammed control. Examples of CPU's with microprogrammed control unit are intel 8080, Motorola 68000, etc.

The main advantage of microprogrammed control is the fact that once the hardware configuration is established, there should be no need for further hardware or wiring changes. If we want to establish a different control sequence for the system, all we need to do is specify a different set of microinstructions for control memory. The hardware configuration should not

be changed for different operations, the only thing that must be changed is the micro program residing in control memory.

Difference between hardwired and microprogrammed control:

Hardwired control	Microprogrammed control
- In hardwired control, the control logic is implemented with gates, flip flop, decoder and other digital circuit to generate the control signal.	- In microprogrammed control, control information is stored in control memory that activate the necessary control signal.
- If logic is changed, we need to change the whole circuit.	- If logic is changed, we only need to change the microprogram in control memory.
- It can be optimized to produce fast mode of operation.	- It is slow.
- Mainly used in RISC CPU.	- Mainly used in CISC CPU.

Hardwired control unit of Basic Computer:-

The block diagram of hardwired control unit is shown below. It consist of two decoders, a sequence counter and number of control logic gates.

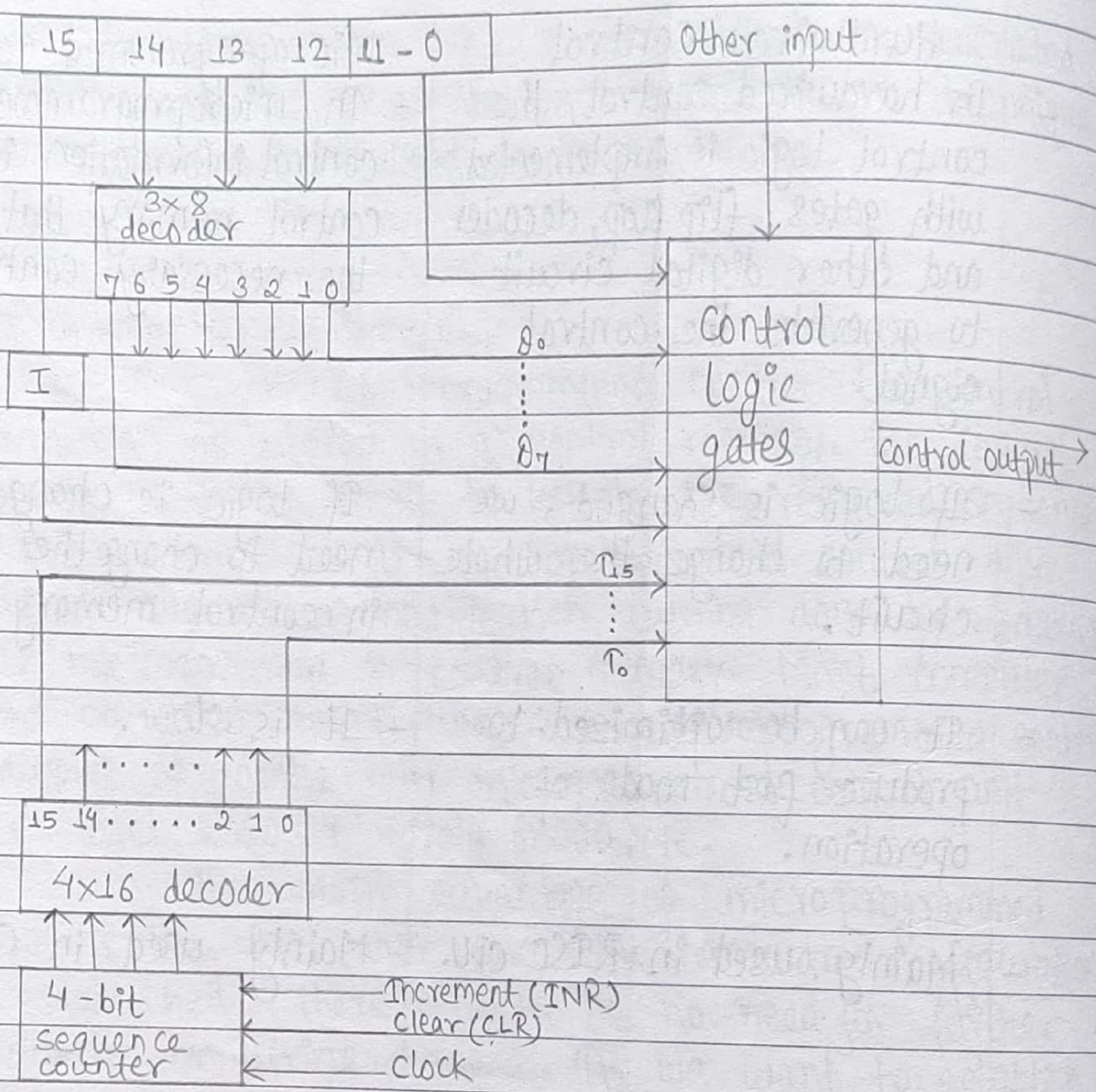


Fig:- Control unit of basic computer

An instruction read from memory is placed in the instruction register (IR) which is divided into three parts: the I bit, operation code and bits 0 through 11. The operation code in bits 12 through 14 are decoded with a 3x8 decoders, producing the output D_0 through D_7. The subscriped decimal number is equivalent to binary value of corresponding operation code. Bit 15 of the instruction is transferred to a flip-flop designated by the symbol I. Bits through 0 to 11 are applied to control logic gates. The 4-bit sequence counter can count in binary from 0 through 15. The output of 4-bit sequence counter are decoded into 16 timing signals T_0 through T_{15}. This means that instruction cycle of basic computer can not take more than 16 clock cycles.

Timing signal :-

The timing signal are generated by decoding the output of 4-bit sequence counter. The sequence counter can be incremented or cleared. Once the counter is cleared to 0, causing the next active timing signal to be T_0. When SC is incremented, it provide the timing signals T_0, T_1, T_2, T_3 and T_4 in sequence.

Assume: At time T_4, SC is cleared to 0 if decoder output D_3 is active

i.e. $D_3T_4 : SC \leftarrow 0$

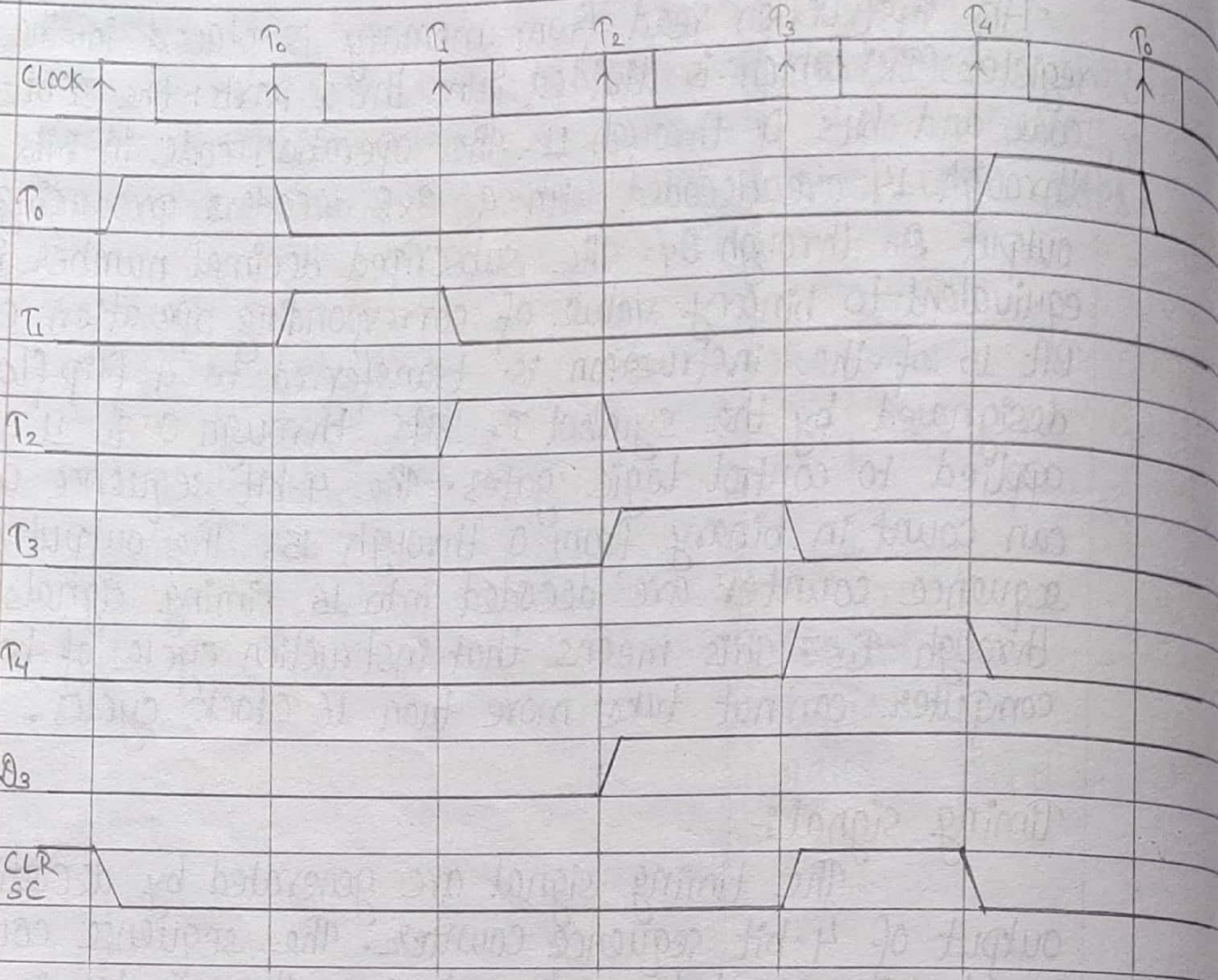


Fig:- Example of control timing signals

Instruction Cycle:-

A program residing in the memory unit of the computer consists of sequence of instruction. The program is executed in the computer by going through a cycle for each instruction. In 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

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

Fetch and decode :-

The microoperations for the fetch and decode phase can be specified by the following register transfer statement.

T_0 : AR ← PC
T_1 : IR ← M[AR], PC ← PC+1
T_2 : $D_0 \ldots D_7$ ← Decode IR (12-14), AR ← IR (0-11), I ← IR (15).

The following figure shows how the first two register transfer statement are implement in the bus system. To provide the data path for the transfer of PC to AR we must apply timing signal T_0 to achieve the following connection:

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

In order to implement the second statement
i.e. T_1 : IR ← M[AR], PC ← PC+1
It is necessary to use timing signal T_1 to provide the following connections in the bus system.

i) enable the read input of memory
ii) place the content of memory onto the bus by making $S_2S_1S_0 = 111$
iii) Transfer the content of the bus to IR by enabling the LD input of IR.
iv) Increment PC by enabling the INR input of PC.

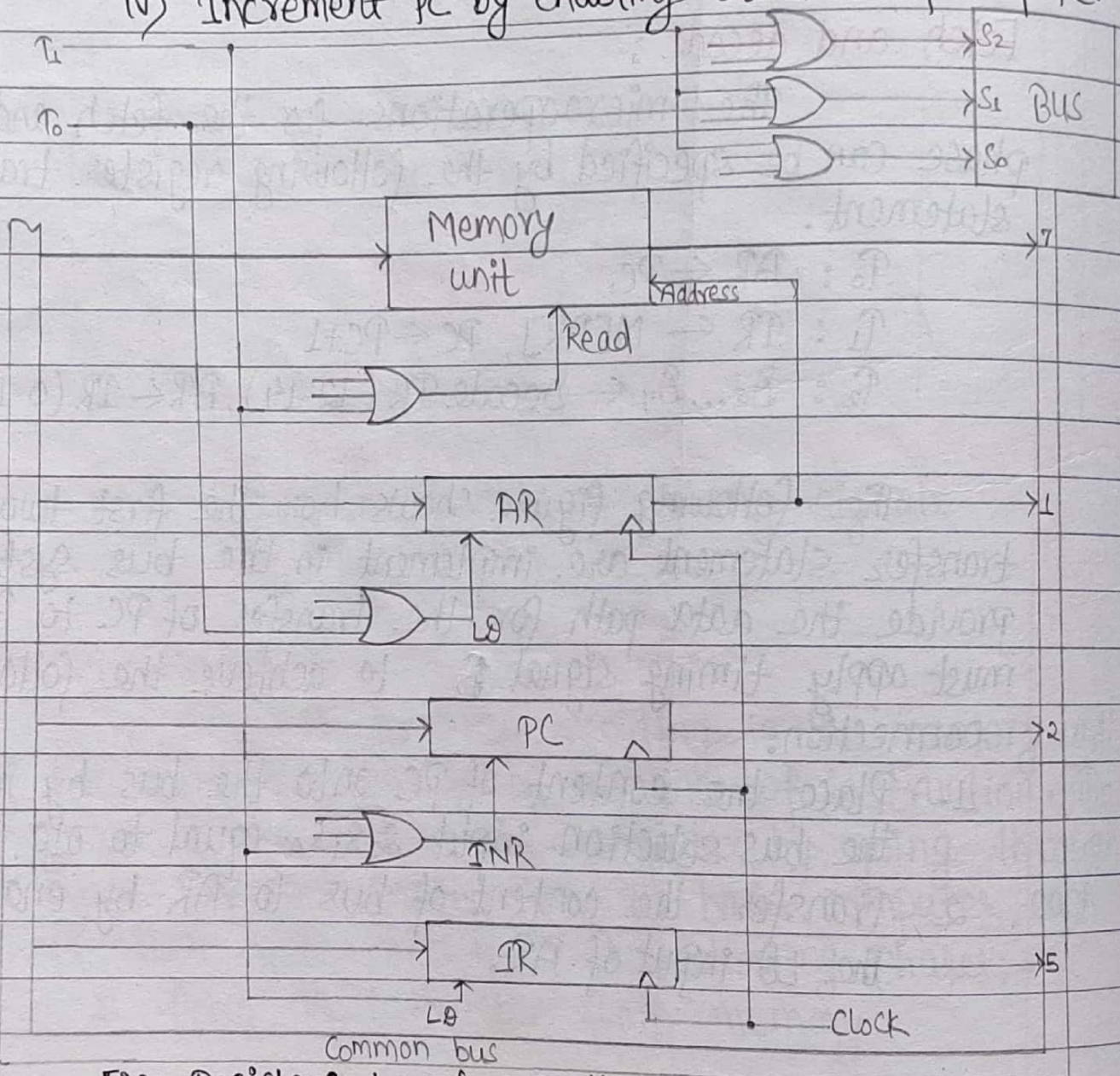


Fig:- Register transfers for the fetch Phase

Flowchart for determining the type of instruction:-

The following flowchart presents an initial configuration for the instruction cycle and shows how the contrd determines the instruction type after the decoding.

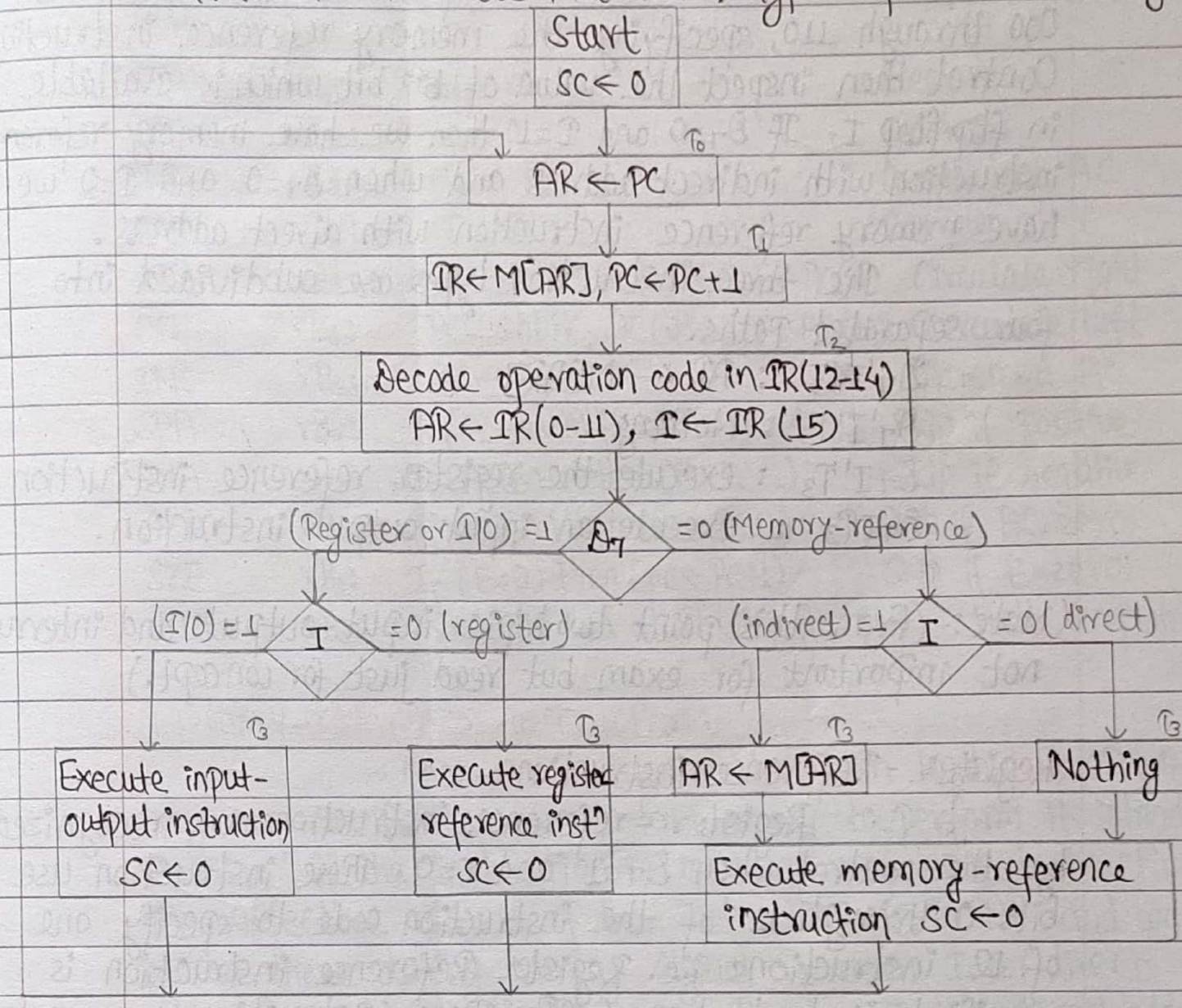


Fig:- Flowchart for instruction cycle (initial configuration)

Decoder output D_7 is equal to 1 if the operation code is equal to binary 111. If $D_7=1$, the instruction must be a register-reference or input-output type. If $D_7=0$, the operation code must be one of the other seven values 000 through 110, specifying the memory reference instruction. Control then inspect the value of 15th bit which is available in flip flop I. If $D_7=0$ and $I=1$ then we have memory reference instruction with indirect address and when $D_7=0$ and $I=0$ we have memory reference instruction with direct address.

The three instruction types are subdivided into four separated paths.

$D_7'IT_3$: $AR \leftarrow M[AR]$

$D_7'I'T_3$: Nothing

$D_7I'T_3$: execute the register reference instruction

D_7IT_3 : execute an input output instruction.

(Note:- From this point two topic input-output and interrupt not important for exam but read just for concept.)

Register-Reference instruction :-

Register-reference instructions are recognized by the control when $D_7=1$ and $I=0$. These instruction use bits 0 through 11 of the instruction code to specify one of 12 instructions i.e. Register Reference instruction is specified in b_0 to b_{11} of IR. These instruction are executed with the clock transition associated with timing variable T_3.

Let, $r = D_7I'T_3$ (common to all register reference instruction)

$B_i = IR(i)$ $i = 0, 1, \ldots 11$. [Bit in IR (0-11) that specifies the operation].

Then, the control functions and microoperations for the register reference instruction are listed below:-

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:	$AC \rightarrow AC + 1$	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 $(PC \leftarrow PC + 1)$	Skip if AC zero
SZE	rB_1:	If $(E = 0)$ then $(PC \leftarrow PC + 1)$	Skip if E zero
HLT	rB_0:	$S \leftarrow 0$ (S is a start-stop flip-flop)	Halt computer

Memory-reference instruction:-

Once an instruction has been loaded to IR, it may require further access to memory to perform its intended function, if the instruction is memory-reference instruction. The effective address of the instruction is in AR and was placed there during timing signal T_2 when $I=0$ (direct address) or during timing signal T_3 when $I=1$ (indirect address). The execution of memory-reference instruction start with timing signal T_4. The symbolic description of each instruction is specified in the following table in terms of

memory register transfer notation:

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

AND to AC:-

This instruction perform the AND logic operation on pairs of bits in AC and the memory word specified by the effective address and result of operation is transferred to AC.

The micro operation that execute this instruction are:

D_0T_4 : $DR \leftarrow M[AR]$ // Read operand

D_0T_5 : $AC \leftarrow AC \wedge DR, SC \leftarrow 0$ // AND with AC

ADD to AC:-

D_1T_4 : $DR \leftarrow M[AR]$ // Read operand

D_1T_5 : $AC \leftarrow AC + DR, E \leftarrow C_{out}, SC \leftarrow 0$ // Add to AC and store carry in E.

LDA: Load to AC :-

D_2T_4 : $DR \leftarrow M[AR]$ // Read operand

D_2T_5 : $AC \leftarrow DR, SC \leftarrow 0$ // Load AC with DR.

STA: Store AC :-

D_3T_4 : M[AR] ← AC, SC ← 0 // store data into memory location.

BUN : Branch Unconditionally :-

D_4T_4 : PC ← AR, SC ← 0 // Branch to specified address

BSA: Branch and Save Return Address :-

D_5T_4 : M[AR] ← PC, PC ← AR + 1

D_5T_5 : PC ← AC, SC ← 0

ISZ : Increment and skip if zero :-

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.

Input - Output and Interrupt :-

A computer can serve no useful purpose unless it communicates with the external environment. Instructions and data stored in memory must come from some input device. The computational result must be transmitted to the user through some output device. Commercial computer include many types of input and output devices such as keyboard, printer, mouse, etc.

Input - Output Configuration :-

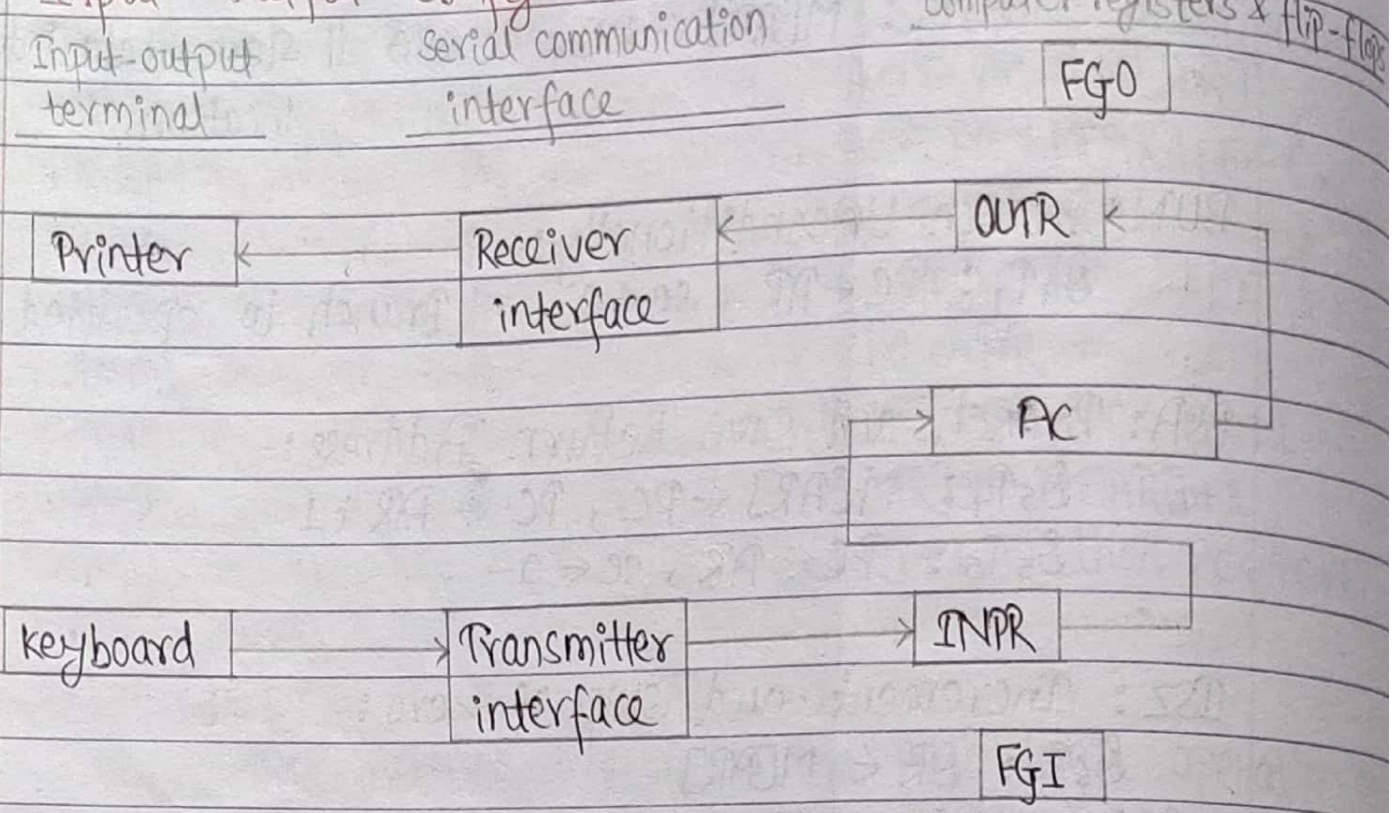


Fig :- Input - Output Configuration

Figure above shows input-output configuration. The terminal send and receives serial information. Each quantity of information has eight bits of alphanumeric code. The serial information from the keyboard is shifted into the input register INPR. The serial information for the printer is stored in the output register. These two registers communicate with a communication interface serially and with the AC in parallel.

i) Input register:-

The input register INPR consist of 8-bits and holds an alphanumeric input information. The 1-bit input flag is control flip-flop. The flag bit set to 1 when new information is available in the input device and is cleared to zero when the information is accepted by the computer. The process of information transfer is as follows :-

Initially, the input flag FGI is cleared to 0. When key is struck in the keyboard, an 8-bit alphanumeric code is shifted into INPR and input flag FGI is set to 1. As long as the flag is set, the information in INPR cannot be changed by striking the another key. The computer checks the flag bit, if it is 1, the information from INPR is transferred in parallel into AC and FGI is cleared to 0. Once the flag is cleared, new information can be shifted into INPR by striking another key.

ii) Output register:-

The output register OUTR works similarly but the direction of information flow is received. Initially, the output flag FGO is set to 1. The computer checks the flag bit; if it is 1, the information from AC is transferred in parallel to OUTR and FGO is cleared to 0. The output device accepts the coded information, prints the corresponding character, and when the operation is completed, it sets FGO to 1.

Input-Output Instruction:-

- Input and output instruction are needed for transferring information to and from AC register, for checking the flag bits and controlling the interrupt facility.
- Input-Output instructions have an operation code 1111 and are recognized by the control when $D_7 = 1$ and $I = 1$. The remaining bits of the instruction specify the particular operation.

The control function and microoperation for input-output instructions are listed below:

Let, $D_7 I T_3 = P$ (common to all input-output instruction).
$IR(i) = B_i$ (bits in IR (6-11) that specifies the instructi

	$P:$	$SC \leftarrow 0$	Clear SC
INP	$pB_{11}:$	$AC(0-7) \leftarrow INPR, FGI \leftarrow 0$	Input character
OUT	$pB_{10}:$	$OUTR \leftarrow AC(0-7), FGO \leftarrow 0$	Output character
SKI	$pB_9:$	if(FGI=1) then ($PC \leftarrow PC+1$)	skip on input flag
SKO	$pB_8:$	if(FGO=1) then ($PC \leftarrow PC+1$)	skip on output flag
ION	$pB_7:$	$IEN \leftarrow 1$	Interrupt enable on
IOF	$pB_6:$	$IEN \leftarrow 0$	Interrupt enable off

Program interrupt:-

In programmed control transfer, the computer keeps checking the flag bit and when it find it set, it initiate the information transfer. The difference of information flow rate between the computer and that of input-output device makes this type of transfer inefficient because the computer is wasting time while checking the flag instead of doing some useful processing task.

An alternative to the programmed controlled procedure is to let the external device inform the computer when it is ready for the transfer. This type of transfer uses interrupt facility. While the computer is running the program, it does not check the flags. However, when the flag is set, the computer is momentarily interrupted from processing with the current program and informed of the fact that ~~they~~ flag has been set. The computer deviates momentarily from what it is doing to take care of input or output transfer. It then return to current program to continue what it was doing before the interrupt.

Interrupt enable flip-flop (IEN):-

The IEN can be set and cleared with two instructions. When IEN is cleared to 0 with IOF instruction, the flag cannot interrupt the computer. When IEN is set to 1 with ION instruction, the computer can be interrupted.

Flowchart for interrupt cycle:-

The way that the interrupt is handled by the computer can be explained by means of flowchart shown in figure below:

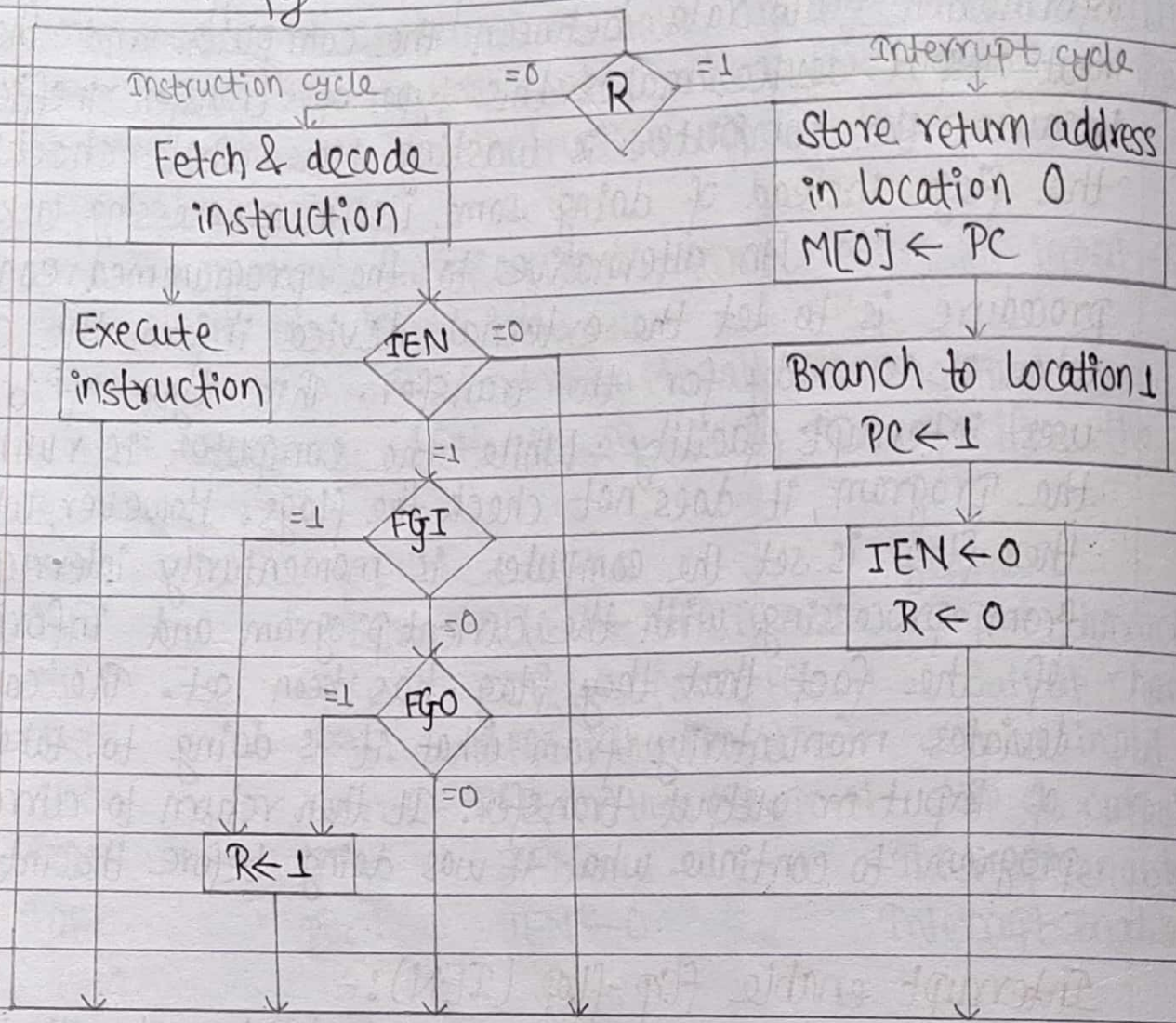


Fig:- Flowchart for interrupt cycle.

When R=0, the computer goes through an instruction cycle and during execution phase IEN is checked by control. If it is 0, it indicate that programmer does not want to use the interrupt but if it is 1, the control checks the flag bit.

if both flag are 0, it indicates neither input nor output register are ready for transfer. In such a case control continues with next instruction cycle. If either flag is 1 while IEN=1, flipflop R is set to 1. At the end of execute phase, control checks value of R and if it is equal to 1, goes an interrupt cycle instead of instructruction cycle.

Interrupt cycle:-

The interrupt cycle is a hardware implementation of branch and save return address operation. The return address available in PC is stored in a specified location where it can be found later. In the flowchart we choose the memory location at address 0 as the place for storing the return address control then insert address 1 to PC and clears IEN and R so that no more interruption can occur until the interrupt request from the flag has been serviced.

As an example following figure shows what happens during the interrupt cycle.

Memory

0	
1	0 BUN 1120
	Main
256	Program
PC=256	
1120	I/O
	program
	1 BUN 0

(a) before interrupt

Memory

0	256
PC=1	0 BUN 1120
255	
256	Main Program
1120	I/O program
	1 BUN 0

(b) after interrupt

Fig:- Demonstration of the interrupt cycle.

Register transfer operation in interrupt cycle:-

- The interrupt cycle is initiated after the last execute phase if 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.
- So the condition for setting flip-flop R to 1 can be expressed with the following register transfer statement.

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

- The fetch and decode phases will be recognized from the three control function $R'T_0$, $R'I$, $R'T_2$.
- The interrupt cycle stores the return address available in PC in memory location 0, branches to memory location 1 and clears IEN, R and SC to 0. This can be done with following sequence of microoperations.

$RT_0: AR \leftarrow 0, TR \leftarrow PC$

$RT_1: M[AR] \leftarrow TR, PC \leftarrow 0.$

$RT_2: PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0.$

Complete description of basic computer :-

The final flowchart of instruction cycle, including the interrupt cycle for the basic computer is shown in figure below:

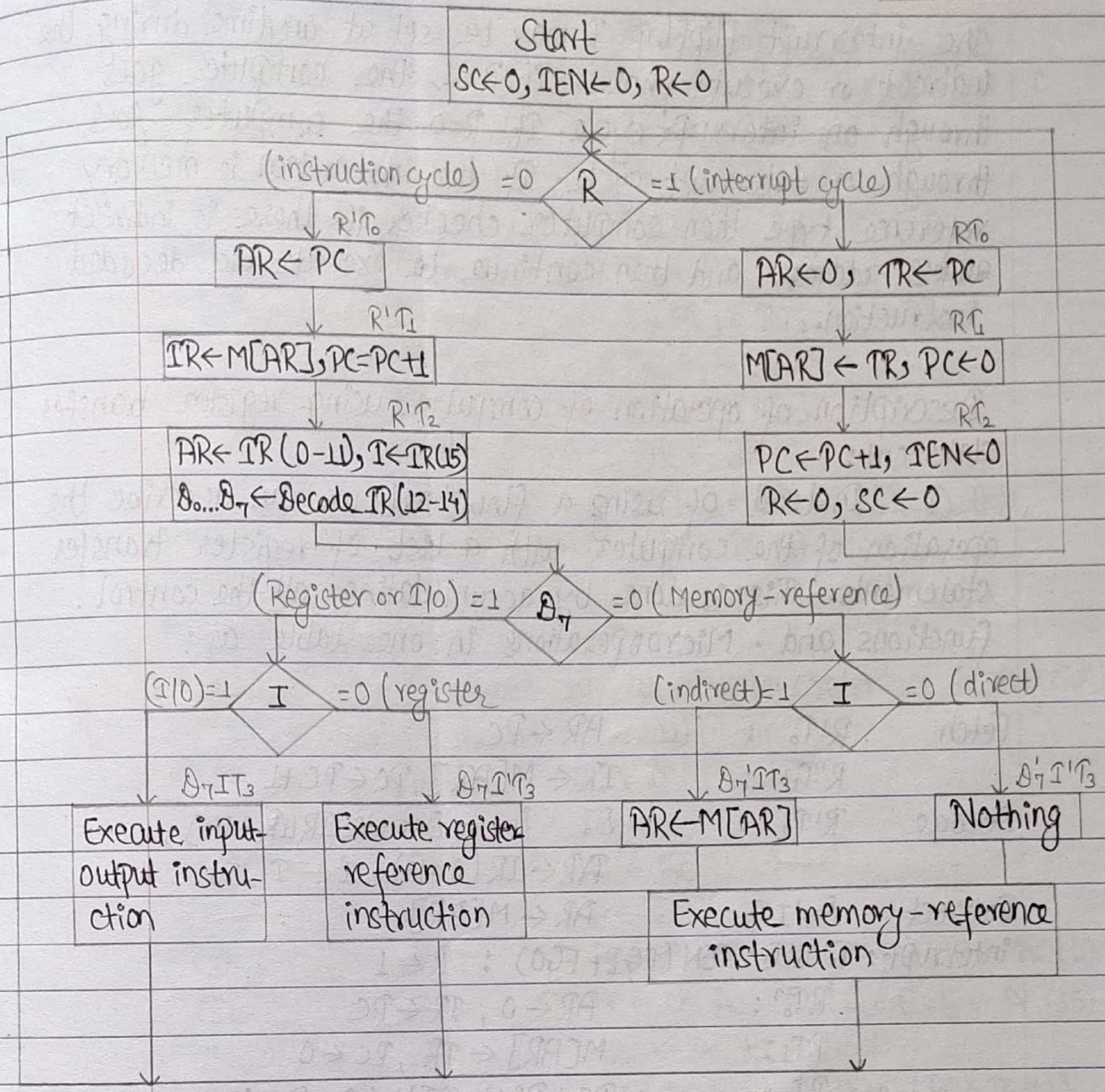


Fig:- Flowchart for computer operation.

The interrupt flipflop R may be set at anytime during the indirect or execute phase. If R=1, the computer goes through an interrupt cycle. If R=0, the computer goes through an interrupt cycle. If the instruction is memory reference type then computer checkes if there is indirect address and then continue to execute the decoded instruction.

Description of operation of computer using register transfer statement:

Instead of using a flowchart, we can describe the operation of the computer with a list of register transfer statements. This is done by accumulating all the control functions and Microoperations in one table as:

Fetch	$R'T_0$:	$AR \leftarrow PC$
	$R'T_1$:	$IR \leftarrow M[AR], PC \leftarrow PC+1$
Decode	$R'T_2$:	$D_0 \ldots D_7 \leftarrow$ Decode $IR(12-14)$, $AR \leftarrow IR(0-11), I \leftarrow I(15)$.
Indirect	$D_7'IT_3$	$AR \leftarrow M[AR]$
interrupt:	$T_0'T_1'T_2'$ IEN (FGI + FGO) :	$R \leftarrow 1$
	RT_0 :	$AR \leftarrow 0, TR \leftarrow PC$
	RT_1:	$M[AR] \leftarrow TR, PC \leftarrow 0$
	RT_2:	$PC \leftarrow PC+1, IEN \leftarrow 0, R \leftarrow 0, SC \leftarrow 0$
Memory reference:		
AND	D_0T_4:	$DR \leftarrow M[AR]$
	D_0T_5:	$AC \leftarrow AC \wedge DR, SC \leftarrow 0$

ADD	D_1T_4:	$DR \leftarrow M[AR]$
	D_1T_5:	$AC \leftarrow AC + DR$, $E \leftarrow C_{out}$ $SC \leftarrow 0$
LDA	D_2T_4:	$DR \leftarrow M[AR]$
	D_2T_5:	$AC \leftarrow DR$, $SC \leftarrow 0$
STA	D_3T_4:	$M[AR] \leftarrow AC$, $SC \leftarrow 0$
BUN	D_4T_4:	$PC \leftarrow AR$, $SC \leftarrow 0$
BSA	D_5T_4:	$M[AR] \leftarrow PC$, $AR \leftarrow AR + 1$
	D_5T_5:	$PC \leftarrow AR$, $SC \leftarrow 0$
ISZ	D_6T_4:	$DR \leftarrow M[AR]$
	D_6T_5:	$DR \leftarrow DR + 1$
	D_6T_6:	$M[AR] \leftarrow DR$ if $(DR = 0)$ then, $(PC \leftarrow PC + 1)$ $SC \leftarrow 0$.

Register reference:

$D_7I'T_3 = r$ (common to all register reference instruction)

$IR(i) = B_i$ $(i = 0, 1, 2, \ldots 11)$

CLA	rB_{11}:	$AC \leftarrow 0$
CLE	rB_{10}:	$E \leftarrow 0$
CMA	rB_9:	$AC \leftarrow \overline{AC}$
CME	rB_8:	$E \leftarrow \overline{E}$
CIR	rB_7:	$AC \leftarrow shr\ AC$, $AC(15) \leftarrow E$, $E \leftarrow AC(0)$
CIL	rB_6:	$AC \leftarrow shl\ AC$, $AC(0) \leftarrow E$, $E \leftarrow AC(15)$
INC	rB_5:	$AC \leftarrow AC + 1$
SPA	rB_4:	if $(AC(15) = 0)$ then $(PC \leftarrow PC + 1)$
SNA	rB_3:	if $(AC(15) = 1)$ then $(PC \leftarrow PC + 1)$
SZA	rB_2:	if $(AC = 0)$ then $(PC \leftarrow PC + 1)$
SZE	rB_1:	if $(E = 0)$ then $(PC \leftarrow PC + 1)$
HLT	rB_0:	$S \leftarrow 0$

Input-output:

$D_7 I T_3 = P$ (common to all input-output instruction)

$IR(i) = B_i$ ($i = 6, 7, 8, \ldots, 11$)

INP	PB_{11}:	$AC(0\text{-}7) \leftarrow INPR$, $FGI \leftarrow 0$
OUT	PB_{10}:	$OUTR \leftarrow AC(0\text{-}7)$, $FGO \leftarrow 0$
SKI	PB_9:	if ($FGI = 1$) then ($PC \leftarrow PC + 1$)
SKO	PB_8:	if ($FGO = 1$) then ($PC \leftarrow PC + 1$)
ION	PB_7:	$IEN \leftarrow 1$
IOF	PB_6:	$IEN \leftarrow 0$

Basic Computer design and accumulator logic:

Basic Computer design:-

The basic computer consists of following hardware components:

1. A memory unit with 4096 words of 16 bits each.
2. Nine register AR, PC, DR, AC, IR, TR, OUTR, INPR and SC
3. Seven flipflops: I, S, E, R, IEN, FGI, & FGO
4. Two decoders: a 3x8 operational decoder and 4x16 timing decoder.
5. A 16 bit common bus.
6. Control logic gates.
7. Adder and logic circuit connected to the input of AC.

The memory unit is a standard component that can be obtained readily from commercial source. The register have different function and are similar to integrated circuit. The flip flop are either D or Jk type. The common bus system is

constructed with sixteen 8x1 multiplexer.

Control logic gates:

The input to the control logic gates comes from the two decoders, the flipflop and bit 0 through 11 of IR. The other input to the control logic are: AC bits 0 through 15 to check if AC=0 and to detect the sign bit, DR bits 0 through 15 to check if DR=0 and values of seven flip flop.

The output of the control logic circuit are:

1. Signal to control the inputs of nine registers.
2. Signal to control the read and write input of memory.
3. Signal to set, clear or complement the flip flops.
4. Two decoders: a 3x8 operational decoder and 4x16 timing decoder.
5. A 16 bit common bus.
6. Control logic gates.
7. Adders and logic circuit connected to the input of AC.

4. Signals for S_2, S_1, S_0 to select register for the bus.
5. Signals to control the AC adder and logic circuit.

Control of register and memory:-

The control input of the register are LD (load), INR (increment) and CLR (clear).

Control of AR register:-

To derive the gate structure associated with control input of AR, we find all the register transfer statement that change the content of AR:

$R'T_0 : AR \leftarrow PC$

$R'T_2 : AR \leftarrow IR(0-11)$

$D_7'IT_3 : AR \leftarrow M[AR]$

$RT_0 : AR \leftarrow 0$

$D_5T_4 : AR \leftarrow AR + 1$

Here first three statement load the AR from a register or memory. The fourth statement clears AR to 0 and last statement increments AR by 1.

Now,

The control functions can be combined into three boolean expression as follows:

$$LD(AR) = R'T_0 + R'T_2 + D_7'IT_3$$

$$CLR(AR) = RT_0$$

$$INR(AR) = D_5T_4$$

Hence, the control logic gates associated with AR is shown in figure below:

P.T.O

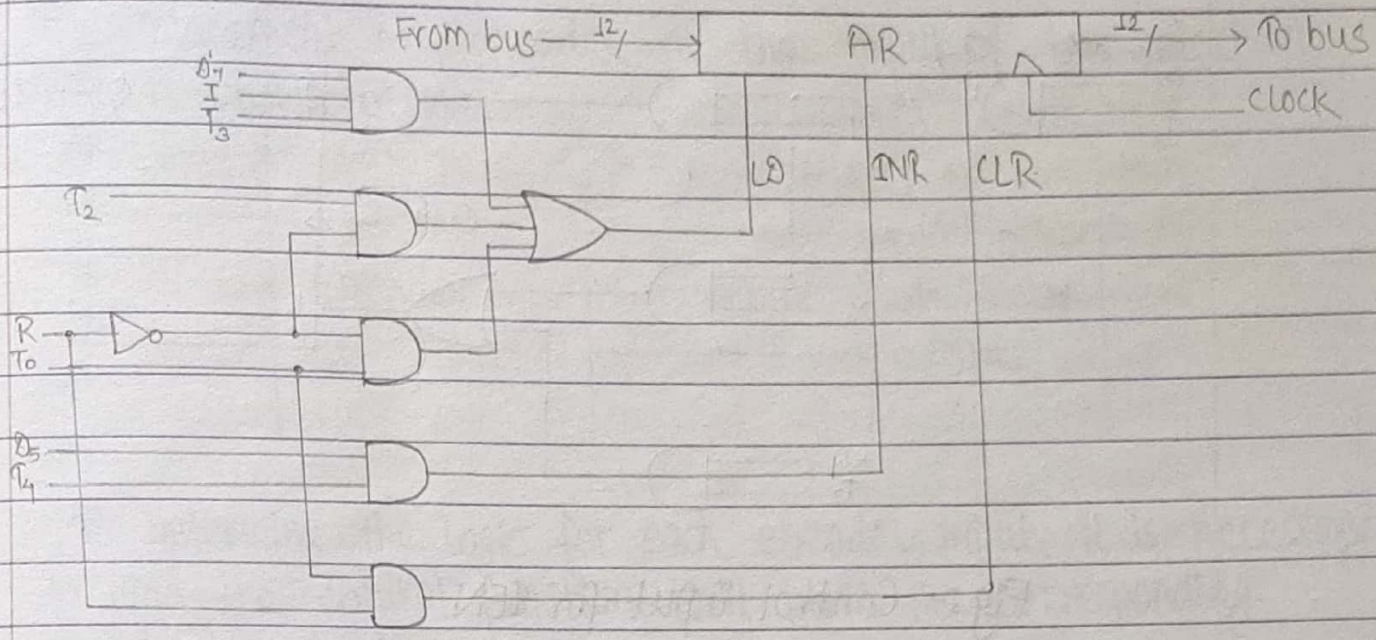


Fig:- Control gates associated with AR

In the similar fashion, we can derive the control gates for the other registers as well as logic needed to control read and write input of memory.

Control of single flip flop:

The control gates for the seven flip flop can be determined in a similar manner. For example; to construct the control logic gates for IEN flipflop we may find the statement that change the IEN.

$PB_7 : IEN \leftarrow 1$

$PB_6 : IEN \leftarrow 0$

$RT_2 : IEN \leftarrow 0$

where, $P = D_7 I T_3$ and B_7 and B_7 are bits 7 and 6 of IR, respectively. If we use the J-K flip flop for IEN, the control logic gate is shown in the figure below:

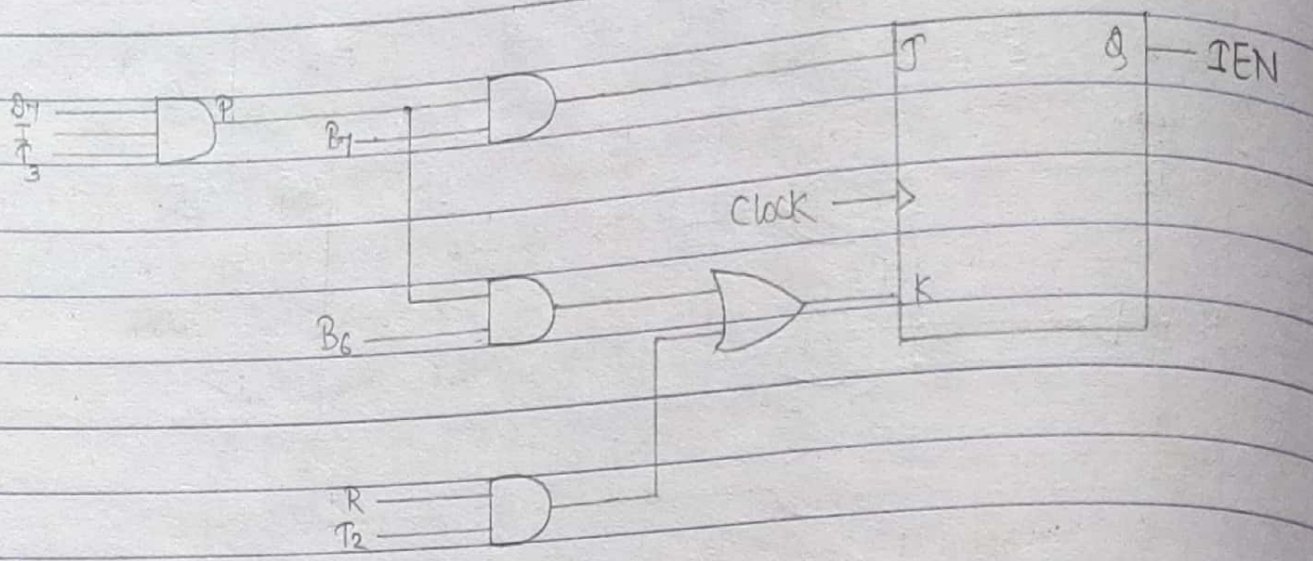


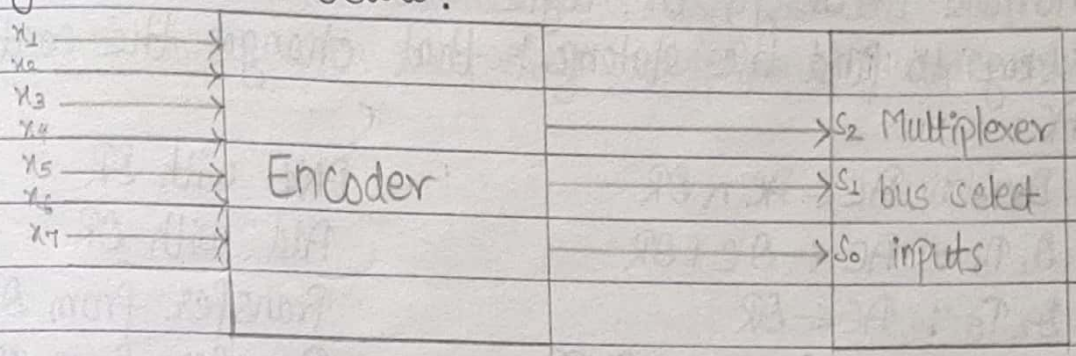
Fig:- Control input for IEN

Control of Common bus:-

The 16-bit common bus is controlled by the selection inputs S_2, S_1 and S_0. Binary number for S_2, S_1, S_0 is associated with a boolean variable x_1 through x_7, which must be active inorder to select the register or memory for the bus. This is described by the truth table given below:

x_1	x_2	x_3	x_4	x_5	x_6	x_7	S_2	S_1	S_0	Register selected for bus.
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

The placement of encoder at the input of bus selection logic is shown below:



To determine the logic for each encoder input; it is necessary to find the control function that place the corresponding register on to the bus.

For example:- to find the logic that makes $x_1 = 1$, we scan all register transfer statement that use AR as source.

$$D_4T_4 : PC \leftarrow AR$$
$$D_5T_5 : PC \leftarrow AR$$

Therefore, boolean function for X_1 is :

$$x_1 = D_4T_4 + D_5T_5$$

Design of accumulator logic :-

The circuit associated with the AC register are shown in the figure below:

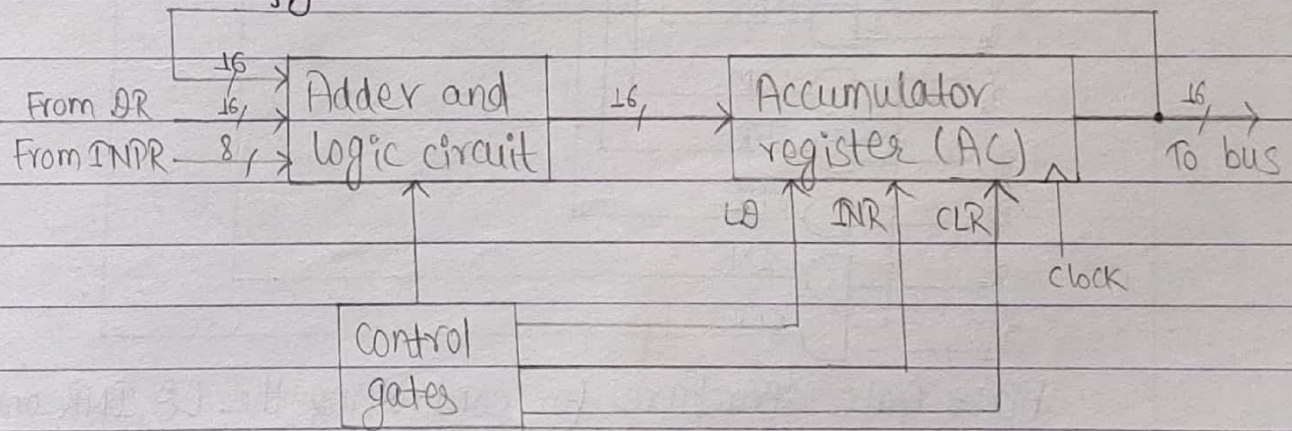


Fig:- Circuits associated with AC

Inorder to design the logic associated with AC, it is necessary to find the statement that change the content of AC.

D_0T_5 : $AC \leftarrow AC \wedge DR$	AND with DR
D_1T_5 : $AC \leftarrow AC + DR$	Add with DR
D_2T_5 : $AC \leftarrow DR$	Transfer from DR
PB_{11} : $AC(0\text{-}7) \leftarrow INPR$	Transfer from INPR
rB_9 : $AC \leftarrow \overline{AC}$	Complement
rB_7 : $AC \leftarrow shr\ AC,\ AC(15) \leftarrow E$	Shift right
rB_6 : $AC \leftarrow shl\ AC,\ AC(0) \leftarrow E$	Shift left
rB_{11} : $AC \leftarrow 0$	Clear
rB_5 : $AC \leftarrow AC + 1$	Increment

Control of AC Register:

The gate structure that controls the LD, INR and CLR inputs of AC is shown in figure below:

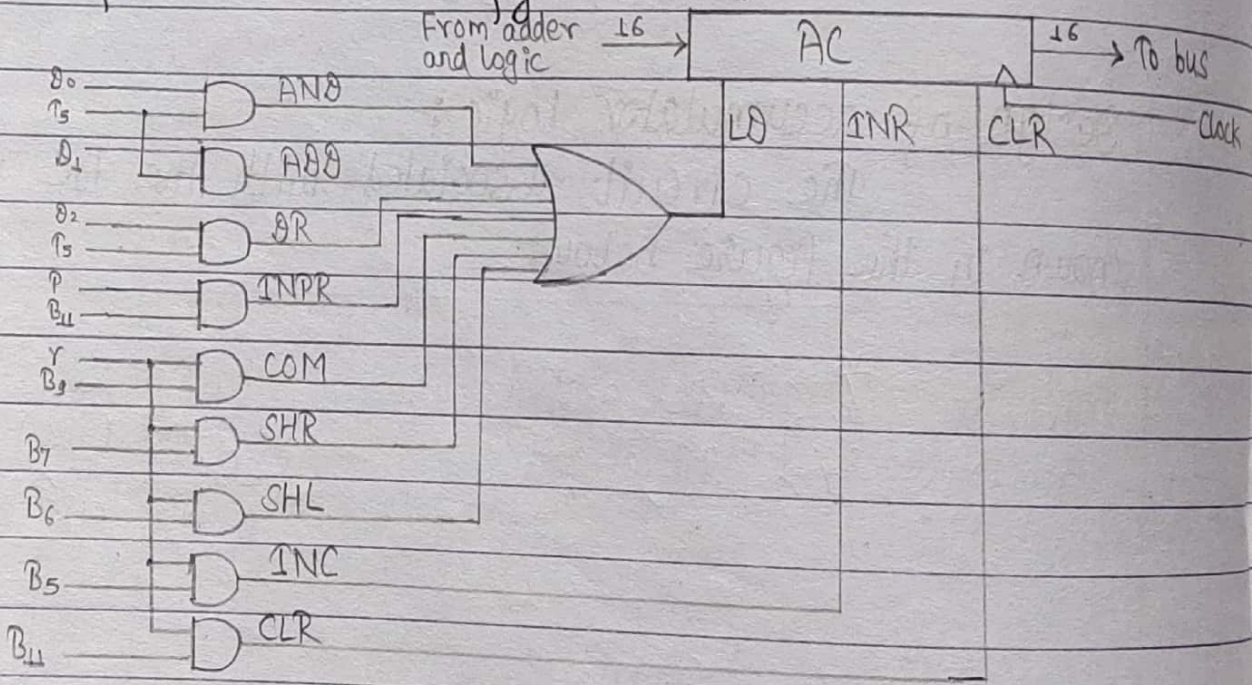


Fig:- Gate structure for controlling the LD, INR, and CLR of AC

Adder and Logic Circuit:

The adder and logic circuit can be subdivided into 16 stages with each stage corresponding to one bit of AC.

Unit 4:
Control Unit

Introduction:

Control Unit is a part of CPU whose function in a digital computer is to initiate sequences of microoperations. The number of different types of microoperations that are available in a given system is finite. The complexity of digital system is derived from the number of sequences of microoperations that are performed. Two method of implementing control unit are hardwired control and microprogrammed control.

When the control signals are generated by hardware using conventional logic design techniques, the control unit is said to be hardwired. In microprogrammed control unit, control information is stored in control memory and control memory is programmed to initiate the required sequence of microoperation.

Microprogrammed control
(Some terminology)

Control word:-

Control unit initiates a series of sequential steps of microoperations. During any given time, certain microoperations are to be initiated, while other remains idle. The control variable at any given time can be represented by a string of 0's and

It's called a control word. In short, control word contains all the control information required for one clock cycle. A control unit whose binary control variables are stored in memory is called a microprogrammed control unit.

Control Memory:-

The control memory is a storage in the microprogrammed control unit to store microprogram. Each word in control memory contains microinstruction. The microinstruction specify one or more microoperations for the system. The sequence of microinstruction constitute microprogram. The control memory can be ROM whose content of the word are fixed and cannot be altered by simple programming or writeable memory that permits the dynamic microprogramming.

Microprogram:-

The program stored in control memory that generates all control signal required to execute the instruction set correctly. The microprogram consists of microinstruction that specify various internal control signals for execution of microoperation.

Microinstruction:-

The microinstruction contains control word and sequencing word. The control word initiate the microoperation and sequencing word contains the information needed to decide next microinstruction address.

Microprogrammed control organization :-

A computer that employ microprogrammed control unit will have two separate memories, main memory and control memory. The main memory is available to the user for storing program. The users program in main memory consist of machine instruction and data.

In contrast, control memory hold a fixed microprogram that cannot be altered by the ocassional user. The microprogram in control memory consist of microinstruction that specify control signal for microoperation. Each machine instruction initiates series of microinstruction that generates the microoperation to fetch instruction, evaluate the effective address execute the operation specified instruction and return control to fetch phase to repeat the cycle for next instruction.

The general configuration of a microprogrammed control unit is shown in ~~figure~~ block diagram below:

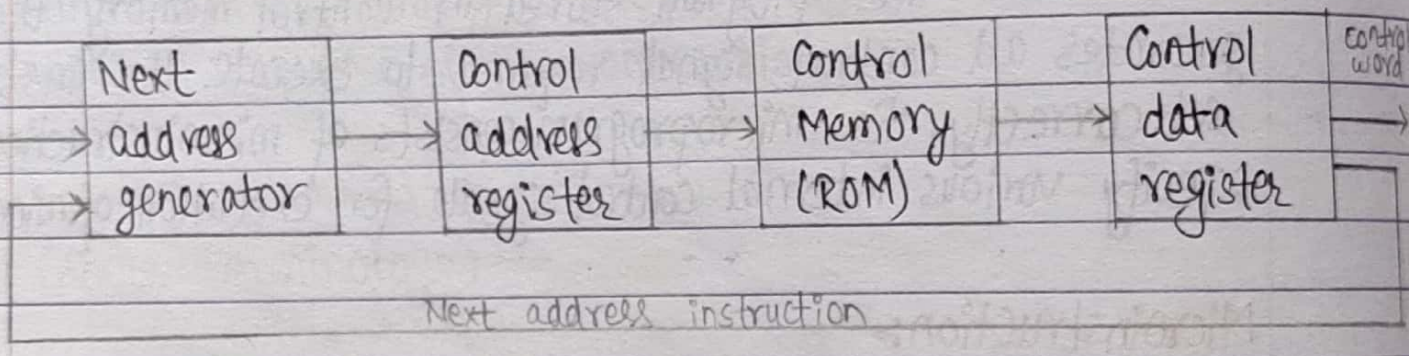


Fig :- Microprogrammed control organization

- Here we assumed control memory is to be a ROM on which all control information is parmanently stored.
- The control address register specify the address of microinstruction.
- The control data register hold microinstruction read from memory. The microinstrution contains control word that specify one or more microperation and sequence word that specify the address sequence for memory.
- While the microoperation are being executed, the next address is computed in the next address generator circuit and transfer it into control address register to read the next microinstruction. The next address generator is sometimes called sequencer.
- Typical functions of microprogram sequencer are incrementing control address register by one, loading into control address register an address from control memory, transferring an external address, or loading an initial address to start the control operation.

Address sequencer:-

Process of finding address of next instruction to be executed is called address sequencing. Each computer instruction has its own microprogram routine in control memory to generate the microoperation that execute the instruction. The hardware that controls the address sequencing of control memory (i.e. address sequencer) must have capabilities of finding address of next instruction in following situation:

- In-line sequencing
- Unconditional branch
- Conditional branch
- Sub-routine call and return
- Looping
- Mapping from instruction opcode to address in control memory.

The following figure shows block diagram of control memory and associated hardware for selecting next microinstruction address.

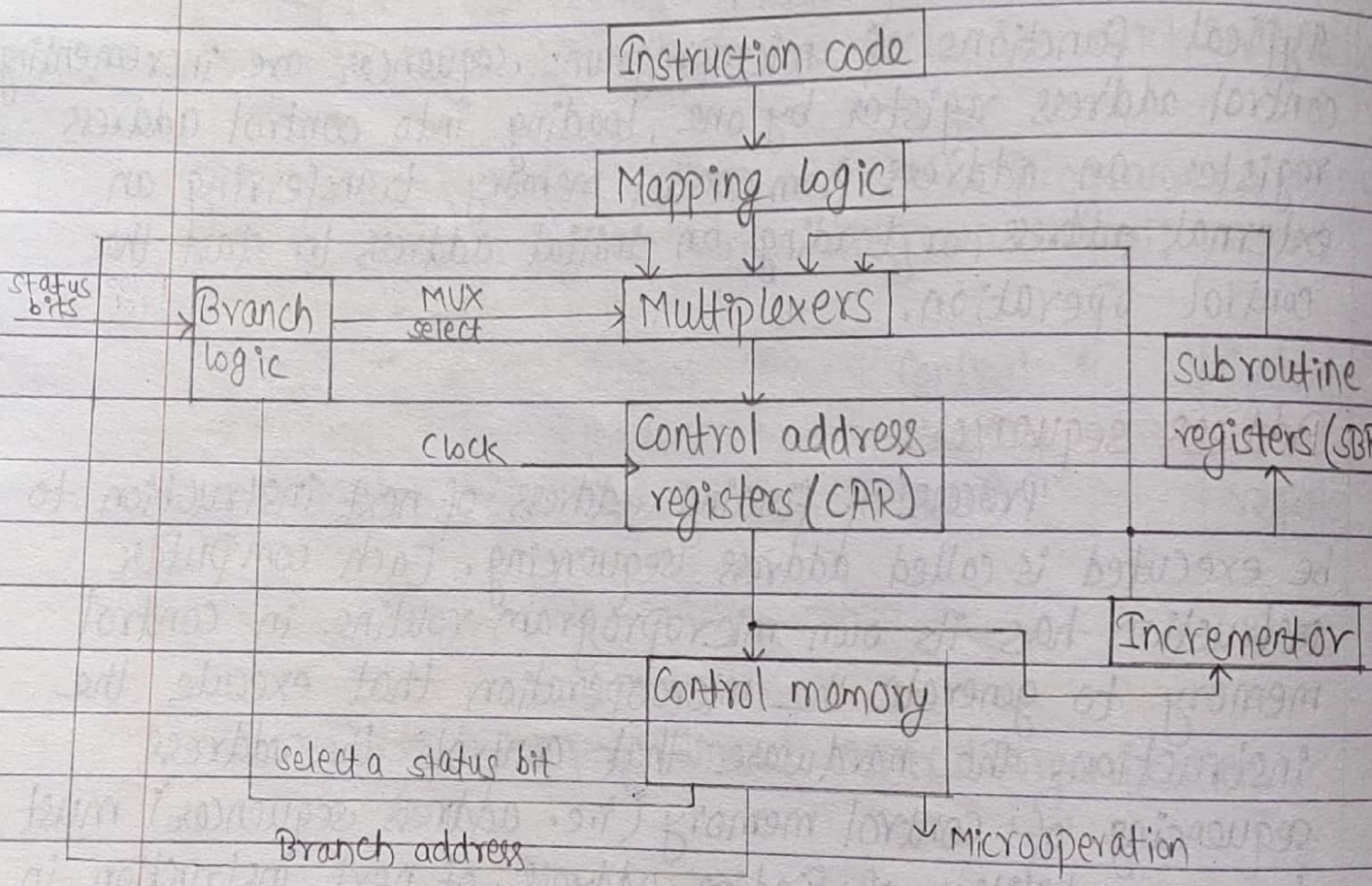


Fig:- Selection of address for control memory

The diagram shows four different paths from which the control address register (CAR) receives the address. The incrementor increments the content of the control address register by one, to select the next microinstruction in sequence. In case of the branching, branch address is specified in one of the field of the microinstruction. An external address is transferred into memory via mapping logic circuit. The return address for sub-routine is stored in a special register SBR which is used when returning from called subroutine.

Conditional branching:-

The branch logic provides decision making capabilities in control unit. The status bit, together with the field in microinstruction that specifies a branch address, controls the conditional decision generated in branch logic.

The simplest way of implementing branch logic is to test the specified condition and branch to the indicated address if the condition is met, otherwise address register is incremented. The condition are tested for O (overflow), N(Negative) Z(zero), C (carry), etc.

Unconditional branch:-

An unconditional branch microinstruction can be implemented by loading branch address from control memory into the control address register. This can be accomplished by fixing the value of one status bit at input

of the multiplexer, so it is always 1 and branching is done.

Mapping of instruction:-

For each operation code, there exists a microprogram routine in control memory that executes the instruction. Mapping is a rule that transform opcode of instruction into a address of microinstruction which is the starting microinstruction of its subroutine in control memory. One simple mapping process that converts the 4-bit operation code to a 7-bit address for control memory is shown in following figure:

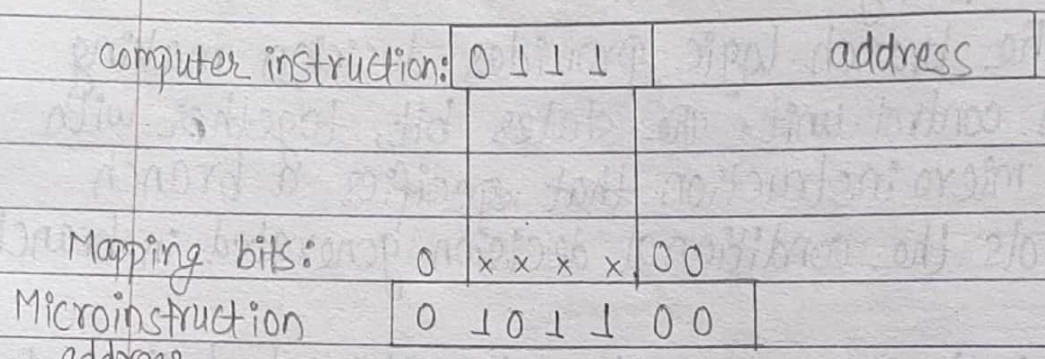


Fig:- Mapping from instruction code to microinstruction address

This mapping consists of placing 0 in the most significant bit of the address, transferring the four operation code bits and clearing the two least significant bits of the control address register.

One can extend this concept to more general mapping rule by using a ROM to specify the mapping function.

Subroutines :-

Subroutines are programs that are used by other routine to accomplish a particular task. A subroutine can be called from any point within the main body of the microprogram, frequently, many microprogram contain identical sections of code. Microinstruction can be saved by employing subroutines that use common sections of microcode.

For example; the sequence of microoperation needed to generate the effective address of the operand for an instruction is common to all memory reference instructions. This sequence could be a sub-routine that is called from within many other routines to execute effective address computation.

Microprogram that use subroutine must have a provision for storing the return address during the subroutine call and restoring the address during subroutine return. The subroutine register is used for this purpose.

Microprogram (example) :-

Microprogram consist of microinstruction. It is a program stored in control memory that generates all the control signals required to execute the instruction set correctly.

Once the configuration of a computer and its microprogrammed control unit is established, the designer's task is to generate the microcode for the control memory. This code generation is called microprogramming and is similar

to conventional machine language programming. To understand this process assume a simple digital computer similar but not identical to basic computer introduced in chapter 3. The block diagram of computer is shown in figure below:

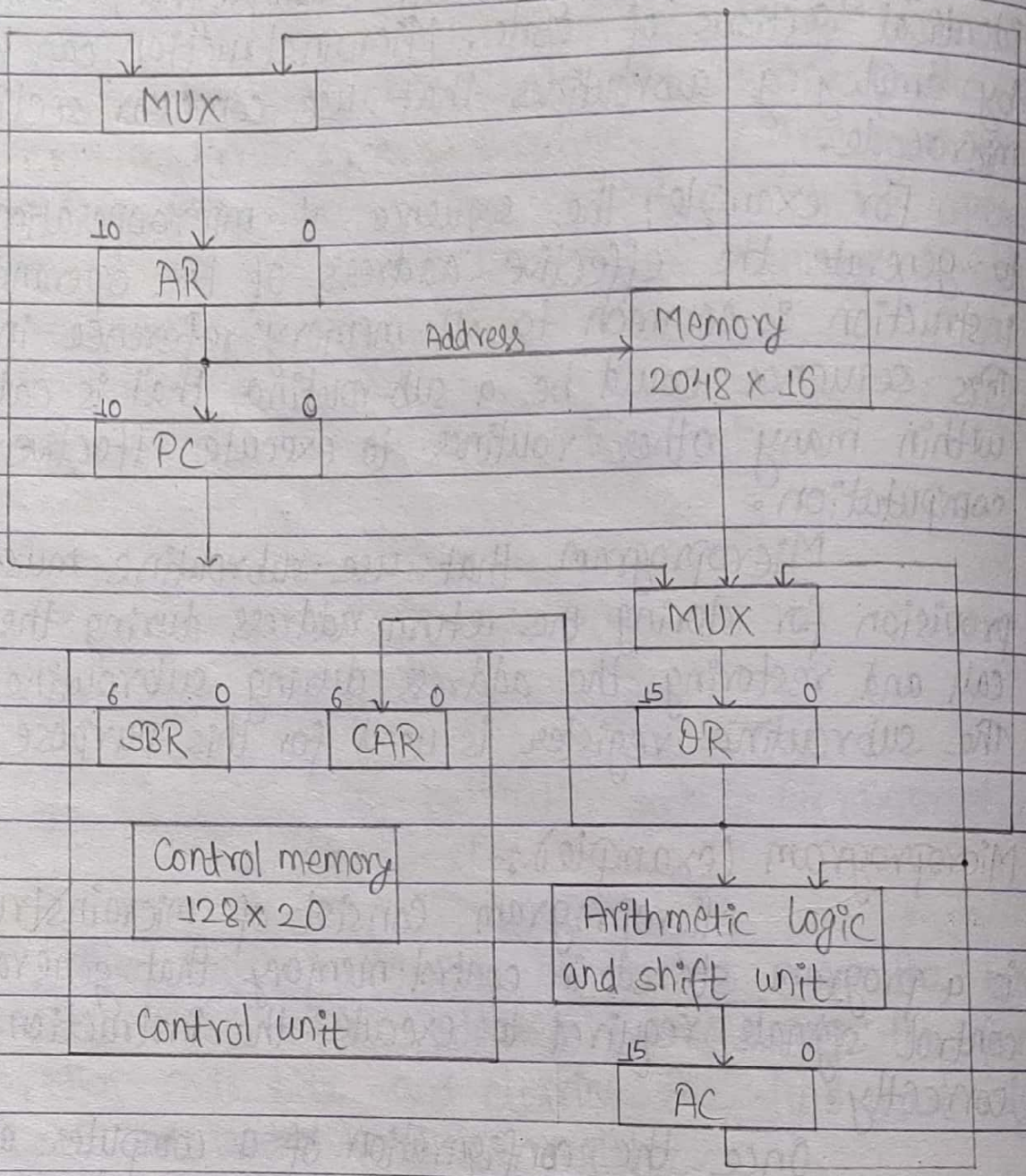
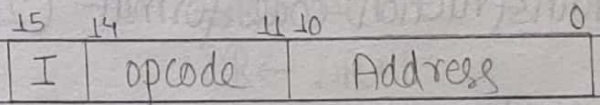


Fig:- Computer hardware configuration

It consists of two memory unit, a main memory for storing instructions and data and control memory for storing the microprogram. Four registers are associated with processor unit and two with control unit. The control memory and its registers are organized as microprogrammed control unit.

The transfer of information among the registers in the processors is done through multiplexer rather than a common bus. DR can receive information from AC, PC or memory. AR can receive information from PC or DR. PC can receive information only from AR. Arithmetic logic and shift unit perform microoperations with data from AC and DR and places the result in AC.

The computer instruction format of this computer is shown in fig(a) below and fig(b) lists four of the 16 possible memory reference instructions.



fig(a): Instruction format

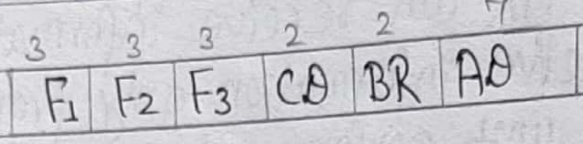
Symbol	opcode	Description
ADD	0000	AC ← AC + M[EA]
BRANCH	0001	if (AC < 0) then (PC ← EA)
STORE	0010	M[EA] ← AC
EXCHANGE	0011	AC ← M[EA], M[EA] ← AC.

fig(b): Four computer instructions

Note: Now proceed to microprogram each instruction. Inorder to not complicate the microprogramming example, only four instruction are considered here.

Microinstruction format:-

The microinstruction format for the control memory is of 20 bits are divided into four functional part as shown in figure below:



where; F_1, F_2, F_3 : Microoperation field
CD : Condition for branching
BR : Branch field
AD : address field

Fig:- Microinstruction code format (20 bit)

The three fields F_1, F_2, F_3 specify microoperation for the computer. CD field selects status bit conditions. The BR field specifies the type or branch to be used. The AD field contains a branch address.

Microoperation field:-

The microoperation field is subdivided into three fields of three bit each. The three bits in each field are encoded to specify seven distinct microoperation

as listed in table below:

F_1	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC + DR$	ADD
010	$AC \leftarrow 0$	CLRAC
011	$AC \leftarrow AC + 1$	INCAC
100	$AC \leftarrow DR$	DRTAC
101	$AR \leftarrow DR(0-10)$	DRTAR
110	$AR \leftarrow PC$	PCTAR
111	$M[AR] \leftarrow DR$	WRITE

F_2	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC - DR$	SUB
010	$AC \leftarrow AC \vee DR$	OR
011	$AC \leftarrow AC \wedge DR$	AND
100	$DR \leftarrow M[AR]$	READ
101	$DR \leftarrow AC$	ACTDR
110	$DR \leftarrow DR + 1$	INCDR
111	$DR(0-10) \leftarrow PC$	PCTDR

F_3	Microoperation	Symbol
000	None	NOP
001	$AC \leftarrow AC \oplus DR$	XOR
010	$AC \leftarrow \overline{AC}$	COM
011	$AC \leftarrow shl\ AC$	SHL
100	$AC \leftarrow shr\ AC$	SHR
101	$PC \leftarrow PC + 1$	INCPC
110	$PC \leftarrow AR$	ARTPC
111	Reserved	

Condition field :-

The condition field (CD) consists of two bits which are encoded to specify four status bit condition as listed in table below:

CD	Condition	Symbol	Comments
00	Always = 1	U	Unconditional branch
01	DR(15)	I	Indirect address bit
10	AC(15)	S	Sign bit of AC
11	AC=0	Z	Zero value in AC

Branch field :-

The BR (Branch) field consist of two bits. It is used in conjunction with the address field AD, to choose the address of the next microinstruction.

BR	Symbol	Function
00	JMP	CAR ← AD if condition = 1 CAR ← CAR+1 if condition=0
01	CALL	CAR ← AD, SBR ← CAR+1 if condition=1 CAR ← CAR+1 if condition=0
10	RET	CAR ← SBR (Return from subroutine)
11	MAP	CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

Symbolic Microinstruction :-

The symbols can be used to specify microinstruction in symbolic form called symbolic microinstruction. A symbolic microprogram can be translated into its binary equivalent by means of assembler. Each symbolic microinstruction is divided into five fields: label, microoperations, CD, BR and AD. The field specify following information:

i) Label :-

The label field may be empty or it may specify a symbolic address. A label is terminated with colon (:).

ii) Microoperation :-

The microoperations field consist of one, two or three symbols, separated by commas. There may be no more than one symbol from each F field.

iii) CD :-

The CD field has one of the letters U, I, S, or Z.

iv) BR :-

The BR field contains one of the four symbol JMP, CALL, RET, MAP.

v) AD :-

The AD field specifies a value for the address field of the microinstruction in one of the three possible ways;

(a. symbolic address, b. NEXT, c. left empty in case of RET and MAP).

Symbolic Microprogram :-

The microprogram written in symbolic form are called symbolic microprogram. Symbolic microinstruction are used to carry out these symbolic microprogram. Symbolic microprogram are converted into equivalent binary microprogram by means of assembler.

Example:

Microoperation needed for fetch routine are:

AR ← PC

DR ← M[AR], PC ← PC + 1

AR ← DR(0-10), CAR(2-5) ← DR(11-14), CAR(0,1,6) ← 0

The fetch routine needs three microinstruction which are placed in memory 64, 65, 66 because previous 64 words are to be occupied by the routine for the 16 instruction. The symbolic microprogram for fetch routine as follows:

```
        ORG 64
FETCH:  PCTAR      U  JMP  NEXT
        READ, INCPC U  JMP  NEXT
        DRTAR      U  MAP
```

Partial Symbolic Microprogram :-

The following microprogram shows the symbolic microprogram for fetch routine and routine that executes four computer instruction.

Label:	Microoperation	CD	BR	AD
	ORG 0			
ADD:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ADD	U	JMP	FETCH
	ORG 4			
BRANCH:	NOP	S	JMP	OVER
	NOP	U	JMP	FETCH
OVER:	NOP	I	CALL	INDRCT
	ARTPC	U	JMP	FETCH
	ORG 8			
STORE:	NOP	I	CALL	INDRCT
	ACTDR	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 12			
EXCHANGE:	NOP	I	CALL	INDRCT
	READ	U	JMP	NEXT
	ACTDR, DRTAC	U	JMP	NEXT
	WRITE	U	JMP	FETCH
	ORG 64			
FETCH:	PCTAR	U	JMP	NEXT
	READ, INCPC	U	JMP	NEXT
	DRTAR	U	MAP	

Label:	Microoperations	CD	BR	AD
INDRCT:	READ	U	JMP	NEXT
	ARTAR	U	RET	

Fig:-Symbolic Partial Microprogram.

The execution of the ADD instruction is carried out by the microinstruction at address 1 and 2. The first microinstruction read the operand from memory into DR. The second microinstruction perform an add microoperation with the content of DR and AC and then jump back to the beginning of the fetch routine.

Binary Microprogram:-

The symbolic microprogram is a convenient form for writing microprograms in a way that people can read and understand but this is not a way that microprogram is stored in memory. The symbolic microprogram must be translated to binary by means of assembler program or by the user. Thus microprogram obtained by converting symbolic microprogram into binary equivalent form is called binary microprogram.

For example; the binary microprogram for the fetch routine in the previous symbolic microprogram is:

Micro Routine	address Decimal	binary	F_1	F_2	F_3	CD	BR	AD
FETCH:	64	1000000	110	000	000	00	00	1000001
	65	1000001	100	100	101	00	00	1000010
	66	1000010	000	000	000	00	11	0000000

Field decoding (decoding of F-field):-

The nine bit of microoperation field of microinstruction are divided into three subfields of three bit each. The control memory output of each subfield must be decoded to provide distinct microoperations. The output of decoders are connected to appropriate inputs in the processor unit.

Since there are three microoperation field we need three decoders, The figure below shows three decoders and some of the connections that must be made from their outputs.

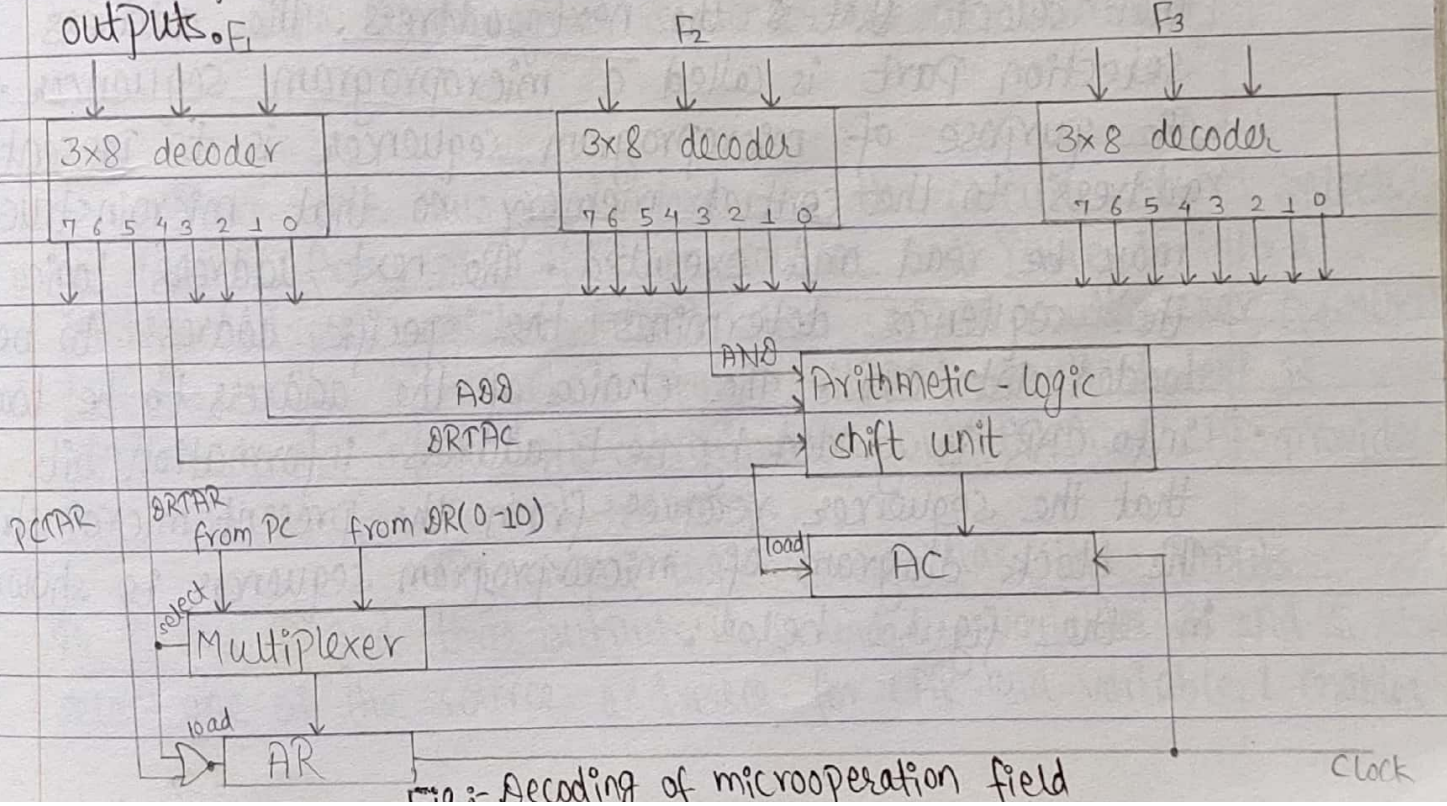


Fig :- Decoding of microoperation field

Here, Each of the three fields of microinstruction presently available in the output of control memory are decoded with a 3x8 decoder to provide eight outputs. Each of these outputs must be connected to the proper circuit to initiate the corresponding microoperation.

For example; when $F_1 = 101(5)$, the next clockpulse transition transfers the content of $BR(0-10)$ to AR. Similarly when $F_1 = 110(6)$ there is a transfer from PC to AR. Figure above shows that output 5 and 6 of decoder are connected to load input of AR. The multiplexer select the information BR when output 5 is active and from PC when output 5 is inactive.

10 marks
2067

Microprogram sequencer:-

The basic components of microprogrammed control unit are the control memory and the circuit that select the next address. The address selection part is called a microprogram sequencer. The purpose of microprogram sequencer is to present an address to the control memory so that microinstruction may be read and executed. The next address logic of the sequencer determines the specific address to be loaded into CAR. The choice of the address to be loaded into CAR is guided by next address information bits that the sequencer receives from the present microinstruction. The block diagram of microprogram sequencer is shown in the figure below:

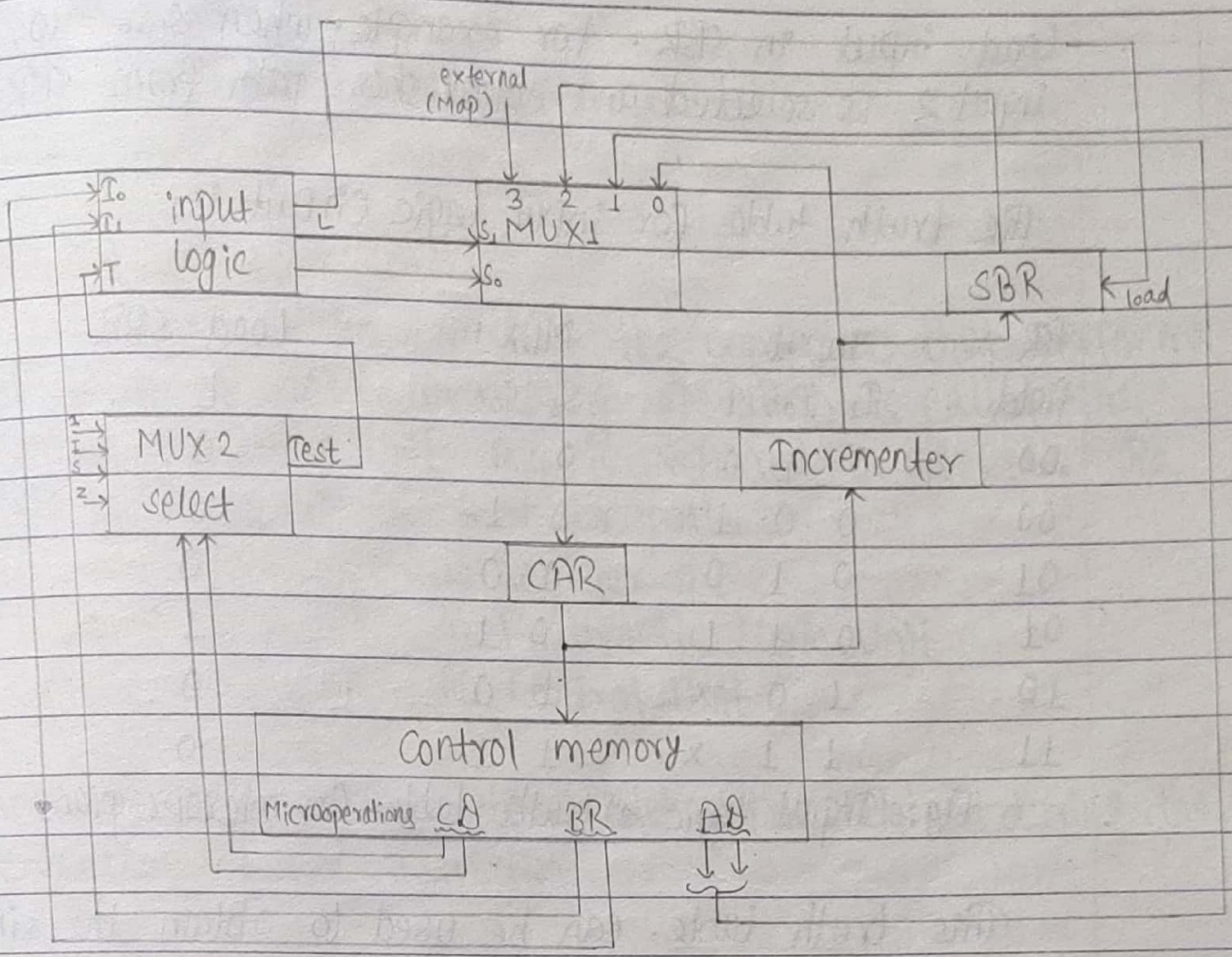


Fig:- Microprogram sequencer for control memory

There are two multiplexers in the circuit. MUX1 selects an address from one of four sources and routes it into a control address register (CAR). The second multiplexer (i.e. MUX2) test the value of selected status bit and result of test is applied to an input logic circuit. The output from CAR provides the address for control memory.

The input logic circuit has three inputs I_0, I_1 and T and three outputs S_0, S_1 and L. Variables S_0 and S_1 select one of the source addresses for CAR and variable L enables

Load input in SBR. For example; when $S_1S_0 = 10$, MUX input 2 is selected and establishes path from SBR to CAR.

The truth table for input logic circuit is:

BR field	input I_1	I_0	T	MUX 1 S_1	S_0	Load SBR L
00	0	0	0	0	0	0
00	0	0	1	0	1	0
01	0	1	0	0	0	0
01	0	1	1	0	1	1
10	1	0	x	1	0	0
11	1	1	x	1	1	0

Fig:- Input logic truth table for microprogram sequencer.

This truth table can be used to obtain the simplified boolean functions for the input logic circuit.

$$S_1 = I_1$$
$$S_0 = I_1I_0 + I_1'T$$
$$L = I_1'I_0T$$

Hence input logic circuit can be constructured with three AND gates, an OR gates and an inverter.

UNIT-5
Central Processing Unit

Introduction to CPU:-

The part of the computer that performs the bulk of data processing operation is called the central processing Unit and is referred to as CPU. The CPU is made up of 3 major parts:

i) Register set
ii) Arithmetic Logic Unit
iii) Control Unit

i) **Register set:-** It stores intermediate data used during the execution of the instructions.

ii) **Arithmetic Logic Unit:-** It performs the required microoperation for executing the instruction.

iii) **Control Unit:-** It supervise the transfer of information among the registers and instruct the ALU as to which operation to performed.

The following figure shows the major component of CPU:

P.T.O

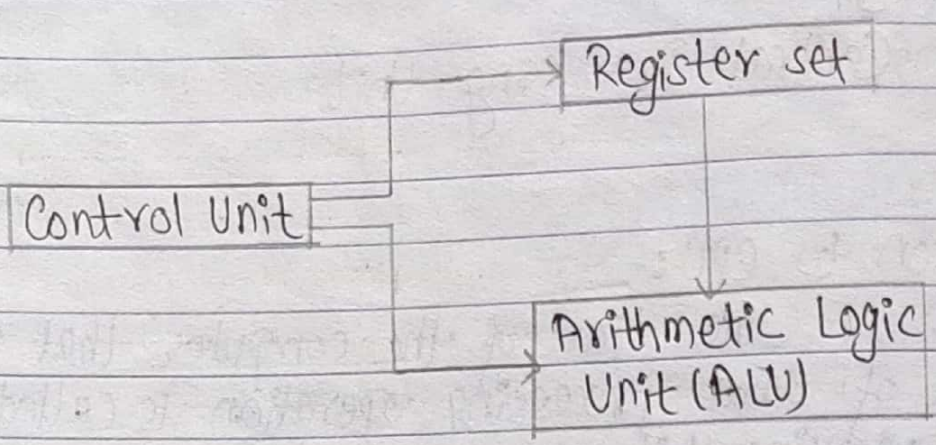


Fig:- Major component of CPU.

General Register Organization:- (Why we need computer register)?

During the instruction execution, we could store pointers, counters, return address, temporary result in memory location. Having to refer to memory location for such application is time consuming operation in a computer so for convenient and more efficient processing we need a processor register connected through a common bus system to store intermediate results.

The bus organization for seven CPU register is shown in figure below:

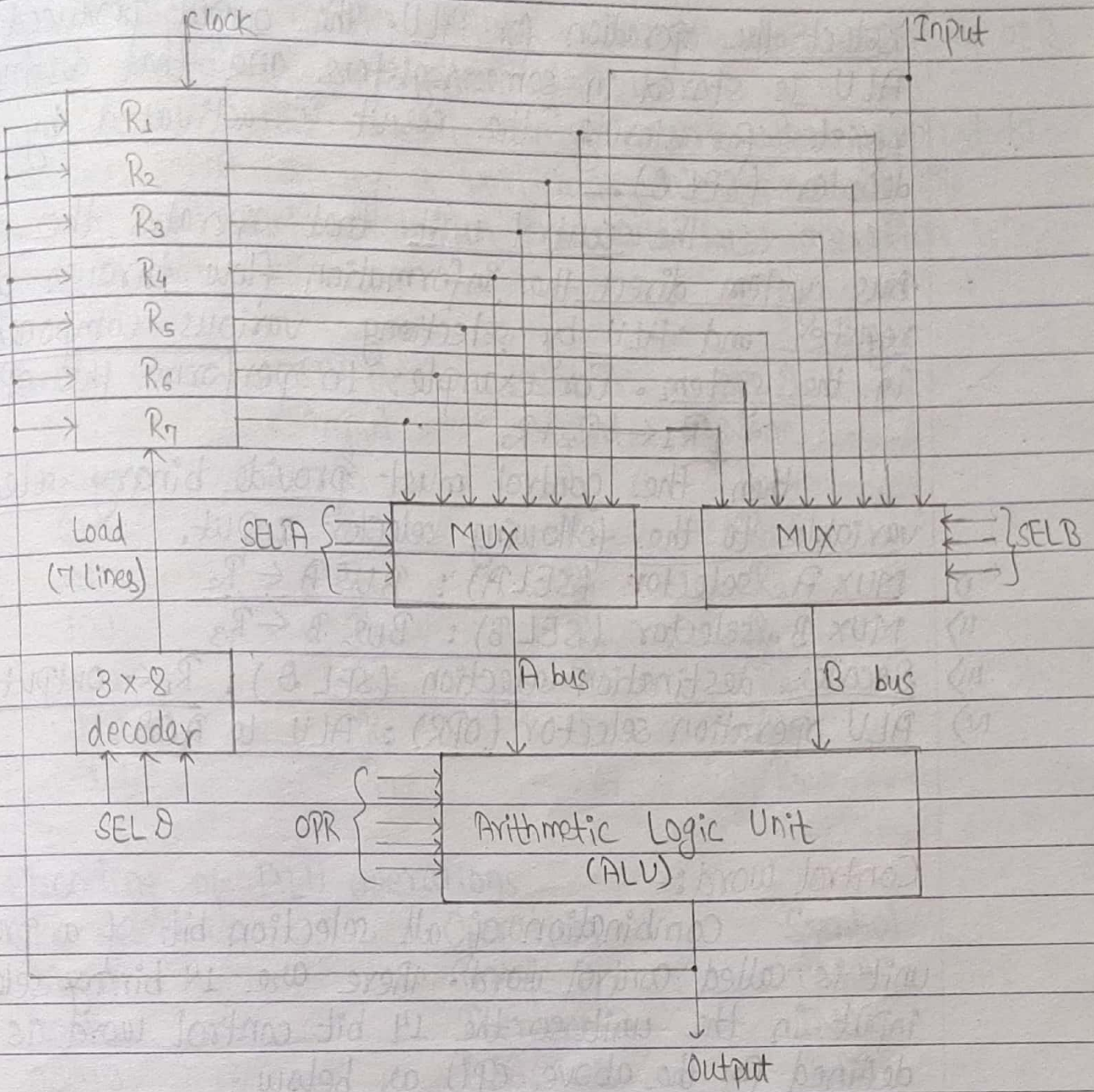


Fig:- Block diagram of register organization with bus system

Here, output of each register is connected to two multiplexer that select a register for bus A and B. The A and B buses form the input to a common arithmetic logic unit. The selector OPR connected to ALU

select the operation for ALU. The output produced by ALU is stored in some registers and that destination register for storing the result is activated by destination decoder (SELD).

The control unit that operates the CPU bus system direct the information flow through the register and ALU by selecting various component in the system. For example; to perform the operation

$R_1 \leftarrow R_2 + R_3$

then the control must provide binary selection variable to the following selector input.

i) MUX A selector (SELA): BUS A $\leftarrow R_2$
ii) MUX B selector (SELB): Bus B $\leftarrow R_3$
iii) Decoder destination selection (SELD): $R_1 \leftarrow$ output bus
iv) ALU operation selector (OPR): ALU to ADD

Control word:-

Combination of all selection bit of a processing unit is called control word. There are 14 binary selection input in the unit so the 14 bit control word is defined for the above CPU as below:

3	3	3	5
SELA	SELB	SELD	OPR

The 3 bit of SELA selects a register for the A input of the ALU. The 3 bit of SELB selects a register for B input of ALU. The 3 bit of SELD selects

a destination register using a decoder and 5 bit of OPR select one of the operation of ALU.

The 14 bit control work when applied to selection input specify a particular microoperation. The encoding of register selection field and ALU operation is given below:

Binary code	SEL A	SEL B	SEL D
000	input	input	None
001	R_1	R_1	R_1
010	R_2	R_2	R_2
011	R_3	R_3	R_3
100	R_4	R_4	R_4
101	R_5	R_5	R_5
110	R_6	R_6	R_6
111	R_7	R_7	R_7

Encoding of ALU operations

OPR select	Operation	Symbol
00000	Transfer A	TSFA
00001	Increment A	INCA
00010	ADD A+B	ADD
00101	Subtract A-B	SUB
00110	Decrement A	DECA
01000	AND A and B	AND
01010	OR A and B	OR
01100	XOR A and B	XOR
01110	Complement A	COMA
10000	shift right A	SHR A
11000	shift left A	SHL A

Examples of Microoperation:

A Control word of 14 bits is needed to specify a microoperation in the CPU. The control word for a given microoperation can be derived from the selection variables. For example; the subtract microoperation;

$R_1 \leftarrow R_2 - R_3$ then,

Field:	SELA	SELB	SELD	OPR
Symbol:	R_2	R_3	R_1	SUB
Control word:	010	011	001	00101

The control word for few other microoperation are listed in the table below:

	Symbolic Designation				
Microoperation	SELA	SELB	SELD	OPR	Control Word
$R_1 \leftarrow R_2 - R_3$	R_2	R_3	R_1	SUB	010 011 001 00101
$R_4 \leftarrow R_4 \vee R_5$	R_4	R_5	R_4	OR	100 101 100 01010
$R_6 \leftarrow R_6 + 1$	R_6	-	R_6	INCA	110 000 110 00001
$R_7 \leftarrow R_1$	R_1	-	R_7	TSFA	001 000 111 00000
Output $\leftarrow R_2$	R_2	-	None	TSFA	010 000 000 00000
Output $\leftarrow$ Input	Input	-	None	TSFA	000 000 000 00000
$R_4 \leftarrow$ shl R_4	R_4	-	R_4	SHLA	100 000 100 11000
$R_5 \leftarrow 0$	R_5	R_5	R_5	XOR	101 101 101 01100

Fig:- Examples of Microoperation for the CPU.

Stack organization :-

This is useful last-In-First-Out (LIFO) list included in the CPU of most computer. A stack is storage device that stores information in such a manner that item stored last is the first item retrieved. The stack in digital computer is a memory unit with stack pointer (SP). SP is simply the register that always point to top of stack. The two operation of stack are insertion (PUSH) and deletion (POP) of the items.

Register stack :-

The collection of finite number of register organized in LIFO manner is called register stack. The following figure shows the organization of 64 word register stack. The stack pointer, contains a binary number whose value is equal to the address of word that is currently on the top of stack.

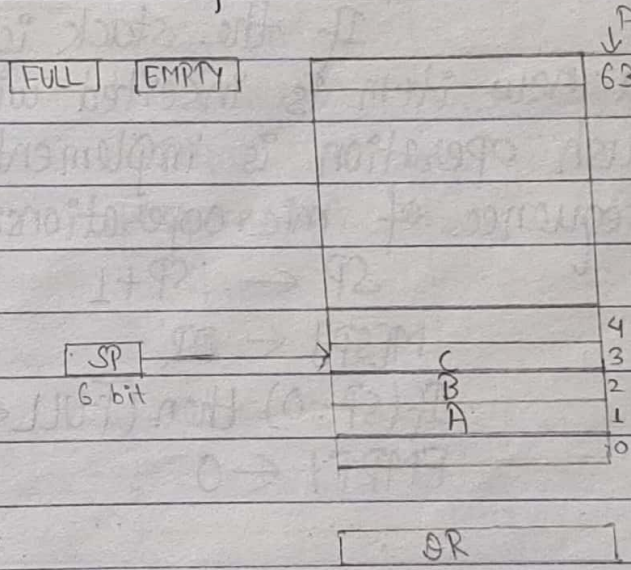


Fig :- Block diagram of a 64-word stack

Here, 3 items are placed in the stack. Item C is on the top of stack so the content of SP is now 3. To remove the top item, the stack is popped by reading the memory word at address 3 and decrementing the SP. To insert an item, the stack is pushed by incrementing SP and writing the word in the stack. Since here stack is 64 word so stack pointer contains 6 bit because $2^6 = 64$.

The one bit register FULL is set to 1 when stack is full and 1 bit register EMPTY is set to 1 when stack is empty of item. DR is the data register that holds the binary data to be written into or read out of the stack.

PUSH and POP:

Initially, SP is cleared to 0 and EMPTY is set to 1 and FULL is cleared to 0.

If the stack is not full i.e. FULL=0, a new item is inserted with Push operation. The push operation is implemented with the following sequence of microoperations:

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
EMPTY ←0	Mark the stack not empty

A new item is deleted or popped from the stack if the stack is not empty i.e. EMPTY = 0. The pop operation is implemented with the following sequence of microoperations:

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

Memory Stack :-

A stack can be implemented in a random access memory attached to CPU called memory stack. The memory stack is implemented by assigning a portion of memory to a stack operation and using processor register as a stack pointer. The following figure shows portion of computer memory partitioned into 3 segments; program, data and stack.

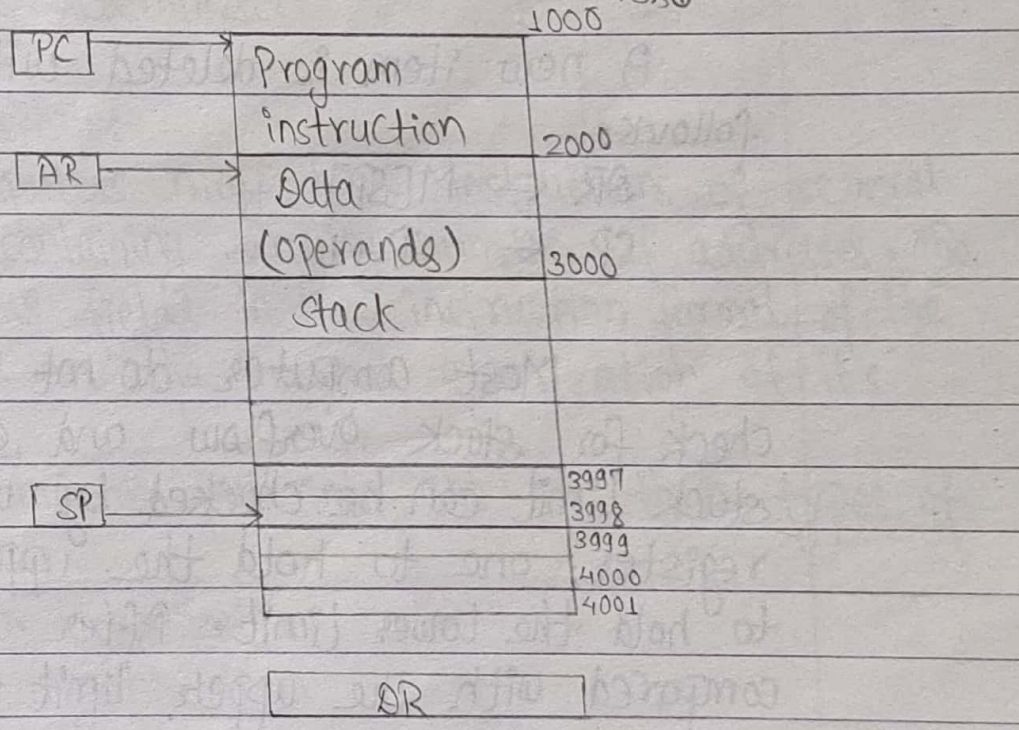


Fig :- Computer Memory with program, data and stack segment

Here, the program counter PC points at the address of next instruction in the program and PC is used during the fetch phase to read an instruction. The address register AR points data and is used during the execute phase to read an operand. The stack pointer (SP) point at the top of stack and is used to push or pop items into or from the stack.

Here, the initial value of SP is 4001 and stack grows with decreasing addresses. Thus, the first item stored in the stack is at address 4000. The second item is stored at address 3999 and last address that can be used for stack is 3000. A new item is inserted with PUSH operation as follows:-

SP ← SP - 1
M[SP] ← DR

A new item is deleted with POP operation as follows:

DR ← M[SP]
SP ← SP + 1

Most computer do not provide hardware to check for stack overflow and stack underflow. The stack limit can be checked by using two processor register; one to hold the upper limit and other to hold the lower limit. After PUSH operation, SP is compared with the upper limit register and after POP

operation, SP is compared with lower limit register.

Instruction format:-

A computer will usually have variety of instruction code formats. It is the function of control unit within the CPU to interrupt each instruction code and provide the necessary control functions needed to process the instruction.

The bit of instruction are divided into groups called fields. The most common fields found in instruction formats are:

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

Processor organization :-

Computers may have instructions of several different lengths containing varying number of addresses. The number of address fields in the instruction format of the computer depends on the internal organization of it's registers.

Most computer fall into one of three types of CPU organizations:

i) Single accumulator organization
ii) General register organization
iii) Stack organization.

1) Single accumulator organization:-

- The basic computer is good example of single accumulator organization.
- All operations are ~~provided~~ performed with an implied accumulator register.
- The instruction format of this type of ~~common~~ computer uses one address field.

For example:-

ADD X	|| AC ← AC + M[X] ||
LDA Y	|| AC ← M[Y] ||

11) General register organization :-

- This type of CPU organization is used by most modern computer.
- General register type computer employ two or three address fields in their instruction format. Each address field may specify a processor register or memory word.
- Any of the register can be used at the source or destination for computer operations.

Example:

ADD	R_1 R_2 R_3	|| $R_1 \leftarrow R_2 + R_3$ ||
ADD	R_1, R_2	|| $R_1 \leftarrow R_1 + R_2$ ||
MOV	R_1, R_2	|| $R_1 \leftarrow R_2$ ||
ADD	R_1, X	|| $R_1 \leftarrow R_1 + M[X]$ ||

Stack organization:-

- In stack organized CPU all operation are done with stack.
- Computers with stack organization would have PUSH and POP instruction which requires an address field.
- Operation type instructions do not need an address field in stack organized computer. This is because operation is performed on the two items that are on top of stack.

Example:

```
PUSH X      // TOS ← M[X]//
ADD         // Popping the two top numbers from stack
            adding the numbers and pushing the
            sum into stack.
```

Types of instruction:-

Instruction format of a computer instruction usually contains 3 fields: operation code field (opcode), address field and mode field. The number of address field in the instruction format depends on the internal organization of CPU. On the basis of no. of address field we can categorize the instruction as below:-

Examples:-

1) Three address instruction:

Computer with three address instruction formats can use each address field to specify either a processor register or memory operand. The program in assembly language that evaluates X = (A+B) * (C+D) is shown below using three address instruction.

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

The advantage of the three address format is that it result in short program. The disadvantage is that binary coded instruction require to many bits to specify three addresses.

ii) Two address instruction:-

Two address instruction are most common in commercial computer. Here again, each address field can specify either a processor register or memory word. The program to evaluate X = (A+B) * (C+D) using two address instruction is as follows:-

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

iii) One address instruction:-

One address instruction use an implied accumulator register for all data manipulation. The program to evaluate X = (A+B) * (C+D) using one address instruction is:

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 ||

iv) Zero address instruction:-

The stack organized computer use this type of instruction. The name zero address is given to this type of computer because of absence of an address field in the computational instruction. The following program shows how X = (A+B) * (C+D) will be written for a stack organized computer:

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

RISC instruction:-

The instruction set of a typical RISC processor is restricted to the use of load and store instructions when communicating between memory and CPU. All other instructions are executed within the registers of the CPU without referring to memory. The LOAD and STORE instruction have one memory and one register address and computational instruction have three address with all three specifying processor registers.

The following is the program to evaluate $X = (A+B) * (C+D)$.

LOAD	R_1, A	$R_1 \leftarrow M[A]$
LOAD	R_2, B	$R_2 \leftarrow M[B]$
LOAD	R_3, C	$R_3 \leftarrow M[C]$
LOAD	R_4, D	$R_4 \leftarrow M[D]$
ADD	R_1, R_1, R_2	$R_1 \leftarrow R_1 + R_2$
ADD	R_3, R_3, R_4	$R_3 \leftarrow R_3 + R_4$
MUL	R_1, R_1, R_3	$R_1 \leftarrow R_1 * R_3$
STORE	X, R_1	$M[X] \leftarrow R_1$

Addressing Mode:-

The way the operands are choosen during program execution is dependent on the addressing mode of the instruction. The addressing mode specifies a rule for interpreting or modifying the address field of the instruction before the operand is actually referenced.

In short, each instruction of computer specifies an operation on data. There are various ways of specifying address of data to be operated on. These different ways of specifying data are called addressing mode.

Mode field:-

In some computers the addressing mode of the instruction is specified with a distinct binary code, just like the operation code is specified. Other computers use a single binary code that designates both the operation and mode of the instruction. Instructions may be defined with variety of addressing mode and sometimes, two or more addressing modes are combined in one instruction.

An example of an instruction format with a distinct addressing mode field is shown in figure below:

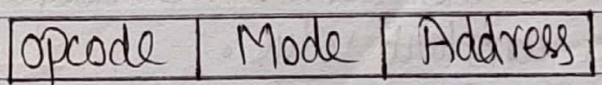


Fig:- Instruction format with mode field

- The operation code specifies the operation to be performed.

- The mode field is used to locate the operand needed for operation.
- There may or may not be an address field in the instruction.

2066 10 mark

* Types of addressing mode:-

Instruction may be defined with variety of addressing mode. Some of the addressing modes are:-

1) Implied addressing mode:-
- In this mode the operands are specified implicitly in the definition of the instruction.
- In this mode no need to specify address in the instruction.
- The instruction CLA, CMA, CLE, etc are implied mode instruction.

11) Immediate addressing mode:-
- In this mode the operand is specified in the instruction itself.
- In otherword, an immediate mode instruction has an operand field rather than an address field.
- Immediate mode instruction are useful for initializing registers to a constant value.
- It is fast to acquire an operand.
- The disadvantage is that sometimes operand needs more bits than address field.

iii) Register addressing mode:-
- In this mode, the operands are in registers that reside within the CPU so address specified in the instruction is the address of a register.
- The particular register is selected from a register field in the instruction. A k-bit field can specify any one of 2^k register.
- It requires shorter address than memory address so saving the address field in the instruction.
- Faster to acquire an operand than the memory addressing.

iv) Register indirect addressing mode:-
- In this mode, the instruction specifies a register whose content give the address of the operand in memory.
- In other word, the selected register contains the address of the operand rather than operand itself.
- The advantage of register indirect addressing instruction is that it saves instruction bits since register address is shorter than the memory address.
- Disadvantage is that slower to acquire an operand than both register addressing or memory addressing.
- In this mode effective address of operand EA = content of register.

v) Autoincrement or Autodecrement mode:-
- It is similar to the register indirect mode exept that the register is incremented or decremented after or before its value is used to access memory.

- When address stored in the register refers to a table of data in memory, it is necessary to increment or decrement the register after every access to the table. This can be achieved by using auto increment or decrement addressing mode in the instruction.

VI) Direct addressing mode:-
- In this mode the effective address is equal to the address part of the instruction.
- The operand resides in memory and its address is given directly by the address field of the instruction.
- It requires one memory reference to read the operand from given location.
- It is faster than other memory addressing mode.
- Too many bits are needed to specify the address for large physical memory space.
- In this mode EA = IR (address).

VII) Indirect addressing mode:-
- In this mode, the address field of instruction gives address of memory location that contains the address of the operand (i.e. effective address).
- Control fetches the instruction from memory and uses its address part to read the effective address.
- In this addressing mode, acquiring an operand is slow because of additional memory access.
- In this addressing mode,

$$EA = M[IR(address)]$$

VIII) Relative addressing mode :-

- In this mode, the content of the program counter is added to the address part of the instruction inorder to obtain the effective address.
- The address part of instruction is usually a signed number, which can be either positive or negative. When this number is added to the program counter, the result produces an effective address whose position in memory is relative to address of next instruction.
- In this mode,

$$EA = PC + IR\ (address)$$

- It results in a shorter address field in the instruction format since relative address can be specified with smaller number of bits.

IX) Index addressing mode :-

- In this mode, the content of an index register is added to the address part of instruction to obtain effective address.
- The index register is a special CPU register that contains an index value.
- In this mode,

$$EA = IX + IR\ (address)$$

where, IX = index register

x) Base register addressing mode:

- In this mode the content of a base register is added to the address part of instruction to obtain the effective address. i.e.

  EA = BAR + IR (address)

  where, BAR is a base register.

- This is similar to index addressing mode except that the register is now called a base register instead of index register.
- The difference between base and index register addressing mode is in the way they are used rather than in the way they are computed. An index register is assumed to hold an index number that is relative to address part of instruction but a base register is assumed to hold a base address and address field of instruction gives a displacement relative to this base address.
- The base register addressing is used in computers to facilitate the relation of program in memory.

Addressing Mode example:

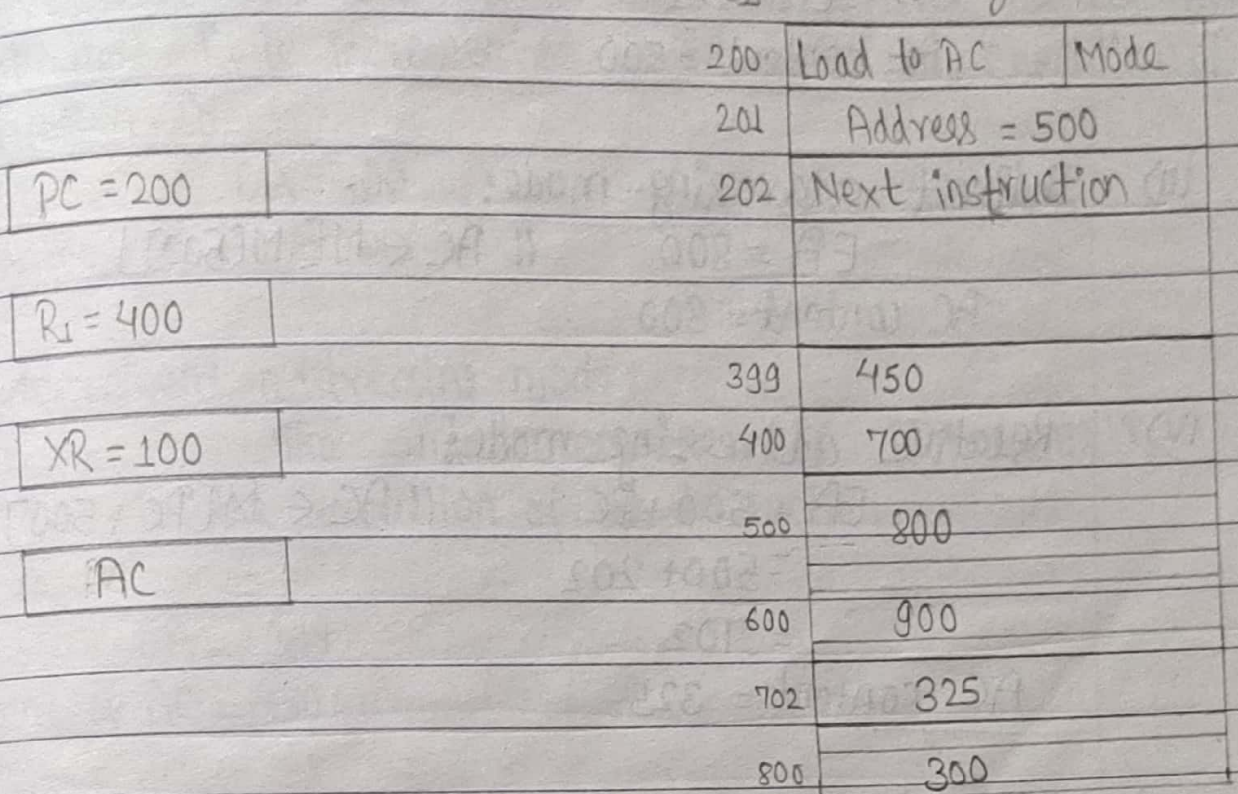


Fig:- Numerical example for addressing modes

Here we have, 2-word instruction "Load to AC" occupying addresses 200 and 201. First word specifies an operation code and mode. and second word specifies an address part (i.e. 500 here)

Mode field specify an one of the number of addressing mode. For each possible mode we calculate effective address (EA) and operand that must be loaded into AC as:-

1) Direct addressing mode:

EA = 500 || AC ← M[500]

AC content = 800

i) Immediate addressing mode:

EA = 201 // AC ← 500

AC content = 500

ii) Indirect addressing mode:

EA = 800 // AC ← M[M[500]]

AC content = 300

iv) Relative addressing mode:

EA = 500 + PC // AC ← M[PC + 500]

= 500 + 202

= 702

AC content = 325

v) Indexed addressing mode:

EA = 500 + XR // AC ← M[XR + 500]

= 500 + 100

= 600

AC content = 900

vi) Register addressing mode:

Content of AC is = 400 // AC ← R_1

vii) Register indirect addressing mode:

EA = content of R_1 = 400 // AC ← M[R_1]

Content of AC = 700

viii) Autoincrement addressing mode:

Autoincrement is same as register indirect mode except that R_1 is incremented to 401 after the execution of instruction. So,

EA = 400

Content of AC = 700

ix) Autodecrement addressing mode:

The autodecrement mode decrements R_1 to the execution of the instruction.

So,

EA = 399

Content of AC = 450

Data transfer and manipulation:

Computers provide an extensive set of instructions to give the user the flexibility to carryout various computational tasks. The actual operations available in the instruction set are not very different from one computer to another although binary encodings and symbolic name may vary. Most computer instructions can be classified into three categories on the basis of type of operation performed by instruction.

i) Data transfer instruction
ii) Data manipulation instruction
iii) Program control instruction.

1) Data transfer instructions:-

Data transfer instruction moves data from one place in computer to another without changing the data content. The most common transfers are between memory and processor register, between processor registers and input or output and between processor register themselves. Table below gives list of eight data transfer instructions used in many computers.

Name	Mnemonic	
Load	LD	→ transfer data from memory to register (AC)
store	ST	→ transfer data from register to memory
Move	MOV	→ transfer data between register, between memory words or memory and register.
exchange	XCH	→ swap information between two register or between register and memory word.
input	IN	→ transfer data between register and I/O terminal
output	OUT	
PUSH	PUSH	→ transfer data between processor register and memory stack.
POP	POH	

10 marks 2057, 11 2059

Note: It must be realized that different computer use different mnemonic for same instruction.

Eight addressing mode for bad operation:-

It must be realized that the instructions are often associated with a variety of addressing modes. Assembly language conventions use a special character to designate the addressing mode. As an example, consider the load to accumulator instruction when used with eight different addressing modes. The table below shows the recommended assembly language convention and actual transfer accomplished in each case.

Mode	Assembly convention	Register transfer
Direct address	LD ADR	AC ← M[ADR]
Indirect address	LD @ ADR	AC ← M[M[ADR]]
Relative address	LD $ ADR	AC ← M[PC + ADR]
Immediate operand	LD #NBR	AC ← NBR
Index addressing	LD ADR(X)	AC ← M[ADR + XR]
Register	LD R_1	AC ← R_1
Register indirect	LD (R_1)	AC ← M[R_1]
Autoincrement	LD (R_1)+	AC ← M[R_1], R_1 ← R_1 + 1

ii) Data Manipulation instruction:

Data manipulation instructions perform operation data and provide the computational capabilities for the computer. The data manipulation instruction in a typical computer are usually divided into three basic types:-

a) Arithmetic instruction
b) Logical and bit manipulation instruction
c) Shift instruction

a) Arithmetic instructions:-

The four basic arithmetic operation are addition, subtraction, multiplication and division. Most computers provide instruction for all four operations. A list of typical arithmetic instructions is given in table below:-

Name	Mnemonic	
Increment	INC	→ Add one to the value stored in register or memory word.
Decrement	DEC	→ Subtract 1 from the value stored in register or memory word.
Add	ADD	available for different types of data i.e. fixed, floating point, binary or decimal.
Subtract	SUB	
Multiply	MUL	
Divide	DIV	
Add with carry	ADC	
Subtract with borrow	SUB B	
Negate (2's complement)	NEG	

b) Logical and bit manipulation instruction:-

Logical instructions perform binary operations on string of bits stored in a register and are useful for manipulating individual bits or group of bits that represent the binary coded information. The logical instructions consider each bit of the operand separately and by the proper application of logical instructions it is possible to change bit value, to clear a group of bits, etc.

Some of the typical logical and bit manipulation instructions are listed in table below:-

Name	Mnemonic
Clear	CLR
Complement	COM
AND	AND
OR	OR
exclusive-OR	XOR
clear carry	CLRC
set carry	SETC
complement carry	COMC
enable interrupt	EI
Disable interrupt	DI

The clear instruction causes the specified operand to be replaced by 0's. The complement instruction produces the 1's complement by inverting all the bits of the operand. The AND, OR and XOR instruction produce the corresponding logical operand operations on individual bits of the operands.

c) Shift instruction:-

Instructions to shift the content of an operand are quite useful and are often provided in several variations. Shift are operations in which bits of the word are moved to the left or right. The bit shifted in at the end of the word determines the type of shift used. Shift instruction may specify three different type of shift.

i) Logical shift
ii) Arithmetic shift
iii) Circular shift.

The following table list some typical shift instructions.

Name	Mnemonic
Logical shift right	SHR
Logical shift left	SHL
Arithmetic shift right	SHRA
Arithmetic shift left	SHLA
Rotate right	ROR
Rotate left	ROL
Rotate right through carry	RORC
Rotate left through carry	ROLC

iii) Program Control instruction:-

Program control instruction when executed may change the address value in the program counter and cause the flow of control to be altered. The change in value of the program counter as a result of the execution of a program control instruction causes a break in the sequence of instruction execution. Program control instruction provides control over the flow of program execution and a capability for branching to different program segments.

Some typical program control instruction are listed in table below:-

Name	Mnemonics
Branch	BR
Jump	JMP
Skip	SKP
Call	CALL
Return	RET
Compare	CMP
Test	TST

- The branch and jump instruction are used interchangeably. The branch is usually one address instruction. Branch and jump instruction may be conditional and unconditional.
- Skip is the zero address instruction and this will skip next instruction. It may also be conditional or unconditional.
- Call and return instruction are used in conjunction with subroutine call.

Program interrupt:-

Program interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.

The interrupt procedure is, in principle, quite similar to subroutine call except for three variations:-

i) The interrupt is usually initiated by an internal or external signal rather than from the execution of an instruction except for the s/w interrupt.

ii) The address of the interrupt service program is determined by hardware rather than from address field of instruction.

iii) An interrupt procedure usually stores all the information necessary to define the state of CPU rather than storing only the program counter.

Types of interrupt:-

There are three major types of interrupt that cause a break in normal execution of a program. They can be classified as:-

1. External interrupts
2. Internal interrupts
3. Software interrupts

1. External interrupts:-

External interrupts come from input-output devices, from timing devices, from circuit monitoring the power supply or from other external source. Example that cause the external interrupts are I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of event, etc.

2. Internal interrupts:-

Internal interrupts arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps. Example of interrupts caused by internal error condition are register overflow, an invalid operation code, etc. The service program that post processes the internal interrupt determines the corrective measures to be taken.

The difference between internal and external interrupt is that:

i) Internal interrupt is initiated by some exceptional condition caused by the program itself rather than by external event.
ii) Internal interrupts are synchronous with the program while external interrupt are asynchronous.
iii) If the program is rerun, the internal interrupts will occur in the same place each time but external interrupts depends on external conditions that are independent of program being executed at the time.

3. Software interrupt:-

A software interrupt is initiated by executing an instruction. Software interrupt is special call instruction that behaves like an interrupt rather than a subroutine call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The most common use of s/w interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to supervisor mode.

CISC and RISC :-

An important aspect of computer architecture is the design of the instruction set for the processor. The instruction set choosen for particular computer determins the way that machine language programs are constructed. Early computer had small and simple instruction set, forced mainly by the need to minimize the hardware used to implement them. As the digital hardware became cheaper with the advent of integrated circuits, computer instruction tended to increase both in number and complexity.

1. Complex instruction set computer (CISC) :-

Computer with many instruction that employ variety of data types and large number of addressing mode is classified as complex instruction set computer (CISC). One reason for the trend to provide complex instruction set computer is to have a machine language instruction to match each high level language type so that job of the compiler writer becomes easy. example of CISC architecture are : VAX computer and IBM 370 computer.

The major characteristics of CISC computer are :

i) A large number of instruction - typically from 100 to 250 instructions.

ii) Some instruction perform specialized task and are used infrequently.

iii) Large variety of address mode - typically from 5 to 20 different modes.
iv) Variable length instruction format.
v) Instruction in a typical CISC processor provide direct manipulation of operands residing in memory.
vi) As more instruction and addressing modes, the more hardware logic is needed to implement and support them and this may cause computation to slow down.

2. Reduced Instruction Set Computer (RISC):-

In the early 1980's, a number of computer designers recommended that computer use fewer instructions with simple constructs so they can be executed much faster within the CPU without having to use memory as often. This type of computer is classified as a reduced instruction set computer or RISC. The Reduced instruction set computer (RISC) were proposed as an alternative to CISC. The underlying idea behind RISC processor is to simplify the instruction set and reduce the instruction execution time.

The major characteristics of RISC processor are:-

i) Relatively few instructions
ii) Relatively few addressing modes
iii) Memory access limited to load and store instructions.
iv) All operations done within the registers of the CPU.
v) Fixed length easily decoded instruction format.
vi) Single cycle instruction execution.
vii) Hardwired rather than microprogrammed control.

vii) RISC processor consists mostly of register to register operations with only load and store operation for memory access.

viii) The use of only a few addressing mode result from the fact that almost all instruction have simple register addressing.

ix) Relatively large number of register in the processor unit.

x) Efficient instruction pipelining and use of overlapped register windows to speedup procedure call and return.

Overlapped register window:

Procedure call and return occurs quite often in high-level programming languages. When translated into machine language, a procedure call produces a sequence of instructions that save register values, pass parameters needed for the procedure and then calls a subroutine to execute the body of procedure. After a procedure return, the program restores the old register values, passes result to the calling program and returns from the subroutine. Saving and restoring registers and passing parameters and results involve time consuming operations.

A RISC processor use overlapped register window to provide the passing of parameters and avoid the need for saving and restoring the register value. Each procedure call results in allocation of new window consisting of set of registers for use by the new procedure rather than

saving registers in memory. Window for adjacent procedure have overlapping registers that are shared to provide the passing of parameter and register result.

The concept of overlapped register window is illustrated in figure below:-

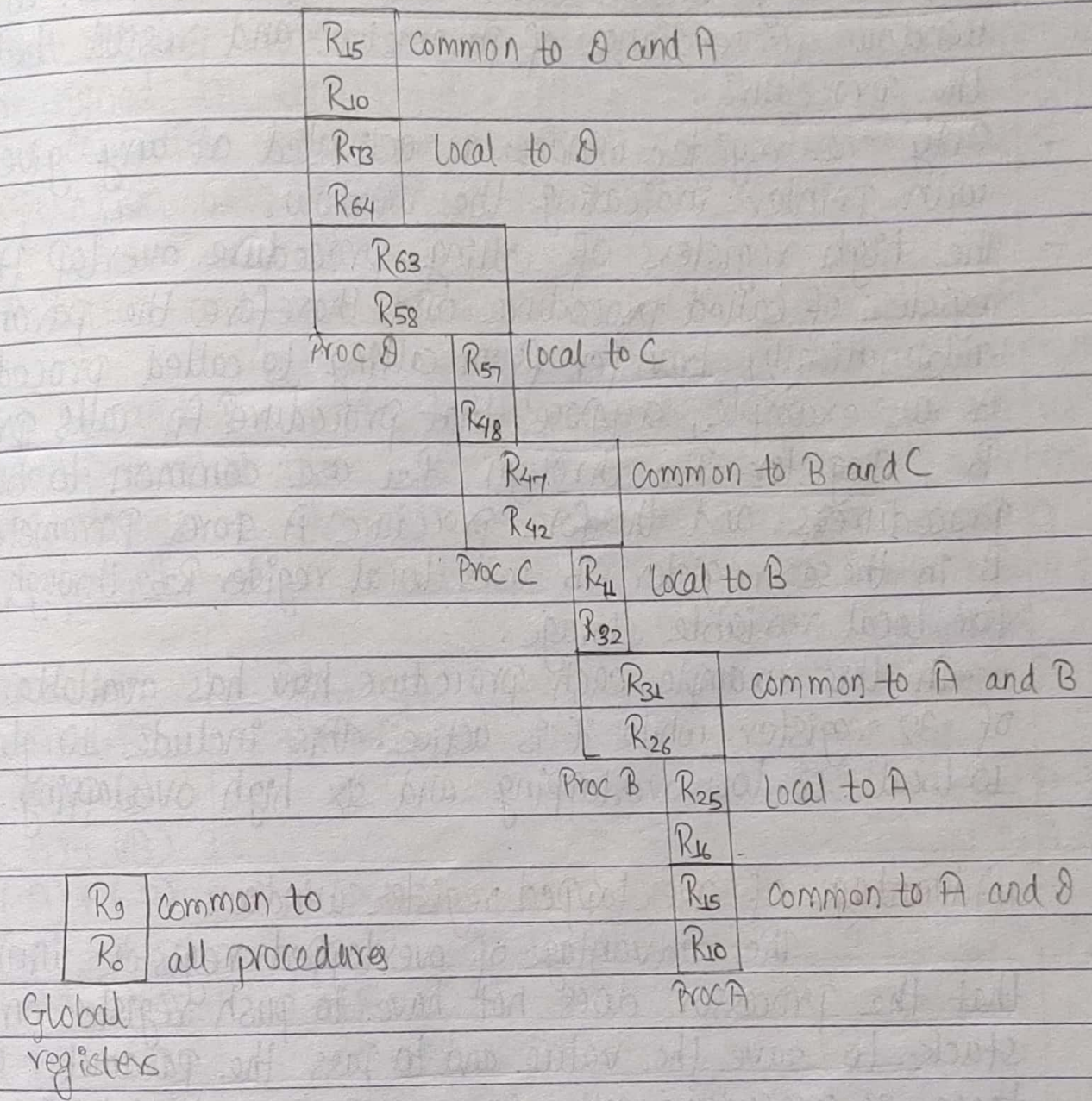


Fig:- Overlapped register windows.

- The system has total of 74 registers.
- Register R_0 through R_9 are global registers that hold parameters shared by all procedure.
- The other 64 register are divided into four windows for procedure A, B, C and D.
- The six register on each window are common to adjacent window for exchange of parameter and result between the procedure.
- Only one register window is activated at any given time with pointer indicating the window.
- The high registers of calling procedure overlap the low register of called procedure and therefore the parameter automatically transfer from calling to called procedure. As an example, suppose that procedure A calls procedure B, Register R_{26} through R_{31} are common to both procedures and therefore procedure A stores parameter for B in these register. B uses local register R_{32} through R_{41} for local variable storage.

In this example, each procedure now has available total of 32 register while it is active. This include 10 global, 10 local, six low overlapping and six high overlapping register.

Advantage of over lapped register window:-

The advantage of overlapped register window is that the processor does not have to push register on a stack to save the value and to pass the parameter when there is procedure call. Since procedure call and returns are so common, it results in significant savings relative to a stack-based approach.

UNIT-6

FIXED POINT COMPUTER ARITHMETIC

Addition and Subtraction:-

There are 3 ways of representing negative fixed point binary number: signed magnitude, signed 1's complement or signed 2's complement. Most computer use the signed 2's computer representation when performing arithmatic operation with integer.

Addition and Subtraction with signed magnitude data:-

Let we designate the magnitude of two number by A and B. When signed number are added or subtracted, we find that there are eight different condition to consider depending on the sign of the number and operation performed. These condition are listed in the table below:

		Subtract Magnitude		
Operation	Add magnitude	When A>B	When A<B	When 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)

Addition and Subtraction algorithm for signed Magnitude data:-

Let A and B are two signed number.

For addition:-

When the sign of A and B are identical, add the two magnitude and attach the sign of A to the result. When sign of A and B are different, compare the magnitudes and subtract the smaller number from larger. Choose the sign of the result to be same as A if A>B or complement of A if A<B. If two magnitude are equal, subtract B from A and make the sign of the result positive.

For subtraction:-

When sign of A and B are different, add the two magnitude and attach the sign of A to the result. When sign of A and B are identical, compare the magnitudes and subtract the smaller number from the larger and choose the sign of result to be same as A if A>B or the complement of A if A<B. If two magnitude are equal, subtract B from A and make the sign of result positive.

Hardware implementation:-

The following figure shows the block diagram of the hardware for implementing the addition and subtraction operation on signed magnitude data.

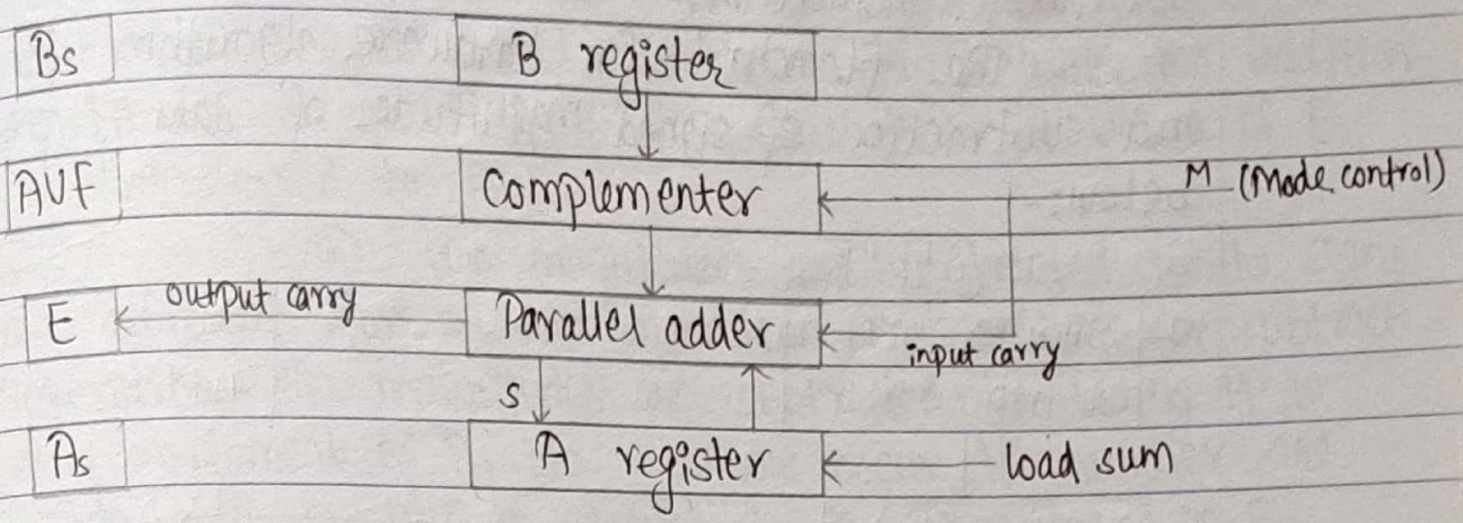


Fig:- Hardware for signed magnitude addition and subtraction

It consits of register A and B to hold the magnitude of number and flipflop As and Bs to hold the corresponding sign. Subtraction is done by adding A to the 2's complement of B. The output carry is transferred to flip flop E, where it can be checked to determine relative magnitude of two numbers. The overflow flipflop holds the overflow bit when A and B are added.

The addition of A and B is done through parallel adder. The sum 'S' is applied to input of A register. The complementer provides an output of B or complement of B depending on the state of mode control M. When M=0, the output of adder is equal to the sum A+B. When M=1, the output of adder equals to A+B+1, which is equivalent to subtraction A-B.

Hardware Algorithm:-

The flowchart for hardware algorithm for addition and subtraction of signed magnitude of data is presented below:-

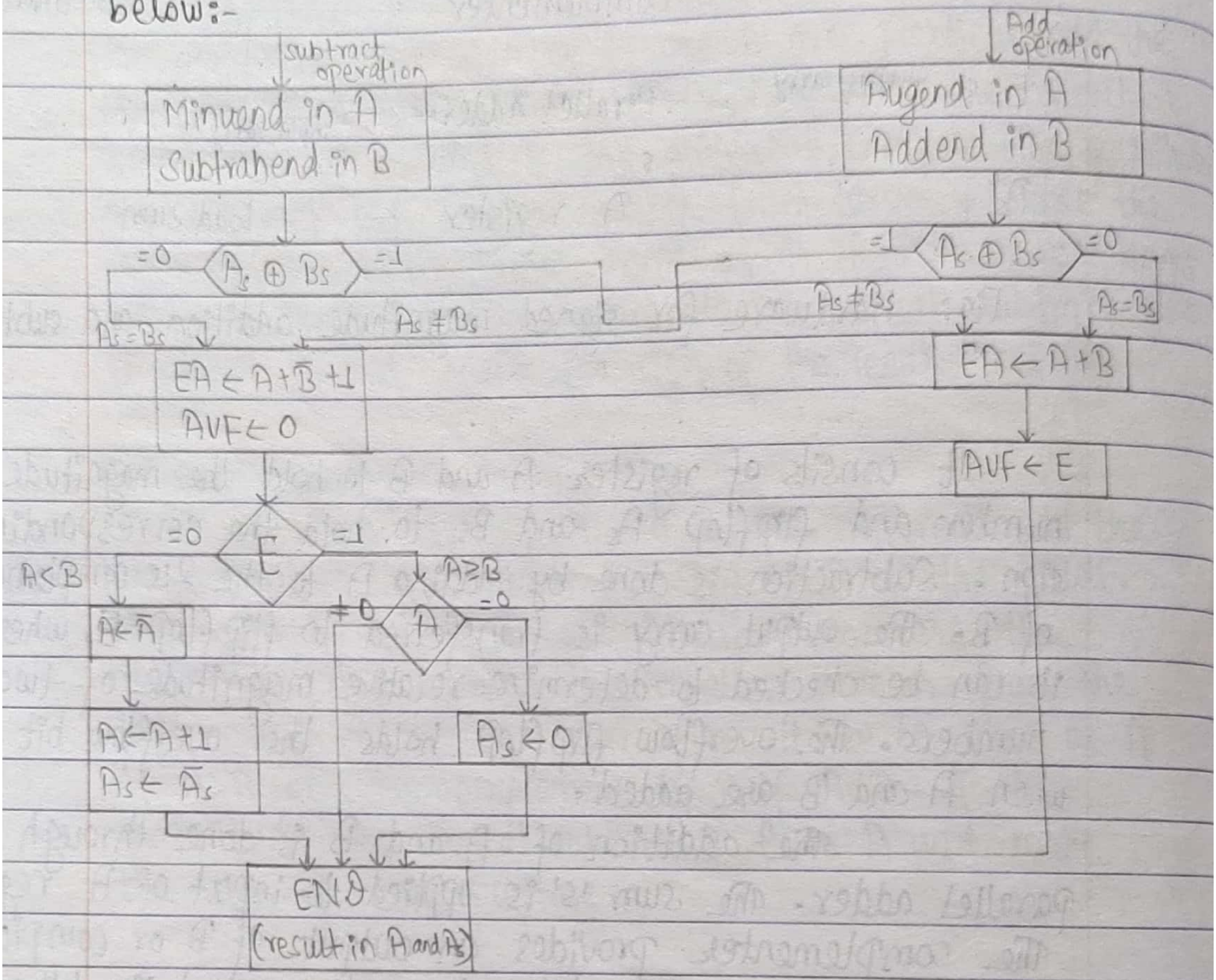


Fig:- Flowchart for add and subtract operation

Here, sign A_s and B_s are compared by an exclusive-OR gate. If the output of the gate is 0, signs are identical and if it is 1, signs are different. Identical sign for add

operation and different sign for subtract operation dictate that magnitude be added. The carry in E after the addition constitutes an overflow if it is equal to 1. The value of E is transferred to AVF.

The two magnitudes are subtracted if the signs are different for an add operation or identical for subtract operation. The magnitudes are subtracted by adding A to 2's complement of B. No overflow occur if number are subtracted so AVF is cleared to 0. The value 1 in E indicates that $A \geq B$ and the number in A is correct result. If value in E is 0, indicates that $A < B$, for this case 2's complement of value in A is necessary for correct result and it is necessary to complement As to obtain the correct sign. The final result is found in register A and it's sign in As.

Example:-

Q. Perform 45 + (-23).

→ Here, operation is add.

45 = 00101101

-23 = 10010111

Now,

$A_s = 0$ A = 0101101

$B_s = 1$ B = 0010111

Here, $A_s \oplus B_s = 1$

So,

$EA = A + \bar{B} + 1$

$= 0101101 + 1101000 + 1 = 10010110.$

AVF=0

→ Here E=1, A=0010110

Now, result is AsA = 00010110.

Assignment:

Perform (-65)+(50), (-30)+(-12), (-20)-(50), (40)-(60)

Addition and Subtraction with signed 2's complement data:-

In signed 2's complement data the leftmost bit of a binary number represent the sign bit: 0 for positive and 1 for negative. If the sign bit is 1, entire number is represented in 2's complement form.

For example: $+33 = 00100001$

$-33 = 11011111$

Addition in signed 2's complement data:

→ Add the two numbers, including their sign bits and discard any carry out of the sign bit position.

Subtraction in signed 2's complement data:

→ Take the 2's complement of the subtrahend and then adding it to the minuend including the sign bit. A carry out of sign bit is discarded.

When two numbers of n digits are added and sum occupies n+1 digits, we say that an overflow occurred. The overflow can be detected by inspecting the last two carries out of the addition. When the two carries are applied to an X-OR gate, the overflow is detected when output of the gate is 1.

Hardware for signed 2's complement addition and subtraction:

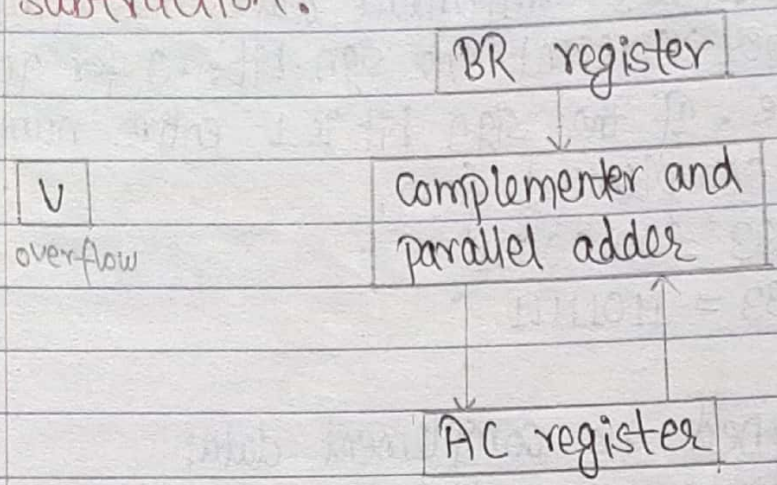


Fig:- Hardware for 2's complement addition and subtraction

The leftmost bit in AC and BR represent sign bit of the number. The two sign bits are added or subtracted together with the other bits in the complementer and adder. The overflow flip flop V is set to 1 if there is an overflow. The output carry is discarded.

Hardware algorithm:

The algorithm for adding and subtracting two binary number in signed 2's complement representation is shown in flowchart below:

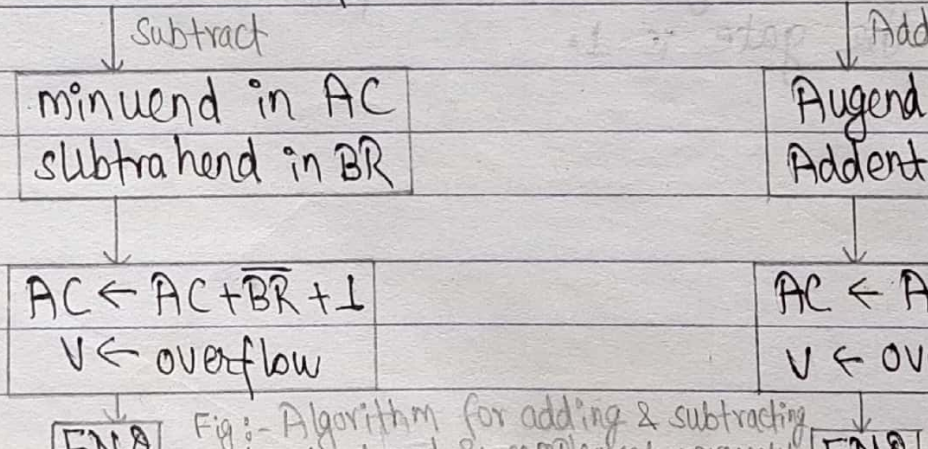


Fig:- Algorithm for adding & subtracting number in signed 2's complement representation

Comparing this algorithm with it's signed magnitude counterparts, we not that it is much simpler to add and subtract numbers if negative numbers are maintained in signed-2's complement representation. For this reason most computer adopt this representation.

Example:-

33 + (-35)

AC = 33 = 00100001

BR = -35 = 2's complement of 35 = 11011101

AC + BR = 00100001
11011101
11111110 = -2 is the result.

2's complement of ans.

00000010.

Multiplication Algorithms:

Multiplication with signed magnitude data:-

Multiplication of two fixed point binary numbers in signed magnitude representation is done by process of successive shift and add operation.

P.T.O

For example:

```
   23      10111      Multiplicand
   19    x 10011      Multiplier
           10111
          10111
         00000
        00000
       10111
  ------------------
  437  110110101      product.
```

The sign of the product is determined from the sign of the multiplicand and multiplier if they are same sign is positive and if they are different sign is negative.

Hardware implementation:-

The hardware implementation for multiplication of signed magnitude data is shown in figure below:

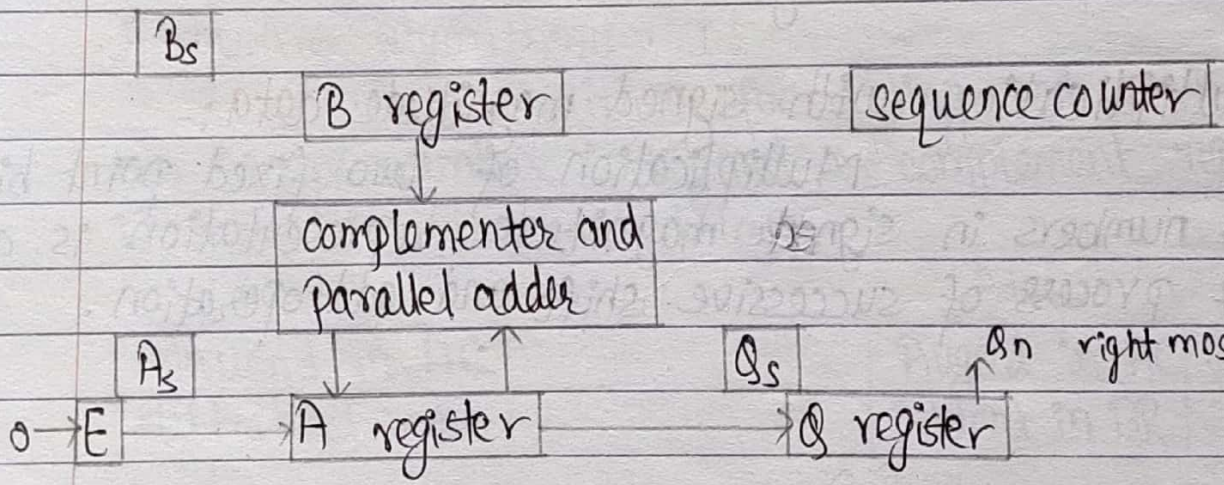


Fig:- Hardware for multiply operation

The idea here is to successively accumulate partial product and shift it right. The multiplier is stored in the Q-register and its sign in Q_s. The sequence counter sc is initially set to number of bit in multiplier. The counter is decremented by 1 after performing each partial product. When the content of counter reaches to 0 the product is formed and process stops.

Initially, multiplicand is in register B and multiplier in Q. The sum of A and B forms a partial product which is transferred to the EA register. Both partial product and multiplier are shifted to the right. In this manner, the rightmost flip flop in register Q designed by Q_n will hold the bit of the multiplier, which must be inspected next.

Hardware Algorithm:

Figure below shows the hardware multiply algorithm.

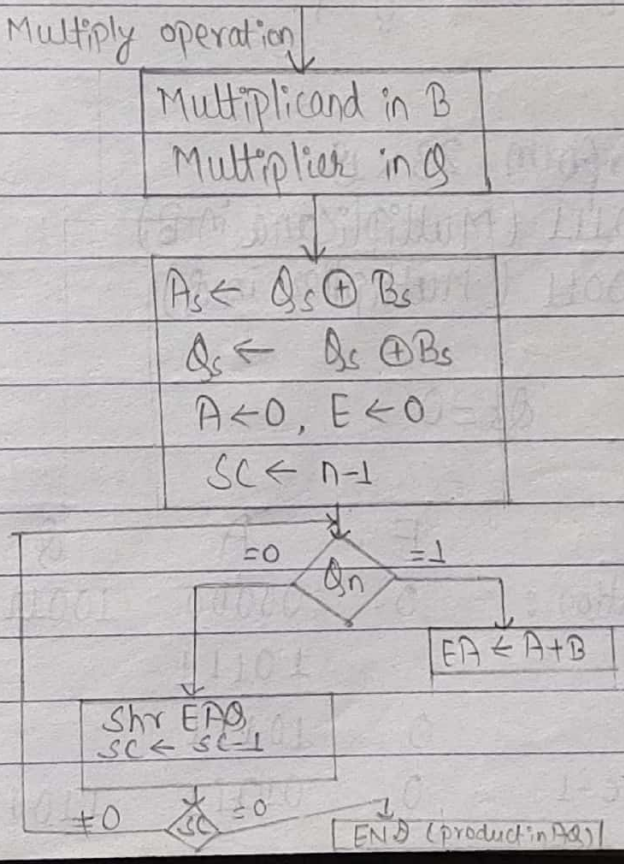


Fig:- Flowchart for multiply operation

Initially, the multiplicand in B and multiplier in Q, their corresponding sign are B_s and Q_s. The signs are compared, and both A_s and Q_s are set to correspond to the sign of product because double length product will be stored in register A and Q. Register A and E are cleared and SC is set to number equal to number of bits of the multiplier (magnitude only).

After initialization, the low order bit of the multiplier in Q_n is tested. If it is 1, multiplicand in B is added to present partial product in A, if it is 0 nothing is done. Register EAQ is then shifted to right to form the new partial product. Sequence counter is decremented by 1 and new value is checked and above process is repeated until SC=0. The final product is available in both A and Q with A holding most significant and Q holding least significant bit.

Example: Perform 23 × 19.

B_sB = 0 10111 (Multiplicand in B)

Q_sQ = 0 10011 (Multiplier in Q)

Now,

$A_s = 0$ $Q_s = 0$

Operation	E	A	Q	SC
initial configuration :	0	00000	10011	101
1: Q_n=1, add B		10111		
	0	10111		
shr EAQ, SC←SC-1	0	01011	11001	100

		E	A	Q	SC
Step2:	$Q_n=1$, add B	0	01011		
			10111		
		1	00010		
	shr EAQ, SC←SC-1	0	10001	01100	011
Step3:	$Q_n=0$, shr EAQ	0	01000	10110	010
	SC←SC-1				
Step4:	$Q_n=0$, shr EAQ	0	00100	01011	001
	SC←SC-1				
Step5:	$Q_n=1$, add B		10111		
		0	11011		
	shr EAQ, SC←SC-1	0	01101	10101	000

Final product in AQ = 0110110101 #

Example 2: Perform 5×(-4)

$B_s B = 5 = 0\ 0101$ (Multiplicand in B)

$Q_s Q = -4 = 1\ 0100$ (Multiplier in Q)

Now,

$A_s = 1$ $Q_s = 1$

Operation	E	A	Q	SC
initial configuration:	0	0000	0100	4
Step1: $Q_n=0$, shr EAQ	0	0000	0010	3
SC←SC-1				

		E	A	Q	SC
Step 2:	$Q_n=0$, shr EAQ $SC \leftarrow SC-1$	0	0000	0001	2
Step 3:	$Q_n=1$, add B		0101		
	shr EAQ, $SC \leftarrow SC-1$	0	0010	1000	1
Step 4:	$Q_n=0$, shr EAQ $SC \leftarrow SC-1$	0	0001	0100	0

Final result: 00010100
$= -20_d$

Multiplication in signed 2's complement representation:

Booth Multiplication Algorithm:-

Booth algorithm is used for multiplying binary integers in signed 2's complement representation. As in all multiplication scheme, Booth algorithm requires examination of the multiplier bits and shifting of partial product. Prior to shifting multiplicand may be added to the partial product, subtracted from the partial product, or unchanged according to the following rule:

1) Multiplicand is subtracted from the partial product upon encountering the first least significant 1 in a string of

1's in the multiplier.

ii) The multiplicand is added to the partial product upon encountering the first 0 in the string of 0's in the multiplier.

iii) The partial product does not change when the multiplier bit is identical to previous multiplier bit.

Hardware implementation:-

The hardware implementation of Booth algorithm requires the register configuration shown in figure below:

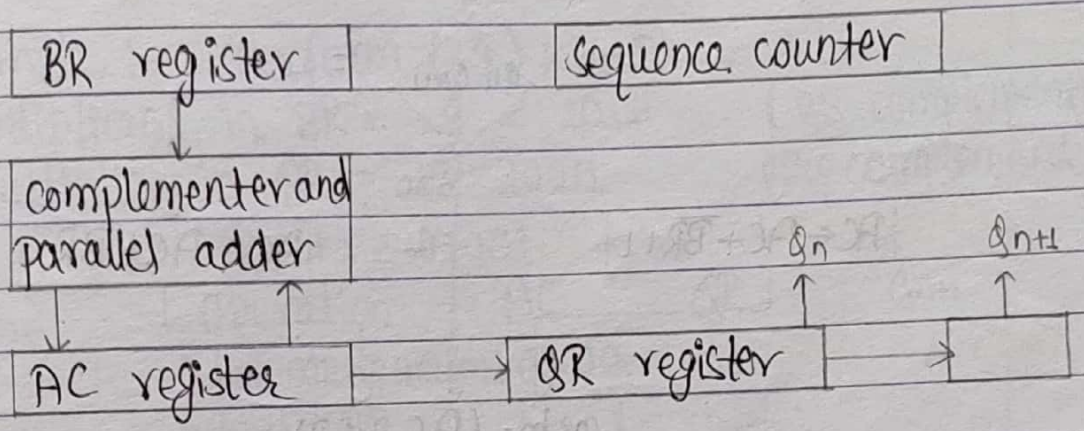


fig:- Hardware of booth algorithm

BR register holds multiplicand with sign bit. QR register holds multiplier with sign bit. Q_n designates least significant bit of multiplier and Q_{n+1} is append to Q_n to facilitate double bit inspection of the multiplier.

Flowchart of booth algorithm:

The flowchart of booth algorithm is shown below:-

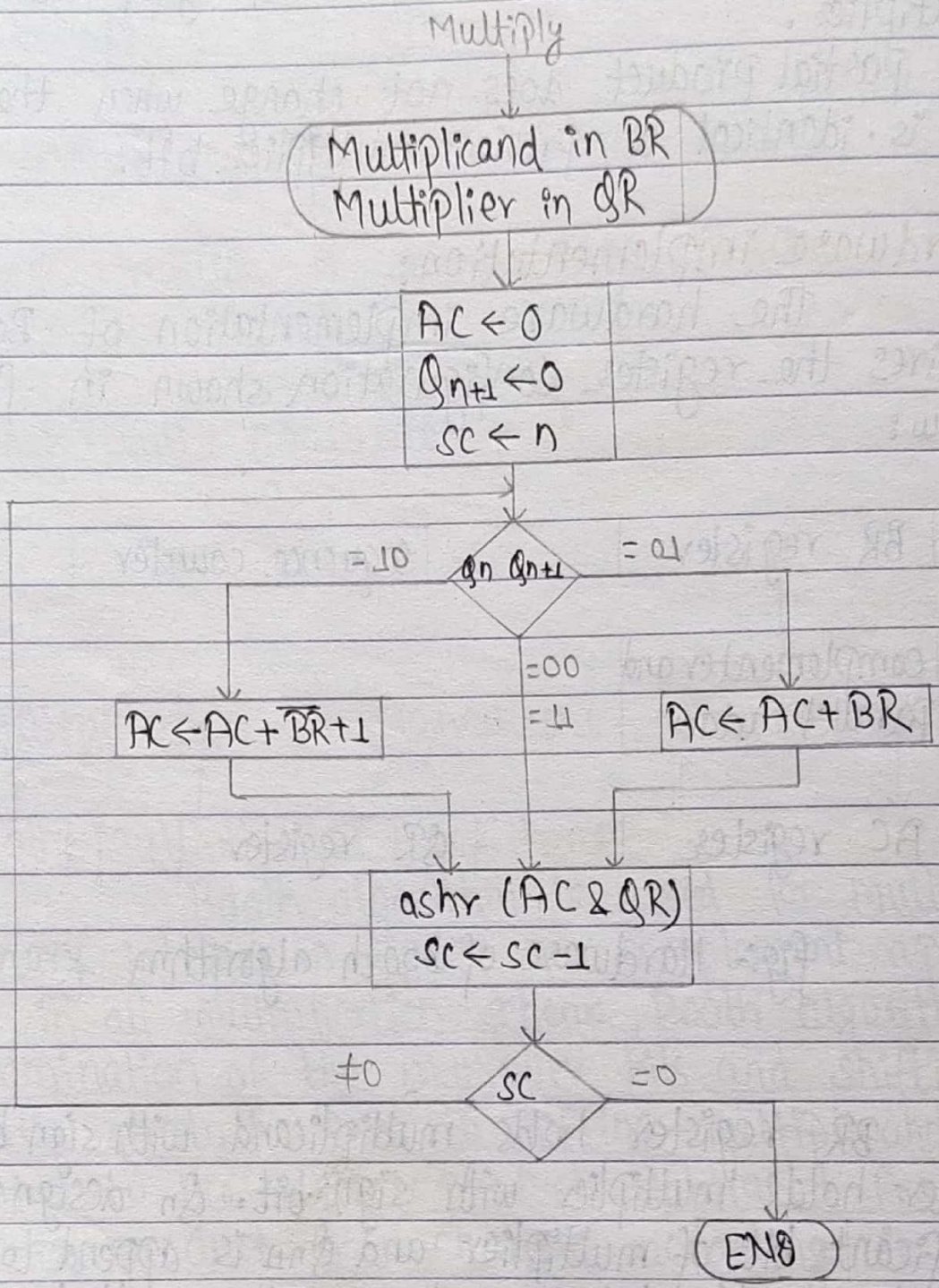


Fig:- Booth algorithm for multiplication of signed-2's complement numbers.

- AC and appended bit Q_{n+1} are initially 0 and sequence counter SC is set of number n i.e. number of bit in multiplier.
- Two bits of multiplier Q_n and Q_{n+1} are inspected. If two bit are equal to 10 then subtract the multiplicand from partial product in AC. If two bits are 01 then addition of multiplicand to partial product in AC. When two bits are equal the partial product does not change.
- The next step is to arithmatic shift right and sequence counter is decremented and computational loop is repeated n times.

Example :- Perform $(-9) \times (-13) = +117$

Sol:
Multiplicand in BR = -9 = 10111 (2's complement)
Multiplier in QR = -13 = 10011 (2's complement)
$\overline{BR} + 1 = 01001$

	$Q_n\ Q_{n+1}$	Operation	AC	QR	Q_{n+1}	SC
		initial configuration:	00000	10011	0	5
Step 1:	1 0	subtract BR	01001			
			01001			
		ashr, SC←SC-1	00100	11001	1	4
Step 2:	1 1	ashr, SC←SC-1	00010	01100	1	3
Step 3:	0 1	add BR	10111			
			11001			
		ashr, SC←SC-1	11100	10110	0	2
Step 4:	0 0	ashr, SC←SC-1	11110	01011	0	1
Step 5:	1 0	subtract BR	01001			
			10.0111			
		ashr, SC←SC-1	00011	10101	1	0

∴ Final result (AC and QR) 0001110101 = 117

(K)

Array Multiplier:-

The multiplication of two binary numbers can be done with one microoperation by means of a combinational circuit that form a product bits all at once. An array multiplier requires large number of gates so it was not economical until the development of integrated circuits.

To see how array multiplier can be implemented with combinational circuit, consider the multiplication of 2-bit number as shown in figure below:-

$$
\begin{array}{cccc}
 & & b_1 & b_0 \\
 & & a_1 & a_0 \\
\hline
 & & a_0b_1 & a_0b_0 \\
 & a_1b_1 & a_1b_0 & \\
\hline
c_3 & c_2 & c_1 & c_0
\end{array}
$$

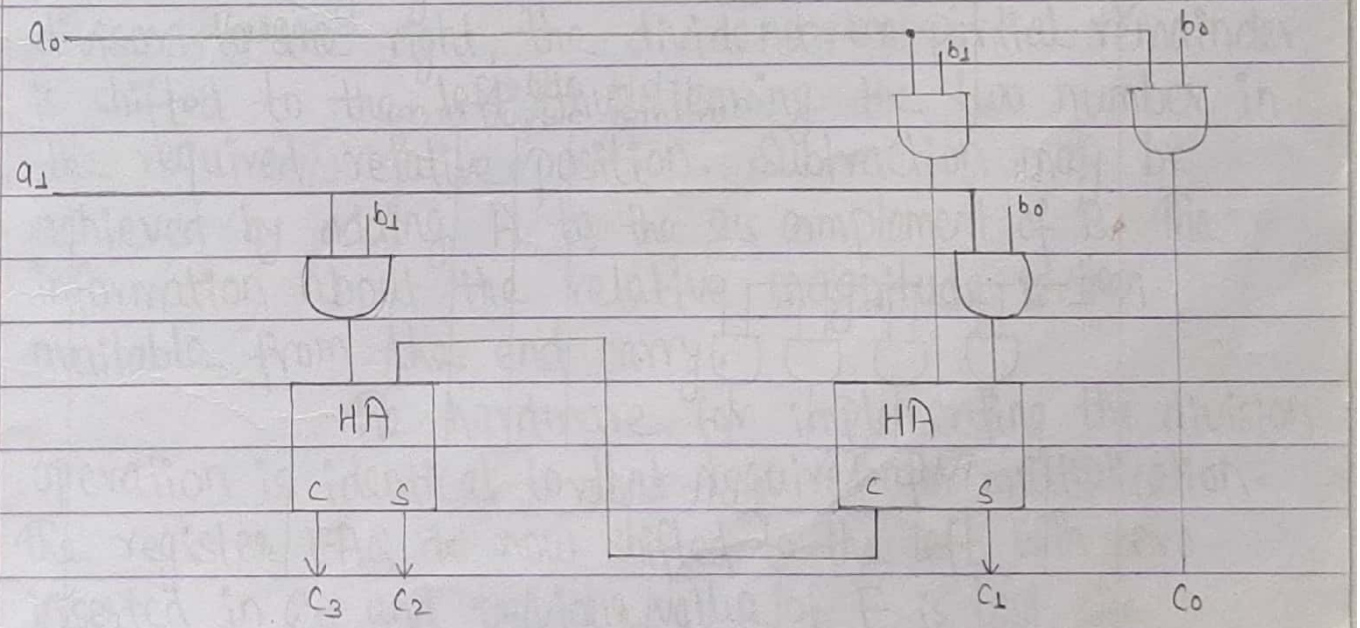


Fig:- Array multiplier

A combinational circuit binary multiplier with more bits can be constructed in similar fashion. A bit of multiplier is ANDed with each bit of the multiplicand in as many levels as there are bits in the multiplier. The binary output in each level of AND gate is added in parallel with partial product of previous level to form a new partial product.

For J multiplier bits and K multiplicand bits we need J x K AND gates and (J-1) K-bit address to product the product of J+K bits.

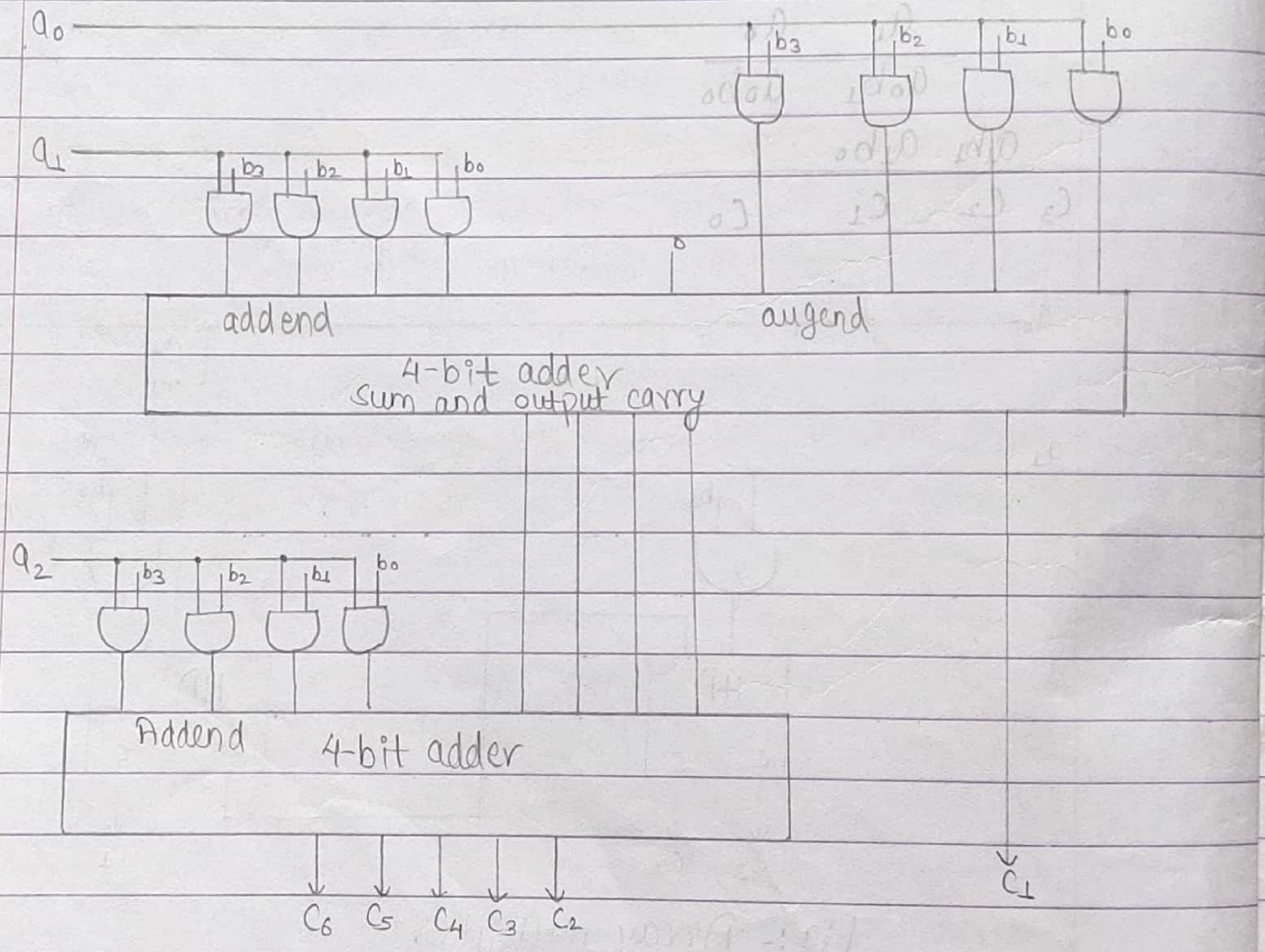


Fig:- 4-bit by 3-bit array multiplier

(16)

Division algorithm:-

Division of two fixed-point binary numbers in signed magnitude representation is done by the process of successive compare shift and subtract operation. The binary division process is illustrated in the example below:-

```
                1101        Quotient = Q
B = 10001 ) 01111000000 ( Divident = A
          - 10001↓
          ---------
            001011 0
             - 1000 1↓
            ---------
              00101 00
               - 1000 1
              ---------
                00011 0   ⇒ Remainder
```

Hardware implementation for signed magnitude data:-

While implementing division in digital system, process is slightly different. Instead of shifting the divisor to the right, the dividend or partial remainder, is shifted to the left, thus leaving the two number in the required relative position. Subtraction may be achieved by adding A to the 2's complement of B. The information about the relative magnitude is then available from the end carry.

The hardware for implementing the division operation is identical to that required for multiplication. The register EAQ is now shifted to the left with zero inserted in Q_n and previous value of E is lost. The hardware configuration is given below:

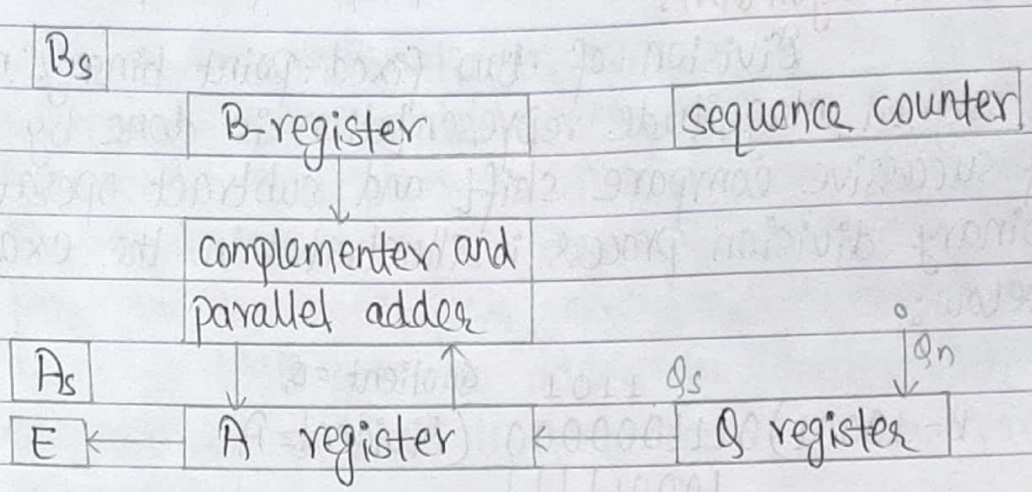


Fig:- Hardware for division.

Divide overflow:-

- The division operation may result in a quotient with an overflow when working with finite size register.
- When the dividend is twice as long as the divisor, the condition for overflow can be stated as follows: A divide overflow condition occurs if the higher order half bits of the dividend constitutes a number greater than or equal to divisor.
- Overflow condition is usually detected when special flip flop is set. We call it a divide overflow flip flop DVF.

10 marks 2069.

Hardware algorithm:- (Restoring method)

The flowchart for hardware divide algorithm is shown below:-

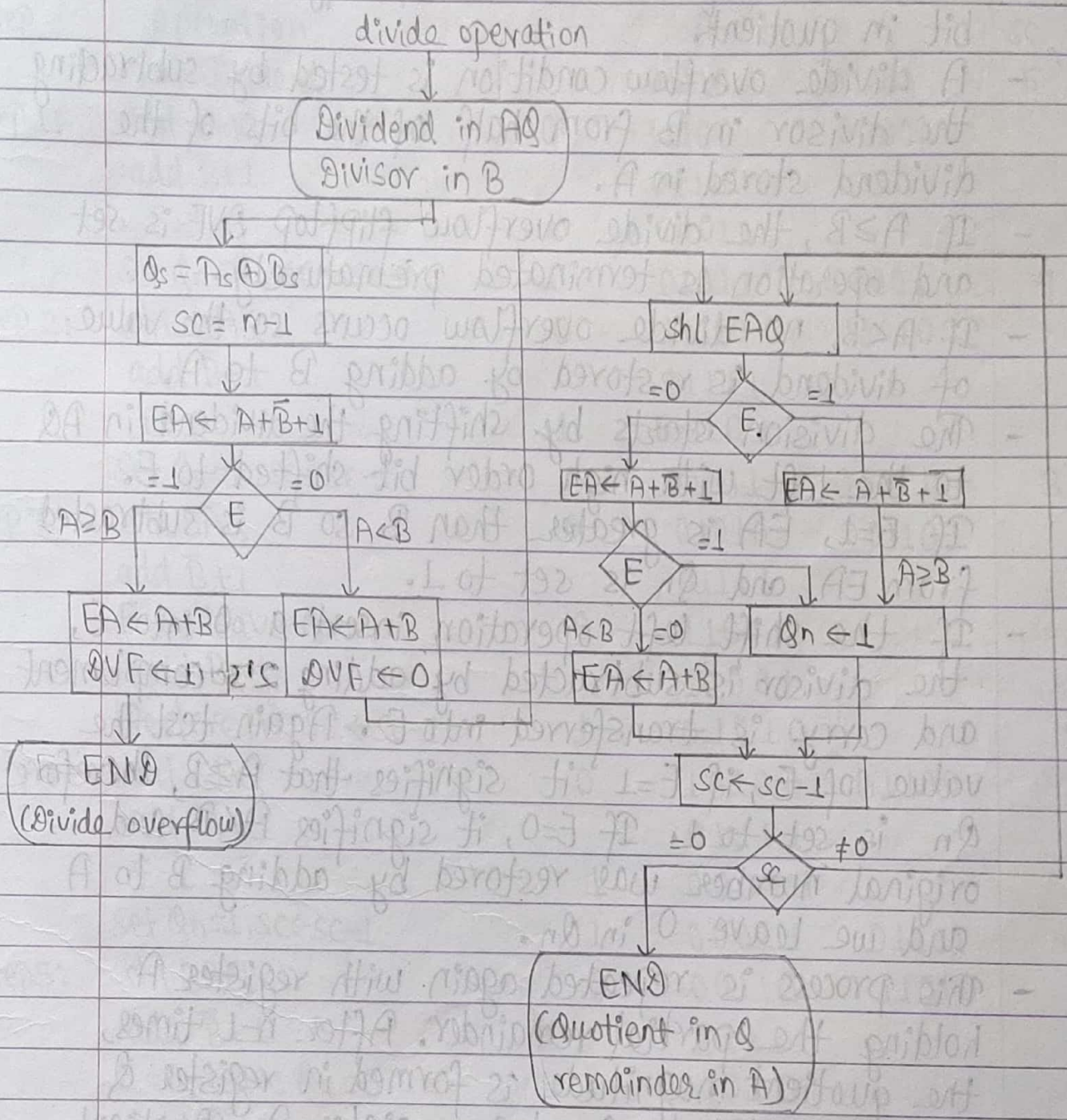


Fig:- Flowchart for divide operation (restoring method)

- The dividend in A and Q and divisor in B. The sign of the result (i.e. sign of quotient) is transferred in Qs. The sequence counter specify number of bit in quotient.
- A divide overflow condition is tested by subtracting the divisor in B from half of the bits of the dividend stored in A.
- If $A \geq B$, the divide overflow flipflop DVF is set and operation is terminated prematurely.
- If $A < B$, no divide overflow occurs so the value of dividend is restored by adding B to A.
- The division starts by shifting the dividend in AQ to the left with high order bit shifted to E. If E=1, EA is greater than B so B is subtracted from EA and Q_n is set to 1.
- If the shift left operation insert a 0 into E, the divisor is subtracted by adding 2's complement and carry is transferred into E. Again test the value of E, if E=1 it signifies that $A \geq B$, therefore Q_n is set to 1. If E=0, it signifies $A < B$ and original number was restored by adding B to A and we leave 0 in Q_n.
- This process is repeated again with register A holding the partial remainder. After n-1 times, the quotient magnitude is formed in register Q and remainder is found in register A. Quotient sign is in Q_n and remainder sign in As.

Numerical example:-

Divisor B = 10001 Dividend = 01110 00000

First calculate $\overline{B}+1$ = 01111

Step	Operation	E	A	Q	SC
	Put dividend		01110	00000	5
Step 1:	shl EAQ	0	11100	00000	
	add $\overline{B}+1$		01111		
	E=1	1	01011		
	Set Q_n=1, SC←SC-1	1	01011	00001	4
Step 2:	shl EAQ	0	10110	00010	
	add $\overline{B}+1$		01111		
	E=1	1	00101		
	Set Q_n=1, SC←SC-1	1	00101	00011	3
Step 3:	shl EAQ	0	01010	00110	
	add $\overline{B}+1$		01111		
	E=0, leave Q_n=0	0	11001		
	Add B,		10001		
	(Restore A), SC←SC-1	1	01010		2
Step 4:	shl EAQ	0	10100	01100	
	add $\overline{B}+1$		01111		
	E=1	1	00011		
	Set Q_n=1, SC←SC-1	1	00011	01101	1
Step 5:	shl EAQ	0	00110	11010	
	add $\overline{B}+1$		01111		
	E=0, leave Q_n=0	0	10101		
	Add B		10001		
	(Restore A), SC←SC-1	1	00110		0

Neglect E:

Remainder in A: 00110

Quotient in Q: 11010_4

(21)

Other division algorithm:

1) Comparision method:-

In the comparision method, A and B are compared prior to the subtraction operation. If $A \geq B$, B is subtracted from A. If $A < B$ nothing is done. The partial remainder is shifted left and number are compared again. The comparision can be determined prior to subtraction by inspecting the end carry out of the parallel adder prior to its transfer to register E. The flowchart is given below:-

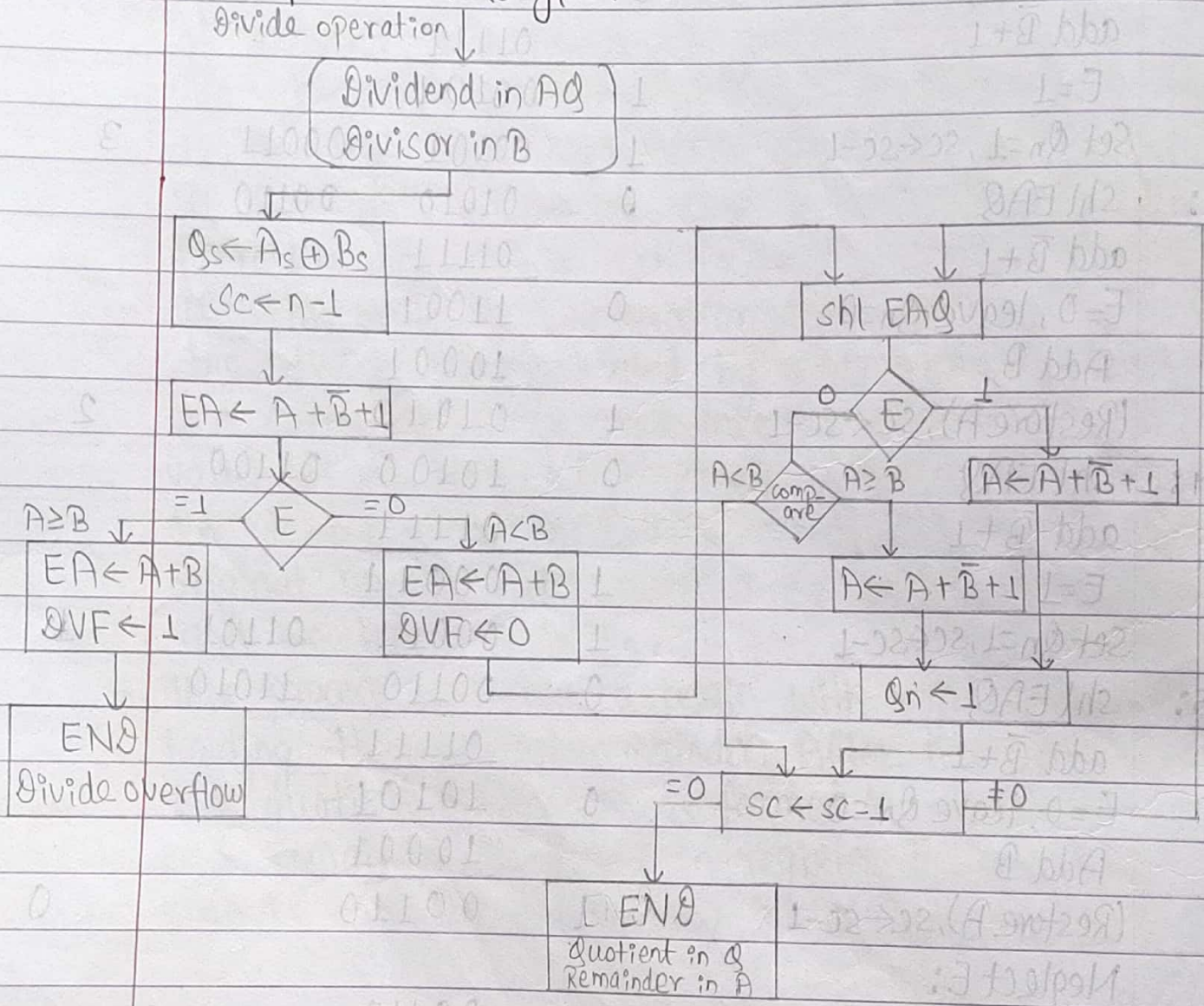


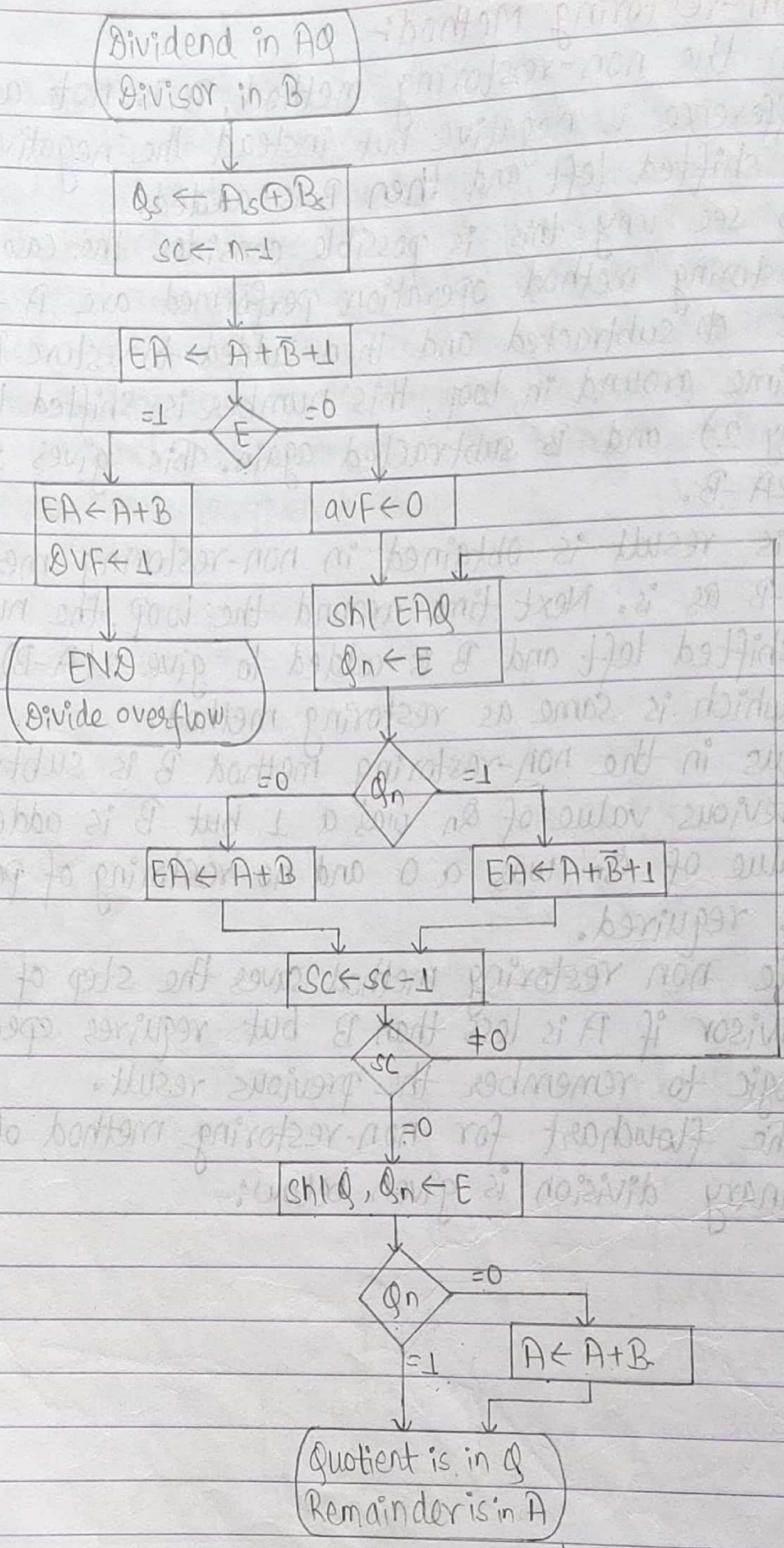
Fig:- Flowchart of comparison method.

10-marks 2067, 2069.

Non-restoring Method:-

- In the non-restoring method, B is not added if the difference is negative but instead the negative difference is shifted left and then B is added.
- To see why this is possible, consider the case A<B, in restoring method operation performed are A-B+B i.e. B is subtracted and then added to restore A. The next time around in loop, this number is shifted left (or multiplied by 2) and B subtracted again. This gives 2(A-B+B)-B = 2A-B.
- This result is obtained in non-restoring method by leaving A-B as is. Next time around the loop, the number is shifted left and B is added to give 2(A-B)+B = 2A-B which is same as restoring method.
- Thus in the non-restoring method B is subtracted if the previous value of Q_n was a 1 but B is added if previous value of Q_n was a 0 and no restoring of partial remainder is required.
- The non restoring method saves the step of adding the divisor if A is less than B but requires special control logic to remember the previous result.
- The flowchart for non-restoring method of fixed point binary division is given below:-

P.T.O



Example:-

Divisor B = 10001 Dividend = 0111000000

→ First we calculate; $\bar{B}+1$ = 01111

Step:	Operation	E	A	Q	SC
	put Dividend:		01110	00000	5
	$EA \leftarrow A+\bar{B}+1$:		01111		
		0	11101		
Step 1:	shl EAQ, $Q_n \leftarrow E$	1	11010	00000	
	$Q_n=0$, $EA \leftarrow A+B$		10001		
	$SC \leftarrow SC-1$	1	01011		4
Step 2:	shl EAQ, $Q_n \leftarrow E$	0	10110	00001	
	$Q_n=1$, $EA \leftarrow A+\bar{B}+1$		01111		
	$SC \leftarrow SC-1$	1	00101		3
Step 3:	shl EAQ, $Q_n \leftarrow E$	0	01010	00011	
	$Q_n=1$, EA, $A+\bar{B}+1$		01111		
	$SC \leftarrow SC-1$	0	11001		2
Step 4:	shl EAQ, $Q_n \leftarrow E$	1	10010	00110	
	$Q_n=0$, $EA \leftarrow A+B$		10001		
	$SC \leftarrow SC-1$	1	00011		1
Step 5:	shl EAQ, $Q_n \leftarrow E$	0	00110	01101	
	$Q_n=1$, $EA \leftarrow A+\bar{B}+1$		01111		
	$SC \leftarrow SC-1$	0	10101		0
Step 6:	SC=0, shl Q, $Q_n \leftarrow E$			11010	
Step 7:	$Q_n=0$, $A \leftarrow A+B$		10001		
		1	00110		
	Quotient in Q:			11010	
	Remainder in A:		00110		

(ES)

Unit - 7

Input - Output Organization

Input-Output subsystem and peripherals:-

The input output subsystem of a computer, referred to as I/O, provides an efficient mode of communication between the central system and the outside environment. Programs and data must be entered into computer memory for processing and result obtained from computations must be recorded or displayed for the user. A computer serves no useful purpose without the ability to receive information from an outside source and to transibit result in a meaningful form.

Input or output devices attached to the computer are called peripherals. These peripherals may be analog or digital and serial or parallel. Most common peripherals are keyboard, display unit and printers. Peripherals that provides auxiliary storage for the system are magnetic disk and tapes. Not all input comes from people, and not all output is intended for people.

The input-output organization of a computer is function of the size of the computer and the devices connected to it. The difference between a small and large system is mostly dependent on the amount of hardware the computer has available for communicating with peripheral units and number of peripherals connected to the system.

Input output interface:-

Input output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer needs special communication links for interfacing them with the central processing unit. The purpose of communication link is to resolve the differences that exist between central computer and each peripheral. The major differences are:-

i) Peripherals are electromechanical devices and their manner of operation is different from operation of the CPU and memory which are electronic devices. Therefore, conversion of signal may be required.

ii) The data transfer rate of peripherals is usually slower than the transfer rate of CPU so synchronization mechanism is needed.

iii) Data codes and formats in peripherals differs from the word format in CPU and memory.

iv) The operating mode of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer system includes special hardware component between the CPU

and peripherals to supervise and synchronize all input and output transfer. These component are called interface unit. In general, interface is a point of contact between two part of the system. Two main types of interfaces are CPU interface that correspond to system bus and input output interface that depends on the nature of input output device.

2066, 2068
10 marks

I/O bus and interface Module:-

Peripherals connected to computer needs special communication link to interface with CPU. This special link is called I/O bus. The concept of I/O bus and interface module is illustrated by the following figure:-

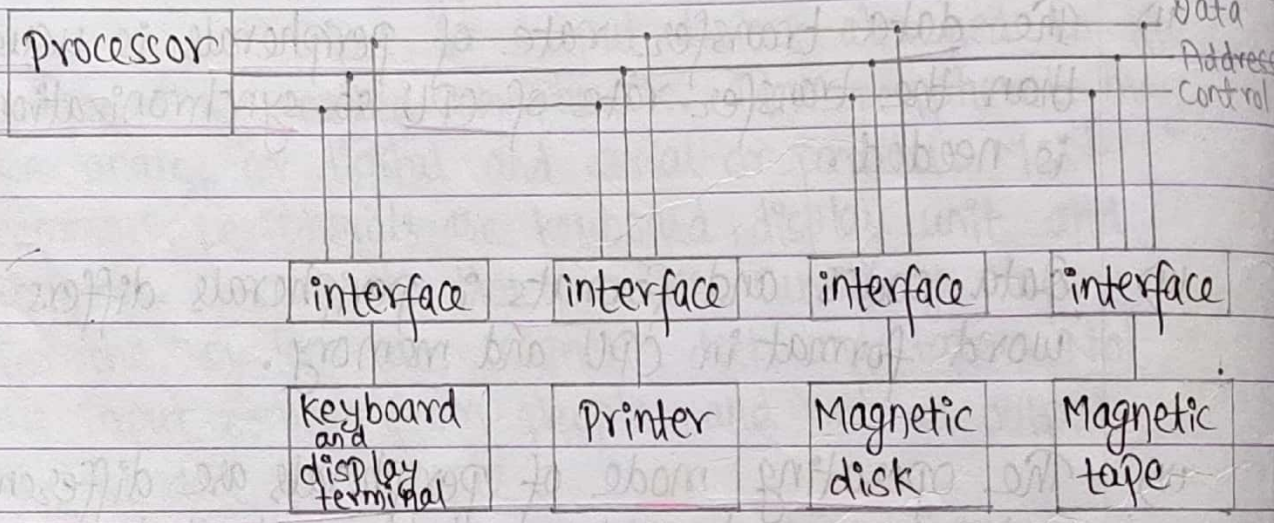


Fig:- Connection of I/O bus to input output devices

- The I/O bus consist of data lines, address line and control lines.
- Each peripheral device has an interface associated

with it. The function of interface unit are:-

i) decode the address and control received from I/O bus.
ii) interprete the signal received from I/O bus and provides the signals for peripheral controller.
iii) synchronize the data flow and supervise the transfer between peripheral and processor.

- The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address line. When interface detect it's own address, it activates the path between the bus lines and device. ~~it controls~~ and other devices are disabled by their interface.

- At the same time, the processor provides the function code in control lines and interface selected responds to function code and proceeds to execute it.

I/O command:-

The function code provided by processor in control line is called I/O command. The interpretation of command depends on the peripheral that the processor is addressing. There are four types of commands that an interface may receive.

1) Control command:-

This command is issued to activate the peripheral and to inform it what to do. For example; magnetic tape unit may be instructed to backspace a tape by one record, to rewind the tape or to start the tape moving in the forward direction. The control command issued depends on the peripherals and each peripheral receive its own sequence of control command depending on its mode of operation.

ii) Status command:-

The status command is used to test various status condition in the interface and peripherals. For example; the computer may wish to check the status of peripherals before start transfer.

iii) Data input command:-

It causes the interface to read data from the peripherals and place it into interface buffer. Processor check if data are available using status command and then issue the data input command. The interface places the data on data lines, where they are accepted by the processor.

iv) Data output command:-

It causes the interface to respond by transferring data from bus into one of its register. The interface then communicates with device controller and send the data to the device.

I/O Versus Memory bus:-

In addition to communicating with I/O, the processor must communicate with the memory unit. The communication link between memory and processor is called memory bus. Like the I/O bus the memory bus contains data, address and read/write control lines.

There are three ways that computer buses can be used to communicate with memory and I/O.

1. use two separate buses one for memory and another for I/O.
2. use one common bus for both memory and I/O but have separate control lines for each.
3. use one common bus for memory and I/O with common control lines.

Isolated I/O Versus Memory mapped I/O:-

Isolated I/O:-

Many computer use one common bus to transfer information between memory or I/O and CPU. The distinction between memory transfer and I/O transfer is made through separate read write lines. The CPU specifies address by enabling one of two possible read, write lines. The I/O read and I/O write enabled during I/O transfer and memory read/write enabled during memory transfer. This configuration isolates all I/O interface addresses from addresses of memory and is referred to as Isolated I/O method.

In isolated I/O, the CPU has distinct input and output instruction, each of these is associated with the address of an interface register. When the CPU fetches and decode these instruction, it places the address associated with these instruction into common address line and it enables I/O read/write control line. This informs the external component attached to common bus that address is for I/O device not for memory. On other hand, when the CPU is fetching an operand from memory, it places memory address on address lines and enable the memory read/write control lines. This informs external component that address is for memory not for I/O interface.

⇒ The key characteristics of isolated I/O in summary are:-

i) Separate I/O read write control lines in addition to memory read write control lines.
ii) Separate memory and I/O address space.
iii) CPU has distinct input and output instruction.
iv) During I/O transfer CPU enable I/O read/write line and during memory transfer CPU enable memory read/write lines.

Memory Mapped I/O:-

- The configuration in which computer employ only one set of read and write signal and do not distinguish between memory and I/O addresses is called memory mapped I/O.

- In this configuration, computer treats an interface register as being part of the memory system. The assigned addresses for interface register can not be used for memory word, which reduces the memory address range available.
- In this configuration CPU can manipulate I/O data with the same instruction that are use to manipulate memory word.
- Computers with memory mapped I/O can use memory type instruction to access I/O data. It allows the computer to use the same instruction for input-output and memory transfer.

⇒ The key characteristics of memory mapped I/O are :-

i) Single set of read/write control lines (no distinction between memory and I/O transfer).
ii) Memory and I/O addresses share common address space.
iii) No specific input output instruction.
iv) Same memory reference instruction can be used for I/O transfer.

I/O interface unit :-

I/O interface unit is shown in the block diagram below. It consist of two data register called ports, control register, a status register, bus buffer and timing and control circuit. The interface communicates with CPU through data bus. Chip select (CS) and register select (RS) input determine the address assigned to the interface.

I/O read and write are two control lines that specify an input and output respectively. The four registers communicates directly with I/O device attached to the interface.

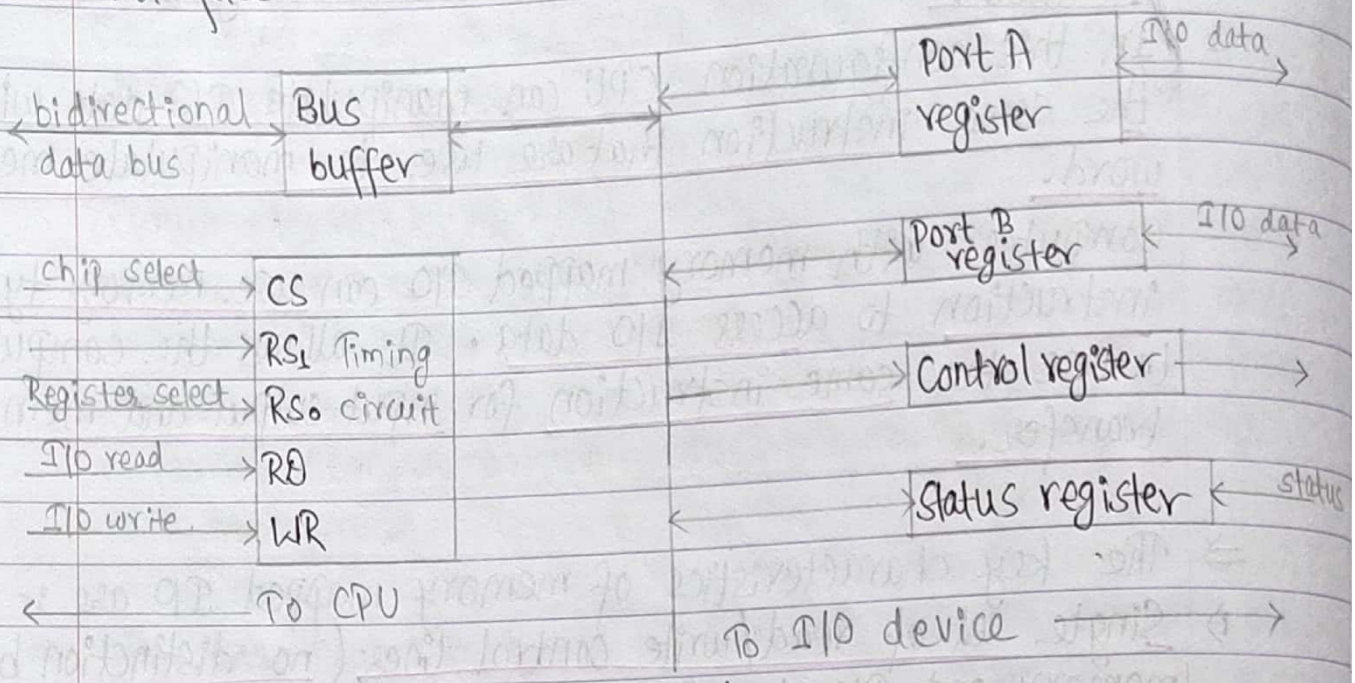


Fig:- Interface Unit

CS	RS_1	RS_0	Register selected
0	x	x	None: data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Fig:- Example of I/O interface unit.

I/O data can be transferred into either port A or B port This interface unit may operate with an output device, input device or with device that require both input and output.

Mode of data transfer :-

Binary information received from an external device is usually stored in memory for later processing. The CPU merely executes the I/O instruction and may accept the data temporarily, but the ultimate source and destination is memory unit. Data transfer between the central computer and I/O devices may be handled in variety of modes. The following are three possible mode of data transfer to and from the peripherals :-

i) programmed I/O
ii) interrupt initiated I/O
iii) Direct Memory access (DMA)

i) Programmed I/O :-

In programmed I/O, method, the I/O device does not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instruction by CPU, including input instruction to transfer the data from device to CPU and store instruction to transfer data from CPU to memory. Other instruction may be needed to verify that data are available from device and to count the number of word transferred.

Programmed I/O are the result of I/O instruction written in the computer program. In programmed I/O, each data item transfer is initiated by an instruction in the program. Usually, transfer is to and from CPU register and peripherals. Other instruction are needed to transfer data to and from CPU and memory. Transferring

the data under program control requires constant monitoring of peripherals by CPU. In the Programmed I/O method, the CPU stays in program loop until the I/O unit indicates that it is ready for data transfer. This is time consuming process since it keeps the processes busy need lessly.

2) Interrupt initiated I/O :-

In Programmed I/O method, CPU constantly monitoring the peripheral to check whether device is ready to transfer but this is time consuming process so alternative to programmed I/O is to use interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from device. This type of I/O is called interrupt initiated I/O.

In interrupt initiated I/O, when interface determines that device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stop the task, it is processing, branches to a service program to process the I/O transfer and then returns to the task it was originally performing. In this type of I/O, CPU can proceed to execute another program until the interrupt arrives.

3) Direct Memory Access (DMA):-

DMA is a sophisticated I/O technique that allows peripheral device to manage memory bus directly for fast data transfer between memory and I/O devices. During the DMA transfer, CPU is idle and has no control of the memory buses but a DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed inside state in variety of ways. One common method used is to disable the bus through control signal and capture the bus. The bus request (BR) input is used by DMA controller to request a CPU bus and CPU activate the bus grantt (BG) output to inform the DMA that bus are available. The DMA that originates the bus request take the control of bus to conduct the memory transfer without the processor intervention. When the DMA terminates the transfer, it disables the bus request line and CPU disable the bus grant line, takes the control of buses and returns to it's normal operation.

When the DMA takes control of bus system, it communicates with memory and transfer can be made in following two ways:-

1) Brust transfer:-

In brust transfer, a block sequence consisting of memory word is transferred in a continuous burst. This mode of transfer is needed for fast device such as magnetic disk.

ii) Cycle stealing :-

It allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU. The CPU delay its operation for one memory cycle to allow DMA to steal one memory cycle.

10 marks 2019.

DMA controller :-

The block diagram of typical DMA controller is shown in figure below:-

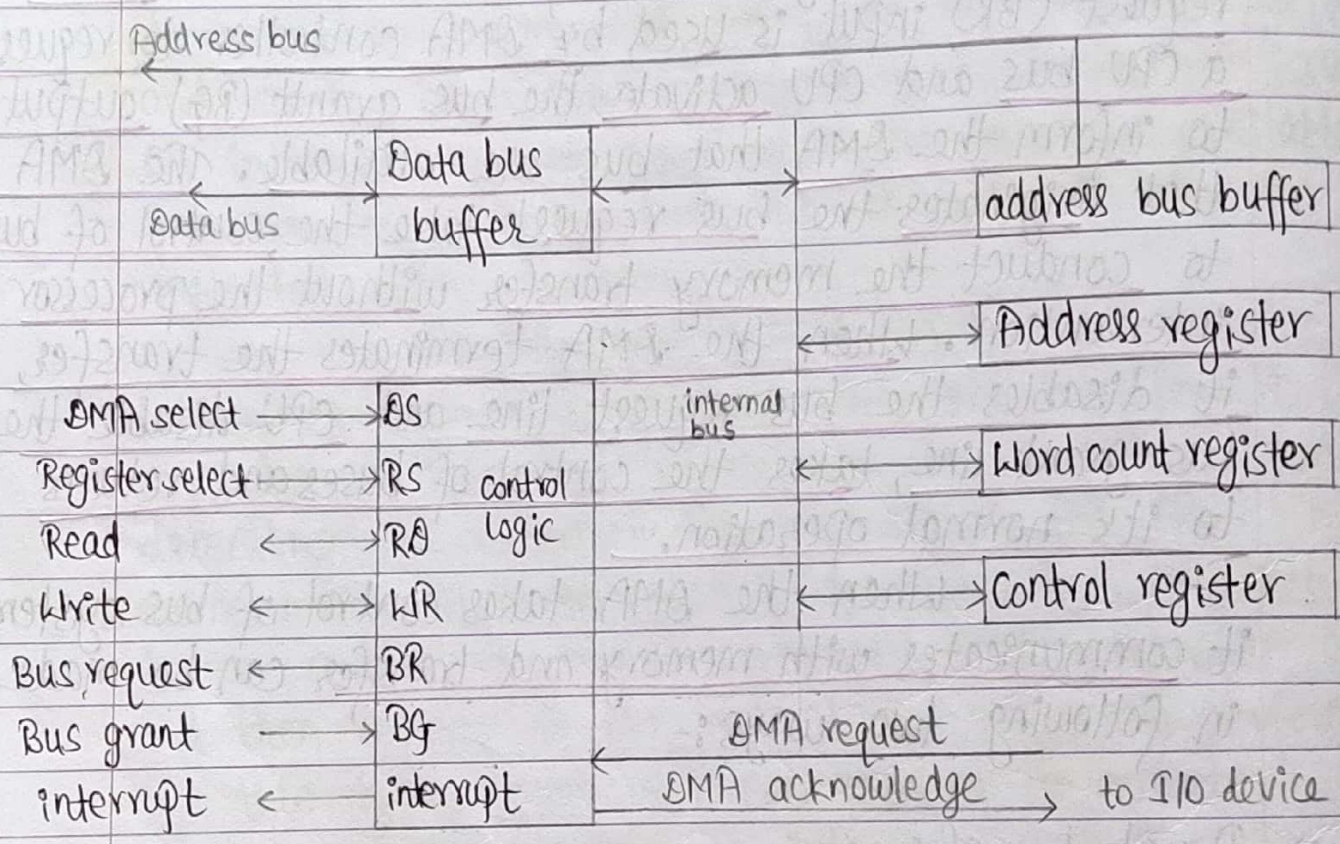


Fig:- Block diagram of DMA controller

- The DMA controller communicates with the CPU via the data bus and control lines. The register in the DMA are selected by the CPU through the address bus by enabling DS and RS input.
- When the BG (bus grant) input is zero the CPU can communicate with the DMA register through the data bus to read from and write to the DMA register.
- When BG=1, the CPU gives the control of bus to the DMA controller and DMA controller communicates directly with the memory by address in address bus and activating the RD and WR control.
- The DMA controller communicates with external peripheral through the request and acknowledge line.
- The DMA controller has three register: an address register, a word count register and control register. The address register contains an address to specify the desired location in memory. The address register is incremented after each word is transferred to memory. The word count register hold the number of word to be transferred. This register is decremented by one after each word to be transfered. The control register specify the mode of transfer.

The DMA is first initialized by the CPU. After that, the DMA start and continues to transfer data between memory and peripheral unit until an entire block is transferred. The CPU initialize the DMA by sending following information through the data bus:

1. The starting address of memory block where data are available or where data are to be stored.

2. The word count, which is the number of words in memory block.
3. Control to specify mode of transfer.
4. Control to start the DMA transfer.

DMA transfer :-

The position of DMA controller among the other components in a computer system is illustrated in figure below:-

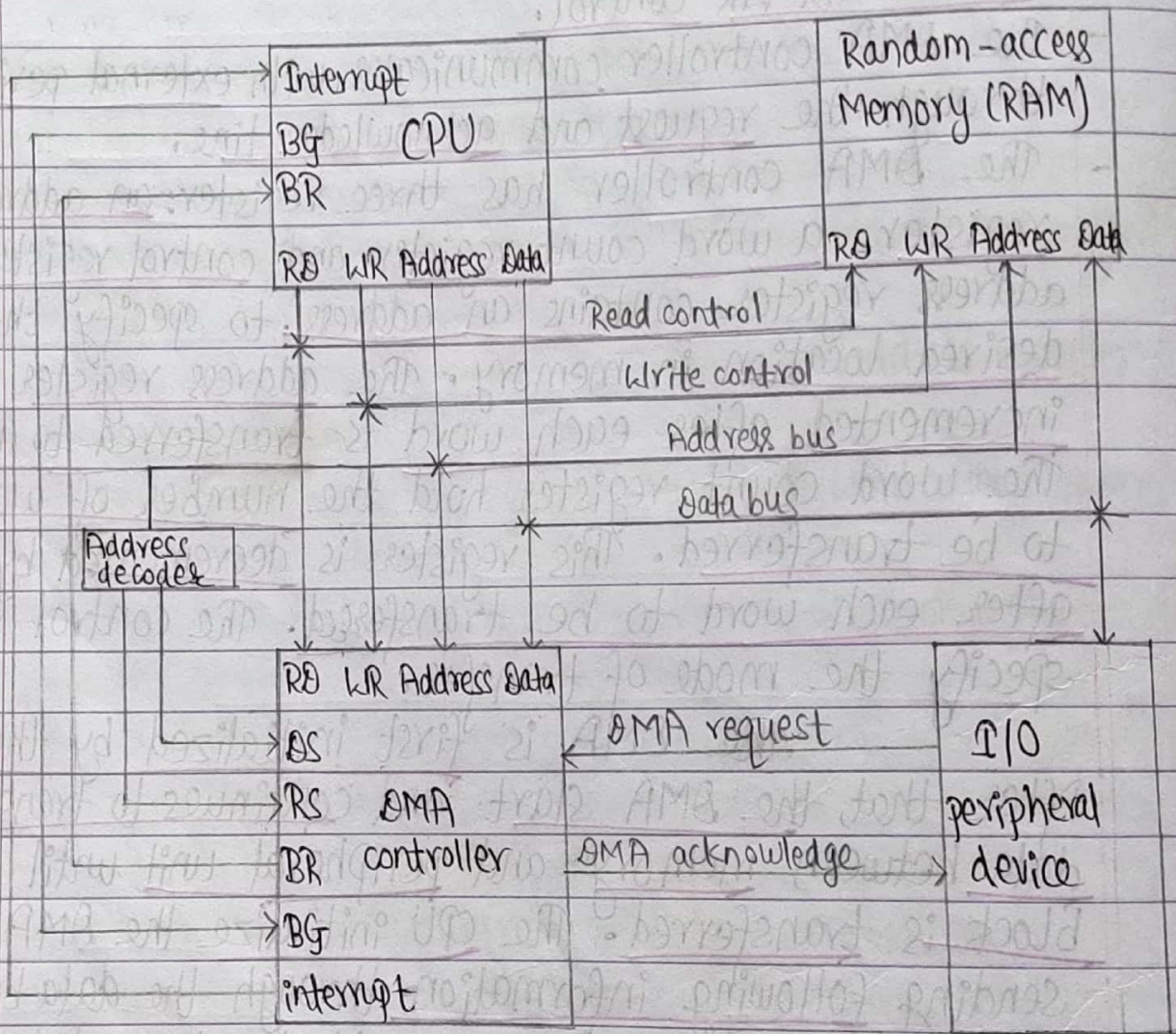


Fig:- DMA transfer in computer system

{Is paragraph change}

The CPU communicates with the DMA through the address and data buses as with the interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command it start the transfer between peripheral device and memory.

When the peripheral device sends a DMA request, the DMA controller activates the BR line. The CPU responds with BG lines. The DMA then puts current value of its address register into the address bus, initiate the RD or WR signal and sends a DMA acknowledge to the peripheral device. When BG=0, the RD and WR input lines allowing the CPU to communicate with the internal DMA register and when BG=1, the communication between memory and DMA controller.

When the peripheral device receives a DMA acknowledge, it puts a word in data bus or receive a word from the data bus. The peripheral unit can then communicate with memory for direct transfer between two unit. For each word that is transferred, the DMA increments its address register and decrement its word count register. The DMA stops any further transfer and remove its bus request if word count register reaches zero. The DMA informs the CPU of the termination by the means of an interrupt. When CPU respond to the interrupt, it check the content of count register, if it is zero, it indicates that word were transferred successfully.

10-May-2020

Input Output Processor (IOP):-

Instead of having each interface communicate with the CPU, a computer may incorporate one or more external processor and assign them to task of communicating directly with all I/O devices. Such a processor is called input output processor. In short, processor with direct memory access capability that communicates with I/O device is called I/O processor.

The IOP is similar to a CPU except that it is designed to handle the detail of I/O processing. Unlike the DMA controller that must be set up entirely by the CPU, the IOP can fetch and execute its own instructions. IOP instructions are specifically designed to facilitate I/O transfer. The block diagram of computer with two processor is shown in figure below:-

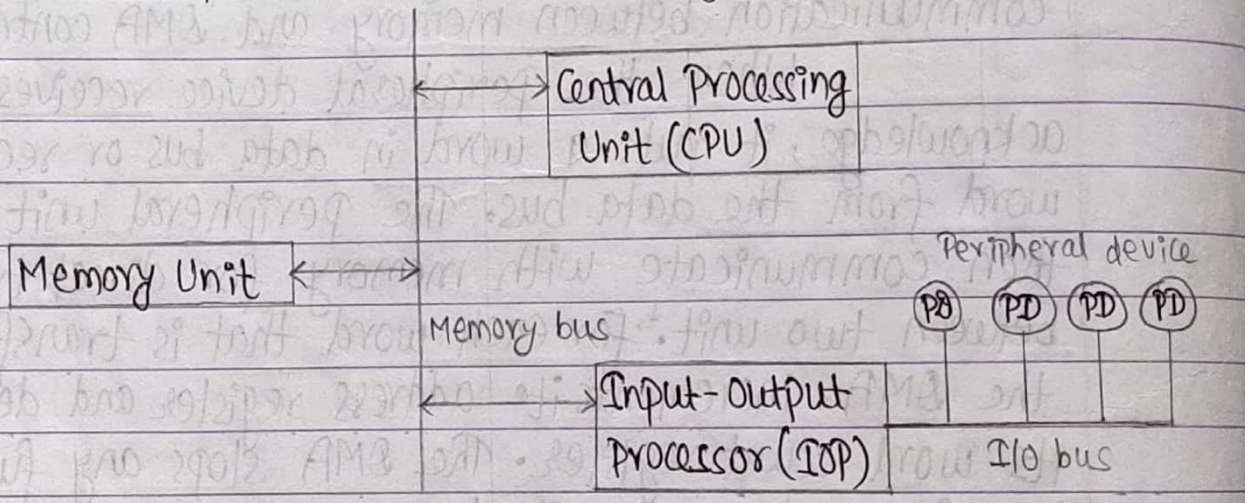


Fig:- Block diagram of computer with I/O processor

Here, memory occupies a central position and can communicates with each processor by means of memory bus. The CPU is responsible for processing data needed

for the solution of computational task. The IOP provides a path for transfer of data between peripheral device and the memory unit. The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources.

CPU and IOP communication:-

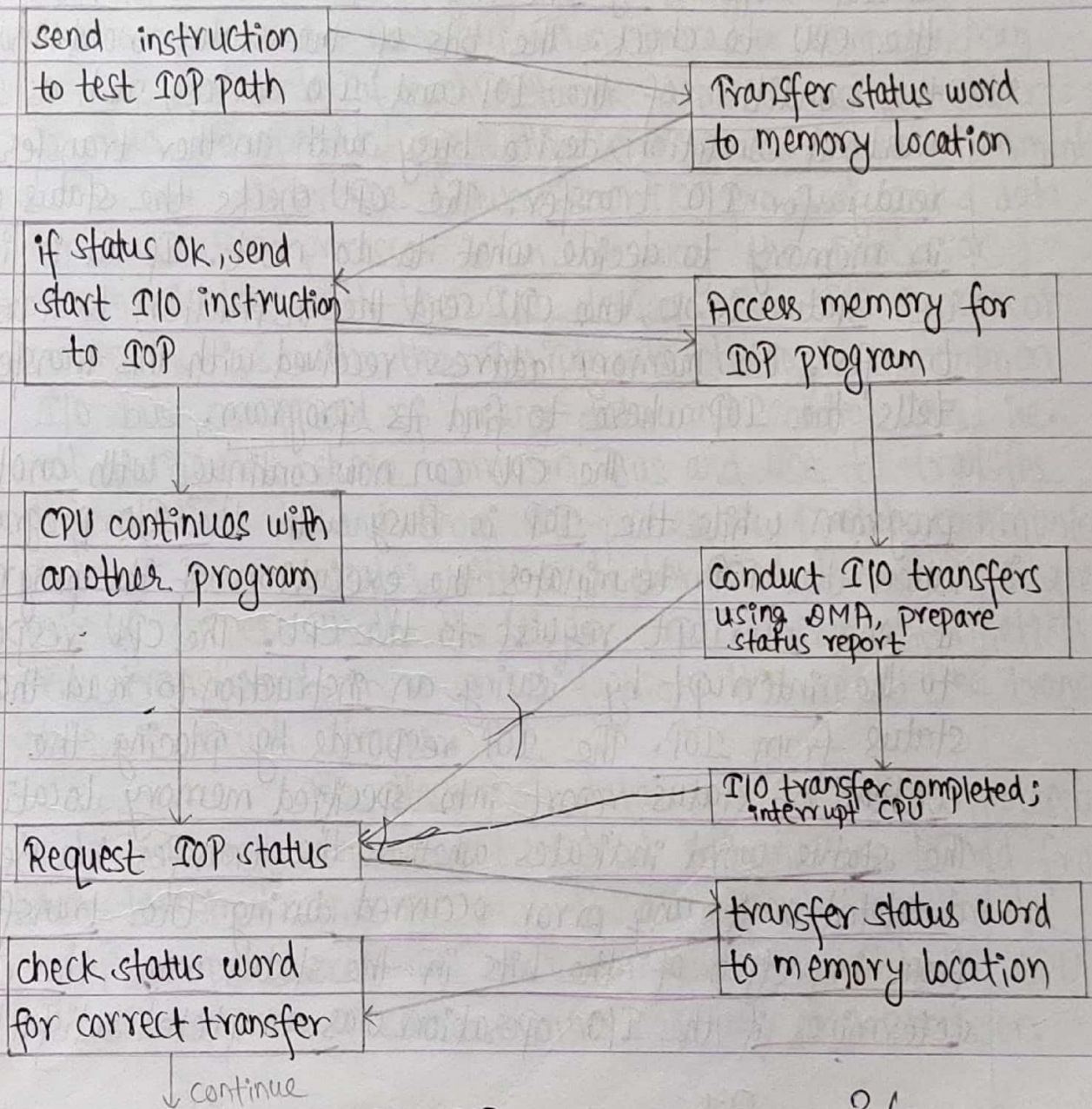


Fig:- CPU-IOP communication

The communication between CPU and IOP may take different forms depending on the particular computer considered. In the most cases, the memory unit acts as a message center where each processor leaves information for the other. The above figure illustrates the method by which CPU and IOP communicate

Mechanism:

The CPU sends an instruction to test the IOP path. The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer or ready for I/O transfer. The CPU checks the status word in memory to decide what to do next. If all is in order i.e. status is ok, the CPU send the instruction to start I/O transfer. The memory address received with this transfer tells the IOP where to find its program.

The CPU can now continue with another program while the IOP is busy with the I/O program. When the IOP terminates the execution of its program, it send interrupt request to the CPU. The CPU responds to the interrupt by issuing an instruction to read the status from IOP. The IOP responds by placing the content of status report into specified memory location. The status word indicates whether the transfer has been completed or if any error occurred during the transfer from inspection of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily

without errors.

Data communication processor:-

Data communication processor is an I/O processor that distribute and collects data from many remote terminals connected through telephone and other communication lines. It is a specialized I/O processor designed to communicate directly with data communication network. A communication network may consist of wide variety of devices such as printers, interactive display devices, digital sensors, etc. with the use of data communication processor, computer can service fragments of each network demand and computer is able to operate efficiently in time sharing environment.

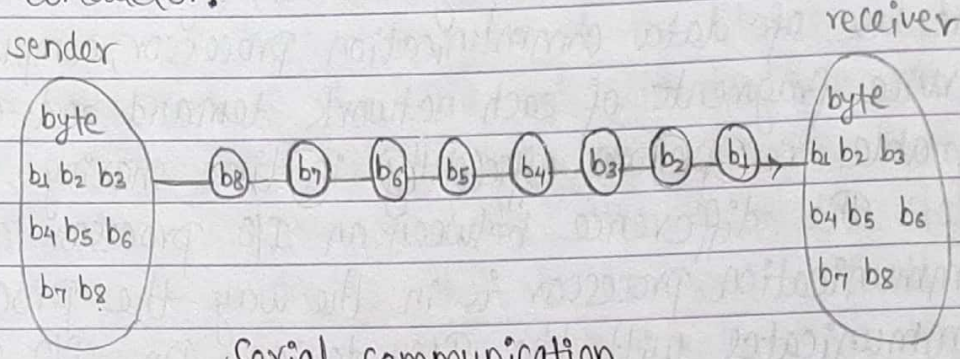
The difference between an I/O processor and a data communication processor is in the way the processor communicates with the I/O devices. An I/O processor communicates with the peripherals through a common I/O bus comprised of many data and control lines i.e. all peripheral share common bus and use to transfer information to and from I/O processor. A data communication processor communicates with each terminal through a single pair of wires. Both the data and control information are transferred in a serial fashion that result the transfer rate is much slower.

The task of data communication processor is to transmit and collect digital information to and from each terminal and respond to all request according to predefined procedures. The DCP communicates with the CPU and memory in a same manner as any I/O processor.

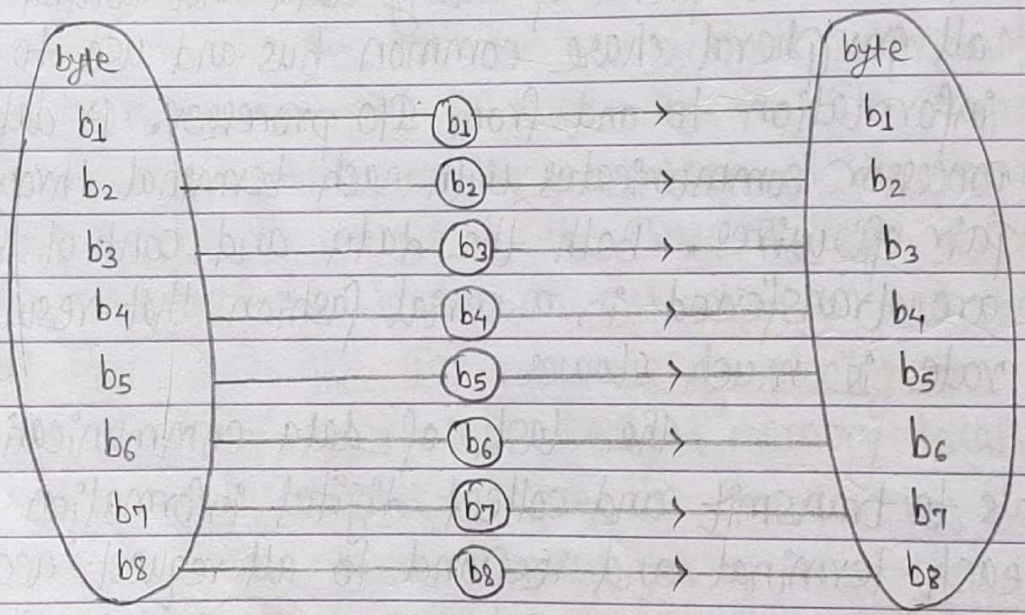
Serial and parallel communication:-

Serial communication is process of sending one bit at a time sequentially over a communication channel or computer bus. This method requires the use of one pair of conductor line.

Parallel communication is faster but requires many wires. It is used for short distance where speed is important. In contrast, serial transmission is slower and but less expensive since it requires only one pair of conductor.



Serial communication



Parallel communication

Mode of data transmission in Data Communication Processor:

Data can be transmitted between two points in three different modes:

i) Simplex
ii) half duplex
iii) full duplex

i) Simplex :-

In simplex mode of data transfer, information carries in one direction only. This mode is seldom used in data communication because the receiver can not communicate with transmitter to indicate the occurrance of error.

Example of simplex transmission are Radio and television broadcasting.

ii) Half duplex :-

The half duplex transmission system is one that is capable of transmitting in both directions but data can be transmitted in only one direction at a time. The time required to switch a half duplex line from one direction to another is called the turn around time.

Example of half duplex transmission is Walkie-talkie.

iii) Full duplex :-

The full duplex transmission can send and receive data in both directions simultaneously.

Example are telephone, mobile phone, etc.

Unit 8

Memory organization:-

Memory:-

The memory unit is an essential component in any digital computer since it is needed needed for storing programs and data. A very small computer with a limited application may be able to fulfill it's intended task without the need of additional storage capacity. There is just not enough space in one memory unit to accommodate all the programs used in a typical computer. Therefore it is more economical to use low cost storage device to serve as a backup for storing the information that is not currently used by the CPU.

The Memory unit that communicates directly with the CPU is called the main memory. Devices that provide backup storage are called auxiliary memory, only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transfered to main memory when needed. The most common auxiliary memory device used in computer systems are Magnetic disks and tapes.

Memory Types:- (on the basis of access):

1) Sequential Access Memory (SAM):-

In computing, SAM is a class of data storage device that read their data in sequence. This is in contrast to Random access memory (RAM) where data can be accessed in any order. Sequential access devices are usually a form of magnetic memory ~~magnetic~~ and typically used for secondary storage in general purpose computer due to their higher density at lower cost.

Example of SAM devices are Magnetic tapes.

ii) Random Access Memory:-

RAM is the form of computer data storage that allows stored data to accessed in any order. RAM is often associated with volatile type of memory where it's stored information is lost if the power is removed.

iv) Memory hierarchy:-

The total memory capacity of a computer can be visualized as being a hierarchy of components. The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high capacity auxiliary memory to relatively faster main memory, to an even smaller and faster cache memory. The following figure illustrates the component of typical memory hierarchy.

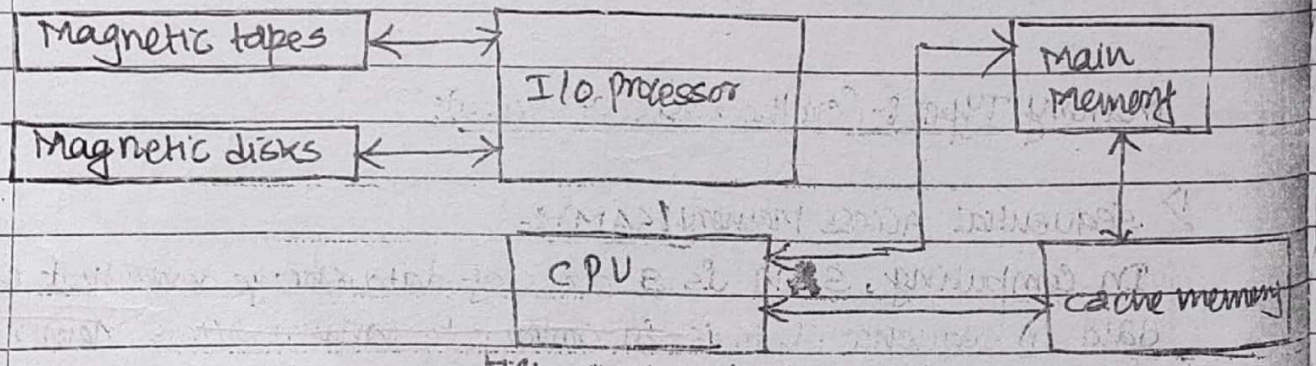


fig:- Component of memory hierarchy in computer system

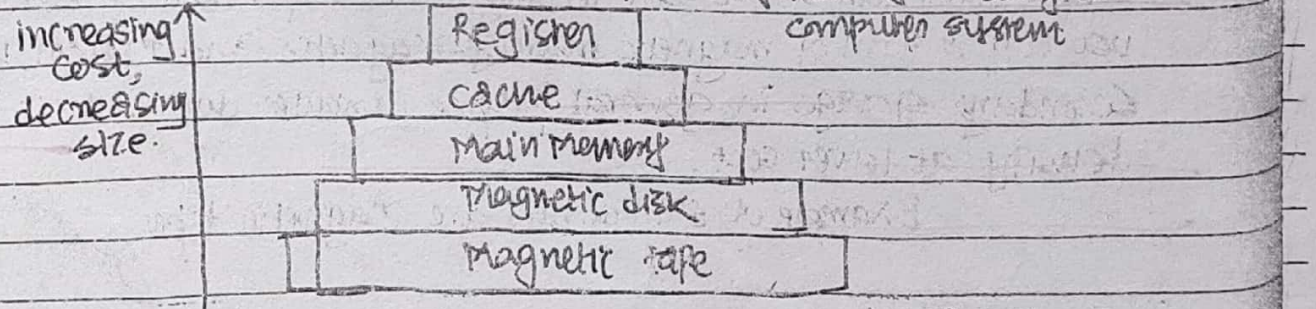


fig:- Typical memory hierarchy.

At the bottom of the hierarchy are the relatively slow magnetic tapes used to store the large amount of files. Next are the magnetic disk used as a backup storage. The main memory occupies the central position by being able to communicates directly with cpu and with auxiliary memory device through an I/o processor. When a programs not residing in main memory are needed by the cpu, they are brought in from auxiliary memory. programs not currently needed in main memory are transfered on to auxiliary memory to provide the space for currently used program and data. cache memory is employed in computer system to compensate speed mis-match between memory and cpu.

As storage capacity of the memory increases, the cost per bit for storing binary information decreases and access time of memory becomes longer. The auxiliary memory has large storage capacity, inexpensive but slow access speed compared to main memory. The cache memory is very small, relatively expensive and very high access speed. The overall goal of using memory hierarchy is to obtain the highest possible average access speed while minimizing the total cost of the entire memory system.

Primary Memory (Main Memory):-

The Main memory is the central storage unit in a computer system. It is relatively large and fast memory used to store and program and data during the computer operation. The principle technology used for the main memory is based on semiconductor integrated circuits. Mainly main memory are of two types:

i) Random access memory (RAM):-

RAM chips are available in two possible operating mode, static and dynamic. The static RAM consist of flip flops that stores the binary information. The stored information remains valid as long as power is applied

to the unit.

The dynamic RAM stores the binary information in the form of electric charges that are applied to capcitors. The capacitors are provided inside the chip by MOS transistors. The stored charge on the capacitors tend to discharged with time and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few millisecond to restore the decaying charge. The static RAM is used in implementing cache memory and dynamic RAM is used for implementing main memory.

(ii) Read only memory (ROM):-

ROM is used for storing programs and data that are permanently resident in the computer. ROM is needed for storing an initial program called bootstrap loader.

The bootstrap loader is a program whose function is to start the computer when power is turned on. Since RAM is volatile, its contents are distroyed when power is turned off. The contents of ROM remain unchanged after power is turned off and on again.

Computer startup:-

The startup of a computer consist of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads the portion of operating system from disk to main memory and control is then transfered to the operating system, which prepares the computer for general use.

RAM and ROM chip:- RAM and ROM chips are available in verety of sizes. If the memory needed is larger than capacity of one chip, it is necessary to combine a number of chip to form the required memory size.

RAM chip:-

RAM chip is better suited for communication with the cpu if it has one or more control input that select a chip when needed. The following figure shows the block diagram of RAM chip.

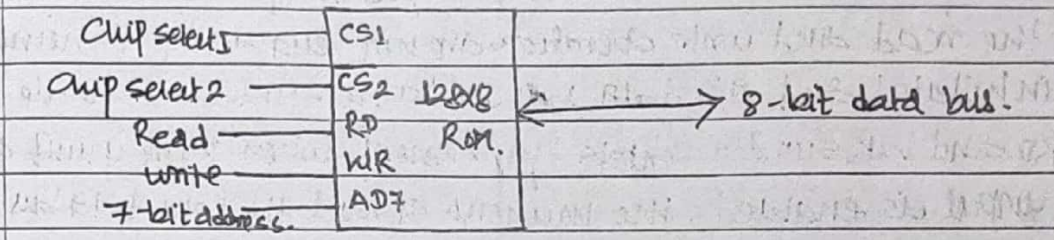


fig:- Block diagram of typical RAM chip.

Here the capacity of the memory is 128 words of eight bit per word. This requires 7-bit address and 8-bit bidirectional data bus that allow the transfer of data between cpu and memory. The read and write input specify the memory operation and chip select control inputs are for enabling the chip only when it is selected by the CPU. The following function table specify operation of RAM chip:-

CS_1	$\overline{CS_2}$	RD	WR	Memory function	State of data bus
0	0	X	X	Inhibit	High-impedance.
0	1	X	X	Inhibit	" "
1	0	0	0	inhibit	" "
1	0	0	1	write	Input data to RAM.
1	0	1	X	Read	output data from RAM
1	1	X	X	inhibit	High-Impedance.

fig: function table for RAM chip.

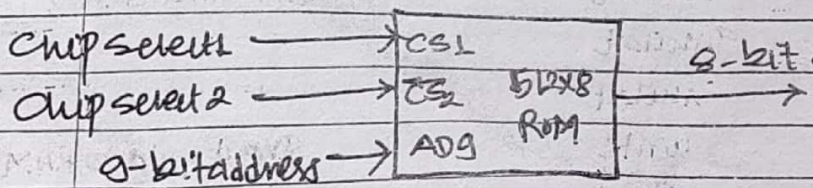
~~The unit is~~

Note:-

- The high impedence state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.
- The unit is in operation only when $CS1 = 1$ and $\overline{CS_2} = 0$.
- If the chip select input are not enabled or if they are enabled but the read and write operation are not enabled, the memory is inhibited and it's data bus in high-impedence state.
- The RD and WR signals controls the memory. When memory unit and WR input is enabled, the memory stores byte from data bus into location specified by the address input line.
- When memory unit and RD input is enabled, the content of selected byte is placed on to the data bus.

ROM chip:-

A ROM chip is organized externally in a similar manner as an the RAM chip. However, since ROM can only read, the data bus can only be in output mode. The block diagram of ROM chip is shown in figure below:-



- The 9-bit address line in the ROM chip specify any one of the 512 byte stored in it.

- The two chip select inputs must be CS1=1 and CS2=0 for the unit to operate otherwise data bus is in high-impedence state.
- There is no need for a read or write control because the unit can only read.
- Thus when the chip is enabled by two select inputs, the byte selected by the address line appears on the data bus.

Memory Address Map:-

The addressing of memory can be established by means of table that specifies the memory address assigned to each chip. The table is called memory address map. It is a pictorial representation of assigned address space for each chip in the system.

Example:

Assume that computer system needs 512 bytes of RAM and 512 bytes of ROM. Let RAM and ROM chip used are of 128x8 and 512x8 respectively. The memory address map for this configuration is:

		Address Bus									
Component	Hexadecimal	10	9	8	7	6	5	4	3	2	1
RAM1	0000 - 007F	0	0	0	x	x	x	x	x	x	x
RAM2	0080 - 00FF	0	0	1	x	x	x	x	x	x	x
RAM3	0100 - 017F	0	1	0	x	x	x	x	x	x	x
RAM4	0180 - 01FF	0	1	1	x	x	x	x	x	x	x
ROM	0200 - 03FF	1	x	x	x	x	x	x	x	x	x

- here Component column specify weather RAM or ROM chip is used.
- The hexadecimal address column specifies range of address for each chip
- The address bus lines are listed in the third column. Although there are 16 lines in the address bus, the table only shows 10 lines because

6 are not used in this example.

- The RAM chip has 128 bytes and needs seven address lines and ROM chip has 512 bytes and needs 9 address lines. The distinction between RAM and ROM chip is made by line 10. When line 10 is 1, it select the ROM and when it is 0, CPU select RAM.
- x's represent a binary number ranging from all 0's to all 1's.

Memory CPU connection:-

The RAM and ROM chips are connected to a CPU through the data and address buses. The connection of memory chips to the CPU is shown on figure below:-

This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. Each RAM receives the seven low order bits of the address bus to select one of 128 position bytes. The particular RAM chip selected is determined from line 8 and 9 in the address bus. This is done through 2x4 decoder. The selection between RAM and ROM chip is achieved through bus line 10. The RAM is selected when the bit in the line is 0 and ROM when the bit is 1. The data bus of ROM has only an output capability, whereas data bus connected to the RAM can transfer information in both direction.

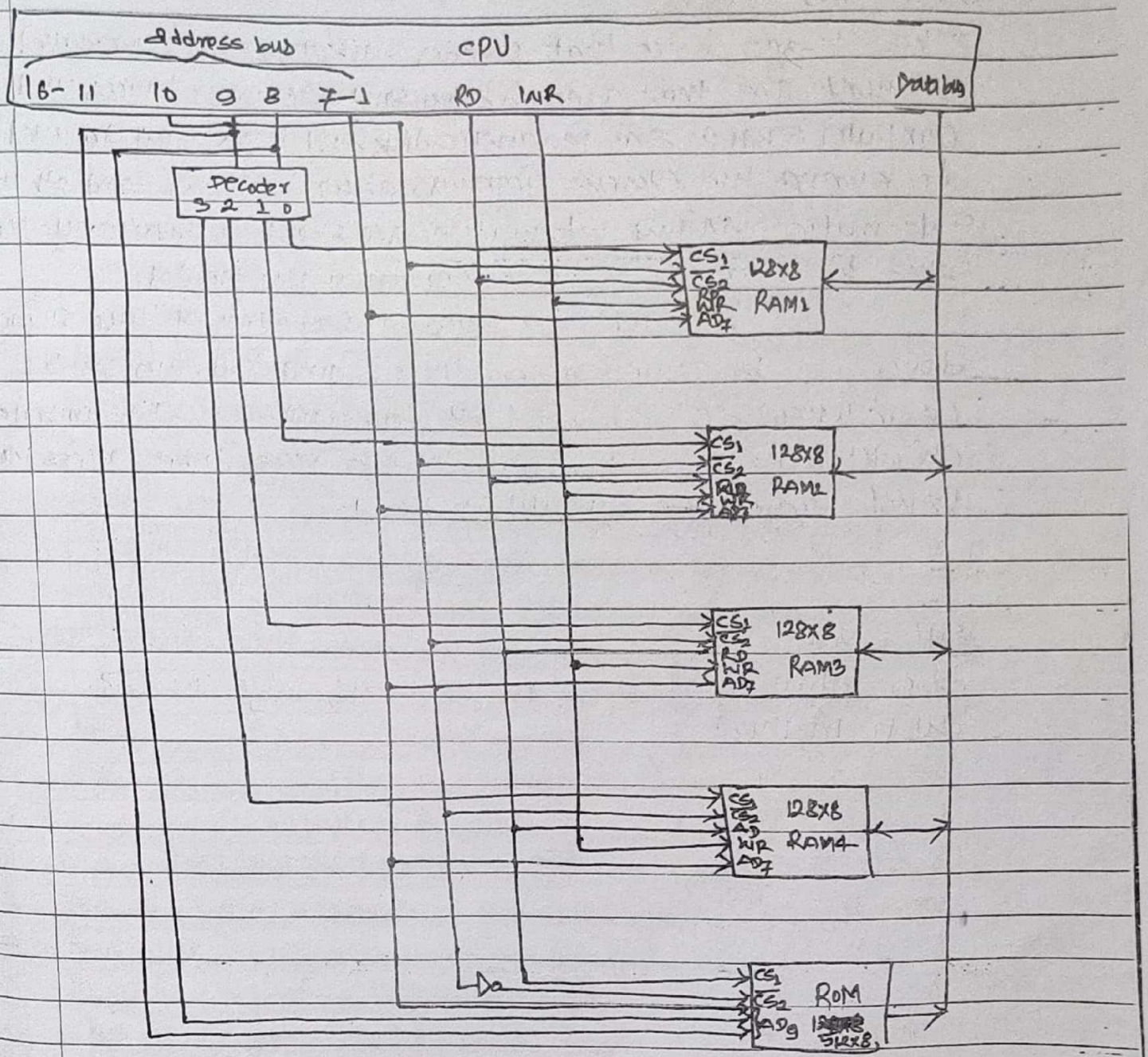


fig:- Memory connection to cpu.

This example gives an indication of the interconnection complexity that can exist between Memory chips and cpu. The designer must establish a memory map that assigns addresses to the various chips from which the require connections are determined.

Auxiliary Memory:-

The storage device that provides backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer system are magnetic disk and tapes. They are used for storing the system programs, larger data files and other backup information. Many information are stored in auxiliary memory and transfered into main memory when needed.

Although the physical properties of these storage device can be quite complex, their logical properties can be characterized and compared by few parameters. The important characteristics of any device are its access time, access mode, transfer rate ~~and~~ capacity and cost.

Self Study:-

Associative memory } Refer book, page no: 468.
cache memory }

Virtual Memory:-

In memory hierarchy system, programs and data are first stored in auxiliary memory. portion of a program or data are brought in to main memory as they are needed by the cpu.

virtual memory is a concept used in some large computer system that permit the user to construct program larger than main memory space available, equal to totality of auxiliary memory. Virtual memory is used to give a programmer the illusion that they have a very large memory even though computer actually has a relatively small main memory. A virtual memory system provides a mechanism for translating program generated address in to correct main memory locations. This is done dynamically while programs are being executed in the cpu. The translation or mapping is handled automatically by the hardware by means of mapping table.

Address space and memory space:-

- An address used by the programmer is called virtual address and set of such addresses is called address space.
- An address in main memory is called location or physical address. The set of such location or physical addresses is called memory space.
- Thus, the address space is the set of addresses generated by programs as they reference instruction and data but Memory space consist of actual main memory location directly addressable for processing. The address space is allowed to be larger than memory space in computers with virtual memory.

Example: Consider a computer with main memory capacity of 32KB which is equal to 2^{15} byte and auxiliary memory of size 1024KB $= 2^{20}$ byte. Thus we need 15 bit to address the physical memory and 20 bit for virtual memory.

here, address space N = 1024-k
and memory space M = 32k.
In multiprogramming computer system, program and data are transfered to and from auxiliary memory and main memory based on the demand imposed by the cpu.

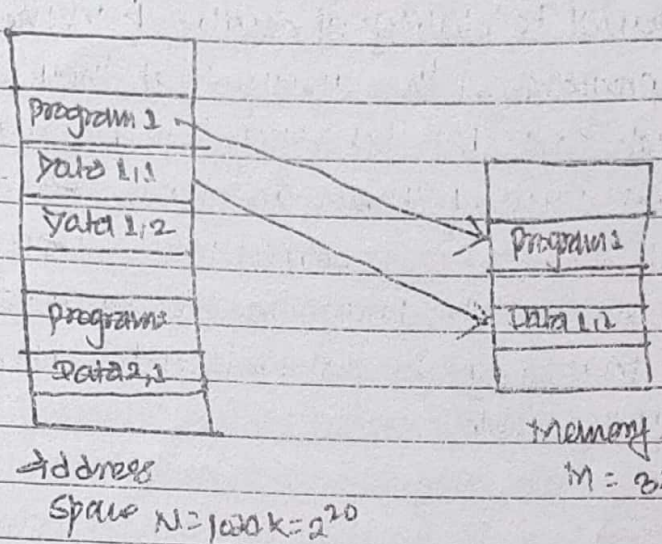


Fig:- Relation between address and memory space in virtual memory system

In a virtual memory system, the address field of instruction code has sufficient number of bits to specify all virtual addresses. In our example, the address field of an instruction code will consist of 20 bit, but physical memory addresses must be specified with only 15 bits. So the table is needed to map the virtual address of 20 bits to physical address of 15 bits. The mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced by cpu.

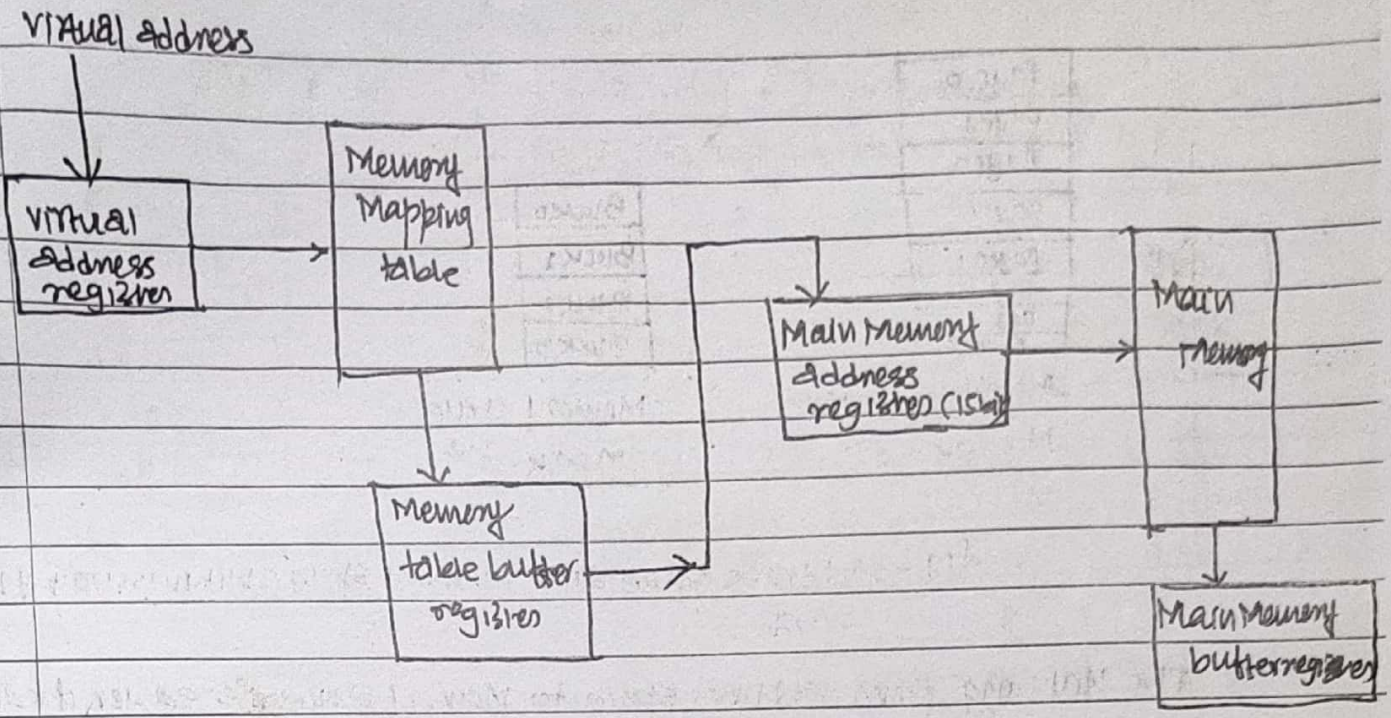
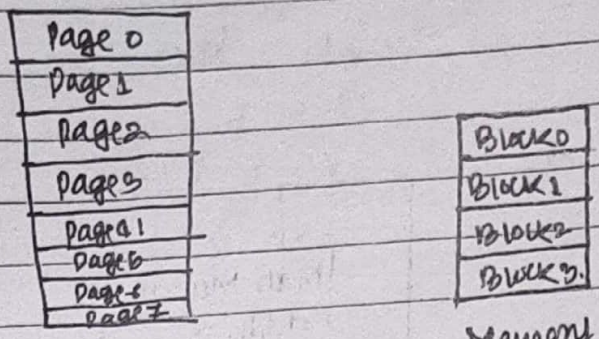


fig:- Memory table for mapping a virtual address.

Address Mapping using paging:-

- The table implementation of address mapping is simplified if the information in the address space and memory space are divided in to fixed size block.
- The physical memory is broken down in to group of equal size block, called page frame and virtual memory is subdivided in to same size block as the page frame called pages. The page refers to the group of address space of same size.

Consider a computer with an address space of 8KB and memory space of 4KB. If we split each in to group of 1KB words we obtain eight pages and 4 page frame or block as in figure below:-



Address space
$N = 8K = 2^{13}$

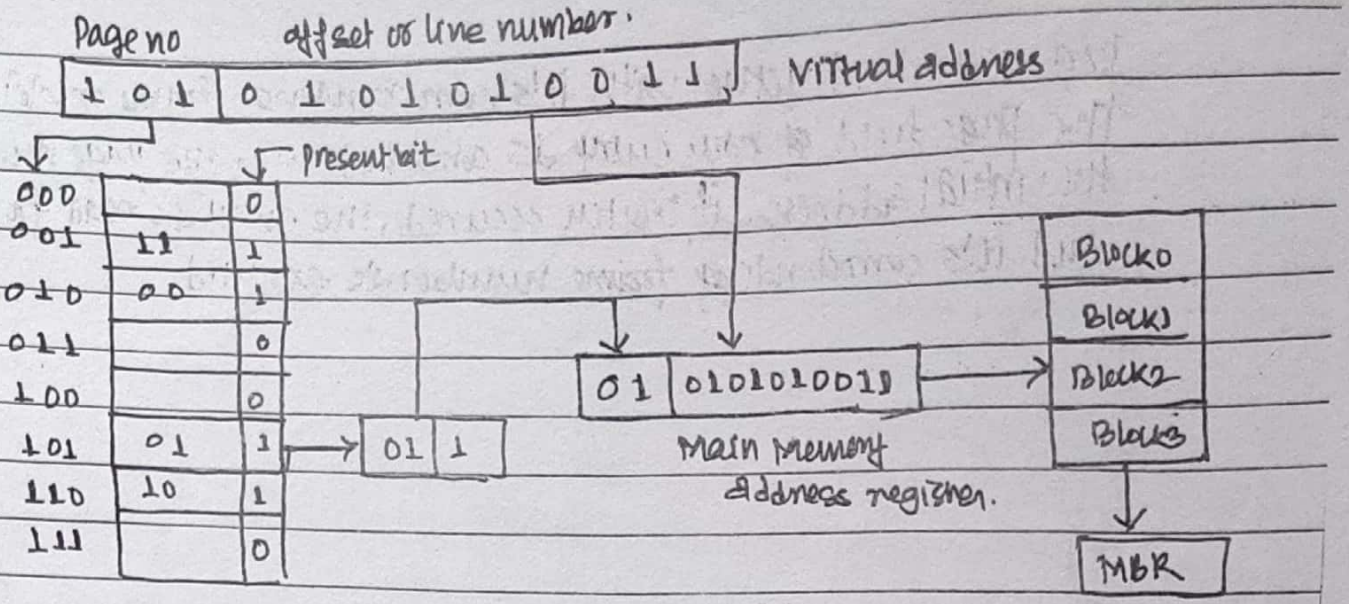
Memory space
$M = 4K = 2^{12}$.

fig:- Address space and memory space split into group of 1KB word

The mapping from address space to memory space is easier if each virtual address is considered to be represented by two numbers: page number and offset or line with in the page. In a computer with 2^p words per page, p bits are used to specify offset and remaining high order bits of virtual address specify page number.

So in our example, virtual address has 13 bit, since each page consist of $2^{10} = 1024$ words, the higher order 3 bit gives page number and lower order 10 bit gives offset. The process of memory mapping in paged system is shown in figure below:-

P.T.O



Memory page table.

Eg:- Memory table in paged system.

The memory page table consist of a word, one for each page. The address in page table denotes page number and content of each entries gives frame number or block number where page is stored in main memory.
The present or absent bit indicates whether the page has been transfered from auxiliary memory in to main memory.

Associative Memory page table:

A Random access memory page table is inefficient with respect to storage utilization because we need one entry for each page in the page table.
A more efficient way to organize the page table would be to construct it with a number of entry equal to number of blocks in main memory. in this way size of memory is reduced to store page table and each location is fully utilized. This method can be implemented by means of associative memory in which each entry in memory containing

page number together with it's corresponding frame or block number. The page field of each entry is compared with the page number in the virtual address. If match occured, the entry is read from memory and it's corresponding frame number is extracted.

Page replacement:-

- A virtual Memory system is a combination of hardware and software techniques.
- The Memory Management system software handles all the software operation for the efficient utilization of Memory space. It must decide
  1) Which page in the memory to be removed to make a room for new page.
  2) When a new page is to be transfered from auxiliary Memory to Main memory
  3) where the page is to be placed in main memory.

- When the program attempt to reference a page that is not in memory, page fault occurs. When page fault occurs, it signifies that the page referenced by the CPU is not in main memory. A new page is then transfered from auxiliary memory to main Memory. If main memory is full, it would be necessary to remove page from memory to make a room for the new page. The policy for choosing the page to remove is called page replacement algorithm. The goal of replacement policy is to try to

remove the page least likely to be referenced in the immediate future. The two most common replacement algorithm used are:-

i) First in first out (FIFO)
ii) least recently used (LRU).

i) FIFO :-

The FIFO algorithm select a page for replacement that has been in memory for longest time

ii) LRU:

LRU algorithm select a page for replacement that has been least recently used. Recently used pages are not removed but least recently used pages are replaced by this algorithm.

iv) Memory management hardware:-

A memory management system is a collection of hardware and software procedures for managing the various programs residing in memory. The memory management software is part of overall operating system available in many computers it is called memory manager. The hardware which perform memory management is called MMU. The basic component of memory management unit are:-

i) A facility for dynamic storage relocation that maps logical memory reference to physical memory addresses.
ii) A provision for sharing common programs stored in memory by different users.
iii) protection of information against unauthorized access between

remove the page least likely to be referenced in the immediate future. The two most common replacement algorithm used are:-

i) First in first out (FIFO)
ii) least recently used (LRU).

i) FIFO :-
The FIFO algorithm select a page for replacement that has been in memory for longest time

ii) LRU:
LRU algorithm select a page for replacement that has been least recently used. Recently used pages are not removed but least recently used pages are replaced by this algorithm.

v) Memory management hardware:-
A memory management system is a collection of hardware and software procedures for managing the various programs residing in memory. The memory management software is part of overall operating system available in many computers it called memory manager. The hardware which perform memory management is called MMU. The basic component of memory management unit are:-
i) A facility for dynamic storage relocation that maps logical memory reference to physical memory addresses.
ii) A provision for sharing common programs stored in memory by different users.
iii) protection of information against unauthorized access between

user and preventing users from changing operating system function.

Segment:-

- The fixed page size used in the virtual memory system causes certain difficulties with respect to program size and the logical structure of program.
- It is more convinient to divide program and data in to logical parts called segment.
- A segment is a set of logically related instructions or data element associated with a given name. Segment may be generated by the operating system or programmer. example of segments are subroutine, an array, symbol table etc.
- The address generated by segmented program is called logical address which is associated with variable length segment.
- The logical address is mapped in to physical address by memory management unit. The mapping between logical address in to physical address under segmentation is shown in figure below:-

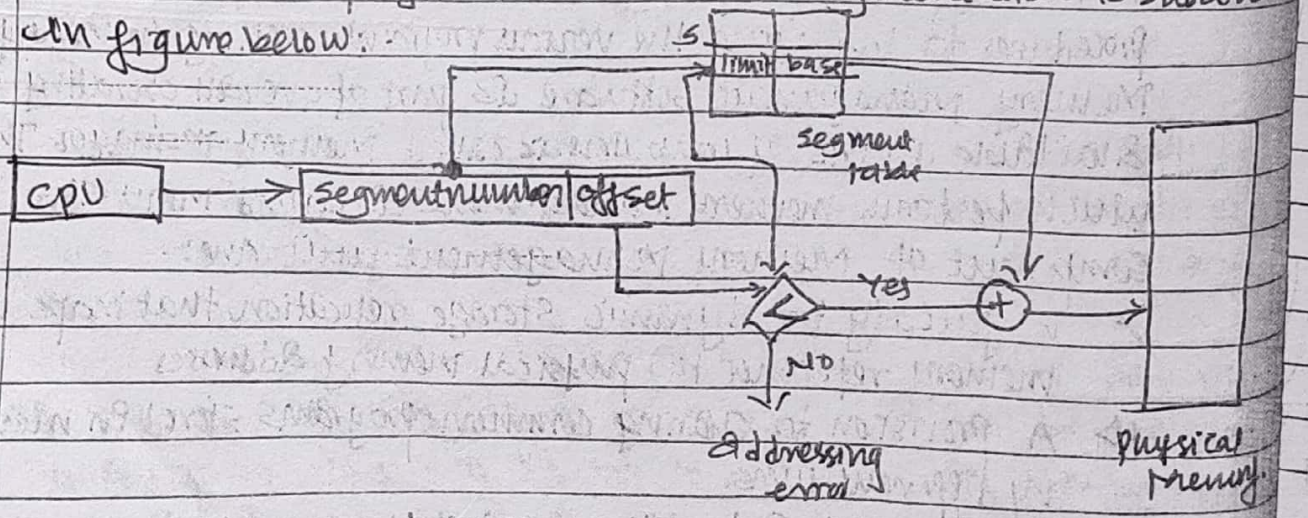


fig: Segmentation hardware.

Segmented page mapping :-

- In segmentation with paging the logical address is partitioned in to three field segment no, page no and word offset.
- The segment no specifies the segment number to which logical address belong, page number specifies page number with in the segment and offset field specifies word with in the page.
- The length of segment would vary according to the no of page with in it.
- The mapping of the logical address in to physical address is done by means of two tables as shown in figure below:-

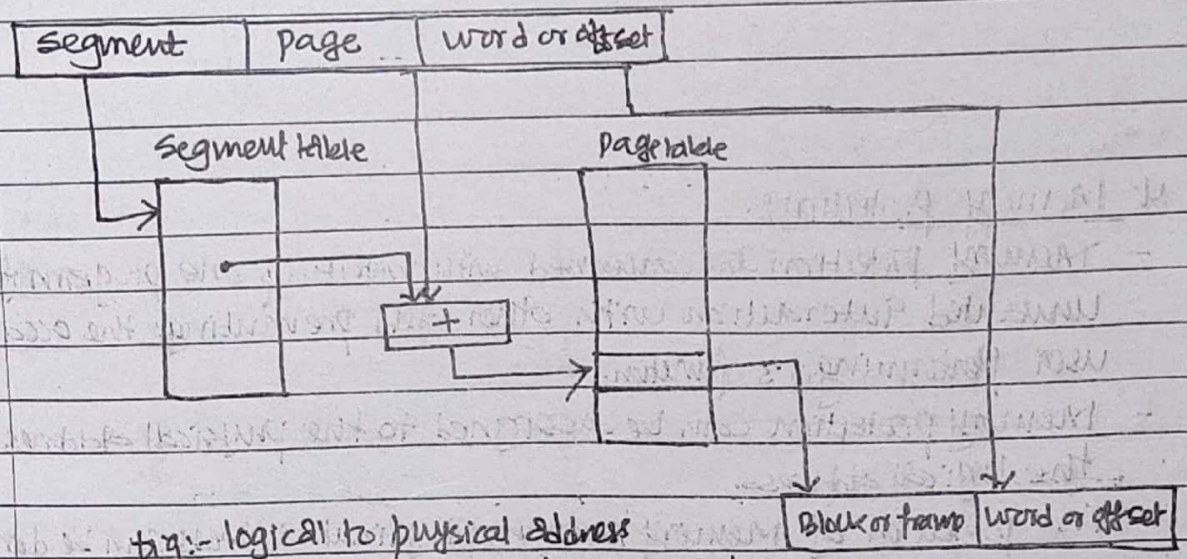


fig:- logical to physical address Mapping in segmentation with paging.

- The segment number of the logical address specifies the address for segment table. The entry in the segment table is a pointer address for page table base. The page table base is added to page number given in the logical address. The sum produces a pointer address to entry in page table. The value found in page table provide frame number or block number in physical memory which is concatenated with offset to produce the physical address.

physical mapped address.

The two mapping tables may be stored in two seperate small memories or in main memory. In either case, a memory reference from CPU will require three accesses to memory. one from the segment table, one from the page table and third from main memory. This would slow the system significantly compared to conventional system that requires only one reference to memory.

To avoid this speed penalty, a fast associative memory is used to hold the most recently referenced table entries. This type of memory is called translation lookaside buffer.

Memory protection:-

- Memory protection is concerned with protecting one program from unwanted interaction with other and preventing the occasional user performing o.s function.
- Memory protection can be assigned to the physical address or the logical address
- The protection of memory through physical address can be done by assigning to each block in memory a number of protection bit that indicate the type of access allowed to it's corresponding block.
- A much better place to apply protection is in the logical address space rather than physical address space. This can be done by including protection information with in a segment table or segment register of the memory management hardware

The content of each entry in the segment table or segment registers is called a discriptor. The typical formate of segment discriptor is:

Base address	length	protection

fig:- format of typical segment discriptor.

- The base address gives the base of the page table address in segmented page organization.
- The length field gives the segment size by specifying the maximum number of pages assigned to the segment. The length field is compared against the page number in the logical address. A size violation occurs if page number fall outside segment length boundary.
- The protection field specifies the access rights available to particular segment. In segmented page organization, each entry in page table may have it's own protection field to describe the access right of each page. some of the access rights of interest that are used for protecting the program residing in memory are:
  1) Read and write.
  2) Read only
  3) execute only
  4) system only.