- Introduction about digital system
- Philosophy of number systems
- Complement representation of negative numbers
- Binary arithmetic
- Binary codes
- Error detecting & error correcting codes
- Hamming codes

INTRODUCTION ABOUT DIGITAL SYSTEM

A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.

Now a day's digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

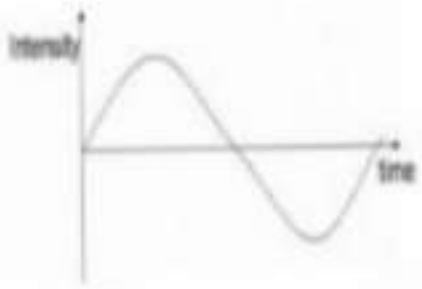
Characteristics of Digital systems

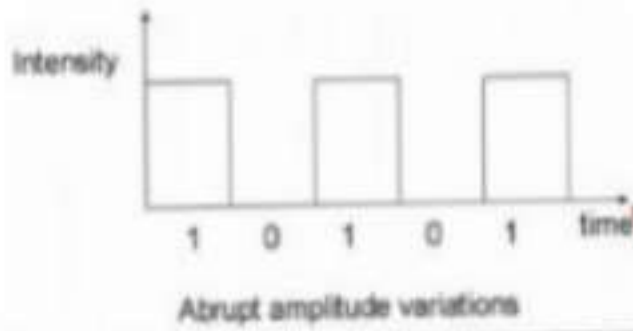
- Digital systems manipulate discrete elements of information.
- Discrete elements are nothing but the digits such as 10 decimal digits or 26 letters of alphabets and so on.
- Digital systems use physical quantities called signals to represent discrete elements.
- In digital systems, the signals have two discrete values and are therefore said to be binary.
- A signal in digital system represents one binary digit called a bit. The bit has a value either 0 or 1.

Analog systems vs Digital systems

Analog system process information that varies continuously i.e; they process time varying signals that can take on any values across a continuous range of voltage, current or any physical parameter.

Digital systems use digital circuits that can process digital signals which can take either 0 or 1 for binary system.





Abrupt amplitude variations

Advantages of Digital system over Analog system

1. Ease of programmability

The digital systems can be used for different applications by simply changing the program without additional changes in hardware.

2. Reduction in cost of hardware

The cost of hardware gets reduced by use of digital components and this has been possible due to advances in IC technology. With ICs the number of components that can be placed in a given area of Silicon are increased which helps in cost reduction.

3.High speed

Digital processing of data ensures high speed of operation which is possible due to advances in Digital Signal Processing.

4. High Reliability

Digital systems are highly reliable one of the reasons for that is use of error correction codes.

5. Design is easy

The design of digital systems which require use of Boolean algebra and other digital techniques is easier compared to analog designing.

6. Result can be reproduced easily

Since the output of digital systems unlike analog systems is independent of temperature, noise, humidity and other characteristics of components the reproducibility of results is higher in digital systems than in analog systems.

Disadvantages of Digital Systems

- Use more energy than analog circuits to accomplish the same tasks, thus producing more heat as well.
- Digital circuits are often fragile, in that if a single piece of digital data is lost or misinterpreted the meaning of large blocks of related data can completely change.
- Digital computer manipulates discrete elements of information by means of a binary code.
- Quantization error during analog signal sampling.

NUMBER SYSTEM

Number system is a basis for counting varies items. Modern computers communicate and operate with binary numbers which use only the digits 0 &1. Basic number system used by humans is Decimal number system.

For Ex: Let us consider decimal number 18. This number is represented in binary as 10010.

We observe that binary number system take more digits to represent the decimal number. For large numbers we have to deal with very large binary strings. So this fact gave rise to three new number systems.

i) Octal number systems

ii) Hexa Decimal number system

iii) Binary Coded Decimal number(BCD) system

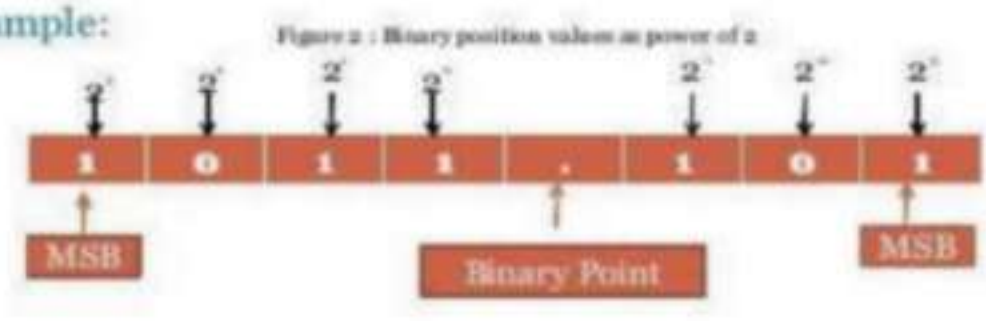
To define any number system we have to specify

- Base of the number system such as 2,8,10 or 16.
- The base decides the total number of digits available in that number system.
- First digit in the number system is always zero and last digit in the number system is always base-1.

Binary number system:

The binary number has a radix of 2. As r = 2, only two digits are needed, and these are 0 and 1. In binary system weight is expressed as power of 2.

- Example:



The left most bit, which has the greatest weight is called the Most Significant Bit (MSB). And the right most bit which has the least weight is called Least Significant Bit (LSB).

For Ex: $1001.01_2 = [(1) \times 2^3] + [(0) \times 2^2] + [(0) \times 2^1] + [(1) \times 2^0] + [(0) \times 2^{-1}] + [(1) \times 2^2]$

$1001.01_2 = [1 \times 8] + [0 \times 4] + [0 \times 2] + [1 \times 1] + [0 \times 0.5] + [1 \times 0.25]$

$1001.01_2 = 9.25_{10}$

Decimal Number system

The decimal system has ten symbols: 0,1,2,3,4,5,6,7,8,9. In other words, it has a base of 10.

Octal Number System

Digital systems operate only on binary numbers. Since binary numbers are often very long, two shorthand notations, octal and hexadecimal, are used for representing large binary numbers. Octal systems use a base or radix of 8. It uses first eight digits of decimal number system. Thus it has digits from 0 to 7.

Hexa Decimal Number System

The hexadecimal numbering system has a base of 16. There are 16 symbols. The decimal digits 0 to 9 are used as the first ten digits as in the decimal system, followed by the letters A, B, C, D, E and F, which represent the values 10, 11,12,13,14 and 15 respectively.

Decimal	Binary	Octal	Hexadecimal
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Number Base conversions

The human beings use decimal number system while computer uses binary number system. Therefore it is necessary to convert decimal number system into its equivalent binary.

i) Binary to octal number conversion
ii) Binary to hexa decimal number conversion

The binary number:	001	010	011	000	100	101	110	111
The octal number:	1	2	3	0	4	5	6	7

The binary number:	0001	0010	0100	1000	1001	1010	1101	1111
The hexadecimal number:	1	2	5	8	9	A	D	F

iii) Octal to binary Conversion

Each octal number converts to 3 binary digits

Code
0 - 000
1 - 001
2 - 010
3 - 011
4 - 100
5 - 101
6 - 110
7 - 111

To convert 653_8 to binary, just substitute code:

6 5 3
↓ ↓ ↓
110 101 011

iv) Hexa to binary conversion

0100 1111 1101 0111

v) Octal to Decimal conversion

Ex: convert 4057.06_8 to octal

$=4x8^3+0x8^2+5x8^1+7x8^0+0x8^{-1}+6x8^{-2}$

$=2048+0+40+7+0+0.0937$

$=2095.0937_{10}$

vi) Decimal to Octal Conversion

Ex: convert 378.93_{10} to octal

378_{10} to octal: Successive division:

```
8 | 378
  |___
8 |47 ---  2
  |_____
8 |5  ---  7      ↑
  |_____
  0  ---  5
```

$=572_8$

0.93_{10} to octal :

0.93x8=7.44
0.44x8=3.52 ↓
0.53x8=4.16
0.16x8=1.28

$=0.7341_8$

$378.93_{10}=572.7341_8$

vii) Hexadecimal to Decimal Conversion

Ex: $5C7_{16}$ to decimal

$=(5x16^2)+(C\ x16^1)+(7\ x16^0)$

$=1280+192+7$

$=147_{10}$

viii) Decimal to Hexadecimal Conversion

Ex: 2598.67510

```
1 6 2598
    |___
16 162      -6
     10     -2
```

$= A26_{(16)}$

0.67510=0.675x16 -- 10.8

=∃.800x16 -- 12.8 ↓
=∃.800x16 -- 12.8
=∃.800x16 -- 12.8
=0.$ACCC_{16}$

2598.675_{10}= A26.$ACCC_{16}$

ix) Octal to hexadecimal conversion:

The simplest way is to first convert the given octal no. to binary & then the binary no. to hexadecimal.

Ex: 756.603_8

7	5	6	.	6	0	3
111	101	110	.	110	000	011
0001	1110	1110	.	1100	0001	1000
1	E	E	.	C	1	8

x) Hexadecimal to octal conversion:

First convert the given hexadecimal no. to binary & then the binary no. to octal.

Ex: B9F.AE16

B	9	F	.	A	E		
1011	1001	1111	.	1010	1110		
101	110	011	111	.	101	011	100
5	6	3	7	.	5	3	4

=5637.534

Complements:

In digital computers to simplify the subtraction operation & for logical manipulation complements are used. There are two types of complements used in each radix system.

i) The radix complement or r's complement
ii) The diminished radix complement or (r-1)'s complement

Representation of signed no.s binary arithmetic in computers:

- Two ways of rep signed no.s
    1. Sign Magnitude form
    2. Complemented form
- Two complimented forms
    1. 1's compliment form
    2. 2's compliment form

Advantage of performing subtraction by the compliment method is reduction in the hardware.(instead of addition & subtraction only adding ckt's are needed.)
i.e, subtraction is also performed by adders only.

Instead of subtracting one no. from other the compliment of the subtrahend is added to minuend. In sign magnitude form, an additional bit called the sign bit is placed in front of the no. If the sign bit is 0, the no. is +ve, If it is a 1, the no is _ve.

Ex:

0	1	0	1	0	0	1

↓

Sign bit =+41 magnitude

↑

1	1	0	1	0	0	1

= -41

Note: manipulation is necessary to add a +ve no to a –ve no

Representation of signed no.s using 2's or 1's complement method:

If the no. is +ve, the magnitude is rep in its true binary form & a sign bit 0 is placed in front of the MSB.I f the no is _ve , the magnitude is rep in its 2's or 1's compliment form &a sign bit 1 is placed in front of the MSB.

Ex:

Given no.	Sign mag form	2's comp form	1's comp form
01101	+13	+13	+13
010111	+23	+23	+23
10111	-7	-7	-8
1101010	-42	-22	-21

Special case in 2's comp representation:

Whenever a signed no. has a 1 in the sign bit & all 0's for the magnitude bits, the decimal equivalent is -2^n , where n is the no of bits in the magnitude .
Ex: 1000= -8 & 10000=-16

Characteristics of 2's compliment no.s:

Properties:

1. There is one unique zero
2. 2's comp of 0 is 0
3. The leftmost bit can't be used to express a quantity . it is a 0 no. is +ve.
4. For an n-bit word which includes the sign bit there are $(2^{n-1}-1)$ +ve integers, 2^{n-1} –ve integers & one 0 , for a total of 2^n uniquestates.
5. Significant information is containd in the 1's of the +ve no.s & 0's of the _ve no.s
6. A _ve no. may be converted into a +ve no. by finding its 2's comp.

Signed binary numbers:

Decimal	Sign 2's comp form	Sign 1's comp form	Sign mag form
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0011	0011	0011
+0	0000	0000	0000

-0	--	1111	1000
-1	1111	1110	1001
-2	1110	1101	1010
-3	1101	1100	1011
-4	1100	1011	1100
-5	1011	1010	1101
-6	1010	1001	1110
-7	1001	1000	1111
8	1000	--	--

Methods of obtaining 2's comp of a no:

- In 3 ways
  1. By obtaining the 1's comp of the given no. (by changing all 0's to 1's & 1's to 0's) & then adding 1.
  2. By subtracting the given n bit no N from 2^n
  3. Starting at the LSB , copying down each bit upto & including the first 1 bit encountered , and complimenting the remaining bits.

Ex: Express -45 in 8 bit 2's comp form

+45 in 8 bit form is 00101101

I method:

1's comp of 00101101 & the add 1

00101101
11010010
+1

— – – – – – – – – – – –

11010011 is 2's comp form

II method:

Subtract the given no. N from 2^n

2^n = 100000000
Subtract 45= -00101101
+1

— – – –

11010011 is 2's comp

III method:

Original no: 00101101
Copy up to First 1 bit 1
Compliment remaining : 1101001

bits 11010011

Ex:

-73.75 in 12 bit 2'compform

I method

01001001.1100

10110110.0011

+1

10110110.0100 is 2's

II method:

2^8 = 100000000.0000

Sub 73.75=-01001001.1100

10110110.0100 is 2's comp

III method :

Orginalno : 01001001.1100

Copy up to 1'st bit 100

Comp the remaining bits: 10110110.0

10110110.0100

2's compliment Arithmetic:

- The 2's comp system is used to rep –ve no.s using modulus arithmetic . The word length of a computer is fixed. i.e, if a 4 bit no. is added to another 4 bit no . the result will be only of 4 bits. Carry if any , from the fourth bit will overflow called the Modulus arithmetic.

  Ex:1100+1111=1011

- In the 2's compl subtraction, add the 2's comp of the subtrahend to the minuend . If there is a carry out , ignore it , look at the sign bit I,e, MSB of the sum term .If the MSB is a 0, the result is positive.& it is in true binary form. If the MSB is a ` (carry in or no carry at all) the result is negative.& is in its 2's comp form. Take its 2's comp to find its magnitude in binary.

Ex: Subtract 14 from 46 using 8 bit 2's comp arithmetic:

+14 = 00001110

-14 = 11110010 2's comp

+46 = 00101110

-14 =+11110010 2's comp form of -14

-32	(1)00100000	ignore carry

Ignore carry , The MSB is 0 . so the result is +ve. & is in normal binary form. So the result is +00100000=+32.

EX: Add -75 to +26 using 8 bit 2's comp arithmetic

+75	= 01001011	
-75	=10110101	2's comp
+26	= 00011010	
-75	=+10110101	2's comp form of -75
-49	11001111	No carry

No carry , MSB is a 1, result is _ve & is in 2's comp. The magnitude is 2's comp of 11001111. i.e, 00110001 = 49. so result is -49

Ex: add -45.75 to +87.5 using 12 bit arithmetic

+87.5 = 01010111.1000
-45.75=+11010010.0100

-41.75	(1)00101001.1100 ignore carry

MSB is 0, result is +ve. =+41.75

1's compliment of n number:

- It is obtained by simply complimenting each bit of the no,.& also , 1's comp of a no, is subtracting each bit of the no. form 1.This complemented value rep the – ve of the original no. One of the difficulties of using 1's comp is its rep o f zero. Both 00000000 & its 1's comp 11111111 rep zero.
- The 00000000 called +ve zero& 11111111 called –ve zero.

Ex: -99 & -77.25 in 8 bit 1's comp

+99	=	01100011
-99	=	10011100
+77.25 =		01001101.0100
-77.25 =		10110010.1011

1's compliment arithmetic:

In 1's comp subtraction, add the 1's comp of the subtrahend to the minuend. If there is a carryout , bring the carry around & add it to the LSB called the **end around carry.** Look at the sign bit (MSB) . If this is a 0, the result is +ve & is in true binary. If the MSB is a 1 (carry or no carry), the result is –ve & is in its is comp form .Take its 1's comp to get the magnitude inn binary.

Ex: Subtract 14 from 25 using 8 bit 1's EX: ADD -25 to +14

25	=	00011001	+14	= 00001110	
-45	=	11110001	-25	=+11100110	
+11		(1)00001010	-11	11110100	

+1

No carry MSB =1

00001011 result=-ve=-11_{10}

MSB is a 0 so result is +ve (binary)

$=+11_{10}$

Binary codes

Binary codes are codes which are represented in binary system with modification from the original ones.

- Weighted Binary codes
- Non Weighted Codes

Weighted binary codes are those which obey the positional weighting principles, each position of the number represents a specific weight. The binary counting sequence is an example.

Decimal	BCD 8421	Excess-3	84-2-1	2421	5211	Bi-Quinary 5043210			5	0	4	3	2	1	0
0	0000	0011	0000	0000	0000	0100001		0		X					X
1	0001	0100	0111	0001	0001	0100010		1		X				X	
2	0010	0101	0110	0010	0011	0100100		2		X			X		
3	0011	0110	0101	0011	0101	0101000		3		X		X			
4	0100	0111	0100	0100	0111	0110000		4		X	X				
5	0101	1000	1011	1011	1000	1000001		5	X						X
6	0110	1001	1010	1100	1010	1000010		6	X					X	
7	0111	1010	1001	1101	1100	1000100		7	X				X		
8	1000	1011	1000	1110	1110	1001000		8	X			X			
9	1001	1111	1111	1111	1111	1010000		9	X		X				

Reflective Code

A code is said to be reflective when code for 9 is complement for the code for 0, and

so is for 8 and 1 codes, 7 and 2, 6 and 3, 5 and 4. Codes 2421, 5211, and excess-3 are reflective, whereas the 8421 code is not.

Sequential Codes

A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. This greatly aids mathematical manipulation of data. The 8421 and Excess-3 codes are sequential, whereas the 2421 and 5211 codes are not.

Non weighted codes

Non weighted codes are codes that are not positionally weighted. That is, each position within the binary number is not assigned a fixed value. Ex: Excess-3 code

Excess-3 Code

Excess-3 is a non weighted code used to express decimal numbers. The code derives its name from the fact that each binary code is the corresponding 8421 code plus 0011(3).

Gray Code

The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit- distance code. In digital Gray code has got a special place.

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

Binary to Gray Conversion

- Gray Code MSB is binary code MSB.
- Gray Code MSB-1 is the XOR of binary code MSB and MSB-1.
- MSB-2 bit of gray code is XOR of MSB-1 and MSB-2 bit of binary code.
- MSB-N bit of gray code is XOR of MSB-N-1 and MSB-N bit of binary code.

8421 BCD code (Natural BCD code):

Each decimal digit 0 through 9 is coded by a 4 bit binary no. called natural binary codes. Because of the 8,4,2,1 weights attached to it. It is a weighted code & also sequential . it is useful for mathematical operations. The advantage of this code is its case of conversion to & from decimal. It is less efficient than the pure binary, it require more bits.

Ex: 14→1110 in binary

But as 0001 0100 in 8421 ode.

The disadvantage of the BCD code is that , arithmetic operations are more complex than they are in pure binary . There are 6 illegal combinations 1010,1011,1100,1101,1110,1111 in these codes, they are not part of the 8421 BCD code system . The disadvantage of 8421 code is, the rules of binary addition 8421 no, but only to the individual 4 bit groups.

BCD Addition:

It is individually adding the corresponding digits of the decimal no,s expressed in 4 bit binary groups starting from the LSD . If there is no carry & the sum term is not an illegal code , no correction is needed .If there is a carry out of one group to the next group or if the sum term is an illegal code then $6_{10}(0100)$ is added to the sum term of that group & the resulting carry is added to the next group.

Ex: Perform decimal additions in 8421 code

(a)25+13

In BCD	25=	0010 0101
In BCD	+13 =	+0001 0011
	38	0011 1000

No carry , no illegal code .This is the corrected sum

(b). 679.6 + 536.8

679.6	=	0110	0111	1001	.0110 in BCD
+536.8	=	+0101	0011	0010	.1000 in BCD
1216.4		1011	1010	0110	. 1110 illegal codes
		+0110	+ 0011	+0110	. + 0110 add 0110 to each
		(1)0001	(1)0000	(1)0101	. (1)0100 propagate carry
		/	/	/	/
		+1	+1	+1	+1
		0001	0010	0001	0110 . 0100
		1	2	1	6 . 4

BCD Subtraction:

Performed by subtracting the digits of each 4 bit group of the subtrahend the digits from the corresponding 4- bit group of the minuend in binary starting from the LSD . if there is no borrow from the next group , then 6_{10}(0110)is subtracted from the difference term of this group.

(a)38-15

In BCD	38=	0011	1000
In BCD	-15 =	-0001	0101
	23	0010	0011

No borrow, so correct difference.

.(b) 206.7-147.8

206.7	=	0010	0000	0110	.	0111	in BCD
-147.8 =		-0001	0100	0111	.	0110	in BCD
58.9		0000	1011	1110	.	1111	borrows are present
	-0110	-0110	.			-0110	subtract 0110
				0101		1000 . 1001	

BCD Subtraction using 9's & 10's compliment methods:

Form the 9's & 10's compliment of the decimal subtrahend & encode that no. in the 8421 code . the resulting BCD no.s are then added.

EX: 305.5 – 168.8

305.5	=	305.5			
-168.8	=	+83.1			9's comp of -168.8
		— – –			
		(1)136.6			
		+1			end around carry
		136.7			corrected difference
305.5_{10}	=	0011 0000 0101	.	0101	
$+831.1_{10}$	=	+1000 0011 0001	.	0001	9's comp of $1_{68.8}$ in BCD
— – –		----------			
		+1011 0011 0110	.	0110	1011 is illegal code
		+0110			add 0110

---

(1)0001	0011	0110	.	0110	
				+1	End around carry

0001 0011 0110 . 0111

= 136.7

Excess three(xs-3)code:

It is a non-weighted BCD code .Each binary codeword is the corresponding 8421 codeword plus 0011(3).It is a sequential code & therefore , can be used for arithmetic operations..It is a self-complementing code.s o the subtraction by the method of compliment addition is more direct in xs-3 code than that in 8421 code. The xs-3 code has six invalid states 0000,0010,1101,1110,1111.. It has interesting properties when used in addition & subtraction.

Excess-3 Addition:

Add the xs-3 no.s by adding the 4 bit groups in each column starting from the LSD. If there is no carry starting from the addition of any of the 4-bit groups , subtract 0011 from the sum term of those groups (because when 2 decimal digits are added in xs-3 & there is no carry , result in xs-6). If there is a carry out, add 0011 to the sum term of those groups(because when there is a carry, the invalid states are skipped and the result is normal binary).

EX:	37	0110	1010	
	+28	+0101	1011	
	65	1011	(1)0101	carry generated
		+1		propagate carry
		1100	0101	add 0011 to correct 0101 &
		-0011	+0011	subtract 0011 to correct 1100
		1001	1000	$=65_{10}$

Excess -3 (XS-3) Subtraction:

Subtract the xs-3 no.s by subtracting each 4 bit group of the subtrahend from the corresponding 4 bit group of the minuend starting form the LSD .if there is no borrow from the next 4-bit group add 0011 to the difference term of such groups (because when decimal digits are subtracted in xs-3 & there is no borrow , result is normal binary). I f there is a borrow , subtract 0011 from the differenceterm(b coz taking a borrow is equivalent to adding six invalid states , result is in xs-6)

Ex: 267-175

267 =	0101	1001	1010
-175=	-0100	1010	1000
	0000	1111	0010
	+0011	-0011	+0011
	0011	1100	+0011 $=92_{10}$

Xs-3 subtraction using 9's & 10's compliment methods:

Subtraction is performed by the 9's compliment or 10's compliment

Ex:687-348 The subtrahend (348) xs -3 code & its compliment are:

9's comp of 348 = 651

Xs-3 code of 348 = 0110 0111 1011

1's comp of 348 in xs-3 = 1001 1000 0100

Xs=3 code of 348 in xs=3 = 1001 1000 0100

687		687
-348	→	+651 9's compl of 348
339		(1)338
		+1 end around carry
		339 corrected difference in decimal

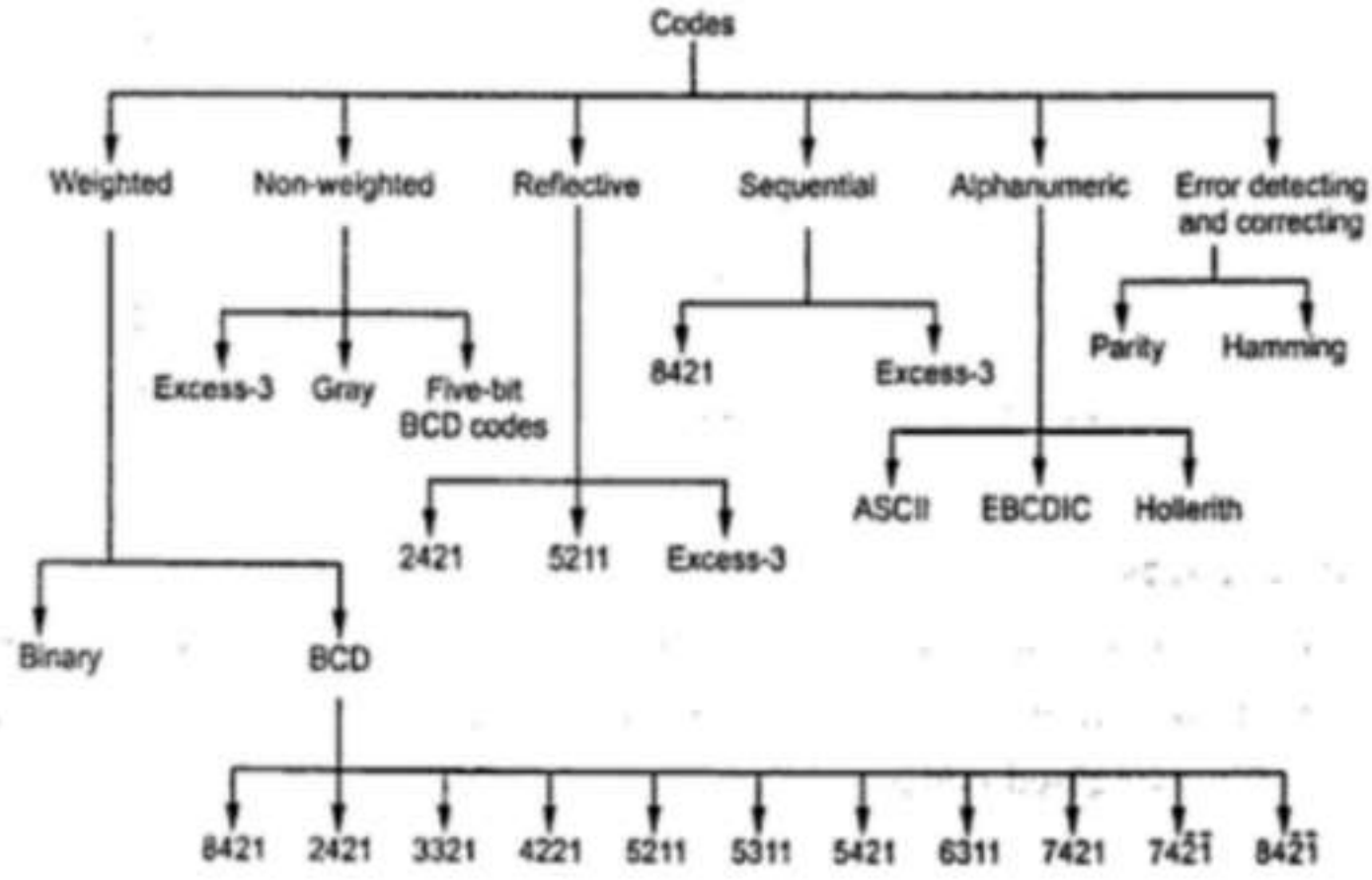
1001	1011	1010	687 in xs-3
+1001	1000	0100	1's comp 348 in xs-3
_ (1)0010	(1)0011	1110	carry generated
+1	+1		propagate carry
(1)0011	0010	1110	
		+1	end around carry
0011	0011	1111	(correct 1111 by sub0011 and
+0011	+0011	+0011	correct both groups of 0011 by
			adding 0011)
0110	0110	1100	corrected diff in xs-3 = 330_{10}

The Gray code (reflective –code):

Gray code is a non-weighted code & is not suitable for arithmetic operations. It is not a BCD code . It is a cyclic code because successive code words in this code differ in one bit position only i.e, it is a unit distance code.Popular of the unit distance code.It is also a reflective code i.e,both reflective & unit distance. The n least significant bits for 2^n through 2^{n+1}-1 are the mirror images of thosr for 0 through 2^n-1.An N bit gray code can be obtained by reflecting an N-1 bit code about an axis at the end of the code, & putting the MSB of 0 above the axis & the MSB of 1 below the axis.

Reflection of gray codes:

Gray Code				Decimal	4 bit binary
1 bit	2 bit	3 bit	4 bit		
0	00	000	0000	0	0000
1	01	001	0001	1	0001
	11	011	0011	2	0010
	10	010	0010	3	0011
		110	0110	4	0100
		111	0111	5	0101
		101	0101	6	0110
		110	0100	7	0111
			1100	8	1000
			1101	9	1001
			1111	10	1010
			1110	11	1011
			1010	12	1100
			1011	13	1101
			1001	14	1110
			1000	15	1111



Binary codes block diagram

Error – Detecting codes: When binary data is transmitted & processed,it is susceptible to noise that can alter or distort its contents. The 1's may get changed to 0's & 1's .because digital systems must be accurate to the digit, error can pose a problem. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

Parity: The simplest techniques for detecting errors is that of adding an extra bit known as parity bit to each word being transmitted.Two types of parity: Oddparity, evenparity forodd parity, the parity bit is set to a _0' or a _1' at the transmitter such that the total no. of 1 bit in the word including the parity bit is an odd no.For even parity, the parity bit is set to a _0' or a _1' at the transmitter such that the parity bit is an even no.

Decimal	8421 code	Odd parity	Even parity
0	0000	1	0
1	0001	0	1
2	0010	0	1
3	0011	1	0
4	0100	0	1
5	0100	1	0
6	0110	1	0
7	0111	0	1
8	1000	0	1
9	1001	1	0

When the digit data is received . a parity checking circuit generates an error signal if the total no of 1's is even in an odd parity system or odd in an even parity system. This parity check can always detect a single bit error but cannot detect 2 or more errors with in the same word.Odd parity is used more often than even parity does not detect the situation. Where all 0's are created by a short ckt or some other fault condition.

Ex: Even parity scheme

(a) 10101010 (b) 11110110 (c)10111001

Ans:

(a) No. of 1's in the word is even is 4 so there is no error
(b) No. of 1's in the word is even is 6 so there is no error
(c) No. of 1's in the word is odd is 5 so there is error

Ex: odd parity

(a)10110111 (b) 10011010 (c)11101010

Ans:

(a) No. of 1's in the word is even is 6 so word has error
(b) No. of 1's in the word is even is 4 so word has error
(c) No. of 1's in the word is odd is 5 so there is no error

Checksums:

Simple parity can't detect two errors within the same word. To overcome this, use a sort of 2 dimensional parity. As each word is transmitted, it is added to the sum of the previously transmitted words, and the sum retained at the transmitter end. At the end of transmission, the sum called the check sum. Up to that time sent to the receiver. The receiver can check its sum with the transmitted sum. If the two sums are the same, then no errors were detected at the receiver end. If there is an error, the receiving location can ask for retransmission of the entire data, used in teleprocessing systems.

Block parity:

Block of data shown is create the row & column parity bits for the data using odd parity. The parity bit 0 or 1 is added column wise & row wise such that the total no. of 1's in each column & row including the data bits & parity bit is odd as

Data	Parity bit
10110	0
10001	1
10101	0
00010	0
11000	1
00000	1
11010	0

data
10110
10001
10101
00010
11000
00000
11010

Error –Correcting Codes:

A code is said to be an error –correcting code, if the code word can always be deduced from an erroneous word. For a code to be a single bit error correcting code, the minimum distance of that code must be three. The minimum distance of that code is the smallest no. of bits by which any two code words must differ. A code with minimum distance of 3 can't only correct single bit errors but also detect (can't correct) two bit errors, The key to error correction is that it must be possible to detect & locate erroneous that it must be possible to detect & locate erroneous digits. If the location of an error has been determined. Then by complementing the erroneous digit, the message can be corrected , error correcting , code is the Hamming code , In this , to each group of m information or message or data bits, K parity checking bits denoted by P1,P2,----------pk located at positions 2^{k-1} from left are added to form an (m+k) bit code word. To correct the error, k parity checks are performed on selected digits of each code word, & the position of the error bit is located by forming an error word, & the error bit is then complemented. The k bit error word is generated by putting a 0 or a 1 in the 2^{k-1}th position depending upon whether the check for parity involving the parity bit P_k is satisfied or not.Error positions & their corresponding values :

Error Position	For 15 bit code $C_4C_3C_2C_1$	For 12 bit code $C_4C_3C_2C_1$	For 7 bit code $C_3C_2C_1$
0	0 0 0 0	0 0 0 0	0 0 0
1	0 0 0 1	0 0 0 1	0 0 1
2	0 0 1 0	0 0 1 0	0 1 0
3	0 0 1 1	0 0 1 1	0 1 1
4	0 1 0 0	0 1 0 0	1 0 0
5	0 1 0 1	0 1 0 1	1 0 1
6	0 1 1 0	0 1 1 0	1 1 0
7	0 1 1 1	0 1 1 1	1 1 1
8	1 0 0 0	1 0 0 0	
9	1 0 0 1	1 0 0 1	
10	1 0 1 0	1 0 1 0	
11	1 0 1 1	1 0 1 1	
12	1 1 0 0	1 1 0 0	
13	1 1 0 1		
14	1 1 1 0		
15	1 1 1 1		

7-bit Hamming code:

To transmit four data bits, 3 parity bits located at positions 2^0 21&2^2 from left are added to make a 7 bit codeword which is then transmitted.

The word format

P_1	P_2	D_3	P_4	D_5	D_6	D_7

D—Data bits P-

Parity bits

Decimal Digit	For BCD P1P2D3P4D5D6D7	For Excess-3 P1P2D3P4D5D6D7
0	0 0 0 0 0 0 0	1 0 0 0 0 1 1
1	1 1 0 1 0 0 1	1 0 0 1 1 0 0
2	0 1 0 1 0 1 1	0 1 0 0 1 0 1
3	1 0 0 0 0 1 1	1 1 0 0 1 1 0
4	1 0 0 1 1 0 0	0 0 0 1 1 1 1
5	0 1 0 0 1 0 1	1 1 1 0 0 0 0
6	1 1 0 0 1 1 0	0 0 1 1 0 0 1
7	0 0 0 1 1 1 1	1 0 1 1 0 1 0
8	1 1 1 0 0 0 0	0 1 1 0 0 1 1
9	0 0 1 1 0 0 1	0 1 1 1 1 0 0

Ex: Encode the data bits 1101 into the 7 bit even parity Hamming Code

The bit pattern is

P1P2D3P4D5D6D7

1 1 0 1

Bits 1,3,5,7 (P_1 111) must have even parity, so P_1=1

Bits 2, 3, 6, 7(P_2 101) must have even parity, so P_2=0

Bits 4,5,6,7 (P_4 101)must have even parity, so P_4=0

The final code is 1010101

EX: Code word is 1001001

Bits 1,3,5,7 (C_1 1001) →no error →put a 0 in the 1's position→C1=0

Bits 2, 3, 6, 7(C_2 0001)) → error →put a 1 in the 2's position→C2=1

Bits 4,5,6,7 (C_4 1001)) →no error →put a 0 in the 4's position→C3=0

15-bit Hamming Code: It transmit 11 data bits, 4 parity bits located 2^0 2^1 2^2 2^3

Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12	D13	D14	D15

12-Bit Hamming Code: It transmit 8 data bits, 4 parity bits located at position 2^0 2^1 2^2 2^3

Word format is

P_1	P_2	D_3	P_4	D_5	D_6	D_7	P_8	D_9	D10	D11	D12

Alphanumeric Codes:

These codes are used to encode the characteristics of alphabet in addition to the decimal digits. It is used for transmitting data between computers & its I/O device such as printers, keyboards & video display terminals.Popular modern alphanumeric codes are ASCII code & EBCDIC code.

Digital Logic Gates

Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these type of gates.

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Properties of XOR Gates

- XOR (also ⊕) : the "not-equal" function
- XOR(X,Y) = X ⊕ Y = X'Y + XY'
- Identities:
  - X ⊕ 0 = X
  - X ⊕ 1 = X'
  - X ⊕ X = 0
  - X ⊕ X' = 1
- Properties:
  - X ⊕ Y = Y ⊕ X
  - (X ⊕ Y) ⊕ W = X ⊕ (Y ⊕ W)

Universal Logic Gates

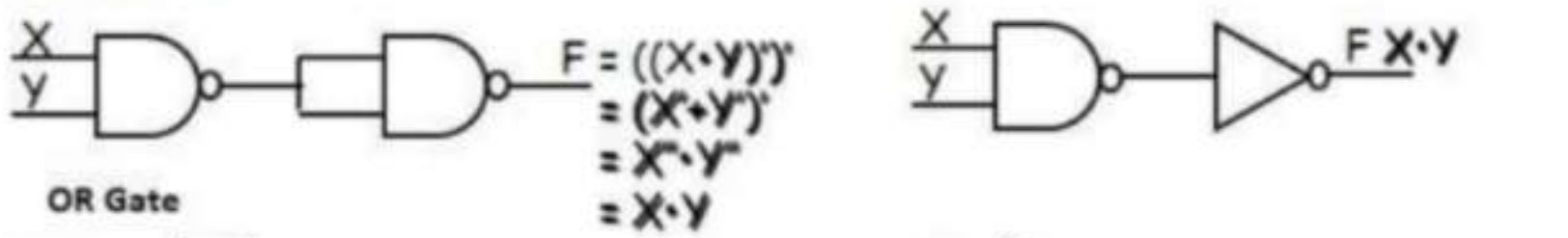
NAND and NOR gates are called Universal gates. All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate. A universal gate provides flexibility and offers enormous advantage to logic designers.

NAND as a Universal Gate

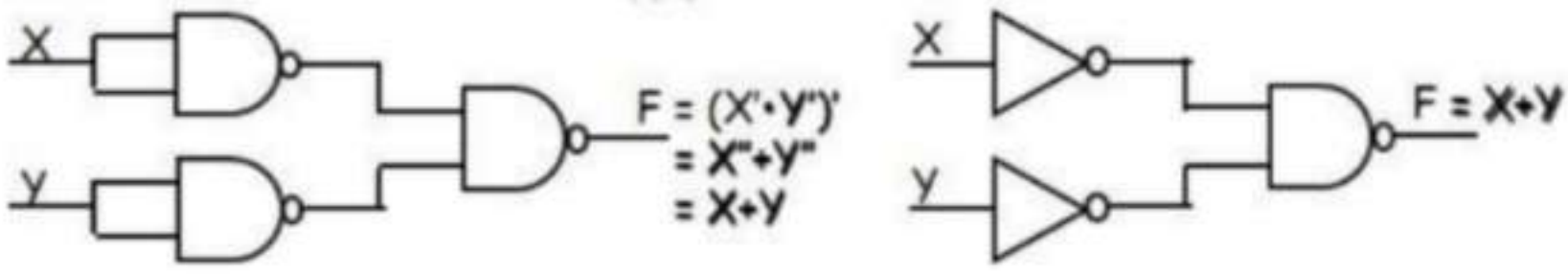
NAND Known as a "universal" gate because ANY digital circuit can be implemented with NAND gates alone.
To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.

NOT Gate

AND Gate



OR Gate



Boolean Algebra: In 1854, George Boole developed an algebraic system now called Boolean algebra. In 1938, Claude E. Shannon introduced a two-valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits. For the formal definition of Boolean algebra, we shall employ the postulates formulated by E. V. Huntington in 1904.

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0, 1), two binary operators called OR, AND, and one unary operator NOT. It is the basic mathematical tool in the analysis and synthesis of switching circuits. It is a way to express logic functions algebraically.

Boolean algebra, like any other deductive mathematical system, may be defined with aset of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is anycollection of objects having a common property. If **S** is a set and ***x*** and ***y*** are certain objects, then **x Î S**denotes that ***x*** is a member of the set **S**, and ***y*** **Ï S** denotes that ***y*** is not an element of **S**. A set with adenumerable number of elements is specified by braces: **A** = {1,2,3,4}, *i.e.* the elements of set **A** are thenumbers 1, 2, 3, and 4. A *binary operator* defined on a set S of elements is a rule that assigns to each pair ofelements from S a unique element from S._ Example: In $a*b=c$, we say that * is a binary operator if it specifies a rule for finding c from the pair $(a,b)$and also if a, b, c Î S.

Axioms and laws of Boolean algebra

Axioms or Postulates of Boolean algebra are a set of logical expressions that we accept without proof and upon which we can build a set of useful theorems.

	AND Operation	OR Operation	NOT Operation
Axiom1 :	0.0=0	0+0=0	$\overline{0}$=1
Axiom2:	0.1=0	0+1=1	$\overline{1}$=0
Axiom3:	1.0=0	1+0=1	
Axiom4:	1.1=1	1+1=1	

AND Law	**OR Law**
Law1: A.0=0 (Null law)	Law1: A+0=A
Law2: A.1=A (Identity law)	Law2: A+1=1
Law3: A.A=A (Impotence law)	Law3: A+A=A (Impotence law)

CLOSURE: The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values,it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

_ A set *S* is closed with respect to a binary operator if, for every pair of elements of *S*, the binary operator specifies a rule for obtaining a unique element of *S*.

_ For example, the set of natural numbers N = {1, 2, 3, 4, ... 9} is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any a, b Î N we obtain a unique c Î N by the operation $a + b = c$.

ASSOCIATIVE LAW:

A binary operator * on a set *S* is said to be associative whenever $(x * y) * z = x * (y * z)$ for all $x, y, z \hat{I} S$, forall Boolean values x, y and z.

COMMUTATIVE LAW:

A binary operator * on a set *S* is said to be commutative whenever $x * y = y * x$ for all $x, y, z \epsilon S$

IDENTITY ELEMENT:

A set *S* is said to have an identity element with respect to a binary operation * on S if there exists an element $e \epsilon S$ with the property $e * x = x * e = x$ for every $x \epsilon S$

BASIC IDENTITIES OF BOOLEAN ALGEBRA

- *Postulate 1(Definition)*: A Boolean algebra is a closed algebraic system containing a set *K* of two or more elements and the two operators · and + which refer to logical AND and logical OR $\bullet x + 0 = x$
- $x \cdot 0 = 0$
- $x + 1 = 1$
- $x \cdot 1 = 1$
- $x + x = x$
- $x \cdot x = x$
- $x + x' = x$
- $x \cdot x' = 0$
- $x + y = y + x$
- $xy = yx$
- $x + (y + z) = (x + y) + z$
- $x (yz) = (xy) z$
- $x (y + z) = xy + xz$
- $x + yz = (x + y)(x + z)$
- $(x + y)' = x'y'$
- $(xy)' = x' + y'$

- $(x')' = x$

DeMorgan's Theorem

(a) $(a + b)' = a'b'$

(b) $(ab)' = a' + b'$

Generalized DeMorgan's Theorem

(a) $(a + b + \ldots z)' = a'b' \ldots z'$
(b) $(a.b \ldots z)' = a' + b' + \ldots z'$

Basic Theorems and Properties of Boolean algebra Commutative law

Law1: A+B=B+A Law2: A.B=B.A

Associative law

Law1: A + (B +C) = (A +B) +C Law2: A(B.C) = (A.B)C

Distributive law

Law1: A.(B + C) = AB+ AC Law2: A + BC = (A + B).(A +C)

Absorption law

Law1: A +AB =A Law2: A(A +B) = A

Solution: A(1+B)
A

Solution: A.A+A.B
A+A.B
A(1+B)
A

Consensus Theorem

Theorem1. AB+ A'C + BC = AB + A'C Theorem2. (A+B). (A'+C).(B+C) =(A+B).(A'+C)

The BC term is called the consensus term and is redundant. The consensus term is formed from a PAIR OF TERMS in which a variable (A) and its complement (A') are present; the consensus term is formed by multiplying the two terms and leaving out the selected variable and its complement

Consensus Theorem1 Proof:

AB+A'C+BC=AB+A'C+(A+A')BC
=AB+A'C+ABC+A'BC

=AB(1+C)+A'C(1+B)

= AB+ A'C

Principle of Duality

Each postulate consists of two expressions statement one expression is transformed into the other by interchanging the operations (+) and (·) as well as the identity elements 0 and 1.
Such expressions are known as duals of each other.
If some equivalence is proved, then its dual is also immediately true.

If we prove: (x.x)+(x'+x')=1, then we have by duality: (x+x)·(x'.x')=0

The Huntington postulates were listed in pairs and designated by part (a) and part (b) in below table.

Table for Postulates and Theorems of Boolean algebra

Part-A	Part-B
A+0=A	A.0=0
A+1=1	A.1=A
A+A=A (Impotence law)	A.A=A (Impotence law)
$A+\bar{A}1$	$A.\bar{A}0$
$\bar{\bar{A}}A$ (double inversion law)	--
Commutative law: A+B=B+A	A.B=B.A
Associative law: A + (B +C) = (A +B) +C	A(B.C) = (A.B)C
Distributive law: A.(B + C) = AB+ AC	A + BC = (A + B).(A +C)
Absorption law: A +AB =A	A(A +B) = A
DeMorgan Theorem: $\overline{(A+B)} = \bar{A}.\bar{B}$	$\overline{(A.B)} = = \bar{A}+\bar{B}$
Redundant Literal Rule: $A+\bar{A}B=A+B$	$A.\overline{(A+B)}=AB$
Consensus Theorem: AB+ A'C + BC = AB + A'C	(A+B). (A'+C).(B+C) =(A+B).(A'+C)

Boolean Function

Boolean algebra is an algebra that deals with binary variables and logic operations.
A Boolean function described by an algebraic expression consists of binary variables, the constants 0 and 1, and the logic operation symbols.
For a given value of the binary variables, the function can be equal to either 1 or 0.

F(vars) = *expression*

Set of binary Variables

Operators (+, •, ')
Constants (0, 1)
Groupings (parenthesis)
Variables

Consider an example for the Boolean function

F1 = x + y'z

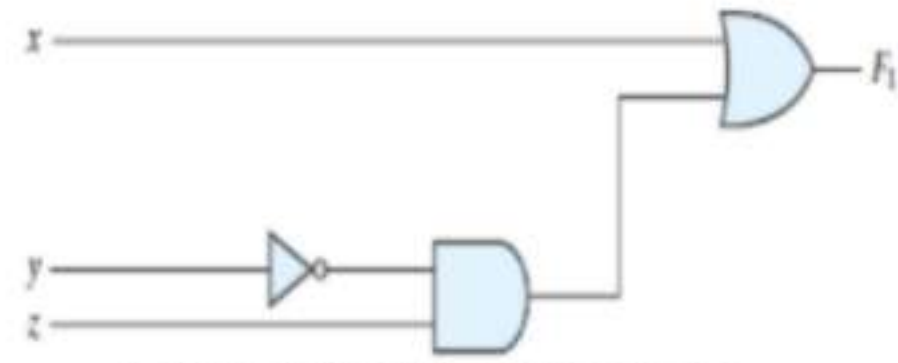
The function F1 is equal to 1 if x is equal to 1 or if both y' and z are equal to 1. F1 is equal to 0 otherwise. The complement operation dictates that when y' = 1, y = 0. Therefore, F1 = 1 if x = 1 or if y = 0 and z = 1.

A Boolean function expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

A Boolean function can be represented in a truth table. The number of rows in the truth table is 2^n, where n is the number of variables in the function. The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$.

Truth Table for F1

x	y	z	F_1
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1



Gate Implementation of F1 = x + y'z

Note:

Q: Let a function F() depend on n variables. How many rows are there in the truth table of F() ?

A: 2^n rows, since there are 2^n possible binary patterns/combinations for the n variables.

Truth Tables

- Enumerates all possible combinations of variable values and the corresponding function value
- Truth tables for some arbitrary functions
  F1(x,y,z), F2(x,y,z), and F3(x,y,z) are shown to the below.

x	y	z	F_1	F_2	F_3
0	0	0	0	1	1
0	0	1	0	0	1

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

- Truth table: a unique representation of a Boolean function
- If two functions have identical truth tables, the functions are equivalent (and vice-versa).
- Truth tables can be used to prove equality theorems.
- However, the size of a truth table grows exponentially with the number of variables involved, hence unwieldy. This motivates the use of Boolean Algebra.

Boolean expressions-NOT unique

Unlike truth tables, expressions epresenting a Boolean function are NOT unique.

- Example:
  - $F(x,y,z) = x' \bullet y' \bullet z' + x' \bullet y \bullet z' + x \bullet y \bullet z'$
  - $G(x,y,z) = x' \bullet y' \bullet z' + y \bullet z'$
- The corresponding truth tables for F() and G() are to the right. They are identical.
- Thus, F() = G()

x	y	z	F	G
0	0	0	1	1
0	0	1	0	0
0	1	0	1	1
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	0	0

Algebraic Manipulation (Minimization of Boolean function)

- Boolean algebra is a useful tool for simplifying digital circuits.
- Why do it? Simpler can mean cheaper, smaller, faster.
- Example: Simplify $F = x'yz + x'yz' + xz$.

$$F = x'yz + x'yz' + xz$$
$$= x'y(z+z') + xz$$
$$= x'y \bullet 1 + xz$$

$= x'y + xz$

- Example: Prove

  $x'y'z' + x'yz' + xyz' = x'z' + yz'$

- **Proof:**

  $x'y'z' + x'yz' + xyz'$

  $= x'y'z' + x'yz' + x'yz' + xyz'$

  $= x'z'(y'+y) + yz'(x'+x)$

  $= x'z' \bullet 1 + yz' \bullet 1$

  $= x'z' + yz'$

Complement of a Function

- The complement of a function is derived by interchanging (• and +), and (1 and 0), and complementing each variable.
- Otherwise, interchange 1s to 0s in the truth table column showing F.
- The *complement* of a function IS NOT THE SAME as the *dual* of a function.

Example

- Find G(x,y,z), the complement of $F(x,y,z) = xy'z' + x'yz$

  Ans: $G = F' = (xy'z' + x'yz)'$

  $= (xy'z')' \bullet (x'yz)'$ *DeMorgan*

  $= (x'+y+z) \bullet (x+y'+z')$ *DeMorgan* again

  Note: The complement of a function can also be derived by finding the function's *dual*, and then complementing all of the literals

Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions.
Identical functions will have exactly the same canonical form.

- Minterms and Maxterms
- Sum-of-Minterms and Product-of- Maxterms
- Product and Sum terms
- Sum-of-Products (SOP) and Product-of-Sums (POS)

Definitions

Literal: A variable or its complement
Product term: literals connected by •
Sum term: literals connected by +
Minterm: a product term in which all the variables appear exactly once, either complemented or uncomplemented.

Maxterm: a sum term in which all the variables appear exactly once, either complemented or uncomplemented.

Canonical form: Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

Minterm

- Represents exactly one combination in the truth table.
- Denoted by m_j, where j is the decimal equivalent of the minterm's corresponding binary combination (b_j).
- A variable in m_j is complemented if its value in b_j is 0, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding minterm is denoted by m_j = A'BC

Maxterm

- Represents exactly one combination in the truth table.
- Denoted by M_j, where j is the decimal equivalent of the maxterm's corresponding binary combination (b_j).
- A variable in M_j is complemented if its value in b_j is 1, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and j=3. Then, b_j = 011 and its corresponding maxterm is denoted by M_j = A+B'+C'

Truth Table notation for Minterms and Maxterms

- Minterms and Maxterms are easy to denote using a truth table.

Example: Assume 3 variables x,y,z (order is fixed)

x	y	z	Minterm	Maxterm
0	0	0	$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1	$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0	$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1	$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0	$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1	$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0	$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1	$xyz = m_7$	$x'+y'+z' = M_7$

Canonical Forms

- Every function F() has two canonical forms:
  - Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)

Canonical Sum-Of-Products:

The minterms included are those m_j such that F( ) = 1 in row j of the truth table for F( ).

Canonical Product-Of-Sums:

The maxterms included are those M_j such that F( ) = 0 in row j of the truth table for F( ).

Example

Consider a Truth table for $f_1(a,b,c)$ at right

The canonical sum-of-products form for f_1 is

$f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

$= a'b'c + a'bc' + ab'c' + abc'$

The canonical product-of-sums form for f_1 is

$f_1(a,b,c) = M_0 \bullet M_3 \bullet M_5 \bullet M_7$

$= (a+b+c)\bullet(a+b'+c')\bullet (a'+b+c')\bullet(a'+b'+c').$

a	b	c	f_1
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- Observe that: $m_j = M_j'$

Shorthand: ∑ and ∏

- $f_1(a,b,c) = \sum m(1,2,4,6)$, where ∑ indicates that this is a sum-of-products form, and m(1,2,4,6) indicates that the minterms to be included are m_1, m_2, m_4, and m_6.
- $f_1(a,b,c) = \prod M(0,3,5,7)$, where ∏ indicates that this is a product-of-sums form, and M(0,3,5,7) indicates that the maxterms to be included are M_0, M_3, M_5, and M_7.
- Since $m_j = M_j'$ for any *j*,
  $\sum m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$
-

Conversion between Canonical Forms

- Replace ∑ with ∏ (or *vice versa*) and replace those *j*'s that appeared in the original form with those that do not.
    - Example:

$$f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$$
$$= m_1 + m_2 + m_4 + m_6$$
$$= \sum(1,2,4,6)$$
$$= \prod(0,3,5,7)$$
$$= (a+b+c)\bullet(a+b'+c')\bullet(a'+b+c')\bullet(a'+b'+c')$$

Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.

There are two types of standard forms: the sum of products and products of sums.

The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

$$F1 = y' + xy + x'yz'$$

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.

A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is

$$F2 = x(y' + z)(x' + y + z')$$

This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

Conversion of SOP from standard to canonical form

Example-1.

Express the Boolean function F = A + B'C as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

$$A = A(B + B') = AB + AB'$$

This function is still missing one variable, so

$$A = AB(C + C') + AB'(C + C')$$
$$= ABC + ABC' + AB'C + AB'C'$$

The second term B'C is missing one variable; hence,

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Combining all terms, we have

$$F = A + B'C$$
$$= ABC + ABC' + AB'C + AB'C' + A'B'C$$

But AB'C appears twice, and according to theorem (x + x = x), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$F = A'B'C + AB'C + AB'C + ABC' + ABC$$
$$= m1 + m4 + m5 + m6 + m7$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$F(A, B, C) = \sum m(1, 4, 5, 6, 7)$$

Example-2.

Express the Boolean function F = xy + x'z as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

$$F = xy + x'z = (xy + x')(xy + z)$$
$$= (x + x')(y + x')(x + z)(y + z)$$
$$= (x' + y)(x + z)(y + z)$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z)$$
$$F = M0M2M4M5$$

A convenient way to express this function is as follows: $F(x, y, z) = \pi M(0, 2, 4, 5)$

The product symbol, π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

Two-variable k-map:

A two-variable k-map can have $2^2=4$ possible combinations of the input variables A and B Each of these combinations, , B,A ,AB(in the SOP form) is called a minterm. The minterm may be represented in terms of their decimal designations – m0 for , m1 for B,m2 for A and m3 for AB, assuming that A represents the MSB. The letter m stands for minterm and the subscript represents the decimal designation of the minterm. The presence or absence of a minterm in the expression indicates that the output of the logic circuit assumes logic 1 or logic 0 level for that combination of input variables.

The expression f= ,+ B+A +AB , it can be expressed using min

term as F= m0+m2+m3=∑m(0,2,3)

Using Truth Table:

Minterm	Inputs A	B	Output F
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

A 1 in the output contains that particular minterm in its sum and a 0 in that column indicates that the particular mintermdoes not appear in the expression for output . this information can also be indicated by a two-variable k-map.

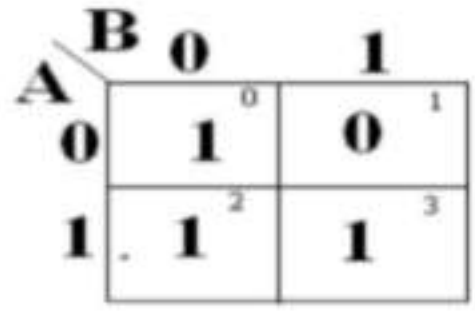
Mapping of SOP Expresions:

A two-variable k-map has 22=4 squares .These squares are called cells. Each square on the k-map represents a unique minterm. The minterm designation of the squares are placed in any square, indicates that the corresponding minterm does output expressions. And a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

A \ B	0	1
0	A'B'	A'B
1	AB'	AB

The minterms of a two-variable k-map

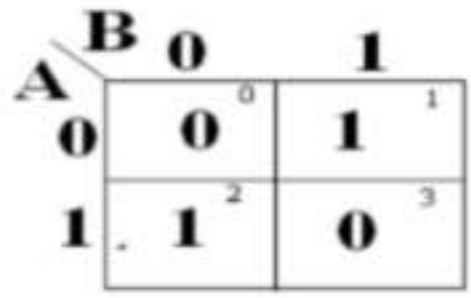
The mapping of the expressions =∑m(0,2,3)is



k-map of ∑m(0,2,3)

EX: Map the expressions f= B+A

F= m_1+m_2=∑m(1,2)The k-map is



Minimizations of SOP expressions:

To minimize Boolean expressions given in the SOP form by using the k-map, look for adjacent adjacent squares having 1's minterms adjacent to each other, and combine them to form larger squares to eliminate some variables. Two squares are said to be adjacent to each other, if their minterms differ in only one variable. (i.e, B & A differ only in one variable. so theymay be combined to form a 2-square to eliminate the variable B.similarly all other.

The necessary condition for adjacency of minterms is that their decimal designations must differ by a power of 2. A minterm can be combined with any number of minterms adjacent to it to form larger squares. Two minterms which are adjacent to each other can be combined to form a bigger square called a 2-square or a pair. This eliminates one variable – the variable that is not common to both the minterms. For EX:

m0 and m1 can be combined to yield,

f_1 = m0+m1= + B= (B+

)= m0 and m2 can be combined to yield,

f_2 = m0+m2= + = (+)=

m1 and m3 can be combined to yield,

f_3= m1+m3= B+AB=B(+)=B

m2 and m3 can be combined to yield,

f_4 = m2+m3=A +AB=A(B+)=A

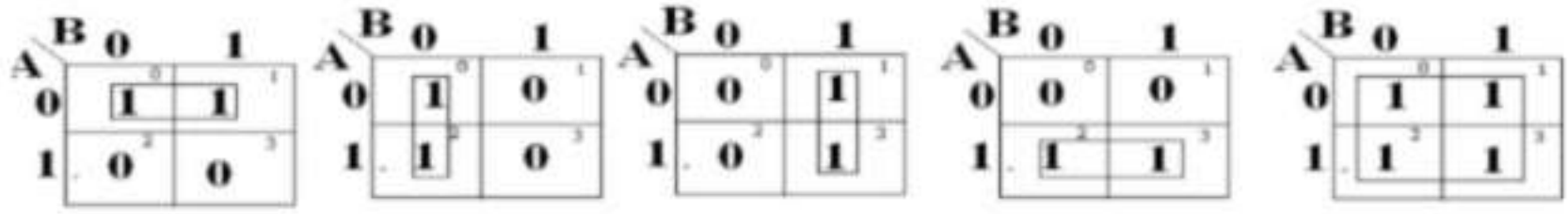
m_0 ,m_1 ,m_2 and m_3 can be combined to yield,

= + +A +AB

= (B+) +A(B+)

= +A

=1

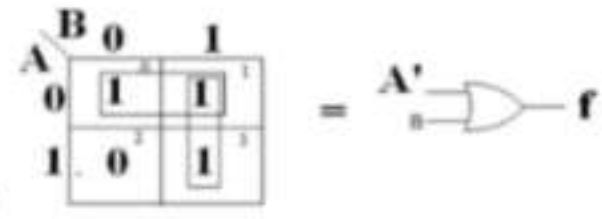


f1= f2= f3=B f4=A f5=1

The possible minterm groupings in a two-variable k-map.

Two 2-squares adjacent to each other can be combined to form a 4-square. A 4-square eliminates 2 variables. A 4-square is called a quad. To read the squares on the map after minimization, consider only those variables which remain constant through the square, and ignore the variables which are varying. Write the non complemented variable if the variable is remaining constant as a 1, and the complemented variable if the variable is remaining constant as a 0, and write the variables as a product term. In the above figure f_1 read as , because, along the square , A remains constant as a 0, that is , as , where as B is changing from 0 to 1.

EX: Reduce the minterm f= +A +AB using mapping Expressed in terms of minterms, the given expression is F=m_0+m_1+m_2+ m_3=m$\sum$(0,1,3)& the figure shows the k-map for f and its reduction . In one 2-square, A is constant as a 0 but B varies from a 0 to a 1, and in the other 2-square, B is constant as a 1 but A varies from a 0 to a 1. So, the reduced expressions is +B.



It requires two gate inputs for realization as

f= +B (k-map in SOP form, and logic diagram.)

The main criterion in the design of a digital circuit is that its cost should be as low as possible. For that the expression used to realize that circuit must be minimal.Since the cost is proportional to number of gate inputs in the circuit in the circuit, an expression is considered minimal only if it corresponds to the least possible number of gate inputs. & there is no guarantee for that k-map in SOP is the real minimal. To obtain real minimal expression, obtain the minimal expression both in SOP & POS form form by using k-maps and take the minimal of these two minimals.

The 1's on the k-map indicate the presence of minterms in the output expressions, where as the 0s indicate the absence of minterms .Since the absence of a minterm in the SOP expression means the presense of the corresponding maxterm in the POS expression of the same .when a SOP expression is plotted on the k-map, 0s or no entries on the k-map represent the maxterms. To obtain the minimal expression in the POS form, consider the 0s on the k-map and follow the procedure used for combining 1s. Also, since the absence of a maxterm in the POS expression means the presence of the corresponding minterm in the SOP expression of the same , when a POS expression is plotted on the k-map, 1s or no entries on the k-map represent the minterms.

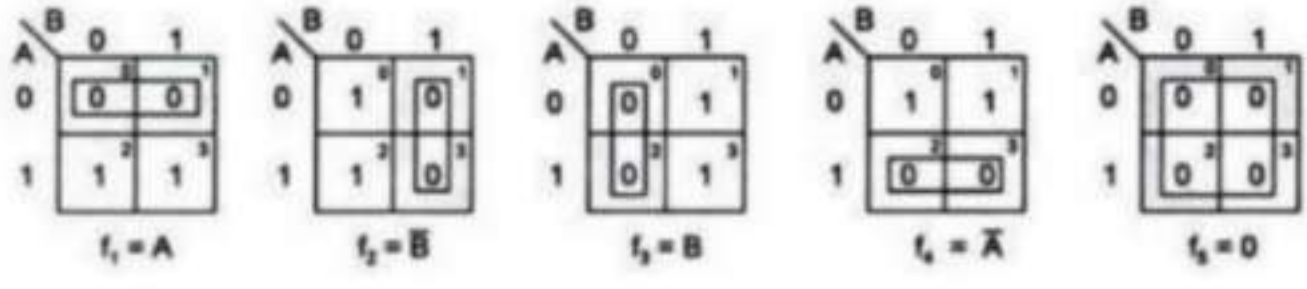
Mapping of POS expressions:

Each sum term in the standard POS expression is called a maxterm. A function in two variables (A, B) has four possible maxterms, A+B,A+ , +B, +

. They are represented as M_0, M_1, M2, and M3respectively. The uppercase letter M stands for maxterm and its subscript denotes the decimal designation of that maxterm obtained by treating the non-complemented variable as a 0 and the complemented variable as a 1 and putting them side by side for reading the decimal equivalent of the binary number so formed.

For mapping a POS expression on to the k-map, 0s are placed in the squares corresponding to the maxterms which are presented in the expression an d1s are placed in the squares corresponding to the maxterm which are not present in the expression. The decimal designation of the squares of the squares for maxterms is the same as that for the minterms. A two-variable k-map & the associated maxterms are asthe maxterms of a two-variable k-map

The possible maxterm groupings in a two-variable k-map



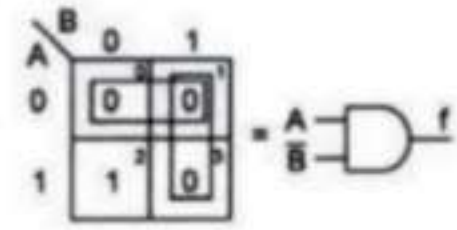
Minimization of POS Expressions:

To obtain the minimal expression in POS form, map the given POS expression on to the K-map and combine the adjacent 0s into as large squares as possible. Read the squares putting the complemented variable if its value remains constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square (ignoring the variables which do not remain constant throughout the square) and then write them as a sum term.

Various maxterm combinations and the corresponding reduced expressions are shown in figure. In this f_1 read as A because A remains constant as a 0 throughout the square and B changes from a 0 to a 1. f_2 is read as B' because B remains constant along the square as a 1 and A changes from a 0 to a 1. f_5
Is read as a 0 because both the variables are changing along the square.

Ex: Reduce the expression f=(A+B)(A+B')(A'+B') using mapping.

The given expression in terms of maxterms is f=πM(0,1,3). It requires two gates inputs for realization of the reduced expression as



F=AB'

K-map in POS form and logic diagram

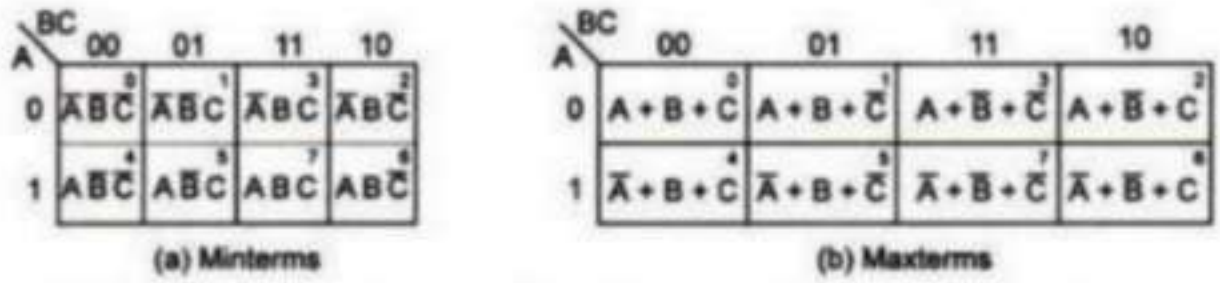
In this given expression ,the maxterm M_2 is absent. This is indicated by a 1 on the k-map. The corresponding SOP expression is $\sum m_2$ or AB'. This realization is the same as that for the POS form.

Three-variable K-map:

A function in three variables (A, B, C) expressed in the standard SOP form can have eight possible combinations: A B C , AB C,A BC ,A BC,AB C ,AB C,ABC , and ABC. Each one of these combinations designate d by m0,m1,m2,m3,m4,m5,m6, and m7, respectively, is called a minterm. A is the MSB of the minterm designator and C is the LSB.

In the standard POS form, the eight possible combinations are:A+B+C, A+B+C , A+B +C,A+B + C ,A + B+ C,A + B + C ,A + B + C,A + B + C . Each oneof these combinations designated by M_0, M1, M2, M3, M4, M5, M6, and M7respectively is called a maxterm. A is the MSB of the maxterm designator and C is the LSB.

A three-variable k-map has, therefore, $8(=2^3)$ squares or cells, and each square on the map represents a minterm or maxterm as shown in figure. The small number on the top right corner of each cell indicates the minterm or maxterm designation.



The three-variable k-map.

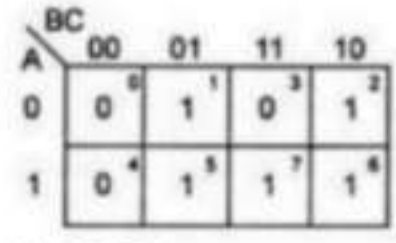
The binary numbers along the top of the map indicate the condition of B and C for each column. The binary number along the left side of the map against each row indicates the condition of A for that row. For example, the binary number 01 on top of the second column in fig indicates that the variable B appears in complemented form and the variable C in non-complemented form in all the minterms in that column. The binary number 0 on the left of the first row indicates that the variable A appears in complemented form in all the minterms in that row, the binary numbers along the top of the k-map are not in normal binary order. They are, infact, in the Gray code. This is to ensure that twophysically adjacent squares are really adjacent, i.e., their minterms or maxterms differ by only one variable.

Ex: Map the expression f=: C+ + + +ABC

In the given expression , the minterms are : C=001=m_1 ; =101=m_5; =010=m_2;

=110=m_6;ABC=111=m_7.

So the expression is f=$\sum$m(1,5,2,6,7)= $\sum$m(1,2,5,6,7). The corresponding k-map is



K-map in SOP form

Ex: Map the expression f= (A+B+C),(+ +) (+ +)(A + +)(+ +)

In the given expression the maxterms are :A+B+C=000=M_0; + + =101=M_5; + + = 111=M_7; A + + =011=M_3; + + =110=M_6.

So the expression is f = π M (0,5,7,3,6)= π M (0,3,5,6,7). The mapping of the expression is

A \ BC	00	01	11	10
0	0 (0)	1 (1)	0 (3)	1 (2)
1	1 (4)	0 (5)	0 (7)	0 (6)

K-map in POS form.

Minimization of SOP and POS expressions:

For reducing the Boolean expressions in SOP (POS) form plotted on the k-map, look at the 1s (0s) present on the map. These represent the minterms (maxterms). Look for the minterms (maxterms) adjacent to each other, in order to combine them into larger squares. Combining of adjacent squares in a k-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS)expression is called *looping*. Some of the minterms (maxterms) may have many adjacencies. Always start with the minterms (maxterm) with the least number of adjacencies and try to form as large as large a square as possible. The larger must form a geometric square or rectangle. They can be formed even by wrapping around, but cannot be formed by using diagonal configurations. Next consider the minterm (maxterm) with next to the least number of adjacencies and form as large a square as possible. Continue this till all the minterms (maxterms) are taken care of . A minterm (maxterm) can be part of any number of squares if it is helpful in reduction. Read the minimal expression from the k-map, corresponding to the squares formed. There can be more than one minimal expression.

Two squares are said to be adjacent to each other (since the binary designations along the top of the map and those along the left side of the map are in Gray code), if they are physically adjacent to each other, or can be made adjacent to each other by wrapping around. For squares to be combinable into bigger squares it is essential but not sufficient that their minterm designations must differ by a power of two.

General procedure to simplify the Boolean expressions:

1. Plot the k-map and place 1s(0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
2. Check the k-map for 1s(0s) which are not adjacent to any other 1(0). They are isolated minterms(maxterms) . They are to be read as they are because they cannot be combined even into a 2-square.
3. Check for those 1s(0S) which are adjacent to only one other 1(0) and make them pairs (2 squares).
4. Check for quads (4 squares) and octets (8 squares) of adjacent 1s (0s) even if they contain some 1s(0s) which have already been combined. They must geometrically form a square or a rectangle.
5. Check for any 1s(0s) that have not been combined yet and combine them into bigger squares if possible.
6. Form the minimal expression by summing (multiplying) the product the product (sum) terms of all the groups.

Reading the K-maps:

While reading the reduced k-map in SOP (POS) form, the variable which remains constant as 0 along the square is written as the complemented (non-complemented) variable and the one which remains constant as 1 along the square is written as non-complemented (complemented) variable and the term as a product (sum) term. All the product (sum) terms are added (multiplied).

Some possible combinations of minterms and the corresponding minimal expressions readfrom the k-maps are shown in fig: Here f_6 is read as 1, because along the 8-square no variable remains constant. F_5 is read as , because, along the 4-square formed by0,m_1,m_2 and m_3, the variables B and C are changing, and A remains constant as a 0. Algebraically,

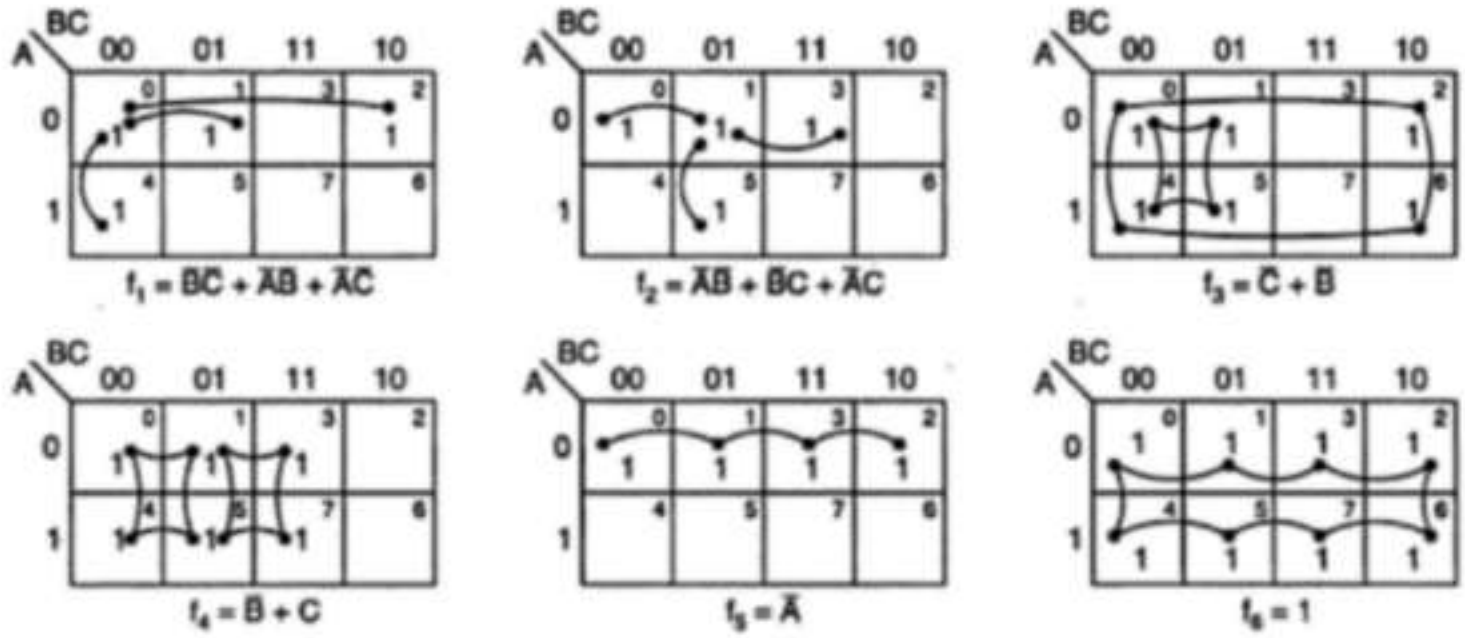
$f_5 = m_0+m_1+m_2+m_3$

$=$ + C+ +

$=$ (+C)+ B(C+)

$=$ + B

$=$ (+B)=

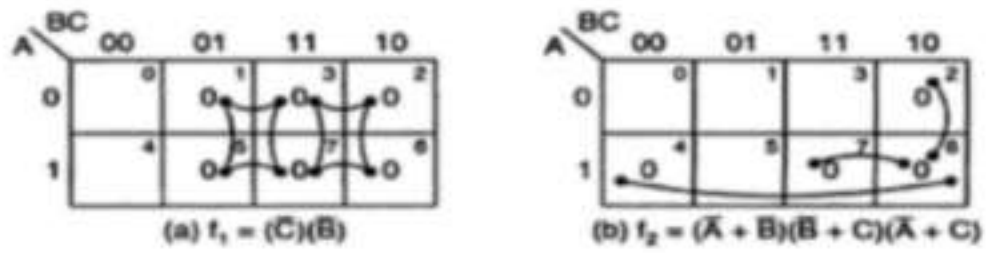


f_3 is read as + , because in the 4-square formed by m0,m2,m6, and m4, the variable A and B are changing , where as the variable C remains constant as a 0. So it is read as . In the 4-square formed by m_0, m1, m4, m_5, A and C are changing but B remains constant as a 0. So it is read as . So, the resultant expression for f_3 is the sum of these two, i.e., + .

f_1 is read as + ,because in the 2-square formed by m_0 and m_4 , A is changing from a0 to a 1. Whereas B and C remain constant as a 0. So it s read as . In the 2-square formed by m_0 and m_1, C is changing from a 0 to a 1, whereas A and B remain constant as a 0. So it is read as .In the 2-square formed by m_0 and m_2 , B is changing from a 0 to a 1 whereas A and C remain constant as a 0. So, it is read as . Therefore, the resultant SOP expression is

+ +

Some possible maxterm groupings and the corresponding minimal POS expressions read from the k-map are



In this figure, along the 4-square formed by M_1, M3, M7, M5, A and B are changing from a 0 to a 1, where as C remains constant as a 1. SO it is read as . Along the 4-squad formed by M_3, M2, M7, and M_6, variables A and C are changing from a 0 to a 1. But B remains constant as a 1. So it is read as . The minimal expression is the product of these two terms , i.e., $f_1 = (\quad)(\quad)$.also in this figure, along the 2-square formed by M_4 and M_6 , variable B is changing from a 0 to a 1, while variable A remains constant as a 1 and variable C remains constant as a 0. SO, read it as

+C. Similarly, the 2-square formed by M_7 andM_6 is read as + , while the 2-square formed by M_2 and M_6 is read as +C. The minimal expression is the product of these sum terms, i.e, f_2 =(+)+(+)+(+C)

Ex:Reduce the expression f=∑m(0,2,3,4,5,6) using mapping and implement it in AOI logic as well as in NAND logic.The Sop k-map and its reduction , and the implementation of the minimal expression using AOI logic and the corresponding NAND logic are shown in figures below

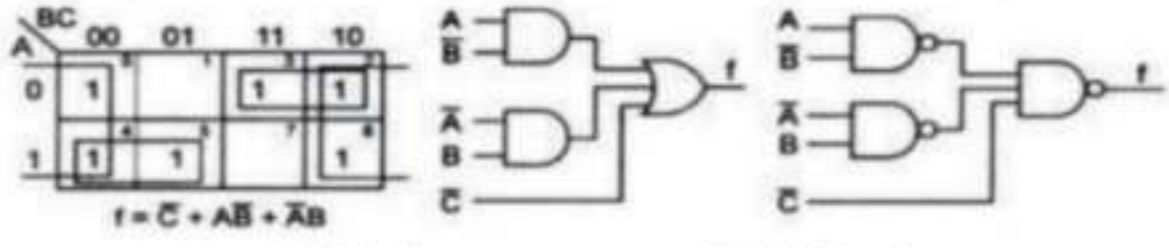
In SOP k-map, the reduction is done as:

1 m_5 has only one adjacency m_4 , so combine m_5 and m_4 into a square. Along this 2-square A remains constant as 1 and B remains constant as 0 but C varies from 0 to 1. So read it as A .

2 m_3 has only one adjacency m_2 , so combine m_3 and m_2 into a square. Along this 2-square A remains constant as 0 and B remains constant as 1 but C varies from 1 to 0. So read it as B.

3 m_6 can form a 2-square with m_2 and m_4 can form a 2-square with m_0, but observe that by wrapping the map from left to right m_0, m_4 ,m_2 ,m_6 can form a 4-square. Out of these m_2 andm_4 have already been combined but they can be utilized again. So make it. Along this 4-square, A is changing from 0 to 1 and B is also changing from 0 to 1 but C is remaining constant as 0. so read it as .

4 Write all the product terms in SOP form. So the minimal SOP expression is



f_{min}=

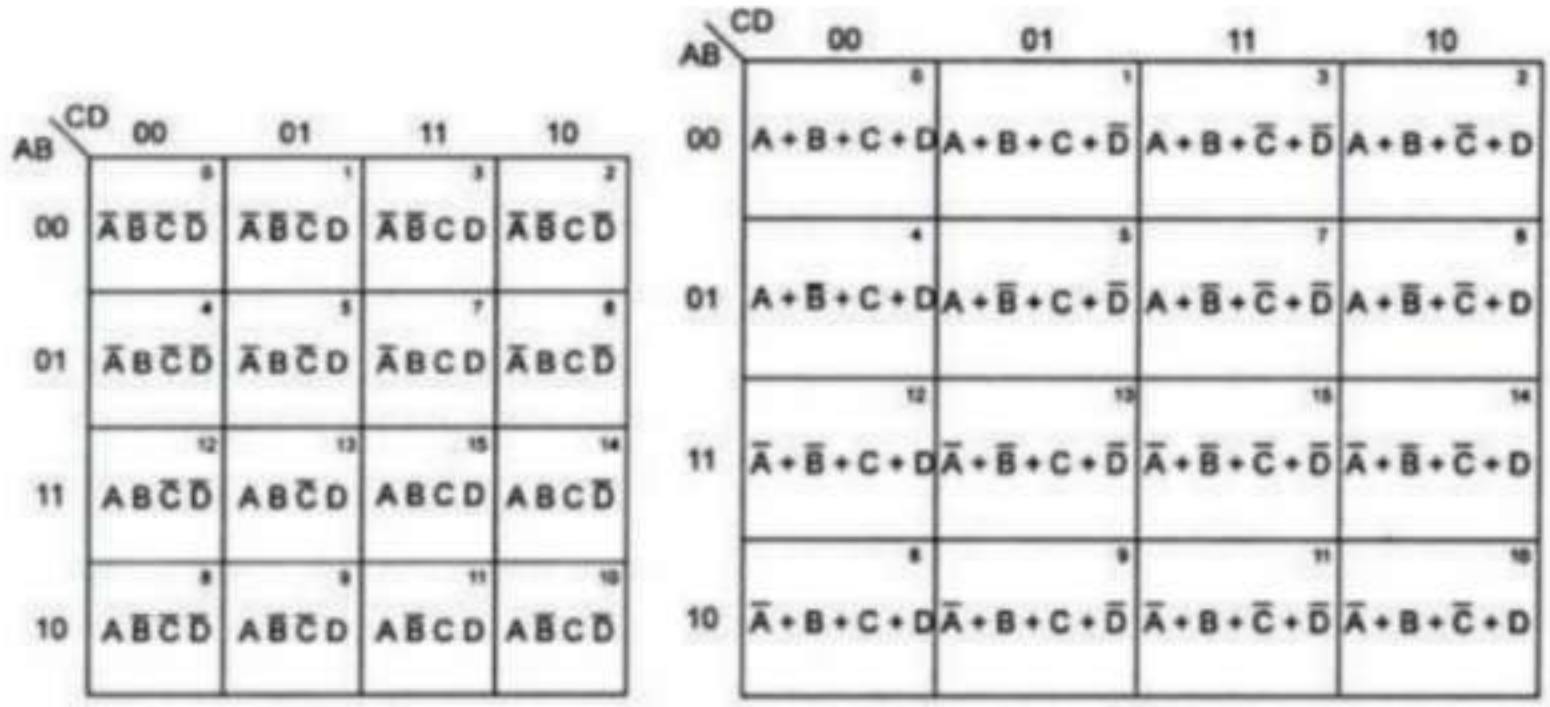
k-map **AOI logic** **NAND logic**

Four variable k-maps:

Four variable k-map expressions can have 2^4=16 possible combinations of input variables such as , ,----------ABCD with minterm designations m_0, m_1 .....m_{15} respectively in SOP form & A+B+C+D, A+B+C+ ,--------- + + + with maxterms M_0, M_1, --------

-

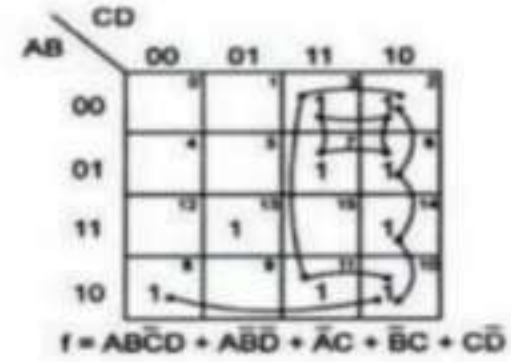
-M_{15} respectively in POS form. It has 2^4=16 squares or cells.The binary number designations of rows & columns are in the gray code. Here follows 01 & 10 follows 11 called Adjacency ordering.



SOP form POS form

EX: Reduce using mapping the expression Σ m(2, 3, 6, 7, 8, 10, 11, 13, 14).

Start with the minterm with the least number of adjacencies. The minterm m_{13} has no adjacency. Keep it as it is. The m_8 has only one adjacency, m_{10}. Expand m_8 into a 2-square with m_{10}. The m_7 has two adjacencies, m_6 and m_3. Hence m_7 can be expanded into a 4-square with m_6, m_3 and m_2. Observe that, m_7, m_6, m_2, and m_3 form a geometric square. The m_{11} has 2 adjacencies, m_{10} and m_3. Observe that, m_{11}, m_{10}, m_3, and m_2 form a geometric square on wrapping the K-map. So expand m_{11} into a 4-square with m_{10}, m_3 and m_2. Note that, m_2 and m_3, have already become a part of the 4-square m_7, m_6, m_2, and m_3. But if m_{11} is expanded only into a 2-square with m_{10}, only one variable is eliminated. So m_2 and m_3 are used again to make another 4-square with m_{11} and m_{10} to eliminate two variables. Now only m_6 and m_{14} are left uncovered. They can form a 2-square that eliminates only one variable. Don't do that. See whether they can be expanded into a larger square. Observe that, m_2, m_6, m_{14}, and m_{10} form a rectangle. So m_6 and m_{14} can be expanded into a 4-square with m_2 and m_{10}. This eliminates two variables.



Five variable k-map:

Five variable k-map can have 2^5 =32 possible combinations of input variable as , E,--------ABCDE with minterms m_0, m_1-----m_{31} respectively in SOP & A+B+C+D+E, A+B+C+ ,--------- + + + + with maxterms M_0,M_1, ---------- M_{31} respectively in POS form. It has 2^5=32 squares or cells of the k-map are divided into 2 blocks of

16 squares each.The left block represents minterms from m_0 to m_{15} in which A is a 0, and the right block represents minterms from m_{16} to m_{31} in which A is 1.The 5-variable k-map may contain 2-squares, 4-squares , 8-squares , 16-squares or 32-squares involving these two blocks. Squares are also considered adjacent in these two blocks, if when superimposing one block on top of another, the squares coincide with one another.

Some possible 2-squares in a five-variable map are m_0, m_{16}; m_2, m_{18}; m_5, m_{21}; m_{15}, m_{31}; m_{11}, m_{27}.

Some possible 4-squares are m_0, m_2, m_{16}, m_{18}; m_0, m_1, m_{16}, m_{17}; m_0, m_4, m_{16}, m_{20}; m_{13}, m_{15}, m_{29}, m_{31}; m_5, m_{13}, m_{21}, m_{29}.

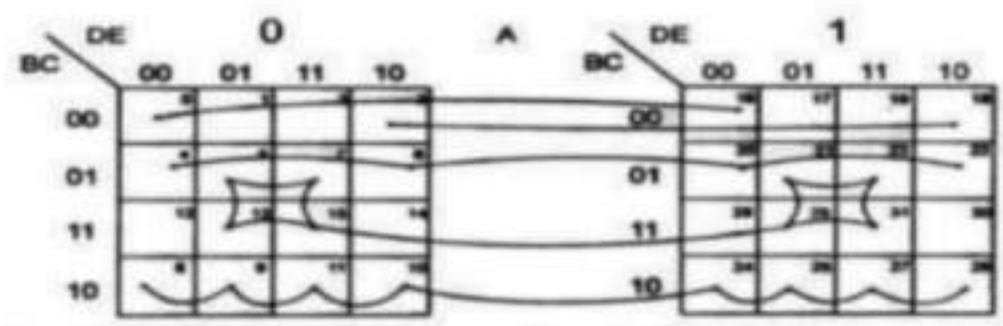
Some possible 8-squares are m_0, m_1, m_3, m_2, m_{16}, m_{17}, m_{19}, m_{18}; m_0, m_4, m_{12}, m_8, m_{16}, m_{20}, m_{28}, m_{24}; m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31}.

The squares are read by dropping out the variables which change. Some possible

Grouping s is

(a) m_0, $m_{16} = \overline{B}\overline{C}\overline{D}E$
(b) m_2, $m_{18} = \overline{B}\overline{C}D\overline{E}$
(c) m_4, m_6, m_{20}, $m_{22} = \overline{B}C\overline{E}$
(d) m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, $m_{31} = CE$
(e) m_8, m_9, m_{10}, m_{11}, m_{24}, m_{25}, m_{26}, $m_{27} = B\overline{C}$

M_0, $M_{16} = B + C + D + E$
M_2, $M_{18} = B + C + \overline{D} + E$
M_4, M_6, M_{20}, $M_{22} = B + \overline{C} + E$
M_5, M_7, M_{13}, M_{15}, M_{21}, M_{23}, M_{29}, $M_{31} = \overline{C} + \overline{E}$
M_8, M_9, M_{10}, M_{11}, M_{24}, M_{25}, M_{26}, $M_{27} = \overline{B} + C$



Ex: F=∑m(0,1,4,5,6,13,14,15,22,24,25,28,29,30,31) is SOP

POS is F=πM(2,3,7,8,9,10,11,12,16,17,18,19,20,21,23,26,27)

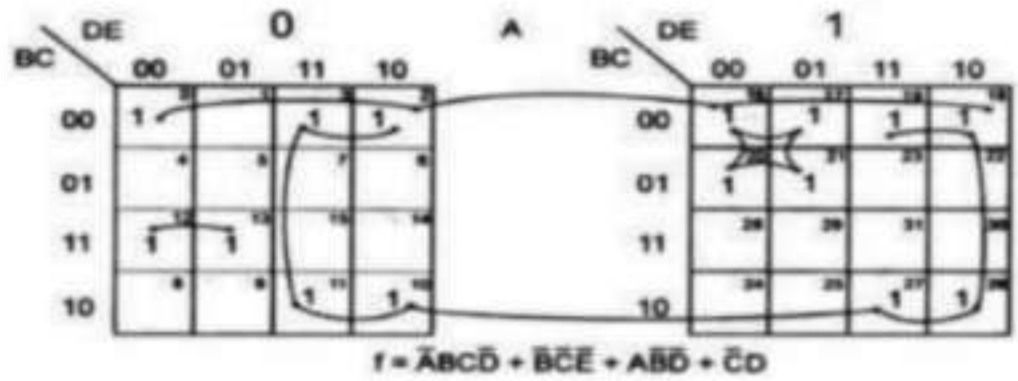
The real minimal expression is the minimal of the SOP and POS forms.

The reduction is done as

1. There is no isolated 1s
2. M_{12} can go only with m_{13}. Form a 2-square which is read as A'BCD'
3. M_0 can go with m_2,m_{16} and m_{18}. so form a 4-square which is read as B'C'E'
4. M_{20},m_{21},m_{17} and m_{16} form a 4-square which is read as AB'D'
5. M2,m3,m18,m19,m10,m11,m26 and m27 form an 8-square which is read as C'd
6. Write all the product terms in SOP form.

So the minimal expression is

F_{min}= A'BCD'+B'C'E'+AB'D'+C'D(16 inputs)



In the POS k-map ,the reduction is done as:

1. There are no isolated 0s

3. **M_1 can go only with M_5. So, make a 2-square, which is read as $(A + B + D + \bar{E})$.**
   M_4 can go with M_5, M_7, and M_6 to form a 4-square, which is read as $(A + B + \bar{C})$.

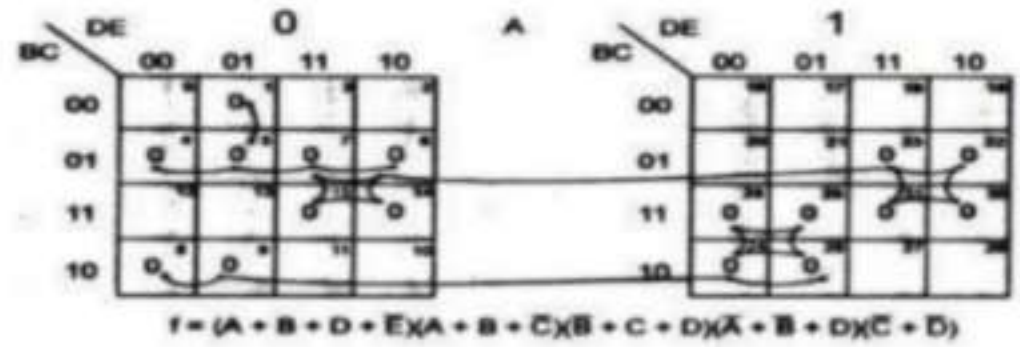
4.M_8

5. M_{28}

6.M_{30}

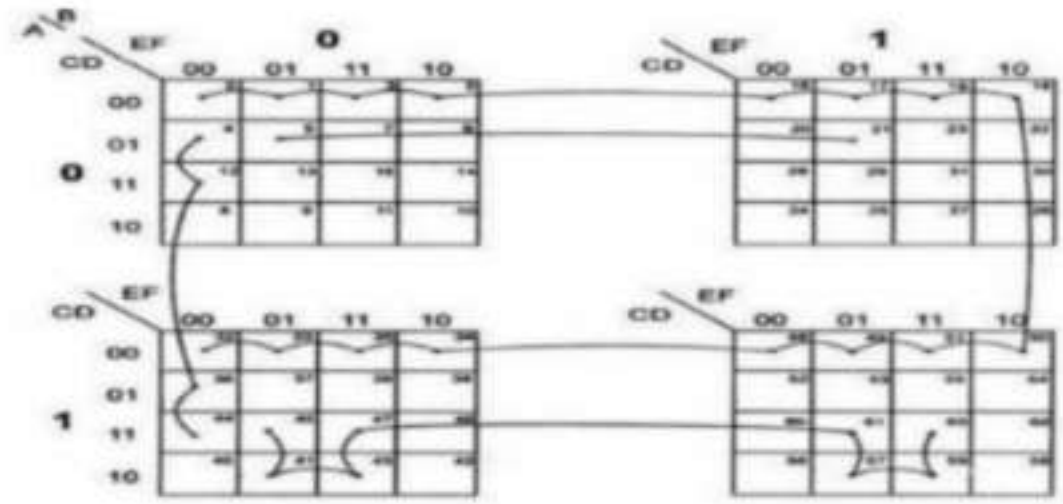
7. Sum terms in POS form. So the minimal expression in POS is

   F_{min}= A'BcD'+B'C'E'+AB'D'+C'D



Six variable k-map:

Six variable k-map can have 2^6 =64 combinations as , ,--------- ---ABCDEF with minterms m_0, m_1-----m_{63} respectively in SOP & (A+B+C+D+E+F), ---------- (+ + + + +) with maxterms M_0,M_1, ----------M_{63} respectively in POS form. It has 2^6=64 squares or cells of the k-map are divided into 4 blocks of 16 squares each.



Some possible groupings in a six variable k-map

Don't care combinations: For certain input combinations, the value of the output is unspecified either because the input combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the value of experiments are not specified are called don't care combinations are invalid or because the precise value of the output is of no consequence. The combinations for which the value of expressions is not specified are called don't care combinations or Optional Combinations, such expressions stand incompletely specified. The output is a don't care for these invalid combinations.

Ex:In XS-3 code system, the binary states 0000, 0001, 0010,1101,1110,1111 are unspecified. & never occur called don't cares.

A standard SOP expression with don't cares can be converted into a standard POS form by keeping the don't cares as they are & writing the missing minterms of the SOP form as the maxterms of the POS form viceversa.

Don't cares denoted by _X' or _φ'

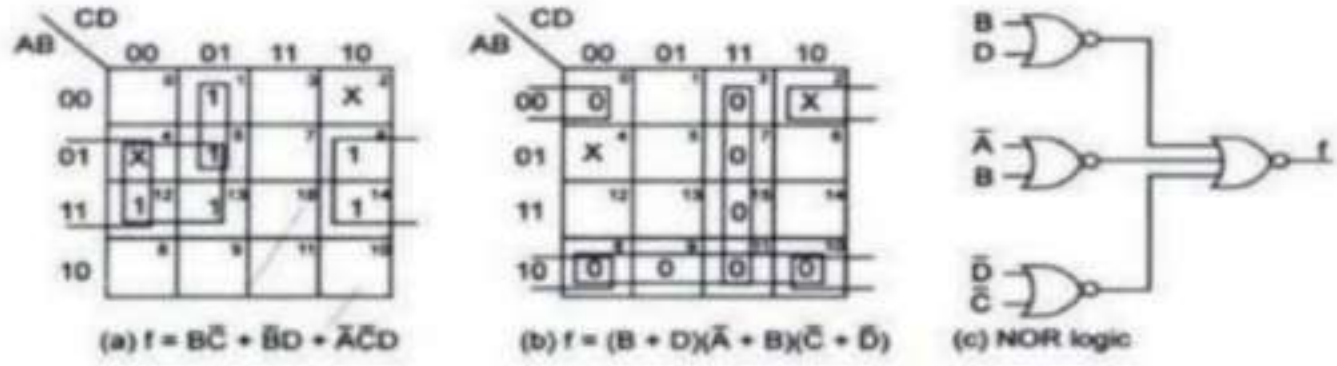
Ex: f=∑m(1,5,6,12,13,14)+d(2,4)

Or f=π M(0,3,7,9,10,11,15).πd(2,4)

SOP minimal form f_{min}= +B +

POS minimal form f_{min}=(B+D)(+B)(+D)

= + + + + (+



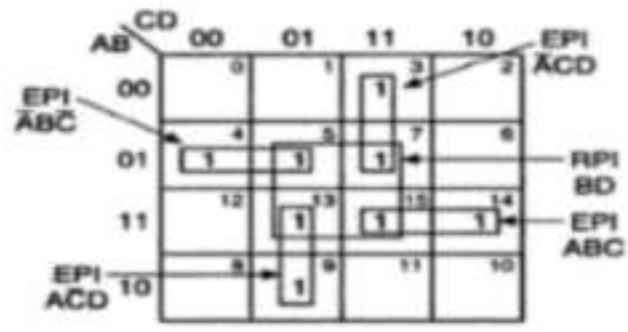
Prime implicants, Essential Prime implicants, Redundant prime implicants:

Each square or rectangle made up of the bunch of adjacent minterms is called a subcube. Each of these subcubes is called a Prime implicant (PI). The PI which contains at leastone which cannot be covered by any other prime implicants is called as Essential Prime implicant (EPI).The PI whose each 1 is covered at least by one EPI is called a Redundant Prime implicant (RPI). A PI which is neither an EPI nor a RPI is called a Selective Prime implicant (SPI).

The function has unique MSP comprising EPI is

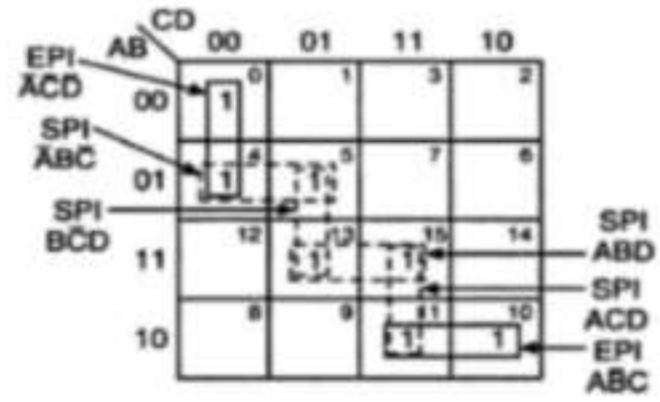
F(A,B,C,D)= CD+ABC+A D + B

The RPI _BD' may be included without changing the function but the resulting expression would not be in minimal SOP(MSP) form.



Essential and Redundant Prime Implicants

F(A,B,C,D)=∑m(0,4,5,10,11,13,15) SPI are marked by dotted squares, shows MSP form of a function need not be unique.



Essential and Selective Prime Implicants

Here, the MSP form is obtained by including two EPI's & selecting a set of SPI's to cover remaining uncovered minterms 5,13,15. & these can be covered as

(A) (4,5) &(13,15) ----------B +ABD
(B) (5,13) & (13,15) --------B D+ABD
(C) (5,13) & (15,11) -------B D+ACD

F(A,B,C,D)= +A C---------EPI's + B +ABD

(OR) F(A,B,C,D)= +A C---------EPI's + B D+ABD

(OR) F(A,B,C,D)= +A C---------EPI's + B D+ACD

False PI's Essential False PI's, Redundant False PI's & Selective False PI's:

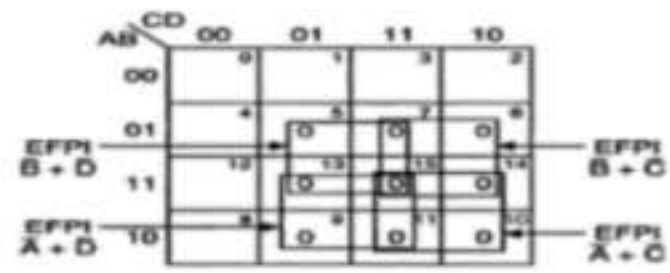
The maxterms are called falseminterms. The PI's is obtained by using the maxterms are called False PI's (FPI). The FPI which contains at least one _0' which can't be covered by only other FPI is called an Essential False Prime implicant (ESPI)

F(A,B,C,D)= ∑m(0,1,2,3,4,8,12)

=π M(5,6,7,9,10,11,13,14,15)

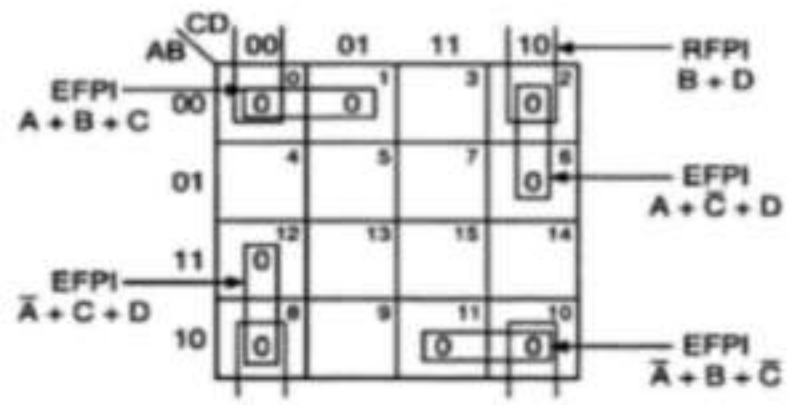
F_{min}= (+)(+)(+)(+)

All the FPI, EFPI's as each of them contain atleast one _0' which can't be covered by any other FPI



Essential False Prime implicants

Consider Function F(A,B,C,D)= π M(0,1,2,6,8,10,11,12)



Essential and Redundant False Prime Implicants

Mapping when the function is not expressed in minterms (maxterms):

An expression in k-map must be available as a sum (product) of minterms (maxterms). However if not so expressed, it is not necessary to expand the expression algebraically into its minterms (maxterms). Instead, expansion into minterms (maxterms) can be accomplished in the process of entering the terms of the expression on the k-map.

Limitations of Karnaugh maps:

- Convenient as long as the number of variables does not exceed six.
- Manual technique, simplification process is heavily dependent on the human abilities.

Quine-Mccluskey Method:

It also known as *Tabular method.* It is more systematic method of minimizing expressions of even larger number of variables. It is suitable for hand computation as well as computation by machines i.e., programmable. . The procedure is based on repeated application of the combining theorem.

PA+P =P (P is set of literals) on all adjacent pairs of terms, yields the set of all PI's from which a minimal sum may be selected.

Consider expression

$\sum m(0,1,4,5) = + C + A + A\ C$

First, second terms & third, fourth terms can be combined

(+)+ (C+)= +A

Reduced to

(+)=

The same result can be obtained by combining m_0& m_4 & m_1&m_5 in first step & resulting terms in the second step .

Procedure:

- Decimal Representation
- Don't cares
- PI chart
- EPI
- Dominating Rows & Columns
- Determination of Minimal expressions in comples cases.

Branching Method:

EXAMPLE 3.29 Obtain the set of prime implicants for the Boolean expression $f = \Sigma m(0, 1, 6, 7, 8, 9, 13, 14, 15)$ using the tabular method.

Solution

Group the minterms in terms of the number of 1s present in them and write their binary designations. The procedure to obtain the prime implicants is shown in Table 3.3.

Table 3.3 Example 3.29

	Column 1		Column 2		Column 3
	Minterm	Binary designation		A B C D	A B C D
Index 0	0	0 0 0 0 ✓	0, 1 (1)	0 0 0 – ✓	0, 1, 8, 9 (1, 8) – 0 0 – Q
Index 1	1	0 0 0 1 ✓	0, 8 (8)	– 0 0 0 ✓	... ...
	8	1 0 0 0 ✓	1, 9 (8)	– 0 0 1 ✓	... ...
Index 2	6	0 1 1 0 ✓	8, 9 (1)	1 0 0 – ✓	6, 7, 14, 15 (1, 8) – 1 1 – P
	9	1 0 0 1 ✓	6, 7 (1)	0 1 1 – ✓	
Index 3	7	0 1 1 1 ✓	6, 14 (8)	– 1 1 0 ✓	
	13	1 1 0 1 ✓	9, 13 (4)	1 – 0 1 S	
	14	1 1 1 0 ✓	7, 15 (8)	– 1 1 1 ✓	
Index 4	15	1 1 1 1 ✓	13, 15 (2)	1 1 – 1 R	
			14, 15 (1)	1 1 1 – ✓	

Comparing the terms of index 0 with the terms of index 1 of column 1, m_0(0000) is combined with m_1(0001) to yield 0, 1 (1), i.e. 000 –. This is recorded in column 2 and 0000 and 0001 are checked off in column 1. m_0(0000) is combined with m_8(1000) to yield 0, 8 (8), i.e. – 000. This is recorded in column 2 and 1000 is checked off in column 1. Note that 0000 of column 1 has already been checked off. No more combinations of terms of index 0 and index 1 are possible. So, draw a line below the last combination of these groups, i.e. below 0, 8 (8), – 000 in column 2. Now 0, 1 (1), i.e. 000 – and 0, 8 (8), i.e. – 000 are the terms in the first group of column 2.

Comparing the terms of index 1 with the terms of index 2 in column 1, m_1(0001) is combined with m_9(1001) to yield 1, 9 (8), i.e. – 001. This is recorded in column 2 and 1001 is checked off in column 1 because 0001 has already been checked off. m_8(1000) is combined with m_9(1001) to yield 8, 9 (1), i.e. 100 –. This is recorded in column 2. 1000 and 1001 of column 1 have already been checked off. So, no need to check them off again. No more combinations of terms of index 1 and index 2 are possible. So, draw a line below the last combination of these groups, i.e. 8, 9 (1), -- 001 in column 2. Now 1, 9 (8), i.e. – 001 and 8, 9 (1), i.e. 100– are the terms in the second group of column 2.

Similarly, comparing the terms of index 2 with the terms of index 3 in column 1,

m_6(0110) and m_7(0111) yield 6, 7 (1), i.e. 011–. Record it in column 2 and check off 6(0110) and 7(0111).

m_6(0110) and m_{14}(1110) yield 6, 14 (8), i.e. –110. Record it in column 2 and check off 6(0110) and 14(1110).

m_9(1001) and m_{13}(1101) yield 9, 13 (4), i.e. 1–01. Record it in column 2 and check off 9(1001) and 13(1101).

So, 6, 7 (1), i.e. 011–, and 6, 14 (8), i.e. –110 and 9, 13 (4), i.e. 1–01 are the terms in group 3 of column 2. Draw a line at the end of 9, 13 (4), i.e. 1–01.

Also, comparing the terms of index 3 with the terms of index 4 in column 1,

m_7(0111) and m_{15}(1111) yield 7, 15 (8), i.e. –111. Record it in column 2 and check off 7(0111) and 15(1111).

m_{13}(1101) and m_{15}(1111) yield 13, 15 (2), i.e. 11–1. Record it in column 2 and check off 13 and 15.

m_{14}(1110) and m_{15}(1111) yield 14, 15 (1), i.e. 111–. Record it in column 2 and check off 14 and 15.

So, 7, 15 (8), i.e. –111, and 13, 15 (2), i.e. 11–1 and 14, 15 (1), i.e. 111– are the terms in group 4 of column 2. Column 2 is completed now.

Comparing the terms of group 1 with the terms of group 2 in column 2, the terms 0, 1 (1), i.e. 000– and 8, 9 (1), i.e. 100– are combined to form 0, 1, 8, 9 (1, 8), i.e. –00–. Record it in group 1 of column 3 and check off 0, 1 (1), i.e. 000–, and 8, 9 (1), i.e. 100– of column 2. The terms 0, 8 (8), i.e. –000 and 1, 9 (8), i.e. –001 are combined to form 0, 1, 8, 9 (1, 8), i.e. –00–. This has already been recorded in column 3. So, no need to record again. Check off 0, 8 (8), i.e. –000 and 1, 9 (8), i.e. –001 of column 2. Draw a line below 0, 1, 8, 9 (1, 8), i.e. –00–. This is the only term in group 1 of column 3. No term of group 2 of column 2 can be combined with any term of group 3 of column 2. So, no entries are made in group 2 of column 2.

Comparing the terms of group 3 of column 2 with the terms of group 4 of column 2, the terms 6, 7 (1), i.e. 011–, and 14, 15 (1), i.e. 111– are combined to form 6, 7, 14, 15 (1, 8), i.e. –11–. Record it in group 3 of column 3 and check off 6, 7 (1), i.e. 011– and 14, 15 (1), i.e. 111– of column 2. The terms 6, 14 (8), i.e. –110 and 7, 15 (8), i.e. –111 are combined to form 6, 7, 14, 15 (1, 8), i.e. –11–. This has already been recorded in column 3; so, check off 6, 14 (8), i.e. –110 and 7, 15 (8), i.e. –111 of column 2.

Observe that the terms 9, 13 (4), i.e. 1–01 and 13, 15 (2), i.e. 11–1 cannot be combined with any other terms. Similarly in column 3, the terms 0, 1, 8, 9 (1, 8), i.e. –00– and 6, 7, 14, 15 (1, 8), i.e. –11– cannot also be combined with any other terms. So, these 4 terms are the prime implicants.

The terms, which cannot be combined further, are labelled as P, Q, R, and S. These form the set of prime implicants.

EX:

Obtain the minimal expression for $f = \Sigma\, m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15)$ using the tabular method.

Solution

The procedure to obtain the set of prime implicants is illustrated in Table 3.4.

Table 3.4 Example 3.30

Step 1		Step 2	Step 3	
Index 1	1 ✓	1, 3 (2) ✓	1, 3, 5, 7 (2, 4)	T
	2 ✓	1, 5 (4) ✓	1, 5, 9, 13 (4, 8)	S
	8 ✓	1, 9 (8) ✓	2, 3, 6, 7 (1, 4)	R
Index 2	3 ✓	2, 3 (1) ✓	8, 9, 12, 13 (1, 4)	Q
	5 ✓	2, 6 (4) ✓	5, 7, 13, 15 (2, 8)	P
	6 ✓	8, 9 (1) ✓		
	9 ✓	8, 12 (4) ✓		
	12 ✓	3, 7 (4) ✓		
Index 3	7 ✓	5, 7 (2) ✓		
	13 ✓	5, 13 (8) ✓		
Index 4	15 ✓	6, 7 (1) ✓		
		9, 13 (4) ✓		
		12, 13 (1) ✓		
		7, 15 (8) ✓		
		13, 15 (2) ✓		

The non-combinable terms P, Q, R, S and T are recorded as prime implicants.

$$P \rightarrow 5, 7, 13, 15\ (2, 8) = X\ 1\ X\ 1 = BD$$

(Literals with weights 2 and 8, i.e. C and A are deleted. The lowest minterm is $m_5(5 = 4 + 1)$. So, literals with weights 4 and 1, i.e. B and D are present in non-complemented form. So, read it as BD.)

$$Q \rightarrow 8, 9, 12, 13\ (1, 4) = 1\ X\ 0\ X = A\overline{C}$$

(Literals with weights 1 and 4, i.e. D and B are deleted. The lowest minterm is m_8. So, literal with weight 8 is present in non-complemented form and literal with weight 2 is present in complemented form. So, read it as $A\overline{C}$.)

$$R \rightarrow 2, 3, 6, 7\ (1, 4) = 0\ X\ 1\ X = \overline{A}C$$

(Literals with weights 1 and 4, i.e. D and B are deleted. The lowest minterm is m_2. So, literal with weight 2 is present in non-complemented form and literal with weight 8 is present in complemented form. So, read it as $\overline{A}C$.)

$$S \rightarrow 1, 5, 9, 13\ (4, 8) = X\ X\ 0\ 1 = \overline{C}D$$

(Literals with weights 4 and 8, i.e. B and A are deleted. The lowest minterm is m_1. So, literal with weight 1 is present in non-complemented form and literal with weight 2 is present in complemented form. So, read it as $\overline{C}D$.)

$$T \rightarrow 1, 3, 5, 7\ (2, 4) = 0\ X\ X\ 1 = \overline{A}D$$

(Literals with weights 2 and 4, i.e. C and B are deleted. The lowest minterm is 1. So, literal with weight 1 is present in non-complemented form and literal with weight 8 is present in complemented form. So, read it as $\overline{A}D$.)

The prime implicant chart of the expression

$$f = \Sigma\, m(1, 2, 3, 5, 6, 7, 8, 9, 12, 13, 15)$$

is as shown in Table 3.5. It consists of 11 columns corresponding to the number of minterms and 5 rows corresponding to the prime implicants P, Q, R, S, and T generated. Row R contains four ×s at the intersections with columns 2, 3, 6, and 7, because these minterms are covered by the prime implicant R. A row is said to cover the columns in which it has ×s. The problem now is to select a minimal subset of prime implicants, such that each column contains at least one × in the rows corresponding to the selected subset and the total number of literals in the prime implicants selected is as small as possible. These requirements guarantee that the number of unions of the selected prime implicants is equal to the original number of minterms and that, no other expression containing fewer literals can be found.

Table 3.5 Example 3.30: Prime implicant chart

		✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	1	2	3	5	6	7	8	9	12	13	15
*P → 5, 7, 13, 15 (2, 8)				×		×				×	×
*Q → 8, 9, 12, 13 (1, 4)							×	×	×	×	
*R → 2, 3, 6, 7 (1, 4)		×	×		×	×					
S → 1, 5, 9, 13 (4, 8)	×			×				×		×	
T → 1, 3, 5, 7 (2, 4)	×		×	×		×					

In the prime implicant chart of Table 3.5, m_2 and m_6 are covered by R only. So, R is an essential prime implicant. So, check off all the minterms covered by it, i.e. m_2, m_3, m_6, and m_7. Q is also an essential prime implicant because only Q covers m_8 and m_{12}. Check off all the minterms covered by it, i.e. m_8, m_9, m_{12}, and m_{13}. P is also an essential prime implicant, because m_{15} is covered only by P. So check off m_{15}, m_5, m_7, and m_{13} covered by it. Thus, only minterm 1 is not covered. Either row S or row T can cover it and both have the same number of literals. Thus, two minimal expressions are possible.

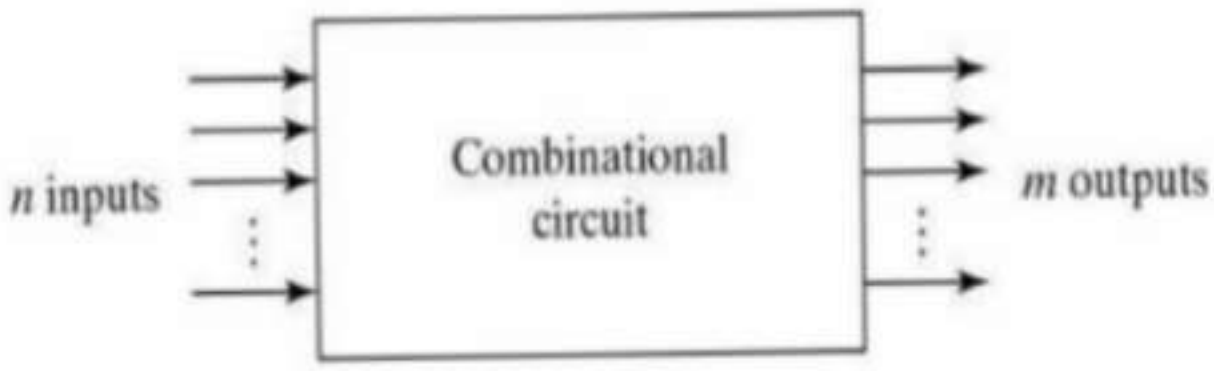
$$P + Q + R + S = BD + A\bar{C} + \bar{A}C + \bar{C}D$$

or

$$P + Q + R + T = BD + A\bar{C} + \bar{A}C + \bar{A}D$$

Combinational Logic

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables.



For n input variables,there are 2^n possible combinations of binary input variables .For each possible input Combination ,there is one and only one possible output combination.A combinational circuit can be described by m Boolean functions one for each output variables.Usually the input s comes from flip-flops and outputs goto flip-flops.

Design Procedure:

1. The problem is stated
2. The number of available input variables and required output variables is determined. 3.The input and output variables are assigned lettersymbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.

Adders:

Digital computers perform variety of information processing tasks,the one is arithmetic operations.And the most basic arithmetic operation is the addition of two binary digits.i.e, 4 basic possible operations are:

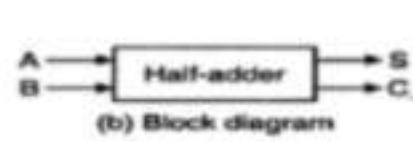
$$0+0=0,0+1=1,1+0=1,1+1=10$$

The first three operations produce a sum whose length is one digit, but when augends and addend bits are equal to 1,the binary sum consists of two digits.The higher significant bit of this result is called a carry.A combinational circuit that performs the addition of two bits is called a half-adder. One that performs the addition of 3 bits (two significant bits & previous carry) is called a full adder.& 2 half adder can employ as a full-adder.

The Half Adder: A Half Adder is a combinational circuit with two binary inputs (augends and addend bits and two binary outputs (sum and carry bits.) It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. It is an arithmetic operation of addition of two single bit words.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table



(b) Block diagram

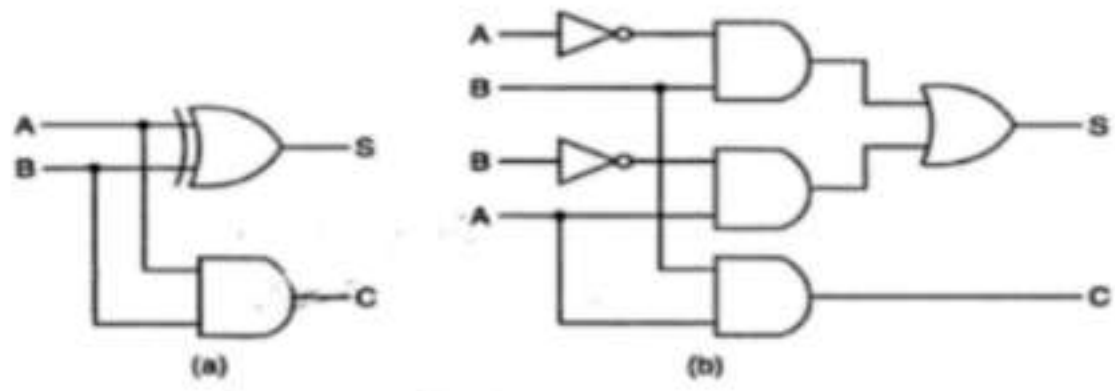
The Sum(S) bit and the carry (C) bit, according to the rules of binary addition, the sum (S) is the X-OR of A and B (It represents the LSB of the sum). Therefore,

$$S=AB+A^{A \oplus B}$$

The carry (C) is the AND of A and B (it is 0 unless both the inputs are 1).Therefore,

$$C=AB$$

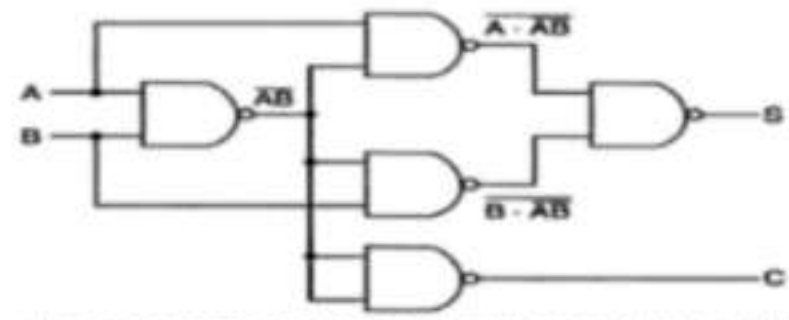
A half-adder can be realized by using one X-OR gate and one AND gate a



Logic diagrams of half-adder

NAND LOGIC:

$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$
$$= A \cdot \overline{A\bar{B}} + B \cdot \overline{A\bar{B}}$$
$$= \overline{A \cdot \overline{A\bar{B}} \cdot B \cdot \overline{A\bar{B}}}$$
$$C = AB = \overline{\overline{AB}}$$

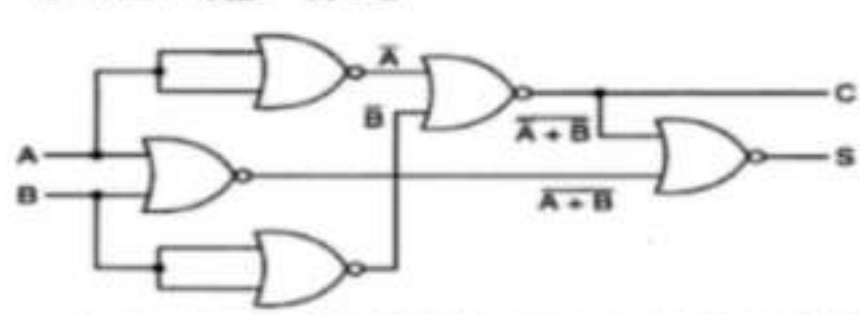


Logic diagram of a half-adder using only 2-input NAND gates.

NOR Logic:

$$S = A\bar{B} + \bar{A}B = A\bar{B} + A\bar{A} + \bar{A}B + B\bar{B}$$
$$= A(\bar{A} + \bar{B}) + B(\bar{A} + \bar{B})$$

$$= (A + B)(\bar{A} + \bar{B})$$
$$= \overline{\overline{A + B} + \overline{\bar{A} + \bar{B}}}$$
$$C = AB = \overline{\overline{AB}} = \overline{\bar{A} + \bar{B}}$$



Logic diagram of a half-adder using only 2-input NOR gates.

The Full Adder:

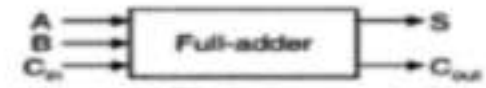
A Full-adder is a combinational circuit that adds two bits and a carry and outputs a sum bit and a carry bit. To add two binary numbers, each having two or more bits, the LSBs can be added by using a half-adder. The carry resulted from the addition of the LSBs is carried over to the next significant column and added to the two bits in that column. So, in the second and higher columns, the two data bits of that column and the carry bit generated from the addition in the previous column need to be added.

The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} . The variable S gives the value of the least significant bit of the sum. The variable C_{out} gives the output carry.The

eight rows under the input variables designate all possible combinations of 1s and 0s that these variables may have. The 1s and 0s for the output variables are determined from the arithmetic sum of the input bits. When all the bits are 0s , the output is 0. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1. The C_{out} has a carry of 1 if two or three inputs are equal to 1.

Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

(a) Truth table



(b) Block diagram

Full-adder.

From the truth table, a circuit that will produce the correct sum and carry bits in response to every possible combination of A,B and C_{in} is described by

$$S = \overline{A}\,\overline{B}C_{in} + A\overline{B}\,\overline{C_{in}} + AB\overline{C_{in}} + ABC_{in}$$

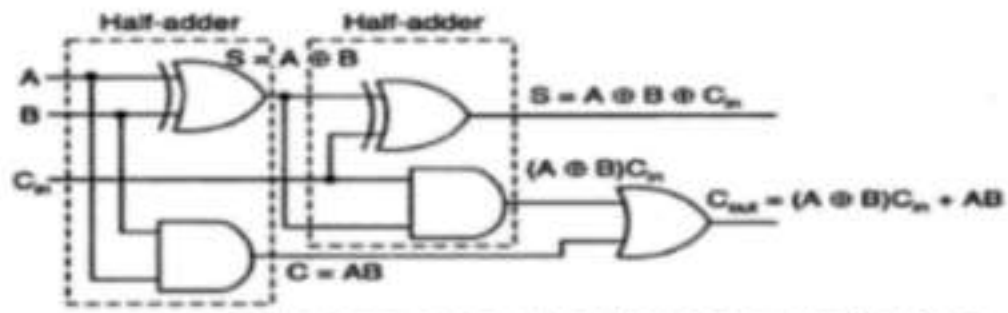
$$C_{out} = \overline{A}BC_{in} + A\overline{B}C_{in} + AB\overline{C_{in}} + ABC_{in}$$

and

$$S = A \oplus B \oplus C_{in}$$

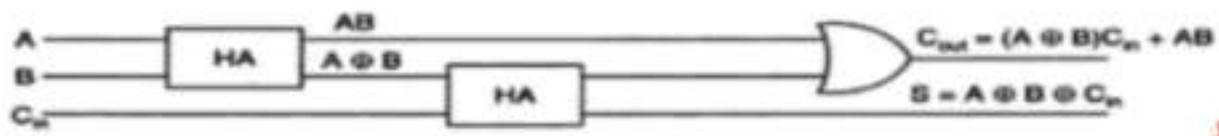
$$C_{out} = AC_{in} + BC_{in} + AB$$

The sum term of the full-adder is the X-OR of A,B, and C_{in}, i.e, the sum bit the modulo sum of the data bits in that column and the carry from the previous column. The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e, Two half adders) and one OR gate is



Logic diagram of a full-adder using two half-adders.

The block diagram of a full-adder using two half-adders is



Block diagram of a full-adder using two half-adders.

Even though a full-adder can be constructed using two half-adders, the disadvantage is that the bits must propagate through several gates in accession, which makes the total propagation delay greater than that of the full-adder circuit using AOI logic.

The Full-adder neither can also be realized using universal logic, i.e., either only NAND gates or only NOR gates as

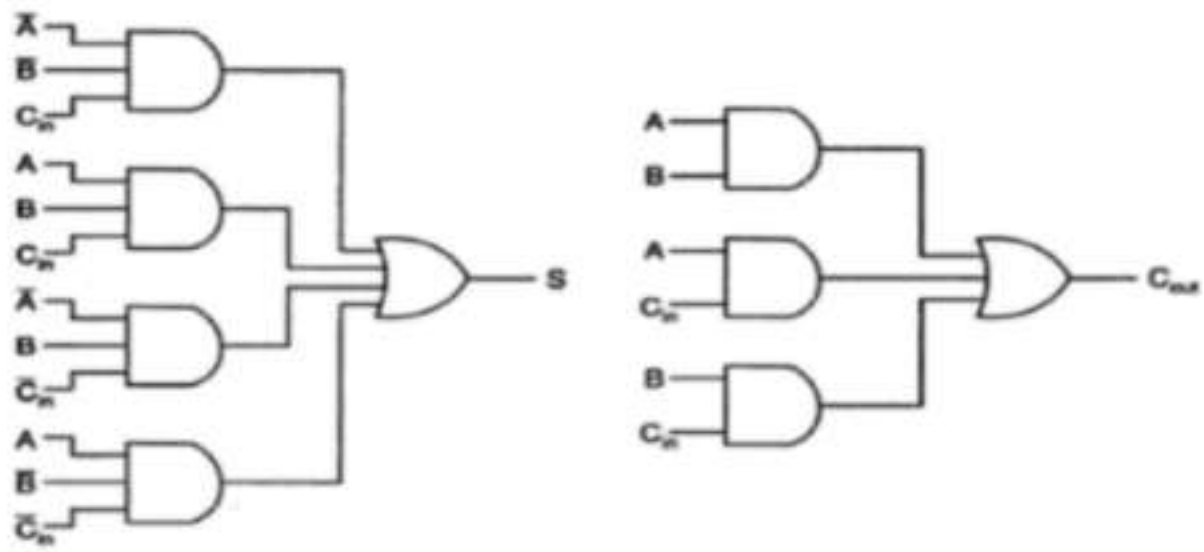
$$A \oplus B = \overline{\overline{A \cdot AB} \cdot \overline{B \cdot AB}}$$

Then

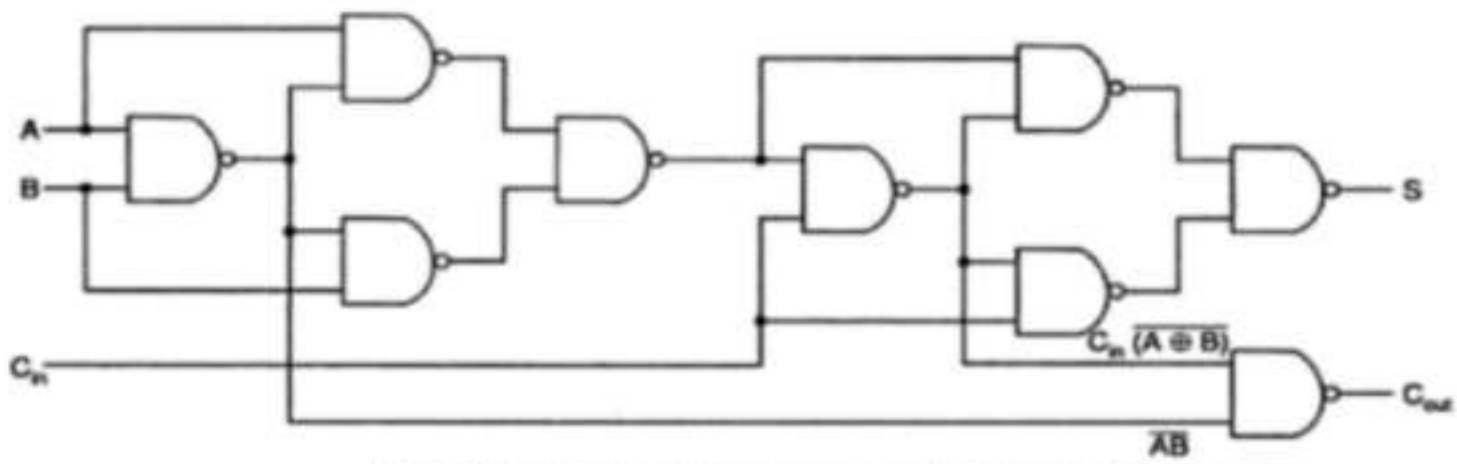
$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) \cdot (A \oplus B)C_{in}} \cdot \overline{C_{in} \cdot (A \oplus B)C_{in}}}$$

NAND Logic:

$$C_{out} = C_{in}(A \oplus B) + AB = \overline{\overline{C_{in}(A \oplus B)} \cdot \overline{AB}}$$



Sum and carry bits of a full-adder using AOI logic.



Logic diagram of a full-adder using only 2-input NAND gates.

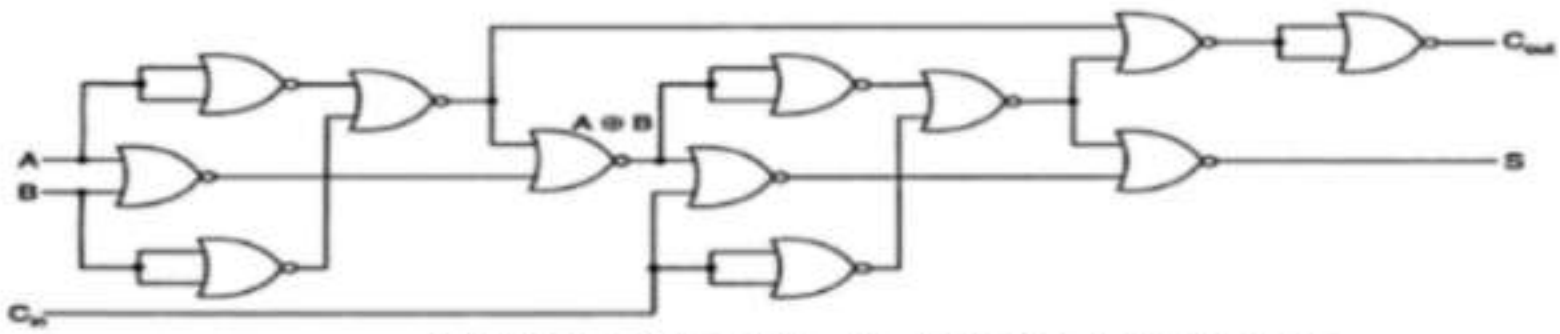
NOR Logic:

$$A \oplus B = \overline{\overline{(A + B)} + \overline{A + B}}$$

Then

$$S = A \oplus B \oplus C_{in} = \overline{\overline{(A \oplus B) + C_{in}} + \overline{(A \oplus B) + C_{in}}}$$

$$C_{out} = AB + C_{in}(A \oplus B) = \overline{\overline{A + B} + \overline{C_{in} + A \oplus B}}$$



Logic diagram of a full-adder using only 2-input NOR gates.

Subtractors:

The subtraction of two binary numbers may be accomplished by taking the complement of the subtrahend and adding it to the minuend. By this, the subtraction operation becomes an addition operation and instead of having a separate circuit for subtraction, the adder itself can be used to perform subtraction. This results in reduction of hardware. In subtraction, each subtrahend bit of the number is subtracted from its corresponding significant minuend bit to form a difference bit. If the minuend bit is smaller than the subtrahend bit, a 1 is borrowed from the next significant position., that has been borrowed must be conveyed to the next higher pair of bits by means of a signal coming out (output) of a given stage and going into (input) the next higher stage.

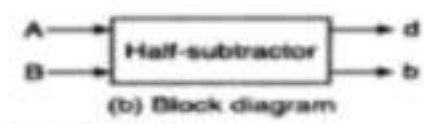
The Half-Subtractor:

A Half-subtractor is a combinational circuit that subtracts one bit from the other and produces the difference. It also has an output to specify if a 1 has been borrowed. . It is used to subtract the LSB of the subtrahend from the LSB of the minuend when one binary number is subtracted from the other.

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs d and b. d indicates the difference and b is the output signal generated that informs the next stage that a 1 has been borrowed. When a bit B is subtracted from another bit A, a difference bit (d) and a borrow bit (b) result according to the rules given as

Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

(a) Truth table



(b) Block diagram

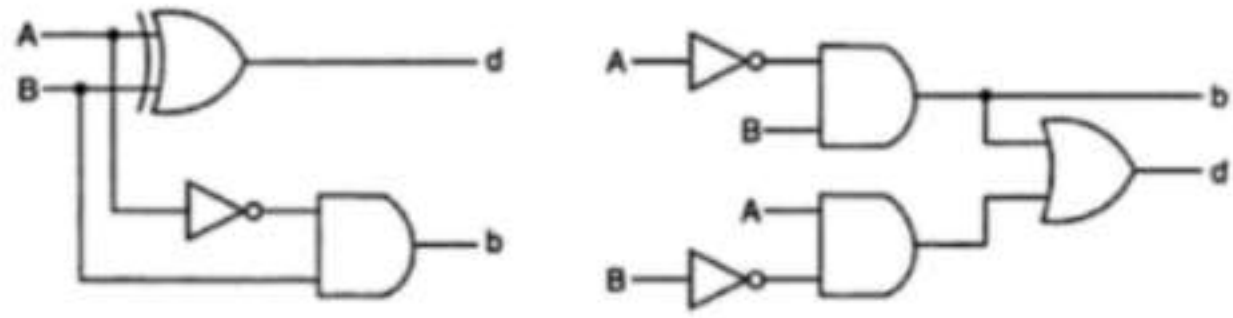
Half-subtractor.

The output borrow b is a 0 as long as A≥B. It is a 1 for A=0 and B=1. The d output is the result of the arithmetic operation2b+A-B.

A circuit that produces the correct difference and borrow bits in response to every possible combination of the two 1-bit numbers is , therefore ,

$$d=AB+A \oplus B \text{ and } b=A\,B$$

That is, the difference bit is obtained by X-OR ing the two inputs, and the borrow bit is obtained by ANDing the complement of the minuend with the subtrahend.Note that logic for this exactly the same as the logic for output S in the half-adder.



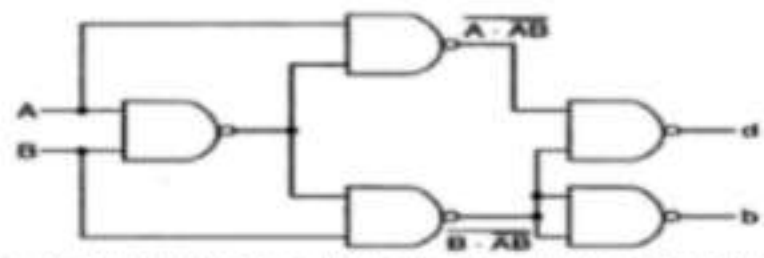
Logic diagrams of a half-subtractor.

A half-substractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

NAND Logic:

$$d = A \oplus B = \overline{\overline{A \cdot AB} \cdot \overline{B \cdot AB}}$$

$$b = \bar{A}B = B(\bar{A} + \bar{B}) = B(\overline{AB}) = \overline{\overline{B \cdot AB}}$$



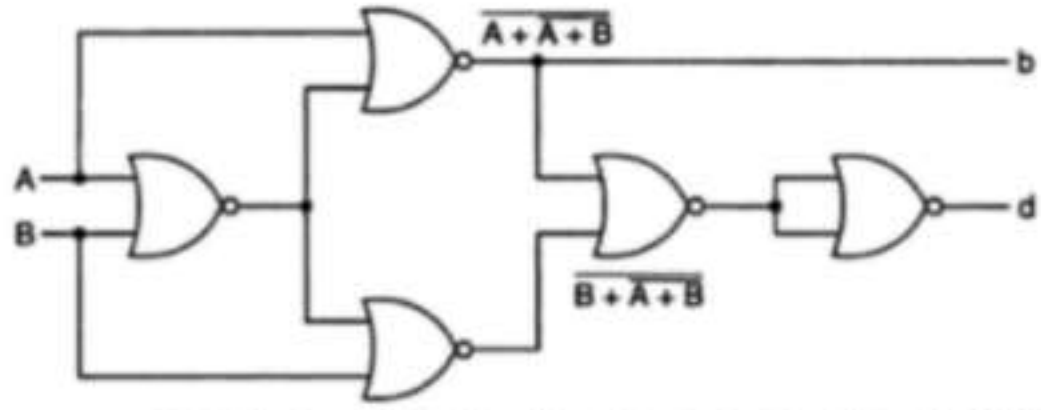
Logic diagram of a half-subtractor using only 2-input NAND gates.

NOR Logic:

$$d = A \oplus B = A\bar{B} + \bar{A}B = A\bar{B} + B\bar{B} + \bar{A}B + A\bar{A}$$

$$= \bar{B}(A + B) + \bar{A}(A + B) = \overline{\overline{B + \overline{A + B}} + \overline{A + \overline{A + B}}}$$

$$d = \bar{A}B = \bar{A}(A + B) = \overline{\overline{\bar{A}(A + B)}} = \overline{A + (\overline{A + B})}$$



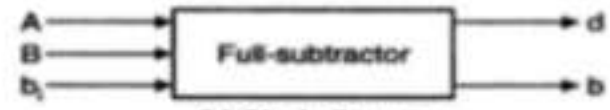
Logic diagram of a half-subtractor using only 2-input NOR gates.

The Full-Subtractor:

The half-subtractor can be only for LSB subtraction. IF there is a borrow during the subtraction of the LSBs, it affects the subtraction in the next higher column; the subtrahend bit is subtracted from the minuend bit, considering the borrow from that column used for the subtraction in the preceding column. Such a subtraction is performed by a full-subtractor. It subtracts one bit (B) from another bit (A) , when already there is a borrow b_i from this column for the subtraction in the preceding column, and outputs the difference bit (d) and the borrow bit(b) required from the next d and b. The two outputs present the difference and output borrow. The 1s and 0s for the output variables are determined from the subtraction of A-B-b_i.

Inputs			Difference	Borrow
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

(a) Truth table



(b) Block diagram

Full-subtractor.

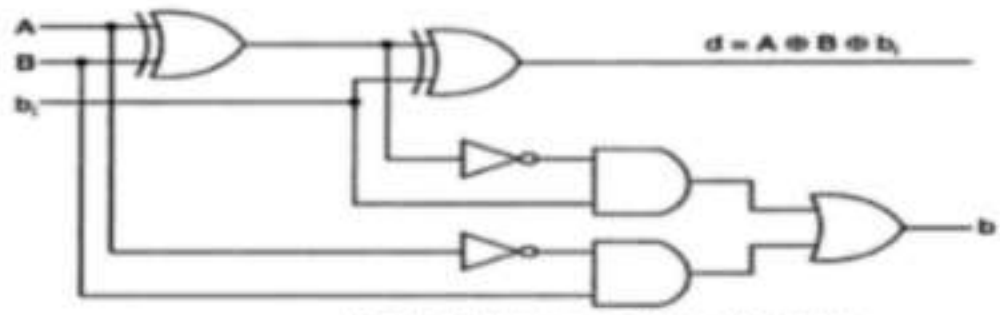
From the truth table, a circuit that will produce the correct difference and borrow bits in response to every possiblecombinations of A,B and b_i is

$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i$$
$$= b_i(AB + \bar{A}\bar{B}) + \bar{b}_i(A\bar{B} + \bar{A}B)$$
$$= b_i(\overline{A \oplus B}) + \bar{b}_i(A \oplus B) = A \oplus B \oplus b_i$$

and

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b}_i) + (AB + \bar{A}\bar{B})b_i$$
$$= \bar{A}B + (\overline{A \oplus B})b_i$$

A full-subtractor can be realized using X-OR gates and AOI gates as



Logic diagram of a full-subtractor.

The full subtractor can also be realized using universal logic either using only NAND gates or using NOR gates as:

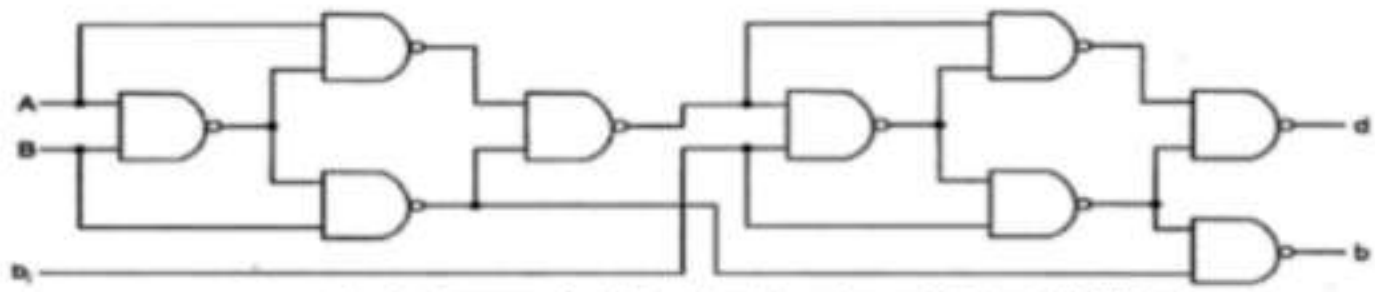
NAND Logic:

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}} = \overline{\overline{(A \oplus B)(\overline{A \oplus B})b_i} \cdot \overline{b_i(\overline{A \oplus B})b_i}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B}) = \overline{\overline{\bar{A}B + b_i(\overline{A \oplus B})}}$$

$$= \overline{\overline{\bar{A}B} \cdot \overline{b_i(\overline{A \oplus B})}} = \overline{\overline{B(\bar{A} + \bar{B})} \cdot \overline{b_i(\bar{b}_i + (\overline{A \oplus B}))}}$$

$$= \overline{B \cdot \overline{AB} \cdot b_i[b_i \cdot (\overline{A \oplus B})]}$$



Logic diagram of a full-subtractor using only 2-input NAND gates.

NOR Logic:

$$d = A \oplus B \oplus b_i = \overline{\overline{(A \oplus B) \oplus b_i}}$$

$$= \overline{\overline{(A \oplus B)b_i} + \overline{(\overline{A \oplus B})\bar{b}_i}}$$

$$= \overline{[(A \oplus B) + \overline{(\overline{A \oplus B})\bar{b}_i}][b_i + \overline{(\overline{A \oplus B})\bar{b}_i}]}$$

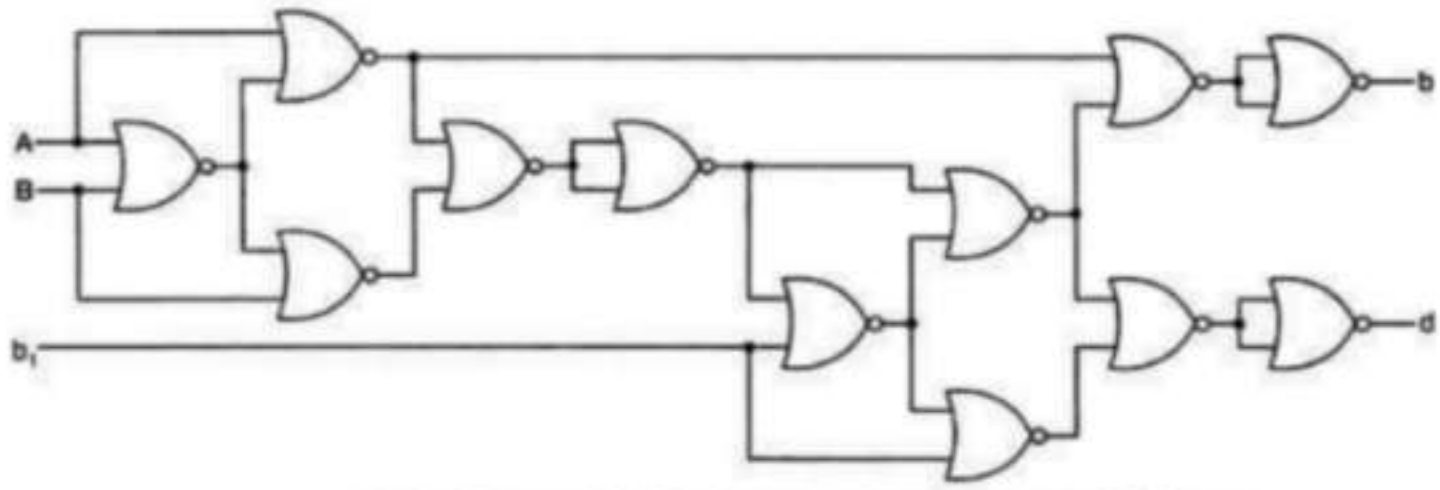
$$= \overline{\overline{(A \oplus B) + \overline{(\overline{A \oplus B}) + b_i}} + \overline{b_i + \overline{(\overline{A \oplus B}) + b_i}}}$$

$$= \overline{\overline{(A \oplus B) + \overline{(\overline{A \oplus B}) + b_i}} + \overline{b_i + \overline{(\overline{A \oplus B}) + b_i}}}$$

$$b = \bar{A}B + b_i(\overline{A \oplus B})$$

$$= \bar{A}(A + B) + (\overline{A \oplus B})[(A \oplus B) + b_i]$$

$$= \overline{\overline{A + \overline{(A + B)}} + \overline{(A \oplus B) + \overline{(A \oplus B) + b_i}}}$$

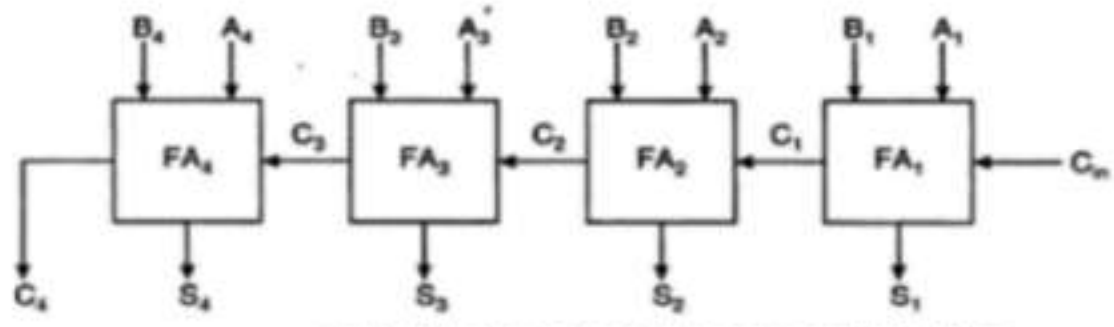


Logic diagram of a full subtractor using only 2-input NOR gates.

Binary Parallel Adder:

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form. It consists of full adders connected in a chain , with the output carry from each full-adder connected to the input carry of the next full-adder in the chain.

The interconnection of four full-adder (FA) circuits to provide a 4-bit parallel adder. The augends bits of A and addend bits of B are designated by subscript numbers from right to left, with subscript 1 denoting the lower –order bit. The carries are connected in a chain through the full-adders. The input carry to the adder is C_{in} and the output carry is C_4. The S output generates the required sum bits. When the 4-bit full-adder circuit is enclosed within an IC package, it has four terminals for the augends bits, four terminals for the addend bits, four terminals for the sum bits, and two terminals for the input and output carries. AN n-bit parallel adder requires n-full adders. It can be constructed from 4-bit, 2-bit and 1-bit full adder ICs by cascading several packages. The output carry from one package must be connected to the input carry of the one with the next higher –order bits. The 4-bit full adder is a typical example of an MSI function.



Logic diagram of a 4-bit binary parallel adder.

Ripple carry adder:

In the parallel adder, the carry –out of each stage is connected to the carry-in of the next stage. The sum and carry-out bits of any stage cannot be produced, until sometime after the carry-in of that stage occurs. This is due to the propagation delays in the logic circuitry.

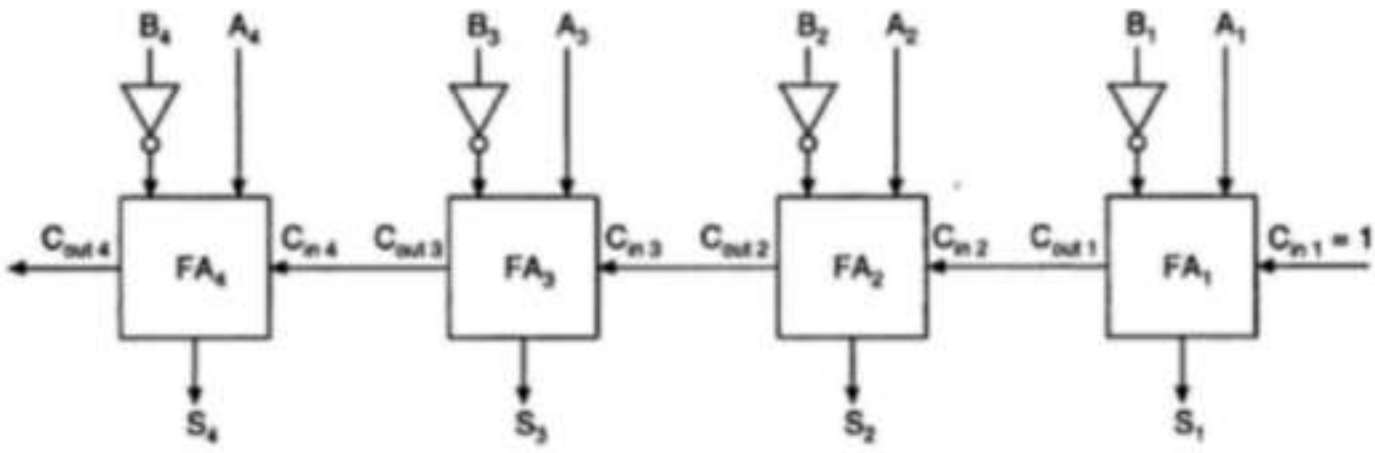
which lead to a time delay in the addition process. The carry propagation delay for each full-adder is the time between the application of the carry-in and the occurrence of the carry-out.

The 4-bit parallel adder, the sum (S_1) and carry-out (C_1) bits given by FA_1 are not valid, until after the propagation delay of FA_1. Similarly, the sum S_2 and carry-out (C_2) bits given by FA_2 are not valid until after the cumulative propagation delay of two full adders (FA_1 and FA_2) , and so on. At each stage ,the sum bit is not valid until after the carry bits in all the preceding stages are valid. Carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total sum (the parallel output) is not valid until after the cumulative delay of all the adders.

The parallel adder in which the carry-out of each full-adder is the carry-in to the next most significant adder is called a ripple carry adder.. The greater the number of bits that a ripple carry adder must add, the greater the time required for it to perform a valid addition. If two numbers are added such that no carries occur between stages, then the add time is simply the propagation time through a single full-adder.

4-Bit ParallelSubtractor:

The subtraction of binary numbers can be carried out most conveniently by means of complements , the subtraction A-B can be done by taking the 2's complement of B and adding it to A . The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters as

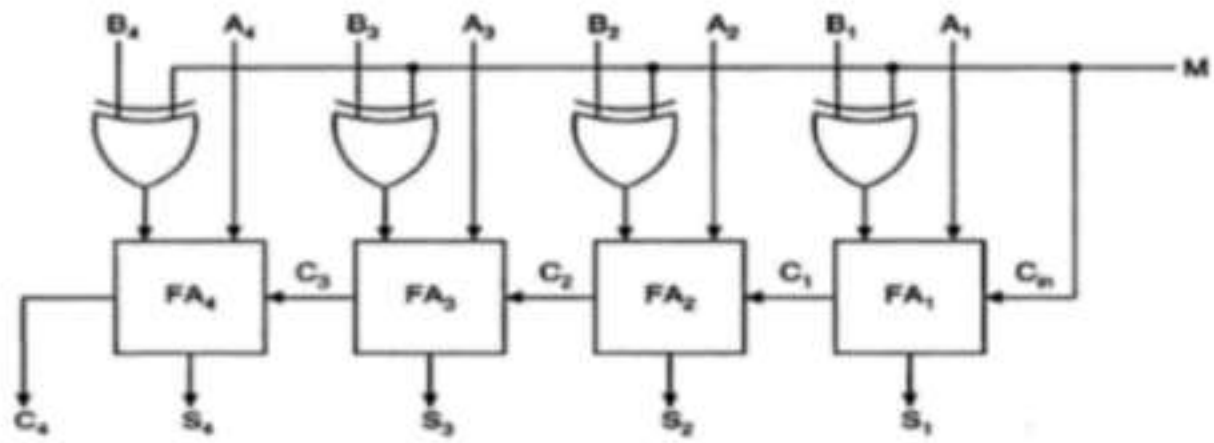


Logic diagram of a 4-bit parallel subtractor.

Binary-Adder Subtractor:

A 4-bit adder-subtractor, the addition and subtraction operations are combined into one circuit with one common binary adder. This is done by including an X-OR gate with each full-adder. The mode input M controls the operation. When M=0, the circuit is an adder, and when M=1, the circuit becomes a subtractor. Each X-OR gate receives input M and one of the inputs of B. When M=0, $B \oplus 0 = B$ .The full-adder receives the value of B , the input carry is 0

and the circuit performs A+B. when $B \oplus 1 = B'$ and $C_1=1$. The B inputs are complemented and a 1 is through the input carry. The circuit performs the operation A plus the 2's complement of B.



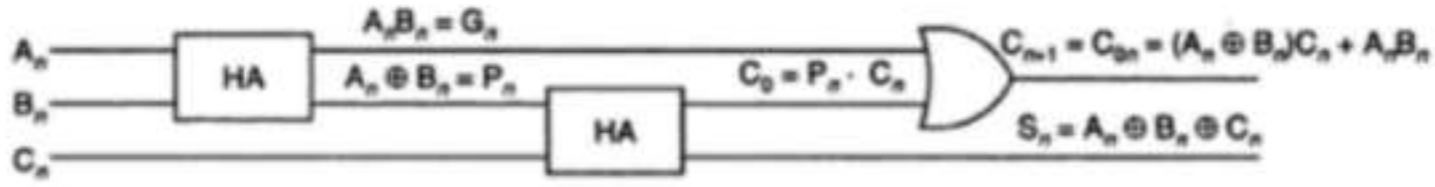
Logic diagram of a 4-bit binary adder-subtractor.

The Look-Ahead –Carry Adder:

In parallel-adder,the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder. The look-ahead carry adder speeds up the process by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry-in bits for all the stages simultaneously.

The method of speeding up the addition process is based on the two additional functions of the full-adder, called the carry generate and carry propagate functions.

Consider one full adder stage; say the nth stage of a parallel adder as shown in fig. we know that is made by two half adders and that the half adder contains an X-OR gate to produce the sum and an AND gate to produce the carry. If both the bits A_n and B_n are 1s, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. This is called generated carry, expressed as $G_n = A_n.B_n$ which has to appear at the output through the OR gate as shown in fig.



A full adder (*n*th stage of a parallel adder).

Thereis another possibility of producing a carry out. X-OR gate inside the half-adder at the input produces an intermediary sum bit- call it P_n –which is expressed as $P_n = A_n \oplus B_n$. Next P_n and C_n are added using the X-OR gate inside the second half adder to produce the final

sum bit and $S_n = P_n \oplus C_n$ where $P_n = A_n \oplus B_n$ and output carry$C_0 = P_n.C_n = (A_n \oplus B_n)C_n$ which becomes carry for the (n+1) thstage.

Consider the case of both P_n and C_n being 1. The input carry C_n has to be propagated to the output only if P_n is 1. If P_n is 0, even if C_n is 1, the and gate in the second half-adder will inhibit C_n. the carry out of the nth stage is 1 when either $G_n=1$ or $P_n.C_n=1$ or both G_n and $P_n.C_n$ are equal to 1.

For the final sum and carry outputs of the nth stage, we get the following Boolean expressions.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$

$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

Observe the recursive nature of the expression for the output carry at the nth stage which becomes the input carry for the (n+1)st stage .it is possible to express the output carry of a higher significant stage is the carry-out of the previous stage.

Based on these , the expression for the carry-outs of various full adders are as follows,

$$C_1 = G_0 + P_0 \cdot C_0$$

$$C_2 = G_1 + P_1 \cdot C_1 = G_1 + P_1 \cdot G_0 + P_1 \cdot P_0 \cdot C_0$$

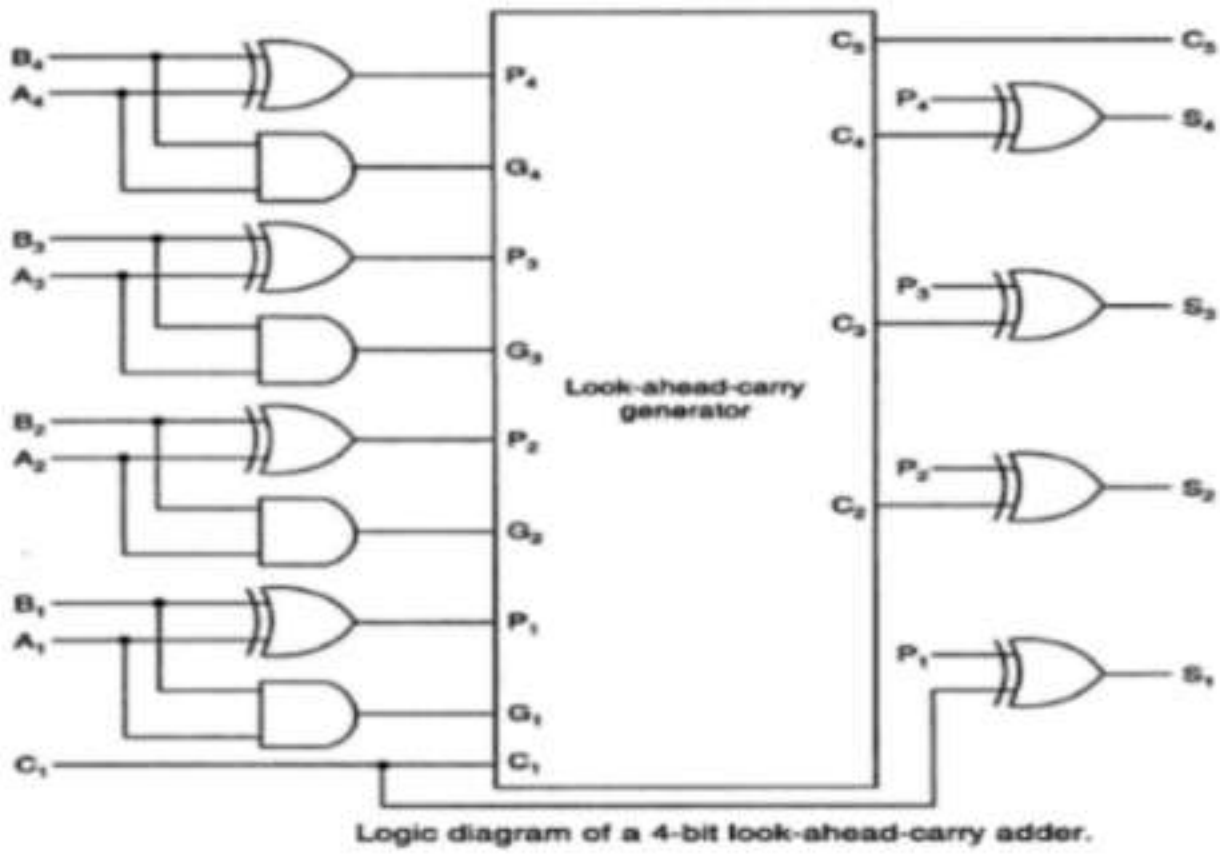
$$C_3 = G_2 + P_2 \cdot C_2 = G_2 + P_2 \cdot G_1 + P_2 \cdot P_1 \cdot G_0 + P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

$$C_4 = G_3 + P_3 \cdot C_3 = G_3 + P_3 \cdot G_2 + P_3 \cdot P_2 \cdot G_1 + P_3 \cdot P_2 \cdot P_1 \cdot G_0 + P_3 \cdot P_2 \cdot P_1 \cdot P_0 \cdot C_0$$

The general expression for *n* stages designated as 0 through (*n* – 1) would be

$$C_n = G_{n-1} + P_{n-1} \cdot C_{n-1} = G_{n-1} + P_{n-1} \cdot G_{n-2} + P_{n-1} \cdot P_{n-2} \cdot G_{n-3} + \cdots + P_{n-1} \cdot \cdots P_0 \cdot C_0$$

Observe that the final output carry is expressed as a function of the input variables in SOP form. Which is two level AND-OR or equivalent NAND-NAND form. Observe that the full look-ahead scheme requires the use of OR gate with (n+1) inputs and AND gates with number of inputs varying from 2 to (n+1).



Logic diagram of a 4-bit look-ahead-carry adder.

2's complement Addition and Subtraction using Parallel Adders:

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction operations of signed numbers can be performed using only the addition operation ,if we use the 2's complement form to represent negative numbers.

The circuit shown can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal ADD/SUB'. When the ADD/SUB' level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the ADD/Sub' level is LOW, the circuit subtract the number in register B from the number in register A. The operation is:

When ADD/SUB' is a 1:

1. AND gates 1,3,5 and 7 are enabled , allowing B_0,B_1,$B_2$and B_3 to pass to the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are disabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates 9,10,11 and 12.

2. The two levels B_0 to B_3 pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3. The sum appears at the output S_0 to S_3

3. Add/SUB' =1 causes no carry into the adder.

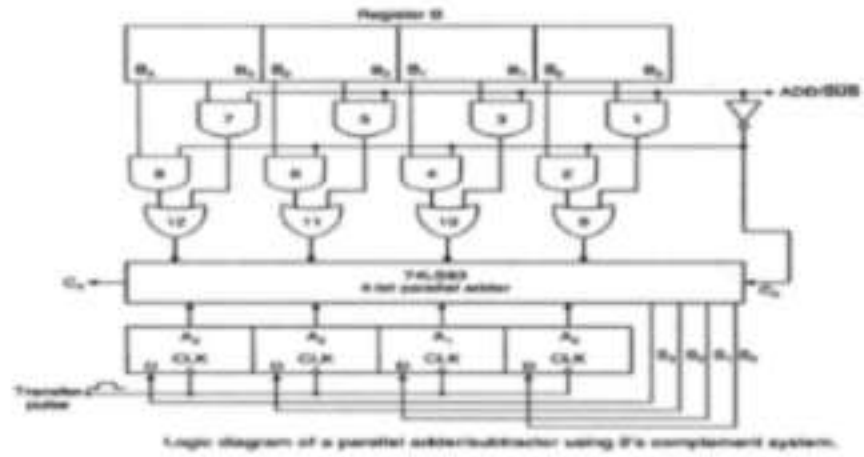
When ADD/SUB' is a 0:

1. AND gates 1,3,5 and 7 are disabled , allowing B_0,B_1,$B_2$and B_3 from reaching the OR gates 9,10,11,12 . AND gates 2,4,6 and 8 are enabled , blocking B_0',B_1',B_2', and B_3' from reaching the OR gates.

2. The two levels B_0' to B_3' pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3.The C_0 is now 1.thus the number in register B is converted to its 2's complement form.

3. The difference appears at the output S_0 toS_3.

Adders/Subtractors used for adding and subtracting signed binary numbers. In computers , the output is transferred into the register A (accumulator) so that the result of the addition or subtraction always end up stored in the register A This is accomplished by applying a transfer pulse to the CLK inputs of register A.



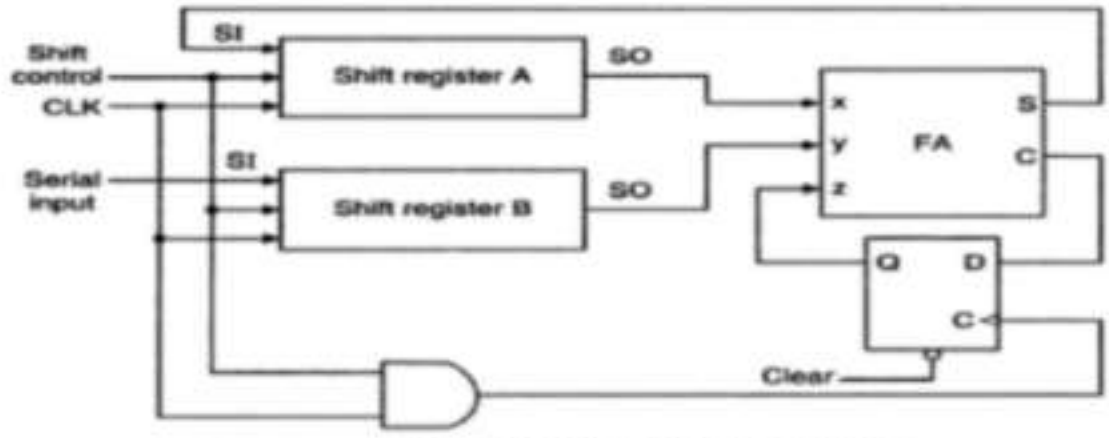
Serial Adder:

A serial adder is used to add binary numbers in serial form. The two binary numbers to be added serially are stored in two shift registers A and B. Bits are added one pair at a time through a single full adder (FA) circuit as shown. The carry out of the full-adder is transferred to a D flip-flop. The output of this flip-flop is then used as the carry input for the next pair of significant bits. The sum bit from the S output of the full-adder could be transferred to a third shift register. By shifting the sum into A while the bits of A are shifted out, it is possible to use one register for storing both augend and the sum bits. The serial input register B can be used to transfer a new binary number while the addend bits are shifted out during the addition.

The operation of the serial adder is:

Initially register A holds the augend, register B holds the addend and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full-adder at x and y. The shift control enables both registers and carry flip-flop , so, at the clock pulse both registers are shifted once to the right, the sum bit from S enters the left most flip-flop of A , and the output carry is transferred into flip-flop Q . The shift control enables the registers for a number of clock pulses equal to the number of bits of the registers. For each succeeding clock pulse a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled. Thus the addition is accomplished by passing each pair of bits together with the previous carry through a single full adder circuit and transferring the sum, one bit at a time, into register A.

Initially, register A and the carry flip-flop are cleared to 0 and then the first number is added from B. While B is shifted through the full adder, a second number is transferred to it through its serial input. The second number is then added to the content of register A while a third number is transferred serially into register B. This can be repeated to form the addition of two, three, or more numbers and accumulate their sum in register A.



Logic diagram of a serial adder.

Difference between Serial and Parallel Adders:

The parallel adder registers with parallel load, whereas the serial adder uses shift registers. The number of full adder circuits in the parallel adder is equal to the number of bits in the binary numbers, whereas the serial adder requires only one full adder circuit and a carry flip-flop. Excluding the registers, the parallel adder is a combinational circuit, whereas the serial adder is a sequential circuit. The sequential circuit in the serial adder consists of a full-adder and a flip-flop that stores the output carry.

BCD Adder:

The BCD addition process:

1. Add the 4-bit BCD code groups for each decimal digit position using ordinary binary addition.
2. For those positions where the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
3. When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimalposition.

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following:

1. Add two 4-bit BCD code groups, usingstraight binaryaddition.

2. Determine, if the sum of this addition is greater than 1101 (decimal 9); if it is, add 0110 (decimal 6) to this sum and generate a carry to the next decimal position.

The first requirement is easily met by using a 4- bit binary parallel adder such as the 74LS83 IC .For example , if the two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are applied to a 4-bit parallel adder, the adder will output $S_4S_3S_2S_1S_0$, where S_4 is actually C_4 , the carry –out of the MSB bits.

The sum outputs $S_4S_3S_2S_1S_0$ can range anywhere from 00000 to 100109when both the BCD code groups are 1001=9). The circuitry for a BCD adder must include the logic needed to detect whenever the sum is greater than 01001, so that the correction can be added in. Those cases , where the sum is greater than 1001 are listed as:

S_4	S_3	S_2	S_1	S_0	Decimal number
0	1	0	1	0	10
0	1	0	1	1	11
0	1	1	0	0	12
0	1	1	0	1	13
0	1	1	1	0	14
0	1	1	1	1	15
1	0	0	0	0	16
1	0	0	0	1	17
1	0	0	1	0	18

Let us define a logic output X that will go HIGH only when the sum is greater than 01001 (i.e, for the cases in table). If examine these cases ,see that X will be HIGH for either of the following conditions:

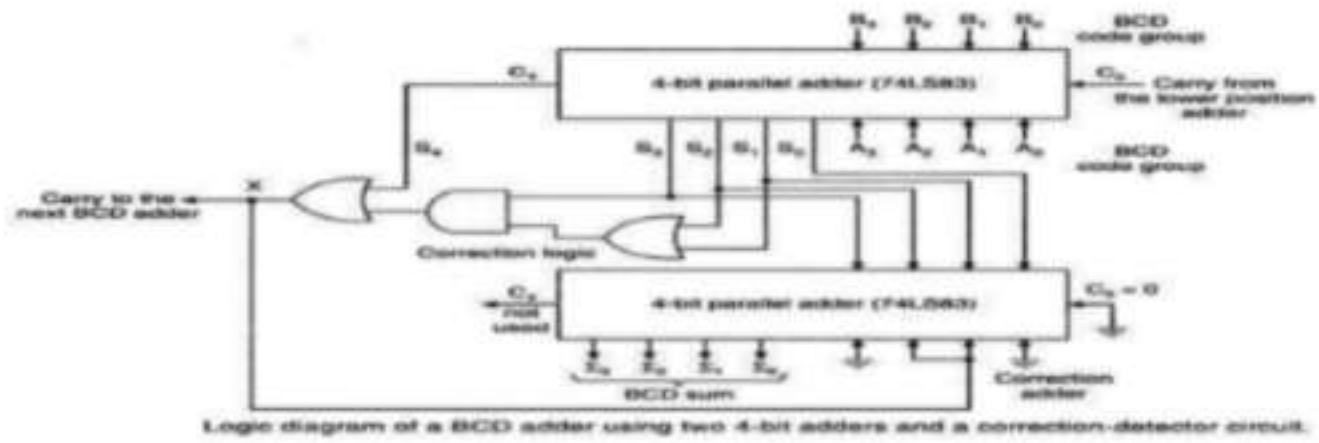
1. Whenever S_4 =1(sum greater than 15)
2. Whenever S_3 =1 and either S_2 or S_1 or both are 1 (sum 10 to 15)

This condition can be expressed as

$$X=S_4+S_3(S_2+S_1)$$

Whenever X=1, it is necessary to add the correction factor 0110 to the sum bits, and to generate a carry. The circuit consists of three basic parts. The two BCD code groups $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are added together in the upper 4-bit adder, to produce the sum $S_4S_3S_2S_1S_0$. The logic gates shown implement the expression for X. The lower 4-bit adder will add the correction 0110 to the sum bits, only when X=1, producing the final BCD sum output represented by $\sum_3\sum_2\sum_1\sum_0$. The X is also the carry-out that is produced when the sum is greater than 01001. When X=0, there is no carry and no addition of 0110. In such cases, $\sum_3\sum_2\sum_1\sum_0= S_3S_2S_1S_0$.

Two or more BCD adders can be connected in cascade when two or more digit decimal numbers are to be added. The carry-out of the first BCD adder is connected as the carry-in of the second BCD adder, the carry-out of the second BCD adder is connected as the carry-in of the third BCD adder and so on.



Logic diagram of a BCD adder using two 4-bit adders and a correction-detector circuit.

EXCESS-3(XS-3) ADDER:

To perform Excess-3 additions,

1. Add two xs-3 codegroups
2. If carry=1, add 0011(3) to the sum of those two codegroups
   If carry =0, subtract 0011(3) i.e., add 1101 (13 in decimal) to the sum of those two code groups.
   Ex: Add 9 and 5

	1100	9 in Xs-3
	+1000	5 in xs-3

1	0100	there is a carry
+0011	0011	add 3 to each group
---------	---------	
0100	0111	14 in xs-3
(1)	(4)	

EX:

(b)	0 1 1 1	4 in XS-3
	+0 1 1 0	3 in XS-3
	1 1 0 1	no carry
	+1 1 0 1	Subtract 3 (i.e. add 13)
Ignore carry	1 1 0 1 0	7 in XS-3
	(7)	

Implementation of xs-3 adder using 4-bit binary adders is shown. The augend (A_3 $A_2A_1A_0$) and addend ($B_3B_2B_1B_0$) in xs-3 are added using the 4-bit parallel adder. If the carry is a 1, then 0011(3) is added to the sum bits $S_3S_2S_1S_0$ of the upper adder in the lower 4-bit parallel

adder. If the carry is a 0, then 1101(3) is added to the sum bits (This is equivalent to subtracting 0011(3) from the sum bits. The correct sum in xs-3 is obtained

Excess-3 (XS-3) Subtractor:

To perform Excess-3 subtraction,

1. Complement thesubtrahend
2. Add the complemented subtrahend to theminuend.
3. If carry =1, result is positive. Add 3 and end around carry to the result . Ifcarry=0, the result is negative. Subtract 3, i.e, and take the 1's complement of the result.

Ex: Perform 9-4

	1100	9 in xs-3
	+1000	Complement of 4 n Xs-3
	--------	
(1)	0100	There is a carry
	+0011	Add 0011(3)

	0111	
	1	End around carry

	1000	5 in xs-3

The minuend and the 1's complement of the subtrahend in xs-3 are added in the upper 4-bit parallel adder. If the carry-out from the upper adder is a 0, then 1101 is added to the sum bits of the upper adder in the lower adder and the sum bits of the lower adder are complemented to get the result. If the carry-out from the upper adder is a 1, then 3=0011 is added to the sum bits of the lower adder and the sum bits of the lower adder give the result.

Binary Multipliers:

In binary multiplication by the paper and pencil method, is modified somewhat in digital machines because a binary adder can add only two binary numbers at a time.
In a binary multiplier, instead of adding all the partial products at the end, they are added two at a time and their sum accumulated in a register (the accumulator register). In addition, when the multiplier bit is a 0,0s are not written down and added because it does not affect the final result. Instead, the multiplicand is shifted left by one bit.

The multiplication of 1110 by 1001 using this processis

Multiplicand 1110		
Multiplier	1001	
	1110	The LSB of the multiplier is a 1; write down the multiplicand; shift the multiplicand one position to the left (1 1 1 0 0)
	1110	The second multiplier bit is a 0; write down the previous result 1110; shift the multiplicand to the left again (1 1 1 0 0 0)

+1110000 The fourth multiplier bit is a 1 write down the new multiplicand add it to the first partial product to obtain the final product.

1111110

This multiplication process can be performed by the serial multiplier circuit , which multiplies two 4-bit numbers to produce an 8-bit product. The circuit consists of following elements

X register: A 4-bit shift register that stores the multiplier --- it will shift right on the falling edge of the clock. Note that 0s are shifted in from the left.

B register: An 8-bit register that stores the multiplicand; it will shift left on the falling edge of the clock. Note that 0s are shifted in from the right.

A register: An 8-bit register, i.e, the accumulator that accumulates the partial products.

Adder: An 8-bit parallel adder that produces the sum of A and B registers. The adder outputs S_7 through S_0 are connected to the D inputs of the accumulator so that the sum can be transferred to the accumulator only when a clock pulse gets through the AND gate.

The circuit operation can be described by going through each step in the multiplication of 1110 by 1001. The complete process requires 4 clock cycles.

1. **Before the first clock pulse**: Prior to the occurrence of the first clock pulse, the register A is loaded with 00000000, the register B with the multiplicand 00001110, and the register X with the multiplier 1001. Assume that each of these registers is loaded using its asynchronous inputs(i.e., PRESET and CLEAR). The output of the adder will be the sum of A and B,i.e., 00001110.
2. **First Clock pulse**:Since the LSB of the multiplier (X_0) is a 1, the first clock pulse gets through the AND gate and its positive going transition transfers the sum outputs into the accumulator. The subsequent negative going transition causes the X and B registers to shift right and left, respectively. This produces a new sum of AandB.
3. **Second Clock Pulse:** The second bit of the original multiplier is now in X_0. Since this bit is a 0, the second clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.
4. **Third Clock Pulse**:The third bit of the original multiplier is now in X_0;since this bit is a 0, the third clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.
5. **Fourth Clock Pulse:** The last bit of the original multiplier is now in X_0 , and since it is a 1, the positive going transition of the fourth pulse transfers the sum into the accumulator. The accumulator now holds the final product. The negative going transition of the clock pulse shifts X and B again. Note that, X is now 0000, since all the multiplier bits have been shifted out.

Code converters:

The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems. It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be

inserted between the two systems if each uses different codes for the same information. Thus a code converter is a logic circuit whose inputs are bit patterns representing numbers (or character) in one cod and whose outputs are the corresponding representation in a different code. Code converters are usually multiple output circuits.

To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and the output lines must generate the corresponding bit combination of code B. A combinational circuit performs this transformation by means of logic gates.

For example, a binary –to-gray code converter has four binary input lines B_4, B_3,B_2,B_1 and four gray code output lines G_4,G_3,G_2,G_1. When the input is 0010, for instance, the output should be 0011 and so forth. To design a code converter, we use a code table treating it as a truth table to express each output as a Boolean algebraic function of all the inputs.

In this example, of binary –to-gray code conversion, we can treat the binary to the gray code table as four truth tables to derive expressions for G_4, G3, G2, and G1. Each of these four expressions would, in general, contain all the four input variables B_4, B3,B_2,and B1. Thus,this code converter is actually equivalent to four logic circuits, one for each of the truth tables.

The logic expression derived for the code converter can be simplified using the usual techniques, including _don't cares' if present. Even if the input is an unweighted code, the same cell numbering method which we used earlier can be used, but the cell numbers --must correspond to the input combinations as if they were an 8-4-2-1 weighted code. s

Design of a 4-bit binary to gray code converter:

$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15)$ $\quad G_4 = B_4$

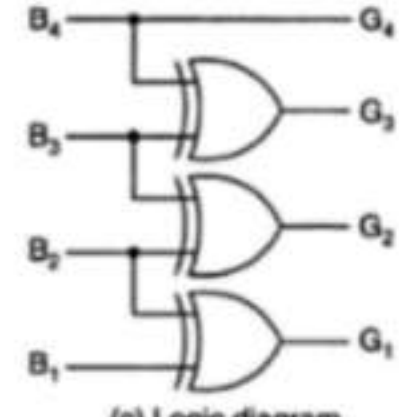
$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11)$ $\quad G_3 = \bar{B}_4B_3 + B_4\bar{B}_3 = B_4 \oplus B_3$

$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13)$ $\quad G_2 = \bar{B}_3B_2 + B_3\bar{B}_2 = B_3 \oplus B_2$

$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14)$ $\quad G_1 = \bar{B}_2B_1 + B_2\bar{B}_1 = B_2 \oplus B_1$

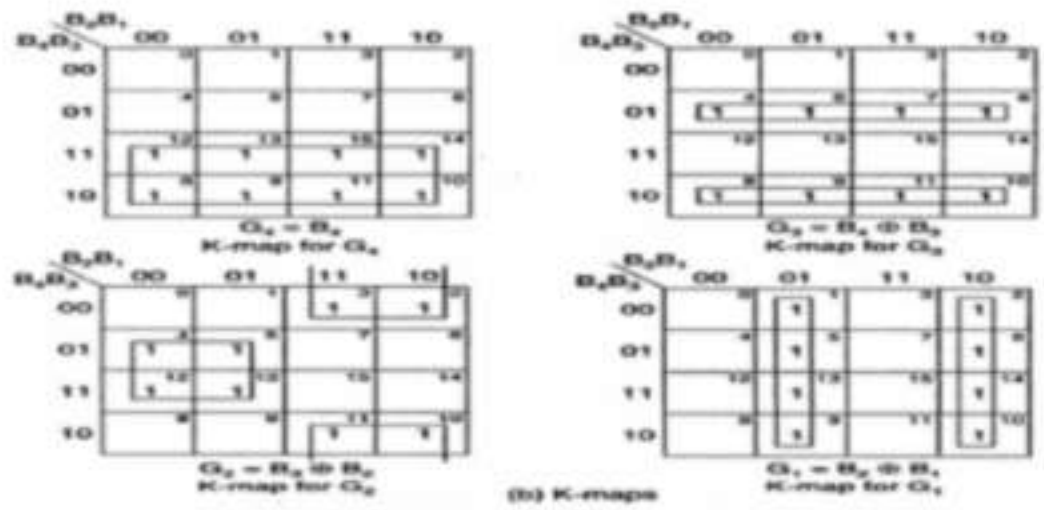
4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

(a) Conversion table



(c) Logic diagram

4-bit binary-to-Gray code converter



(b) K-maps

4-bit binary-to-Gray code converter.

Design of a 4-bit gray to Binary code converter:

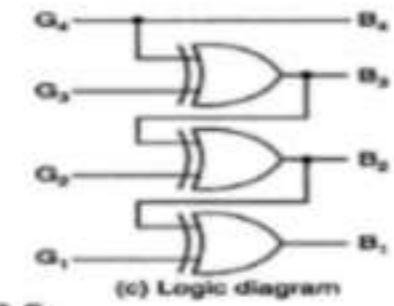
$$B_4 = \Sigma\, m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma\, m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$B_3 = \Sigma\, m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma\, m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$B_2 = \Sigma\, m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma\, m(2, 3, 4, 5, 8, 9, 14, 15)$$
$$B_1 = \Sigma\, m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma\, m(1, 2, 4, 7, 8, 11, 13, 14)$$

$$B_4 = G_4$$
$$B_3 = \bar{G}_4 G_3 + G_4 \bar{G}_3 = G_4 \oplus G_3$$
$$B_2 = \bar{G}_4 G_3 \bar{G}_2 + \bar{G}_4 \bar{G}_3 G_2 + G_4 \bar{G}_3 \bar{G}_2 + G_4 G_3 G_2$$
$$= \bar{G}_4 (G_3 \oplus G_2) + G_4 (\overline{G_3 \oplus G_2}) = G_4 \oplus G_3 \oplus G_2 = B_3 \oplus G_2$$
$$B_1 = \bar{G}_4 \bar{G}_3 \bar{G}_2 G_1 + \bar{G}_4 \bar{G}_3 G_2 \bar{G}_1 + \bar{G}_4 G_3 G_2 G_1 + \bar{G}_4 G_3 \bar{G}_2 \bar{G}_1 + G_4 G_3 \bar{G}_2 G_1$$
$$+ G_4 G_3 G_2 \bar{G}_1 + G_4 \bar{G}_3 G_2 G_1 + G_4 \bar{G}_3 \bar{G}_2 \bar{G}_1$$

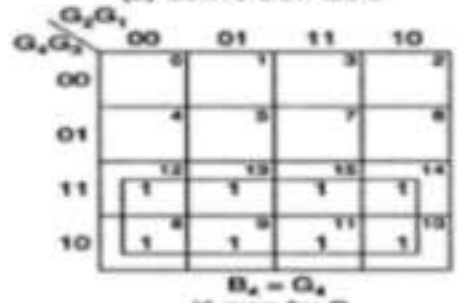
$$= \bar{G}_4 \bar{G}_3 (G_2 \oplus G_1) + G_4 G_3 (G_2 \oplus G_1) + \bar{G}_4 G_3 (\overline{G_2 \oplus G_1}) + G_4 \bar{G}_3 (\overline{G_2 \oplus G_1})$$
$$= (G_2 \oplus G_1)(\overline{G_4 \oplus G_3}) + (\overline{G_2 \oplus G_1})(G_4 \oplus G_3)$$
$$= G_4 \oplus G_3 \oplus G_2 \oplus G_1$$

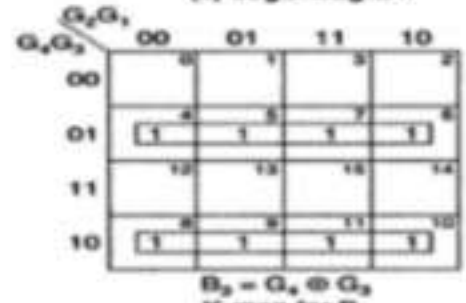
4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

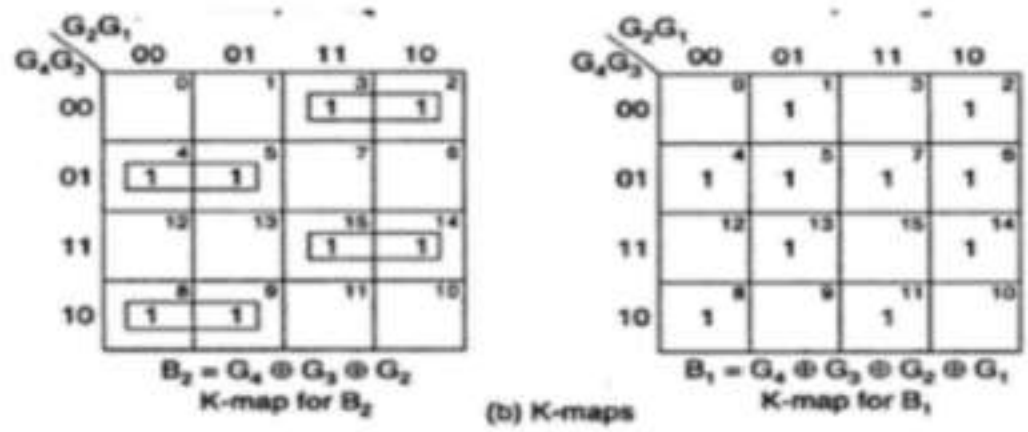
(a) Conversion table



(c) Logic diagram







(b) K-maps

4-bit Gray-to-binary code converter.

Design of a 4-bit BCD to XS-3 code converter:

8421 code				XS-3 code			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

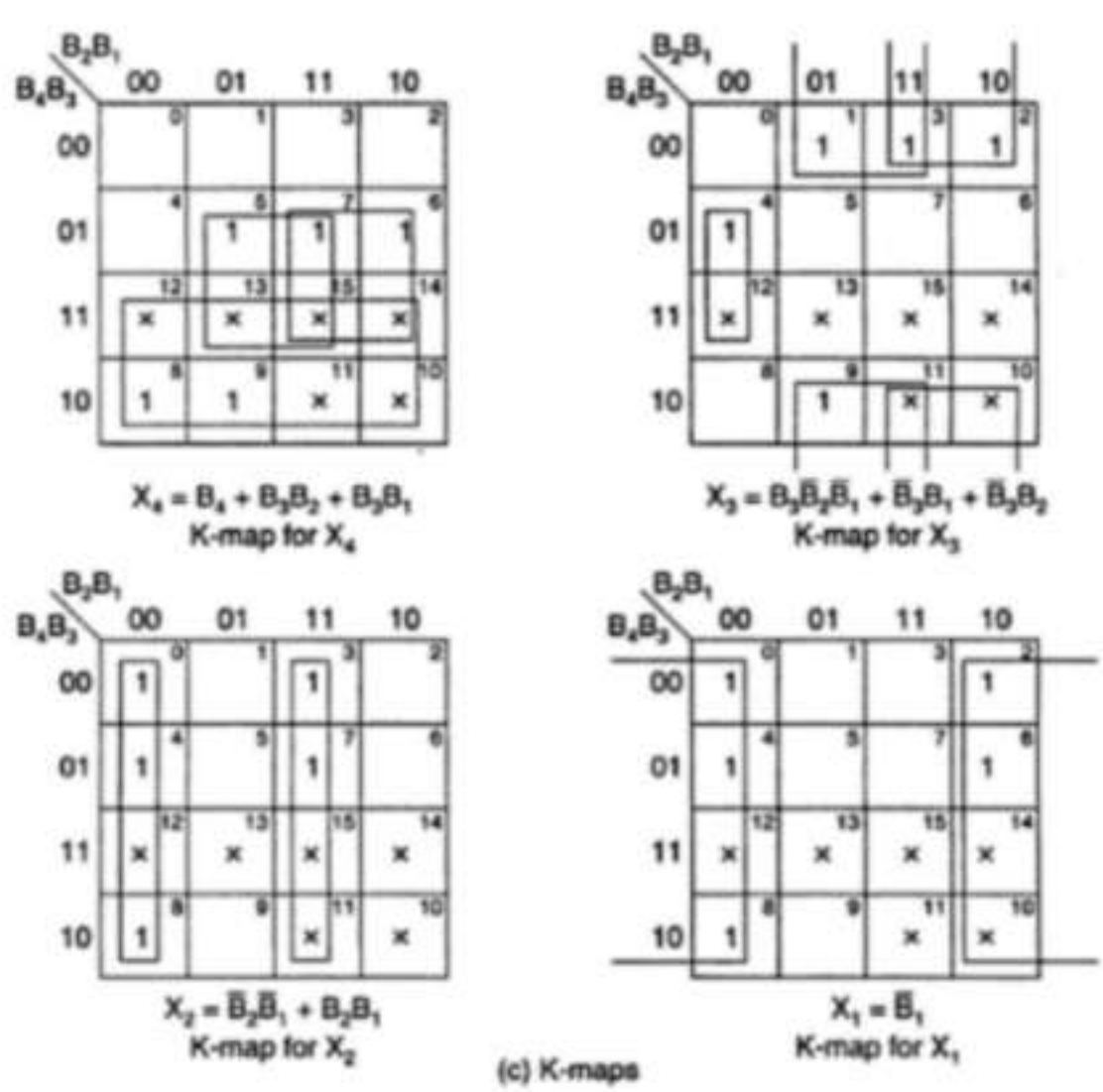
(a) Conversion table

$X_4 = \Sigma\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$
$X_3 = \Sigma\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$
$X_2 = \Sigma\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$
$X_1 = \Sigma\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$

The minimal expressions are
$X_4 = B_4 + B_3B_2 + B_3B_1$
$X_3 = B_3\bar{B}_2\bar{B}_1 + \bar{B}_3B_1 + \bar{B}_3B_2$
$X_2 = \bar{B}_2\bar{B}_1 + B_2B_1$
$X_1 = \bar{B}_1$

(b) Minimal expressions

4-bit BCD-to-XS-3 code converter



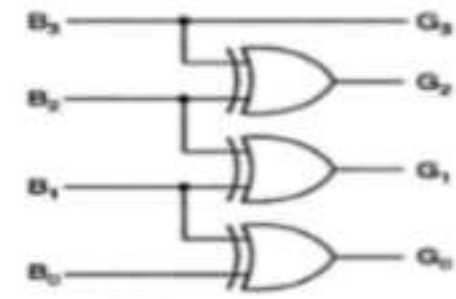
(c) K-maps

4-bit BCD-to-XS-3 code converter.

Design of a BCD to gray code converter:

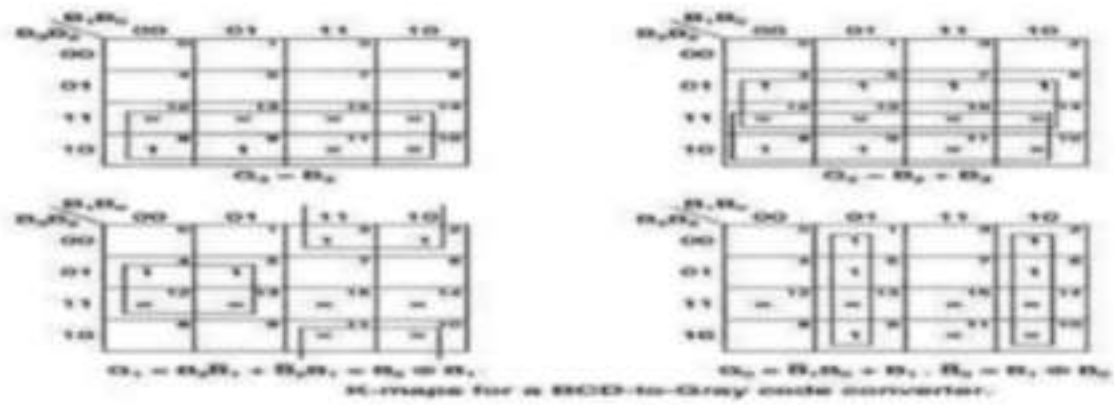
BCD code				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

(a) BCD-to-Gray code conversion table



(b) Logic diagram

BCD-to-Gray code converter.

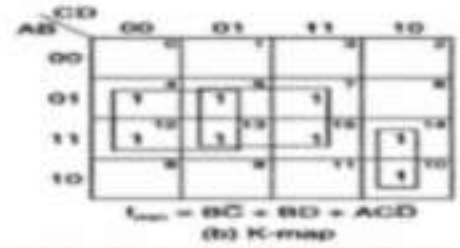


K-maps for a BCD-to-Gray code converter.

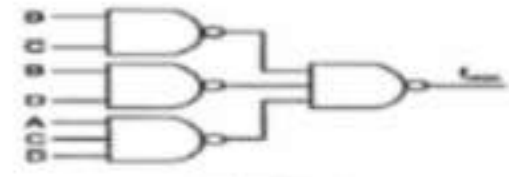
Design of a SOP circuit to Detect the Decimal numbers 5 through 12 in a 4-bit gray code Input:

Decimal number	4-bit Gray code A	B	C	D	Output f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

(a) Truth table



(b) K-map



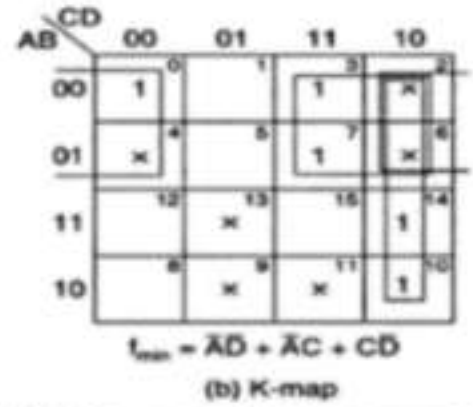
(c) NAND logic

Truth table, K-map and logic diagram for the SOP circuit.

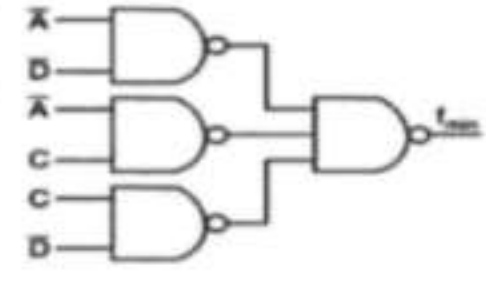
Design of a SOP circuit to detect the decimal numbers 0,2,4,6,8 in a 4-bit 5211 BCDcode input:

Decimal number	5211 code A	B	C	D	Output f
0	0	0	0	0	1
1	0	0	0	1	0
2	0	0	1	1	1
3	0	1	0	1	0
4	0	1	1	1	1
5	1	0	0	0	0
6	1	0	1	0	1
7	1	1	0	0	0
8	1	1	1	0	1
9	1	1	1	1	0

(a) Truth table



(b) K-map



(c) Logic diagram

Truth table, K-map and logic diagram for the SOP circuit.

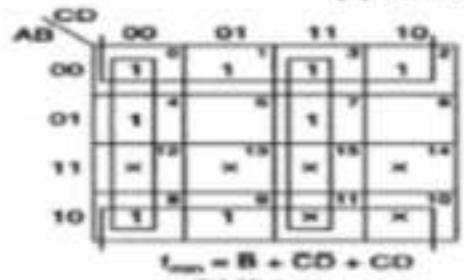
Design of a Combinational circuit to produce the 2's complement of a 4-bit binary number:

Input				Output			
A	B	C	D	E	F	G	H
0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1
0	0	1	0	1	1	1	0
0	0	1	1	1	1	0	1
0	1	0	0	1	1	0	0
0	1	0	1	1	0	1	1
0	1	1	0	1	0	1	0
0	1	1	1	1	0	0	1
1	0	0	0	1	0	0	0
1	0	0	1	0	1	1	1
1	0	1	0	0	1	1	0
1	0	1	1	0	1	0	1
1	1	0	0	0	1	0	0
1	1	0	1	0	0	1	1
1	1	1	0	0	0	1	0
1	1	1	1	0	0	0	1

(a) Conversion table

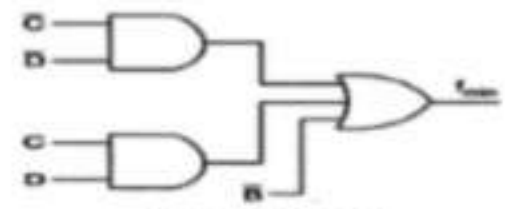
Conversion table and K-maps for the circuit

(a) Seven-segment display



$f_{min} = \bar{B} + C\bar{D} + \bar{C}D$

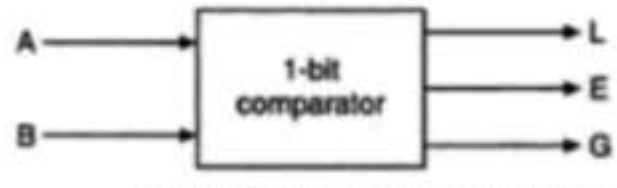
(b) K-map



(c) Logic diagram

Comparators:

$$\text{EQUALITY} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$



Block diagram of a 1-bit comparator.

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be $A = A_0$ and $B = B_0$.
If $A_0 = 1$ and $B_0 = 0$, then $A > B$.
Therefore,

$$A > B: G = A_0\bar{B}_0$$

If $A_0 = 0$ and $B_0 = 1$, then $A < B$.
Therefore,

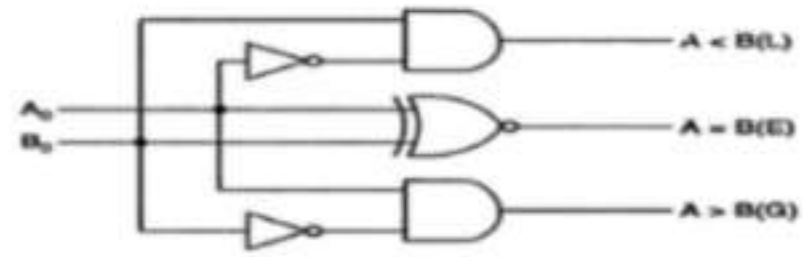
$$A < B: L = \bar{A}_0 B_0$$

If A_0 and B_0 coincide, i.e. $A_0 = B_0 = 0$ or if $A_0 = B_0 = 1$, then $A = B$.
Therefore,

$$A = B : E = A_0 \odot B_0$$

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

1-bit comparator.

1. Magnitude Comparator:

1- bit Magnitude Comparator:

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$. So the logic expression for $A > B$ is

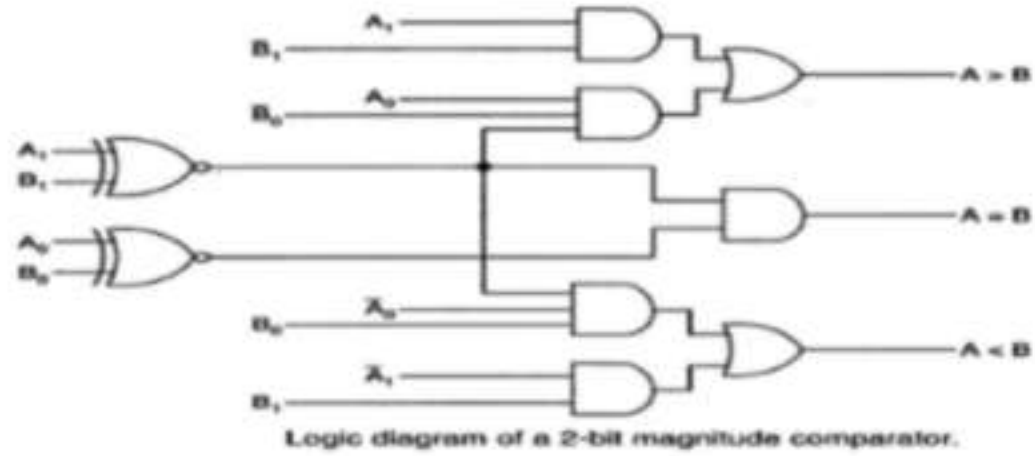
$$A > B : G = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$$

1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$. So the expression for $A < B$ is

$$A < B : L = \bar{A}_1 B_1 + (A_1 \odot B_1)\bar{A}_0 B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$. So the expression for $A = B$ is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$



Logic diagram of a 2-bit magnitude comparator.

4- Bit MagnitudeComparator:

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

1. If $A_3 = 1$ and $B_3 = 0$, then $A > B$. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then $A > B$. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then $A > B$. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then $A > B$.

From these statements, we see that the logic expression for $A > B$ can be written as

$$(A > B) = A_3\bar{B}_3 + (A_3 \odot B_3)A_2\bar{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\bar{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\bar{B}_0$$

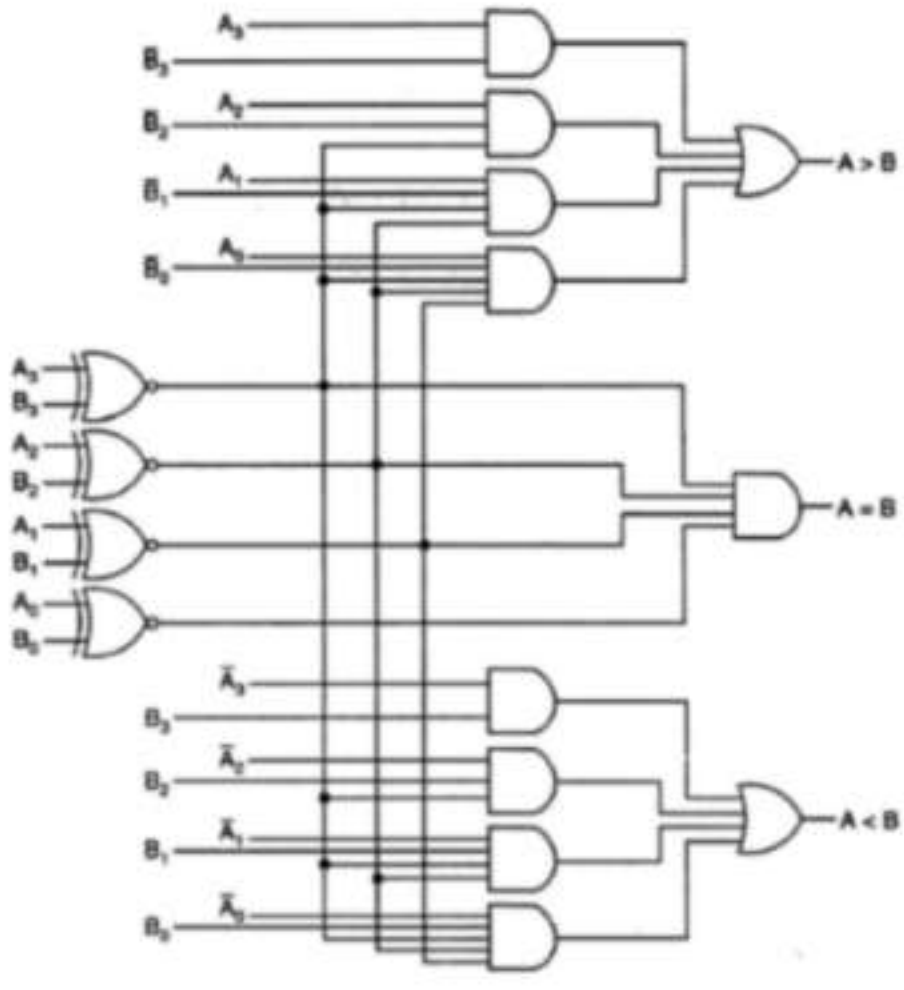
Similarly, the logic expression for $A < B$ can be written as

$$A < B = \bar{A}_3B_3 + (A_3 \odot B_3)\bar{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\bar{A}_1B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\bar{A}_0B_0$$

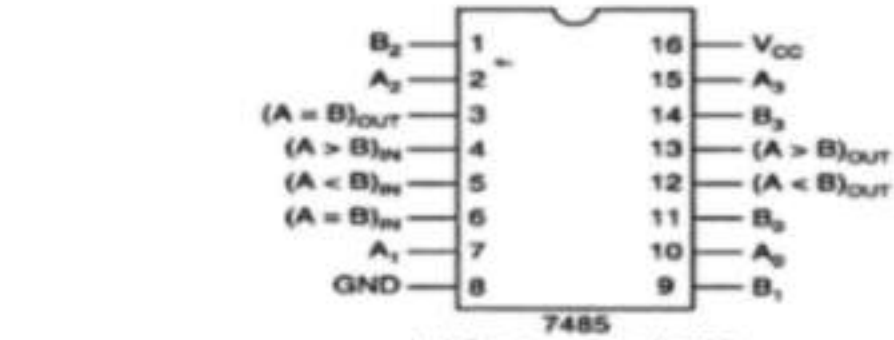
If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_1 and B_1 coincide and if A_0 and B_0 coincide, then $A = B$.

So the expression for $A = B$ can be written as

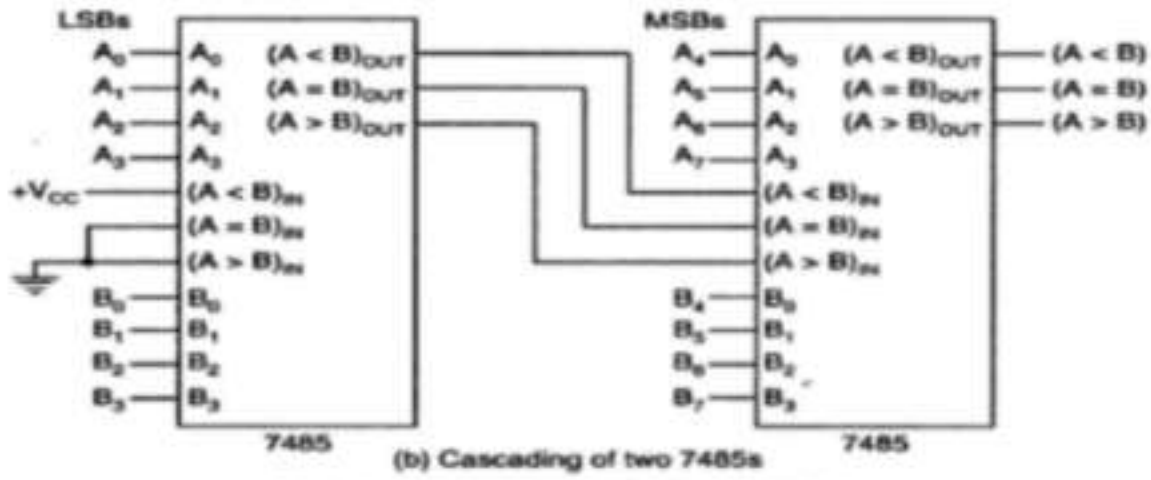
$$(A = B) = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$



IC Comparator:



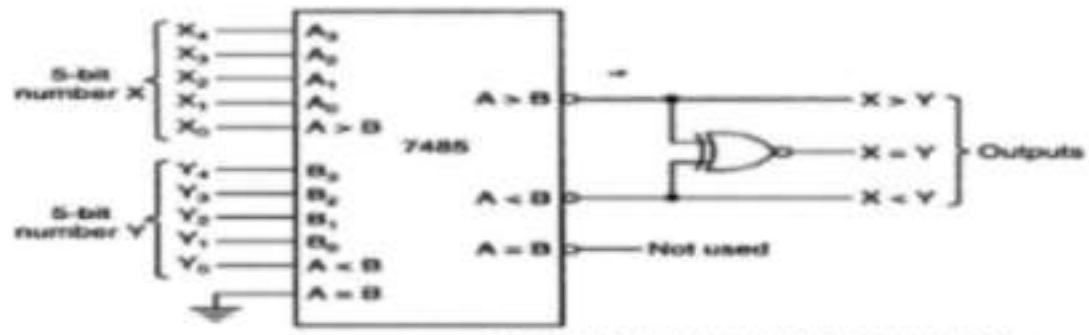
(a) Pin diagram of 7485



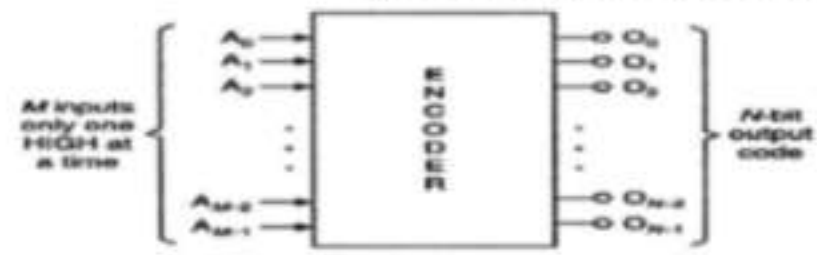
(b) Cascading of two 7485s

Pin diagram and cascading of 7485 4-bit comparators.

ENCODERS:



Use of 7485 as a 5-bit comparator.

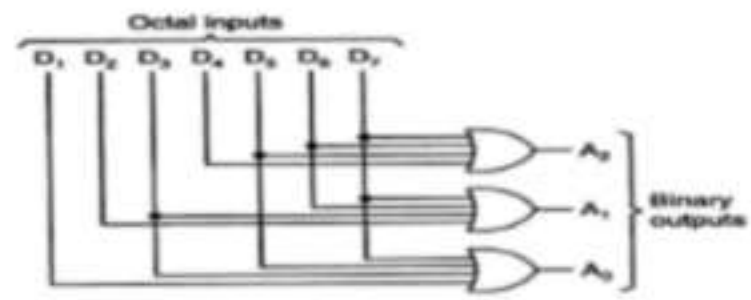


Block diagram of encoder.

Octal to Binary Encoder:

Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

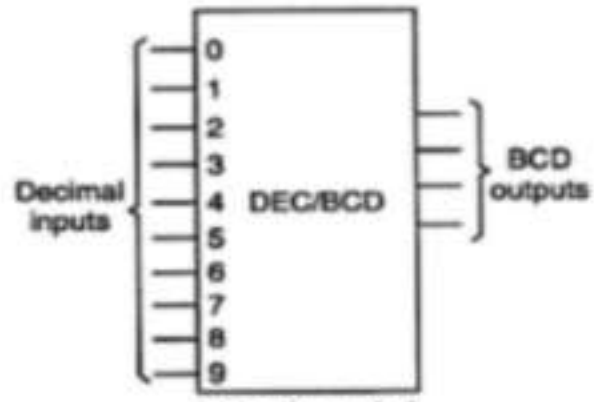
(a) Truth table



(b) Logic diagram

Octal-to-binary encoder.

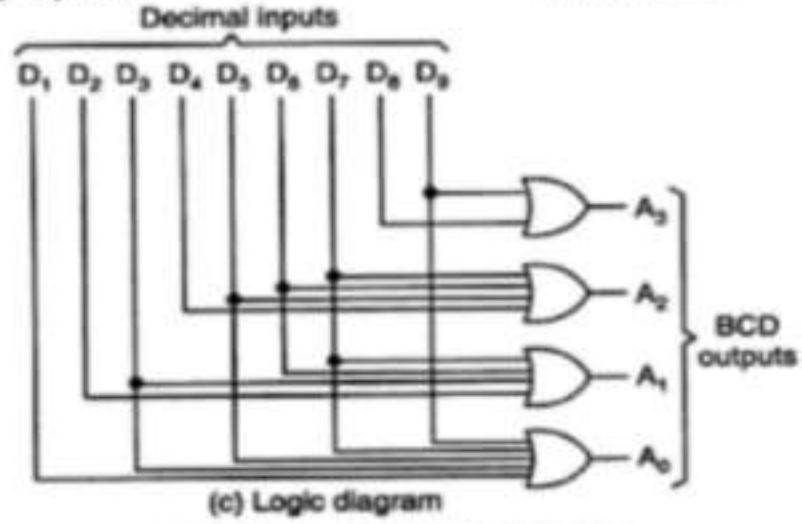
Decimal to BCD Encoder:



(a) Logic symbol

Decimal inputs		Binary			
		A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

(b) Truth table



(c) Logic diagram

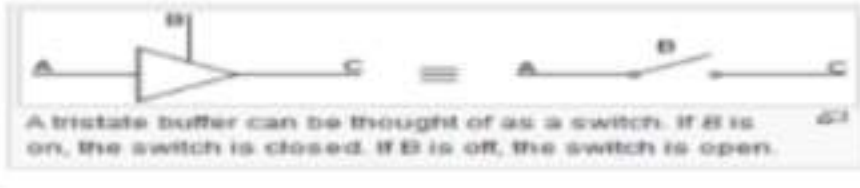
Decimal-to-BCD encoder.

Tristate bus system:

In digital electronics**three-state**, **tri-state**, or **3-state**logic allows an output port to assume a high impedance state in addition to the 0 and 1 logic levels, effectively removing the output from the circuit.

This allows multiple circuits to share the same output line or lines (such as a bus which cannot listen to more than one device at a time).

Three-state outputs are implemented in many registers, bus drivers, and flip-flops in the 7400 and 4000 series as well as in other types, but also internally in many integrated circuits. Other typical uses are internal and external buses in microprocessors, computer memory, and peripherals. Many devices are controlled by an active-low input called OE (Output Enable) which dictates whether the outputs should be held in a high-impedance state or drive their respective loads (to either 0- or 1-level).



A tristate buffer can be thought of as a switch. If B is on, the switch is closed. If B is off, the switch is open.

INPUT		OUTPUT
A	B	C
0	1	0
1		1
X	0	Z (high impedance)

SEQUENTIAL CIRCUITS

The Basic Latch

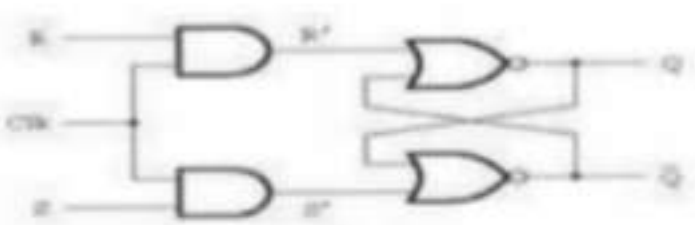
- **Basic latch**is a feedback connection of two NOR gates or two NAND gates
- It can store one bit of information

It can be set to 1 using the *S* input and reset to 0 using the *R* input

The Gated Latch

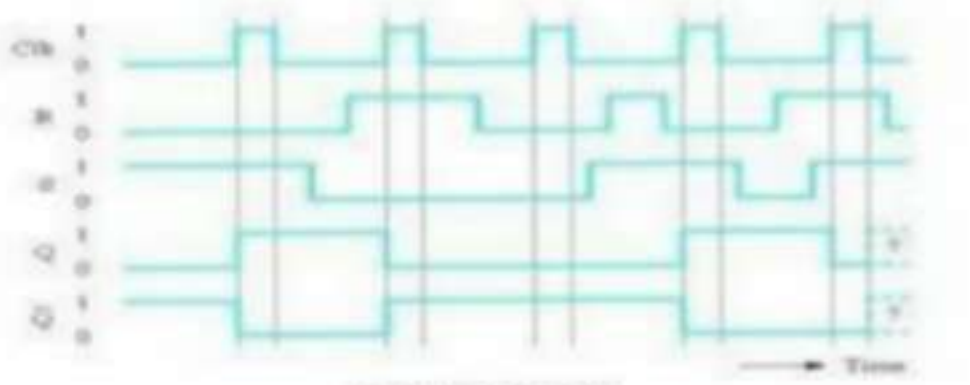
- **Gated latch** is a basic latch that includes input gating and a control signal
- The latch retains its existing state when the control input is equal to0
- Its state may be changed when the control signal is equal to 1. In our discussion we referred to the control input as the clock
- We consider two types of gated latches:
  - **Gated SR latch** uses the *S* and *R* inputs to set the latch to 1 or reset it to 0, respectively.
  - **Gated D latch** uses the *D* input to force the latch into a state that has the samelogic value as the *D* input.

Gated S/R Latch



(a) Circuit

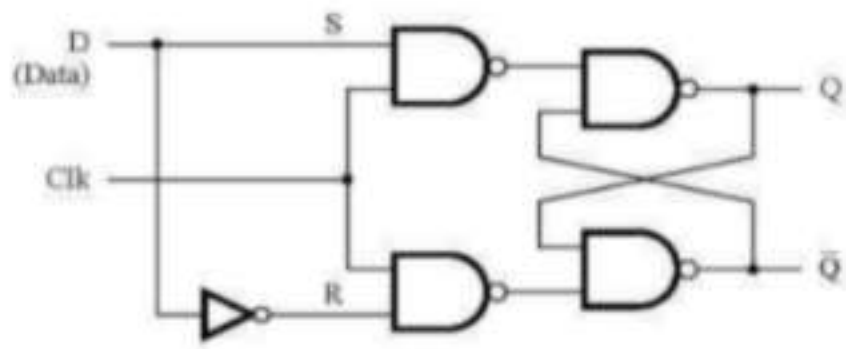
(b) Characteristic table



(c) Timing diagram

(d) Graphical symbol

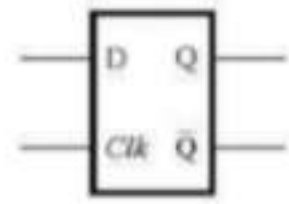
Gated D Latch



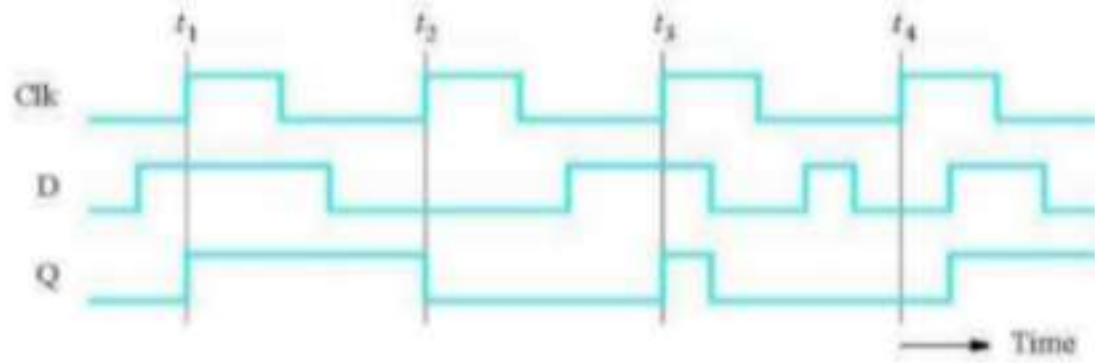
(a) Circuit

Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol



(d) Timing diagram

Setup and Hold Times

- Setup Time t_{su}

  The minimum time that the input signal must be stable prior to the edge of the clock signal.

- Hold Time t_h

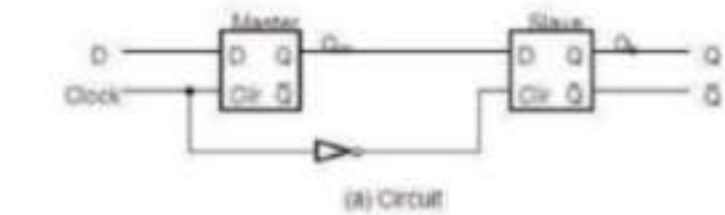
  The minimum time that the input signal must be stable after the edge of the clock signal.

Flip-Flops

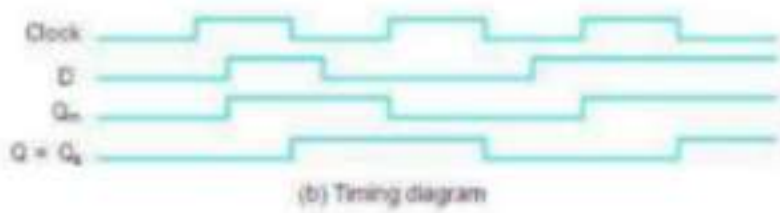
- A **flip-flop** is a storage element based on the gated latch principle
- It can have its output state changed only on the edge of the controlling clock signal
- We consider two types:

- **Edge-triggered flip-flop** is affected only by the input values present when theactive edge of the clock occurs
- **Master-slave flip-flop** is built with two gatedlatches
- The master stage is active during half of the clock cycle, and the slave stage is active during the other half.
- The output value of the flip-flop changes on the edge of the clock that activates the transfer into the slave stage.

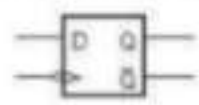
Master-Slave D Flip-Flop



(a) Circuit

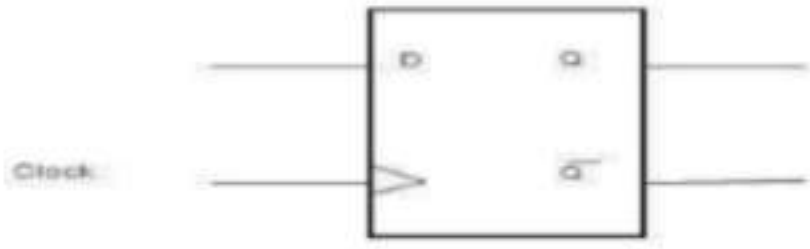


(b) Timing diagram



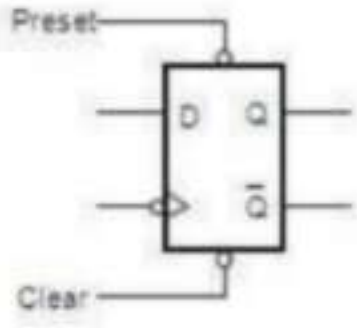
(c) Graphical symbol

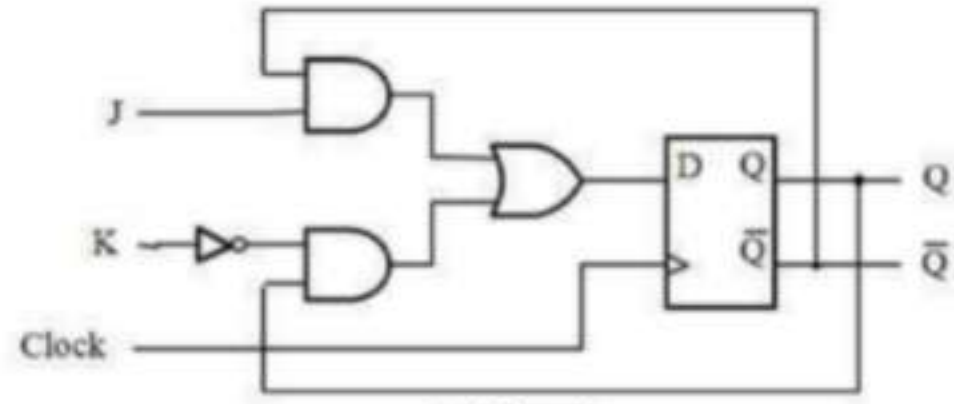
A Positive-Edge-Triggered D Flip-Flop



Graphical symbol

Master-Slave D Flip-Flop with *Clear* and *Preset*

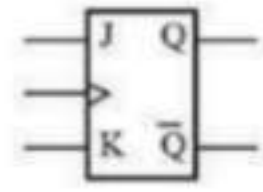




(a) Circuit

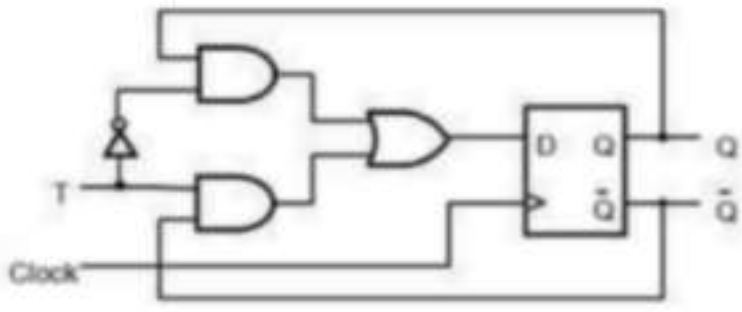
J	K	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	$\overline{Q}(t)$

(b) Characteristic table



(c) Graphical symbol

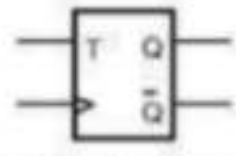
T Flip-Flop



(a) Circuit

T	Q(t + 1)
0	Q(t)
1	$\overline{Q}(t)$

(b) Characteristic table



(c) Graphical symbol

(d) Timing diagram

Excitation Tables

Previous State -> Present State	D
0 -> 0	0
0 -> 1	1
1 -> 0	0
1 -> 1	1

Previous State -> Present State	J	K
0 -> 0	0	X
0 -> 1	1	X
1 -> 0	X	1
1 -> 1	X	0

Previous State -> Present State	S	R
0 -> 0	0	X
0 -> 1	1	0
1 -> 0	0	1
1 -> 1	X	0

Previous State -> Present State	T
0 -> 0	0
0 -> 1	1
1 -> 0	1
1 -> 1	0

Conversions of flip-flops

Example: Use JK-FF to realize D-FF

1) Start transition table for D-FF

2) Create K-maps to express J and K as functions of inputs (D, Q)

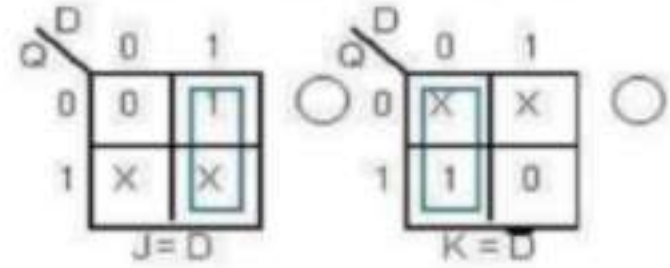
3) Fill in K-maps with appropriate values for J and K
to cause the same state transition as in the D-FF transition table

D	Q	Q^+	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	1	X	0

State-Table

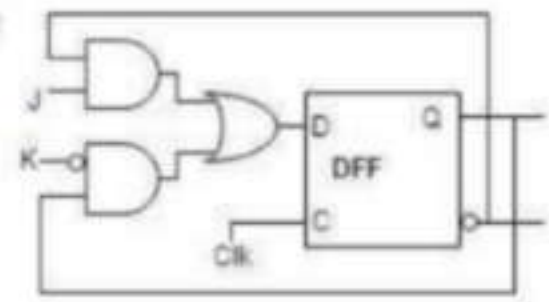
Q	Q^+	R	S	J	K	T	D
0	0	X	0	0	X	0	0
0	1	0	1	1	X	1	1
1	0	1	0	X	1	1	0
1	1	0	X	X	0	0	1

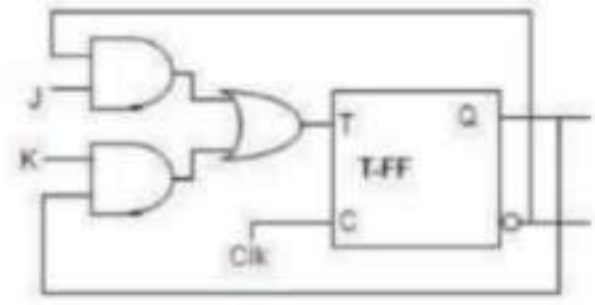
e.g.
when D=Q=0, then $Q^+=0$
the same transition Q-->Q^+
is realize with J=0, K=X

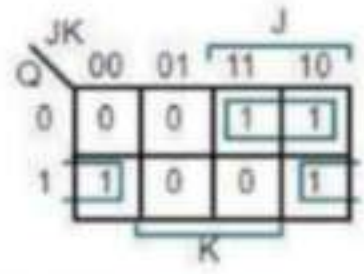


Example: Implement JK-FF using a D-FF

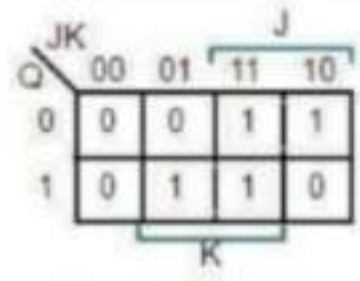
J	K	Q	Q+	D	T
0	0	0	0	0	0
0	0	1	1	1	0
0	1	0	0	0	0
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1







d= jQ + Kq



t= jQ + kq

Sequential Circuit Design

- Steps in the design process for sequential circuits
- State Diagrams and State Tables Examples

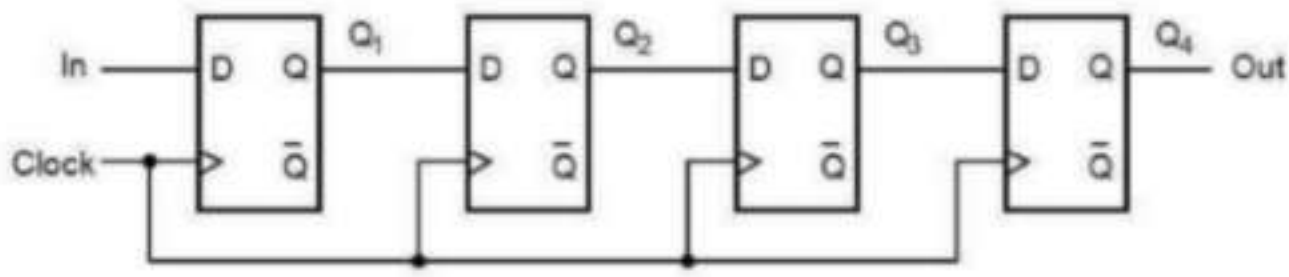
- Steps in Design of a Sequential Circuit
  - 1. Specification – A description of the sequential circuit. Should include a detailing of the inputs, the outputs, and the operation. Possibly assumes that you have knowledge of digital system basics.
  - 2. Formulation: Generate a state diagram and/or a state table from the statement of the problem.
  - 3. State Assignment: From a state table assign binary codes to the states.
  - 4. Flip-flop Input Equation Generation: Select the type of flip-flop for the circuit and generate the needed input for the required state transitions
  - 5. Output Equation Generation: Derive output logic equations for generation of the output from the inputs and current state.
  - 6. Optimization: Optimize the input and output equations. Today, CAD systems are typically used for this in real systems.
  - 7. Technology Mapping: Generate a logic diagram of the circuit using ANDs, ORs, Inverters, and F/Fs.
  - 8. Verification: Use a HDL to verify the design

Registers and Counters

- An n-bit register is a cascade of n flip-flops and can store an n-bit binary data

- A counter can count occurrences of events and can generate timing intervals for control purposes

A Simple Shift Register

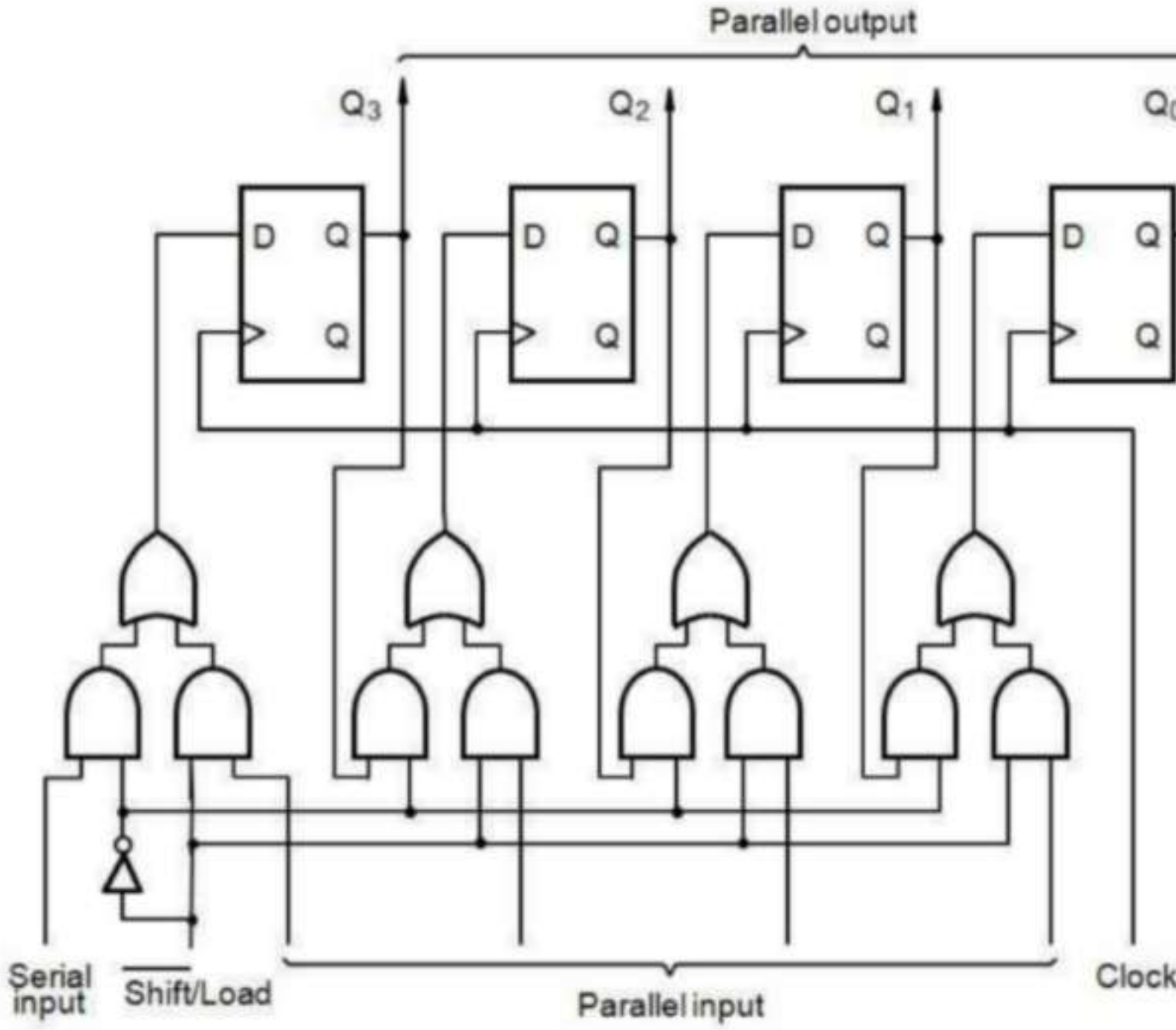


(a) Circuit

	In	Q_1	Q_2	Q_3	Q_4 = Out
t_0	1	0	0	0	0
t_1	0	1	0	0	0
t_2	1	0	1	0	0
t_3	1	1	0	1	0
t_4	1	1	1	0	1
t_5	0	1	1	1	0
t_6	0	0	1	1	1
t_7	0	0	0	1	1

(b) A sample sequence

Parallel-Access Shift Register



Counters

- Counters are a specific type of sequential circuit.
- Like registers, the state, or the flip-flop values themselves, serves as the "output."
- The output value increases by one on each clock cycle.
- After the largest value, the output "wraps around" back to 0.
- Using two bits, we'd get something like this:

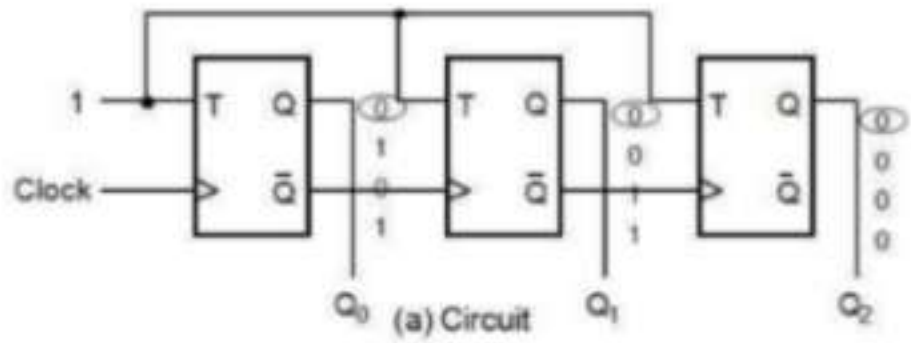
Present State		Next State	
A	B	A	B
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Benefits of counters

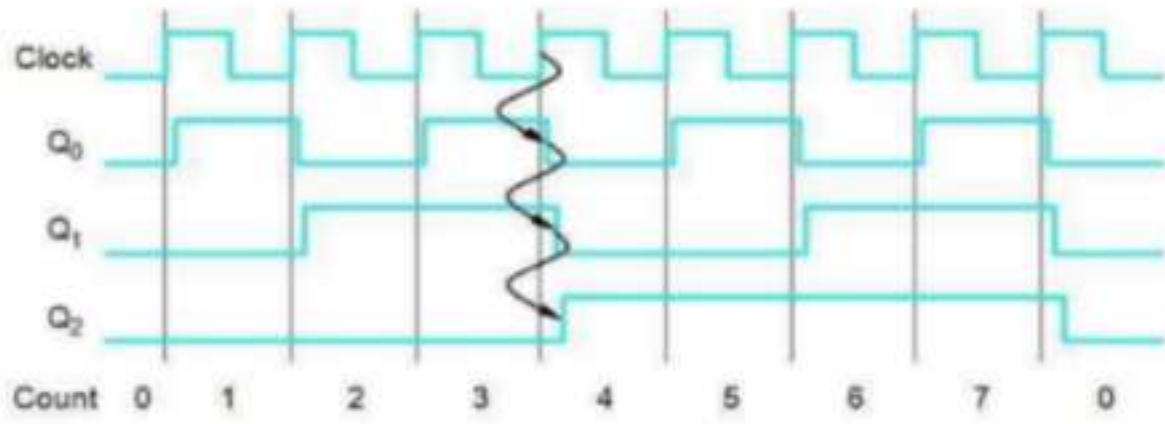
- Counters can act as simple clocks to keep track of "time." • You may need to record how many times something has happened.
  - How many bits have been sent or received?
  - How many steps have been performed in some computation?
- All processors contain a program counter, or PC.
  - Programs consist of a list of instructions that are to be executed one after another (for the most part).
  - The PC keeps track of the instruction currently being executed.
  - The PC increments once on each clock cycle, and the next program instruction is then executed.

A Three-Bit Up-Counter

- Q_1 is connected to clk, Q_2 and Q_3 are clocked by Q' of the preceding stage (hence called asynchronous or ripple counter

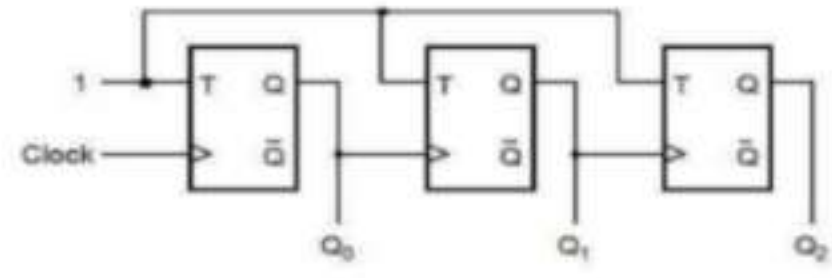


(a) Circuit

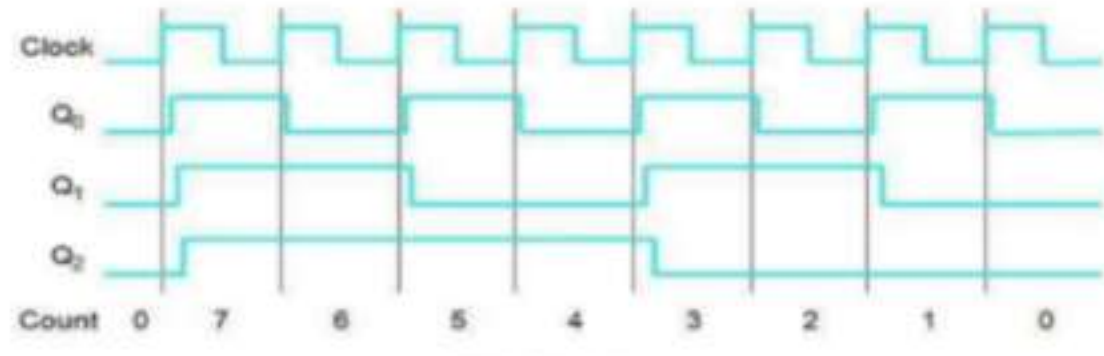


(b) Timing diagram

A Three-Bit Down-Counter



(a) Circuit



(b) Timing diagram

Shift registers:

In digital circuits, a **shift register** is a cascade of flip-flops sharing the same clock, in which the output of each flip-flop is connected to the "data" input of the next flip-flop in the chain, resulting in a circuit that shifts by one position the "bit array" stored in it, *shifting in* the data present at its input and *shifting out* the last bit in the array, at each transition of the clock input. More generally, a **shift register** may be multidimensional, such that its "data in" and stage outputs are themselves bit arrays: this is implemented simply by running several shift registers of the same bit-length in parallel.

Shift registers can have both parallel and serial inputs and outputs. These are often configured as **serial-in, parallel-out** (SIPO) or as **parallel-in, serial-out** (PISO). There are also types that have both serial and parallel input and types with serial and parallel output. There are also **bi-directional** shift registers which allow shifting in both directions: L→R or R→L. The serial input and last output of a shift register can also be connected to create a **circular shift register**

Shift registers are a type of logic circuits closely related to counters. They are basically for the storage and transfer of digital data.

Buffer register:

The buffer register is the simple set of registers. It is simply stores the binary word. The buffer may be controlled buffer. Most of the buffer registers used D Flip-flops.

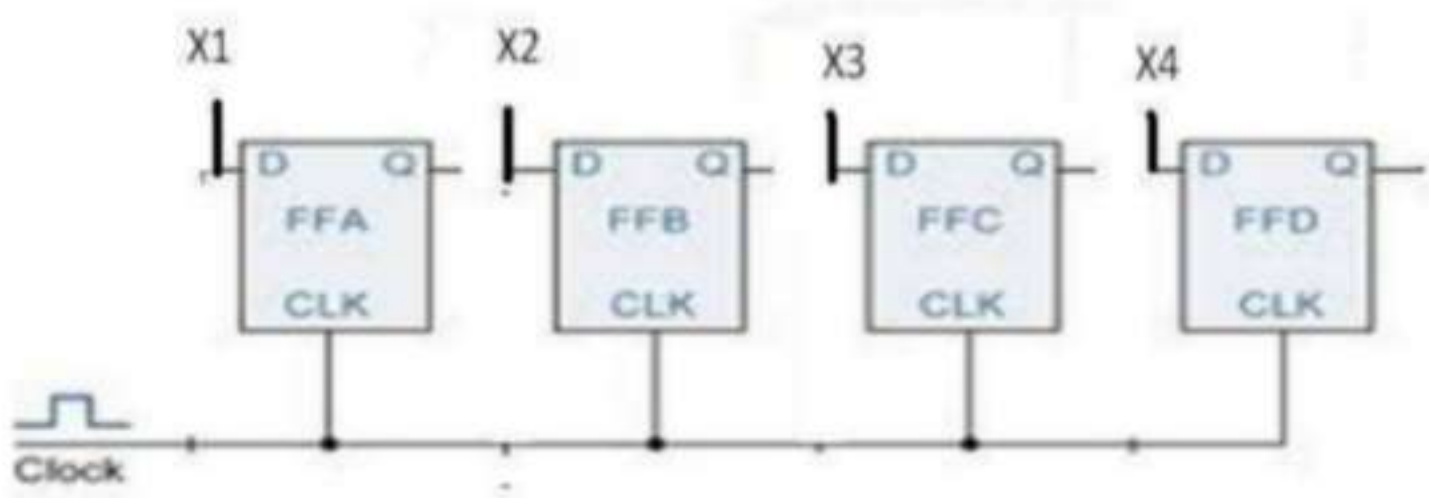


Figure: logic diagram of 4-bit buffer register

The figure shows a 4-bit buffer register. The binary word to be stored is applied to the data terminals. On the application of clock pulse, the output word becomes the same as the word applied at the terminals. i.e., the input word is loaded into the register by the application of clock pulse.

When the positive clock edge arrives, the stored word becomes:

$$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$$

$$Q=X$$

Controlled buffer register:

If goes LOW, all the FFs are RESET and the output becomes, Q=0000.

When is HIGH, the register is ready for action. LOAD is the control input. When LOAD is HIGH, the data bits X can reach the D inputs of FF's.

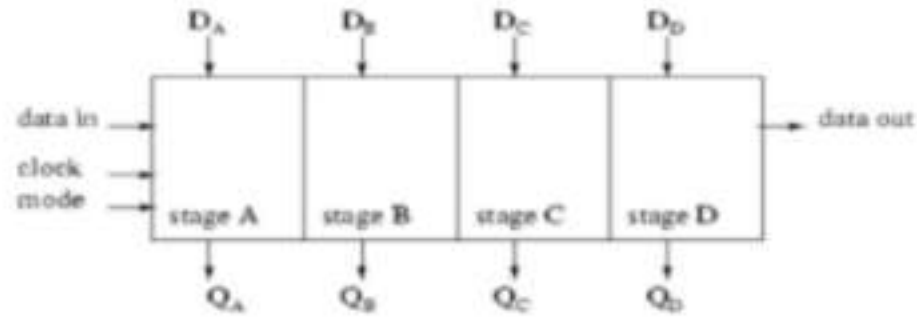
$$Q_4Q_3Q_2Q_1=X_4X_3X_2X_1$$

$$Q=X$$

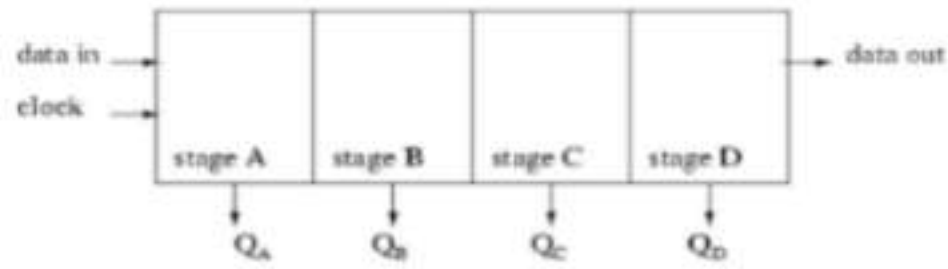
When load is low, the X bits cannot reach the FF's.

Data transmission in shift registers:

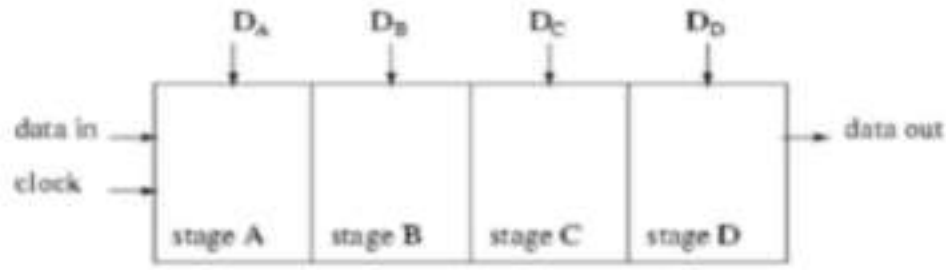
Serial-in, serial-out shift register with 4-stages



Parallel-in, parallel-out shift register with 4-stages



Serial-in, parallel-out shift register with 4-stages



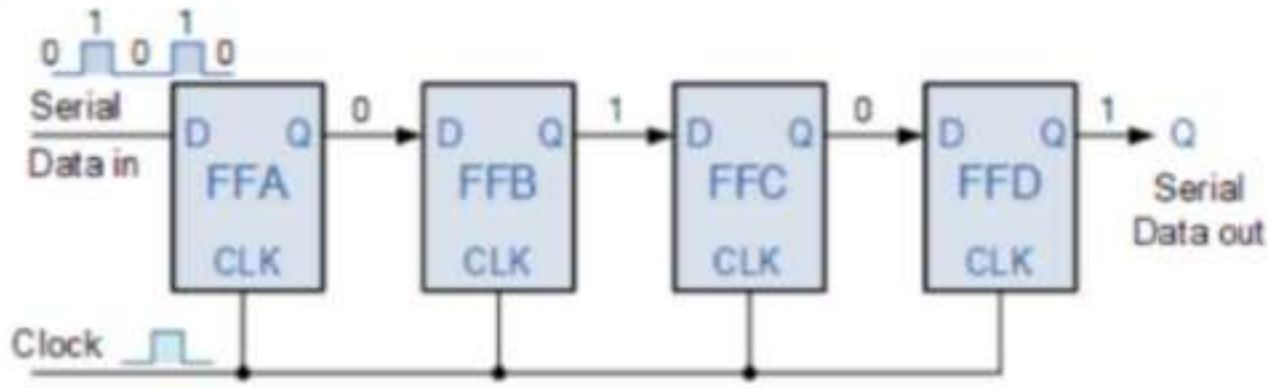
Parallel-in, serial-out shift register with 4-stages

A number of ff's connected together such that data may be shifted into and shifted out of them is called shift register. data may be shifted into or out of the register in serial form or in parallel form. There are four basic types of shift registers.

1. Serial in, serial out, shift right, shift registers
2. Serial in, serial out, shift left, shift registers
3. Parallel in, serial out shift registers
4. Parallel in, parallel out shift registers

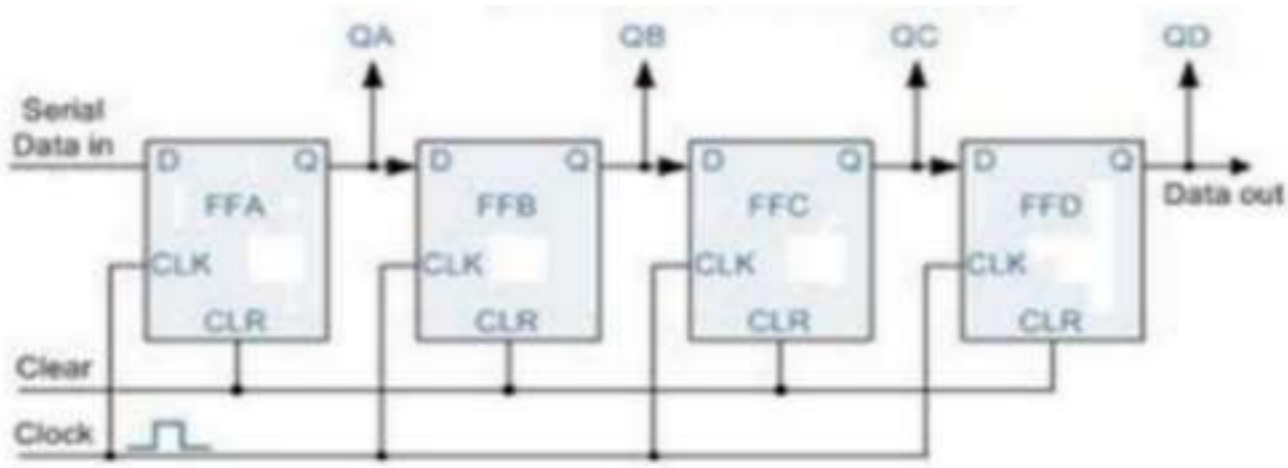
Serial IN, serial OUT, shift right, shift left register:

The logic diagram of 4-bit serial in serial out, right shift register with four stages. The register can store four bits of data. Serial data is applied at the input D of the first FF. the Q output of the first FF is connected to the D input of another FF. the data is outputted from the Q terminal of the last FF.



When serial data is transferred into a register, each new bit is clocked into the first FF at the positive going edge of each clock pulse. The bit that was previously stored by the first FF is transferred to the second FF. the bit that was stored by the Second FF is transferred to the third FF.

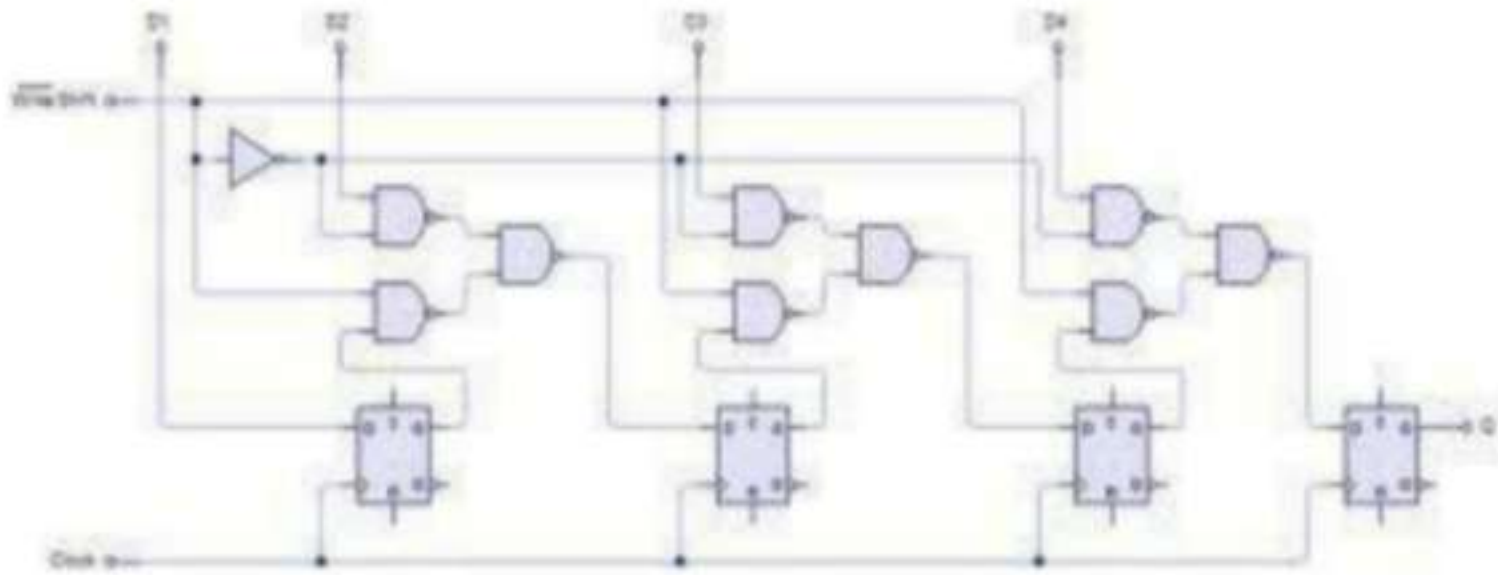
Serial-in, parallel-out, shift register:



In this type of register, the data bits are entered into the register serially, but the data stored in the register is shifted out in parallel form.

Once the data bits are stored, each bit appears on its respective output line and all bits are available simultaneously, rather than on a bit-by-bit basis with the serial output. The serial-in, parallel out, shift register can be used as serial-in, serial out, shift register if the output is taken from the Q terminal of the last FF.

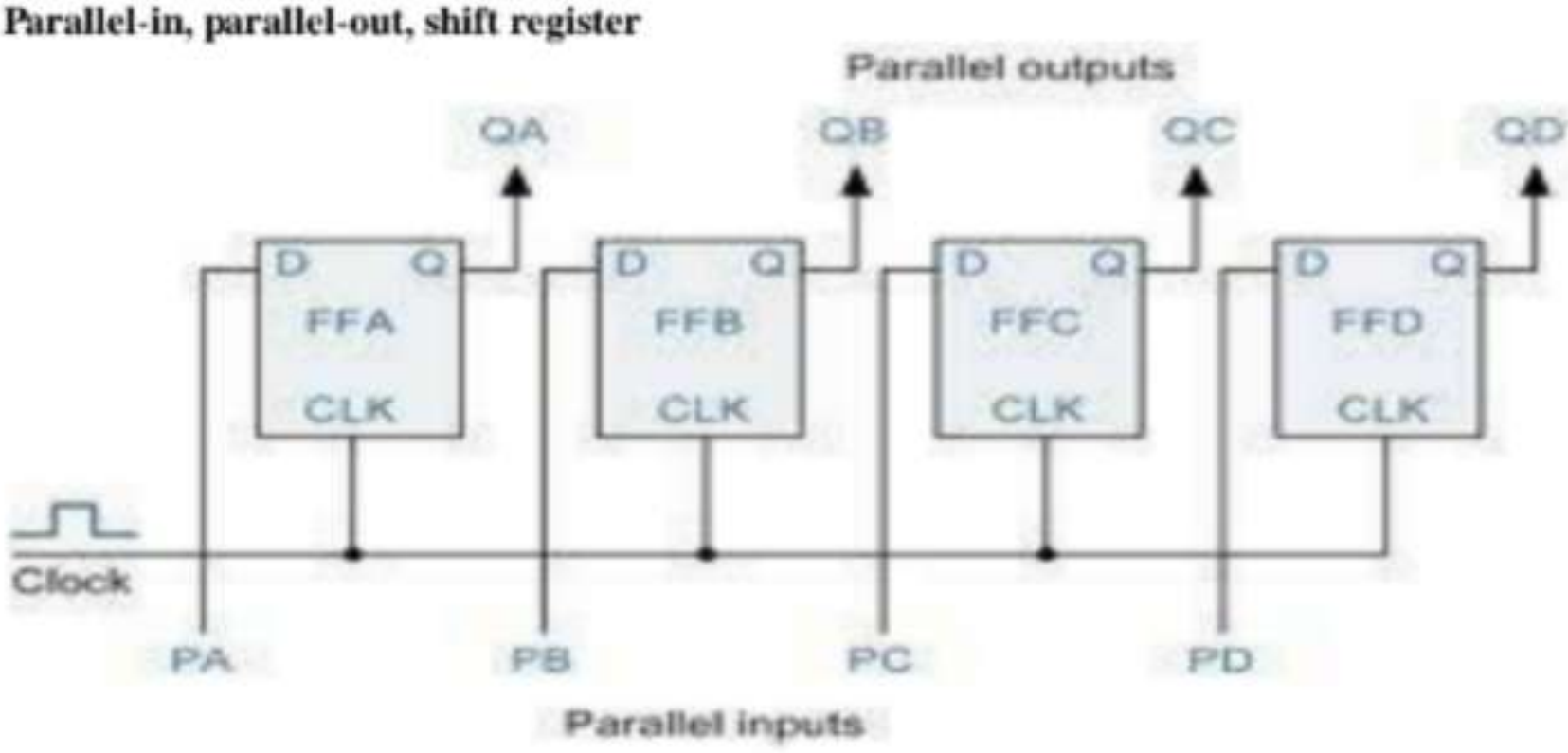
Parallel-in, serial-out, shift register:



For a parallel-in, serial out, shift register, the data bits are entered simultaneously into their respective stages on parallel lines, rather than on a bit-by-bit basis on one line as with serial data bits are transferred out of the register serially. On a bit-by-bit basis over a single line.

There are four data lines A,B,C,D through which the data is entered into the register in parallel form. The signal shift/ load allows the data to be entered in parallel form into the register and the data is shifted out serially from terminalQ4

Parallel-in, parallel-out, shift register



In a parallel-in, parallel-out shift register, the data is entered into the register in parallel form, and also the data is taken out of the register in parallel form. Data is applied to the D input terminals of the FF's. When a clock pulse is applied, at the positive going edge of the pulse, the D inputs are shifted into the Q outputs of the FFs. The register now stores the data. The stored data is available instantaneously for shifting out in parallel form.

Bidirectional shift register:

A bidirectional shift register is one which the data bits can be shifted from left to right or from right to left. A fig shows the logic diagram of a 4-bit serial-in, serial out, bidirectional shift register. Right/left is the mode signal, when right /left is a 1, the logic circuit works as a shift-register.the bidirectional operation is achieved by using the mode signal and two NAND gates and one OR gate for each stage.

A HIGH on the right/left control input enables the AND gates G1, G2, G3 and G4 and disables the AND gates G5,G6,G7 and G8, and the state of Q output of each FF is passed through the gate to the D input of the following FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the right. A LOW on the right/left control inputs enables the AND gates G5, G6, G7 and G8 and disables the And gates G1, G2, G3 and G4 and the Q output of each FF is passed to the D input of the preceding FF. when a clock pulse occurs, the data bits are then effectively shifted one place to the left. Hence, the circuit works as a bidirectional shift register

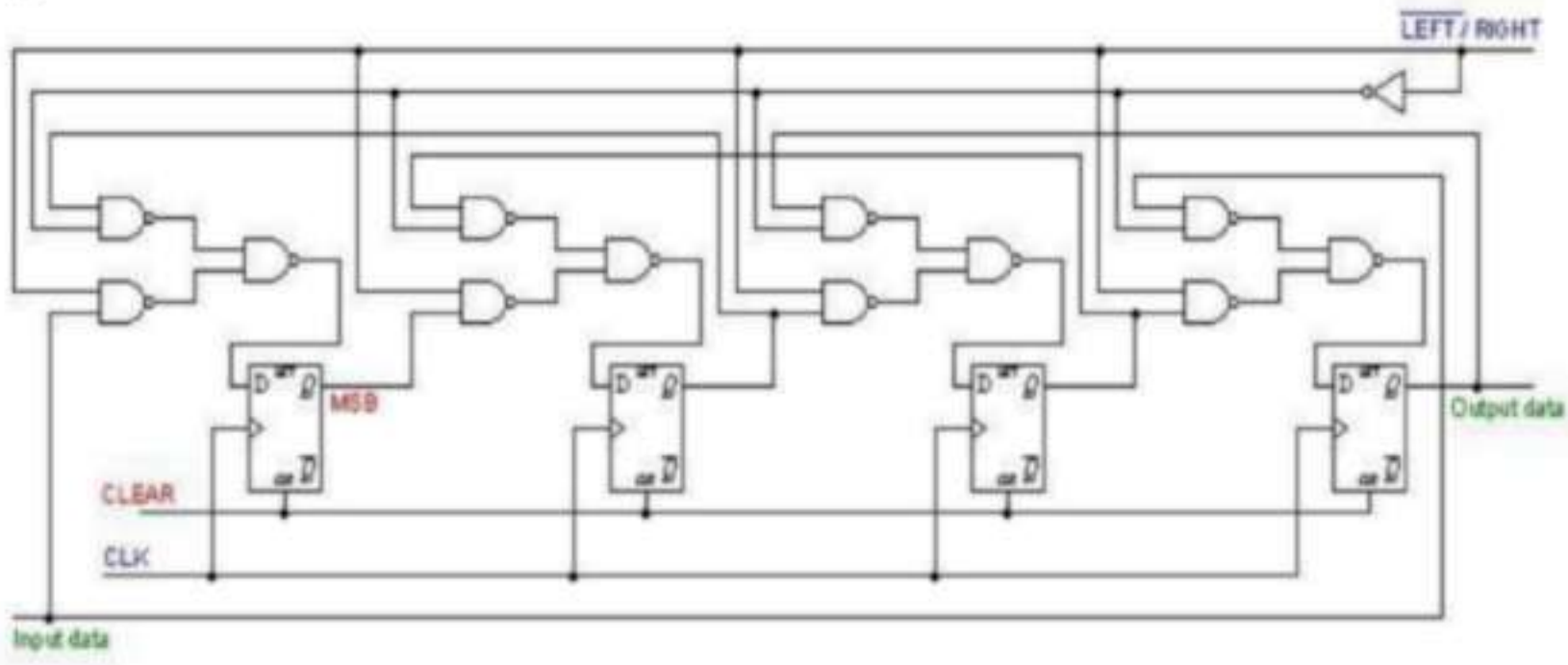


Figure: logic diagram of a 4-bit bidirectional shift register

Universal shift register:

A register is capable of shifting in one direction only is a unidirectional shift register. One that can shift both directions is a bidirectional shift register. If the register has both shifts and parallel load capabilities, it is referred to as a universal shift registers. Universal shift register is a bidirectional register, whose input can be either in serial form or in parallel form and whose output also can be in serial form or I parallel form.

The most general shift register has the following capabilities.

1. A clear control to clear the register to 0
2. A clock input to synchronize the operations
3. A shift-right control to enable the shift-right operation and serial input and output lines associated with the shift-right

4. A shift-left control to enable the shift-left operation and serial input and output lines associated with the shift-left
5. A parallel loads control to enable a parallel transfer and the n input lines associated with the parallel transfer
6. N parallel output lines
7. A control state that leaves the information in the register unchanged in the presence of the clock.

A universal shift register can be realized using multiplexers. The below fig shows the logic diagram of a 4-bit universal shift register that has all capabilities. It consists of 4 D flip-flops and four multiplexers. The four multiplexers have two common selection inputs s1 and s0. Input 0 in each multiplexer is selected when S1S0=00, input 1 is selected when S1S0=01 and input 2 is selected when S1S0=10 and input 4 is selected when S1S0=11. The selection inputs control the mode of operation of the register according to the functions entries. When S1S0=0, the present value of the register is applied to the D inputs of flip-flops. The condition forms a path from the output of each flip-flop into the input of the same flip-flop. The next clock edge transfers into each flip-flop the binary value it held previously, and no change of state occurs. When S1S0=01, terminal 1 of the multiplexer inputs have a path to the D inputs of the flip-flop. This causes a shift-right operation, with serial input transferred into flip-flopA4. When S1S0=10, a shift left operation results with the other serial input going into flip-flop A1. Finally when S1S0=11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock cycle

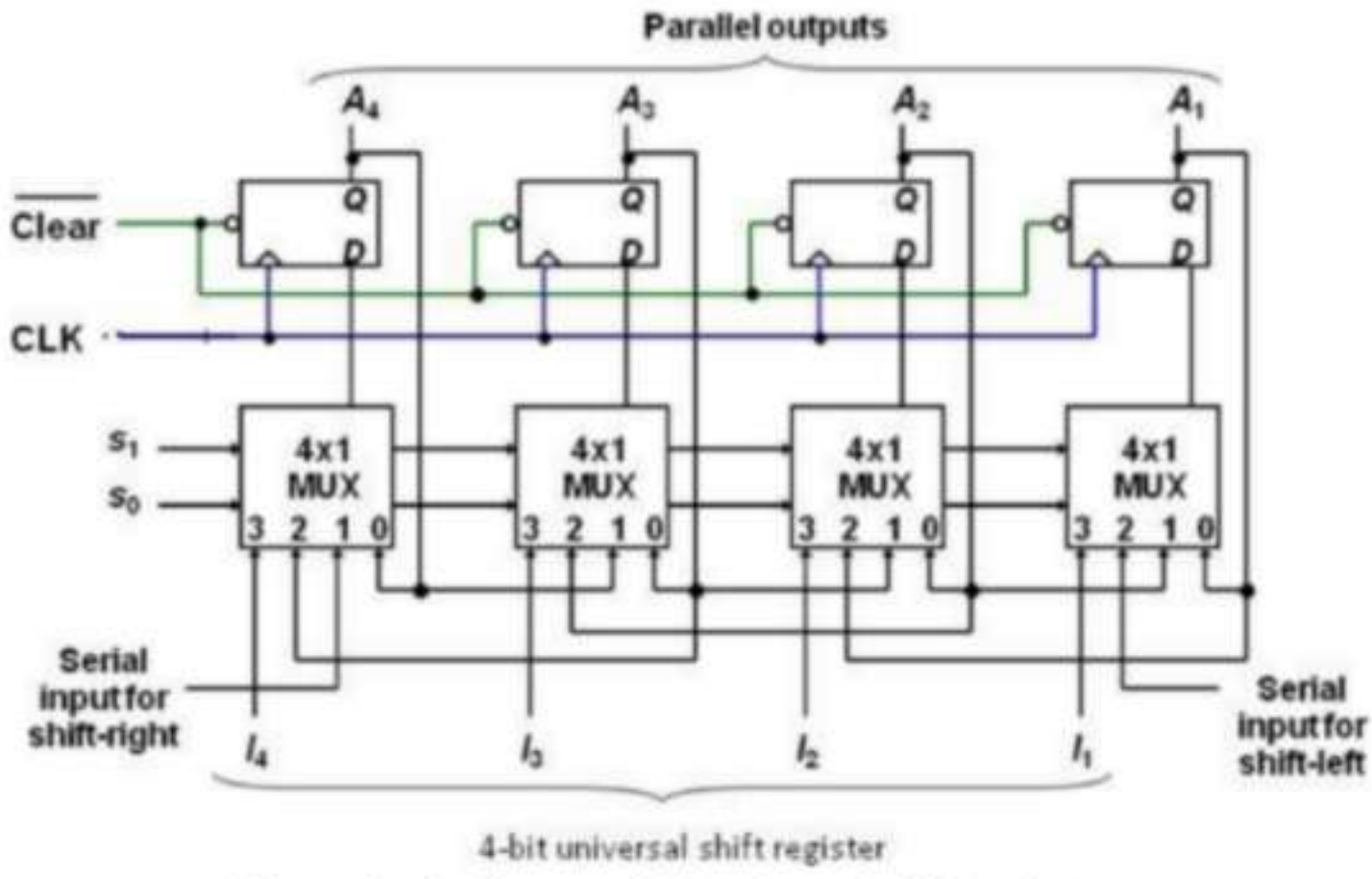


Figure: logic diagram 4-bit universal shift register

Function table for theregister

mode control		
S0	**S1**	**register operation**
0	0	No change
0	1	Shift Right
1	0	Shift left
1	1	Parallel load

Counters:

Counter is a device which stores (and sometimes displays) the number of times particular event or process has occurred, often in relationship to a clock signal. A Digital counter is a set of flip flops whose state change in response to pulses applied at the input to the counter. Counters may be asynchronous counters or synchronous counters. Asynchronous counters are also called ripple counters

In electronics counters can be implemented quite easily using register-type circuits such as the flip-flops and a wide variety of classifications exist:

- Asynchronous (ripple) counter – changing state bits are used as clocks to subsequent state flip-flops
- Synchronous counter – all state bits change under control of a singleclock
- Decade counter – counts through ten states per stage
- Up/down counter – counts both up and down, under command of a control input
- Ring counter – formed by a shift register with feedback connection in a ring
- Johnson counter – a *twisted* ring counter
- Cascaded counter
- Modulus counter.

Each is useful for different applications. Usually, counter circuits are digital in nature, and count in natural binary Many types of counter circuits are available as digital building blocks, for example a number of chips in the 4000 series implement different counters.

Occasionally there are advantages to using a counting sequence other than the natural binary sequence such as the binary coded decimal counter, a linear feed-back shift register counter, or a gray-code counter.

Counters are useful for digital clocks and timers, and in oven timers, VCR clocks, etc.

Asynchronous counters:

An asynchronous (ripple) counter is a single JK-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0). This counter will increment once for every clock cycle and takes two clock cycles to overflow, so every cycle it will alternate between a transition from 0 to 1 and a transition from 1 to 0. Notice that this creates a new clock with a 50% duty cycle at exactly half the frequency of the input clock. If this output is then used as the clock signal for a similarly arranged D flip-flop (remembering to invert the output to the input), one will get another 1 bit counter that counts half as fast. Putting them together yields a two-bit counter:

Two-bit ripple up-counter using negative edge triggered flip flop:

Two bit ripple counter used two flip-flops. There are four possible states from 2 – bit up-counting I.e. 00, 01, 10 and 11.

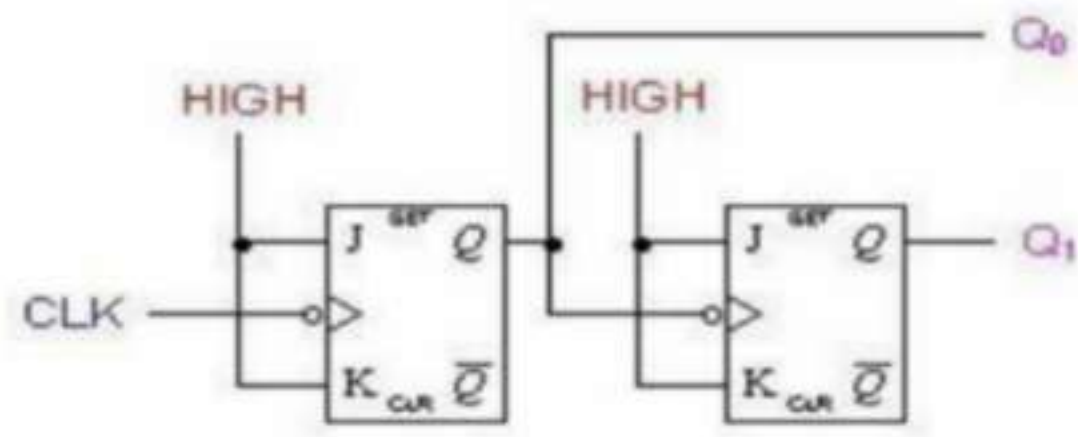
- The counter is initially assumed to be at a state 00 where the outputs of the tow flip-flops are noted as Q_1Q_0. Where Q_1 forms the MSB and Q_0 forms the LSB.

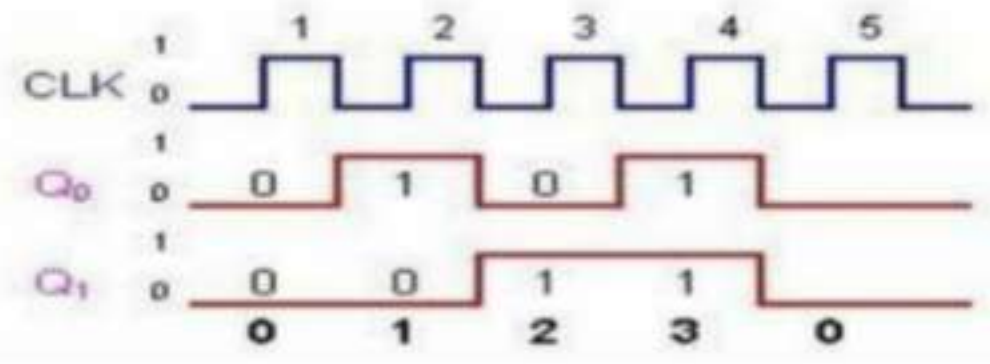
- For the negative edge of the first clock pulse, output of the first flip-flop FF_1 toggles its state. Thus Q_1 remains at 0 and Q_0 toggles to 1 and the counter state are now read as 01.

- During the next negative edge of the input clock pulse FF_1 toggles and $Q_0 = 0$. The output Q0 being a clock signal for the second flip-flop FF_2 and the present transition acts as a negative edge for FF_2 thus toggles its state $Q_1 = 1$. The counter state is now read as 10.

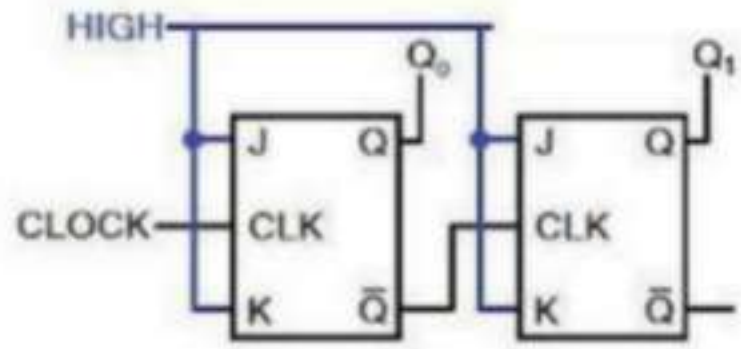
- For the next negative edge of the input clock to FF_1 output Q0 toggles to 1. But this transition from 0 to 1 being a positive edge for FF_2 output Q_1 remains at 1. The counter state is now read as 11.

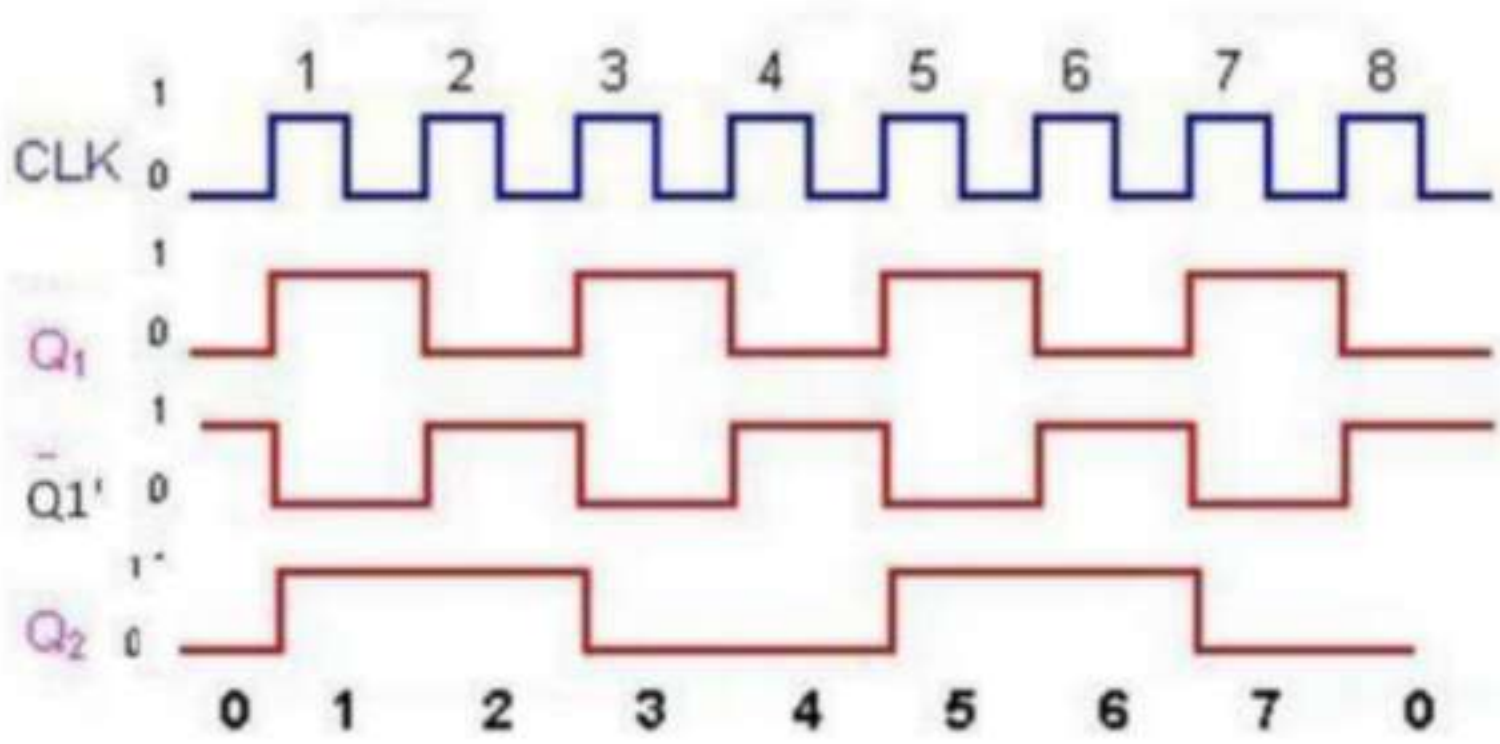
- For the next negative edge of the input clock, Q_0 toggles to 0. This transition from 1 to 0 acts as a negative edge clock for FF_2 and its output Q_1 toggles to 0. Thus the starting state 00 is attained. Figure shown below





Two-bit ripple down-counter using negative edge triggered flip flop:





A 2-bit down-counter counts in the order 0,3,2,1,0,1……,i.e, 00,11,10,01,00,11 …..,etc. the above fig. shows ripple down counter, using negative edge triggered J-K FFs and its timing diagram.

- For down counting, Q1' of FF1 is connected to the clock of Ff2. Let initially all the FF1 toggles, so, Q1 goes from a 0 to a 1 and Q1' goes from a 1 to a 0.

- The negative-going signal at Q1‘ is applied to the clock input of FF2, toggles Ff2 and, therefore, Q2 goes from a 0 to a 1.so, after one clock pulse Q2=1 and Q1=1, I.e., the state of the counter is 11.
- At the negative-going edge of the second clock pulse, Q1 changes from a 1 to a 0 and Q1‘ from a 0 to a 1.
- This positive-going signal at Q1‘ does not affect FF2 and, therefore, Q2 remains at a 1. Hence , the state of the counter after second clock pulse is 10
- At the negative going edge of the third clock pulse, FF1 toggles. So Q1, goes from a 0 to a 1 and Q1‘ from 1 to 0. This negative going signal at Q1‘ toggles FF2 and, so, Q2 changes from 1 to 0, hence, the state of the counter after the third clock pulse is 01.
- At the negative going edge of the fourth clock pulse, FF1 toggles. So Q1, goes from a 1 to a 0 and Q1‘ from 0 to 1. . This positive going signal at Q1‘ does not affect FF2 and, so, Q2 remains at 0, hence, the state of the counter after the fourth clock pulse is 00.

Two-bit ripple up-down counter using negative edge triggered flip flop:

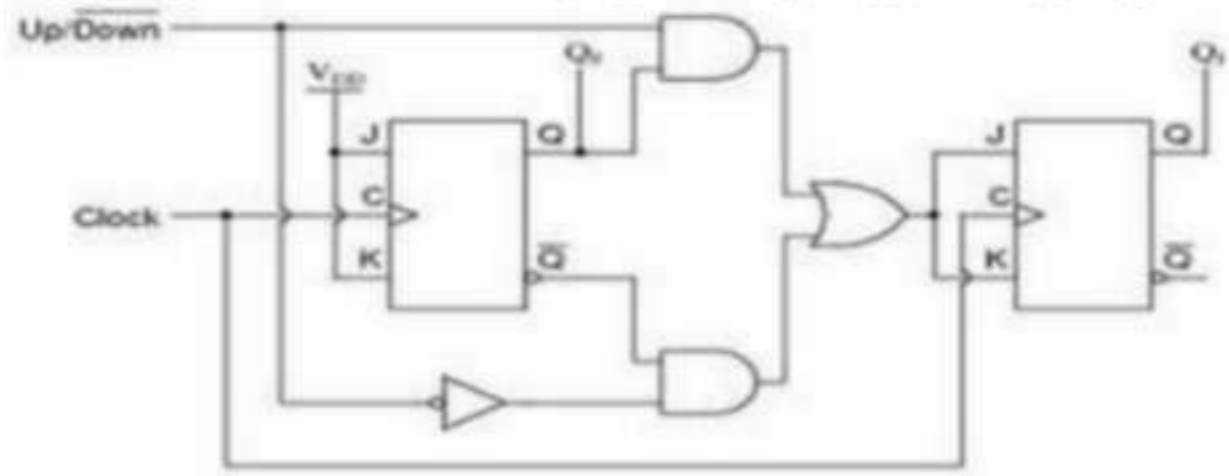


Figure: asynchronous 2-bit ripple up-down counter using negative edge triggered flip flop:

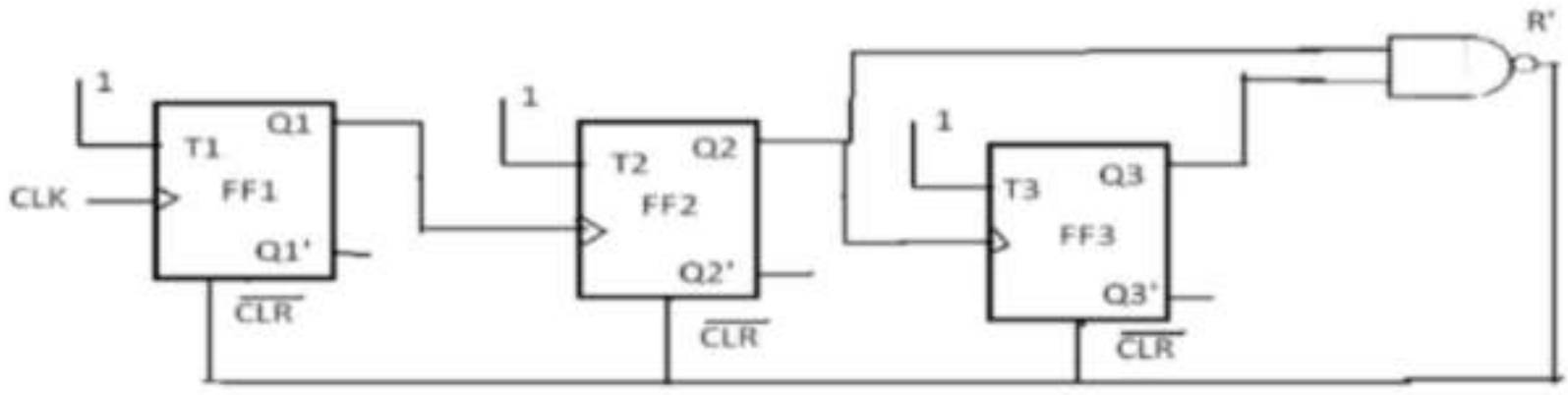
- As the name indicates an up-down counter is a counter which can count both in upward and downward directions. An up-down counter is also called a forward/backward counter or a bidirectional counter. So, a control signal or a mode signal M is required to choose the direction of count. When M=1 for up counting, Q1 is transmitted to clock of FF2 and when M=0 for down counting, Q1‘ is transmitted to clock of FF2. This is achieved by using two AND gates and one OR gates. The external clock signal is applied to FF1.
- Clock signal to FF2= (Q1.Up)+(Q1‘. Down)= Q1m+Q1‘M‘

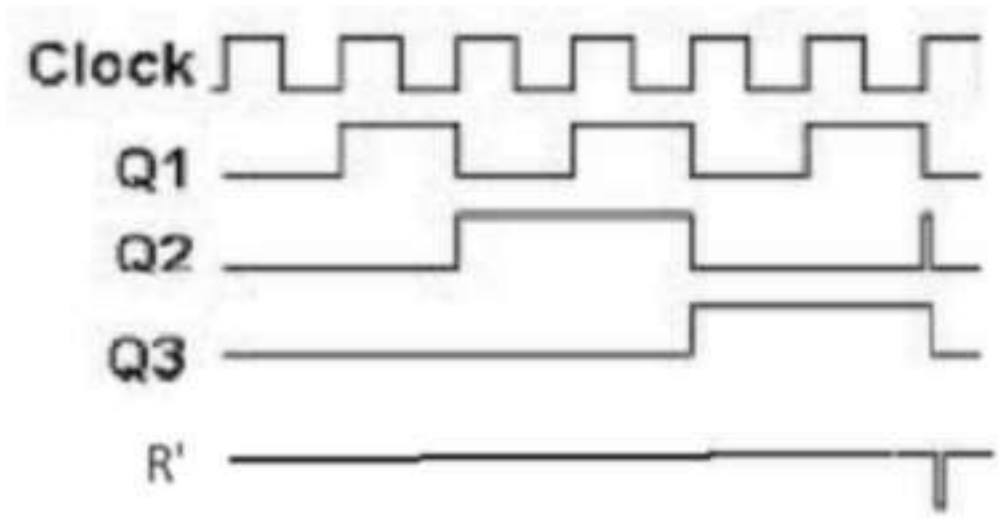
Design of Asynchronous counters:

To design a asynchronous counter, first we write the sequence , then tabulate the values of reset signal R for various states of the counter and obtain the minimal expression for R and R‘ using K-Map or any other method. Provide a feedback such that R and R‘ resets all the FF‘s after the desired count

Design of a Mod-6 asynchronous counter using T FFs:

A mod-6 counter has six stable states 000, 001, 010, 011, 100, and 101. When the sixth clock pulse is applied, the counter temporarily goes to 110 state, but immediately resets to 000 because of the feedback provided. it is -divide by-6-counterll, in the sense that it divides the input clock frequency by 6.it requires three FFs, because the smallest value of n satisfying the conditionN≤2^n is n=3; three FFs can have 8 possible states, out of which only six are utilized and the remaining two states 110and 111, are invalid. If initially the counter is in 000 state, then after the sixth clock pulse, it goes to 001, after the second clock pulse, it goes to 010, and so on.





After sixth clock pulse it goes to 000. For the design, write the truth table with present state outputs Q3, Q2 and Q1 as the variables, and reset R as the output and obtain an expression for R in terms of Q3, Q2, and Q1that decides the feedback into be provided. From the truth table, R=Q3Q2. For active-low Reset, R' is used. The reset pulse is of very short duration, of the order of nanoseconds and it is equal to the propagation delay time of the NAND gate used. The expression for R can also be determined as follows.

R=0 for 000 to 101, R=1 for 110, and R=X=for111

Therefore,

R=Q3Q2Q1'+Q3Q2Q1=Q3Q2

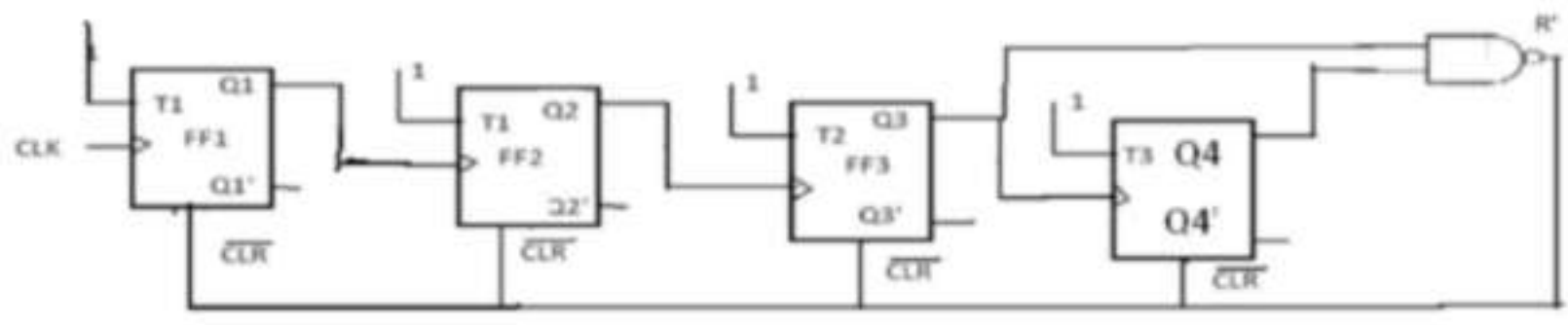
The logic diagram and timing diagram of Mod-6 counter is shown in the above fig.

The truth table is as shown in below.

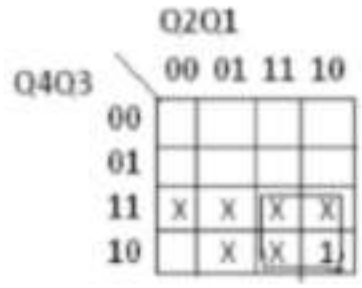
After pulses	States Q3	Q2	Q1	R
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	1
	↓	↓	↓	
	0	0	0	0
7	0	0	0	0

Design of a mod-10 asynchronous counter using T-flip-flops:

A mod-10 counter is a decade counter. It also called a BCD counter or a divide-by-10 counter. It requires four flip-flops (condition $10 \leq 2^n$ is n=4). So, there are 16 possible states, out of which ten are valid and remaining six are invalid. The counter has ten stable state, 0000 through 1001, i.e., it counts from 0 to 9. The initial state is 0000 and after nine clock pulses it goes to 1001. When the tenth clock pulse is applied, the counter goes to state 1010 temporarily, but because of the feedback provided, it resets to initial state 0000. So, there will be a glitch in the waveform of Q2. The state 1010 is a temporary state for which the reset signal R=1, R=0 for 0000 to 1001, and R=C for 1011 to 1111.



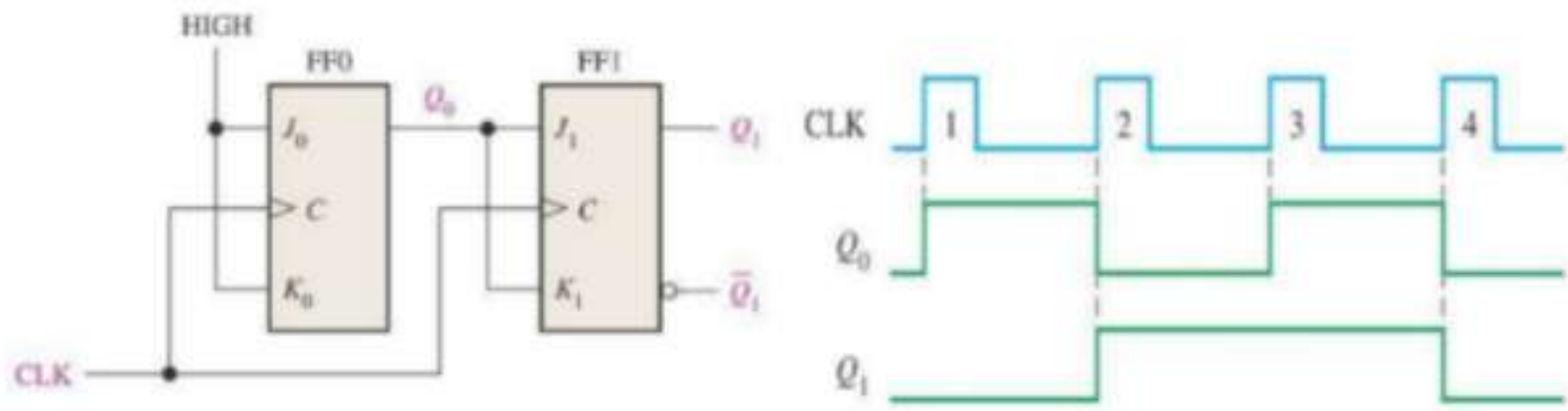
The count table and the K-Map for reset are shown in fig. from the K-Map R=Q4Q2. So, feedback is provided from second and fourth FFs. For active –HIGH reset, Q4Q2 is applied to the clear terminal. For active-LOW reset 4 2 is connected isof allFlip=flops.



After pulses	Count Q4	Q3	Q2	Q1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	0	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	0	1	0	1
10	0	0	0	0

Synchronous counters:

Asynchronous counters are serial counters. They are slow because each FF can change state only if all the preceding FFs have changed their state. if the clock frequency is very high, the asynchronous counter may skip some of the states. This problem is overcome in synchronous counters or parallel counters. Synchronous counters are counters in which all the flip flops are triggered simultaneously by the clock pulses Synchronous counters have a common clock pulse applied simultaneously to all flip-flops. A 2-Bit Synchronous Binary Counter



Design of synchronous counters:

For a systematic design of synchronous counters. The following procedure is used.

Step 1: State Diagram: draw the state diagram showing all the possible states state diagram which also be called nth transition diagrams, is a graphical means of depicting the sequence of states through which the counter progresses.

Step2: number of flip-flops: based on the description of the problem, determine the required number n of the flip-flops- the smallest value of n is such that the number of states $N \leq 2^n$--- and the desired counting sequence.

Step3: choice of flip-flops excitation table: select the type of flip-flop to be used and write the excitation table. An excitation table is a table that lists the present state (ps) , the next state(ns) and required excitations.

Step4: minimal expressions for excitations: obtain the minimal expressions for the excitations of the FF using K-maps drawn for the excitation of the flip-flops in terms of the present states and inputs.

Step5: logic diagram: draw a logic diagram based on the minimal expressions

Design of a synchronous 3-bit up-down counter using JK flip-flops:

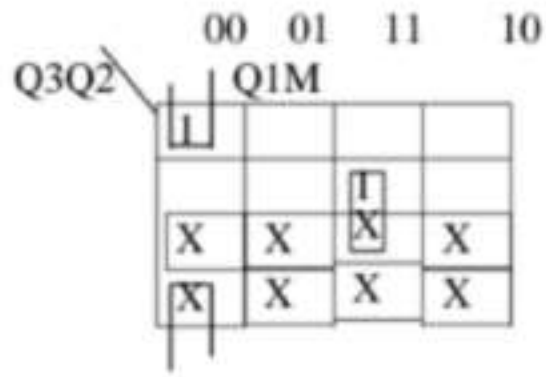
Step1: determine the number of flip-flops required. A 3-bit counter requires three FFs. It has 8 states (000,001,010,011,101,110,111) and all the states are valid. Hence no don't cares. For selecting up and down modes, a control or mode signal M is required. When the mode signal M=1 and counts down when M=0. The clock signal is applied to all the FFs simultaneously.

Step2: draw the state diagrams: the state diagram of the 3-bit up-down counter is drawn as

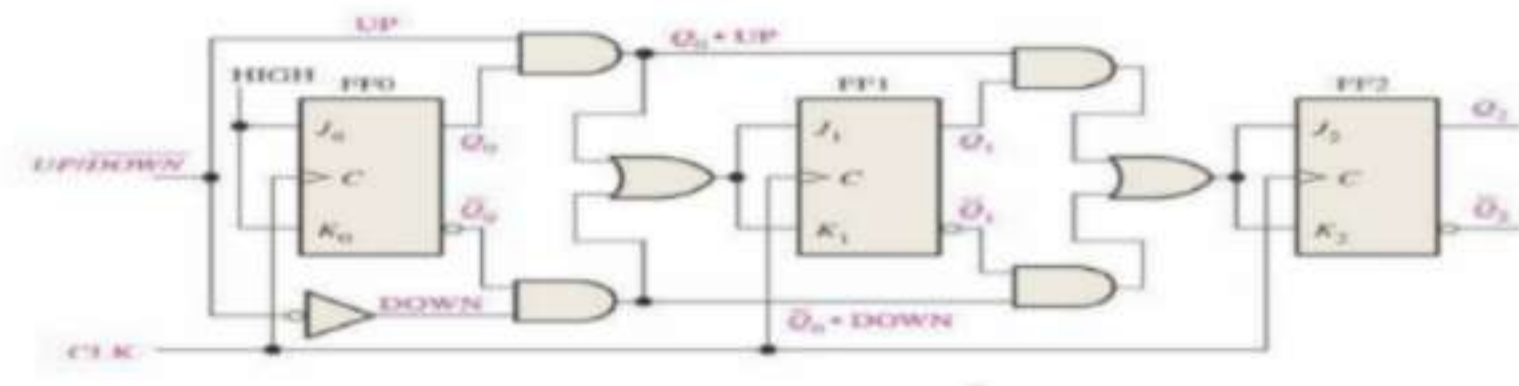
Step3: select the type of flip flop and draw the excitation table: JK flip-flops are selected and the excitation table of a 3-bit up-down counter using JK flip-flops is drawn as shown in fig.

PS			mode	NS			required excitations					
Q3	Q2	Q1	M	Q3	Q2	Q1	J3	K3	J2	K2	J1	K1
0	0	0	0	1	1	1	1	x	1	x	1	x
0	0	0	1	0	0	1	0	x	0	x	1	x
0	0	1	0	0	0	0	0	x	0	x	x	1
0	0	1	1	0	1	0	0	x	1	x	x	1
0	1	0	0	0	0	1	0	x	x	1	1	x
0	1	0	1	0	1	1	0	x	x	0	1	x
0	1	1	0	0	1	0	0	x	x	0	x	1
0	1	1	1	1	0	0	1	x	x	1	x	1
1	0	0	0	0	1	1	x	1	1	x	1	x
1	0	0	1	1	0	1	x	0	0	x	1	x
1	0	1	0	1	0	0	x	0	0	x	x	1
1	0	1	1	1	1	0	x	0	1	x	x	1
1	1	0	0	1	0	1	x	0	x	1	1	x
1	1	0	1	1	1	1	x	0	x	0	1	x
1	1	1	0	1	1	0	x	0	x	0	x	1
1	1	1	1	0	0	0	x	1	x	1	x	1

Step4: obtain the minimal expressions: From the excitation table we can conclude that J1=1 and K1=1, because all the entries for J1and K1 are either X or 1. The K-maps for J3, K3,J2 and K2 based on the excitation table and the minimal expression obtained from them are shown in fig.



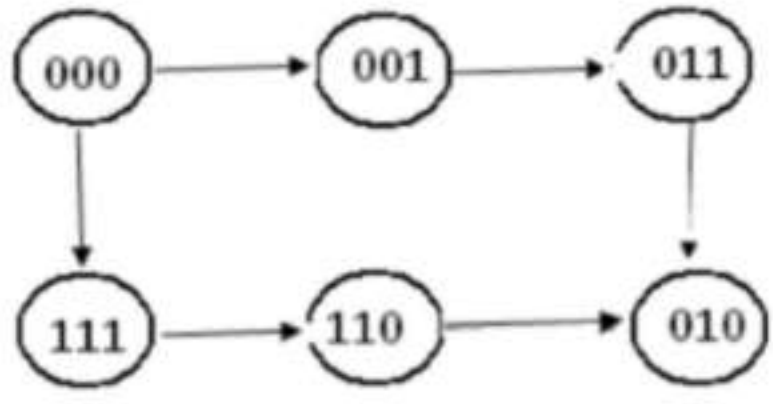
Step5: draw the logic diagram: a logic diagram using those minimal expressions can be drawn as shown in fig.



Design of a synchronous modulo-6 gray cod counter:

Step 1: the number of flip-flops: we know that the counting sequence for a modulo-6 gray code counter is 000, 001, 011, 010, 110, and 111. It requires n=3FFs ($N \leq 2^n$, i.e., $6 \leq 2^3$). 3 FFs can have 8 states. So the remaining two states 101 and 100 are invalid. The entries for excitation corresponding to invalid states are don't cares.

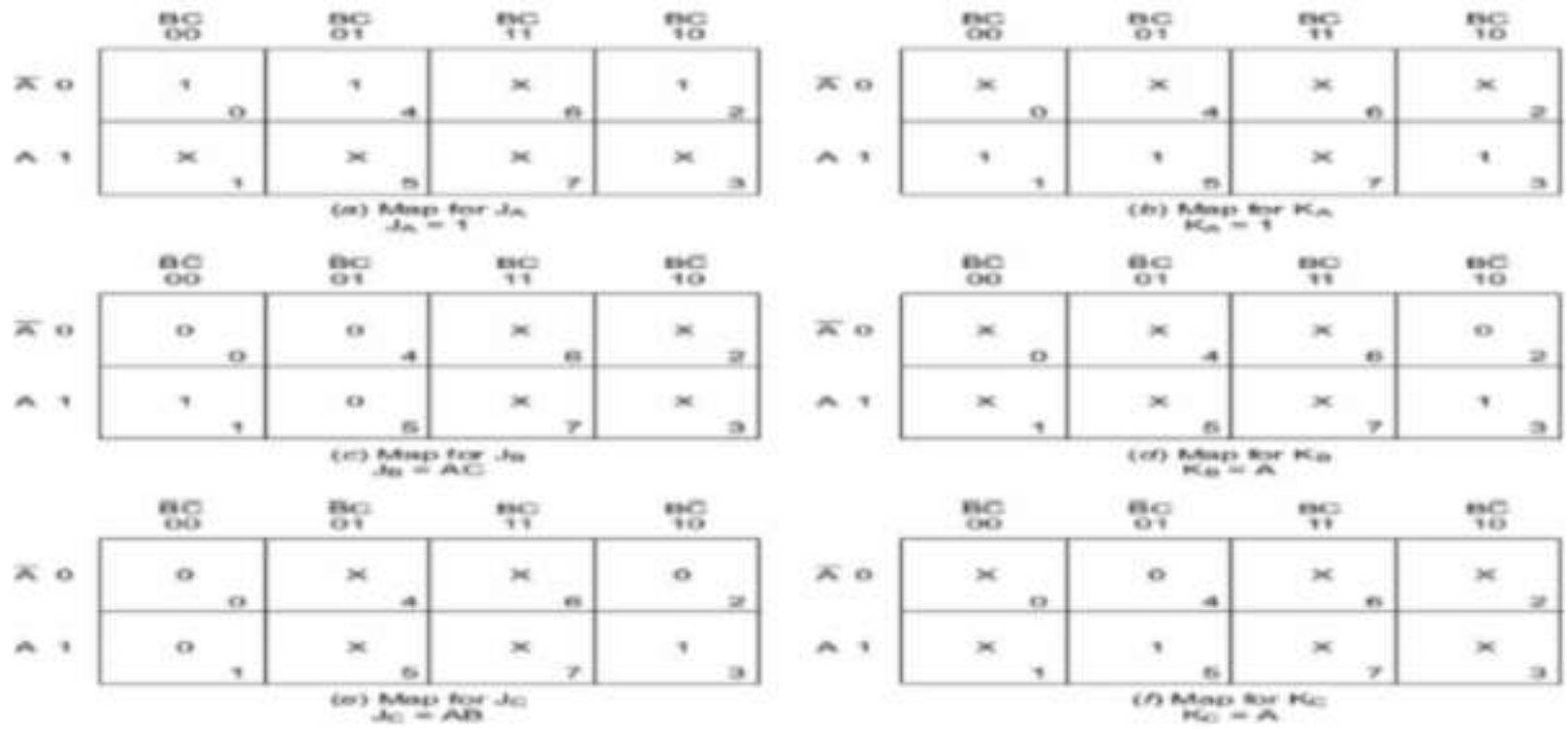
Step2: the state diagram: the state diagram of the mod-6 gray code converter is drawn as shown in fig.



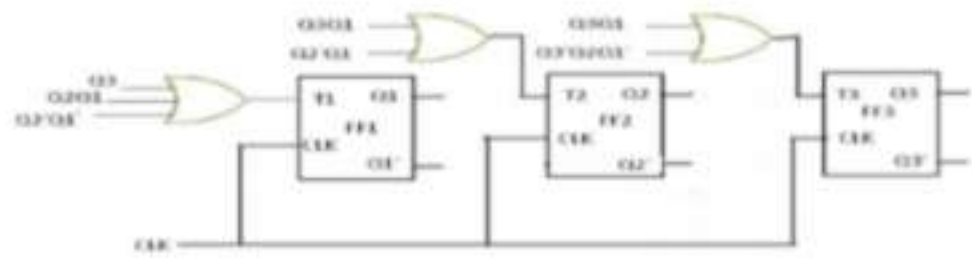
Step3: type of flip-flop and the excitation table: T flip-flops are selected and the excitation table of the mod-6 gray code counter using T-flip-flops is written as shown in fig.

PS			NS			required excitations		
Q3	**Q2**	**Q1**	**Q3**	**Q2**	**Q1**	**T3**	**T2**	**T1**
0	0	0	0	0	1	0	0	1
0	0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	0	1
0	1	0	1	1	0	1	0	0
1	1	0	1	1	1	0	0	1
1	1	1	0	0	0	1	1	1

Step4: The minimal expressions: the K-maps for excitations of FFs T3,T2,and T1 in terms of outputs of FFs Q3,Q2, and Q1, their minimization and the minimal expressions for excitations obtained from them are shown if fig



Step5: the logic diagram: the logic diagram based on those minimal expressions is drawn as shown in fig.



Design of a synchronous BCD Up-Down counter using FFs:

Step1: the number of flip-flops: a BCD counter is a mod-10 counter has 10 states (0000 through 1001) and so it requires n=4FFs($N \leq 2^n$„ i.e., $10 \leq 2^4$). 4 FFS can have 16 states. So out of 16 states, six states (1010 through 1111) are invalid. For selecting up and down mode, a control or mode signal M is required. , it counts up when M=1 and counts down when M=0. The clock signal is applied to all FFs.

Step2: the state diagram: The state diagram of the mod-10 up-down counter is drawn as shown in fig.

Step3: types of flip-flops and excitation table: T flip-flops are selected and the excitation table of the modulo-10 up down counter using T flip-flops is drawn as shown in fig.

The remaining minterms are don't cares($\sum$d(20,21,22,23,24,25,26,37,28,29,30,31)) from the excitation table we can see that T1=1 and the expression for T4,T3,T2 are asfollows.

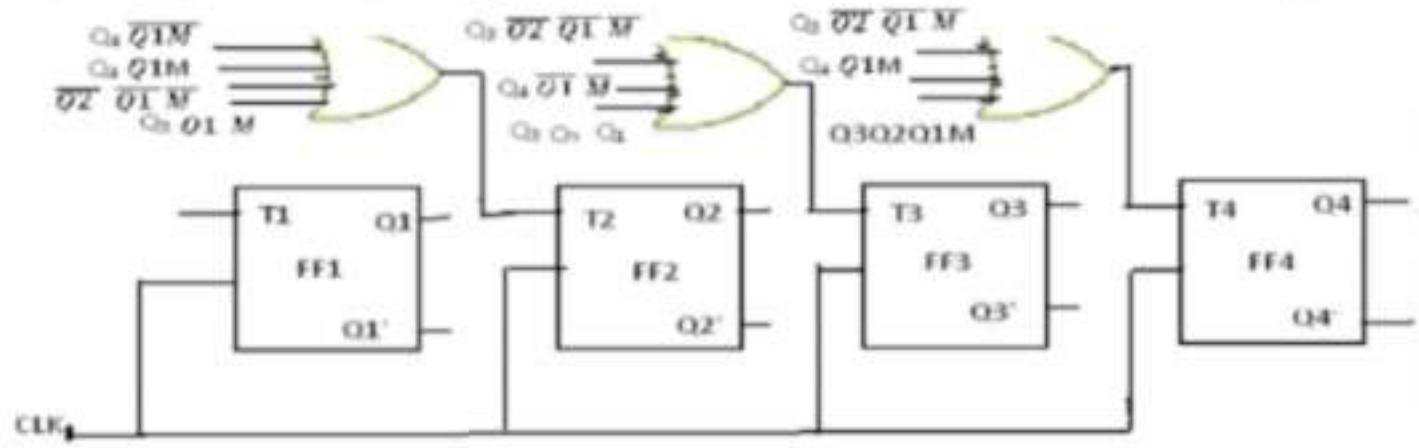
T4=$\sum$m(0,15,16,19)+d(20,21,22,23,24,25,26,27,28,29,30,31)
T3=$\sum$m(7,15,16,8)+d(20,21,22,23,24,25,26,27,28,29,30,31)
T2=$\sum$m(3,4,7,8,11,12,15,16)+d(20,21,22,23,24,25,26,27,28,29,30,31)

PS				**mode**	**NS**				**required excitations**			
Q4	**Q3**	**Q2**	**Q1**	**M**	**Q4**	**Q3**	**Q2**	**Q1**	**T4**	**T3**	**T2**	**T1**
0	0	0	0	0	1	0	0	1	1	0	0	1
0	0	0	0	1	0	0	0	1	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	1
0	0	0	1	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	0	0	1	0	0	1	1
0	0	1	0	1	0	0	1	1	0	0	0	1
0	0	1	1	0	0	0	1	0	0	0	0	1
0	0	1	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	0	0	1	1	0	1	1	1
0	1	0	0	1	0	1	0	1	0	0	0	1
0	1	0	1	0	0	1	0	0	0	0	0	1
0	1	0	1	1	0	1	1	0	0	0	1	1
0	1	1	0	0	0	1	0	1	0	0	1	1
0	1	1	0	1	0	1	1	1	0	0	0	1
0	1	1	1	0	0	1	1	0	0	0	0	1
0	1	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	0	1	0	0	0	1
1	0	0	1	0	1	0	0	0	0	0	0	1
1	0	0	1	1	0	0	0	0	1	0	0	1

Step4: The minimal expression: since there are 4 state variables and a mode signal, we require 5 variable kmaps. 20 conditions of Q4Q3Q2Q1M are valid and the remaining 12 combinations are invalid. So the entries for excitations corresponding to those invalid combinations are don't cares. Minimizing K-maps for T2 we get

$$T2 = Q4Q1'M + Q4'Q1M + Q2Q1'M' + Q3Q1'M'$$

Step5: the logic diagram: the logic diagram based on the above equation is shown in fig.



Shift register counters:

One of the applications of shift register is that they can be arranged to form several types of counters. The most widely used shift register counter is ring counter as well as the twisted ring counter.

Ring counter: this is the simplest shift register counter. The basic ring counter using D flip-flops is shown in fig. the realization of this counter using JK FFs. The Q output of each stage is connected to the D flip-flop connected back to the ring counter.

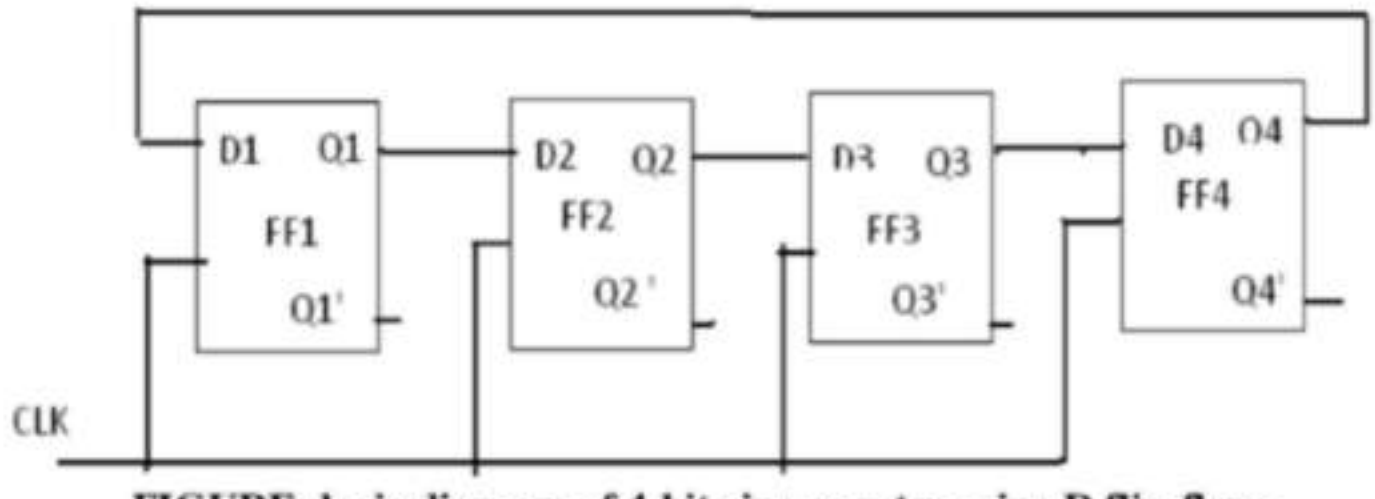
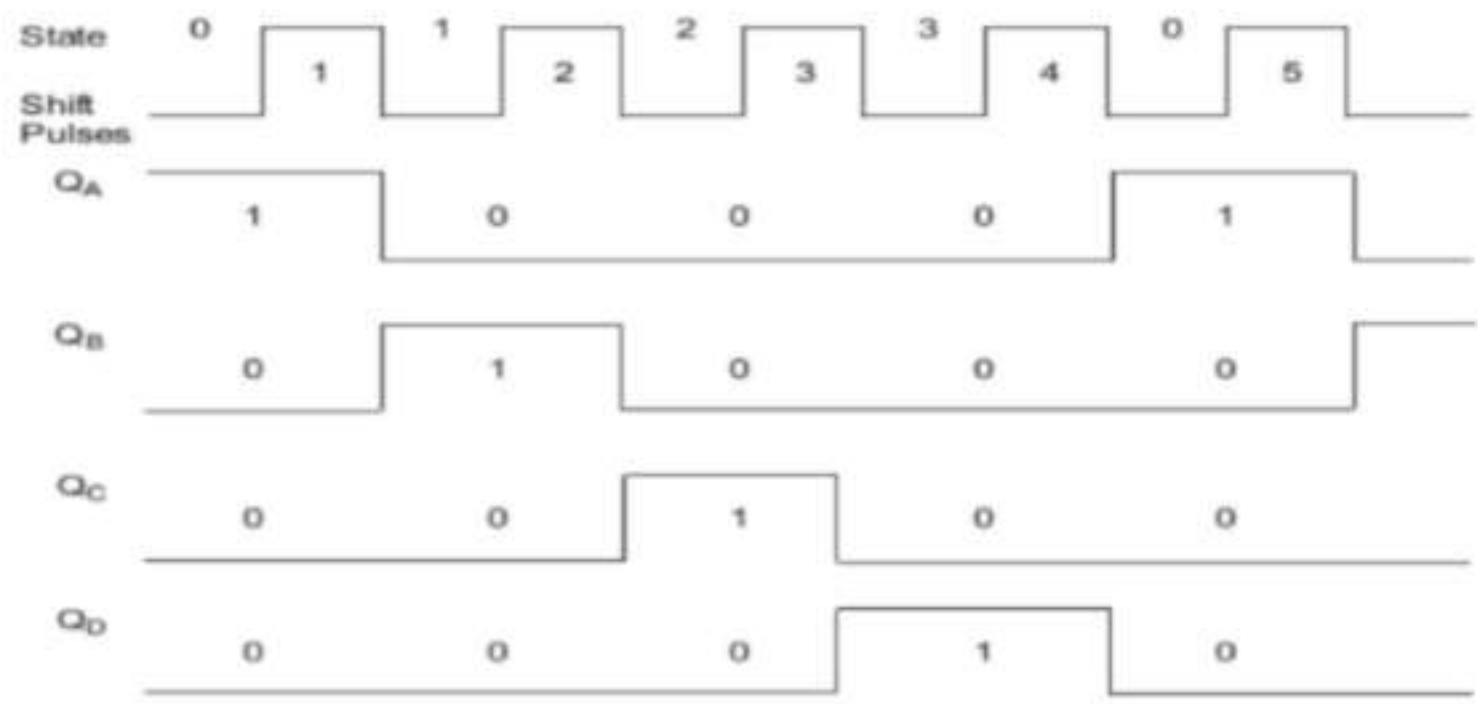


FIGURE: logic diagram of 4-bit ring counter using D flip-flops

Only a single 1 is in the register and is made to circulate around the register as long as clock pulses are applied. Initially the first FF is present to a 1. So, the initial state is 1000, i.e., Q1=1, Q2=0,Q3=0,Q4=0. After each clock pulse, the contents of the register are shifted to the right by one bit and Q4 is shifted back to Q1. The sequence repeats after four clock pulses. The number

of distinct states in the ring counter, i.e., the mod of the ring counter is equal to number of FFs used in the counter. An n-bit ring counter can count only n bits, where as n-bit ripple counter can count 2^n bits. So, the ring counter is uneconomical compared to a ripple counter but has advantage of requiring no decoder, since we can read the count by simply noting which FF is set. Since it is entirely a synchronous operation and requires no gates external FFs, it has the further advantage of being very fast.

Timing diagram:



Twisted Ring counter (Johnson counter):

This counter is obtained from a serial-in, serial-out shift register by providing feedback from the inverted output of the last FF to the D input of the first FF. the Q output of each is connected to the D input of the next stage, but the Q' output of the last stage is connected to the D input of the first stage, therefore, the name twisted ring counter. This feedback arrangement produces a unique sequence of states.

The logic diagram of a 4-bit Johnson counter using D FF is shown in fig. the realization of the same using J-K FFs is shown in fig.. The state diagram and the sequence table are shown in figure. The timing diagram of a Johnson counter is shown in figure.

Let initially all the FFs be reset, i.e., the state of the counter be 0000. After each clock pulse, the level of Q1 is shifted to Q2, the level of Q2to Q3, Q3 to Q4 and the level of Q4'to Q1 and the sequences given in fig.

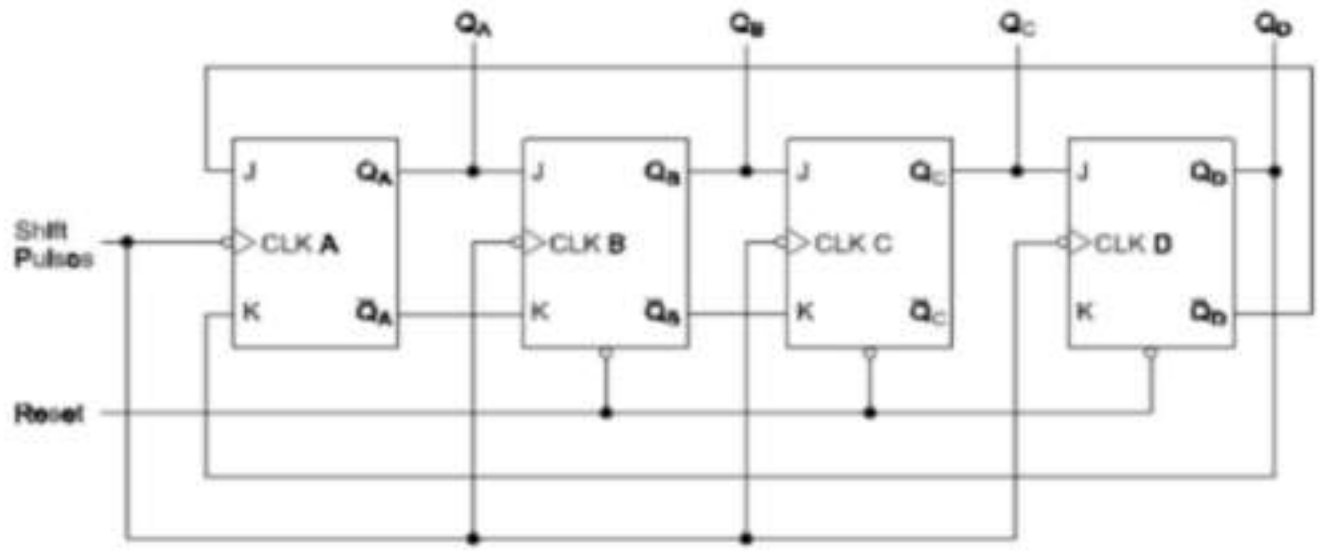


Figure: Johnson counter with JK flip-flops

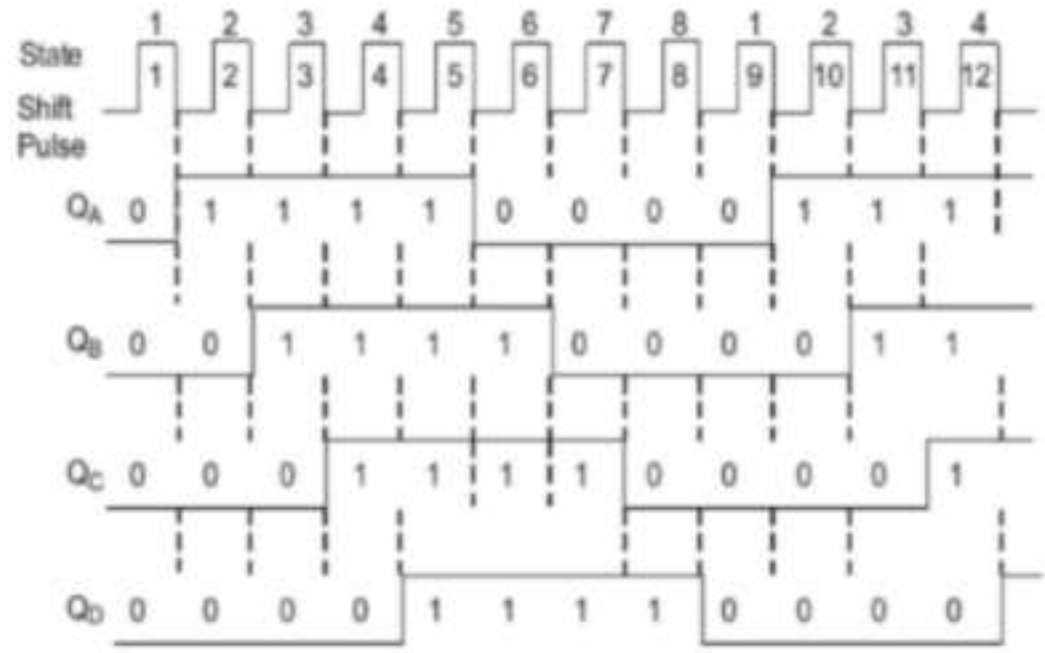


Figure: timing diagram

MEMORY

Memory structures are crucial in digital design. – ROM, PROM, EPROM, RAM, SRAM, (S)DRAM, RDRAM,..

- All memory structures have an address bus and a data bus – Possibly other control signals tocontrol output etc. •E.g. 4 Bit Address bus with 5 Bit Data Bus ADDR DOUT

- There are two types of memories that are used in digitalsystems:
  - Random-access memory(RAM): perform both the write and read operations.
  - Read-only memory(ROM): perform only the readoperation.
- The read-only memory is a programmable logic device. Other such units are the programmablelogic array(PLA), the programmable array logic(PAL), and the field-programmable gatearray(FPGA).

Random-Access Memory:

- A memory unit stores binary information in groups of bits called words.
  - byte = 8 bits
  - word = 2 bytes
- The communication between a memory and its environment is achieved through data input andoutput lines, address selection lines, and control lines that specify the direction oftransfer.

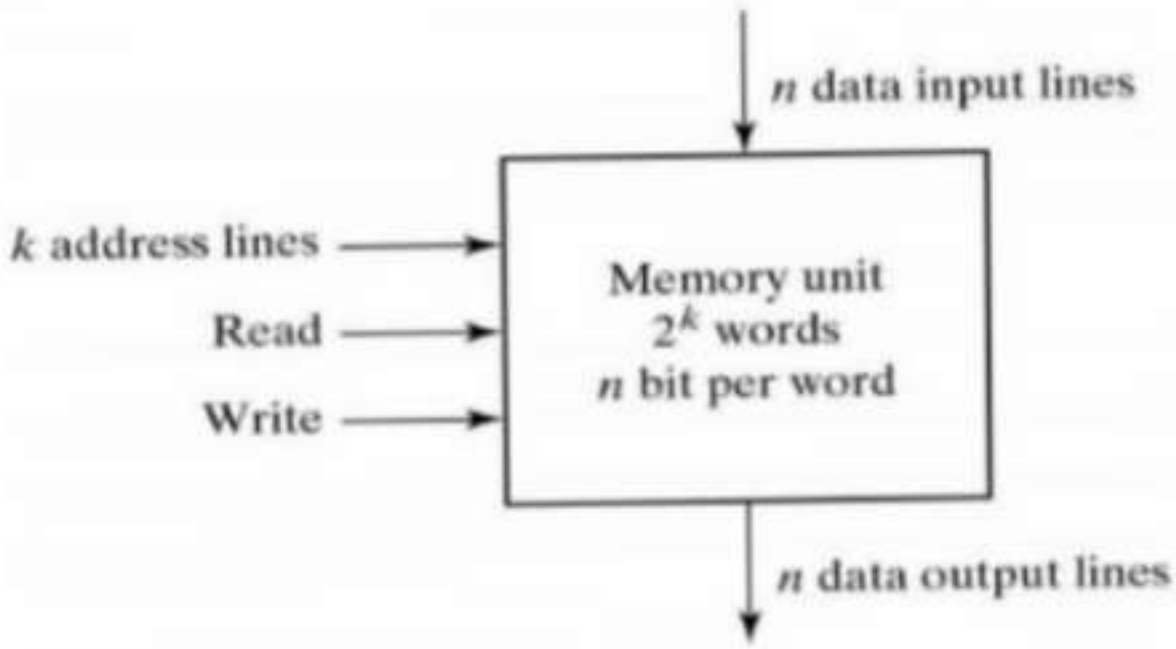


Fig. 7-2 Block Diagram of a Memory Unit

- In random-access memory, the word locations may be thought of as being separated in space, with each word occupying one particular location.
- In sequential-access memory, the information stored in some medium is not immediately accessible, but is available only certain intervals of time. A magnetic disk or tape unit is of this type.
- In a random-access memory, the access time is always the same regardless of the particular locationof the word.

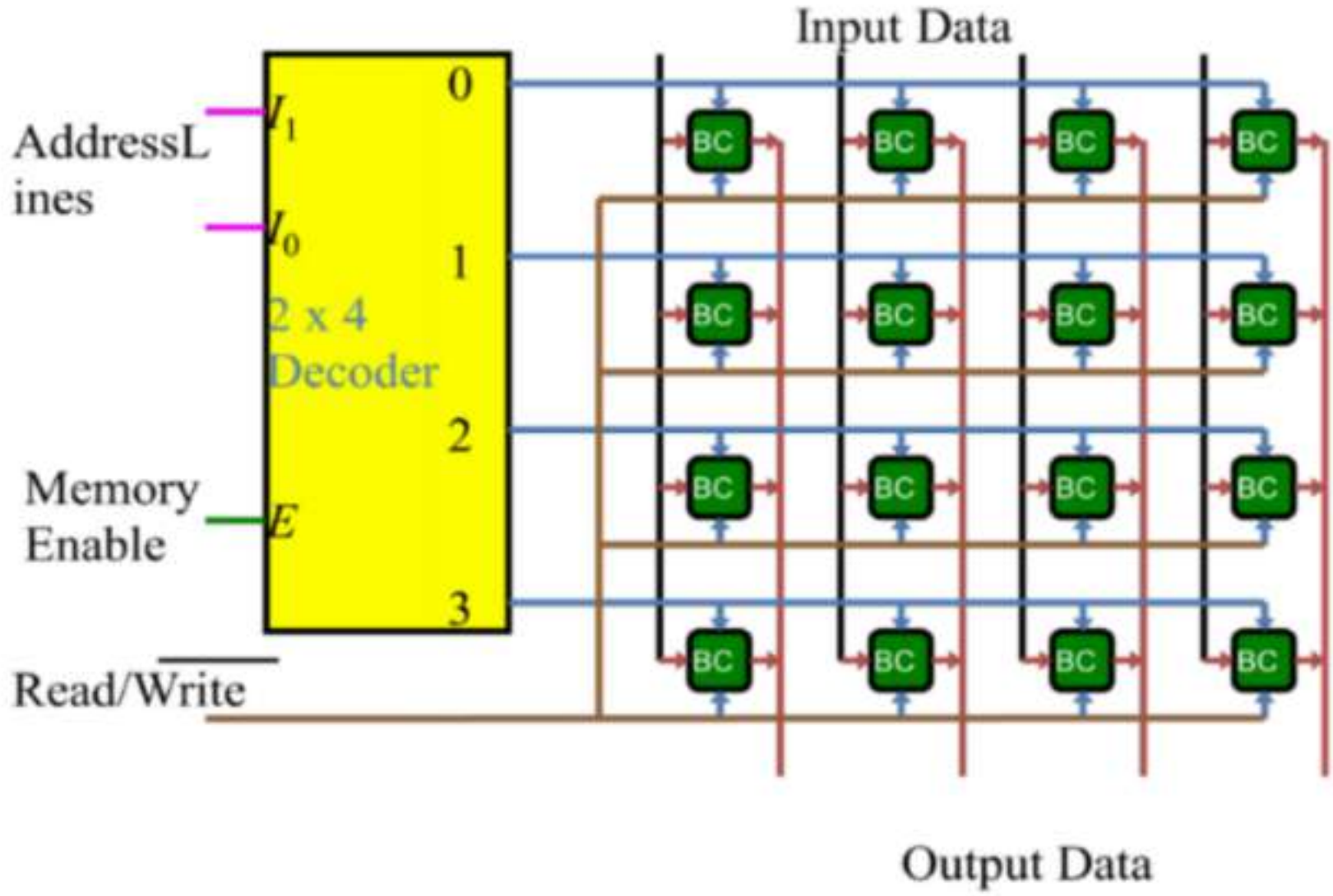
Static RAM

- SRAM consists essentially of internal latches that store the binaryinformation.
- The stored information remains valid as long as power is applied to theunit.
- SRAM is easier to use and has shorter read and write cycles.
- Low density, low capacity, high cost, high speed, high power consumption.

Dynamic RAM

- DRAM stores the binary information in the form of electric charges oncapacitors.
- The capacitors are provided inside the chip by MOStransistors.
- The capacitors tends to discharge with time and must be periodically recharged by refreshing the dynamic memory.
- DRAM offers reduced power consumption and larger storage capacity in a single memorychip.
- High density, high capacity, low cost, low speed, low power consumption.

Memory decoding



- The equivalent logic of a binary cell that stores one bit of information is shown below.
  - Read/Write = 0, select = 1, input data to S-R latch
  - Read/Write = 1, select = 1, output data from S-R latch

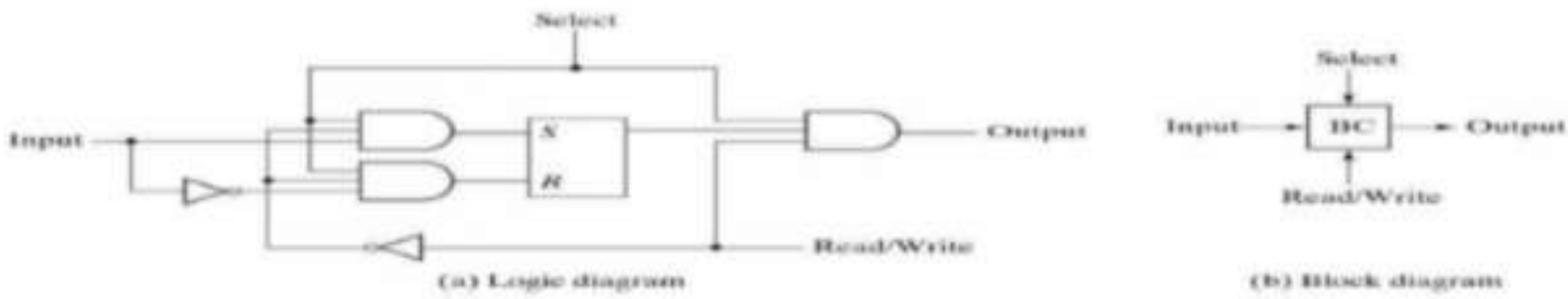


Fig. 7-5 Memory Cell

Programmable Logic Array:

- The decoder in PROM is replaced by an array of AND gates that can be programmed to generate any product term of the input variables.
- The product terms are then connected to OR gates to provide the sum of products forthe required Boolean functions.

➢ The output is inverted when the XOR input is connected to 1 (since $x \oplus 1 = x'$). The output doesn't change and connect to 0 (since $x \oplus 0 = x$).

F1 =

AB'+AC+A'BC'

F2 = (AC+BC)'

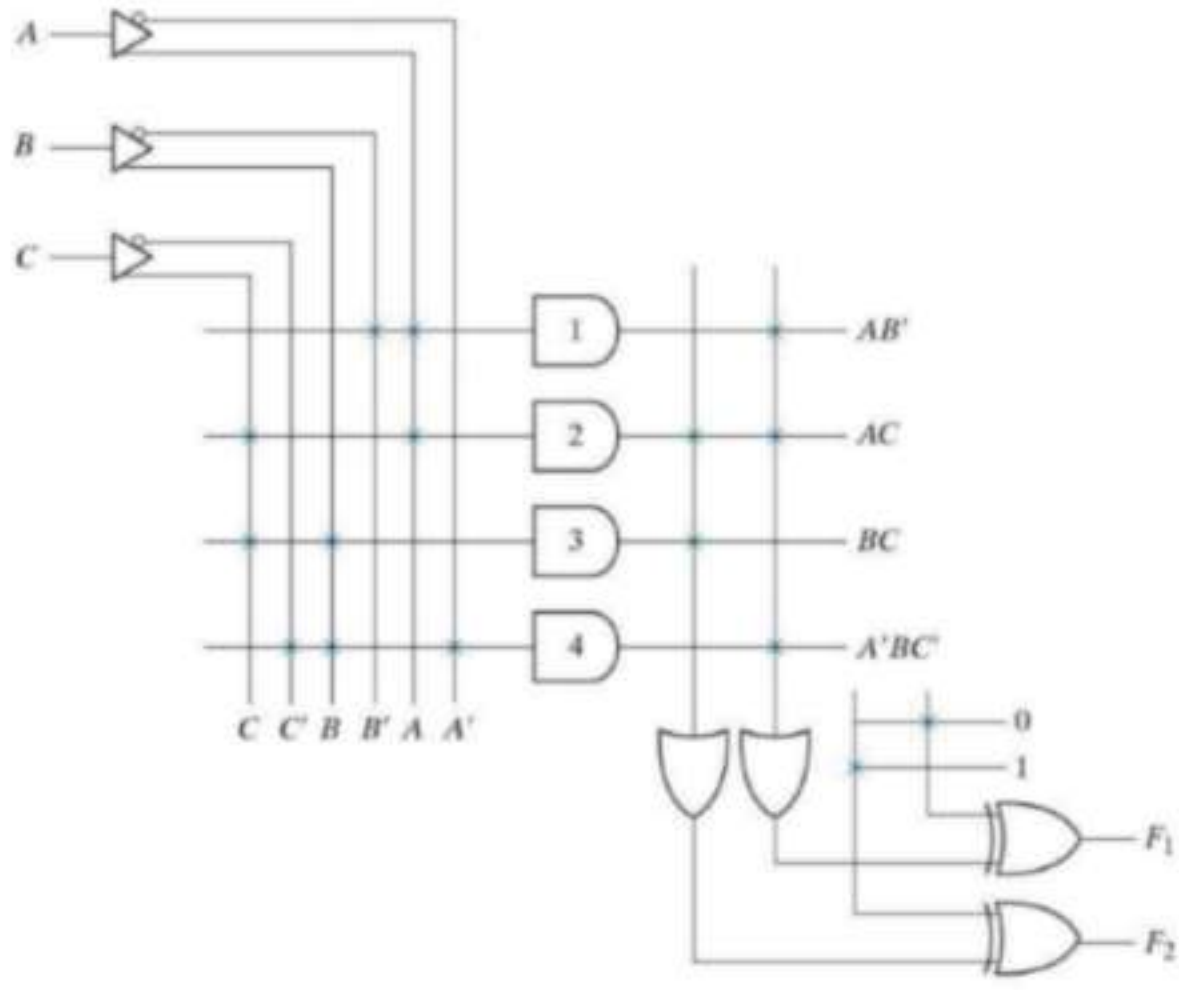
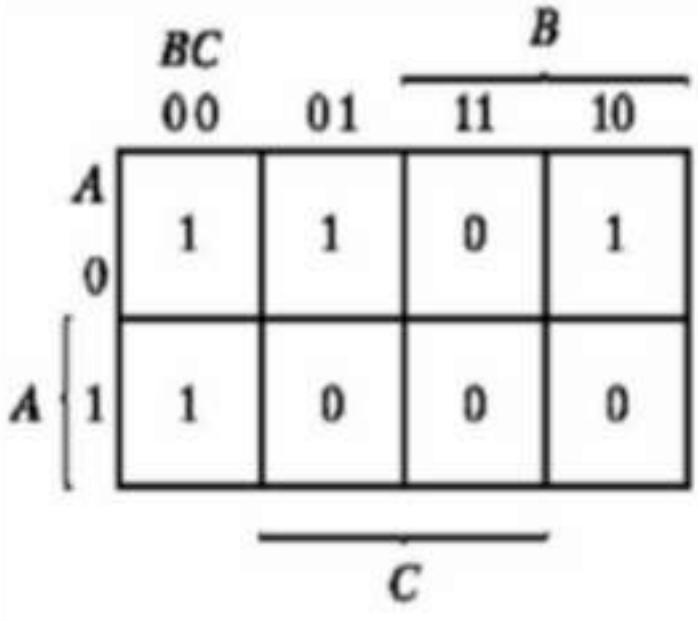


Fig. 7-14 PLA with 3 Inputs, 4 Product Terms, and 2 Outputs

Table 7-5
PLA Programming Table

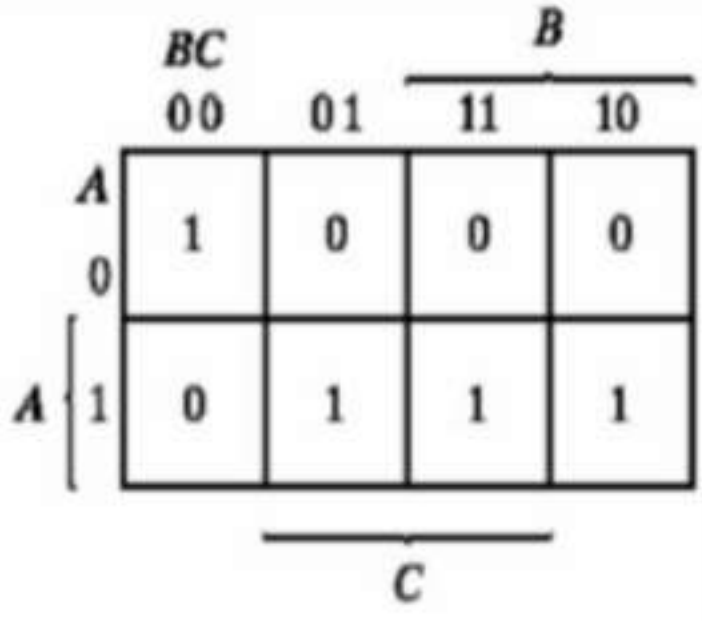
	Product Term	Inputs A	Inputs B	Inputs C	Outputs (T) F_1	Outputs (C) F_2
AB'	1	1	0	-	1	-
AC	2	1	-	1	1	1
BC	3	-	1	1	-	1
A'BC'	4	0	1	0	1	-

- **Implement the following two Boolean functions with a PLA:**
  - **$F_1(A, B, C) = \sum(0, 1, 2, 4)$**
  - **$F_2(A, B, C) = \sum(0, 5, 6, 7)$**



$F_1 = A'B' + A'C' + B'C'$

$F_1 = (AB + AC + BC)'$



$F_2 = AB + AC + A'B'C'$

$F_2 = (A'C + A'B + AB'C')'$

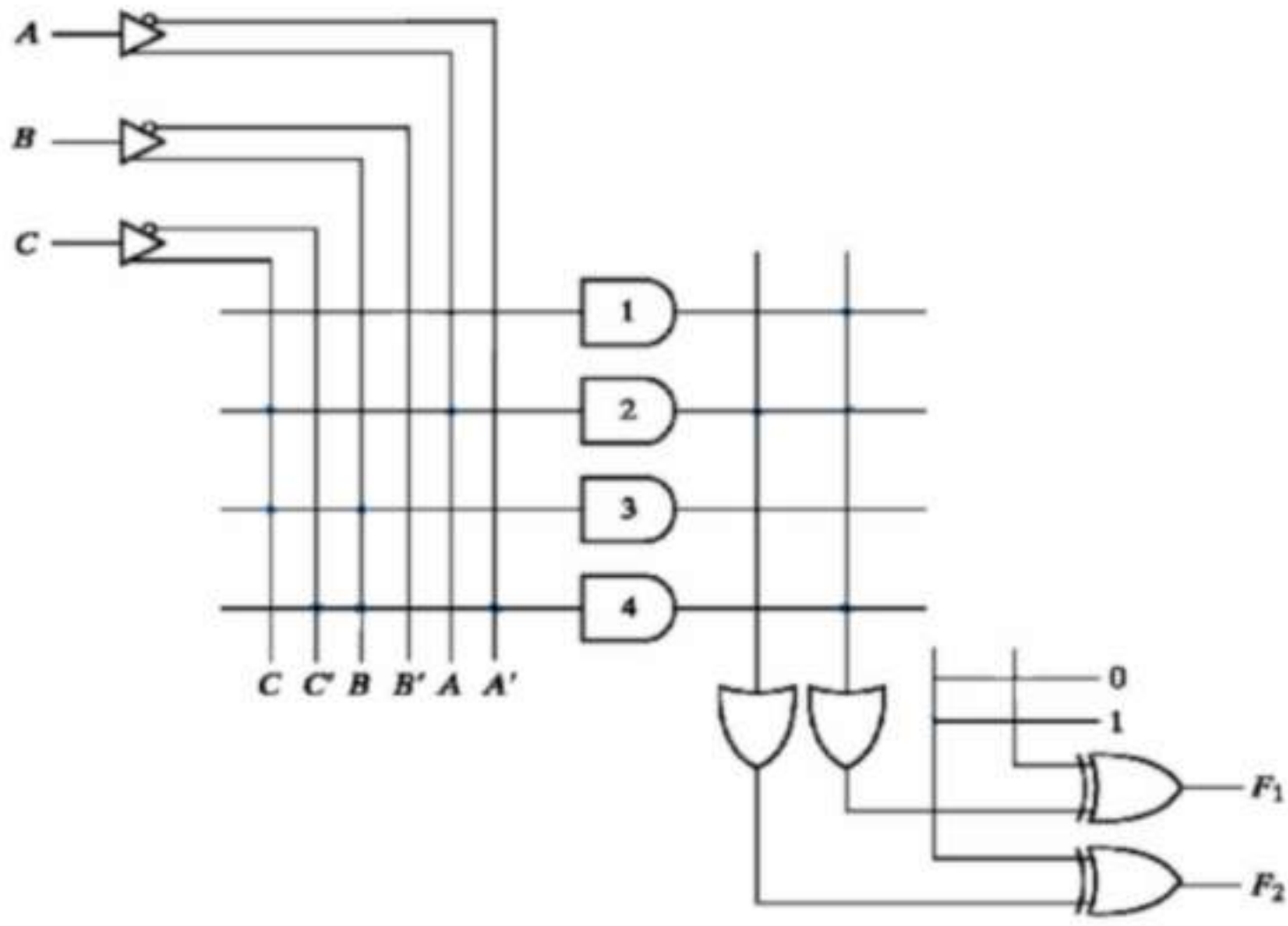
- Both the true and complement of the functions are simplified in sum of products.
- We can find the same terms from the group terms of the functions of F_1, F_1', F_2 and F_2' which will make the minimum terms.

$$F1 = (AB + AC + BC)'$$
$$F2 = AB + AC + A'B'C'$$

PLA programming table

	Product term	Inputs A	B	C	Outputs (C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
A'B'C'	4	0	0	0	–	1

Fig. 7-15 Solution to Example 7-2


A
B
C
1
2
3
4
C C' B B' A A'
0
1
F1
F2

MICROPROCESSOR ARCHITECTURE

The microprocessor is the central processing unit or cpu of a micro computer.it is the heart of the computer.

INTEL 8085:

It is an 8 bit Nmos microprocessor.it is an forty pin IC(integrated circuit) package fabricated on a single LSI (Large scale Integration) chip.

It uses a single +5 volt d.c.(Direct Current) supply for its operation.It clock spee is 3 mhz.It consists of 3 main sections.

1-Arithmetic Logic Unit(ALU)

2-Timing and control unit

3-Several Registers

Arithmetic Logic Unit:

It performs various arithmetic an logical operations like aition,substraction,logical an ,xor,or,not,increment etc.

Timing and control unit:

It generates timing an control signals hich are necessary for the execution of the instructions.it controls the ata flo beteen cpu an peripherals.

Several Registers:

Registers:-it is a collection of flip flops use to store a binary word.they are used by the microprocessor for the temporary storage and manipulation of data and instructions.

8085 has the following registers:

1-8 bit accumulator i.e. register A

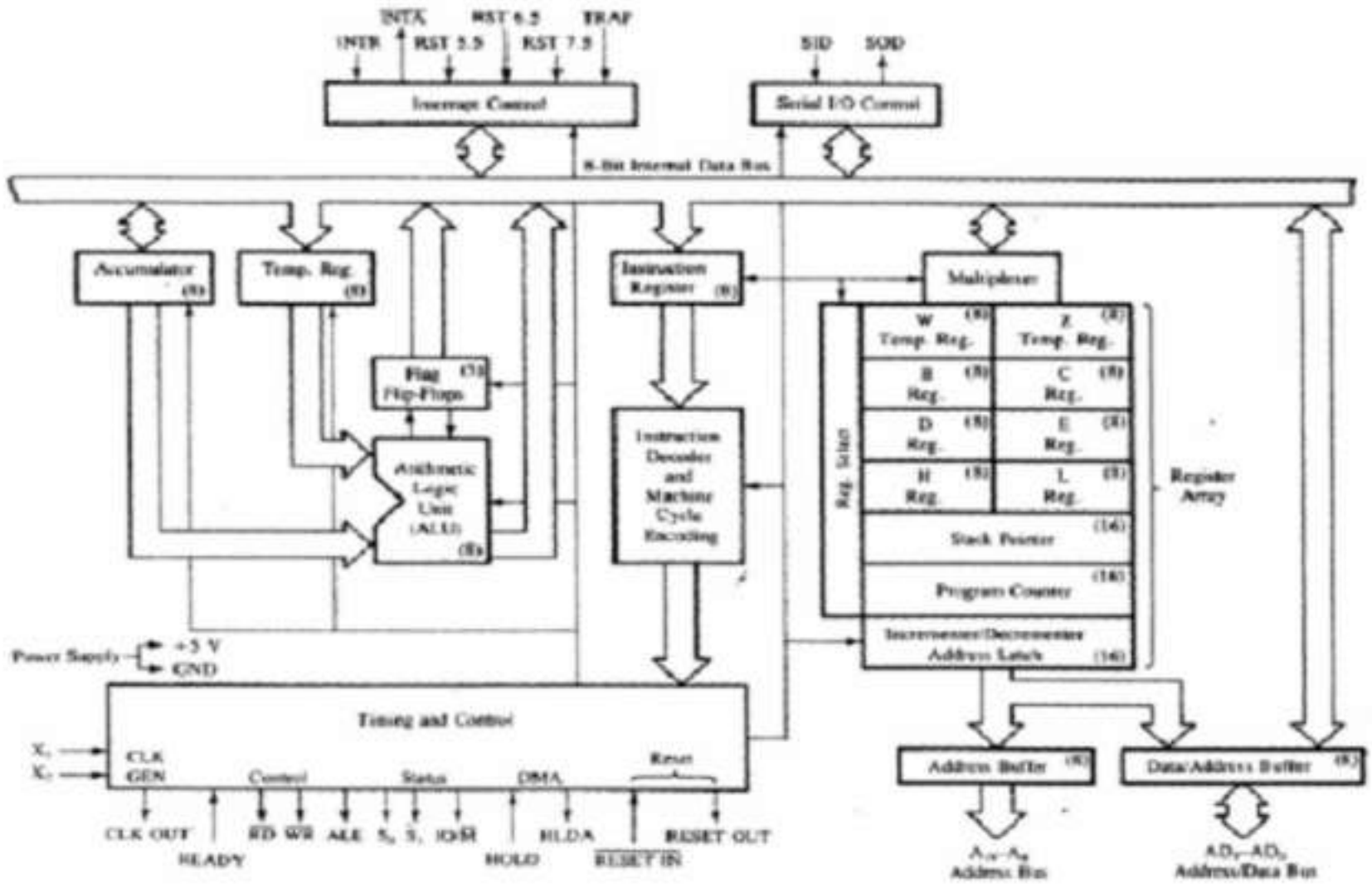
2-6 8 bits general purpose registers i.e. B,C,D,E,H,L

3-one 16 bit regiser i.e.stack pointer

4-16 bit Program counter,Status register,Temporary register,Instruction Register.

The regiser A holds the operands during program execution.

There are 6 8 bits general purpose registers B,C,,E,H,L are to handle 16 bit data.two 8 bit registers can be combined.this is called register pair.valid pair of 8085 are B-C,-E,H-L.The H-L pair is used to as address memory location.B-C,D-E are used for access another function.



BLOCK DIAGRAM OF 8085A

STACK POINTER:

Stack is a sequence of memory location defined by the programmer in LIFO function.That is last

Element to be placed on the stack is first one is to removed .The stack pointer contain the address of the stack cup.

PROGRAM COUNTER:

It is the address of the next instructions to be executed.

INSTRUCTION REGISTER:

It holds a copy of the current instruction until it is decoded.

STATUS REGISTER:

It contains the status flags of 8085 microprocessor.

TEMPORARY REGISTER:

It is used to store intermediate results and for intermediate calculations.

STATUS FLAGS:

It is a set of 5 flip-flops

i. Carry Flag(Cs)
ii. Sign Flag(S)
iii. Zero Flag(Z)
iv. Parity Flag(P)
v. Auxilarity carry flag(Ac)

Carry Flag:

It holds carry out of the resulting from the execution of an arithmetic operation.

If there is a carry from addition or a borrow from substraction or comparision,the carry flag is said to 1 ortherwise it is 0.

Sign Flag:

It is set to 1 if the MSB of the result of an arithemetic or logical operation is 1 ortherwise it is 0.

Zero Flag:

It is said to 1 if the result of an arithmetic or logical operation is zero.for non zero result,it is 0.

Parity Flag:

It is set to 1 when the the result of the operation contains even no.of 1& it is set to 0 if there are odd no.of 1.

Auxilary Carry Flag:

It holds carry from bit 3 to A resulting from the execution of an arithmetic operation.If there is a carry from bit 3 to 4,the AC flag is set to 1 ortherwise it is 0.

Program Status Word(PSW):

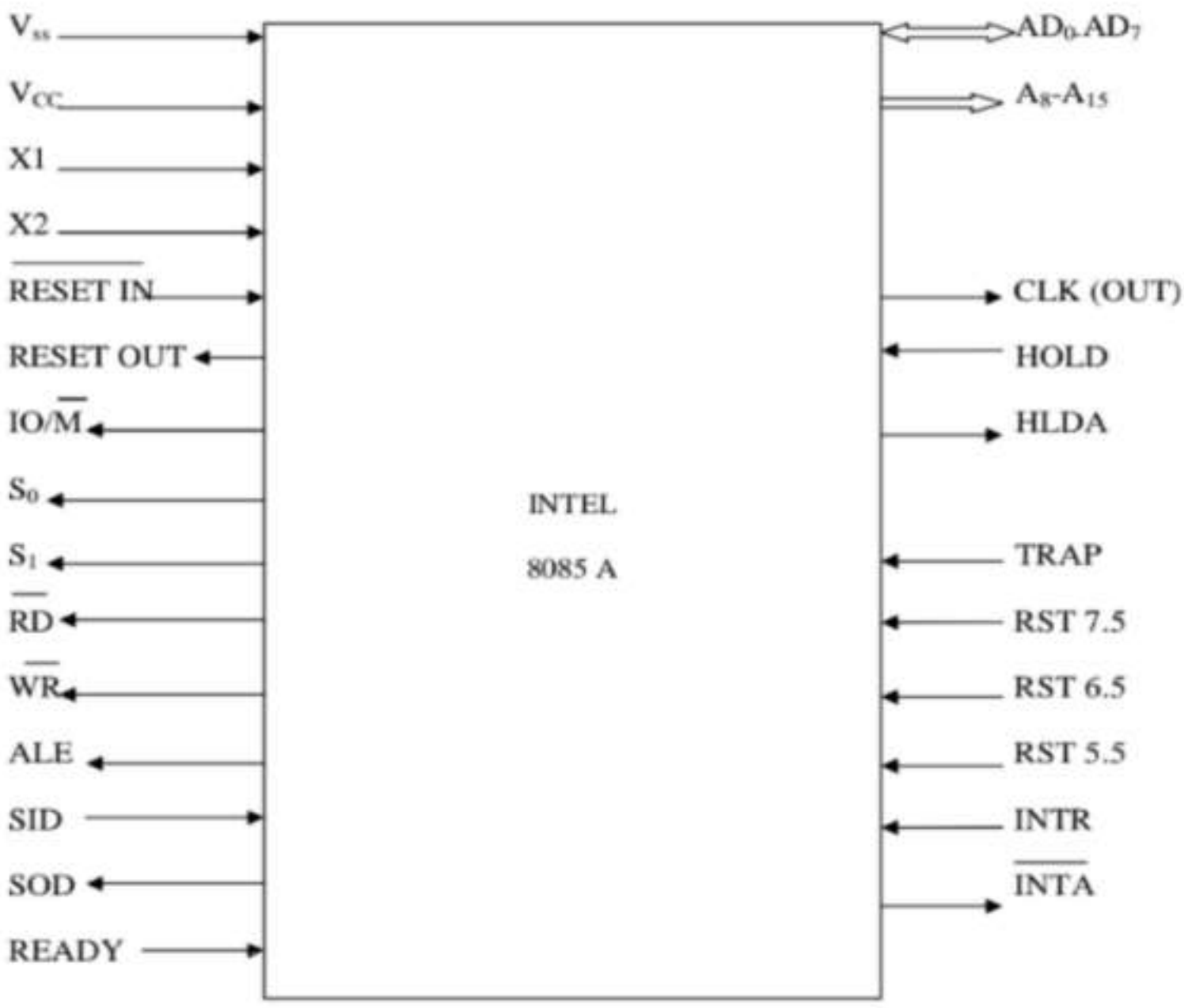
It is a combination of 8-bits where five bits indicates the 5 status flags & three bits are undefined.

Psw and the accumulator treated as a 16 bit unit for stack operation.

BUS ORGANISATION:

INTEL 8085 is a 8 bit micro processor.its data bus is 8 bit wide .8 bit of data can be transmitted in parallel form.or to the microprocessor.

Address bar is 16 bit wide as memory address are of 16 bit.8 msb is the address are transmitted by on A8-A15.8 LSB is is the address are transmitted by the data bus AD0-AD7.The address or data bus transmits data & address at different moments.it can transmits data or address at a time.



[SCHEMATIC /PIN DIAGRAM OF INTEL 8085]

PIN DESCRIPTION OF 8085

A_8-A_{15} (output)-These are address bus and are used for the most significant bits of the memory address or 8 bits of I/O address.

AD_0-AD_7 (input/output)-these are time multiplexed address /data bus that is they serve dual purpose .they are used for the least significant 8 bits of the memory address or I/O address during the first clock cycle of a machine cycle. Again they are used for data during second and third clock cycles.

ALE (output)-it is an address latch enable signal. It goes high during first clock cycle of a machine cycle and enables the lower 8 bits of the address to be latched either into the memory or external latch.

IO/$\overline{M}$(output)-it is a status signal which distinguishes whether the address is for memory or I/O. when it goes high the address on the address bus is for an I/O device. When it goes low the address on the address bus is for a memory location.

S_0, S_1 (output)-these are status signal sent by the microprocessor to distinguish the various types of operation

Status code for Intel 8085

S_1	S_0	Operations
0	0	HALT
0	1	WRITE
1	O	READ
1	1	FETCH

$\overline{RD}$ (output)-it is a signal to control READ operation .when it goes low the selected memory or I/O device is read.

$\overline{WR}$(output)-it is a signal to control WRITE operation .when it goes low the data on the data bus is written into the selected memory or I/O operation.

READY(input)-it is used by the microprocessor to sense whether a peripheral is ready to transfer data or not .a slow peripheral may be connected to the microprocessor through READY line. if READY is high the peripheral is ready .if it is low the microprocessor waits till it goes high.

HOLD (input)-it indicates that another device is requesting for the use of the address and data bus. Having received a HOLD request the microprocessor relinquishes the use of the buses as soon as the current machine cycle is completed. Internal processing may continue. the processor regains the bus after the removal of the HOLD signal. when a HOLD is acknowledged .

HLDA (output)-it is a signal for HOLD acknowledgement. It indicates that the HOLD request has been received. after the removal of a HOLD request the HLDA goes low. the CPU takes over the buses half clock cycle after the HLDA goes low.

INTR (input)-it is an interrupt request signal. Among interrupts it has the lowest priority. An interrupt is used by io devices to transfer data to the microprocessor without wasting its time.

$\overline{\text{INTA}}$ (output)-it is an interrupt acknowledgement sent by the microprocessor after INTR is received.

RST5.5, RST6.5, RST 7.5(input)-these are interrupts. Signals are the restart interrupt, they causes an internal restart to be automatically inserted each of them of a programmable mask.

TRAP-TRAP has the highest priority. It is used in emergency situation. it is an non-mask able interrupt.

Order of priority-

TRAP ⟶ RST 7.5 ⟶ RST 6.5 ⟶ RST 5.5 ⟶ INTR

When an interrupt is recognize the next instruction is executed from a fixed location in memory. A subroutine is executed which is called ISS(interrupt service subroutine).

INTERRUPTS	ISS ADDRESS
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

RESET IN (input)-it resets the program counter to zero .it also resets interrupts enable that is an HLDA flip-flops.

RESETOUT (output)-it indicates that the CPU is being reset.

X1, X2 (input)-these are terminals to be connected to an external crystal oscillator which drives an internal circuitry of the microprocessor to produce a suitable clock for the operation of microprocessor.

CLK (output)-it is a clock output for user, which can be used for other digital integrated circuits.

SID (input)-it is data line for serial input. The data on this line is loaded into the 7^{th} bit of the accumulator when rim (read interrupt mask) instruction is executed.

SOD (output)- it is data line for serial output. The 7^{th} bit of the accumulator is output on sod line when sim instruction is executed.

Vcc-it is +5 volt dc supply.

Vss-it is the ground reference.

INSTRUCTION AND DATA FORMATS

Intel 8085 is an 8-bit microprocessor. It handles 8-bit data at a time. One byte consists of 8-bits.A memory location for Intel 8085 microprocessor is designed to accumulate 8-bit data. If 16-bit data are to be stored, they are stored in consecutive memory locations. The address of memory location is 0f 16-bit i.e. 2 bytes.

The various techniques to specify data for instructions are:

(1) 8-bit or 16-bit data may be directly given in the instruction itself.

(2) The address of the memory location, I/O port or I/O device, where data resides, may be given in the instruction itself.

(3)In some instructions only one register is specified. The content of the specified register is one of the operand and other operand is the accumulator.

(4) Some instructions specify two registers. The contents of the registers are the required data.

Due to different ways of specifying data for instruction are not of same length.

So there are three types of instructions of Intel 8085:

(1)Single byte instruction

(2)2-byte instruction

(3)3-byte instruction

Single-Byte instruction.

The content information regarding operands in the opcode itself .These are of one byte.

Ex-MOV A,B ; Move the content of register B to A

78H is opcode for MOV A,B. The binary form of opcode 78H is 01111000. The first two bit i.e. 01 for MOV operation; the next 3 bits i.e. 111 for register A and last 3 bits 000 are for register B.

Two-Byte instruction.

In case of two byte instruction the 1st byte of the instruction is opcode and 2nd byte is either data or address.

Both bytes are stored in two consecutive memory locations.

Ex-MVI B,05; Move 05 to register B

06,05; MVI B,05 in the code form

Here in this case the 1st byte i.e. 06 is the opcode for MVI B and 2nd byte i.e. 05 is the data which is to be moved to register B.

Three-Byte instruction.

In case of three bytes instruction the 1st byte of instruction is opcode and 2nd and 3rd byte of instruction are either 16-bit data or 16-bit address.

They are stored in three consecutive memory locations.

Ex-LXI H, 2400H ; load H-L pair with 2400H

21,00,24; LXI H, 2400H in code form.

Here 1st byte i.e. 21 is the opcode for instruction LXI H. The 2nd byte i.e. 00 is 8 LSBs of data which is loaded in to register L. The 3rd byte i.e. 24 is 8 MSBs of data which is loaded in to register H.

ADDRESSING MODES OF 8085 :

Addressing mode: These are various technique to specify data for instruction

a) Direct addressing mode
b) Register addressing mode
c) Register addressing mode
d) Immediate addressing mode
e) Implicit addressing mode.

a) Direct addressing mode:

In this addressing mode the address of the operand is given in the instruction.

Ex: STA 2000H
IN 02H

b) Register addressing mode:

In this addressing mode the operands are in the general purpose register.

The opcode specify the address of the register and the operation to be

Perform.

Ex: MOV A,B

ADD B

c) Register indirect addressing mode:

i. In this addressing mode the address of the operand is specify by a register pair.

Ex: LXI H,2000H

MOV A,M

d) Immediate addressing mode:

i. In this adressing mode operand is specify with in the instruction.

ii. Ex: MVI A,05H // Move immediate data 05H to Accumulator.

e) Implicit addressing mode:

I. This instruction operates on the content of the accumulator.

II. They don't required operand address.

III. EX: CMA //Complement

DATA TRANSFER GROUP

1. MOV r1,r2 (Move data; move the content of one register to another)

[r1]←[r2]. State :none. addressing:register addressing. machine cycle:1.

The content of resister r2 is move to register is moved to register 1.For example,the instruction MOV A,B moves the contents of resister b to register A.The instruction MOV B,A moves the content of register A to register B.The time for the execution of this instruction is 4 clock period.One clock period is called is state.No lag is affected.

2. MOV r,M (move the content of memory to register)

[r]←[[H-L]]. State:7.flag:none. Addressing:register indirect. Machine cycle:2

The content of memory location,whose address is in H-L pair is moved to register r.

Example

LXI H,2000H	load H-L pair by 2000H
MOV B,M	Move the content of the memory location 2000H to register B.
HLT	Halt

In this example the instruction LXI H,2000H loads H-L pair with 2000Hwhich is the address of a memory loation.Then the instruction MOV B,M will move the content of memory location 2000H to register B.

3. MOV M,r. (Move the content of register to to memory)

[[H-L]]←[r]. States:7. Flag:none. Addressing:register indirect. Machine cycle:2

The content of register r is moved to the memory location address by H-L pair.For example,MOV M,C moves the content of register c to the memory location whose address is in H-L pair.

4. MVI r,data.(moves immediate data to register)

[r]←data. States:7. Flag:none.adressing:immediate.machine cycle :2

The 1st byte of the instruction is its opcode.the 2nd byte of the instruction is the data which is moved to register r.For example ,the MVI A,05 moves 05 to register A.In the code form it is written as 3E,05.The opcode for MVI A is 3E,05.The opcode for MVI A is 3E and 05 is the data which is to be moved to register A.

5. MVI M,data (Move immediate data to memory)

[[H-L]]←data . states:10.flags:none addressing:immediate/reg. Indirect. Machine cycle:3.

The data is moved to the memory location whose address is in H-L pair.

Example

LXI H,2400H	Load H-L pair with 2400H.
MVI M,08	Move 08 to the memory location 2400H.
HLT	Halt.

In the above example the instruction LXI H,2400H Loads H-L pair with 2400H which is the address of a memory location.Then the instruction MVI M,08 will move 08 to memory location 2400H.In the code form it is written as 36,08.The opcode for MVI M is 36 and 08 is the data which is to be moved to the memory location 2400H.

6. LXI rp, 16-bit data (load register pair immediate)

[rp]←data 16 bits, [rh]←8 MSBs, [rl]←8LSBs of data

States: 10, Flags: none, Addressing: Immediate, Machine Cycles: 3

This instruction loads 16 bit immediate data into register pair rp. This instruction is for register pair; only high order register is mentioned after the instruction. For example; H in the LXI H stands for H-L pair. Similarly, LXI B is for B-C pair. LXI H, 2500H loads 2500H into H-L pair. H with 2500H denotes that the data 2500 is in hexadecimal. In the code form it is written as 21,00,25. The 1st byte of the instruction 21 is the opcode for LXI H. The second byte 00 is of 8LSBs of the memory address and it is loaded inyo register L. The third byte 25 is 8 MSBS of the data and it is loaded into register H

7. LDA addr (Load accumulator direct)

[A]←[addr] States :13, flags :none,Addressing:Direct,Machine cycle:4

The content of memory location , whose address is specified by the 2nd and 3rd bytes of the instruction; is loaded into the accumulator.The instruction LDA 2400H will load the content of the memory location 2400H into the accumulator .Inthe code form it is written as 3A,00,24.The 1st byte 3A is the opcode of the instruction . The 2nd byte 00 is of 8LSBs of the memory address.The 3rd byte 24 is 8Msbs of the memory address.

8. STA addr (store accumulator direct)

[addr←[A].States:13.Flags:none.Addressing:direct.Machine cycle:4.

The content of the accumulator is stored in the memory location whose address is specified by tthe 2nd and 3rd byteof the instruction.STA 2000H will store the content of the accumulator in the memory location 2000H.

9. LHLD addr (load H-L pair direct).

[L]←[addr],[H]←[addr+1].States:16.Flags:none.Addressing:direct.Machine cycle:5

The content of the memory location ,whose address is specified by 2nd and 3rd bytes of the instruction,is loaded into rsister L.The content of the next memory location is loaded into resister H. For example ,LHLD 2500H will load the content of the memory location 2500H into register L. The content of the memory location 2501H is loaded into register H.

10. SHLD addr (store H-L pair direct)

[addr]←[L],[addr+1]←[H].States:16,Flags:none,Addressing:direct.Machine cycles:5

The content of the register L is stored in the memory location whose address is specified by the 2nd and 3rd bytes of the instruction.The content of register H is stored in the next memory location .For example ,SHLD 2500H will stored the content of register L in the memory location 2500H.The content of the register H is stored in the memory location2501H.

11. LDAX rp (LOAD accumulator indirect)

[A]←[[rp]].states:7,Flags:none,Addressing:register indirect,Machine cycle:2

The content of the memory location ,whose address is in the register pair rp,is loaded into the accumulator.For example ,LDAX B will load the content of the memory location,whose address is in B-C pair,into the accumulator.Th instruction is used only for B-C and D-E register pairs.

12. STAX rp (store accumulator indirect)

[[rp]]←[A].States:7.Flags:none.Addressing :register indirect.Machine cycles:2.

The content of the accumulator is stored in the memory location whose address is in the register pair rp.For example ,STAX D will stored the content of the accumulator in the memory location whose address is in D-E pair.This instruction is true only for register pair B-C and D-E.

13. XCHG (Exchange the content of the H-L with D-E pair)

[H-L]←[D-E].States:4,Flags:none,Addressing :register,Machine cycle:1.

The content of H-L pair are exchanged with contents of D-E pair

ARITHMETIC GROUP

The Instruction of this group performs arithmetic operation such as Addition, Subtraction, Increment or Decrement of the content of the register or memory.

1. Add r (Add register to accumulator)
   [A] ← [A] + [r]

The content of register r is added to the content of the accumulator,and the sum is placed in the accumulator.

2. ADD M(Add memory to accumulator)
   [A] ← [A] + [H-L]

The content of the memory location addressed by H-L pair is added to the content to the accumulator. The sum is placed in the accumulator.

3. ADC r (Add register with carry to accumulator)
   [A] ← [A] + [r] + [CS]

The content of register r and carry status are added to the content of the accumulator. The sum is placed in the accumulator.

4. ADC M (Add memory with carry to accumulator)

   [A] ← [A] + [H-L] + [CS]

The content of the memory location addressed by H-L pair add carry status are added to the content of the accumulator. The sum is placed in the accumulator.

5. ADI data (Add immediate data to accumulator)
   [A] ← [A] + data

The immediate data is added to the content to the accumulator. The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is data and it is added to the content of the accumulator. The sum is placed in the accumulator.

FOR EXAMPLE:

The instruction ADI 08 will add 08 to the content of the accumulator and placed the result in the accumulator. In code form the instruction is written as C6 08.

accumulator.

BRANCH CONTROL GROUP

The instruction of this group change the normal sequence of the program.

There are of two types of branch instruction

- **Conditional branch instruction**
- **Unconditional branch instruction**

Conditional branch instruction:-

It transfer the program to the specified level when certain condition is satisfied.

Unconditional branch instruction:-

It transfer the program to the specified level unconditionally.

Example:-JMP addr(label).

***Conditional Jump addr(label):**

1. JZ addr(label):-Jump if the result is zero,Z=1.
2. JNZ addr(label):-jump if the result is not zero,Z=0.
3. JCaddr(label):-jump if there is a carry,CS=1.
4. JNC addr(label):-jump if there is no carry,CS=0.
5. JP addr(label):-jump if the result is plus,S=0.
6. JM addr(label):-jump if the result is minus,S=1.
7. JPE addr(label):-jump if even parity,P=1.
8. JPO addr(label):-jump if odd parity,P=0.

***CALL addr(label):-**

- Used in unconditional branch instruction.
- Used to call a sub-routine,before control its transfer to the subroutine .
- The content of program counter is saved in the stack.
- Call is 3-byte instruction.

***Conditional CALL addr(label):**

1. CC addr(label):-call subroutine if carry status CS=1.
2. CNC addr(label):-call subroutine if carry status CS=0.

3. CZ addr(label):-call subroutine if the result is zero ;the zero status Z=1.
4. CNZ addr(label):-call subroutineif the result is not zero;the zero status Z=0.
5. CP addr(label):-call subroutineif the result is plus;the sign status S=0.
6. CM addr(label):-call subroutineif the result is minus;the sign status S=1.
7. CPE addr(label):-call subroutine if even parity;the parity status P=1.
8. CPO addr(label):-call subroutine if odd parity;the parity status P=0.

***RET(Return sub routine):-**

- It is used at the end of a subroutine.
- Before the execution of a subroutine the address of the next instruction of the main program is saved in the stack.
- The content of the stack pointer is incremented by 2 to indicate the new stack top.

***Conditional Return:-**

1. RC:-Return from subroutine if carry status CS=1.
2. RNC:-Return from subroutine if carry status CS=0.
3. RZ:-Return from subroutine if the result is zero;the zero status Z=1.
4. RNZ:-Return from subroutineif the result is not zero;the zero status Z=0.
5. RP:-Return from subroutineif the result isplus;the sign statusS=0.
6. RM:-:-Return from subroutine if the result is minus ,the sign status S=1.
7. RPE:-:-Return from subroutine if even parity,the parity status P=1.
8. RPO:-_:-Return from subroutine if odd parity,the parity status P=0.

***RST n (restart) Instruction:-**

- It is a one-word call instruction the content of a program counter is saved in the stack,the program jumps to the instruction ,starting at restart location.
- The address of the restart location is 8 times n.

There are 8 RST restart instruction carrying from RST0-RST7.

- These are software interrupts used by the programmer to interrupt the microprocessor.
- The restart instruction and location are as follows:-

Instruction	Opcode	Restart Location
RST0	C7	0000

RST1	CF	0008	
RST2D7		0010	
RST3		DF	0018
RST4		E7	0020
RST5		EF	0028
RST6		F7	0030
RST7FF		0038	

*PCHL instruction:-

- Jump to address specified by H-Lpair.
- The content of H-Lpairare transferred to the program counter.
- The content of register L will be loaded to 8 LSBs of PC and content of register H will loaded to 8 MSBs.

EXAMPLES OF ASSEMBLY LANGUAGE PROGRAMS

1. ALP FOR ADDITION OF TWO 8-BIT NUMBERS; SUM 8-BIT

Mnemonics	Operand	Comments
LXI	H, 2501 H	Get Address of 1st No. in H-L pair
MOV	A, M	1st no. in accumulator
INX	H	Increment content of H-L pair
ADD	M	Add 1st no. and 2nd no.
STA	2503 H	Store sum in 2503 H.
HLT		Stop the program.

DATA

2501- 49 H

2502- 56 H

The sum is stored in memory location 2503 H.

RESULT

2503- 9F H.

2. ALP FOR SUBTRACTION OF TWO 8-BIT NUMBERS

Mnemonics	Operand	Comments
LXI	H, 2501 H	Get address of 1st no. in H-L pair.
MOV	A, M	1st no. in accumulator.
INX	H	Content of H-L pair in 2502 H.
SUB	M	1st no. – 2nd no.
INX	H	Content of H-L pair becomes 2503 H.
MOV	M, A	Store results in 2503 H.
HLT		Stop the program.

DATA

2501- 49 H

2502- 32 H

The result is stored in memory location 2503 H.

RESULT

2503- 17 H.

3. ALP FOR ADDITION OF TWO 8-BIT NUMBERS; SUM:16-BITS

Labels	Mnemonics	Operand	Comment
	LXI	H, 2501 H	Address of 1st no. in H-L pair.
	MVI	C, 00	MSBs of sum in register C. Initial value = 00.
	MOV	A, M	1st no. in accumulator.
	INX	H	Address of 2nd no. 2502 in H-L pair.

	ADD	M	1st no. + 2nd no.
	JNC	AHEAD	Is carry? No, go to the label AHEAD.
	INR	C	Yes, increment C.
AHEAD:	STA	2503 H	LSBs of sum in 2503 H.
	MOV	A , C	MSBs of sum in accumulator.
	STA	2504 H	MSBs of sum in 2504 H.
	HLT		Stop the program.

DATA

2501- 98 H

2502- 9A H

RESULT

2503- 32 H, LSBs of sum.

2504- 01 H, MSBs of sum.

4. ALP FOR DECIMAL ADDITION OF TWO 8-BIT NUMBERS; SUM:16-BITS

Labels	Mnemonics	Operand	Comment
	LXI	H, 2501 H	Address of 1st no. in H-L pair.
	MVI	C, 00	MSDs of sum in register C. Initial value = 00.
	MOV	A, M	1st no. in accumulator.
	INX	H	Address of 2nd no. 2502 in H-L pair.
	ADD	M	1st no. + 2nd no.
	DAA		Decimal adjust.
	JNC	AHEAD	Is carry? No, go to the label AHEAD.
	INR	C	Yes , increment C.
AHEAD:	STA	2503 H	LSDs of sum in 2503 H.

	MOV	A, C	MSDs of sum in accumulator.
	STA	2504 H	MSDs of sum in 2504 H.
	HLT		Stop the program.

DATA

2501- 84 D

2502- 75 D

RESULT

2503- 59 D, LSDs of the sum.

2504- 01 D, MSDs of the sum.

5. ALP FOR 8-BIT DECIMAL SUBTRACTION

Mnemonics	Operand	Comments
LXI	H, 2502 H	Get address of 2nd no. in H-L pair.
MVI	A, 99	Place 99 in accumulator.
SUB	M	9's compliment of 2nd no.
INR	A	10's compliment of 2nd no.
DCX	H	Get address of 1st no.
ADD	M	Add 1st no. & 10's compliment 2nd no.
DAA		Decimal Adjustment.
STA	2503 H	Store result in 2503 H.
HLT		Stop the program.

DATA

2501- 96 H.

2502- 38 H.

The result is stored in memory location 2503 H.

RESULT

2503- 58 H.

6. ALP FOR 8-BIT MULTIPLICATION: PRODUCT 16 BIT

Lable	Mnemonics	Operand	Comments
	LHLD	2501 H	Get multiplicand in H-L pair.
	XCHG		Multiplicand in D-E pair.
	LDA	2503 H	Multiplier in accumulator.
	LXI	H, 0000	Initial value of the product =00
			In H-L pair.
	MVI	C, 08	Count = 8 in register C.
LOOP:	DAD	H	shift partial product left by 1 bit.
	RAL		Rotate multiplier left 1 bit. Is multipliers bit=1?
	JNC	AHEAD	No, go to AHEAD.
	DAD	D	Product=product + multiplicand.
AHEAD:	DCR	C	Decrement count.
	JNZ	LOOP	
	SHLD	2504 H	Store result.
	HLT		Stop the program.

DATA

2501-84 H , LSBs of Multiplicand.

2502-00, MSBs of Multiplicand.

2503-56 H, Multiplier.

RESULT

2504-58 H, LSBs of product.

2505-2C H , MSBs of product.

7. ALP FOR 8-BIT DIVISION

Lable	Mnemonics	Operands	Comments
	LHLD	2501 H	Get dividend in H-L pair.
	LDA	2503 H	Get divisor from 2503 H.
	MOV	B, A	Divisor in register B.
	MVI	C, 08	Count = 08 in register C.
LOOP:	DAD	H	Shift dividend and quotient Left by one bit.
	MOV	A, H	Most significant bits of dividend In accumulator.
	SUB	B	Subtract divisor from Most significant bits of dividend.
	JC	AHEAD	Is most significant part of Dividend > divisor ? No, Go to AHEAD.
	MOV	H, A	Most significant bits of dividend In register H.
	INR	L	Yes, add 1 to quotient.
AHEAD:	DCR	C	Decrement count.
	JNZ	LOOP	Is count =0? No,Jump to LOOP.
	SHLD	2504 H	Store quotient in 2504 and Remainder in 2505 H.
	HLT		Store the program.

DATA

2501-9B H, LSBs of dividend.

2502-48 H, MSBs of dividend.

2503-1A H, divisor.

RESULT

2504-F2, Quotient.

2505-07, Remainder.

8. ALP TO SHIFT AN 8-BIT NUMBER LEFT BY ONE BIT

Example. Shift 65 left by one bit.

The binary representation of 65 is given below:

65=0110 0101

(6) (5)

Result of shifting

65 left by one bit =1100 1010=CA

(C) (A)

To shift a number left by one bit the number is added to itself. If 65 is added to 65,the result is CA as shown below.

65=0110 0101

+ 65=0110 0101

1100 1010=CA

The number is placed in memory 2501 H.The result is to be stored in memory 2502 H.

PROGRAM

Memory Address	Machine Codes	Mnemonics	Operands	Comments
2000	3A,01,25	LDA	2501H	Get data in accumulator
2003	87	ADD	A	Shift it left by one bit
2004	32,02,25	STA	2502H	Store result in 2502H
2007	76	HLT		Halt

DATA

2501-65 H

Result

2502-CA H

The instruction LDA 2501 H transfers the number from memory location 2501 H to the accumulator. ADD A adds the contents of the accumulator to itself. The result is twice the number and thus the number is shifted left by one bit. This program does not take carry into account after ADD instruction. If numbers to be handled are likely to produce carry the program may be modified to store it.

9. ALP TO SHIFT A 16-BIT NUMBER LEFT BY ONE BIT

Shift 7596 H left by one bit. 7596=0111 0101 1001 0110

(7) (5) (9) (6)

Result of shifting left by one bit =1110 1011 0010 1100=EB2C

(E) (B) (2) (C)

The number is placed in the memory locations 2501 and 2502 H.

The result is to be stored in the memory locations 2503 and 2504 H.

PROGRAM

Memory Address	Machine Codes	Mnemonics	Operands	Comments
2000	2A,01,25	LHLD H	2501 H	Get data in H-L pair
2003	29	DAD	H	Shift left by one bit
2004	22,03,25	SHLD	2503 H	Store result in 2503 and 2504 H
2007	76	HLT		Stop

Example 1

DATA

2501-96 H, LSBs of the number.

2502-75 H, MSBs of the number.

Result

2502-2C, LSBs of the result.

2504-EB, MSBs of the result.

Example 2

DATA

2501-BF, LSBs of the number.

2502-00, MSBs of the number.

Result

2503-7E, LSBs of the result.

2504-01, MSBs of the result.

The 16-bit number has been placed in two consecutive memory location 2501 and 2502 H. The instruction LHLD 2501 H transfers the 16-bit number from 2501 and 2502 H to H-L pair.DAD H is an instruction for 16-bit addition. It adds the contents of H-L pair to itself. Thus, the 16-bit number is shifted left by one bit. The 16-bit result is stored in the memory locations 2503 and 2504 H by SHLD instruction. In some cases there may be carry after the execution of instruction DAD H. In that case carry may be stored in some register. The program may be modified accordingly.

If the shifting of an 8-bit number gives a result which is more than 8-bits, the problem can be tackled using the technique of shifting 16-bit number in Example 2.

10. ALP TO MASK OFF LEAST SIGNIFICANT 4 BITS OF AN 8-BIT NUMBER

Example .

Number=A6

=1010 0110

(A) (6)

Result=06=0000 0110

(A) (0)

We want to make off the least significant 4 bits of a given number. The LSD of the given number A6 is 6. It is to be cleared(masked off) i.e. it is to be made equal to zero. The MSD of the number A6 is A. In the binary form it is 1010. It is not to be affected. If this number is added with 1111 i.e. F, it will not be affected. Similarly, the LSD of the number is 6. In the binary form it is represented by 0110. If it is added with 0000, it becomes 0000 i.e. it is cleared. Thus, if the number A6 is added with F0, the LSD of the number is masked off.

PROGRAM

Address	Machine Codes	Mnemonics	Operands	Comments
2000	3A,01,25	LDA	2501 H	Get data in accumulator
2003	E6,F0	ANI	F0	Mask off the least significant 4 bits
2005	32,02,25	STA	2502 H	Store result in 2502 H
2008	76	HLT		Stop

DATA

2501-A6

Result

2502-A0

The instruction LDA 2501 H transfers the content of memory location 2501 H i.e. the given number to the accumulator. ANI F0 logically ANDs the content of the accumulator with F0 to clear the least significant 4 bits of the number. STA 2502 H stores the result in memory location 2502 H. HLT stops the program.

11. ALP TO MASK OFF MOST SIGNIFICANT 4 BITS OF AN 8-BIT NUMBER

Example.

Number=A6

=1010 0110

(A) (6)

Result=06=0000 0110

(0) (6)

To mask off 4 most significant bits of a number, 4 MSBs are added with 0000. The least significant bits are not to be affected and therefore, they are added with 1111 i.e. F. Thus, if an 8-bit number is added with 0F, the 4 most significant bits are cleared.

PROGRAM

Address	Machine Codes	Mnemonics	Operands	Comments
2000	3A,01,25	LDA	2501 H	Get data in accumulator
2003	E6,0F	ANI	0F	Mask off the most significant 4 bits
2005	32,02,25	STA	2502 H	Store result in 2502 H
2008	76	HLT		Stop

DATA

2501-A6

Result

2502-06

The instruction LDA 2501 H transfers the contents of memory location 2501 H to the accumulator. ANI 0F logically ANDs the content of the accumulator with 0F to clear the most significant 4 bits of the number. STA 2502 H stores the result in 2502 H. HLT stops the program.

12. ALP to find larger of two numbers

Example1. Find the larger of 98H and 87H.

The first number 98H is placed in the memory location 2501H.

The 2nd number 87H is placed in the memory location 2502H

The result is stored in the memory location 2503H.

The numbers are represented in the form of hexadecimal system.The first number is moved from its memory location to the accumulator.It is compared with the 2nd number.The larger of the two numbers is then placed in the accumulator.From the accumulator the larger number is moved to the desired memory location.

PROGRAM

Memory addr	machine code	labels	memonics	operands	comments
2000	21,01,25		LXI	H,2501H	Address of 1st no in HLpair
2003	7E		MOV	A,M	1st no in accumulator
2004	23		INX	H	Address of 2nd no in HLpair
2005	BE		CMP	M	Compare 2nd no with 1st no
					Is the 2nd no >1st
2006	D2,0A,20		JNC	AHEAD	No,larger number is in
					Accumulator.Goto AHEAD
2009	7E		MOV	A,M	Yes get 2nd number in the
					accumulator
200A	32,03,25	AHEAD	STA	2503H	Store larger number in
					2503H
200D	76		HLT		STOP

DATA:

2501-98H

2502-87H

Result is 98H and it is stored in memory location 2503H

RESULT:

2503-98H

13. ALP to find the largest number in a data array

The no in a series are 98,75,99 as there are three nos in the series the count=03. The count is placed in the memory location 2500H. The nos are placed in the memory location 2501 to 2503H . The result is to be stored in the memory location 2450H.The 1st no of the series is placed in the accumulator and it is compaied with the 2nd no residing in the memory.The larger of two nos is placed in the accumulator .Again this no which is in the accumulator is compared with the third no of the series and the larger no is placed in the accumulator.This process of comparision is repeated till all the nos of the series are compared and the largest nomis stored in the desired memory location.

PROGRAM:

Memory addr	Machine Code	Labels	Mnemonics	Operands	Comments
2000	21,00,25		LXI	H,2500H	Address for count In HL pair
2003	4E		MOV	C,M	Count in register C
2004	23		INX	H	Address of 1st no in HLpair
2005	7E		MOV	A,M	1ST no in accumulator
2006	0D		DCR	C	Decrement count
2007	23	LOOP	INX	H	Address of next number
2008	BE		CMP	M	Compare next number With pervious maximum. Is next no>previous maximum?
2009	D2,0D,20		JNC	AHEAD	No,larger number is in Accumulator go to the Lable AHEAD
200C	7E		MOV	A,M	Yes, get larger in Accumulator
200D	0D	AHEAD	DCR	C	Decrement count

200E	C2,07,20	JNZ	LOOP	
2011	32,50,24	STA	2450H	Store result in 2450H
2014	76	HLT		Stop

EXAMPLE:

DATA:

2500-03

2501-98

2502-75

2503-99

RESULT:

2450-99

14. ALP to find the smaller of two numbers

Find the smaller of 84H and 99H.

The first no 84H is placed in the memory location 2501H.

The 2nd no 99H is placed in the memory location 2502H

Store the result in the memory location 2503H.

The nos are represented in hexadecimal no system.The first no is moved from its memory location to the accumulator.It is compared with the 2nd no.The smaller of the two is then placed in the accumulator.From the accumulator the smaller no is moved to the desired memory location where it is to be stored.

PROGRAM

Memory addr	machine code	labels	memonics	operands	comments
2000	21,01,25		LXI	H,2501H	Address of 1st no in HLpair
2003	7E		MOV	A,M	1st no in accumulator
2004	23		INX	H	Address of 2nd no in HLpair
2005	BE		CMP	M	Compare 2nd no with 1st no

					Is the 2nd no >1st
2006	D2,0A,20		JC	AHEAD	No,larger number is in Accumulator.Goto AHEAD
2009	7E		MOV	A,M	Yes get 2nd number in the accumulator
200A	32,03,25	AHEAD	STA	2503H	Store larger number in 2503H
200D	76		HLT		STOP

Example- 1

DATA: 2501-98H

2502-87H

Result is 87H and it is stored in memory location 2503H

15. ALP to find the smallest number in a data array

The number of a series are 8658 and 75.

As there are three numbers in the series,count=03

The count is placed in the memory location 2500H

The numbers are placed in the memory location 2501 to 2503H

The result is to be stored in the memory location 2450H

The 1st no of the series is placed in the accumulator and it is compaired with the 2nd no residing in the memory.The Smaller of two nos is placed in the accumulator .Again this no which is in the accumulator is compared with the third no of the series and the Smaller no is placed in the accumulator.This process of comparision is repeated till all the nos of the series are compared and the Smallest number is stored in the desired memory location.

PROGRAM:

Memory addr	Machine Code	Labels	Mnemonics	Operands	Comments
2000	21,00,25		LXI	H,2500H	Address for count In HL pair
2003	4E		MOV	C,M	Count in register C
2004	23		INX	H	Address of 1st no in HLpair
2005	7E		MOV	A,M	1ST no in accumulator
2006	0D		DCR	C	Decrement count
2007	23	LOOP	INX	H	Address of next number
2008	BE		CMP	M	Compare next number With pervious maximum.
2009	D2,0D,20		JC	AHEAD	Is next no>previous maximum? No,larger number is in Accumulator go to the Lable AHEAD
200C	7E		MOV	A,M	Yes, get larger in Accumulator
200D	0D	AHEAD	DCR	C	Decrement count
200E	C2,07,20		JNZ	LOOP	
2011	32,50,24		STA	2450H	Store result in 2450H
2014	76		HLT		Stop

EXAMPLE:

DATA:

2500-03H

2501-86H

2502-58H

2503-75H

RESULT:

2450-58H

TIMING DIAGRAM FOR I/O READ AND MEMORY READ

I/O READ:

1. In I/O read cycle the microprocessor reads the data available at an input port/input device. The data is placed In accumulator.

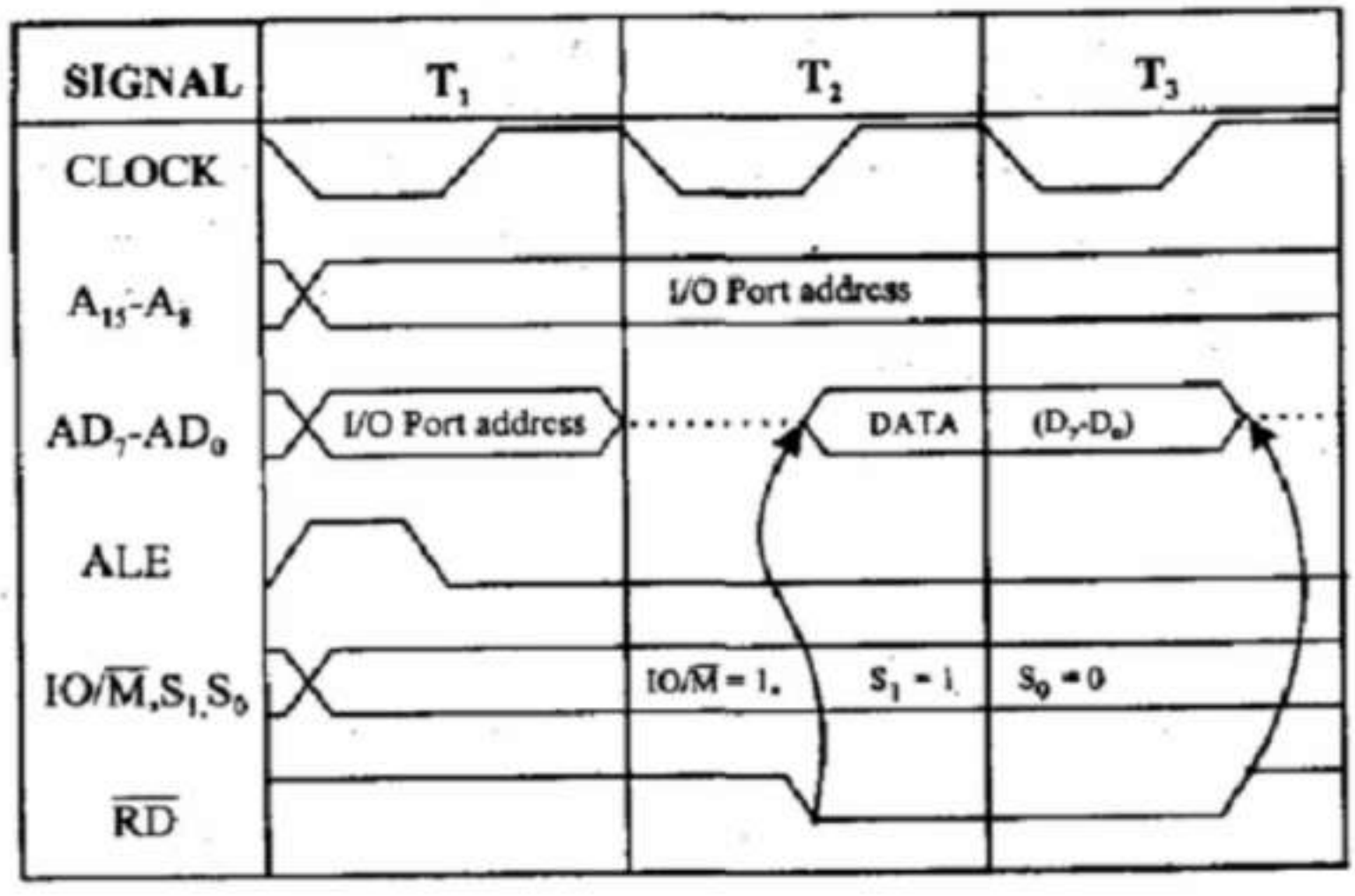
2.An I/O read cycle it is similar to memory read cycle except that signal I/O/M(bar) goes high in case of I/O read.

3.It indicates that the address on the address bus is for an input device.

4.It case of I/O device or I/O port the address is only 8 bit long. So the address of I/O device is duplicated on both address and address data bus.

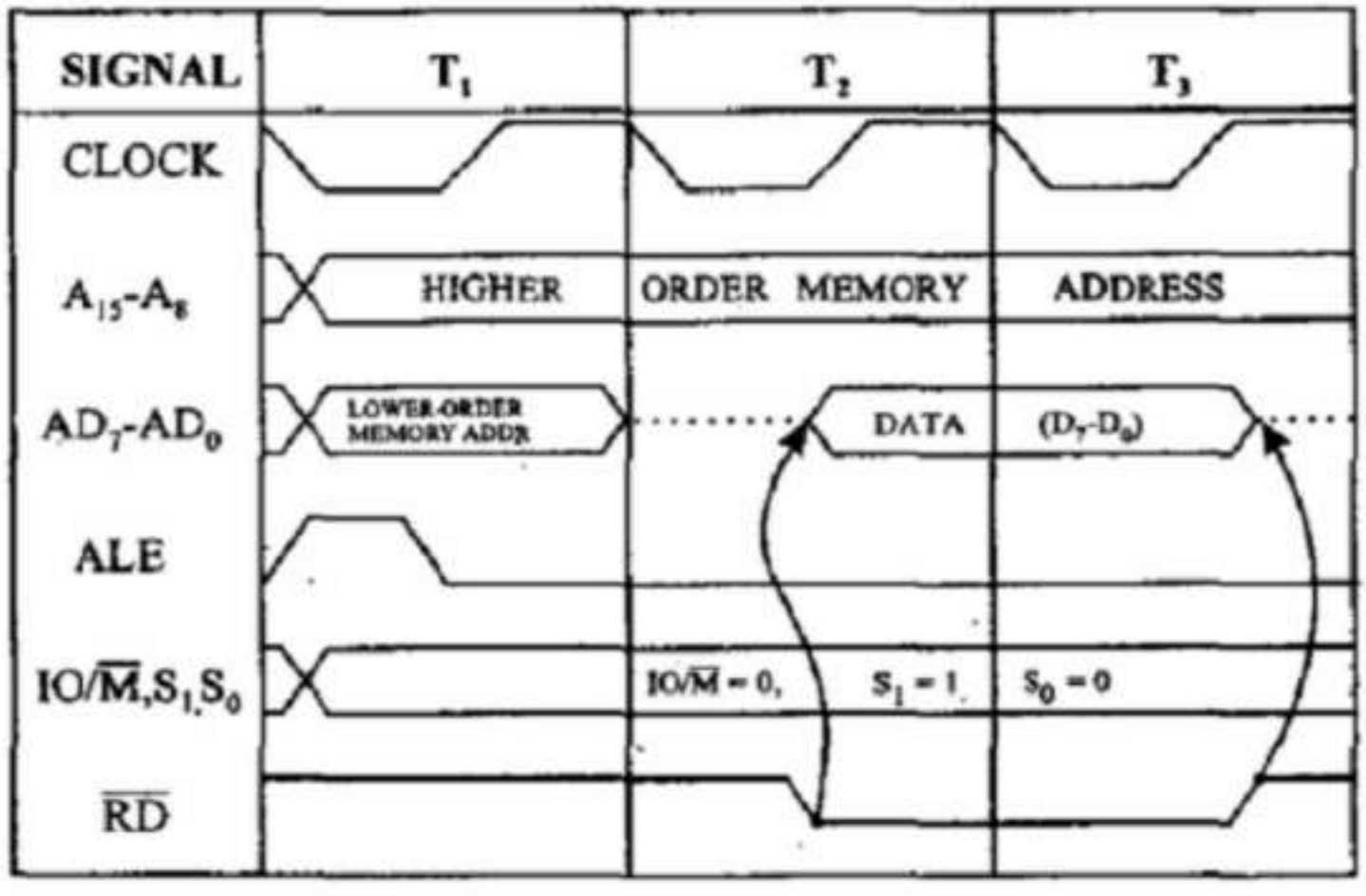
Ex-IN instruction is an example of I/O READ.

5.It is 2bytes long. So it requires 3 machine cycles such as opcode fetch cycle, memory read cycle to read the input port address, I/O read cycle to read data from input port



TIMING DIAGRAM FOR MEMORY READ CYCLE

1. In memory read cycle the microprocessor read the content of memory location
2. The content is the placed in either accumulator any other CPU register.
   Ex-MVI A, 05H
3. In this cycle io/m(bar) goes low indicate that the address is for memory.
4. S1 and S0 are set to be 1 and 0 respectively for read operation.
5. Address lines A8 to A15 carry 8MSBs of memory address of data.
6. Address lines AD0 to AD7 carry 8LSBs of memory address of data.
7. During T2 8LSBs of address is latched AD0 –AD7 are made free for data transfer.
8. RD(bar) goes low in T2 enable the memory read operation.
9. Now data is placed in data bus during T3 data enters the cpu.
   Ex-LXI H,2000H



TIMING DIAGRAM FOR MEMORY WRITE CYCLE-

1. In a memory write cycle the CPU sends data from accumulator or any other register to memory.

2. The status signal S_0 and S_1 are 1 and 0 respectively for write operation.

3. goes low in T_2 indicating that the write operation to be performed.

4. During T_2 the address/data bus is not disable but the data to be sent out to memory placed on the address/data bus.

5. As soon as goes high in T_3 the write operation is terminated.

Example: - 1. MOV M, A

2. STA 2000H

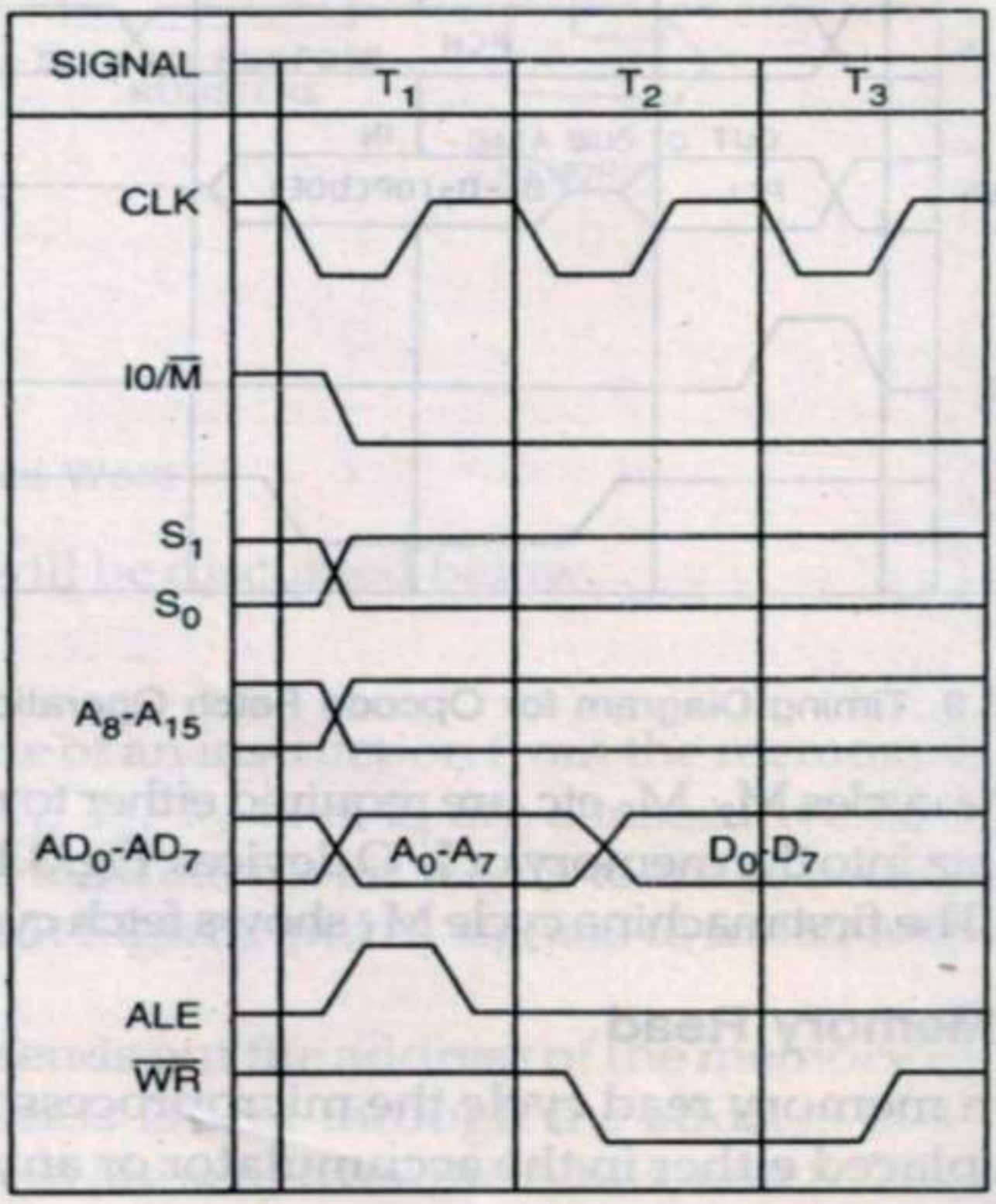


Fig. 3.12. Timing Diagram for Memory Write Operation

TIMING DIAGRAM FOR I/O WRITE CYCLE-

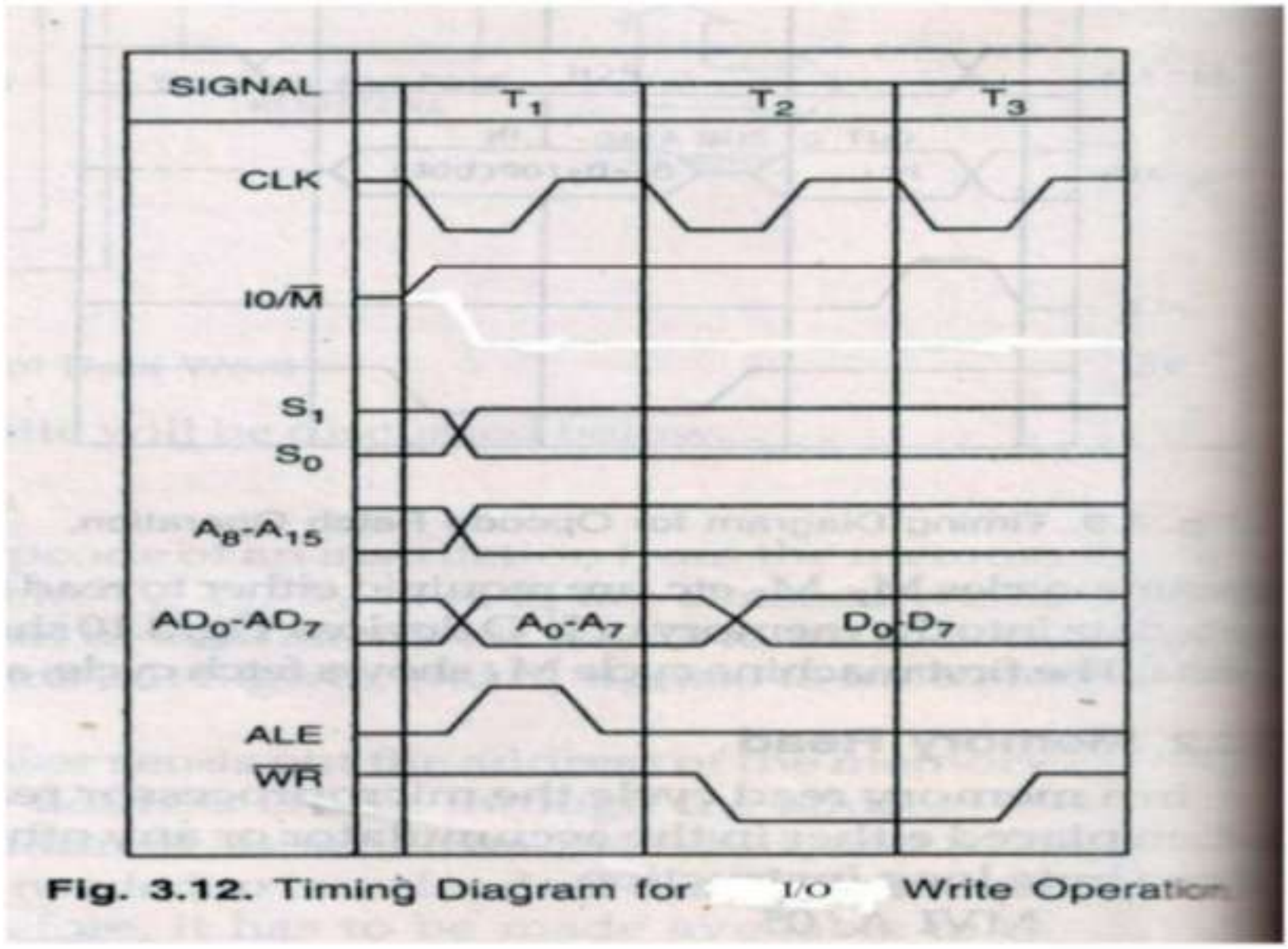


Fig. 3.12. Timing Diagram for I/O Write Operation

1. In an I/O write cycle the CPU sends data to an I/O port or an I/O device from the accumulator.

2. It is similar to a memory write cycle except that IO/$\overline{M}$ goes high indicating that the address sent by CPU is for I/O device or I/O port.

3. The address of an I/O port (8-bit) is duplicated on both address and address/data bus.

4. The OUT instruction is used for I/O write operation. It is a 2 byte instruction and required 3 machine cycle.MC1 is the opcode fetch cycle.MC2 is a memory read cycle for reading the I/O device address from memory.MC3 is an I/O write cycle for saving accumulator data to the I/O device or I/O port.

INTERRUPTS AND INTERRUPT SERVICE SUBROUTINE (ISS):-

HARDWARE INTERRUPTS AND SOFTWARE INTERRUPTS-

Interrupts caused by I/O devices to transfer data to or from the microprocessor are called Hardware Interrupts.

Example- TRAP, RST 7.5, RST 6.5, RST 5.5

When an interrupt is recognized the next instruction is executed from a fixed location in memory as given below

INTERRUPTS	ISS ADDRESS
TRAP	0024
RST 5.5	002C
RST 6.5	0034
RST 7.5	003C

The normal operation of the microprocessor can also be interrupted by abnormal internal conditions on special instruction. Such interrupts are called Software Interrupts.

Example-RST 0 to RST 7 instructions of 8085.

They are used in debugging of a program.

INSTRUCTION	OPCODE	ISS ADDRESS
RST 0	C7	0000
RST 1	CF	0008
RST 2	D7	0010
RST 3	DF	0018
RST 4	E7	0020
RST 5	EF	0028
RST 6	F7	0030
RST 7	FF	0038

The internal abnormal or unusual conditions which prevent the normal processing sequence of a microprocessor are called Exception.

Example-Divide by zero is an exception.

VECTOR INTERRUPTS AND NON VECTOR INTERRUPTS-

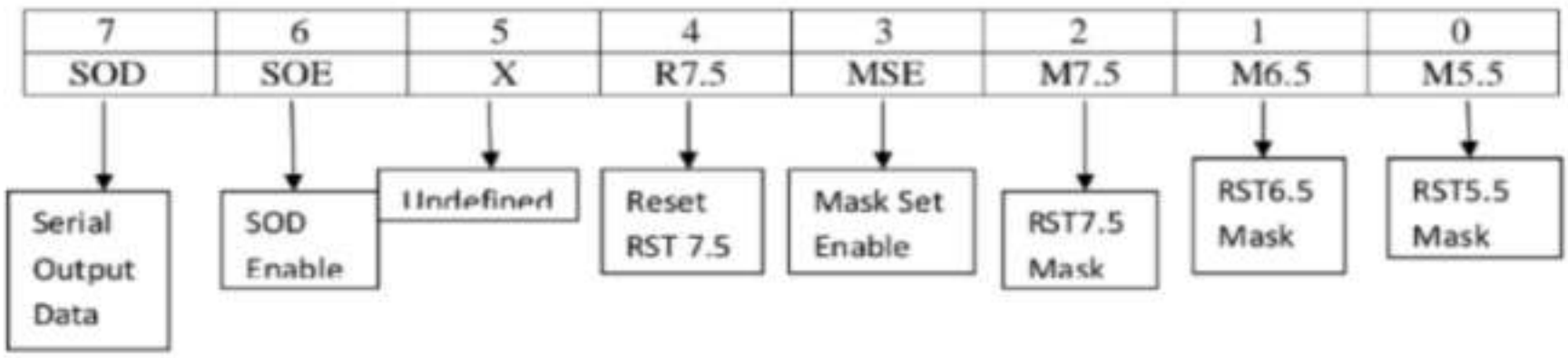
For TRAP, RST 7.5, RST 6.5, RST 5.5 interrupts, the program is automatically transferred to the ISS address without any external hardware .These interrupts for which hardware automatically transfers the program to a specific memory location is known as Vector Interrupt.

When several I/O devices are connected to INTR interrupts line an external hardware is used to interface I/O devices. This circuit generates RST-n codes to implement the multiple interrupts schemes. These are not vector interrupts.

RST 7.5, RST 6.5, RST 5.5-

RST 7.5, RST 6.5, RST 5.5 are maskable interrupts. They are enable by software using instruction EI and SIM. SIM instruction enables or disable according to the bit of accumulator.

ACCUMULATOR CONTENT BEFORE EXECUTION OF SIM-



RST 5.5 mask, RST 6.5 mask, RST 7.5 mask } bit=1, it is masked off / bit=0, it is enable

Mask Set Enable (MSE) should be set to 1 to make bits 0 to 2 effective.

Q- Write a set of instruction to enable all the interrupts.

Ans-	Label	Mnemonics	Operands	Comments
		EI		Enable interrupts
		MVI	A, O8H	Get accumulator bit pattem to enable RST 7.5, 6.5, 5.5
		SIM		Enable RST 7.5, 6.5, 5.5

Q-Write a set of instruction to enable RST 6.5 and disable RST7.5, RST 5.5.

Ans-	Label	Mnemonics	Operands	Comments
		EI		Enable interrupts
		MVI	A, 1DH	Accumulator bit pattern to enable RST 6.5 and masked Off RST 7.5 and 5.5
		SIM		Enable RST 6.5 and disable RST 7.5 and 5.5

PENDING INTERRUPTS:

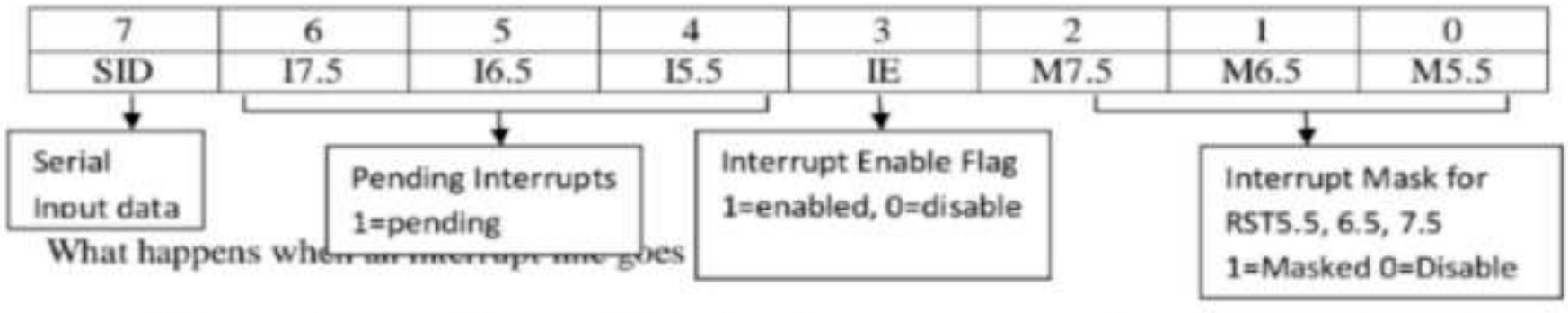
When one interrupt is being served other interrupts may occur resulting in a pending request. When more than one interrupt occur simultaneously the interrupt with higher priority is served and interrupts with lower priority remain pending.

The 8085 has an instruction RIM using which the programmer can know the current status of the pending maskable interrupts.

In case of INTR, the processor needs its status in the last state of the last machine cycle of the instruction.

The bit pattern of the accumulator after execution of RIM instruction is given by;

Bits 0-2 are for interrupt masks.
Bit 3 enable interrupt flag
Bits 4-6 indicate pending interrupts
Bit 7 is SID if any



What happens when an interrupt line goes

1. When an interrupt line goes high, the microprocessor completes its current instruction and saves program counter on the stack.
2. It also resets Interrupt Enable (IE) flip flop before taking up the ISS (Interrupt Service Subroutine) so that the occurrence of further interrupts by other devices is prevented during the execution of ISS.

ADDRESS SPACE PARTITIONING

Intel 8085 uses a 16-bit wide address bus or addressing memory and I/O devices. It can access 2^{16}=64k bytes of memory and I/O devices. There are two schemes for the allocation of address to memories or I/O devices.

1. Memory mapped I/O scheme
2. I/O mapped I/O scheme

1. Memory Mapped I/O Scheme

In this scheme there is only one address space. Address space is defined as set of all possible addresses that a microprocessor can generate. Some address are assigned to memories and some address to I/O devices. Suppose memory locations are assigned the address 2000-2500. One address is assigned to each memory location. These addresses cannot be assigned to I/O devices. The addresses assigned to I/O devices are different from address assigned to memory. For example, 2500, 2501, 2502 etc. may be assigned to I/O devices. One address is assigned to each I/O device.

In this scheme all the data transfer instruction of the microprocessor can be used for both memory as well as I/O devices. For example, MOV A, M will be valid for data transfer from the memory location or I/O device whose address is in H-L pair. This scheme is suitable for small system.

2. I/O Mapped I/O Scheme

In this scheme the address are assigned to memory locations can also be assigned to I/O devices. To distinguish whether the address on an address bus is for memory location or I/O devices. The Intel 8085 issues IO/M…… signal for this purpose. When the signal is high the address of an address bus is for I/O device. When low, the address is for a memory location. Two extra instructions IN and OUT are used to address I/O device. The IN instruction is used to read data from an input device. And OUT instruction is used to an output device. This scheme is suitable for large system.

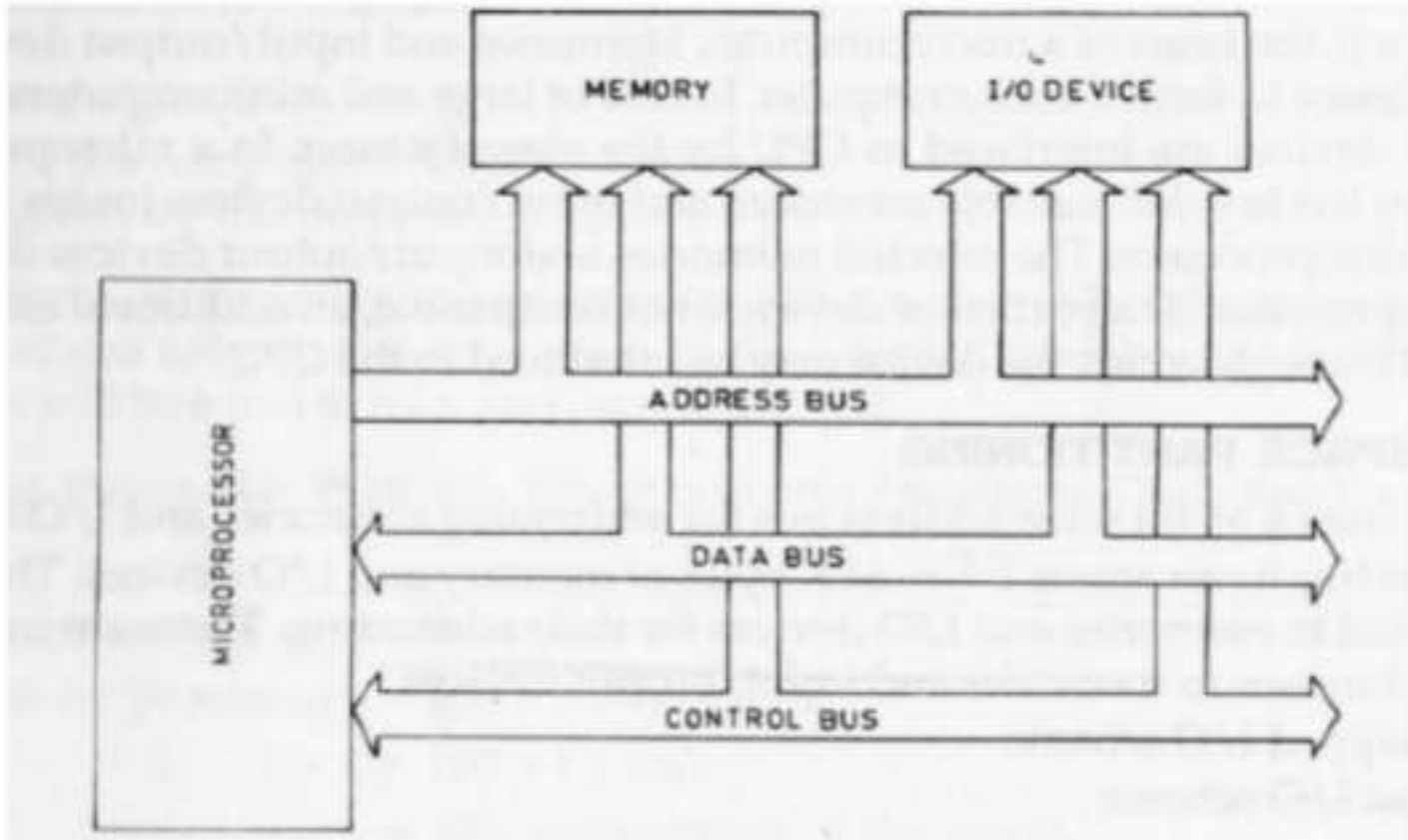


Fig. 7.1 Schematic Diagram for Memory and I/O Interfacing.

Memory Interfacing

An address decoding circuit is employed to select the required I/O device or a memory chip. When IO/M……. is high, decoder is to active and the require IO device is selected. If IO/M….. is low, the decoder1 is activated the required memory chip is selected. A few msb of address line is applied to the decoder to select the memory chip or an I/O device.

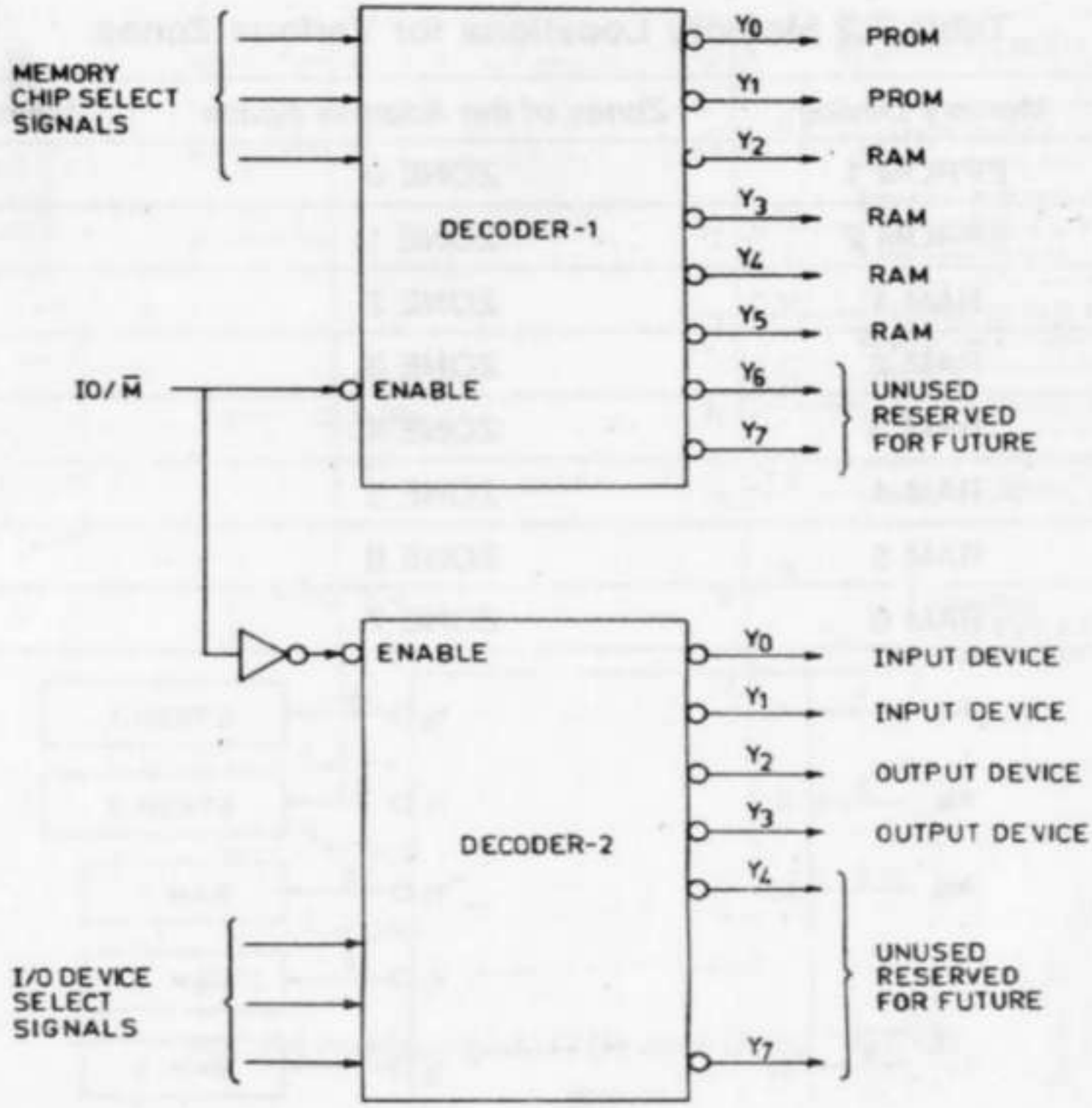


Fig. 7.2 Interfacing of Memory and I/O Devices.

A_{13}	A_{14}	A_{15}	OUTPUT
0	0	0	Y_0
0	0	1	Y_1
0	1	0	Y_2
0	1	1	Y_3
1	0	0	Y_4
1	0	1	Y_5
1	1	0	Y_6
1	1	1	Y_7

MEMORY CLASSIFICATION

The memory is used to store information used by the CPU. The information may be in the form of program, that the CPU executes or data on which the CPU operates. Memory can be classified into two groups.

1- Primary Memory
2- Secondary Memory

Primary memory is constituted by memory blocks within the address space of the processor. They are implemented by using read only memory(ROM) which are not volatile memories and read write memories or random access memory(RAM) which are volatile memories. ROMs are used to store the permanent programs and data while RAMs are used to store intermediate results and data. Secondary memories such as magnetic tapes, floppy disk, hard disk etc are used to store large amounts of data. The CPU cannot directly access secondary units.

1-Primary Memory

It is that part of the memory which can be directly access by the CPU. It can be viewed as a stack of words, each word be associated with a unique address. These words may be instruction or data. A CPU having n address line can access 2^n memory location. The total primary memory area is partition into two separate sections called the ROM area and the read/write area.

ROM Area

On power on or a reset, the CPU always starts executing program from a fixed default address which is usually the first address of the address space. These instructions are usually initialize the sequences which direct the CPU to initialize all sub system in the systems. For example CPU may initialize the display driver display outputs in a particular manner or initialize a key-board to accept the certain commands. This initialization sequence which has to perform each time computer begins to its operation is part of the monitor software which is stored in the permanent or non-volatile memory. The words permanent or non-volatile implies that the program is not lost when the power is switched off and that it is available each time the power is switched on. It is implemented by using a special kind of memory called read only memory or ROM. The data in this selection cannot be written over and can only the read. The ROM is used to store information that should not change. ROMs are available into 4 types. There are

i. Masked ROM
ii. PROM
iii. EPROM
iv. EEPROM

i-Masked ROM

The instructions in such ROMs are permanently installed by the manufacturer as for the specification provided by the system programmer and cannot be altered. This ROM contains call arrays in which 1s and 0s …….. are stored by means of a metallization interconnect most …… at the time of fabrication.

ii-PROM (Programmable Read Only Memory)

The manufacturer provides a memory device which can be programmed by the user by using a PROM program. The PROM uses a fusible links that can be burnt or melted by special PROM burning circuit. A fused link is corresponds to zero.

iii-EPROM (Erasable PROM)

It uses most charge storage technology. It is also programmable by the user. The information stored in the EPROM can be erased by exposing the memory to ultraviolet light which erase the data stored the data in total memory area. Then the memory can be reprogrammable by the user by using EPROM burning circuit.

iv-EEPROM (Electrically Erasable PROM)

This is similar to EPROM except that the erasing done by electrical signal instead of ultraviolet light and that the data in memory location can be selective erased.

Read/Write Area

It is a special kind of memory area where information can be written into or read from whenever necessary. The CPU uses this section memory as a scratch pad memory. It is also called read/write memory or RAM. The CPU can access any memory location by specifying its address. Unlike ROM the conventional is a volatile memory i.e. the contents of the RAM are used when the power is switched off. RAMs are two types.

i-Static RAM

In the case of static RAM once the data is written into a memory location, the data remain unchanged unless on same memory location is written into again. It uses flip-flops for storage elements.

ii-Dynamic RAM

In case of dynamic RAM the basic storage elements is a capacitor. This element contains a 1or a 0 depending on the presence or absence of charge. Unlike static RAM, the contents of the dynamic RAM may change with time due to leakage of charge. So it is necessary to periodically refresh the storage element in a dynamic RAM. Here it has external refresh circuitry. The advantages of dynamic RAM over static RAM are

... It consumes less power than static RAM.

... It has about 5 times more storage element per unit area.

Disadvantages are that

... DRAMS have slower access times and need special circuitry to periodically refresh memory.

i-RAMs:

RAMs are also implemented using other technologies like i-RAM (integrated RAM) which are dynamic RAM devices in which the memory refresh circuitry is implemented within the device.

Bubble Memories

It is a type of DRAM. Here the contents are not lost when the power is switched off.

2-Secondary Memory

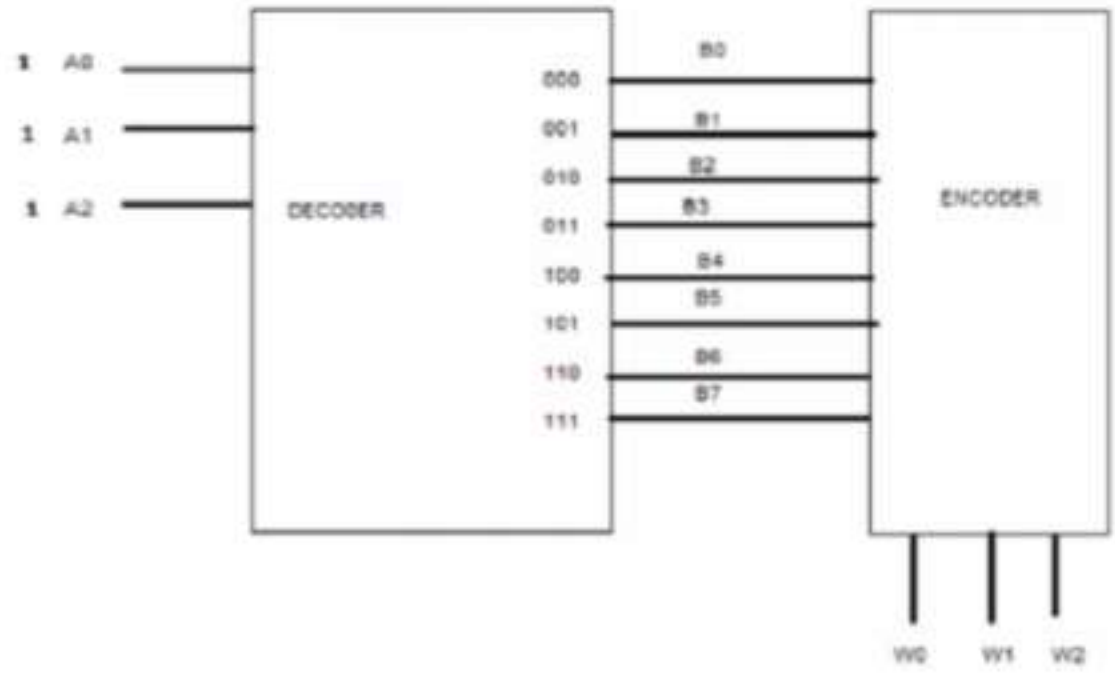
The maximum capacity of primary memory 2^n location. Each of 2^n location where n is the number of CPU address lines. Sometimes it is necessary to handle more data than allowed by the primary memory. In such cases secondary memory is used. The CPU cannot directly

access memory but can access through I/O ports. Examples are magnetic tapes, hard disk, and floppy disk.

MEMORY STRUCTURE:

ROM MODEL:

A read only memory (ROM) is an encoder to select a word in an encoder, only one of the inputs must be made active. But to save lines, the CPU directly puts out the address of the word it wants to access. So it is necessary to insert a device between the address put out by the CPU and the ROM inputs which enables a unique input to the ROM. This device must be a decoder.



(Decoded ROM of capacity 8 bytes)

This is an example with an address bus of n=3 bits and a memory of 8 bytes (2^3=8 words).

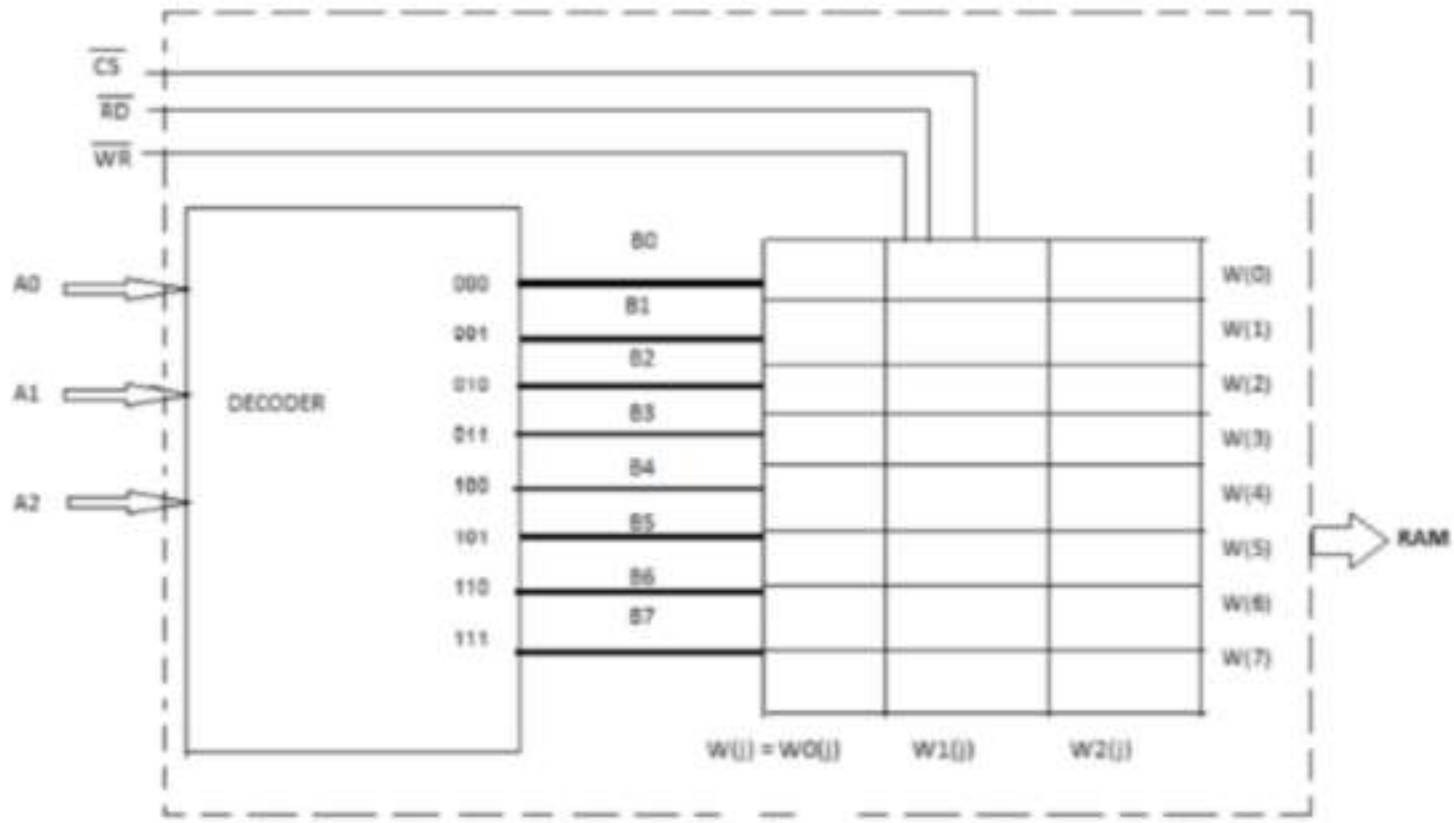
The CPU puts out the address 111 on the address lines $A_2A_1A_0$ to access the 8^{th} word B_7. B_7 is made active by the decoder and the word $W(7) = W_2(7)\ W_1(7)\ W_0(7)$ is put on the data bus.

RAM or READ/WRITE MODEL:

The internal organization of a random access memory (RAM) is similar to that of ROM. But RAM has input which is made active when data is to be written into RAM.

The figure shows a 3 bit = 8 words RAM.

One of the outputs of the decoder is made active depending on the address which is input to it. This active output is fed to the encoder to generate the output W, which is directly connected to the common data bus of the system.



(RAM or R/W MODEL)

MEMORY INTERFACING & ADDRESSING DECODING:

This is done by using 74LS138 which is a 3-8 line decoder.

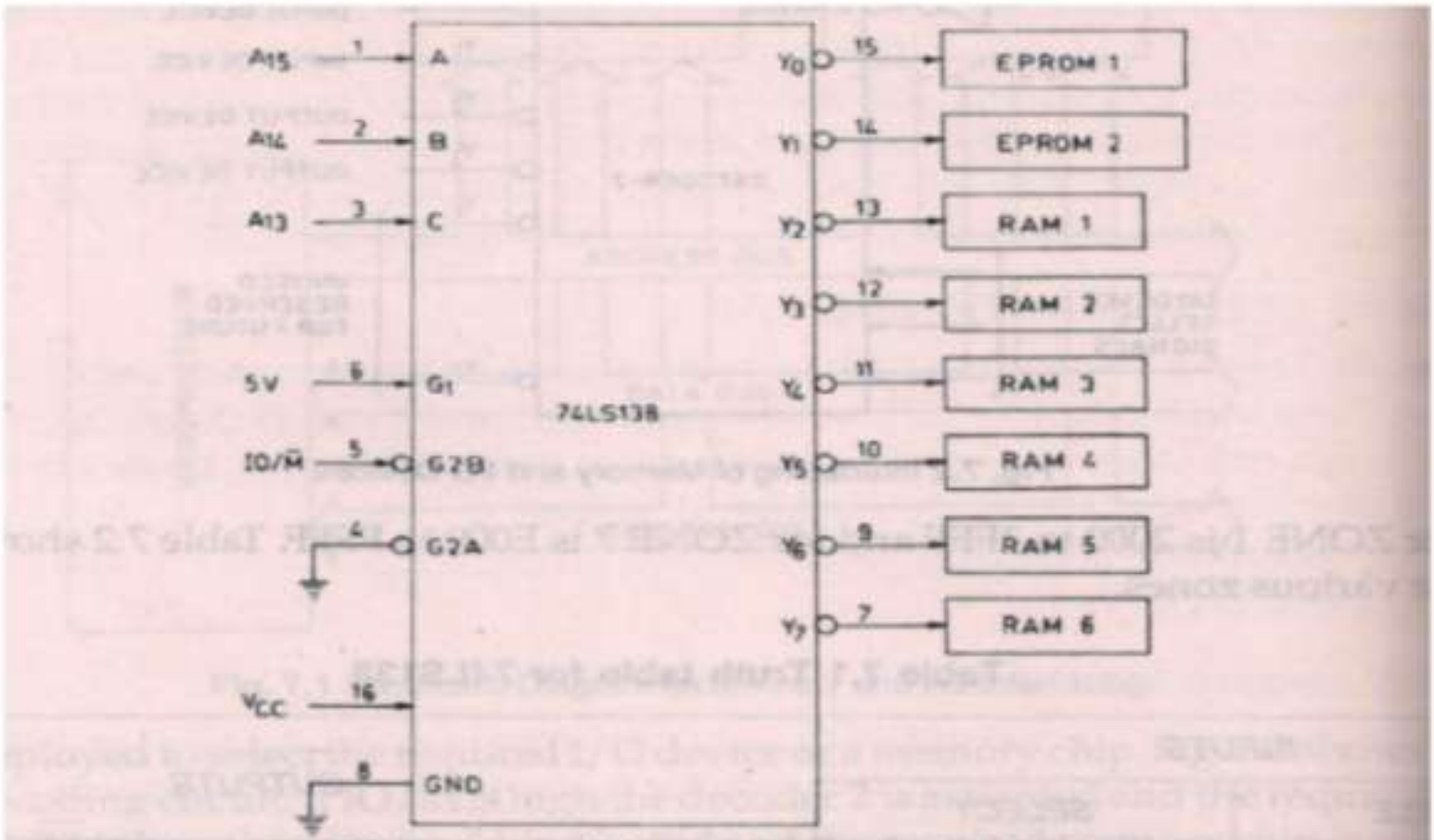


Fig. 7.3 Interfacing of Memory Chips using 74LS138.

(Memory interfacing using 74LS138)

The address of the memory location is sent out by the microprocessor. The corresponding memory chip is selected by a decoding circuit.

G1, G2A, G2B are enable signals of the decoder. To enable 74LS138 decoder, G1 should be high and G2A, G2B should be low. A, B, C are the select lines of the decoder. Y_0-Y_7 are the 8 output lines.

By applying proper logic to select lines, any one of the outputs can be selected. The selected output line goes low while other output lines remain high.

The entire memory address = 64 KB for 8085 has been divided into 8 zones. The address lines A_{13} A_{14} A_{15} have been applied to A, B, C select lines. The logic applied to these select lines A, B, C selects a particular memory device i.e. an EPROM or RAM. The address lines A_0 to A_{12} decide the address of the memory location within a selected memory chip.

IO/ is connected to G2B, G1 is connected to +5volt dc supply and G2A is grounded.

Addresses of 8 zones:

Decoder output	Memory device	Zones of the address spaces	Memory location address
Y_0	EPROM1	ZONE 0	0000 to 1FFF
Y_1	EPROM2	ZONE 1	2000 to 3FFF
Y_2	RAM1	ZONE 2	4000 to 5FFF
Y_3	RAM2	ZONE 3	6000 to 7FFF
Y_4	RAM3	ZONE 4	8000 to 9FFF
Y_5	RAM4	ZONE 5	A000 to BFFF
Y_6	RAM5	ZONE 6	C000 to DFFF
Y_7	RAM6	ZONE 7	E000 to FFFF

Total memory = 0000 to FFFF = 64 KB

Memory of each zone = 64/8 KB = 8KB

8085A Minimum System Microcomputer

An 8085 a based minimum system microcomputer needs the following components

- CPU
- MEMORY
- IO PORTS

*The CPU co-ordinates the activities job all the components on a microcomputer.

* On a reseting or after a power on the CPU executes the programs stored in the permanent memory starting from the first address in the address space.

* The data for program execution can come from memory or from IO ports.

* A scratchpad memory in the RAM area is essential to store intermediate data and also to function as a stack.

*The IO ports are essential for the CPU to collect data from an environment and sends out appropriate signals to control a processing.

*In order to avoid extra circuit tree and keep the number of components to be minimum intel corporation introduced two special chips.

- 8155/8156 chip
- 8355/8755 chip

Both of these are on chip address de-multiplexing circuits.

8155

*The 8155 is a 40 pin chip having 8 bit 256 word RAM memory

*Two programmable 8 bit IO ports i.e. port A ,port B

*one programmable 6 bit IO port i.e. port C

*one programmable 14 bit binary timer/counter

*An internal address latch

8156

8156 is functionally compatible with 8155except that it uses an active high chip enable signals while 8155 uses the active low chip enable.

8355

The 8355 has a 2 KB ROM memory and 8 bit parallel ports i.e. port A and port B.

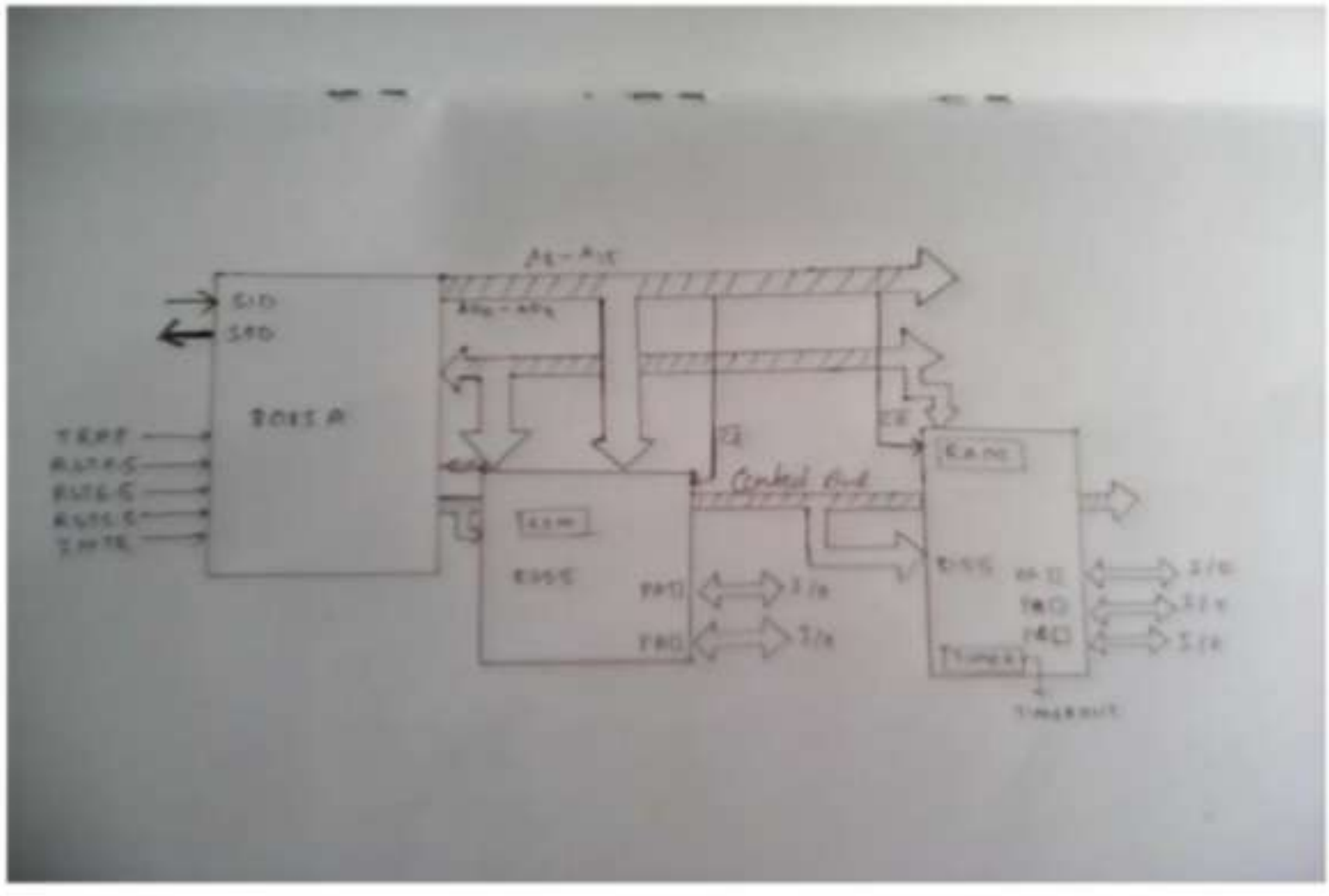
8755

The 8755 has 2 KB of EPROM memory instead of ROM memory and consist of two 8 bit parallel ports i.e. port A and port B.

* 8155/8156 is used as RAM memory.

* 8355/8755 is used as ROM memory.

*8085A is used as CPU.



SCHEMATIC DIAGRAM OF 8085A MINIMUM SYSTEM MICROCOMPUTER

*RST5.5 , RST 6.5 and RST 7.5 are maskable interrupt.

* They are enable by software using instruction EI and SIM.

* SIM instruction enable or disable interrupts according to the bit pattern of the accumulator.

ABSOLUTE VS PARTIAL DECODING

Absolute Decoding	**Partial Decoding**
*When the output port is selected by the decoding all of the 8 address lines it is called absolute decoding.	*When the output port is selected by decoding some of the address lines it is called partial decoding.
*It is good design practice but it is costly.	*It is less costly.

*The output device selected as a unique address.	*The output device selected as multiple addresses.
*It is used in large systems.	*It is used in small systems.

DATA TRANSFER GROUP

Instructions which are used to transfer data from one register to another register, from memory to register or from register to memory they come under this group.

For e.g- MOV r1, r2 – move the content of the one register to another. The content of register r2 is moved to register r1. For example , the instruction MOV A,B moves the content of register B to register A. The instruction MOV B,A moves the content of register A to register B . The time for the execution of this instruction is 4 clock period. One clock period is called state. No flag is affected.

1. MOV r,M (Move the content of the memory to register).

The content of the memory location ,whose address is in HL pair, is moved to register r.

e.g- LXI H,2000H	load HL pair by 2000H
MOV B,M	Move the content of the memory location 2000H to register B.
HLT	Halt.

2. MOV M,r (Move the content of register to memory).

The content of register r is moved to the memory location addressed by HL pair.

3. MVI r,data (move immediate data to register .)

The 1st byte of the instruction is its opcode. The 2nd byte of the instruction is the data which is moved to register r. For example , the instruction MVI A,05 Moves 05 to register A. In the code form it is written as 3E,05. The opcode for MVI A is 3E and 05 is the data which is to be moved to register A.

4. MVI M,data (move immediate data to memory.)

The data is moved to memory location whose address is in HL pair. for example

LXI H,2400H	load HL pair with 2400H
MVI M,08	Move 08 to the memory location 2400H
HLT	Halt

5. LXI rp,data 16- This instruction loads 16 bit data immediate data into register pair rp. This instruction is for register pair only high order register pair is mentioned after the instruction . for example H in the instruction LXI H ,2500H into HL pair. H with 2500H denotes that the data 2500H denotes that the data 2500 is in hexadecimal. In the code form it is written as 21,00,25. The 1st byte of the instruction 21 is the opcode for LXI H. The 2nd byte 00 is 8 LSBs of the data and it is loaded into the register L. The 3rd byte 25 is 8 MSBs of the data and it is loaded into register H.

6. LDA addr (load accumulator direct).-The content of the memory location, whose address is specified by the 2nd and 3rd bytes of the instruction is loaded into the accumulator. The instruction LDA 2400H will load the content of the memory location 2400H into the accumulator . The 2nd byte 00 is of 8 LSBs of the memory address . The 3rd byte 24 is 8 MSBs of the memory address .

7. STA addr (store accumulator direct).-The content of the accumulator is stored in the memory location whose address is specified by the 2nd and 3rd byte of the instruction. STA 2000H will store the content of the accumulator in the memory location 2000H.

8. LHLD addr (Load HL Pair direct)- The content of the memory location , whose address is specified by the 2nd and 3rd bytes of the instruction ,is loaded into the register L. The content of the next memory location is loaded into the register H.

9. SHLD addr (store HLpair direct) –The content of the register L is stored in the memory location whose address is specified by the 2nd and 3rd bytes of the instruction. The content of the register H is stored in the next memory location 2500H . The content of register H is stored in the memory location 2501H.

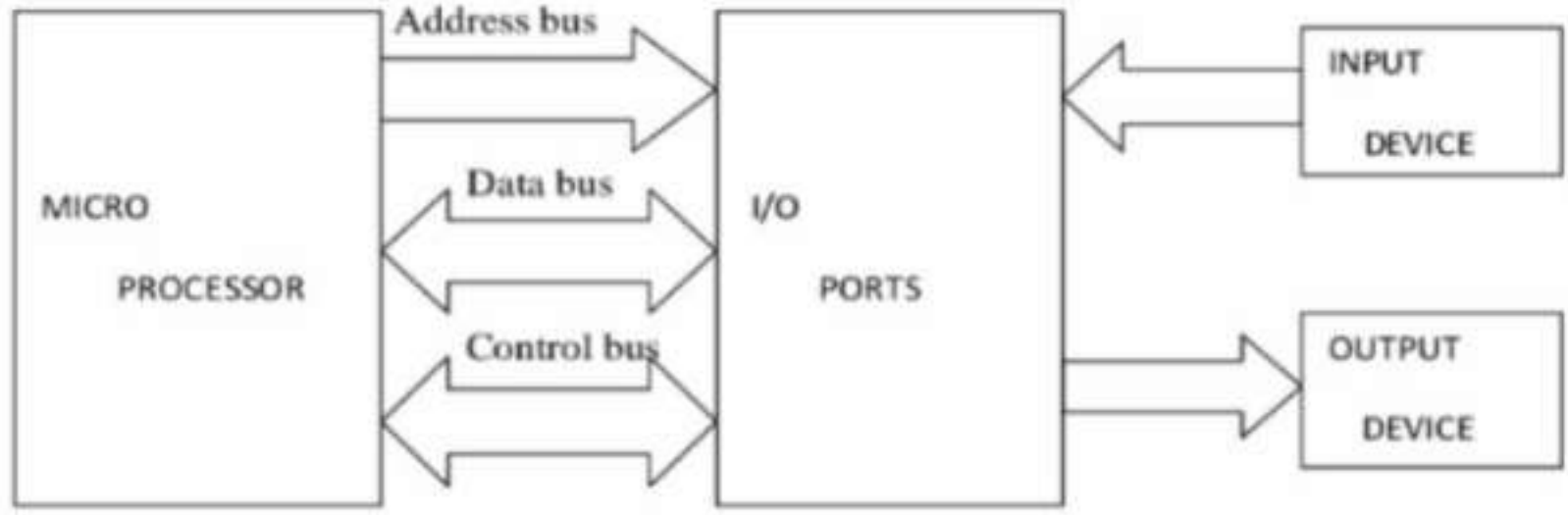
10. LDAX rp (LOAD accumulator direct) –The content of the memory location ,whose address is in the register pair rp is loaded into the accumulator.

11. STAX rp (store the accumulator direct)- The content of the memory location is stored in the memory location whose address is in the register pair rp .

12. XCHG (Exchange the contents of HL with DE pair)-The contents of HL pair are exchanged with contents of DE pair.

I/O PORTS – 8212 :-

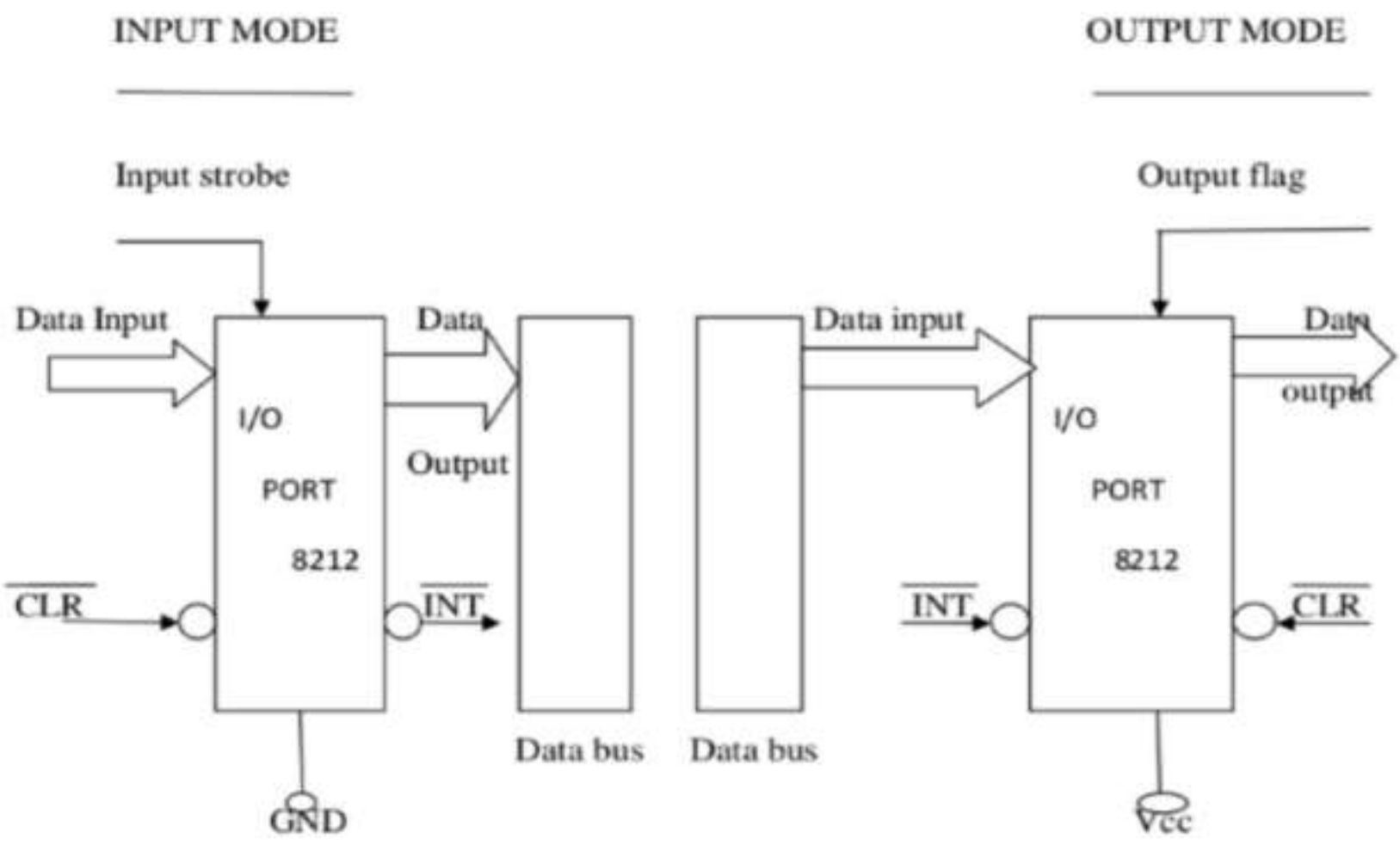
- An input device is connected to the micro processor through an input port.
- An input device is a place for unloading data. An input device unloads data into the port, & the micro processor reads data from the input port.
- Similarly, an output device is connected to the micro processor through an output port.



INTERFACING OF I/O DEVICE THROUGH I/O PORT

- An I/O port may be programmable or non-programmable.
- A non-programmable I/O port behaves as an input port, if it has been designed and connected in output mode. A port connected in output mode acts as an output port.
- But a programmable I/O port can be programmed to act either as an input port or an I/O port.

The Intel 8212 is an 8-bit non-programmable I/O port. It can be connected to the micro processor either as an input port or as an output port. If we required one input port & one output port, two units of 8212 will be required .one of them will be connected in input mode & the other in output mode.



8255A (PPI)

PPI-Programmable Peripheral Interface:

A PPI is a multiport device. The port may be programmed in a verity of ways as required by the programmer. The device is very useful for interfacing peripheral devices.

It has three, 8-bit port.

- Port A
- Port B
- Port C

Port C is further subdivided into two, 4-bit ports.

- Port C upper
- Port C lower

Each port can be programmed either as an input port or as an output port,by setting proper bits in the control word.

This control word is written into a control word register (CWR).

CONTROL GROUPS OF 8255:

The 24 lines of I/O port is divided into 2 groups.

- Group A
- Group B

Each group contains one 8-bit port & a 4-bit port.

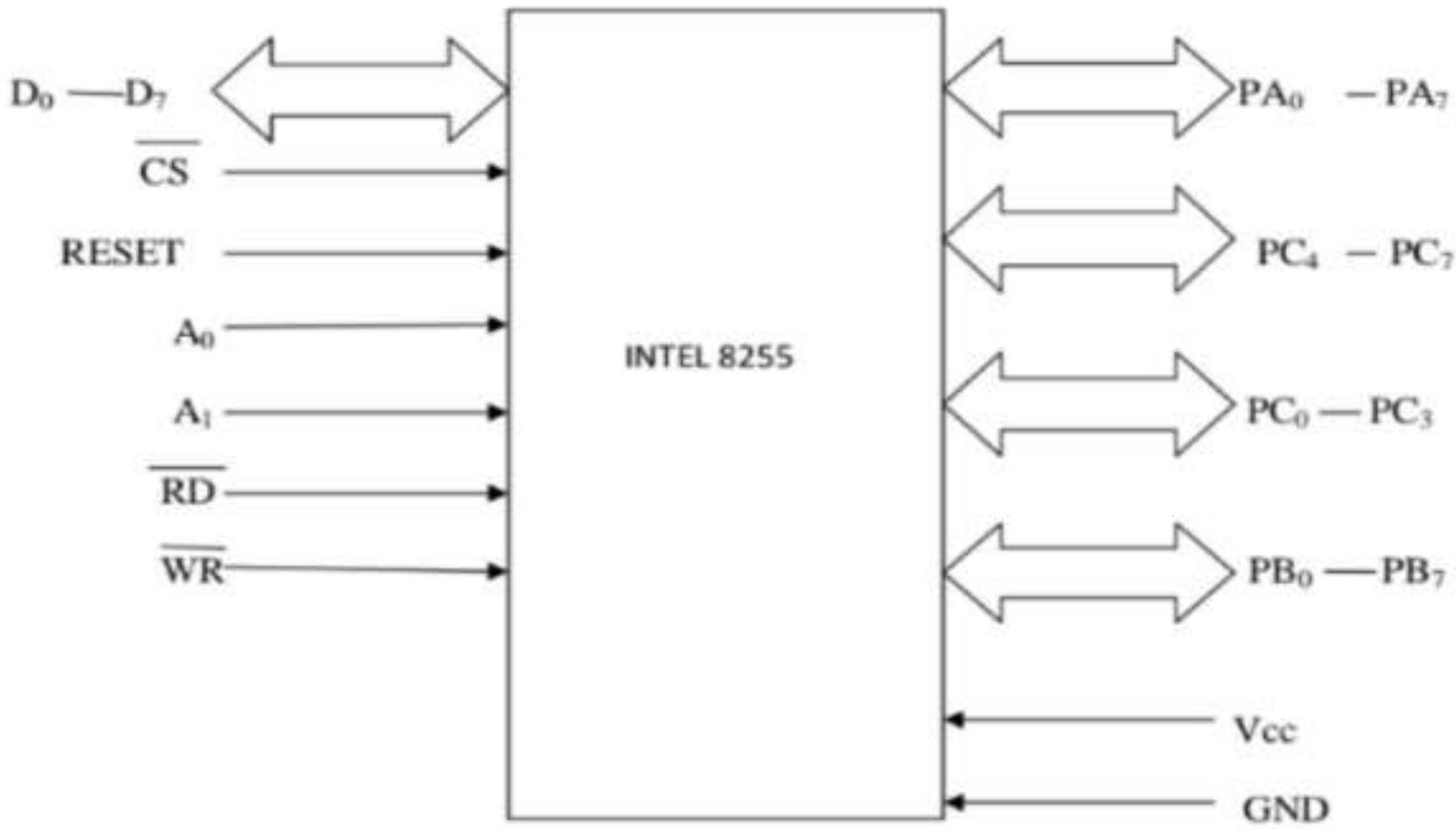
Group A	**Group B**
Port A	Port B
Port C upper	Port C lower

The control word initialises the ports.

ARCHITECTURE OF INTEL 8255A :

- It is a 40 pin IC package.
- It operates on a single +5V dc supply.

SCHEMATIC DIAGRAM OF INTEL 8255A:



PC4 – PC7 - Port C upper

PC0 – PC3 ⇨ Port C lower

The pins for various ports are:-

- $PA_0 - PA_7$ → 8 pins of Port A
- $PB_0 - PB_7$ → 8 pins of Port B
- $PC0 - PC_3$ → 4 pins of Port C lower
- $PC_4 - PC_7$ → 4 pins of Port C upper

The important control signals are:

D0 – D7 ⟶ These are 8-bit bidirectional data lines.

$\overline{CS}$ (Chip Select) ⟶ It is a chip select signal. The low status of this signal enable communication between CPU & 8255.

$\overline{RD}$ (Read) ⟶ It allows the CPU to read data from the input port of 8255.

$\overline{WR}$ (Write) ——→ when WR goes low, the CPU writes data or control word into 8255.

The CPU writes data into the output port of 8255 & the control word into the control word register (CWR).

RESET ——→ It is used to reset the 8255 chip.

A_0, A_1 ——→ The selection of I/O port & CWR is done using A_0 & A_1 in Conjunction with RD & WR . A_0 & A_1 are normally connected to the LSBs of the address bus. They are used for addressing the 4 register, any one of ports and a i.e. 3 I/O CWR.

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A1	A2	Function
X	X	X	X	X	CWR is tristated
1	1	0	X	X	Data bus is tristated

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A1	A0	Output (write) cycle
1	0	0	0	0	Data bus to port A
1	0	0	0	1	Data bus to port B
1	0	0	1	0	Data bus to port C
1	0	0	1	1	Data bus to CWR

$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	A1	A0	Input (read) cycle
0	1	0	0	0	Port A to data bus
0	1	0	0	1	Port B to data bus
0	1	0	1	0	Port C to data bus
0	1	0	1	1	CWR to data bus

X=don't care, as $\overline{CS}$ is 1

As $\overline{RD}$, $\overline{WR}$=1 so even if $\overline{CS}$ =0, A_1, A_0=X

Mode of operation of 8255:

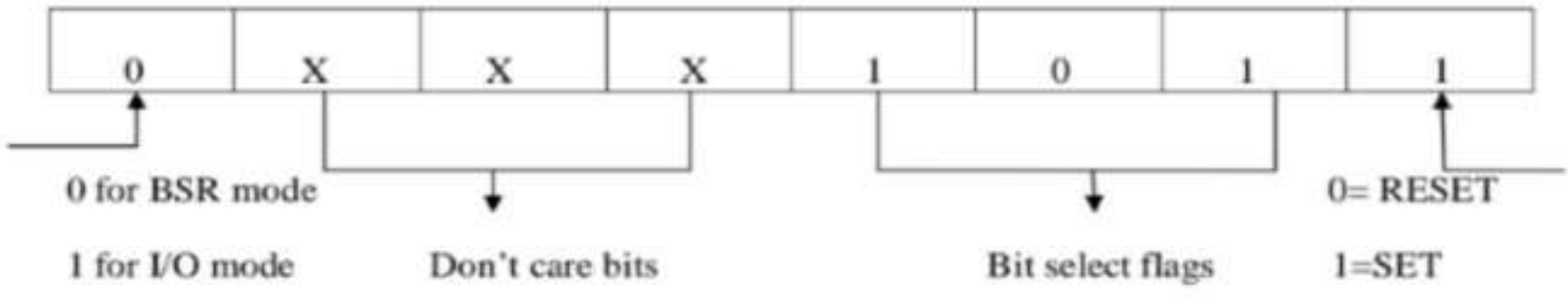
There are 2 basic modes of operation of 8255.

- **BSR Mode (Bit Set Reset mode)**
- **I/O Mode (Input / Output mode)**

BSR Mode:

In this mode, any of the 8-bits of Port C can be set or reset depending on bit B0 of the control word. The bit to set or reset is selected by bit select flag B3, B2, B1 of the CWR.

BSR mode control word register format:



B3	B2	B1	Selected bits for port C
0	0	0	B0

0	0	1	B1
0	1	0	B2
0	1	1	B3
1	0	0	B4
1	0	1	B5
1	1	0	B6
1	1	1	B7

I/O Mode:

In this mode,the 8255 works as programmable I/O ports.

There are three modes of operation of 8255 under I/O mode.

i.e. Mode 0

Mode 1

Mode 2

Mode 0 – Simple input / output

Mode 1 – Strobed input / output

Mode 2 – Bidirectional input / output

All these modes can be selected by programming a register internal to 8255 known as control word register (CWR), which has two formats – one for BSR mode and another for I/O mode.

Mode 0:

- It is also known as basic Input / Output mode.
- This mode provide simple Input / Output capability using each of the three ports.

Features of Mode 0:--

- Two, 8-bit ports i.e. port A and port B & two,4-bit ports i.e. port C upper and port C lower are avaliable. The two 4-bit ports of port C can be combinedly used as a 3rd 8-bit port.
- Any port can be used as an input or output port.
- Output ports are latched. But, input ports are not latched.
- A maximum of 4 ports are avaliable. So that 16 I/O configurations are possible.

Mode 1 :

This mode is also called strobed I/O mode. In this mode port C is used for generating hand shake. Signals to control the input or output action of port A or port B.

Features of Mode 1:

- Two groups i.e. group A and group B are avaliable for strobe data transfer. Each groups contains one 8-bit data I/O port, one 4-bit control port.
- The 8-bit data port can be either used as input or output port.
- Both the inputs and outputs are latched. Out of 8-bit port C i.e. PC0 to PC2 are used to generate control signals for port B and PC3 to PC5 are used to generate control signals for port A. The line PC6 and PC7 may be used as independent data lines.

Mode 2:

It is known as strobe bidirectional I/O mode.

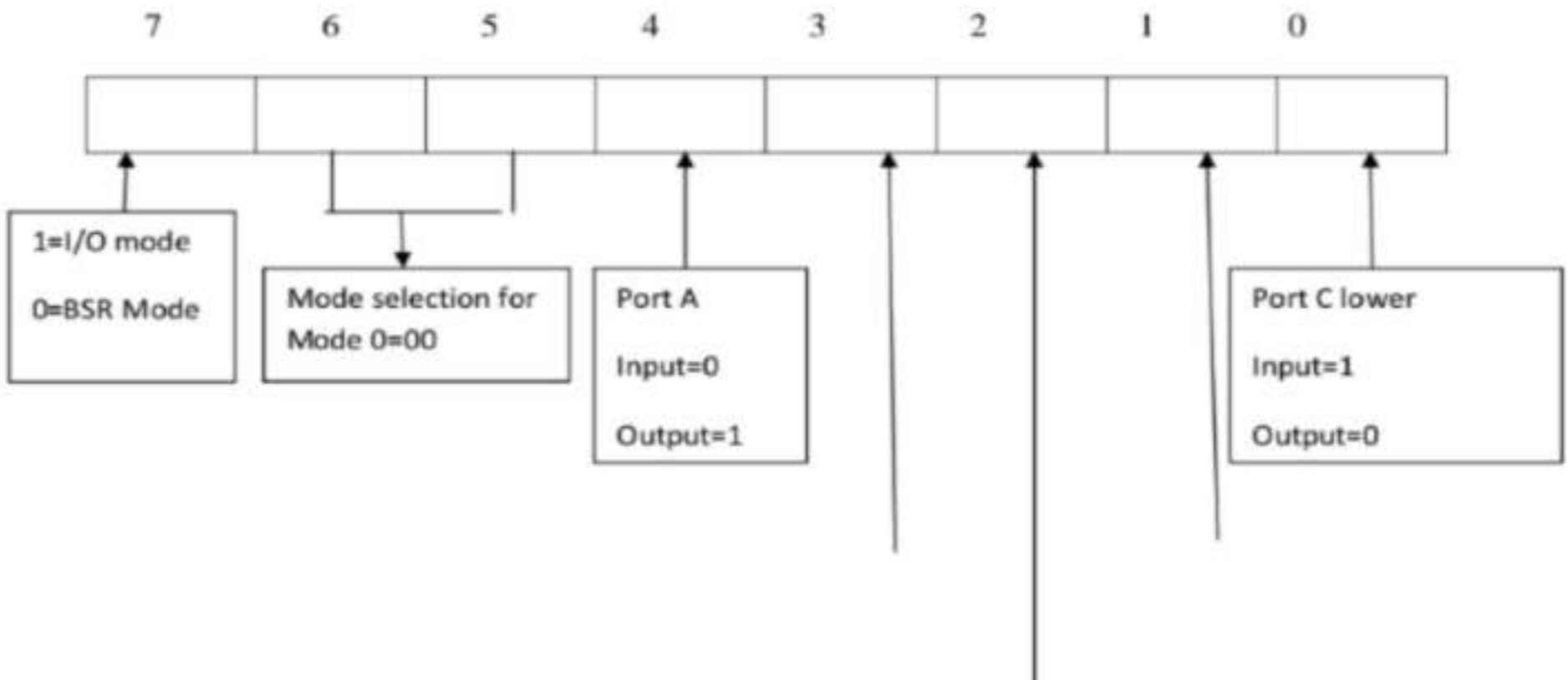
Here only port A is available. In this mode 8255 acts as a bidirectional 8-bit port with handshakes signal.

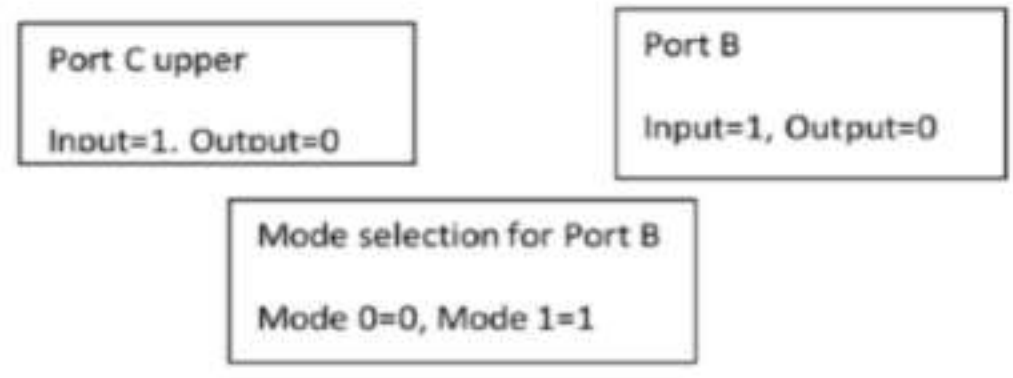
Features of mode 2:

- Here only port A is available.
- The 8-bit port A is bidirectional and additionally a 5-bit control port C is available.
- Input and outputs are both latched.
- Three I/O lines are available at port C i.e. PC0 to PC2.
- The 5-bit control port C i.e. PC3 to PC7 is used for generating handshake signals for 8-bit data transfer on port A.

Control Word format in I/O mode:

For programming the ports of 8255 a control word is formed. The CPU outputs the control word to 8255 which is written into the control word register in 8255.No read operation of control word register is allowed.




Port C upper
Input=1. Output=0
Port B
Input=1, Output=0
Mode selection for Port B
Mode 0=0, Mode 1=1

DMA CONTROLLER (Intel 8257/ 8237):

- Intel 8257 is a four channel DMA controller used to facilitted the direct memory access or DMA mode of data transfer.
- In this mode the devices transfers data directly to or from memory with any interference on the CPU.
- For facilitating DMA data transfer between several devices a DMA controller is used.

INTERNAL ARCHITEHCHTURE OF 8257:

- The chip supports four DMA channel i.e four peripheral devices can independently request for DMA data transfer through these channel at a time.
- The DMA controller has 8-bit internal data buffer ,a read/write a control unit ,a priority reserving unit along with a set of register.

REGISTER ORGANISATION OF 8257:

- Each of the four channels of 8257 has a pair of two 16-bit registers.
  i.e DMA register and terminal count register.
- There are two common register for all the channel.
  (a) Mode set register
  (b) Status register

DMA address register:

- Each DMA channel has DMA address register . Its function is to store the address of the starting memoy location which will be access by the DMA channel.

Terminal count register:

- This 16-bit register is used to make sure that the data transfer through a DMA channel stops after the required number of DMA cycles.
- The load of 14-bit terminal count register are initialize with the binary equivalent of the number of DMA cycle minus one.
- After each DMA cycle the terminal count register contained will be decremented by 1 and finally it becomes 0 after the required number of DMA cycle.
- The bits 14 and 15 of this register indicate the type of the DMA operations
- There are three type of DMA cycle.
  (a) Read DMA cycle
  (b) Write DMA cycle
  (c) Verify DMA cycle

Bits 15	Bits 14	Types of DMA operation
0	0	Verify DMA cycle
0	1	Write DMA cycle
1	0	Read DMA cycle
1	1	(Illegal)

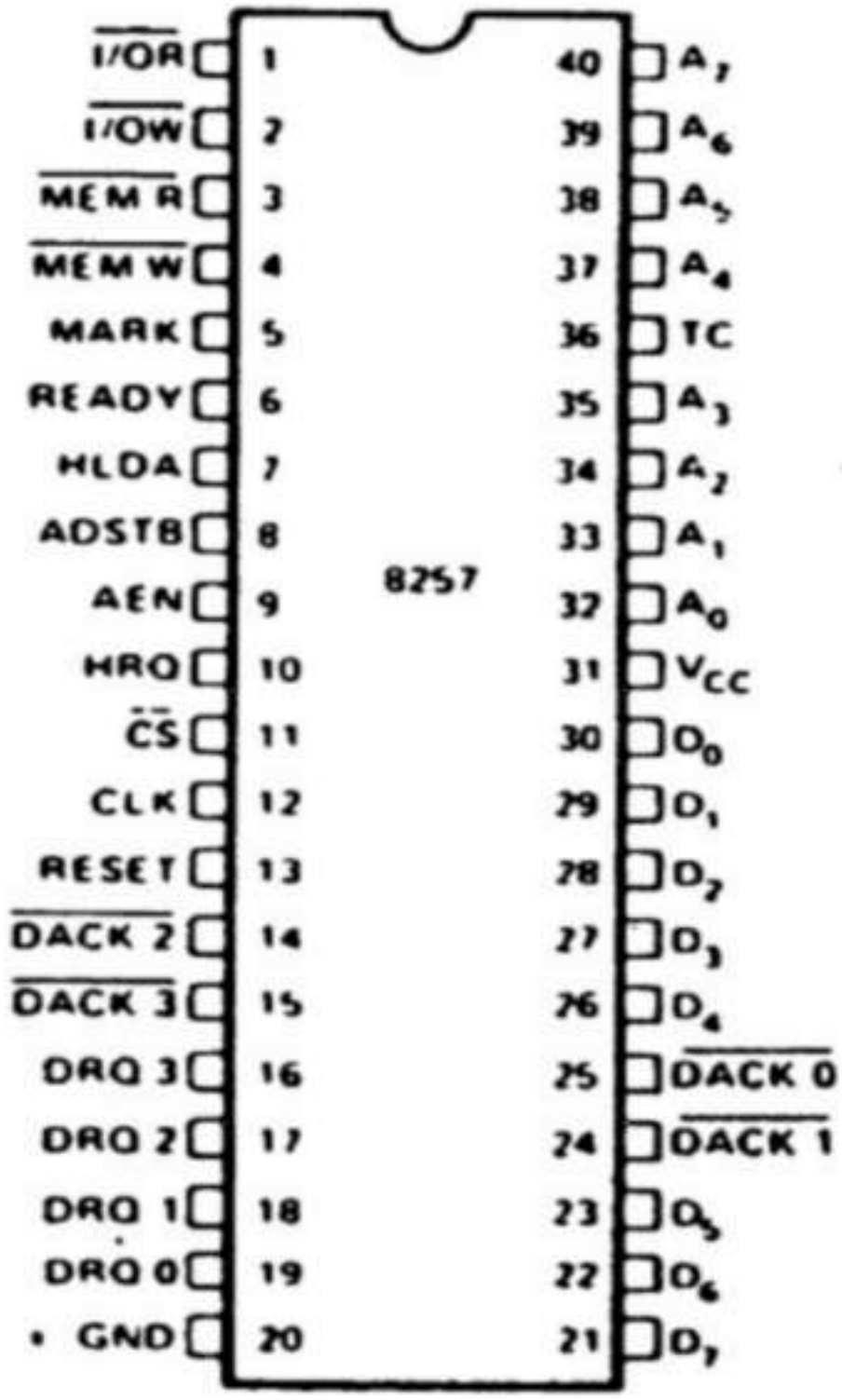
MODE SET REGISTER:

The function of this register is to enable the DMA channel indivisually and also to set the various modes of operation

- The bits D_0–D_3 enable one of the four DMA channel of 8257.
  For example : If D_0 is 1,channel0 is enable.
  If D_4 is set rotating priority is enable
  otherwise fixed priority is enable.
- If Bit D_6 (TC) is set the selected channel is disable after terminal count register is least and it prevents any further DMA cycle on the channel.
- The auto laod bit that is D_7 if set enable channel for the repeat block chaining operations.
- After the first block is transfer using DMA the channel 2 register are reloaded with corresponding channel 3 register for the next block transfer if the update flag is set.
- Extended write bit D_5 is set to 1extends the duration of memory write ($\overline{MEMW}$) and I/O write signals $\overline{IOW}$.

STATUS REGISTER:

- Bits D_0-D_3 contain the terminal count status for the four indivisual channels.
- If any of these is set it indicates that the specific channel is reached the terminal count register.
- Bit D_4 repreasents the update flag if it is set the contentsof channel 3 registerare reloaded with corresponding channel 2 registers without any software intervention.



(Pin diagram of 8257)

SIGNAL DESCRIPTION OF 8257

DRQ_0-DRQ_3 :

- These are the 4 individual channels DMA request inputs used by the peripheral devices for requesting DMA services.
- The DRQ_0 has the highest priority ,while DRQ_3 has the lowest,if fixed priority mode is selected.

$\overline{DACK_0}$ - $\overline{DACK_3}$:

- These are the active low DMA acknowledgement output lines which inform the requesting peripheral that the DMA request has been honoured.

D_0-D_7 :

- These are bidirectional data lines used to interface the system bus with the internal data bus of 8257.

$\overline{IOR}$:

- In Master mode ,this signal is used to read data from a peripheral during a memory write cycle.
- In slave mode, this input signal is used by the CPU to read internal registers of 8257.

$\overline{IOW}$:

- In Master mode , it is a control output that is used to write data to a peripheral during DMA memory read cycle
- In slave mode ,it is used to load the contents of the data bus to the 8 bit mode register.

CLK :

- This is a clock frequency input required to generate system timing for internal operations of 8257.

RESET :

- This input disables all the DMA channels by clearing the mode registers & translate all the control lines.

A_0-A_7 :

- These are address lines.

CS :

- It is an active low chip select line that enables R/W operations from or to 8257 in slave mode.
- In master mode it is disabled.

READY:

- This input is used to stretch memory read & write cycles of 8257 by inserting wait states.

HRQ :

- Hold Request output requests access of system bus of 8085 CPU.

HLDA :

- This input if high indicates to the DMA controller that the system bus has been generated to the requested peripheral to the CPU.

$\overline{\text{MEMR}}$:

- This active low memory read output is used to read data from the addressed memory location during DMA read cycle.

$\overline{\text{MEMW}}$:

- This active low memory write output is used to write data to the addressed memory location during DMA write operation.

ADSTB (Adress strobe) :

- This output from 8257 strobe the higher byte of the memory address generated by the DMA controller into the latches.

AEN (Adress Enble) :

- This output is used to disable the system address & data to stop the non DMA devices from responding during DMA operations.

TC (Terminal Count) :

- This output indicates to the currently selected peripheral that the present DMA cycle is the last for the previously programmed data bus.

MARK :

- The modulo 128 mark output indicates the selected peripheral that the current DMA cycle is the 128^{th} cycle since the previous marked output.

PRIORITY OF DMA REQUESTS :

The priority resolver resolves the priority of the 4 DMA channels.

There are 2 schemes :-

1. Fixed priority scheme
2. Rotating priority scheme

1.**Fixed Priority Scheme :**

In Fixed priority scheme ,each device connected to a channel is assigned a fixed priority.

DRQ_0 has the highest priorty

DRQ_1

DRQ_2

DRQ_3 has the lowest priority.

2.**Rotating Priority Scheme :**

In Rotating Priority scheme ,the priority assigned to the channels are not fixed.
A channel that gets service becomes the lowest priority channel . This avoids the dominance of any one channel. For example, after channel 0 is served, it becomes the lowest priority channel.

INTERFACING A DMA CONTROLLER WITH A SYSTEM :

The DMA controller interfacing circuit implements a switching arrangement for the address,data & control buses of the memory & peripheral sub-system from or to the CPU To or from the DMA controller.

1. The peripheral device that wants to do a DMA data transfer sends a request to DMA controller on DRQ line.
2. The DMA controller then sends a HOLD request to CPU on HRQ line to get control of system bus.
3. The CPU completes the current machine cycle,it releases the system bus to the DMA controller and sends a high HLDA signal.
4. The controller then sends a DMA acknowledgement $\overline{DACK}$ to the requesting peripheral.Now it is ready for DMA data transfer.
5. During DMA data transfer CPU remains in HOLD state.

PROGRAMABLE INTERRUPT CONTROLLER (PIC) 8259A

A PIC takes care of a number of simultaneously appearing interrupt requests along with their types and priorities in a multiple interrupt system.

ARCHITECTURE OF 8259A

- **INTERRUPT REQUEST REGISTER (IRR):**
  IRR stores all the interrupt request in it in order to solve them one by one on priority basis.

- **IN – SERVICE REGISTER (ISR):**
  It stores all the interrupt request that are being served.

- **PRIORITY RESOLVER:**
  This unit determines the priorities of the interrupt request appearing simultaneously. (In fixed priority mode IR0 has the highest priority while IR7 has the lowest)

- **INTERRUPT MASK REGISTER (IMR):**
  The register stores the bits required to mask the interrupt inputs.

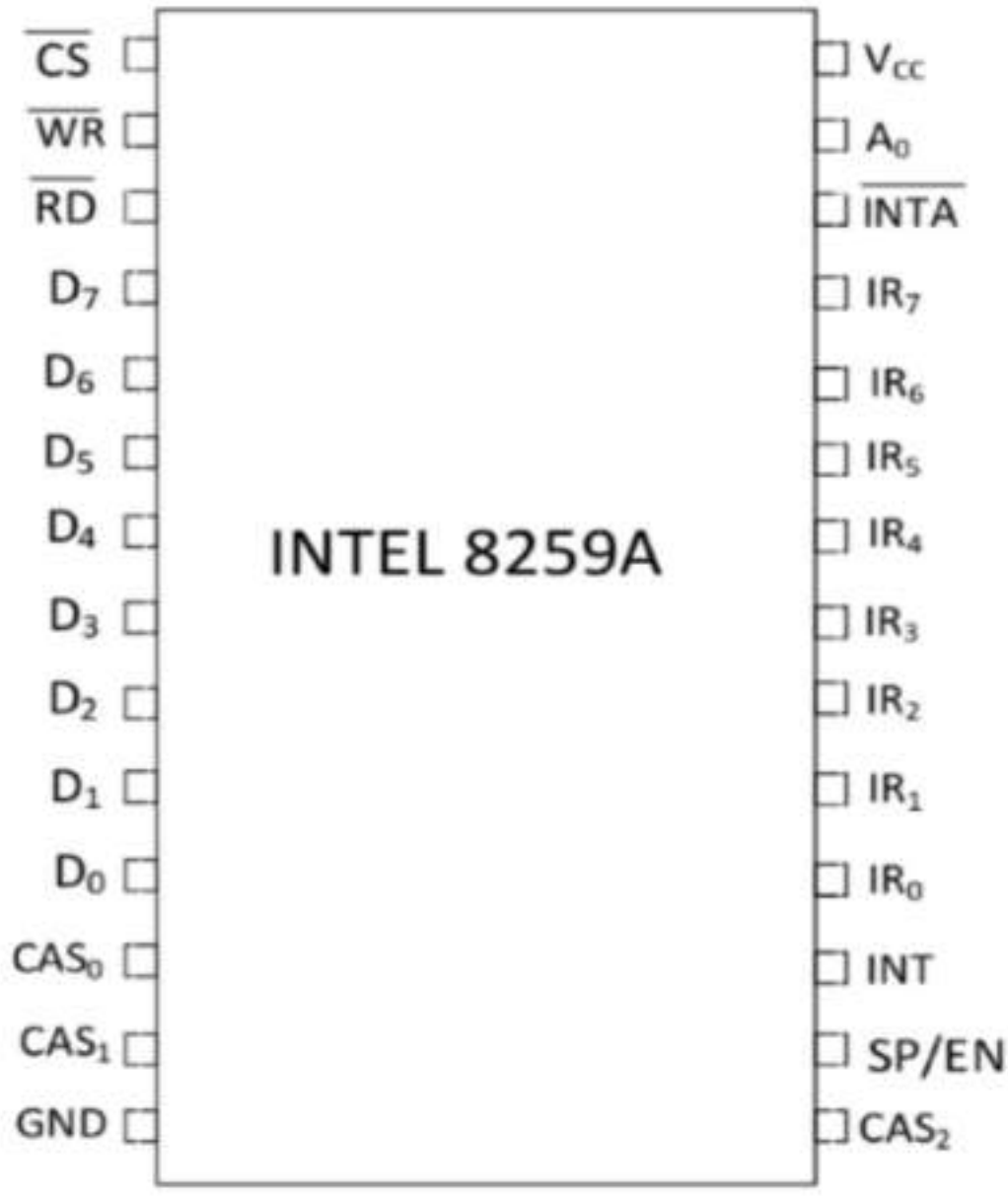
- **INTERRUPT CONTROL LOGIC:**
  This block manages the interrupt and interrupt acknowledge signal to be send to the CPU for serving one of the 8 interrupt request.

- **DATA BUS BUFFER:**
  This bidirectional buffer interface internal 8259A bus to the microprocessor system data bus

- **R/W CONTROL LOGIC:**
  This circuit accepts and decodes command from the CPU.

- **CASCADE BUFFER /COMPARATION:**
  This blocks stores and compare to IDs of all the 8259A use in the system.

8259A PIN DIAGRAM



PIN- DESCRIPTION :

It is a 28 pin IC package.

- **$\overline{CS}$** :- It is an active low chip selected signal for enabling read and write operation of 8259A.
- **$\overline{WR}$** :- This pin is an active low write enable input to 8259A.
- **$\overline{RD}$** :- It is an active low read enable input to 8259A.

- **D_0-D_7** :-They form a bidirectional data bus that carry 8-bit data and interrupt vector information.

- **CAS_0-CAS_2** (CASCADE LINE) :- A single 8259A provides 8 vectored interrupts. If more interrupt are required , the 8259A is used in the cascade mode in which a master 8259A along with 8 slave 8259A can provide up to 64 vectored interrupt lines. These cascade lines attached select line for addressing the slave 8259A.

- **$\overline{SP}/\overline{EN}$** :- (SLAVE PROGRAM / ENABLE LIGIC)

  This pin is dual purpose pin. When the chip is used in buffered mode, it can be used as a buffer enable to control buffer trans-receiver.
  In buffer mode $\overline{EN} = 0$
  If not in buffer mode then the pin is used as input to designate whether the chip is used as a master. ($\overline{SP} = 1$) or a slave ($\overline{EN} = 0$)

- **INT** :- This pin goes high whenever a valid interrupt request is asserted. It is used to interrupt the CPU.
- **$IR_0 - IR_7$** (INTERRUPT REQUEST)
  This pin act as a inputs to accept interrupt request to the CPU.
- **$\overline{INTA}$** :- (INTERRUPT ACKNOWLEDGE)
  The pin is an input used to strobe in 8259A interrupt vectored data on to the data bus.

The device 8259A can be interfaced with any CPU using either device polling or interrupt schemes.

DEVICE POLLING

→ Here the CPU keeps on checking each peripheral device in sequence to make sure if it requires any service from the CPU.
→ If any such service request is noticed, the CPU serves the request and then goes on to next device in sequence.
→ After all the peripheral devices are scanned, the CPU again starts from the 1st device.
→ This results in reduction of processing speed because most of the CPU time is consumed in polling the peripheral devices.

COMMAND WORDS OF 8259A

There are two types of command words i.e

i. **Initialization Command Words (ICWs)**
ii. **Operation Command Words (OCWs)**

i. **Initialization Command Words (ICWs)**

→ Before it starts functioning, 8259A must be initialized by writing 2-4 command word in to the respective command word register. These are called as ICWs.

➢ If $A_0 = 0$ & $D_4 = 1$

The control word is recognized as ICW_1.

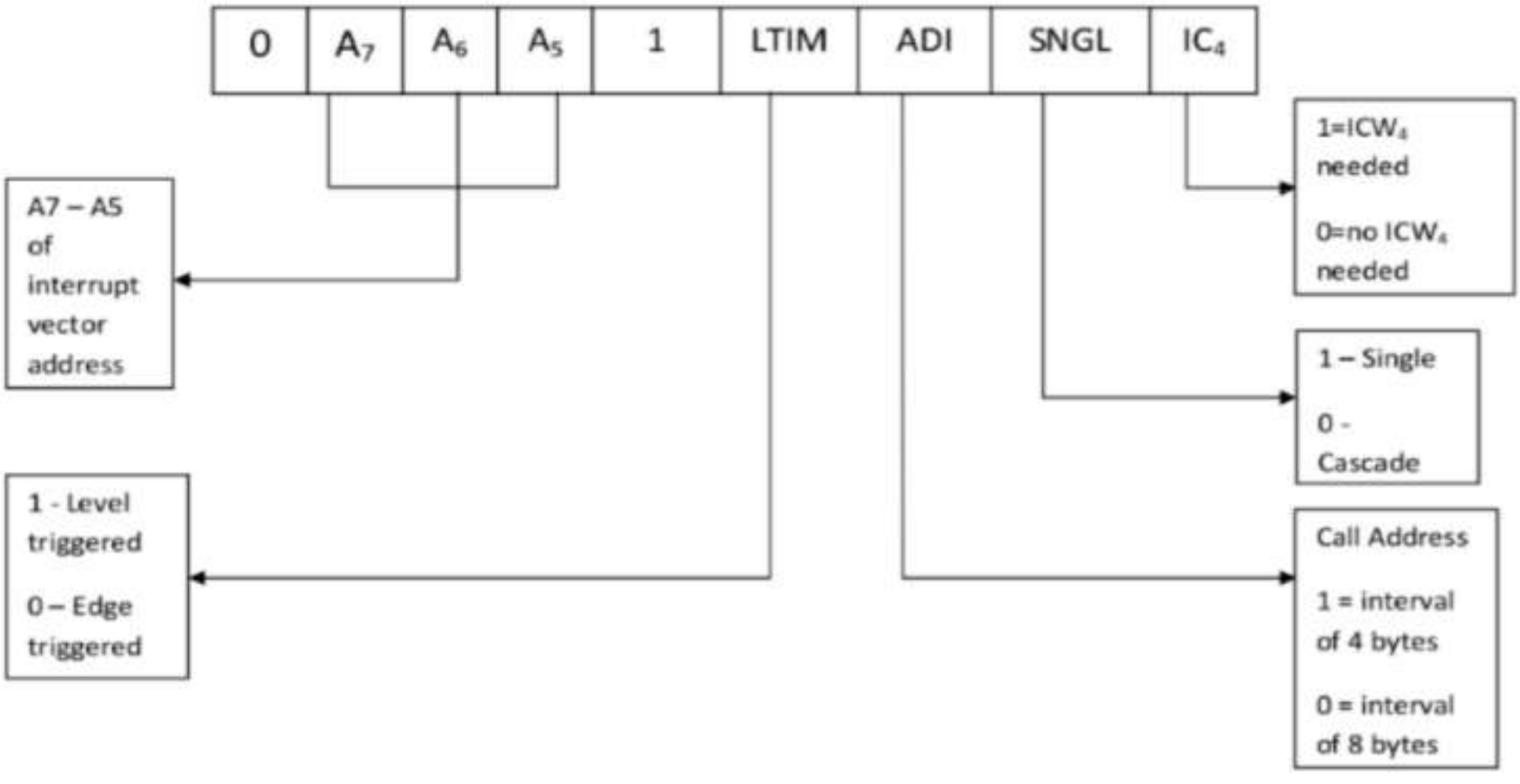
→ It contains the control bits for edge or level triggered mode, single or cascade mode, call address interval and whether ICW_4 is needed or not.

➢ If $A_0 = 1$

The control word is recognized as ICW_2.

→ The ICW_2 stores details regarding interrupt vector address.

ICW_1



$A_0 = 0$ & $D_4 = 1$, ICW_1

$A_0 = 1$, **ICW_2**

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8

- ICW_1 & ICW_2 are compulsory command word in initialization sequence of 8259A while ICW_3 & ICW_4 are optional.
- The ICW_3 is read only when there are more than one 8259A in the system i.e cascading is used.
- The ICW_3 loads an 8-bit slave register.

MASTER MODE ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S0

Sn = 1 – IRn (Input as a slave)
= 0 – IRn (Input doesn't have a slave)

SLAVE MODE ICW_3

A_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	ID_2	ID_1	ID_0

D_2 D_1 D_0 - 000 to 111 for IR_0 to IR_7 or slave 0 to slave 7.

ICW_4

The use of this Command Word depends on the IC_4 bit of ICW_1.

If IC_4 = 1, ICW_4 is used otherwise it is neglected.

A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	SFNM	BUF	M/$\overline{S}$	AEOI	MPM

SFNM :-

If SFNM = 1, the special fully nested mode is selected.

BUF:-

If BUF = 1, the buffer mode is selected.

M/$\overline{S}$:-

The bit decided the master and the slave mode of 8259A.

If M/$\overline{S}$ = 1, 8259A is a master.

If M/$\overline{S}$ = 0, 8259A is a slave.

If BUF = 0, M/$\overline{S}$ is to be neglected.

AEOI :-

If AEOI = 1, the automatic end of interrupt mode is selected.

MPM:-

If MPM bit is 0, the 8085 system operation is selected.

If MPM = 1, 8086/88 operation is selected.

Operation Command Words (OCWs):-

- The modes of operation of 8259A can be selected by writing three internal register called as operation command words register.
- The data written into them is called as operation commands word.
- There are three OCWs i.e.

  i. OCW_1
  ii OCW_2
  iii OCW_3

  - OCW_1 is used to mask the unwanted interrupt requests.
  - OCW_2 controls the end of interrupt, the rotate mode and their combinations.
  - OCW_3 handles the special mask mode.

Operating Modes Of 8259A:-

1. **Fully nested mode:-**
   → This is the default mode of operation of 8259A.
   → IR_0 has the highest priority and IR_7 has the lowest.
   → When interrupt request are noticed, the highest priority request among them is determined and the vectored is placed on the data bus.

2. **End of Interrupt (EOI) Mode:-**
   The ISR bit can be set by EOI command issued before returning from the interrupt service in this mode.

3. **Automatic Rotation:-**
   This is used in application where all the interrupting devices are of equal priority.

4. **Automatic EOI Mode:-**
   In this mode 8259A performs an non specific EOI operation.

5. **Specific Rotation:-**
   In this mode a bottom priority level can be selected to fix other priorities.
   If IR5 is selected as a bottom priority then IR5 will have least priority and IR4 will have a next priority and IR6 will have the highest priority.

6. **Special Mask Mode:-**
   In this mode when a mask bit is set, it inhibits/stops further interrupt at that label & enables interrupt from other label which are not masked.

7. **Edge and Level Triggered Mode:-**
   This mode decides whether the interrupts should be edge triggered or label triggered.

8. **Reading 8259 Status.**
   The status of internal register of 8259A can be read using this mode.

9. **Poll Command:-**
   In this mode the INT output of 8259A is neglected.

10. **Special Fully Nested Mode:- (SFNM)**
    This mode is used in more complicated system where cascade is used.
    In this mode, the master interrupt the CPU only when the interrupting device has a higher or the same priority than the one currently being solved.

11. Buffered Mode:-

When the 8259A is used in systems where bus driving buffers are used, it is usedin buffered mode.

.

12. Cascade Mode:-

In this mode, 8259SA is connected in system containing one master and maximum 8 slave to handle up to 64 priority label.

DATA ACQUISITION SYSTEM

The peripheral devices are connected to the PIC maximum 8 I/O devices can be connected to 8259 in single mode through IR0-IR7 lines. The PIC function is overall manager in an interrupts driven system. If more than one I/O devices then send interrupt request at the same time the PIC determines the priority.

It first entered into the I/O devices the higher priority it sends an interrupt request to the microprocessor through INT line. The microprocessor sends a acknowledgement through INTA line. On the receive top INTA signal all the interrupt of low priority are inhibited.

If more than 8 I/O devices to transfer data using interrupt to 8259 ICs can be connected in series such a connection is known as cascading up to 64 I/O devices connected employing to 8259 ICs.

More than one 8259 are employed in the system the 8259 which is connected to the processor is called master then 8259 which is connected to the master 8259 is called slave. For the measurement and control of physical quantity such speed, displacement etc transducers are used which give electrical voltage proportional to physical quantity.

This electrical voltage obtain as an output of an transducer is an analog quantity it must be converted into digital quantity by the A/D converter before it is applied to a microprocessor. To handle multiple signals an analog multiplexer is used.

Working Principle of Successive Approximation type A/D Converter:-

An A/D converter analog multiplexer is used to convert analog signals to digital quantity. This digital o/p is fed to the microprocessor for processing.

- The most popular method of analog to digital conversion is successive approximation method.
- Here an unknown voltage Vin is compared with a fraction of the reference voltage Vref.

➢ In the first step the unknown voltage Vin is compared with ½Vr. If Vin>=1/2 Vr, the MSB of the digital output is set to 1.If Vin<1/2Vr, MSB is set to 0.

➢ In the next step Vin is compared with (1/2b1+1/4)Vr, where b1 is the MSB value already determine. If Vin>=1/2(b1+1/4)Vr the 2nd bit is set to 1.If Vin<1/2(b1+1/4)Vr the 2nd bit is set to 0.

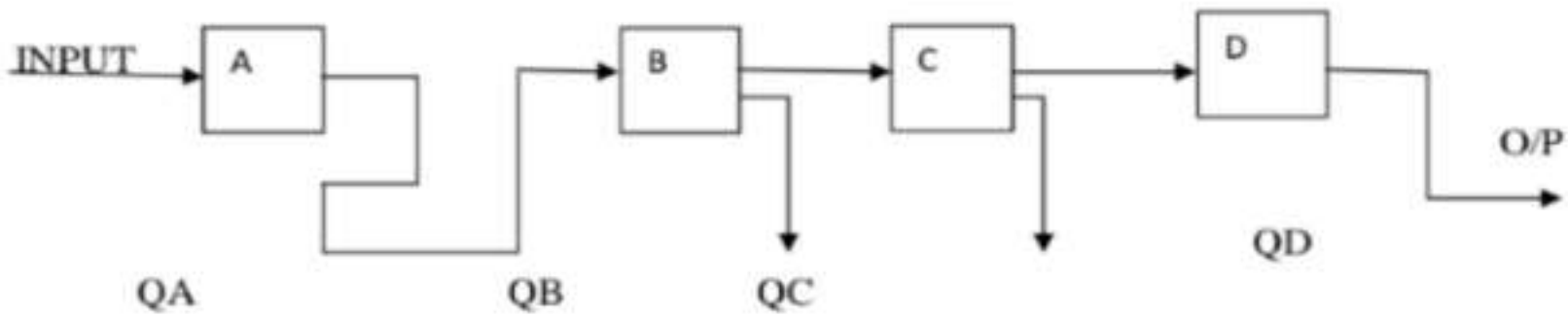
➢ To obtain the 3rd bit of the digital output Vin is compared with (1/2b1+1/4b2+1/8)Vr.

Clock for A/D converter:-

- The clock frequency required for A/D converter lies in the range of 50 KHz to 800 KHz.
- The clock frequency available & microprocessor kit for user is about 3MHz.

The higher frequency of the microprocessor of the microprocessor can be reduced by using 4 master slave flip-flops which reduced the frequency by 1/10th.

Fig: -



Sample & hold circuit:-

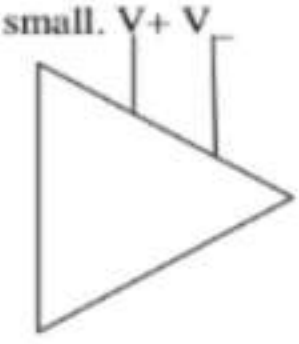
This circuit samples the instantaneous value of the AC signal and maintains it at a constant level .It make this constant voltage available to A/D converter during the conversion period.

LF398:-

LF398, LF298, LF198 are monolithic sample and hold circuit of national semiconductor. An external capacitor known as hold capacitor is used with LF398 to hold the voltage request upon it.

Droop rate:-

It is the rate at which the o/p of sample & hold circuits decreases. To make the o/p partially constant this rate should be very small. V+ V-

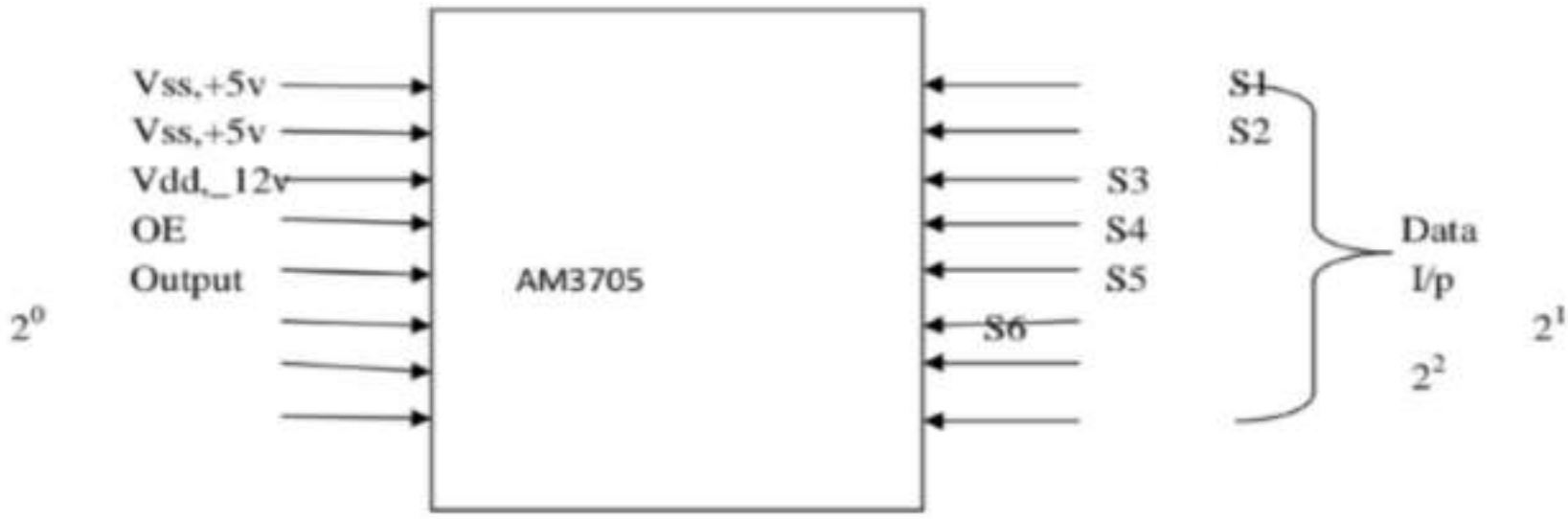


Analog Multiplexer:-

When a number of inputs are to be processed by the computer an analog multiplexer is used. The multiplexer has logic to switch ON a particular desired channel.

- The computer sends appropriate logic to the multiplexer to switch ON the desired channel.
- A multiplexer may be of 8 or 16 channel input multiplexer.
  Example:- AM3705 is an 8 channel analog multiplexer of national semiconductor.

Schematic diagram of AM3705:-



LOGIC FOR AM3705:-

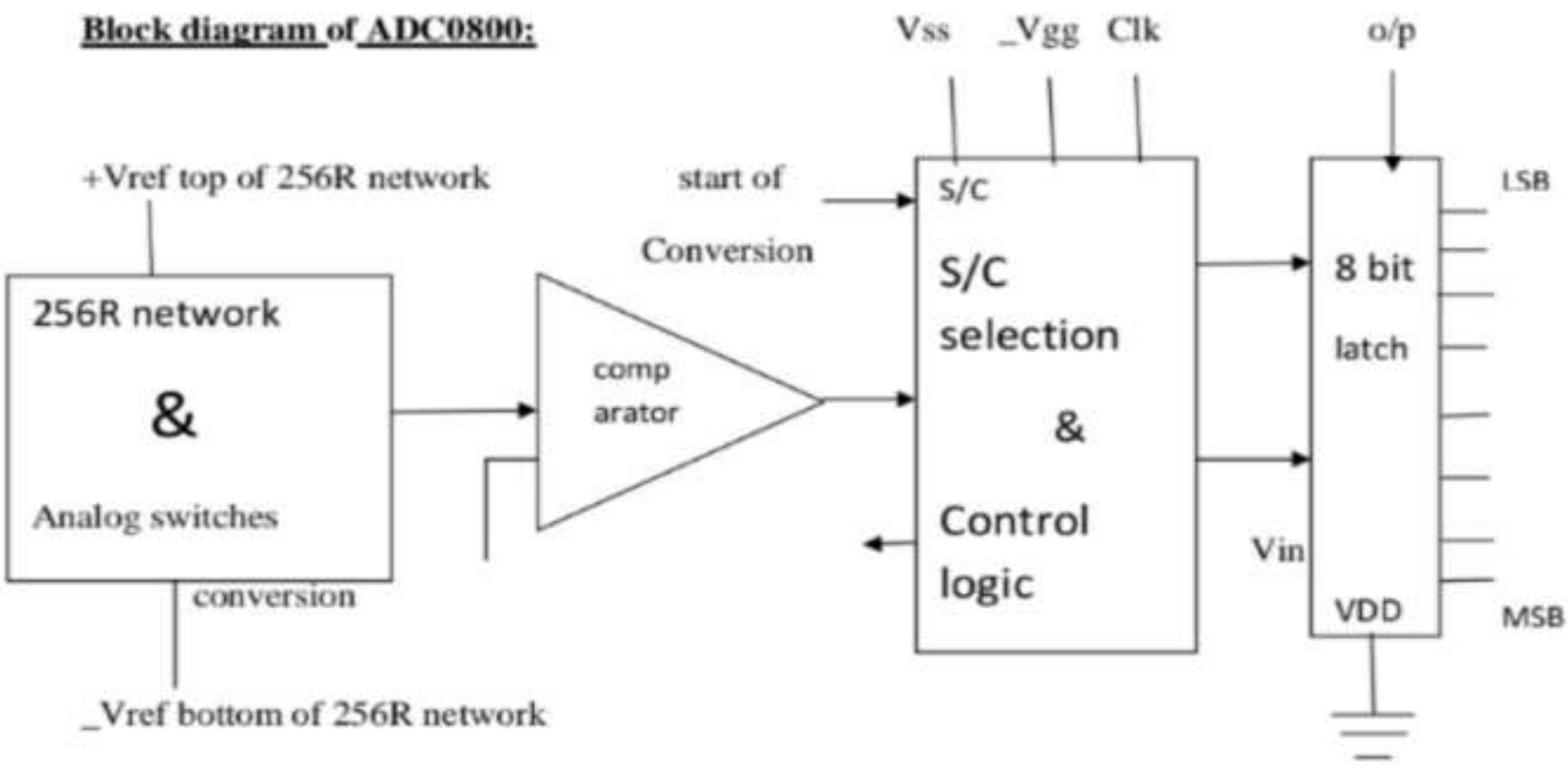
LOGIC INPUT 2^2	2^1	2^0	OUTPUT ENABLE	CHANNEL
L	L	L	H	S_1
L	L	H	H	S_2
L	H	L	H	S_3
L	H	H	H	S_4
H	L	L	H	S_5
H	L	H	H	S_6
H	H	L	H	S_7
H	H	H	H	S_8

ADC 0800 :-

- The ADC0800 is an 8 bit monolithic A/D converter of national semiconductor .It contains a 256 series register which is called 256R network.
- Analog switches selection & control logic a high input impedance comparator and 8 bit latch have been incorporated.
- It uses accessive approximation technique for analog to digital conversion the reference voltage is applied to 256R network.
- The unknown analog input voltage is compared with a fraction of reference voltage with help of analog switches and R network.

The reference voltage applied approx the 256R network determines the analog input range in case of Vr=100volts the top of R network is connected to +5volt supply at the bottom to -5volt.

Block diagram of ADC0800:



Zero & Full scale Adjustment:-

The ADC0800 gives complementary digital output. The digital o/p corresponding to 5v input is 10000000. If the o/p is not zero when the input is +5V adjustment of Vref . This adjustment is called full scale adjustment.

When the input voltage is zero the actual output is 80 decimal. If the digital output corresponding to zero volts the output is not 80 hexadecimal then it is adjusted by using 1 kilo ohm variable register. The relationship between input voltage +5v is

Vin=Vref(1/2b7+1/4b6+1/8b5+1/16b4+1/32b3+1/64b2+1/128b1+1/256b0)_5V.

DIGITAL TO ANALOG CONVERT(D/A CONVERTER)-

Digital to analog converters are used to convert digital quantity to analog quantity.

They produce an output current or voltage proportional to digital quantity apply to its input.

They are used for the control of relay, small motor, actuator etc.

In communication system digital transmission is faster and convenient but the digital signal have to be converted back to analog signal at the receiving terminal.

D/A converter are also used as a part of circuit tree of several A/D converter.

OPERATING PRINCIPLE OF A DAC-

DAC contain a ladder network. The network has input prints for binary bits of the digital word.

When MSB of the digital word is 1 it produces an output current Iref/2.

The bit next to MSB produces Iref/4 and so on.

The output current is given by

$$Iout=Iref(1/2\ b_{n-1}+1/4\ b_{n-2}+1/8\ b_{n-3}\ldots\ 1/2^n b_0)$$

for an n-bit digital input.

For an 8-bit DAC

$$Iout = Iref\ (1/2\ b_7+1/4\ b_6+1/8\ b_5+1/16\ b_4+1/32\ b_3+1/64\ b_2+1/128\ b_1+1/256\ b_0)$$

where b0, b1, . . . b7 are the binary bits of digital word apply to DAC.

$$V=IR \Rightarrow I=V/R$$

$$Iref=Vref/R$$

where R is resistance in series with Vref.

Example-

Vref=5V, R=2.5 KΩ

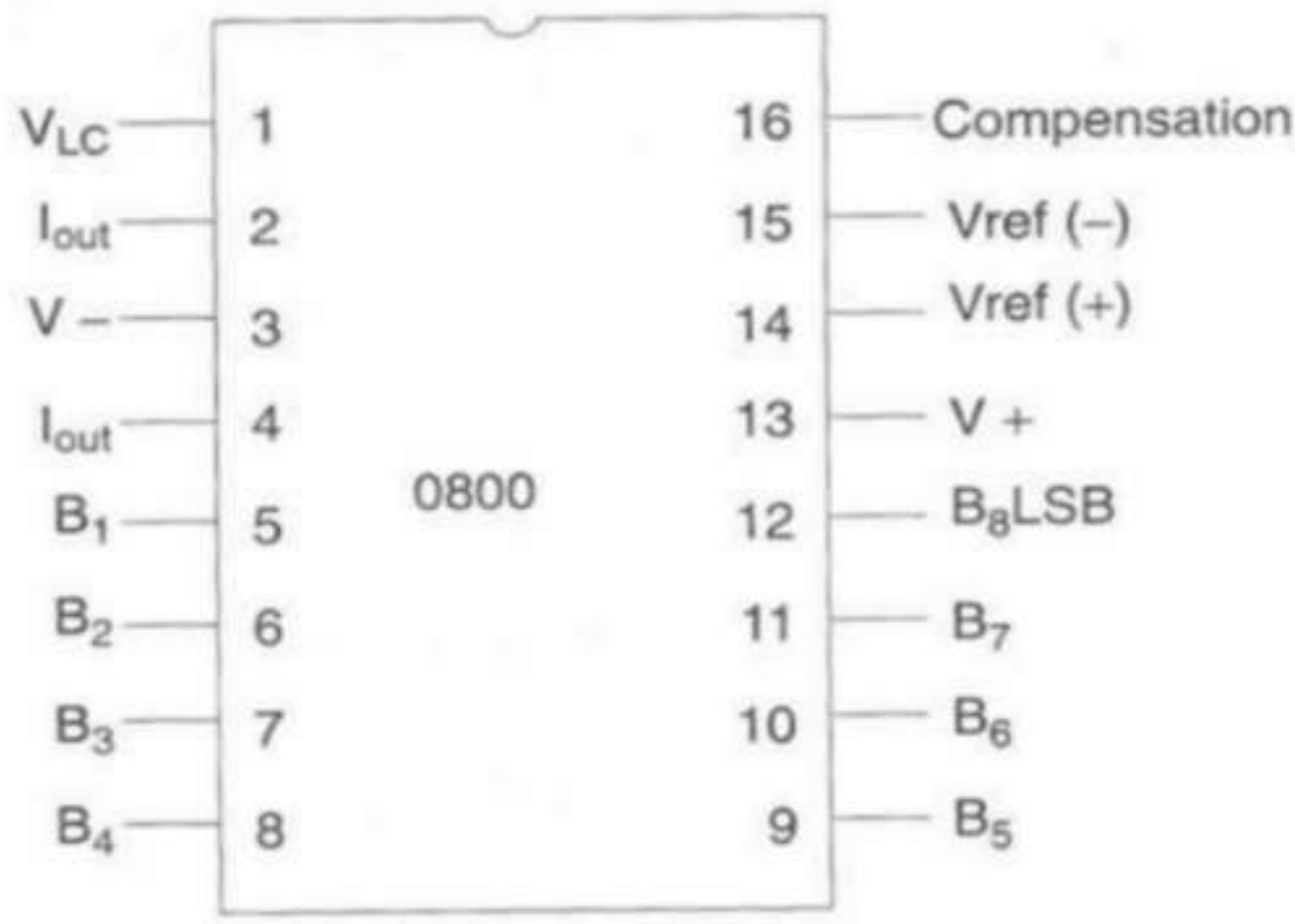
Iref= $5/2.5\times10^{3}$ =$50/25\times10^{-3}$ =2MA

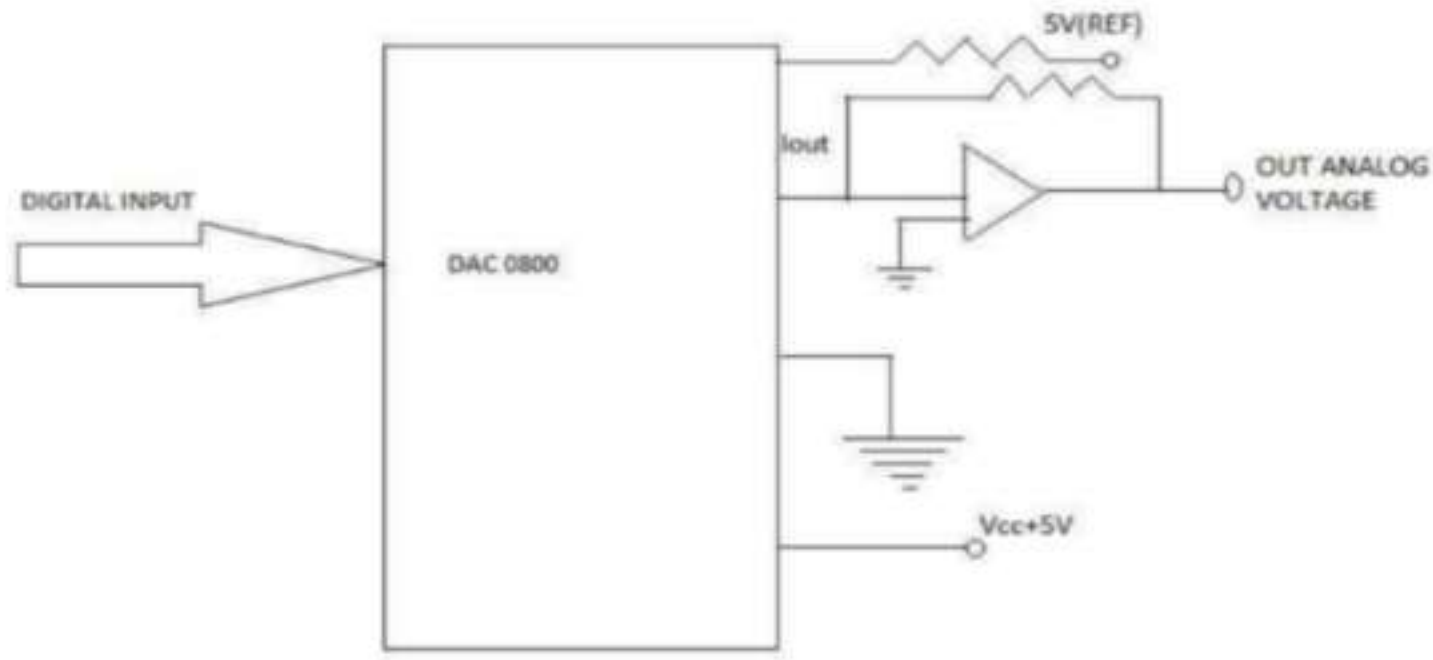
Using an external opamp a voltage output proportional to Iout is produced.

DAC 0800-

It is a simple monolithic 8-bit DA converter.

PIN DIAGRAM OF DAC 0800





ANALOG SIGNAL CONDITIONING

The signal conditioning means manipulating an analog signal in such a way that it means the requirement of the next stage for further processing.

Most common use is in A/D converter.

They are commonly used in control engineering application. Operational amplifier are use to carry out the amplification of the signal in signal conditioning stage.

INPUTS TO SIGNAL CONDITIONING

Signal inputs accepted by signal conditioner are DC voltage and current , AC voltage and current, frequency and electric charge.

Sensor inputs can be accelerometer, thermocouple, thermistor, registance thermometer etc.

Outputs for signal conditioning equipment can be voltage ,current , frequency timer or counter ,registance etc.

SIGNAL CONDITIONING PROCESSES

It includes amplification , filtering, converting , range matching ,isolation of analog signal. And any other processes required to make sensor output suitable for processing after conditioning.

1. FILTERING

It is the most common signal conditioning function as not all the signal frequency spectrum contain valid data.

60 Hz AC power lines present is most environment will produce noise if amplified.

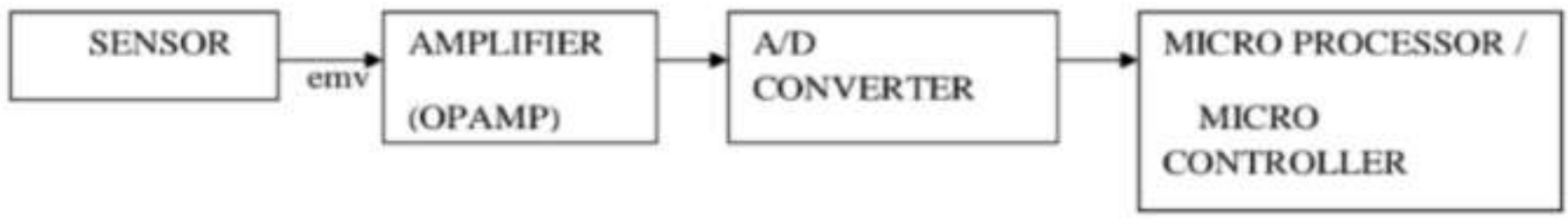
2. AMPLIFING

Signal amplification performs 2 important functions.

a. Increases the resolution of the input signal.

b. Increases its signal to noise ratio .

The output of an electronic temperature sensor which is in the range of mili volt is too low for an A/D convertor to process directly. So it is necessary to use an amplifier to bring the voltage level to that required by the ADC.



Commonly used amplifier on signal conditioning are S/H amplifier , peak detectors, log amplifier ,antilog amplifier etc.

3. ISOLATION

Signal isolation must be used in order to pass the analog signal from the source to the source to the measurement device without a physical connection.

It is important to isolate the expensive equipment used for signal processing after conditioning from the sensor.

There are 2 types of isolation

A. Magnetic

B. Optic

A. Magnetic

Magnetic isolation transforms the signals from voltage to a magnetic field allowing the signal to be transmitted without a physical connection.

B. Optic

Optic isolation takes an electronic signal and modulates its to a signal coded by light transmission (optical encoding) .Which is there used for the next stage of processing.

Types of devices that used signal conditioning are isolation amplifier ,multiplexer , A/D convertor , D/A convertor , inverter etc.

It is mostly used for data acquisition in which sensor signal must be normalised and filter to level suitable for analog frequency to voltage convertor, voltage to frequency convertor ,current to voltage convertor etc.

4. COMPARATOR

Sometimes there is no need to send the entire range of voltage from a sensor to the analog to digital convertor.

A circuit called comparator is used which takes an analog sensor voltage and compares it to a threshold voltage, Vth.

If the sensor voltage is greater than the threshold the output of the circuit is maximum.

$V_{SO} > V_{th} \longrightarrow$ maximum

$V_{SO} < V_{th} \longrightarrow$ minimum

If the sensor voltage is less than the threshold the output of the circuit is minimum.

Registers of Intel 8086

The Intel 8086 contains the following register.

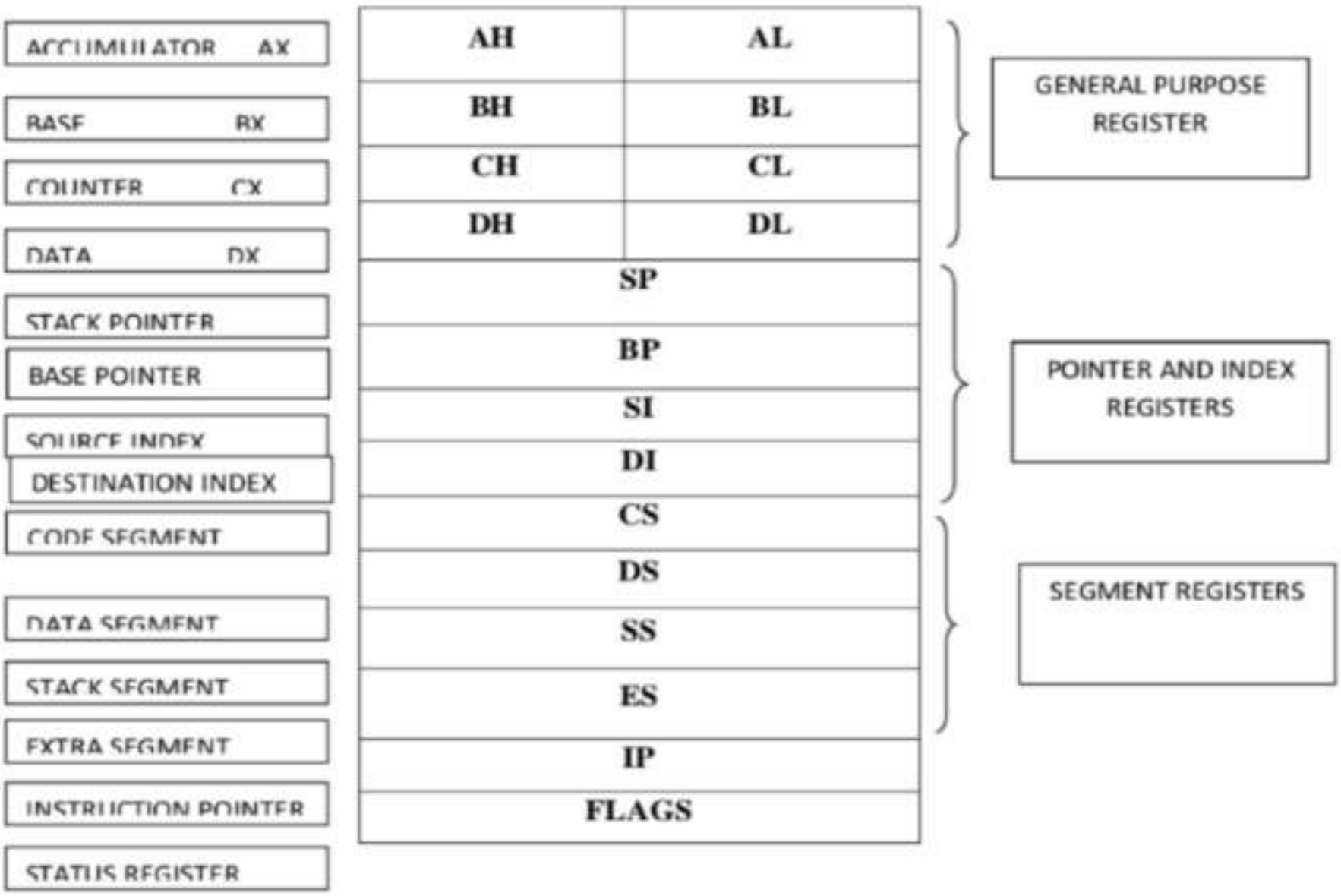
(i) General purpose Registers
(ii) Pointer and Index Registers
(iii) Segment Registers
(iv) Instruction Pointer
(v) Status Flags

General purpose Registers

There are four 16-bit general purpose register: AX, BX, CX and DX. Each of these 16-bit registers are further subdivided in to two 8-bit register as shown below.

16-bit register	8-bit high order register	8-bit low order register
AX	AH	AL
BX	BH	BL
CX	CH	CL
DX	DH	DL

Register AX serves as an accumulator. Register BX,CX and DX are used as general purpose register in addition to serving as general purpose register they also serve as special purpose register . as a special purpose register BX serves as a base register for computation of memory address .in 8086 memory addresses are to be calculated using the contents of the segment register and effective memory address . These will be explained later on . Register CX is also used as a counter in case of multi-iteration instruction. When the content of CX becomes zero such instruction terminate the execution. DX register is also used for memory addressing when data are transferred between I/O port and memory using certain I/O instruction.



REGISTER ORGANIZATION OF INTEL 8086

Pointer and Index Register. The following four figures are in the group of pointer and index register.

1. Stack Pointer, SP
2. Base pointer , BP
3. Source index ,SI
4. Destination Index, DI

The function of SP is same as the function of stack pointer in Intel 8085. BP,SI and DI are used in memory address computation.

Segment Register. There four segment registers in 8086.

1.Code Segment Register , CS

2. Data Segment Register , DS

3. Stack Segment Register, SS

4. Extra Segment Register, ES

In an 8086 microprocessor-based system memory is divided in to the following four segments:

1 .Code Segment

2. Data Segment

3. Stack Segment

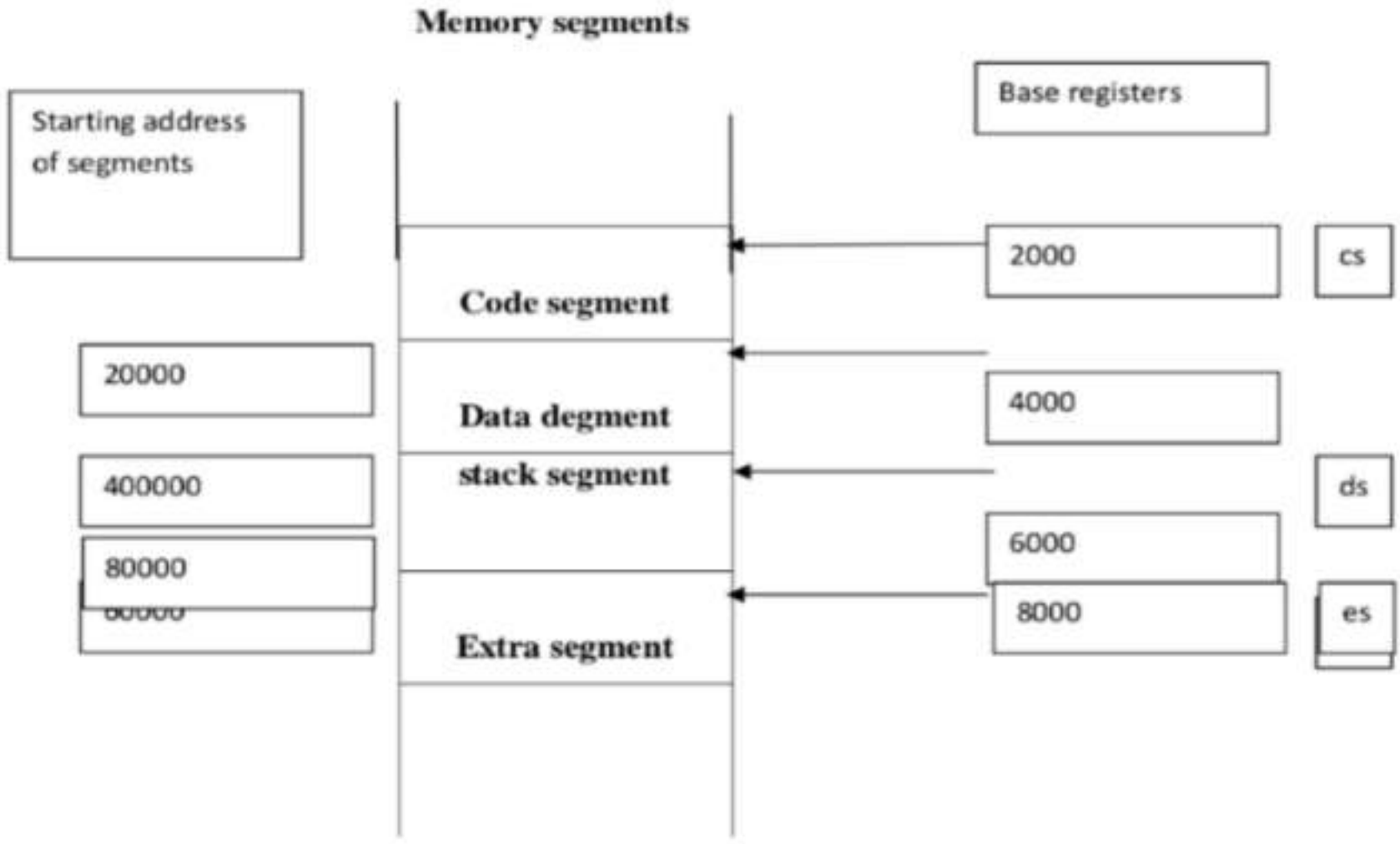
4. Extra Segment

The code segment of the memory holds instruction code of a program. The data variables and constant given in the program are held in the data segment of the memory. Stack segment holds address and data of subroutines. It also holds the contents of register and memory location given in PUSH instructions. Before attending an interrupt the microprocessor saves the contents of program counter on the stack. Also when CALL instruction is executed , before the execution of the subroutine the address of the next instruction of the program is saved on the stack. The extra segment holds the destination address of some data of certain string instruction, and so on.

A segment register point to the starting address of a memory segment currently being used. For example the code segment registers points to the starting address of the data segment, and so on. T he maximum capacity of a segment may be up to 64kbyte . The starting address of a segment is divisible by 16 . The segment shown in the figure is currently used segment. There more number of such segment to make the total memory capacity 1Mbyte.

The 8086 instruction specify 16-bit memory address. The actual addresses are of 20 bits. They are calculated using the contents of the segment register and effective memory address. The effective memory address is computed in a variety of ways. It depends on the such as PUSH,POP,CALL or RET , The content of the stack pointer(SP) and the content of the stack segment register(SS) are used to complete the stack of the location to be accessed. The index register SI and DI together with segment register DS and ES are used to perform string operations. The source addresses for string operation are computed using the content of SI and

DS . The destination addresses for string operations are computed using the contents of DI and ES.



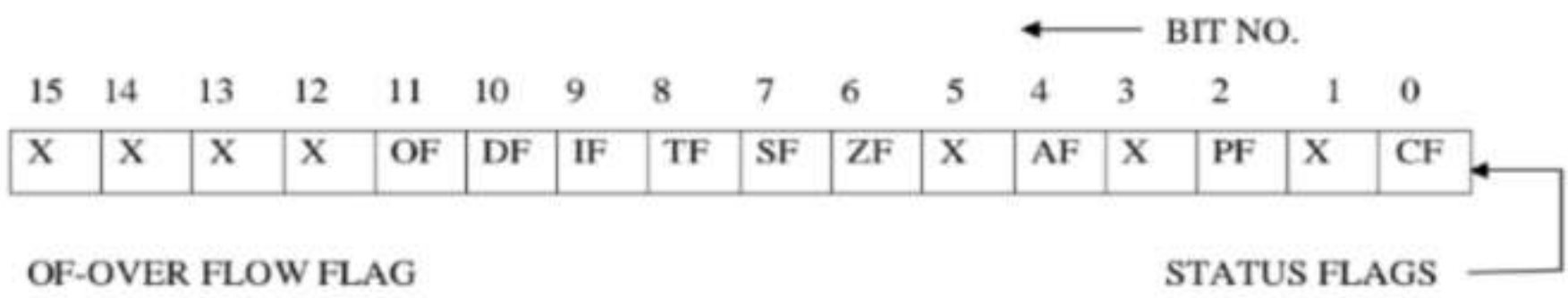
[Memory segments]

Instruction pointer (IP).:

The instruction pointer in the 8086 microprocessor act as a program counter. It point to the address of the next instruction to be executed. Its content is automatically incremented when the execution of a program proceeds further. The content of the instruction pointer (IP) and the content of the code segment register (CS) are used to compute the memory address of the instruction code to be fetched. This is done during instruction fetch operation.

Status Register :

The 8086 contains a 16-bit status register. It also called flag register or program status word (PSW) . There are 9 status flags that are: over flow flag, Direction flags, Input enable flag, Trap flag, Carry flag, parity flag, Auxiliary carry flag, zero flag, sign flag. Out of nine flag 6 are condition flag and three are control flags. These six condition flag are carry, auxiliary carry, zero, sign and parity and overflow flag. The flag are set/reset by the processor after the execution of an arithmetic and logic operation. The three control flag are trap (or trace) interrupt and directional flag. These flags are set/reset by the programmer as the required by certain instruction in the program. The overflow, trap, interrupt and directional flags are new. Other flags are same as those available in Intel 8085. The overflow flag is set to 1 if the result of a signed operation become out of range other it is reset it is made 0.



OF-OVER FLOW FLAG

DF-DIRECTION FLAG

IF-INTERRUPT ENABLE FLAG

TF-TRAP FLAG

SF-SIGN FLAG

ZF-ZERO FLAG

AF-AUXILIARY CARRY FLAG

PF-PARITY FLAG

CF-CARRY FLAG

[Status flags of Intel 8086]

When the trap flag (TF) is set to 1 a program can be run in single-step mode. The interrupt flag (IF) is set to 1 to enable INTR of 8086. If it is 0, INTR is disabled. It is set by STI instruction and declared by CLI instruction, IF is automatically cleared when an interrupt is recognised. It disabled ITNTR. IRET used at the end of interrupt service sub routine (ISS) restore IF flag in the state in the it was before interrupt occurred.

The directional flag DF is used in string operation .It can be set by STD instruction and cleared by CLD instruction. If it is set to 1, string bytes are accessed from higher memory address to lower memory address. When it is set to 0 the string bytes are accessed from lower memory address to higher memory address. For MOVES instruction, if DF is set to 1, the content of index register SI and DI are automatically decremented by the processor to access the string from the highest memory address down to the lowest memory address. If DF is made zero, SI and DI are automatically incremented to access the string starting with the lowest address.

Addressing Modes of Intel 8086

The way by which an operand is specify for instruction is called addressing mode. Intel 8086 has 8 addressing mode.

1. Register Addressing
2. Immediate Addressing
3. Direct Addressing
4. Register Indirect Addressing
5. Based Addressing
6. Indexed Addressing
7. Based Indexed Addressing
8. Based Indexed with Displacement Addressing

1.REGISTER ADDRESSING

- Here the operand is placed in one of the 16-bit or 8-bit general purpose registers.
- Example- MOV AX,CX (move 16-bit data in base register to accumulator)

2.IMMEDIATE ADDRESSING

- Here the operand is specified in the instruction itself.
- Example-MOV AL, 35H (move immediate data 35H to 8-bit register A)

MOV BX,0305H (move 16-bit data0305 to register base.

The remaining 6 addressing mode specify the location of an operand which is placed in memory. When an operand is stored in memory location, how far the operand's memory location is within a memory segment from the starting address of the segment, is called offset or effective address (EA).

An offset is determined by adding any combination of three address elements: displacement, base and index.

Displacement:- It is an 8-bit or 16-bit immediate value given in the instruction.

Base:- It is the content of the base register, BX or BP.

Index:- It is the content of the index register, SI or DI.

The Combination of these three address elements give six memory addressing modes as described below.

3. DIRECTADDRESSING

In this mode the operand's offset is given in the instruction as an 8-bit or 16-bit displacement element.

- Example:- 1.ADD AL,[0301]
  The contents of memory 0301 is added to the content of AL ,and the result is placed in AL .
- 2. ADD [0301],AX
  This instruction adds the content of AX to the content of memory location 0301 and 0302.

4. REGISTER INDIRECT ADDRESSING

The operand's offset is placed in any one of the registers BX,BP,SI or DI as specified in the instruction.

Example:- MOV AX,[BX]

- This instruction moves the content of memory location addressed by the register BX to the register AX.
- For examples, BX contains 0301,and the content of 0301 is 53H and the content of next memory location is 95H.The 9553 will move to AX.
-

5.BASED ADDRESSING

The operand's offset is the sum of an 8-bit or 16-bit displacement and the content of the base register BX or BP. BX is used as a base register for data segment, and BP is used as a base register for stack segment.
Offset=[BX +8-bit or 16-bit displacement]

- Examples are:-

MOV AL, [BX+05]; an example of an 8-bit displacement.

Suppose ,the register BX contain 0301.The offset will be 0301+05=0306.The content of the memory location 0306 will move to AL.

- MOV AL, [BX+1346H]; an example of 16-bit displacement.
  If [BX] =0301.The offset =0301+1346=1647H
  The content of 1647H will move to AL.

6. INDEXED ADDRESSING

The operand's offset is the sum of the content of an index register SI or DI and an 8-bit or 16 bit displacement.

Offset= [SI or DI+8-bit or 16-bit displacement]

- Examples are:
  MOV AX, [SI+05] ; an example of 8-bit displacement.
  MOV AX, [SI+1528H] ; an example of 16-bit displacement.

7. BASED INDEXED ADDRESSING

The operand's offset is the sum of the content of a base register BX or BP and an index register SI or DI.BX is used as a base register for data segment, and BP is used as a base register for stack segment.

Offset= [BX or BP] + [SI or DI]

- Examples are:
  ADD AX, [BX+SI]
  MOV CX, [BX+SI]

8. BASED INDEXED WITH DISPLACEMENT

In this mode of addressing the operand's offset given by

Offset= [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement

- Examples are:
  MOV AX, [BX+SI+05]; an example of 8-bit displacement
  MOV AX,[BX+SI+1235H]; an example of 16-bit displacement.

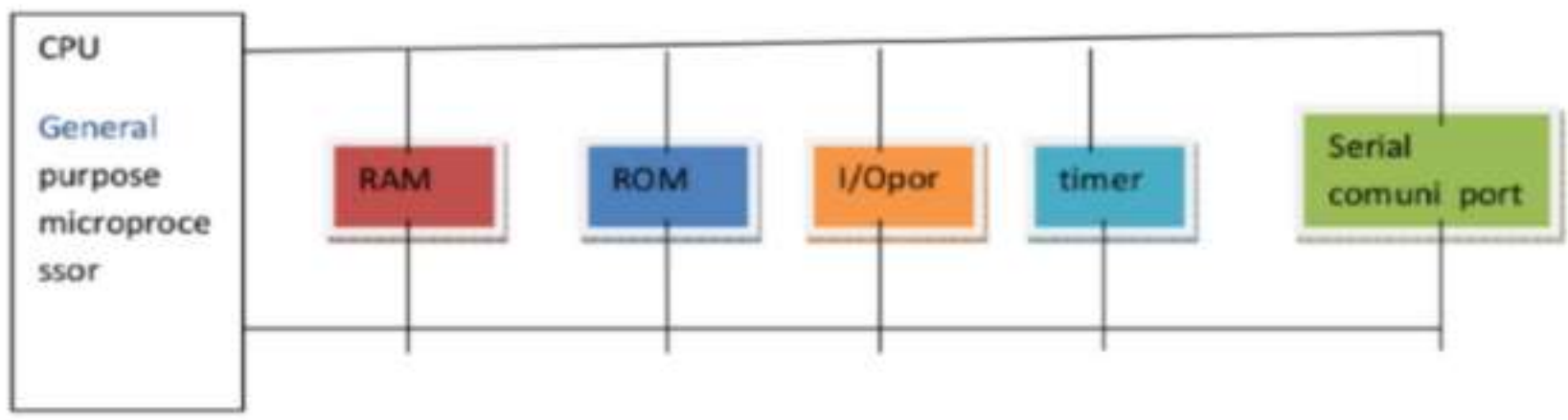
Difference between MICROPROCESSOR and MICROCONTROLLER

- Microprocessor contains no RAM,ROM and no I/O port on the chip itself.it is only a CPU.
- They are commonly refered to as general purpose microprocessor.
- The system designer using a general purpose microprocessor such as a Pentium must add RAM,ROM,I/O ports and timer externally to make them fuctional.

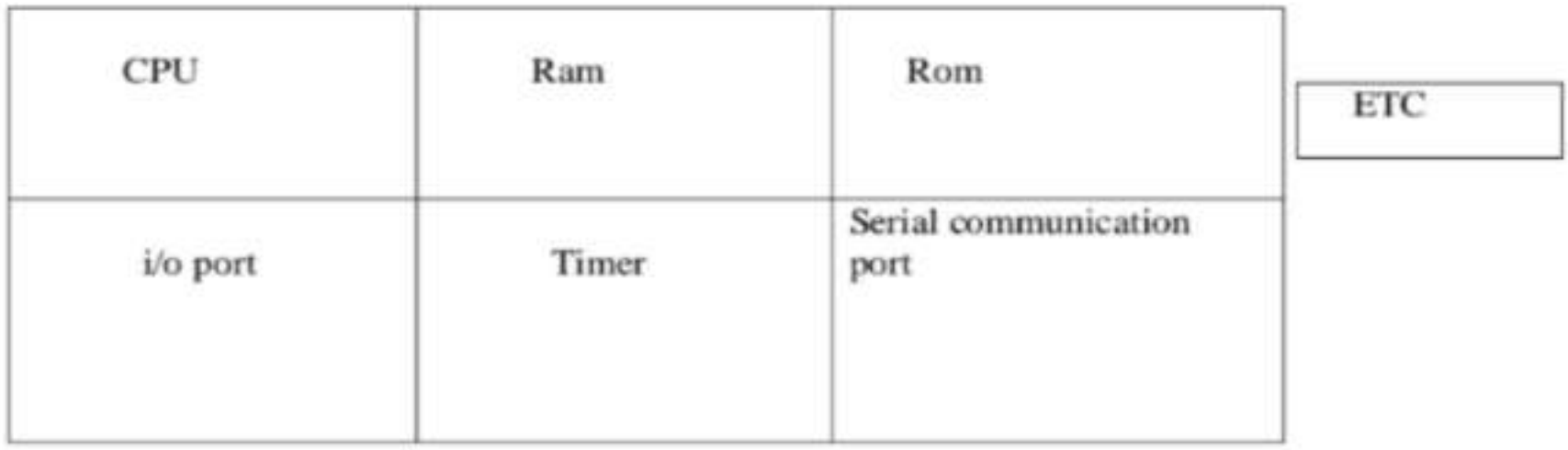
MICROCONTROLLER

- Microcontroller has a cpu(microprocessor) in addition to a fixed amount of RAM,ROM,I/O port and timer are all on a single chip.
- So the designer can't add any external memory, i/o port or timer to it.
- The fixed amount of on chip RAM,ROM and the no of i/o ports in microcontrollers make them ideal for any applications in which cost and space are critical.
  Ex-TV remote control
- These application open required some i/o operation to read signal and turn on/off certain bits so these processors are called IBP (Itty bitty Processor)
- In 8051 it allows to set or reset indivisual pins of i/o ports and status registers .it also provides bytes addressing so it is called as bit and byte processor.

General Purpose Microprocessor System



Microcontroller

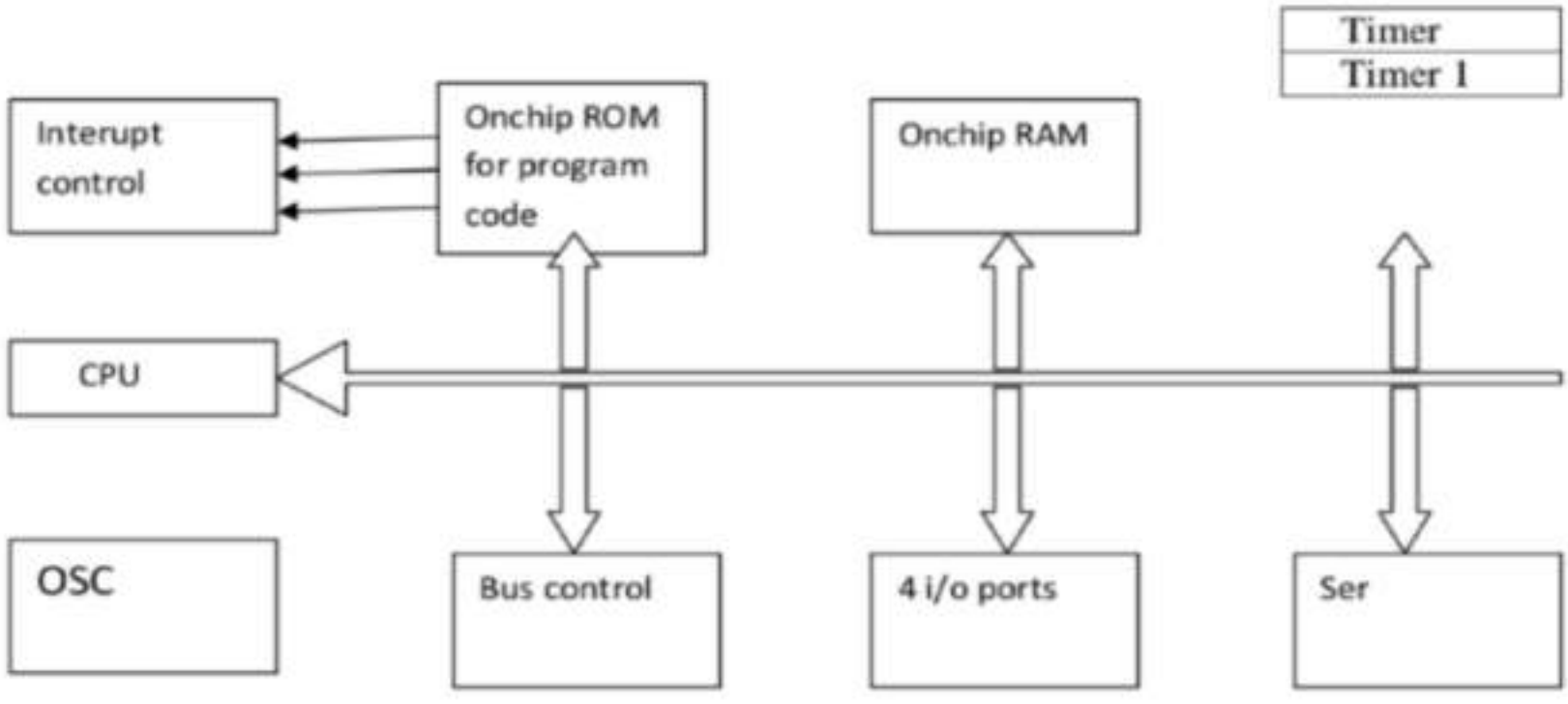


- Disadvantages of the microcontroller is that if the system designer requires more memory or more no. of i/o ports cant add them externally.
- Microprocessors and Microcontrollers are widely used in embedded system product .
- An embedded product uses a microprocessor or microcontroller to do one task and one task only.
  Ex- a printer is an embedded system in which the processor inside it performs only one task i.e breaking the data and print it.
- In an embedded system there is only one application software that is typically burned into run.
  Ex-A pc is connected to various embedded products such as keyboard,printer,modem,disk controller,sound card,cd driver etc.
- Each of the peripheral has the microcontroller inside it that performs only one task.
  Ex-Inside every mouse there is a microcontroller that performs the task of finding the mouse position and sending it to the pc.
- Some embedded products used in homes i.e TV,telephones,remote controller,camera etc.

Overview of 8051 family

- In 1981 Intel corporation introduced an8-bit microcontroller called 8051.
- This microcontroller has 120 bytes of RAM 4-kilobytes of microchip ROM,two timers ,one serial port and four i/o ports are 8-bit wide all on a single chip.
- It is an 8- bit processor meaning that the CPU can work pon on only 8 bit of data at a time.
- Intel refers to 8051 MCS-51
- Other members of 8051 family they are 8052 and 8053.

Data larger than the 8-bit has to be broken into 8 bit process to be processed by the CPU.



- **8052 microcontroller**

It has all the standard features of 8051 along with extra 120 bytes of RAM and an extra timer or 8052 has 8 kilobytes of onchip program ROM ,256 bytes of RAM,4 i/o ports and 6 timers.

- **8031 microcontroller**

This chip refered to as a ROM less 8051.Since it has 0 kilobytes of onchip ROM.to use this chip the designer must add an external rom chip.

Comparison of 8051 family members

Feature	**8051**	**8052**	**8031**
ROM(onchip program space in bytes)	4k	8k	0k
RAM(bytes)	128	256	128
Timers	2	3	2
i/o pins	32	32	32
Serial ports	1	1	1

REFERENCES :

1. B. Ram, "*Fundamentals of Microprocessors and Microcomputers*", Dhanpat Rai Publications.

2. A.K.Ray and K.M.Bhurchandi – "*Advanced Microprocessors & Peripherals*" Tata McGraw Hill.

3. M.A. Mazidi and J.G. Mazidi, "*The 8051 Microcontroller and Embedded Systems*", Pearson Education, India.

4. Ramesh S. Gaonkar, "*Microprocessor Architecture, Programming and Application with the 8085*", Fourth Edition, Penram International Publishing (India).

5. D.V. Hall, "*Microprocessors and Interfacing*",2nd Edition McGraw-Hill Book Company.