GOVERNMENT OF TAMILNADU
DIRECTORATE OF TECHNICAL EDUCATION
CHENNAI – 600 025

STATE PROJECT COORDINATION UNIT

Diploma in Electronics and Communication Engineering

Course Code: 1040

M – Scheme

e-TEXTBOOK
on
DIGITAL ELECTRONICS
for
IV Semester DECE

Convener for ECE Discipline:

Dr.M.Jeganmohan,
Principal,
Government Polytechnic College,
Uthapanaickanoor,
Usilampatti – 625 536.

Team Members for Digital Electronics:

Mr.S.Balakrishnan,
Lecturer (SS) / ECE,
228, ArasanGanesan Polytechnic College,
Sivakasi – 626 130.

Mr.L.Murugesan,
Instructor / ECE,
228, ArasanGanesan Polytechnic College,
Sivakasi – 626 130.

Validated By

Dr. S.MohammedMansoorRoomi,
Assistant Professor / ECE,
Thiagarajar College of Engineering,
Madurai – 625 015.

CONTENTS

UNIT – I

NUMBER SYSTEM AND BOOLEAN ALGEBRA

UNIT –II

COMBINATIONAL CIRCUITS

UNIT – III

SEQUENTIAL CIRCUITS

UNIT - IV

MEMORY DEVICES

UNIT – V

MICROPROCESSOR – 8085

SYLLABUS

UNIT – I :NUMBER SYSTEM AND BOOLEAN ALGEBRA

Binary, Octal, Decimal, Hexadecimal - Conversion from one to another. Binary codes – BCD code, Gray code, Excess 3 code.

Boolean Algebra- Boolean postulates and laws- De-Morgan's theorem- Simplification of Boolean expressions using Karnaugh map (up to 4 variables-pairs, quad, octets)- Don't care conditions and constructing the logic circuits for the Boolean expressions.

LOGIC GATES AND DIGITAL LOGIC FAMILIES:

Gates – AND, OR, NOT, NAND, NOR, EX-OR, EX-NOR - Implementation of logic functions using gates - Realization of gates using universal gates- Simplification of expression using Boolean techniques- Boolean expression for outputs.

Digital logic families –Fan in, Fan out, Propagation delay - TTL,CMOS Logics and their characteristics - comparison and applications -Tristate logic.

UNIT – II :COMBINATIONAL CIRCUITS

Arithmetic circuits - Binary – Addition, subtraction, 1's and 2's complement - Signed binary numbers- Half Adder- Full Adder- Half Subtractor - Full Subtractor- Parallel and serial Adders- BCD adder.

Encoder and decoder – 3 to 8 decoder, BCD to seven segment decoder- Multiplexer- basic 2 to 1 MUX, 4 to 1 MUX, 8 to 1 MUX - applications of the MUX – Demultiplexer - 1 to 2 demultiplexer, 1 to 4 demultiplexer, 1 to 8 demultiplexer - Parity Checker and generator.

UNIT – III :SEQUENTIAL CIRCUITS

FLIP FLOPS – SR, JK, T, D FF, JK- MS FF - Triggering of FF – edge & level, Counters – 4 bit Up - Down Asynchronous / ripple counter - Decade counter- Mod 3, Mod 7 counter.

4 bit Synchronous Up - Down counter - Johnson counter, Ring counter

REGISTERS

4-bit shift register- Serial IN Serial OUT- Serial IN Parallel OUT - Parallel IN Serial OUT- Parallel IN Parallel OUT

UNIT – IV :MEMORY DEVICES

Classification of memories - RAM organization - Address Lines and Memory Size- Read /write operations- Static RAM - Bipolar RAM cell- Dynamic RAM- SD RAM- DDR RAM.

Read only memory – ROM organization- Expanding memory- PROM- EPROM- and EEPROM - Flash memory- Anti Fuse Technologies.

UNIT – V :MICROPROCESSOR – 8085

Evolution of microprocessor 8085 – Architecture of 8085-

Instruction sets- Addressing modes - Memory mapped I/O and I/O mapped I/O and its Comparison.

Machine cycle – Opcode fetch - memory read- memory write- I/O read, I/O write - Instruction cycle (Timing diagram) for MOV r1, r2 instruction.

Interrupts (types & Priorities)

UNIT - I

NUMBER SYSTEM AND BOOLEAN ALGEBRA

1.1 NUMBER SYSTEMS AND BINARY CODES

1.1.1 Number systems

Number systems other than the familiar **decimal (base 10) number system** are used in the computer field and digital systems. Digital computers internally use the **binary (base 2) number system** to represent data and perform arithmetic calculations. The **hexadecimal (base 16) number system** is a shorthand method of working with binary numbers. The **octal (base 8)** number system is also less commonly used in digital computers.

The different number systems are:

1. Decimal number system
2. Binary number system
3. Octal number system and
4. Hexadecimal number system

Decimal number system

We use the decimal number system in our day-to-day life for representing numbers. It is a positionalnumber system. It has ten symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The base (or radix, or weight) of the decimal number system is 10. The place value of each position is given below:

10^{3}	10^{2}	10^{1}	10^{0}	.	10^{-1}	10^{-2}	10^{-3}
1000	100	10	1	.	0.1	0.01	0.001

Example:

$$\begin{aligned} 3472.65 &= 3 \times 10^{3} + 4 \times 10^{2} + 7 \times 10^{1} + 2 \times 10^{0} + 6 \times 10^{-1} + 5 \times 10^{-2} \\ &= 3 \times 1000 + 4 \times 100 + 7 \times 10 + 2 \times 1 + 6 \times 0.1 + 5 \times 0.01 \end{aligned}$$

Symbols used: 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9.
Base (or radix, or weight): 10
Place value: ….. 10^{3}, 10^{2}, 10^{1}, 10^{0} . 10^{-1}, 10^{-2}, 10^{-3} …..

Binary number system

Binary number system is used in computers and digital circuits for representing numbers. It has only two symbols: 0 and 1. The base (or radix, or weight) of the binary number system is 2. The place value of each position is given below:

2^{3}	2^{2}	2^{1}	2^{0}	.	2^{-1}	2^{-2}	2^{-3}
8	4	2	1	.	0.5	0.25	0.125

Example:

$$\begin{aligned} 1011.01 &= 1 \times 2^{3} + 0 \times 2^{2} + 1 \times 2^{1} + 1 \times 2^{0} + 0 \times 2^{-1} + 1 \times 2^{-2} \\ &= 1 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1 + 0 \times 0.5 + 1 \times 0.25 \\ &= 11.25 \text{ (in decimal)} \end{aligned}$$

Symbols used: 0 and 1
Base (or radix, or weight): 2
Place value: ….. 2^3, 2^2, 2^1, 2^0 . 2^{-1}, 2^{-2}, 2^{-3} …..

Octal number system

Octal numbers are used to represent binary numbers. It has eight symbols: 0, 1, 2, 3, 4, 5, 6 and 7. The base (or radix, or weight) of the octal number system is 8. The place value of each position is given below:

8^3	8^2	8^1	8^0	.	8^{-1}	8^{-2}	8^{-3}
512	64	8	1	.	0.125	0.015625	0.001953125

Example:

$$\begin{aligned}72.2 &= 7 \text{ x } 8^1 + 2 \text{ x } 8^0 + 2 \text{ x } 8^{-1}\\ &= 7 \text{ x } 8 + 2 \text{ x } 1 + 2 \text{ x } 0.125\\ &= 58.25 \text{ (in decimal)}\end{aligned}$$

Symbols used: 0, 1, 2, 3, 4, 5, 6 and 7
Base (or radix, or weight): 8
Place value: ….. 8^3, 8^2, 8^1, 8^0 . 8^{-1}, 8^{-2}, 8^{-3} …..

Hexadecimal number system

Hexadecimal numbers are used to represent binary numbers. It has sixteen symbols: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, E and F. The base (or radix, or weight) of the hexadecimal number system is 16. Here the decimal number for ‘A’ is 10, ‘B’ is 11, ‘C’ is 12, ‘D’ is 13, ‘E’ is 14 and ‘F’ is 15. The place value of each position is given below:

16^3	16^2	16^1	16^0	.	16^{-1}	16^{-2}	16^{-3}
4096	256	16	1	.	0.0625	0.00390625	0.000244140625

Example:

$$\begin{aligned}3C.A &= 3 \text{ x } 16^1 + 12 \text{ x } 16^0 + 10 \text{ x } 16^{-1}\\ &= 3 \text{ x } 16 + 12 \text{ x } 1 + 10 \text{ x } 0.0625\\ &= 60.625 \text{ (in decimal)}\end{aligned}$$

Symbols used: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F
Base (or radix, or weight): 16
Place value: ….. 16^3, 16^2, 16^1, 16^0 . 16^{-1}, 16^{-2}, 16^{-3} …..

1.1.2 Conversion

Decimal to binary conversion

i) Integer numbers

The “double dabble” method is used to convert decimal number to binary number. In this method, the decimal number is divided by 2 repeatedly and the remainders are noted after each division. We have to continue until we get zero in the quotient. The remainders are taken from bottom to top to get the binary number.

Example: To convert the decimal number 27 to binary number

27 ÷ 2 = 13 → Remainder 1

13 ÷ 2 = 6 → Remainder 1

6 ÷ 2 = 3 → Remainder 0

3 ÷ 2 = 1 → Remainder 1

1 ÷ 2 = 0 → Remainder 1

Therefore, $27_{10} = 11011_2$

ii) Fractional numbers

The decimal number is multiplied by 2 repeatedly and the carries integer part (whole numbers) is noted after each multiplication. We have to continue until we get zero in the fraction. The process may be stopped after six places. The carries are taken from top to bottom to get the binary number.

Example: To convert the decimal number 0.85 to binary number.

0.85 x 2 = 1.7 = 0.7 with carry → 1

0.7 x 2 = 1.4 = 0.4 with carry → 1

0.4 x 2 = 0.8 = 0.8 with carry → 0

0.8 x 2 = 1.6 = 0.6 with carry → 1

0.6 x 2 = 1.2 = 0.2 with carry → 1

0.2 x 2 = 0.4 = 0.4 with carry → 0

Therefore, $0.85_{10} = 0.110110_2$

Decimal to octal conversion

i) Integer numbers

The “octal dabble” method is used to convert decimal number to octal number. In this method, the decimal number is divided by 8 repeatedly and the remainders are noted after each division. We have to continue until we get zero in the quotient. The remainders are taken from bottom to top to get the octal number.

Example: To convert the decimal number 175 to octal number

175 ÷ 8 = 21 → Remainder 7
21 ÷ 8 = 2 → Remainder 5
2 ÷ 8 = 0 → Remainder 2

Therefore, $175_{10} = 257_8$

ii) Fractional numbers

The decimal number is multiplied by 8 repeatedly and the carries (ie.) integer part (whole numbers) is noted after each multiplication. We have to continue until we get zero in the fraction. The process may be stopped after three places. The carries are taken from top to bottom to get the octal number.

Example: To convert the decimal number 0.23 to octal number.

0.23 x 8 = 1.84 = 0.84 with carry → 1
0.84 x 8 = 6.72 = 0.72 with carry → 6
0.72 x 8 = 5.76 = 0.76 with carry → 5

Therefore, $0.23_{10} = 0.165_8$

Decimal to hexadecimal conversion

i) Integer numbers

The "hex dabble" method is used to convert decimal number to hexadecimal number. In this method, the decimal number is divided by 16 repeatedly and the remainders are noted after each division. We have to continue until we get zero in the quotient. The remainders are taken from bottom to top to get the hexadecimal number.

Example: To convert the decimal number 2479 to hexadecimal number

2479 ÷ 16 = 154 → Remainder 15 → F
154 ÷ 16 = 9 → Remainder 10 → A
9 ÷ 16 = 0 → Remainder 9 → 9

Therefore, $2479_{10} = 9AF_{16}$

ii) Fractional numbers

The decimal number is multiplied by 16 repeatedly and the carries (ie.) integer part (whole numbers) is noted after each multiplication. We have to continue until we get zero in the fraction. The process may be stopped after three places. The carries are taken from top to bottom to get the hexadecimal number.

Example: To convert the decimal number 0.23 to hexadecimal number.

0.23 x 16 = 3.68 = 0.68 with carry → 3 → 3
0.68 x 16 = 10.88 = 0.88 with carry → 10 → A
0.88 x 16 = 14.08 = 0.08 with carry → 14 → E

Therefore, $0.23_{10} = 0.3AE_{16}$

Binary to decimal conversion

i) Integer numbers

The given binary number can be converted to decimal number by using the following procedure:

Step 1 : Write the binary number
Step 2 : Write the place value (weight) directly under the binary number
Step 3 : If the binary digit is zero, cross out the decimal weight
Step 4 : Add the remaining weights to get the decimal number.

Example: To convert the binary number 1011 to decimal number

Step 1 : 1 0 1 1 (Binary number)
Step 2 : 8 4 2 1 (Place value)
Ste$\cancel{p}$ 3 : 8 4 2 1 (Cross out)
Step 4 : 11 (Add, to get the decimal number)

Therefore, $1011_2 = 11_{10}$

ii) Fractional numbers

The same procedure used for the integer numbers can be used for fractional numbers also.

Example: To convert the binary number 0.101 to decimal number

Step 1 : 1 0 1 (Binary number)
Step 2 : 0.5 0.25 0.125 (Place value)
Step 3 :$\cancel{0}$.5 0.25 0.125 (Cross out)
Step 4 : 0.625 (Add, to get the Decimal number)

Therefore, $0.101_2 = 0.625_{10}$

Binary to octal conversion

The binary numbers for the octal numbers from 0 to7 are given in the following table:

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

The following procedure can be used for converting the binary number to octal number:

Step 1 : Group the binary digits in threes, starting at the binary point.
Step 2 : Convert each group of three to its octal equivalent

Example: To convert the binary number 1101110.101011 to octal number

Step 1 : 001 101 110 . 101 011 (Group)
Step 2 : 1 5 6 . 5 3 (Octal equivalent)

Therefore, $1101110.101011_2 = 156.53_8$

Binary to hexadecimal conversion

The binary numbers for the hexadecimal numbers from 0 to F are given in the following table:

Binary	Hexadecimal
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

The following procedure can be used for converting the binary number to octal number:

Step 1 : Group the binary digits in fours, starting at the binary point.
Step 2 : Convert each group of four to its hexadecimal equivalent

Example: To convert the binary number 1101110.101011 to hexadecimal number

Step 1 : 0110 1110 . 1010 1100 (Group)
Step 2 : 6 E . A C (Hexadecimal equivalent)

Therefore, $1101110.101011_2 = 6E.AC_{16}$

Octal to decimal conversion

i) Integer numbers

The given octal number can be converted to decimal number by using the following procedure:

Step 1 : Write the octal number
Step 2 : Write the place value (weight) directly under the octal number
Step 3 : Multiply the octal number by the place value
Step 4 : Add the values to get the decimal number.

Example: To convert the octal number 257 to decimal number

Step 1 :	2	5	7	(Octal number)
Step 2 :	64	8	1	(Place value)
Step 3 :	128	40	7	(Multiply)
Step 4 :	175			(Add, to get the Decimal number)

Therefore, $257_8 = 175_{10}$

ii) Fractional numbers

The same procedure used for the integer numbers can be used for fractional numbers also.

Example: To convert the octal number 0.41 to decimal number

Step 1 :	4	1	(Octal number)
Step 2 :	0.125	0.015625	(Place value)
Step 3 :	0.5	0.015625	(Multiply)
Step 4 :	0.515625175		(Add, to get the Decimal number)

Therefore, $0.41_8 = 0.515625175_{10}$

Octal to binary conversion

The following procedure can be used for converting the octal number to binary number:

Step 1 : Write the octal number
Step 2 : Write the three digit equivalent binary number

Example: To convert the octal number 34.56 to binary number

Step 1 :	3	4	.	5	6	(Octal number)
Step 2 :	011	100	.	101	110	(Binary equivalent)

Therefore, $34.56_8 = 11100.10111_2$

Octal to hexadecimal conversion

The following procedure can be used for converting the octal number to binary number:

Step 1 : First convert the octal number to binary number
Step 2 : Then, convert the binary number to hexadecimal number (procedure already given)

Example: To convert the octal number 34.56 to binary number

Step 1 :	3	4	.	5	6	(Octal number)
Step 2 :	011	100	.	101	110	(Binary equivalent)
Step 3 :	0001	1100	.	1011	1000	(Group of 4-bits)
Step 4 :	1	C	.	B	8	(Hexadecimal number)

Therefore, 34.56_8 = $1C.B8_{16}$

Hexadecimal to decimal conversion

i) Integer numbers

The given hexadecimal number can be converted to decimal number by using the following procedure:

Step 1 : Write the hexadecimal number
Step 2 : Write the place value (weight) directly under the hexadecimal number
Step 3 : Multiply the hexadecimal number by the place value
Step 4 : Add the values to get the decimal number.

Example: To convert the hexadecimal number 8E6 to decimal number

Step 1 :	8	E	6	(Hexadecimal number)
Step 2 :	256	16	1	(Place value)
Step 3 :	2048	224	6	(Multiply)
Step 4 :	2278			(Add, to get the Decimal number)

Therefore, $8E6_{16}$ = 2278_{10}

ii) Fractional numbers

The same procedure used for the integer numbers can be used for fractional numbers also.

Example: To convert the hexadecimal number 0.39 to decimal number

Step 1 :	3	9	(Hexadecimal number)
Step 2 :	0.0625	0.00390625	(Place value)
tep 3 :	0.18755	0.03515625	(Multiply)
Step 4 :	0.22270625		(Add, to get the Decimal number)

Therefore, 0.39_{16} = 0. 0.22270625_{10}

Hexadecimal to binary conversion

The following procedure can be used for converting the hexadecimal to binary number:

Step 1 : Write the Hexadecimal number
Step 2 : Write the four digit equivalent binary number

Example: To convert the hexadecimal number 7A.2D to binary number

Step 1 : 7 A . 2 D (Hexadecimal number)
Step 2 : 0111 1010 . 0011 1101 (Binary equivalent)

Therefore, $7A.2D_{16} = 1111010.00111101_{2}$

Hexadecimal to Octal conversion

The following procedure can be used for converting the hexadecimal number to octal number:

Step 1 : First convert the hexadecimal number to binary number
Step 2 : Then, convert the binary number to octal number (procedure already given)

Example: To convert the hexadecimal number 7A.2D to octal number

Step 1 : 7 A . 2 D (Hexadecimal number)
Step 2 : 0111 1010 . 0011 1101 (Binary equivalent)
Step 3 : 001 111 010 . 001 111 010 (Group of 3-bits)
Step 4 : 1 7 2 . 1 7 2 (Octal number)

Therefore, $7A.2D_{16} = 172.172_{8}$

1.1.3 Binary codes

We use the decimal code to represent numbers. Digital electronic circuits in computers and calculators use mostly the binary code to represent numbers. Many other special codes are used in digital electronics to represent numbers, letters, and punctuation marks and control characters. These special codes are generally called binary codes. Some of the special binary codes are BCD code, Gray code, and Excess 3 code.

BCD Code (Natural BCD code)

BCD is an abbreviation for Binary-Coded Decimal. In this code, decimal digits 0 through 9 are represented by their binary equivalents using 4 bits (binary digits) individually. For example, the decimal number 429 is expressed in BCD as follows:

Decimal number : 4 2 9
BCD number : 0100 0010 1001

The lowest BCD digit is 0000 and the highest BCD digit is 1001. In general, BCD codes are also called 8421 code. The main advantage of 8421 code is that converting to and from decimal numbers is very easy.

Excess-3 Code

This is another form of BCD code. The code for each decimal digit is obtained by adding decimal number 3 to the natural BCD code. For example, decimal number 2 is coded as 0010 + 0011 = 0101 in Excess-3 code. It is a self-complementing code which is very much useful in performing subtraction operation in digital systems. The Excess-3 codes for the decimal numbers from 0 to 9 are given in the following table.

Decimal number	Natural BCD code	Excess-3 code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

Gray Code

It is a special binary code used in optical encoders. In this code, only one bit will change each time the decimal number is incremented. The BCD codes and Gray codes for the decimal numbers from 0 to 15 are given in the following table.

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

1.2 BOOLEAN ALGEBRA AND DE-MORGAN’S THEOREMS

1.2.1 Boolean Algebra

In the middle of the 19th century, an English mathematician George Boole developed rules for manipulations of binary variables, known as Boolean algebra. This is the basis for all digital systems like computers, calculators, etc. Binary variables can be represented by a letter symbol such as A, B, X, Y, etc. The variable can have only one of the two possible values at any time either 0 or 1. The following are the operators used in Boolean algebra.

Operator	Operation
=	Equal (Assignment)
+	OR (Logic addition)
.	AND (Logic multiplication)
$\overline{\quad}$(Bar)	NOT (Complement)

<u>Boolean postulates and laws</u>

<u>Theorems</u>

S.No	Theorem
1.	$\overline{\overline{A}}$ = A
2.	A ($\overline{A}$ + B) = A B
3.	A + A B = A
4.	(A + B) (A + $\overline{B}$) = A
5.	A B + $\overline{A\ B}$ = A
6.	A (A + B) = A
7.	A + A $\overline{B}$ = A + B
8.	A + B C = (A + B) (A + C)

<u>Laws of complementation (NOT Laws)</u>

S.No	Law
1.	$\overline{0}$ = 1
2.	$\overline{1}$ = 0
3.	If A = 0, $\overline{A}$ = 1
4.	If A = 1, $\overline{A}$ = 0
5.	$\overline{\overline{A}}$ = A

AND Laws

S.No	Law
1.	A . 0 = 0
2.	A . 1 = A
3.	A . A = A
4.	A .$\overline{A}$ = 0

OR Laws

S.No	Law
1.	A + 0 = A
2.	A + 1 = 1
3.	A + A = A
4.	A +$\overline{A}$ = 1

Commutative Laws

The commutative laws allow the change in position of an AND or OR variable.

S.No	Law
1.	A + B = B + A
2.	A . B = B . A

Associative Laws

The associative laws allow the grouping of variables.

S.No	Law
1.	A + (B + C) = (A + B) + C
2.	A . (B . C) = (A .B) . C

Distributive Laws

The distributive laws allow the factoring or multiplying out of expressions.

S.No	Law
1.	A . (B + C) = (A . B) + (A . C)
2.	A + (B . C) = (A + B) . (A + C)

1.2.2 DeMorgan's theorems

De Morgan contributed a lot for the Boolean algebra. Out of them, the following two theorems are very important. It allows transformation from a sum-of-products form to a products-of-sums form. De Morgan's theorems are also useful in simplifying Boolean equations.

First theorem

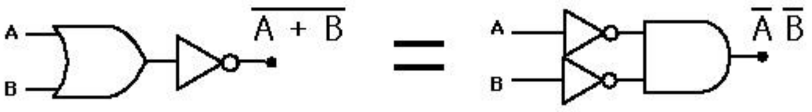
Statement:

The complement of a sum of variables is equal to the product of their complements.

Equation:

$$\overline{A + B} = \overline{A} \,.\, \overline{B}$$

Logic diagram :



Proof:

We have to show the left side equals the right side for all possible values of A and B. The following table proves the De Morgan's first theorem.

A	B	A + B	$\overline{A + B}$ (LHS)	$\overline{A}$	$\overline{B}$	$\overline{A} . \overline{B}$ (RHS)
0	0	0	1	1	1	1
0	1	1	0	1	0	0
1	0	1	0	0	1	0
1	1	1	0	0	0	0

In the above table, LHS = RHS. Hence, the De Morgan's first theorem is proved.

Second theorem

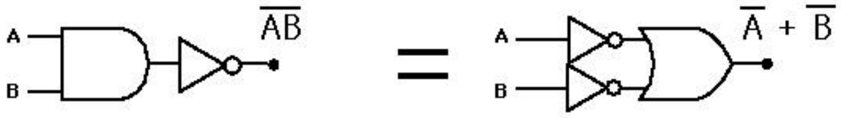
Statement:

The complement of a product of variables is equal to the sum of their complements.

Equation:

$$\overline{A \,.\, B} = \overline{A}+\overline{B}$$

Logic diagram :



Proof:

We have to show the left side equals the right side for all possible values of A and B. The following table proves the De Morgan's second theorem.

A	B	A . B	$\overline{A.B}$ (LHS)	$\overline{A}$	$\overline{B}$	$\overline{A}$ + $\overline{B}$(RHS)
0	0	0	1	1	1	1
0	1	0	1	1	0	1
1	0	0	1	0	1	1
1	1	1	0	0	0	0

In the above table, LHS = RHS. Hence, the De Morgan's second theorem is proved.

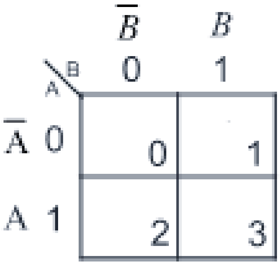
1.3 KARNAUGH MAP

Karnaugh map (K-map) is a graphical technique which provides a systematic method for simplifying Boolean expressions. In this technique, the information contained in truth table is represented in a map form called K-map.

1.3.1 Two-variable K-map

Follow the steps given below to draw a two-variable K-map.

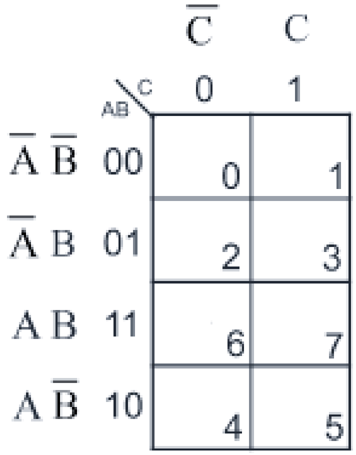
1. Draw a table with 2 rows and 2 columns with an extension line at the top left corner.
2. Mark A for the Rows and B for the Columns.
3. Put $\overline{A}$ and 0 for the first row. Put A and 1 for the second row.
4. Put $\overline{B}$ and 0 for the first column. Put B and 1 for the second column.
5. Put the numbers inside each box (at the bottom right corner) as 0, 1, 2 and 3 row wise.



Three-variable K-map

Follow the steps given below to draw a three-variable K-map.

1. Draw a table with 4 rows and 2 columns with an extension line at the top left corner.
2. Mark AB for the Rows and C for the Columns.
3. Put $\overline{AB}$ and 00 for the first row, $\overline{A}$B and 01 for second row, A B and 11 for third row, $A\overline{B}$ and 10 for forth row.
4. Put $\overline{C}$ and 0 for the first column. Put C and 1 for the second column.
5. Put the numbers inside each box (at the bottom right corner) as 0, 1, 2, 3, 6, 7, 4 and 5 row wise.

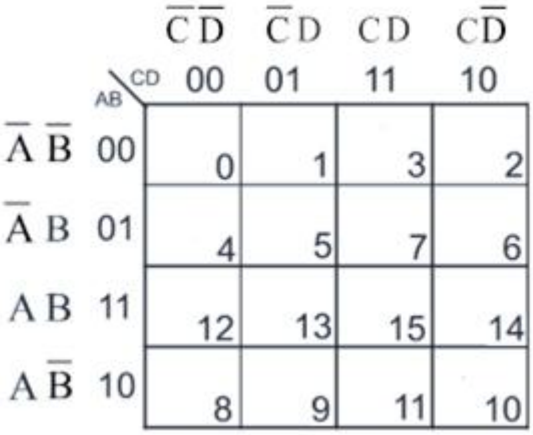


		$\overline{C}$	C
	AB \ C	0	1
$\overline{A}\,\overline{B}$	00	0	1
$\overline{A}$ B	01	2	3
A B	11	6	7
A $\overline{B}$	10	4	5

Four-variable K-map

Follow the steps given below to draw a three-variable K-map.

1. Draw a table with 4 rows and 4 columns with an extension line at the top left corner.
2. Mark AB for the Rows and CD for the Columns.
3. Put $\overline{AB}$ and 00 for the first row, $\overline{A}$B and 01 for second row, A B and 11 for third row, $A\overline{B}$ and 10 for forth row.
4. Put $\overline{CD}$ and 00 for the first row, $\overline{C}$ D and 01 for second row, C D and 11 for third row, C$\overline{D}$ and 10 for forth row.
5. Put the numbers inside each box (at the bottom right corner) as 0, 1, 3, 2, 4, 5, 7, 6, 12, 13, 15, 14, 8, 9, 11 and 10 row wise.



		$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
	AB \ CD	00	01	11	10
$\bar{A}\bar{B}$	00	0	1	3	2
$\bar{A}B$	01	4	5	7	6
AB	11	12	13	15	14
$A\bar{B}$	10	8	9	11	10

1.3.2 Truth table to K-map

Once the K-map is drawn we have to transfer the truth table information to the K-map.

Follow the steps given below to fill up the K-map.

1. Look at the truth table carefully.
2. Identify the input combinations (A, B, C and D) for which the outputs are 1s. (Use complements of the variables for zeros).
3. Put 1s in the corresponding boxes in the K-map.
4. Put 0s in all the remaining boxes.

<u>Example</u>

Inputs				Output
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

		$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
	AB \ CD	00	01	11	10
$\bar{A}\bar{B}$	00	1 $_{0}$	0 $_{1}$	0 $_{3}$	0 $_{2}$
$\bar{A}B$	01	1 $_{4}$	0 $_{5}$	0 $_{7}$	0 $_{6}$
AB	11	0 $_{12}$	0 $_{13}$	1 $_{15}$	1 $_{14}$
$A\bar{B}$	10	0 $_{8}$	0 $_{9}$	1 $_{11}$	1 $_{10}$

Logic equation to K-map

Sometimes, logic equation will be given instead of truth table. Follow the steps given below to fill up the K-map using the logic equation.

1. Look at the logic equation carefully.
2. Each term in the equation is called min-term. Put 1s in the corresponding boxes in the K-map for all the min-terms.
3. Put 0s in all the remaining boxes.

Example

$$F = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D} + A\overline{B}C\overline{D} + A\overline{B}CD + ABC\overline{D} + ABCD$$

		$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
	AB \ CD	00	01	11	10
$\overline{A}\,\overline{B}$	00	1_{0}	0_{1}	0_{3}	0_{2}
$\overline{A}B$	01	1_{4}	0_{5}	0_{7}	0_{6}
AB	11	0_{12}	0_{13}	1_{15}	1_{14}
$A\overline{B}$	10	0_{8}	0_{9}	1_{11}	1_{10}

Logic function to K-map

If the logic function is given, follow the steps given below to fill up the K-map.

1. Look at the logic function carefully.
2. Put 1s in the corresponding boxes in the K-map for all the numbers given in the equation.
3. Put 0s in all the remaining boxes.

Example

$$F = \sum m(0,4,10,11,14,15)$$

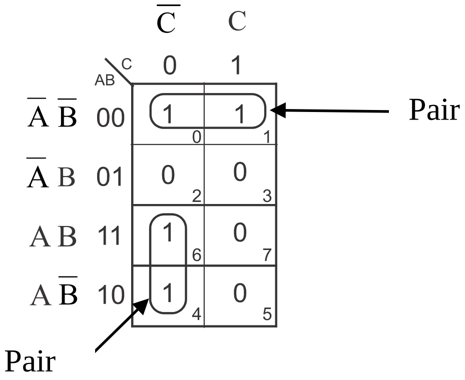
		$\overline{C}\,\overline{D}$	$\overline{C}D$	CD	$C\overline{D}$
	AB \ CD	00	01	11	10
$\overline{A}\,\overline{B}$	00	1_{0}	0_{1}	0_{3}	0_{2}
$\overline{A}B$	01	1_{4}	0_{5}	0_{7}	0_{6}
AB	11	0_{12}	0_{13}	1_{15}	1_{14}
$A\overline{B}$	10	0_{8}	0_{9}	1_{11}	1_{10}

1.3.3 Pairs, Quads and Octets

Once the K-map is filled with 1s and 0s, we have to identify the pairs, quads and octets in order to simplify the Boolean expressions.

Pairs

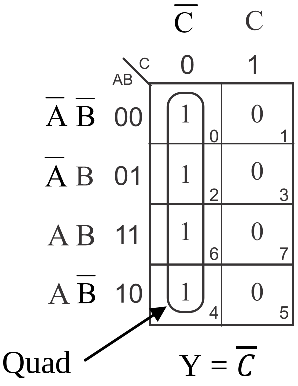
If there are two 1s adjacent to each other, vertically or horizontally, we can form a pair. A pair eliminates one variable and its complement.



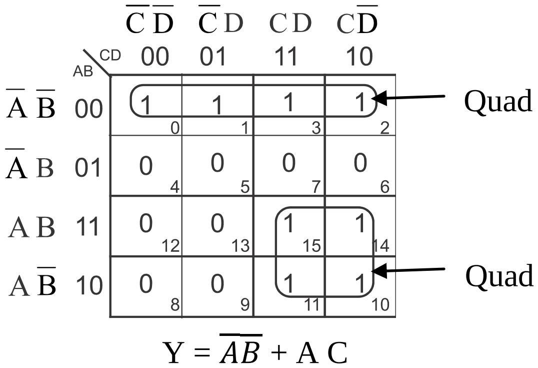
Here, for the boxes 0 and 1 ie. first row and first column & second column, C is changing from complement form to uncomplement form. Hence, C is eliminated and we get $\overline{AB}$.For the boxes 4 and 6 ie.first column and third row & forth row, B is changing from uncomplement form to complement form. Hence, B is eliminated and we get A$\overline{C}$. The simplified equation is $\overline{AB}$ + A$\overline{C}$.

Quads

If there are four 1s adjacent to each other, vertically or horizontally, we can form a quad. Two variables and their complements are eliminated.



Y = $\overline{C}$

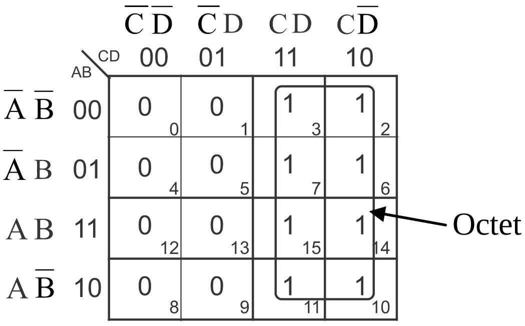


$\overline{C}\,\overline{D}$ $\overline{C}$D CD C$\overline{D}$
AB \ CD 00 01 11 10
$\overline{A}\,\overline{B}$ 00 1 1 1 1
$\overline{A}$ B 01 0 0 0 0
A B 11 0 0 1 1
A $\overline{B}$ 10 0 0 1 1
0 1 3 2 4 5 7 6 12 13 15 14 8 9 11 10
Quad
Quad

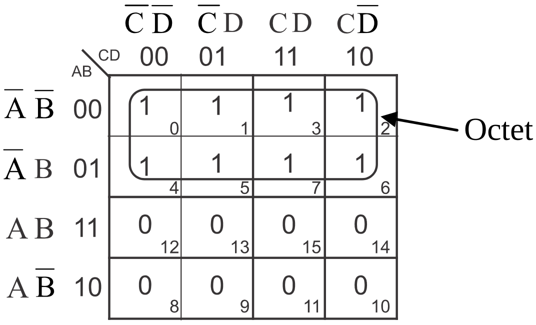
Y = $\overline{AB}$ + A C

Octets

If there are eight 1s adjacent to each other, vertically or horizontally, we can form an octet. Three variables and their complements are eliminated.



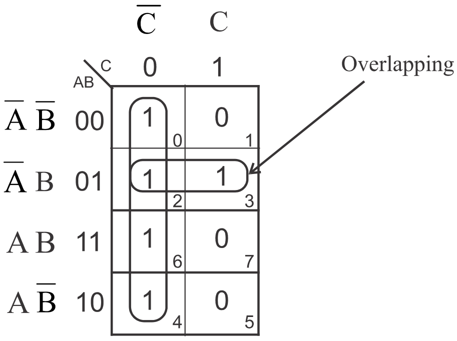
Y = C



Y = $\bar{A}$

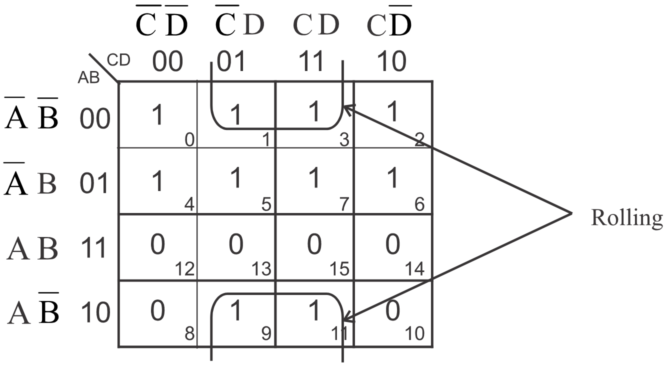
Overlapping groups

We can use 1s in more than one loop of pairs, quads and octets. This type of groups is called overlapping group.



Rolling

It is obvious that when we roll the map, the first row and last row are adjacent to each other. Similarly, the first column and last column are adjacent to each other. This is called rolling the map. By this way also, we can form pairs, quads and octets.



Redundant group

It is a group whose all 1s are overlapped by other groups. Redundant groups must be removed.

1.3.4 Simplification using K-map

- Note down the corresponding variables for the pairs, quads and octets from the K-map.
- If any variable goes from un-complemented form to complemented form, the variable can be eliminated to form the simplified equation. Write the simplified products.
- It is noted that one variable & their complement will be eliminated for pairs, two variables & their complements for quads and three variables & and their complements for octets.
- By OR ing all the simplified products, we get the Boolean equation corresponding to the entire K-map.

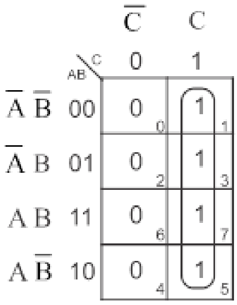
Example 1 :

Simplify the following logic equation

Y = $\overline{AB}$ C +$\overline{A}$BC + A$\overline{B}$C + ABC

Solution:

It is a three variable equation. The K-map for the given equation is,



		$\overline{C}$	C
	AB \ C	0	1
$\overline{A}\,\overline{B}$	00	0 0	1 1
$\overline{A}$ B	01	0 2	1 3
A B	11	0 6	1 7
A $\overline{B}$	10	0 4	1 5

ng

Here, only one quad is formed. Hence, the simplified logic equation is,

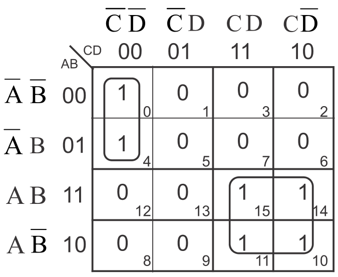
Y = C

Example 2 :

Simplify the following logic equation

F = $\overline{ABCD}$ + $\overline{A}$ B $\overline{CD}$ + A $\overline{B}$ C $\overline{D}$ + A $\overline{B}$ C D + A B C $\overline{D}$ + A B C D

It is a four variable equation. The K-map for the given equation is,



		$\overline{C}\,\overline{D}$	$\overline{C}$ D	C D	C $\overline{D}$
	AB \ CD	00	01	11	10
$\overline{A}\,\overline{B}$	00	1 0	0 1	0 3	0 2
$\overline{A}$ B	01	1 4	0 5	0 7	0 6
A B	11	0 12	0 13	1 15	1 14
A $\overline{B}$	10	0 8	0 9	1 11	1 10

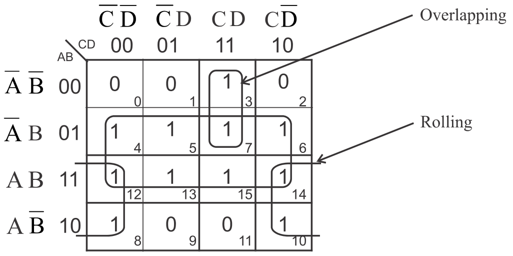
Here, one pair and one quad are formed. Hence, the simplified logic equation is,

Y = $\overline{ACD}$ + AC

Example 3 :

Simplify the following min term function
Y = Σm (3, 4, 5, 6, 7, 8, 10, 12, 13, 14, 15)

The maximum number in the function is 15. Hence, we have to draw a four variable K-map.



AB \ CD	$\overline{C}\,\overline{D}$ 00	$\overline{C}$D 01	CD 11	C$\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00	0 (0)	0 (1)	1 (3)	0 (2)
$\overline{A}$B 01	1 (4)	1 (5)	1 (7)	1 (6)
AB 11	1 (12)	1 (13)	1 (15)	1 (14)
A$\overline{B}$ 10	1 (8)	0 (9)	0 (11)	1 (10)

Here, we have one pair, one quad and one octet. For pair, the equation is $\overline{A}$CD. For quad, A$\overline{D}$.For octet, B. Hence, the simplified equation is,

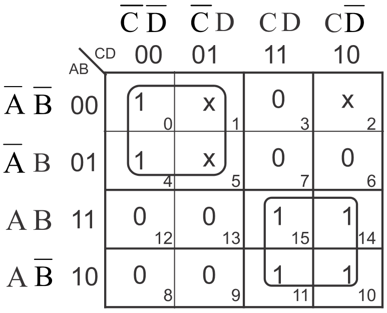
$$Y = \overline{A}CD + A\overline{D} + B$$

1.3.5 Don’t care conditions

In some logic circuits, certain input conditions do not produce any specified outputs ie. neither 0 nor 1. These conditions are called don’t care conditions and they are marked as ‘X’ in the truth table. The same may be marked in the K-map as ‘X’ in the corresponding boxes. We can also use ‘X’ as ‘1’ to form pairs, quads and octets, if necessary.

Consider the following truth table.

Inputs				Output
A	B	C	D	Y
0	0	0	0	1
0	0	0	1	X
0	0	1	0	X
0	0	1	1	0
0	1	0	0	1
0	1	0	1	X
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1



AB \ CD	$\overline{C}\,\overline{D}$ 00	$\overline{C}$D 01	CD 11	C$\overline{D}$ 10
$\overline{A}\,\overline{B}$ 00	1 (0)	x (1)	0 (3)	x (2)
$\overline{A}$B 01	1 (4)	x (5)	0 (7)	0 (6)
AB 11	0 (12)	0 (13)	1 (15)	1 (14)
A$\overline{B}$ 10	0 (8)	0 (9)	1 (11)	1 (10)

When we consider ‘X’ condition, we get two quads. Hence, the simplified equation is,

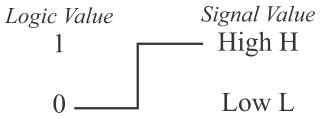
$$Y = \overline{AC} + AC$$

1.4 LOGIC GATES

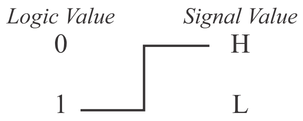
1.4.1 Logic gates

A gate is a logic circuit with one output and one or more inputs. The output signal occurs only for certain input combinations. The input and output signals are digital in nature ie. either 0 (low) or 1 (high). All the possible input combinations and their corresponding output conditions are noted in a table, called truth table.

In digital systems, two different voltage levels (0v and 5v) are used to represent the logic levels ‘0’ and ‘1’. If the high voltage level (5v) represents logic ‘1’ and low voltage level (0v) represents logic ‘0’, then the system is called positive logic. But some systems use the low voltage level (0v) for logic ‘1’ and the high voltage level (5v) for logic ‘0’. This is called negative logic. This is shown in the following figure.



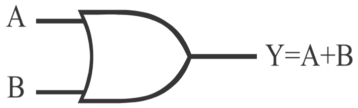
Positive Logic



Negative Logic

OR gate

OR gate has two or more inputs and only one output. It works according to the OR boolean function. It follows the addition (+) law. It gives output when any one of the input is at logic ‘1’. The symbol, logic equation and truth table for OR gate are shown in the following figure.



Y = A + B

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

For a two input OR gate, there are four cases of input combinations, 00, 01, 10 and 11.

Case 1 : A = 0 and B = 0
In this case, the output (A + B) = (0 + 0) = 0

Case 2 : A = 0 and B = 1
In this case, the output (A + B) = (0 + 1) = 1

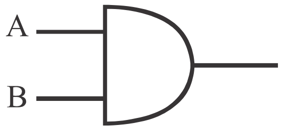
Case 3 : A = 1 and B = 0
In this case, the output (A + B) = (1 + 0) = 1

Case 4 : A = 1 and B = 1
In this case, the output (A + B) = (1 + 1) = 1

For a three input OR gate, there are eight input combinations, 000, 001, 010, 011, 100, 101, 110 and 111. The generalized formula for the number of input combinations is 2^n, where ‘n’ is the number of inputs in the gate.

AND gate

AND gate has two or more inputs and only one output. It works according to the AND boolean function. It follows the multiplication (.) law. It gives output only when all the inputs are at logic ‘1’. The symbol, logic equation and truth table for AND gate are shown in the following figure.



Y=A.B

Y = A . B

Inputs		Output
A	**B**	**Y**
0	0	0
0	1	0
1	0	0
1	1	1

For a two input AND gate, there are four cases of input combinations, 00, 01, 10 and 11.

Case 1 : A = 0 and B = 0
In this case, the output (A . B) = (0 . 0) = 0

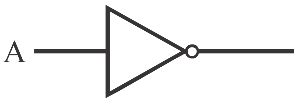
Case 2 : A = 0 and B = 1
In this case, the output (A . B) = (0 . 1) = 0

Case 3 : A = 1 and B = 0
In this case, the output (A . B) = (1 . 0) = 0

Case 4 : A = 1 and B = 1
In this case, the output (A . B) = (1 . 1) = 1

NOT gate

NOT gate has only one input and only one output. It works according to the complementation law of boolean algebra. Its output is the complement of the input ie. the output is ‘1’ when the input is ‘0’ and the output is ‘0’ when the input is ‘1’. The symbol, logic equation and truth table for NOT gate are shown in the following figure.



$Y = \overline{A}$

$Y = \overline{A}$

Input	Output
A	**Y**
0	1
1	0

There are two cases of input combinations, 0 and 1.

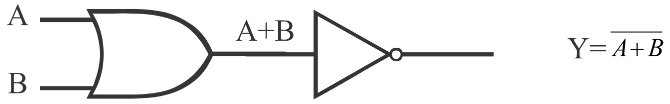
Case 1 : A = 0
In this case, the output ($\overline{A}$) = ($\overline{0}$) = 1

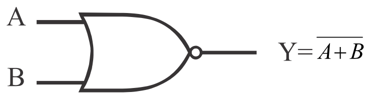
Case 2 : A = 1
In this case, the output ($\overline{A}$) = ($\overline{1}$) = 0

The other names of NOT gate are Inverter gate and Complement gate.

NOR gate

NOR gate is a combination of OR gate and NOT gate. An OR gate followed by a NOT gate is a NOR gate. NOR gate has two or more inputs and only one output. It gives output only when all the inputs are at logic ‘0’. The symbol, logic equation and truth table for NOR gate are shown in the following figure.





Y = $\overline{A + B}$

Inputs		Output
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

For a two input NOR gate, there are four cases of input combinations, 00, 01, 10 and 11.

Case 1 : A = 0 and B = 0
In this case, the output ($\overline{A + B}$) = ($\overline{0 + 0}$) = $\overline{0}$ = 1

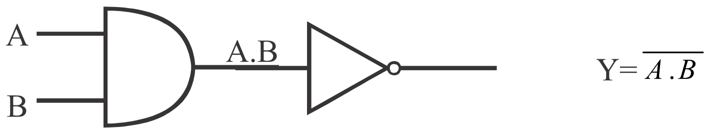
Case 2 : A = 0 and B = 1
In this case, the output ($\overline{A + B}$) = ($\overline{0 + 1}$) = $\overline{1} = 0$

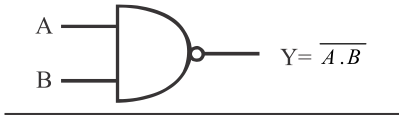
Case 3 : A = 1 and B = 0
In this case, the output ($\overline{A + B}$) = ($\overline{1 + 0}$) = $\overline{1} = 0$

Case 4 : A = 1 and B = 1
In this case, the output ($\overline{A + B}$) = ($\overline{1 + 1}$) = $\overline{1} = 0$

NAND gate

NAND gate is a combination of AND gate and NOT gate. An AND gate followed by a NOT gate is a NAND gate. NAND gate has two or more inputs and only one output. It gives output only when any one of the input is at logic ‘0’. The symbol, logic equation and truth table for NAND gate are shown in the following figure.





$$\boxed{Y = \overline{A\,.B}}$$

Inputs		Output
A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

For a two input NAND gate, there are four cases of input combinations, 00, 01, 10 and 11.

Case 1 : A = 0 and B = 0
In this case, the output ($\overline{A\,.B}$) = ($\overline{0\,.0}$) = $\overline{0}$ = 1

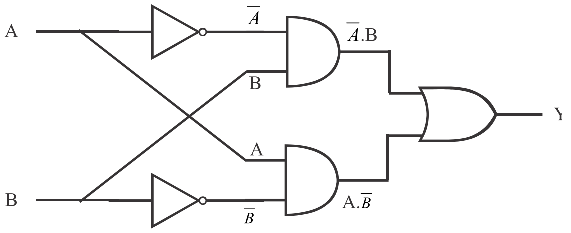
Case 2 : A = 0 and B = 1
In this case, the output ($\overline{A\,.B}$) = ($\overline{0\,.1}$) = $\overline{0}$ = 1

Case 3 : A = 1 and B = 0
In this case, the output ($\overline{A\,.\;B}$) = ($\overline{1\,.0}$) = $\overline{0}$ = 1

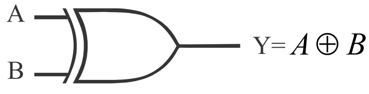
Case 4 : A = 1 and B = 1
In this case, the output ($\overline{A\,.\;B}$) = ($\overline{1\,.1}$) = $\overline{1}$ = 0

Exclusive OR (EX-OR) gate

EX-OR gate has two or more inputs and only one output. It gives output only when the inputs are at different logic levels ie. when one input is '0' the other input must be '1'. This gate can be constructed using AND, OR and NOT gates. The symbol, logic circuit, logic equation and truth table for EX-OR gate are shown in the following figure.



$Y = \overline{A}\,B + A\,\overline{B}$



$$\boxed{Y = A \oplus B}$$

Inputs		Output
A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

For a two input EX-OR gate, there are four cases of input combinations, 00, 01, 10 and 11.

Case 1 : A = 0 and B = 0

In this case, the output ($\overline{A}$. B + A .$\overline{B}$) = (1 . 0 + 0 . 1) = (0 + 0) = 0

Case 2 : A = 0 and B = 1

In this case, the output ($\overline{A}$. B + A .$\overline{B}$) = (1 . 1) + 0 . 0) = (1 + 0) = 1

Case 3 : A = 1 and B = 0

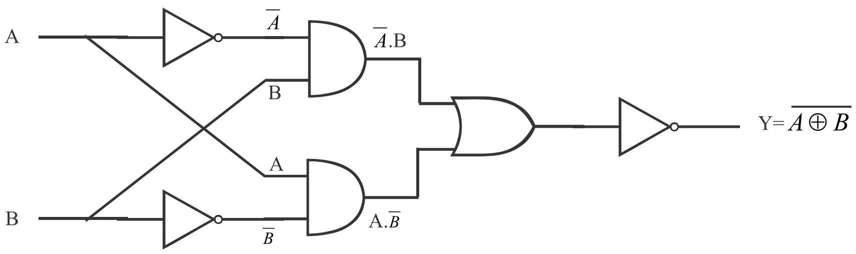
In this case, the output ($\overline{A}$. B + A .$\overline{B}$) = (0 . 0 + 1 . 1) = (0 + 1) = 1

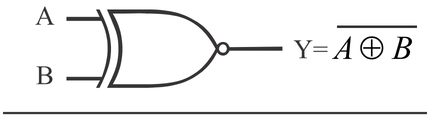
Case 4 : A = 1 and B = 1

In this case, the output ($\overline{A}$. B + A .$\overline{B}$) = (0 .1 + 1 . 0) = (0 + 0) = 0

Exclusive NOR (EX-NOR) gate

It is a combination of EX-OR gate and NOT gate. An EX-OR gate followed by a NOT gate is EX-NOR gate. It has two or more inputs and only one output. It gives output only when the inputs are at same logic levels ie. ‘00’ and ‘11’. This gate can be constructed using AND, OR and NOT gates. The symbol, logic circuit, logic equation and truth table for EX-NOR gate are shown in the following figure.





$$\boxed{Y=\overline{A \oplus B}}$$

Inputs		Output
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	1

For a two input EX-NOR gate, there are four cases of input combinations, 00, 01, 10 and 11.

Case 1 : A = 0 and B = 0

In this case, the output ($\overline{\overline{A} .B + A .\ \overline{B}}$) =$\overline{\overline{0} .0 + 0 .\ \overline{0}}$ = ($\overline{1 .0 \ + 0 .\ 1}$) = ($\overline{0 +\ 0}$) = $\overline{0}$ = 1

Case 2 : A = 0 and B = 1

In this case, the output ($\overline{\overline{A} .B + A .\ \overline{B}}$) =$\overline{\overline{0} .1 + 0 .\ \overline{1}}$ = ($\overline{1 .1 \ + 0 .\ 0}$) = ($\overline{1 +\ 0}$) = $\overline{1}$ = 0

Case 3 : A = 1 and B = 0

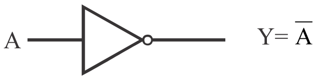
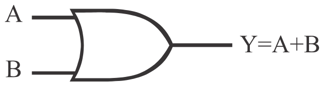
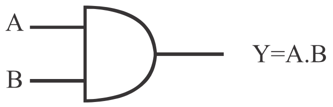
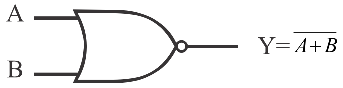
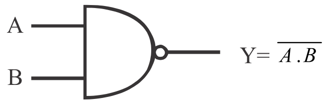
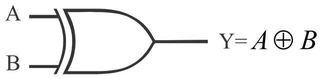
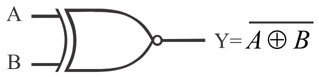
In this case, the output ($\overline{\overline{A} .B + A .\ \overline{B}}$) =$\overline{\overline{1} .0 + 1 .\ \overline{0}}$ = ($\overline{0 .0 \ + 1 .\ 1}$) = ($\overline{0 +\ 1}$) = $\overline{1}$ = 0

Case 4 : A = 1 and B = 1

In this case, the output ($\overline{\overline{A} .B + A .\ \overline{B}}$) =$\overline{\overline{1} .1 + 1 .\ \overline{1}}$ = ($\overline{0 .1 \ + 1 .\ 0}$) = ($\overline{0 +\ 0}$) = $\overline{0}$ = 1

1.4.2 Implementation of logic functions using gates

The following table shows the various logic equations and their equivalent gates.

Logic equation	Name of the gate	Symbol
$\overline{A}$	NOT	
A + B	OR	
A . B	AND	
$\overline{A\ +\ B}$	NOR	
$\overline{A\,.B}$	NAND	
$\overline{A}$. B + A .$\overline{B}$	EX-OR	
$\overline{\overline{A}\,.B + A\,.\ \overline{B}}$	EX-NOR	

The given logic function can be realized using gates as per the above table.

<u>Different forms of logic functions:</u>

There are two different forms of logic functions. They are,

1. Sum-of-products form (or) Minterm form - SOP
2. Product-of-sums form (or) Maxterm form - POS

Consider the following two logic functions.

Y = ($\overline{A}$. B) + (A .$\overline{B}$) + ($\overline{B}$. C) …………. 1
Y = (A + B + C) . ($\overline{A}$+ $\overline{B}$) …………. 2

The first equation has three terms. Each term is a product term. The logic function is the sum of the product terms. This function is said to be in Sum-of-products form (or) Min term form. Similarly, the second equation has two terms. Each term is a sum term. The logic function is the product for the sum terms. Hence, this function is said to be in Product-of-sums form (or) Max term form.

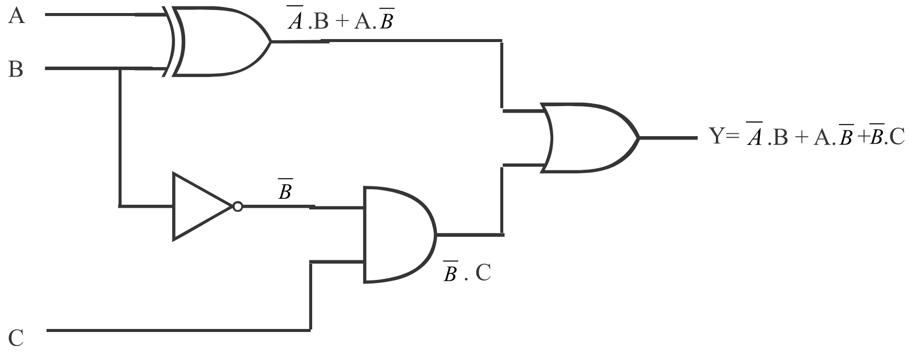
Implementation of logic functions using gates

Follow the steps given below to draw the logic diagram for the given logic function.

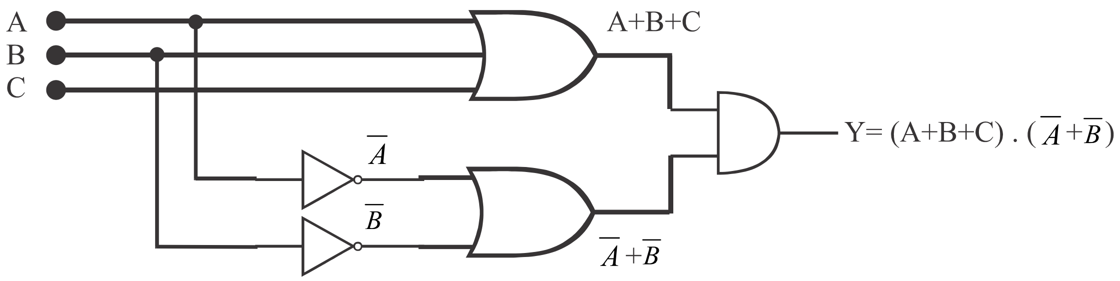
1. For a function in Sum-of-products form, the final output gate is an OR gate. For a function in Product-of-sums form, the final output gate is an AND gate.
2. Realize the (.) function using AND gate.
3. Realize the (+) function using OR gate.
4. Realize the complement ($\overline{}$) function using NOT gate.

Example

1. Draw the logic diagram for the following logic function.
   Y = $(\overline{A} . B) + (A .\overline{B}) + (\overline{B} . C)$

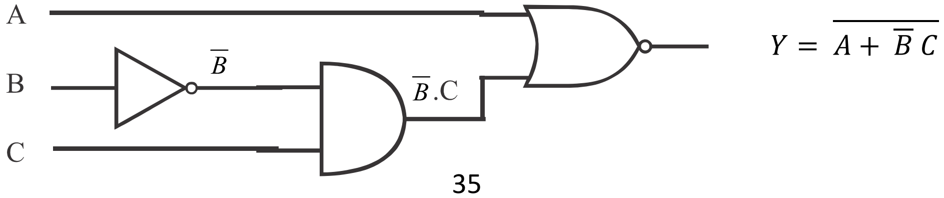

A
B
$\overline{A}$.B + A.$\overline{B}$
Y= $\overline{A}$.B + A.$\overline{B}$+$\overline{B}$.C
$\overline{B}$
$\overline{B}$. C
C

2. Draw the logic diagram for the following logic function.
   Y = $(A + B + C) . (\overline{A} + \overline{B})$


A
B
C
A+B+C
Y= (A+B+C) . ($\overline{A}$+$\overline{B}$)
$\overline{A}$
$\overline{B}$
$\overline{A}$+$\overline{B}$

3. Draw the logic diagram for the following logic function.

$$Y = \overline{A + \overline{B}\,C}$$



1.4.3 Realization of gates using universal gates

We can use only NAND gates for constructing all the other gates (ie. NOT, OR, AND, NOR, EX-OR and EX-NOR). Similarly, we can useonly NOR gates for constructing all the other gates. Hence, NAND and NOR gates are called universal gates.

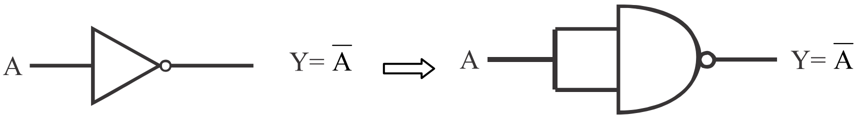
Realization of gates using NAND gates only

<u>1) NOT gate using only NAND gates</u>

Y = $\overline{A}$

= $\overline{(A\,.\,A)}$ ……. Because A . A = A

Hence, when we tie the two inputs of an NAND gate together we get a NOT gate.



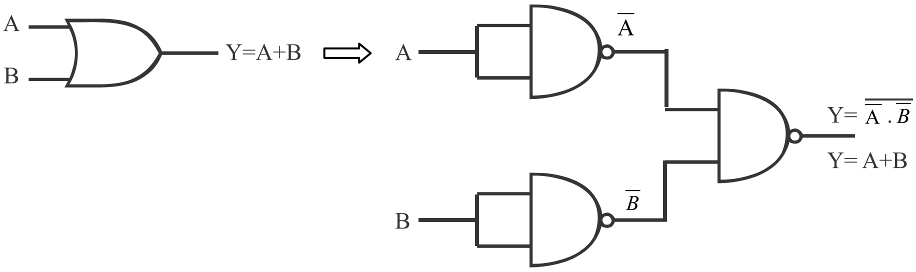
<u>2) OR gate using only NAND gates</u>

Y = A + B

= $\overline{\overline{A+B}}$ ……. Double complementation will give the same function

= $\overline{\overline{A}\,.\,\overline{B}}$ ….. Using De-Morgan's first theorem, $\overline{A\,+\,B}\;=\;\overline{A}\;.\;\overline{B}$

Two NAND gates are used for the two NOT functions. A third NAND gate is used for the (.) and complement functions.

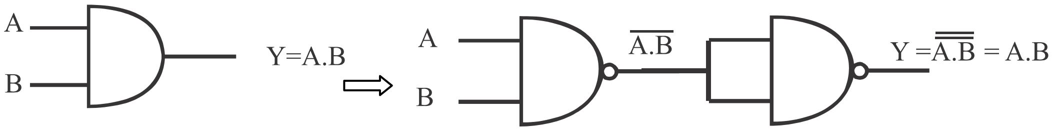


<u>3) AND gate using only NAND gates</u>

Y = A . B

= $\overline{\overline{A\,.\,B}}$ ……. Double complementation will give the same function

One NAND gate is used for the $\overline{A\,.\,B}$ function. Another NAND gate is used for the complement function.



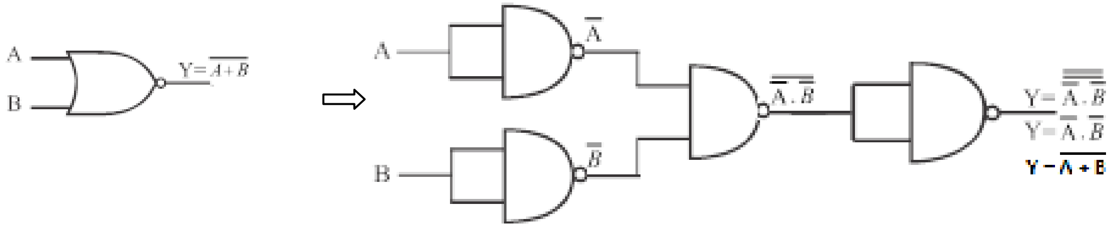
4) NOR gate using only NAND gates

Y = $\overline{A + B}$

= $\bar{A} . \bar{B}$ ….. Using Using De-Morgan's first theorem, $\overline{A + B} = \bar{A} . \bar{B}$

=

Two NAND gates are used for the two NOT functions. Another two NAND gates are required for implementing the (.) ie. AND function. Hence, totally four NAND gates are required for realizing a NOR gate using only NAND gates.



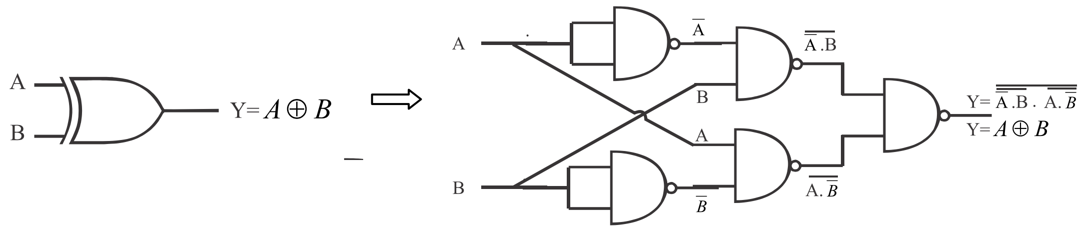
5) EX-OR gate using only NAND gates

Y = $\bar{A}$. B + A .$\bar{B}$

= $\overline{\overline{\bar{A}.B + A.\bar{B}}}$ ….Double complementation will give the same function

= $\overline{\overline{\bar{A}.B} \; . \; \overline{A.\bar{B}}}$ ….. Using De-Morgan's first theorem, $\overline{A + B} = \bar{A} . \bar{B}$

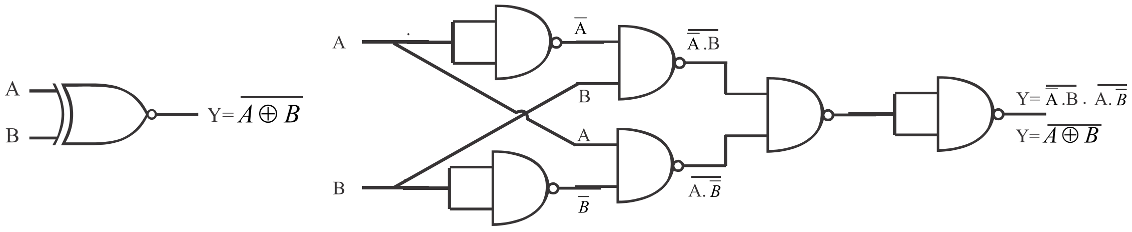
Two NAND gates are used for the two NOT functions $\bar{A}$and $\bar{B}$. Another two NAND gates are required for $\overline{\bar{A}.B}$ and $\overline{A.\bar{B}}$. A final NAND gate is required for the (.) and complement function. Totally five NAND gates are required for realizing an EX-OR gate using only NAND gates.



6) EX-NOR gate using only NAND gates

Y = $\overline{\bar{A}.B + A.\bar{B}}$

When the output of an EX-OR gate is connected to a NOT gate, we get an EX-NOR gate. Hence, EX-OR gate is realized first using five NAND gates. Sixth NAND gate is used to realize the NOT gate. Totally six NAND gates are required for realizing an EX-NOR gate using only NAND gates.

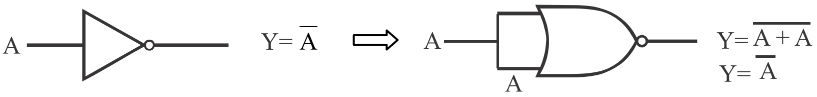

A
B
Y= $\overline{A \oplus B}$
A
B
$\overline{A}$
$\overline{A}$.B
B
A
A.$\overline{B}$
$\overline{B}$
Y= $\overline{\overline{\overline{A}.B}\ .\ \overline{A.\overline{B}}}$
Y= $\overline{A \oplus B}$

Realization of gates using NAND gates only

<u>1) NOT gate using only NOR gates</u>

Y = $\overline{A}$
  = $(\overline{A + A})$ ……. Because A + A = A

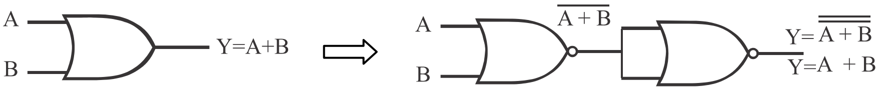
Hence, when we tie the two inputs of a NOR gate together we get a NOT gate.



<u>2) OR gate using only NOR gates</u>

Y = A + B
  = $\overline{\overline{A + B}}$ ……. Double complementation will give the same function

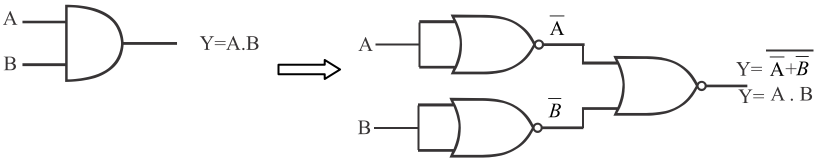
When the output of a NOR gate is connected to a NOT gate, we get an OR gate. One NOR gate is used for $\overline{A + B}$. Another NOR gate is used to implement the NOT function.



<u>3) AND gate using only NOR gates</u>

Y = A . B
  = $\overline{\overline{A\ .\ B}}$ ……. Double complementation will give the same function
  = $\overline{\overline{A} + \overline{B}}$ ……. Using De-Morgan's second theorem, $\overline{A\ .\ B}\ =\ \overline{A}+\overline{B}$

Two NOR gates are used for $\overline{A}$and $\overline{B}$. Another NOR gate is used for (+) and complement functions.

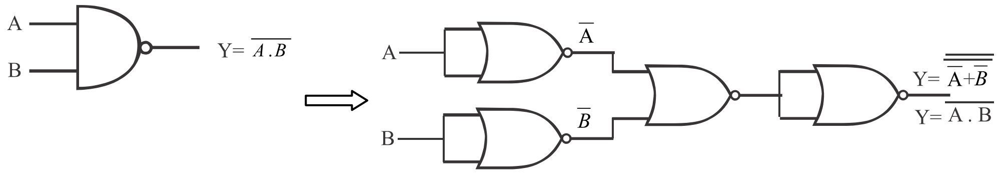


4) NAND gate using only NOR gates

$Y = \overline{A\ .\ B}$

$= \overline{A} + \overline{B}$ ….. Using De-Morgan’s second theorem, $\overline{A\ .\ B} = \overline{A}+\overline{B}$

Two NOR gates are used for the two NOT functions. Another two NOR gates are required for implementing the (+) ie. OR function. Hence, totally four NOR gates are required for realizing a NAND gate using only NOR gates.



5) EX-OR gate using only NOR gates

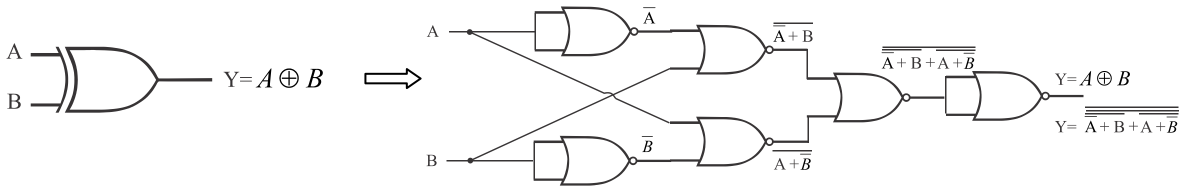
$Y = \overline{A}\ .\ B + A\ .\overline{B}$

$= \overline{\overline{\overline{A}.B + A.\ \overline{B}}}$ ….. Double complementation will give the same function

$= \overline{\overline{\overline{A}.\ B}\ .\ \overline{A.\ \overline{B}}}$ ….. Using De-Morgan’s first theorem, $\overline{A + B} = \overline{A}\ .\ \overline{B}$

$= \overline{A + \overline{B}} + \overline{\overline{A} + B}$ ….. Using De-Morgan’s second theorem, $\overline{A\ .\ B} = \overline{A}+\overline{B}$

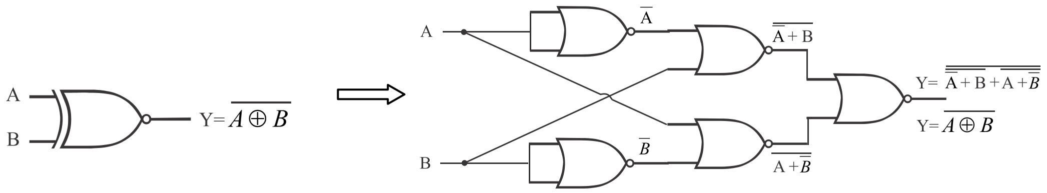
Two NOR gates are used for the two NOT functions $\overline{A}$and $\overline{B}$. Another two NOR gates are required for (+) and complement functions. Two more NOR gates are required for the final (+) operation. Totally six NOR gates are required for realizing an EX-OR gate using only NOR gates.


A
B
Y=$A \oplus B$
A
B
$\overline{A}$
$\overline{B}$
$\overline{\overline{A}+B}$
$\overline{A+\overline{B}}$
$\overline{\overline{\overline{\overline{A}+B}+\overline{A+\overline{B}}}}$
Y=$A \oplus B$
Y= $\overline{\overline{\overline{\overline{\overline{A}+B}+\overline{A+\overline{B}}}}}$

6) EX-NOR gate using only NOR gates

$Y = \overline{\overline{A}.B + A.\ \overline{B}}$

When the output of an EX-OR gate is connected to a NOT gate, we get an EX-NOR gate. Hence, EX-OR gate is realized first using five NOR gates. Another NOR gate is used to realize the NOT gate. The last two NOR gates implemented for the NOT operation can be removed because double complementation give the same function. Totally five NOR gates are required for realizing an EX-NOR gate using only NOR gates.


A
B
Y=$\overline{A\oplus B}$
A
B
$\overline{A}$
$\overline{\overline{A}+B}$
$\overline{B}$
$\overline{A+\overline{B}}$
Y= $\overline{\overline{\overline{A}+B}+\overline{A+\overline{B}}}$
Y=$\overline{A\oplus B}$

1.4.4 Simplification of expression using Boolean techniques

We can simplify logic expressions using Boolean techniques (ie.)Boolean laws and theorems.

Example 1 : Simplify the expression Y = A $\overline{B}$ + A B

Y = A $\overline{B}$ + A B
 = A ($\overline{B}$ + B) …. By factoring
 = A (1) ….. $\overline{B}$ + B = 1
 = A ….. A . 1 = A

Example 2 : Simplify the expression Y = $\overline{A}$B$\overline{C}$ + AB$\overline{C}$ + ABC

Y = $\overline{A}$B$\overline{C}$ + AB$\overline{C}$ + ABC
 = $\overline{A}$B$\overline{C}$ + AB$\overline{C}$ + AB$\overline{C}$ + ABC … AB$\overline{C}$ + ABC = ABC
 = B$\overline{C}$($\overline{A}$+A) + AB($\overline{C}$+C) …. By factoring
 = B$\overline{C}$ + AB …. $\overline{A}$+A = 1

Example 3 : Simplify the expression Y = AB + $\overline{AC}$ + A$\overline{B}$C(AB+C)

Y = AB + $\overline{AC}$ + A$\overline{B}$C(AB+C)
 = AB + $\overline{AC}$ + AA$\overline{B}$BC + A$\overline{B}$CC …. Multiply
 = AB + $\overline{AC}$ + 0 + A$\overline{B}$C …. $\overline{B}$B = 0, CC = C
 = AB + $\overline{A}$ + $\overline{C}$ + A$\overline{B}$C …. DeMorgan’s theorem $\overline{AC}$ = $\overline{A}$ + $\overline{C}$
 = AB + $\overline{C}$ + $\overline{A}$ + A$\overline{B}$C …. Rearrange
 = AB + $\overline{C}$ + $\overline{A}$ + $\overline{B}$C …. Distributive law
 = $\overline{A}$ + AB + $\overline{C}$ + $\overline{B}$C …. Rearrange
 = $\overline{A}$+ B + $\overline{C}$ + $\overline{B}$ …. Distributive law
 = $\overline{A}$+ $\overline{C}$ + B + $\overline{B}$ …. Rearrange
 = $\overline{A}$ + $\overline{C}$ + 1 …. B + B’ = 1
 = 1 ….. Reduce

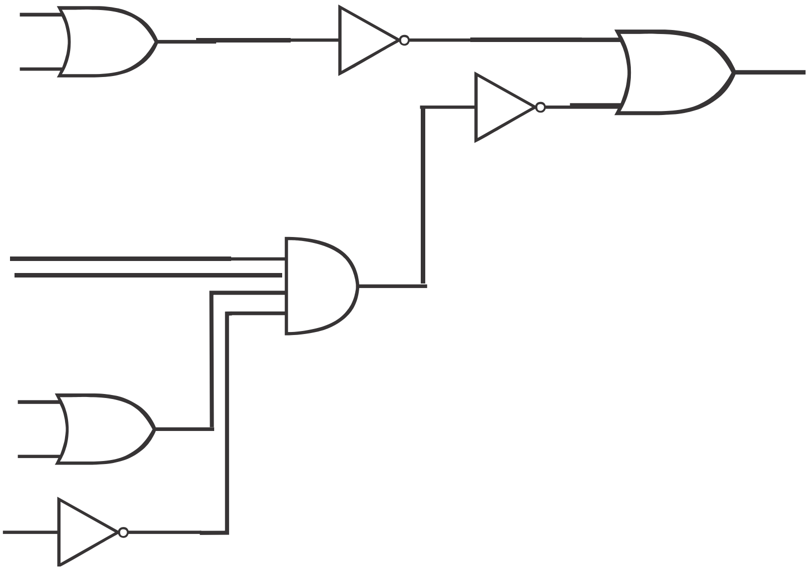
1.4.5 Boolean expression for outputs

The following steps are involved to find out the boolean expression for the output of the given logic circuit.

Step 1 : Label all the inputs ie. A, B, C, D, etc.
Step 2 : Label all the gates in the circuit ie. A1, A2, A3, A4, etc.
Step 3 : Start with the input signals, write down the logic equation for the output of each gate until the output is reached.

Example :

Find out the Boolean expression for the circuit given below:



Step 1 : Label all the inputs ie. A, B, C, D, E, F and G.

Step 2 : Label all the gates in the circuit ie. A1, A2, A3, A4, A5, A6 and A7.

Step 3 : The output of A1 is A + B

The output of A2 is $\overline{A + B}$

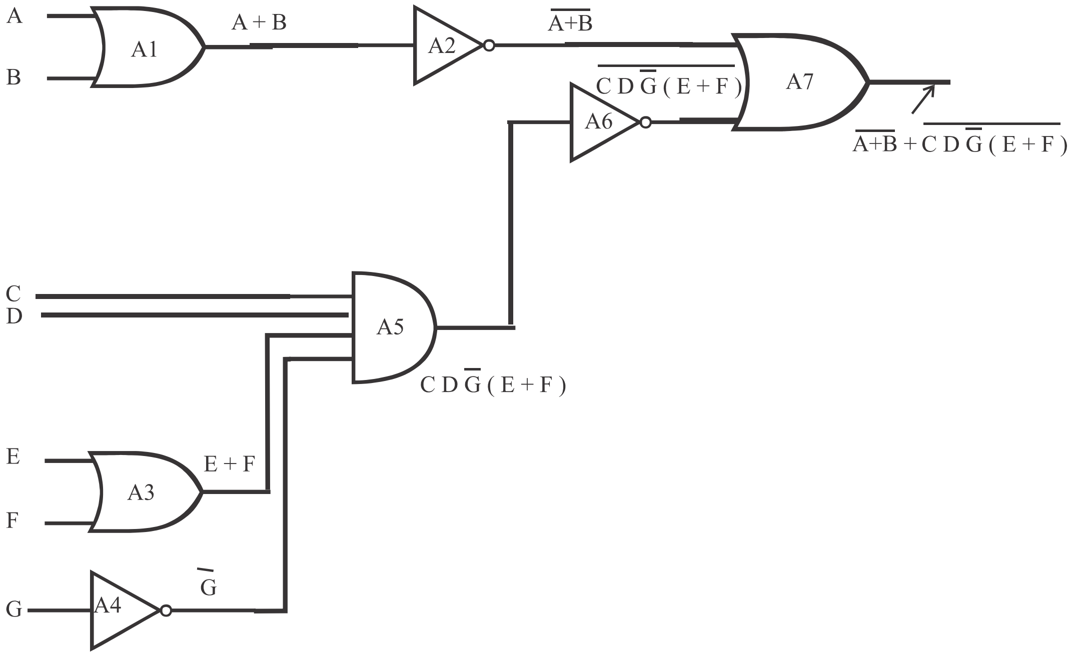
The output of A3 is E + F

The output of A4 is $\overline{G}$

The output of A5 is CD$\overline{G}$ (E+F)

The output of A6 is $\overline{C D \overline{G} (E + F)}$

The output is A7 is $\overline{A + B}+\overline{C D \overline{G} (E + F)}$


A
B
A1
A + B
A2
$\overline{A+B}$
$\overline{C D \overline{G} (E + F)}$
A7
A6
$\overline{A+B} + \overline{C D \overline{G} (E + F)}$
C
D
A5
C D $\overline{G}$ (E + F)
E
F
A3
E + F
G
A4
$\overline{G}$

1.5 DIGITAL LOGIC FAMILIES

1.5.1 Logic families

Gates are manufactured using different technologies. The following are some of the technologies (or logic families) available in the market today.

1. Resistor-Transistor Logic (RTL)
2. Diode-Transistor Logic (DTL)
3. Transistor-Transistor Logic (TTL)
4. Integrated Injection Logic (I^2L)
5. Complementary Metal Oxide Semiconductor (CMOS) Logic
6. Emitter-Coupled Logic (ECL)

Out of these logic families, TTL and CMOS Logics are familiar.

1.5.2 Characteristics of Logic families

Some of the important characteristics of digital logic families are:

1. Power supply voltage
2. Fan-in
3. Fan-out
4. Propagation delay
5. Noise immunity and Noise margin
6. Power dissipation

Power supply voltage
It is the permissible power supply voltage range of the gate IC.

Fan-in
Fan-in of a logic gate is defined as the maximum number of inputs connected to a gate without degradation of the input voltage levels.

Fan-out
Fan-out of a logic gate is defined as the maximum number of similar logic gates that can be connected at the output without any degradation in output voltage levels.

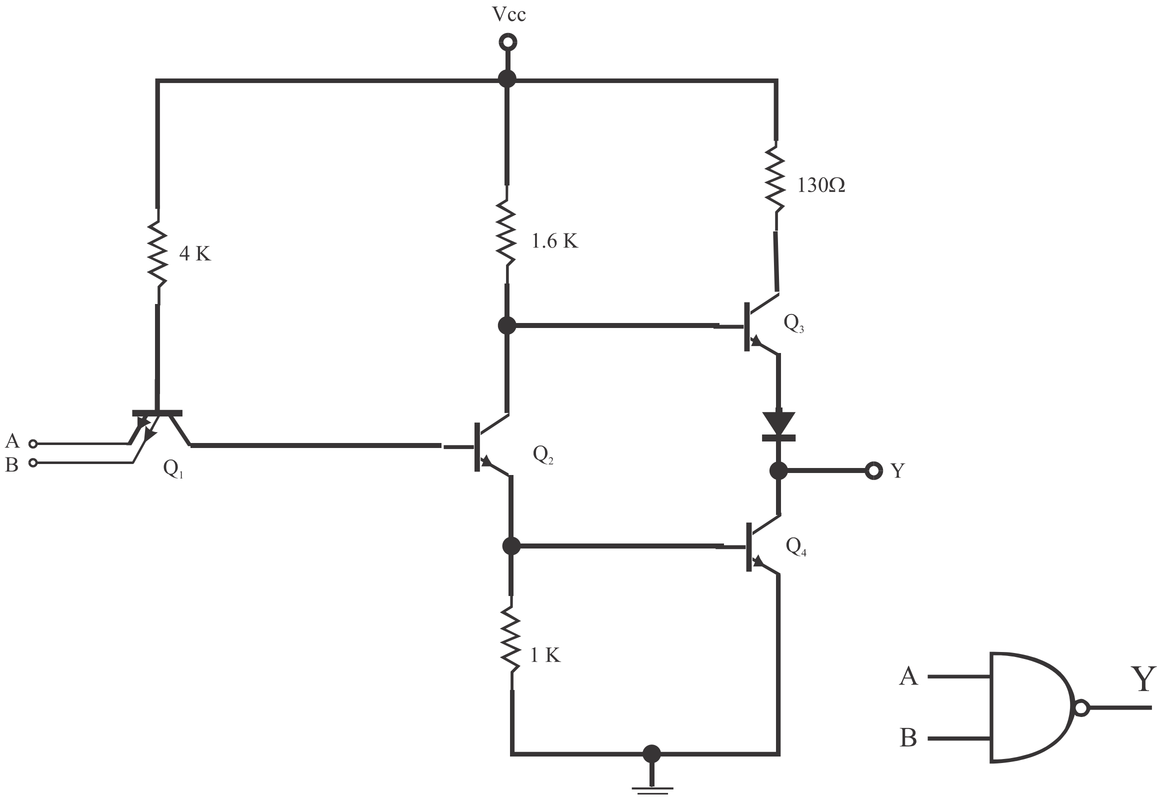
Propagation delay
It is the time taken by gate for the output to change after inputs have changed. It is normally represented in milliseconds or nanoseconds.

Power dissipation
It is the measure of the power consumed by a gate when all the inputs are fully driven. It is normally expressed in milli watts or nano watts.

Noise immunity
It refers to the ability of a gate to tolerate noise without causing spurious changes in the output voltage. It is also called noise margin.

1.5.3 Transistor-Transistor Logic (TTL) NAND gate

TTL gates (74 series) were introduced in the market to provide greater speed than DTL. A typical TTL NAND gate is shown in the figure.



This circuit has four bipolar junction transistors (Q1, Q2, Q3 and Q4), four resistors and one diode. Transistor Q1 is a multi-emitter transistor in which the inputs A and B are applied.

When both A and B are high:
When both A and B are high, Q1 has no emitter current and its base-emitter junction is forward biased. Hence, base current flows to Q2. This in turn, feeds base current to Q4. Hence, Q4 conducts. As the Q2 collector goes low, Q3 is cut-off. Therefore, Q4 is conducting and Q3 cut-off. Hence, we will get 0V (ie.) logic ‘0’ at the output.

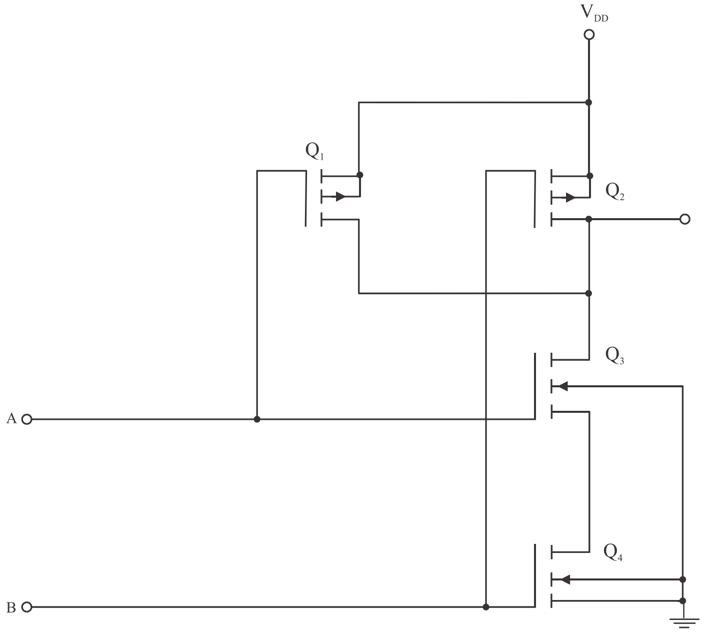
When either A or B goes low:
When either A or B goes low, Q1 will have base-emitter current and saturated. Hence, the base of Q2 will be pulled to ground and cut-off. This will cause Q3 to conduct and Q4 to cut-off. Hence, we will get supply voltage (Vcc) (ie.) logic ‘1’ at the output.

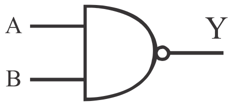
Specifications:

Power supply	(Vcc)	+5 volts
Power dissipation	(Pd)	100 mw
Propagation delay	(Td)	15 nsec
Noise margin	($V_{NM)}$	0.4 volts
Fan-out	(FO)	10

1.5.4 CMOS Logic NAND gate

CMOS Logic has high package density and low power consumption. Hence, it is used in battery operated devices. It works with a wide range of power supply. The figure shows the circuit diagram of CMOS Logic NAND gate.





This circuit has four MOS transistors. Q1 and Q2 are PMOS and Q3 and Q4 are NMOS. A PMOS device will be turned ON when its gate input is low. An NMOS device will be turned ON when its gate input is high. The circuit operation is explained below.

<u>When both A and B are high</u>:
When both A and B are high, Q1 and Q2 are cut off and Q3 and Q4 are turned ON. Hence, the output is connected to the ground ie. the output is in ‘0’ level.

<u>When A is low</u>:
When A is low, Q1 will conduct and Q3 will turn off. Hence, the output is connected to supply ie. the output is high.

<u>When B is low</u>:
Similarly, when B is low, Q2 will conduct and Q4 will turn off. Hence, the output is connected to supply ie. the output is high.

<u>Specifications</u>:

Power supply	(Vcc)	3 to 15 volts
Power dissipation	(Pd)	10 nw
Propagation delay	(Td)	25 nsec
Noise margin	(V_{NM})	45% of V_{DD}
Fan-out	(FO)	Greater than 50

In CMOS logic any input should not be left open, either it must be connected to supply (ie.) logic ‘1’ or to ground (ie.) logic ‘0’.

Comparison of TTL and CMOS logic families

Sl. No.	Parameter	TTL	CMOS
1.	Power Supply voltage	5 volts	3 to 15 volts
2.	Power dissipation	100 mW (milli watt)	10 nW (nano watt)
3.	Propagation delay	15 nsec (nano seconds)	25 nsec (nano seconds)
4.	Noise margin	0.4 volts	45% of V_{DD}
5.	Fan-out	10	Greater than 50

Applications of TTL and CMOS logic gates

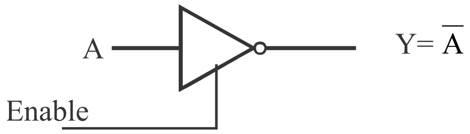
TTL and CMOS logic gates are used in digital circuits, memories, etc. CMOS gates are preferred over TTL for lower power applications such as battery operated devices.

1.5.5 Tristate Logic (TSL) gate

We know only two logic states namely low level ‘0’ and high level ‘1’. But there is a third logic state which is called ‘high impedance’ state. In this third state, even though the output is physically connected in the circuit, it behaves like an open circuit. A gate with high impedance state is called Tristate gate.

Tristate Inverter Gate

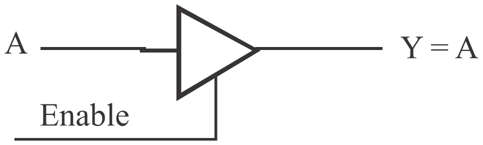
This is similar to a NOT gate with an additional ENABLE input. When it is enabled, the gate will work as ordinary NOT gate (ie.) the output is the complement of the input. When the gate is not enabled, the output will be in ‘high impedance’ state. The symbol and truth table of Tristate Inverter gate are shown in figure.



ENABLE	Input (A)	Output (Y)
0	X (0 or 1)	High impedance
1	0	1
1	1	0

Tristate buffer

The symbol and truth table of Tristate buffer are shown in figure.



ENABLE	Input (A)	Output (Y)
0	X (0 or 1)	High impedance
1	0	0
1	1	1

When the buffer is enabled, the output follows the input. When the buffer is not enabled (ie. disabled), the output will be in high impedance state.

☺☺☺☺☺

UNIT - II

COMBINATIONAL CIRCUITS

Combinational circuits

In digital systems, there are two types of circuits, 1) Combinational logic circuits and 2) Sequential logic circuits. In combinational circuits, the output depends only on the present input conditions. But, in sequential circuits, the output depends not only on the present input conditions but also on the previous output conditions.

2.1 Binary Arithmetic

Arithmetic circuits

Arithmetic circuits are the combinational logic circuits used to perform arithmetic operations such as addition, subtraction, multiplication and division on binary numbers.

2.1.1 Binary addition

In addition, the first number is called augend and the second number is called is addend. The result is called the sum. Sometimes, we may get carry also.

$$\begin{array}{rr} & \text{Augend} \\ & +\ \text{Addend} \\ \hline \text{Carry} & \text{Sum} \\ \hline \end{array}$$

The following rules are used in binary addition.

$$\begin{array}{r} 0 \\ +\ 0 \\ \hline 0 \\ \hline \end{array} \qquad \begin{array}{r} 0 \\ +\ 1 \\ \hline 1 \\ \hline \end{array} \qquad \begin{array}{r} 1 \\ +\ 0 \\ \hline 1 \\ \hline \end{array} \qquad \begin{array}{r} 1 \\ +\ 1 \\ \hline 10 \\ \hline \end{array}$$ Sum = 0 and Carry = 1

$$\begin{array}{r} 1 \\ 1 \\ +\ 1 \\ \hline 11 \\ \hline \end{array}$$ Sum = 1 and Carry = 1

In a binary number, the left most bit is called Most Significant Bit (MSB) and the rightmost bit is called Least Significant Bit (LSB). Addition must be carried out bit by bit starting from the LSB.

Examples :

1) Add the binary number 101 and 10. (Apply basic rules)

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

Answer :$(111)_2$

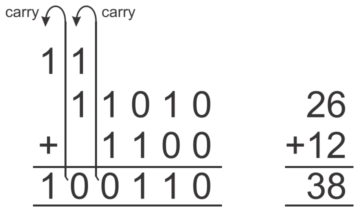
2) Add $(10)_{10}$ and $(3)_{10}$ in binary.

First convert the decimal numbers to binary.

$$\begin{array}{r} \overset{\text{carry}}{1}\ \\ 1\ 0\ 1\ 0 \\ +\ \ \ 1\ 1 \\ \hline 1\ 1\ 0\ 1 \\ \hline \end{array} \qquad \begin{array}{r} 10 \\ +\ 3 \\ \hline 13 \\ \hline \end{array}$$

Answer :$(1101)_2 = (13)_{10}$

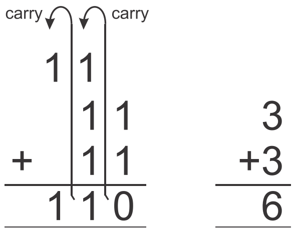
3) Add the binary numbers 11010 and 1100.



Answer :$(100110)_2$

4) Add $(3)_{10}$ and $(3)_{10}$ in binary.

First convert the decimal numbers to binary.



Answer :$(110)_2 = (6)_{10}$

2.1.2 Binary subtraction

In subtraction, the first number is called minuend and the second number is called is subtrahend. The result is called the difference. Sometimes, we may get borrow also.

$$\begin{array}{rr} & \text{Minuend} \\ & \text{- Subtrahend} \\ \hline \text{Borrow} & \text{Difference} \\ \hline \end{array}$$

The following rules are used in binary subtraction.

$$\begin{array}{r} 0 \\ -\,0 \\ \hline 0 \\ \hline \end{array} \qquad \begin{array}{r} 1 \\ -\,0 \\ \hline 1 \\ \hline \end{array} \qquad \begin{array}{r} 1 \\ -\,1 \\ \hline 0 \\ \hline \end{array} \qquad \begin{array}{r} 0 \\ -\,1 \\ \hline 11 \\ \hline \end{array}$$ Difference =1, Borrow = 1

Subtraction must be carried out bit by bit starting from the LSB.
When we borrow a 1 from the next bit, we will get 10 (ie. 2 in decimal).

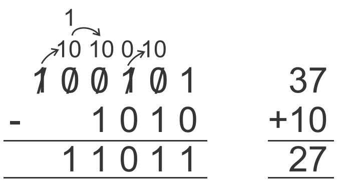
Examples :

1) Subtract : 1011 – 1001

$$\begin{array}{r} 1\,0\,1\,1 \\ -\,1\,0\,0\,1 \\ \hline 0\,0\,1\,0 \\ \hline \end{array}$$

Answer : $(10)_2$

2) Subtract 1010 from 100101



Answer :$(11011)_2$

2.1.3 Representation of negative numbers

In decimal systems, we use (+) sign for positive numbers and (–) sign for negative numbers. But in binary numbers, complement method is used for representing negative numbers. Two types of complement methods are available.

1) 1's complement method
2) 2's complement method

Either 1's complement or 2's complement method is adopted in a particular digital system.

1's complement

1's complement is used to represent negative numbers. Each bit is subtracted from 1 (borrow is ignored) to get the 1's complement. A shortcut method is also used. Each bit in the given number is complemented (0 to 1, 1 to 0) to get its 1's complement.

Examples :

1) Find the 1's complement of 101011.

   010100 ---- 0 to 1, 1 to 0

   Answer :10100

2) Find the 1's complement of 11110.

   00001 ---- 0 to 1, 1 to 0

   Answer : 1

2's complement

2's complement is used to represent negative numbers. When we add 1 to 1's complement, we will get the 2's complement of the given number.

Examples :

1) Find the 2's complement of 101011.

```
1's complement        010100   ----   0 to 1, 1 to 0
Add 1               + 000001
2's complement        010101
```

   Answer : $(010101)_2$

2) Find the 2's complement of 11110.

```
1's complement        00001   ----   0 to 1, 1 to 0
Add 1               + 00001
2's complement        00010
```

   Answer :$(00010)_2$

2.1.4 Subtraction using complement methods

The number to be subtracted (subtrahend) is converted into 1’s complement form or 2’s complement form to represent it as a negative number. Then it will be added to the first number (minuend).

$$\begin{array}{r} \text{Minuend} \\ +\ \text{Complement of Subtrahend} \\ \hline \text{Difference} \\ \hline \end{array}$$

Binary subtraction using 1’s complement method

The following steps are used to subtract a number using 1’s complement method

1) Find the 1’s complement of the subtrahend.
2) Add the 1’s complement with the minuend.
3) If there is any carry:
    a. Add the carry (ie. add 1) to the result
    b. The result is a positive number.
4) If there is no carry:
    a. Find the 1’s complement of the result.
    b. The result is a negative number.

Example 1 : Subtract 100101 from 101100

Here, the minuend is 101100
And the subtrahend is 100101

$$\begin{array}{r} 101100 \\ -\ 100101 \\ \hline \text{Difference} \\ \hline \end{array}$$

The 1’s complement of subtrahend 100101 is 011010. This should be added to the minuend.

Minuend	101100	
Add 1’s complement of Subtrahend	+ 011010	
	1 000110	There is a carry 1.
Add Carry	+ 1	
	+ 000111	Result is positive.

Answer : + $(111)_2$

Example 2 : $(54)_{10}$ - $(62)_{10}$

Binary of 54 (Minuend) = 110110
Binary of 62 (Subtrahend) = 111110

1’s complement of 111110 = 000001

Minuend	110110	
Add 1’s complement of Subtrahend	+ 000001	
	110111	There is no carry.
Find 1’s complement		
	001000	
	- 001000	Result is negative.
Convert to Decimal	- 8	

Answer : - 8

Binary subtraction using 2’s complement method

The following steps are used to subtract a number using 2’s complement method.

1) Find the 2’s complement of the subtrahend.
2) Add the 2’s complement with the minuend.
3) If there is any carry:
    a. Ignore the carry
    b. The result is a positive number.
4) If there is no carry:
    a. Ignore the carry
    b. Find the 2’s complement of the result.
    c. The result is a negative number.

Example 1 : Subtract 100101 from 101100

Here, the minuend is 101100
And the subtrahend is 100101

101100
- 100101
Difference

The 2’s complement of subtrahend 100101 is 011011. This should be added to the minuend.

Minuend	101100	
Add 2’s complement of Subtrahend	+ 011011	
	$\cancel{1}$ 000111	There is a carry 1. Ignore it.
	+ 000111	Result is positive.

Answer : + $(111)_2$

Example 2 : $(54)_{10}$ - $(62)_{10}$

Binary of 54 (Minuend) = 110110
Binary of 62 (Subtrahend) = 111110

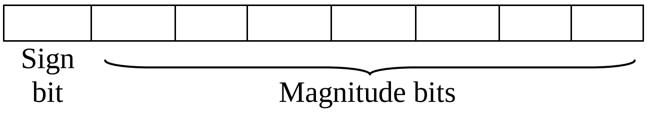
2’s complement of 111110 = 000010

Minuend	110110	
Add 2’s complement of Subtrahend	+ 000010	
	111000	There is no carry.
Find 2’s complement	000111	
(1’s complement Plus 1)	+ 1	
	- 001000	Result is negative.
Convert to Decimal	- 8	

Answer : - 8

Signed binary numbers

In decimal systems, we use (+) sign for positive numbers and (–) sign for negative numbers. In binary number system, an additional bit is used as the sign bit. A ‘0’ is used to represent a positive number and a ‘1’ is used to represent a negative number.



For example, an 8-bit signed number 01000100 represents a positive number, because the sign bit is ‘0’. Its magnitude is 1000100. Hence, the equivalent decimal number is +68. Similarly, the signed number 11000100 represents a negative number, because the sign bit is ‘1’. Its magnitude is 1000100. Hence, the equivalent decimal number is -68.

ie. 01000100 = +68
11000100 = -68

This type of representation for signed numbers is called ‘sign-and-magnitude representation’.

2.2 Half adder

Half adder adds two binary digits at a time. It has two inputs A and B, and two outputs SUM and CARRY.

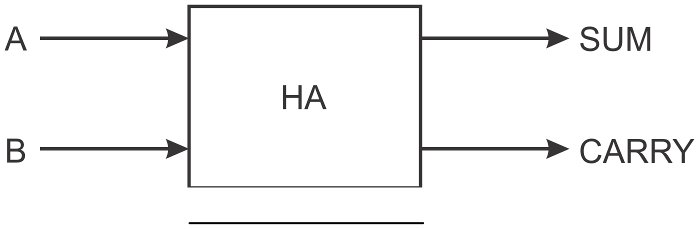


Figure : Block diagram

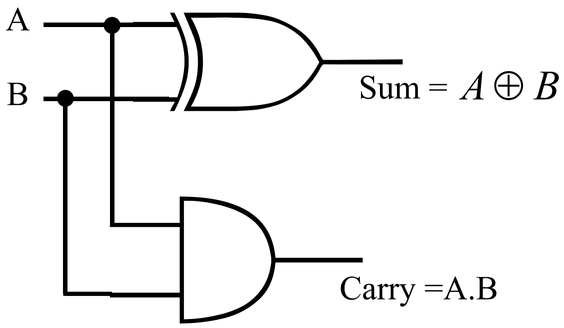


Figure : Logic diagram

$$SUM = \overline{A}B + A\overline{B} = A \oplus B$$

CARRY = A B

Inputs		Outputs	
A	B	CARRY	SUM
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

As per the laws of binary addition, we get,

When A= 0 and B = 0
0 + 0 ===> Carry = 0, Sum = 0

When A= 0 and B = 1
0 + 1 ===> Carry = 0, Sum = 1

When A= 1 and B = 0
1 + 0 ===> Carry = 0, Sum = 1

When A= 1 and B = 1
1 + 1 ===> Carry = 1, Sum = 0

The logic diagram, truth table and logic equations for SUM and CARRY are given in figure.

From the truth table (A, B and SUM), we find that an EX-OR can be used to produce the SUM. From the truth table (A, B and CARRY), we find that an AND gate can be used to produce the CARRY.

2.3 Full adder

When we add two bits, we may get a carry bit. Hence, an adder circuit needs to add three bits (ie. two input bits A and B and one carry bit C). The logic circuit which adds three bits is called Full adder. The truth table of full adder is given in table.

Inputs			Outputs	
A	B	C	CARRY	SUM
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

A full adder can be implemented using two half adders and an OR gate as shown in the figure.

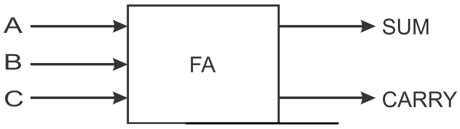


Figure : Block diagram

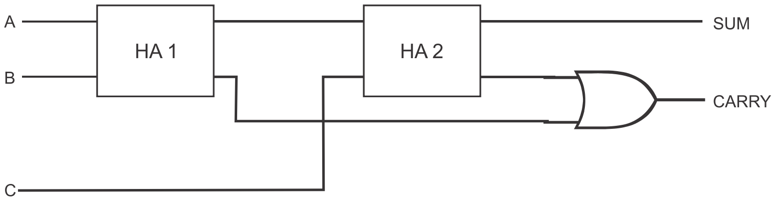


Figure : Block diagram of full adder using two half adders

The logic diagram and logic equations for SUM and CARRY are given in figure.

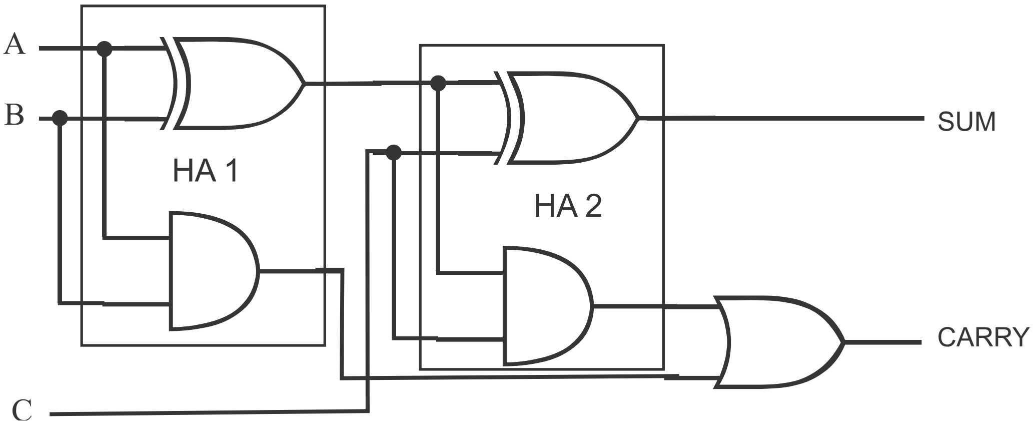


Figure : Logic diagram

$$\text{SUM} = A \oplus B \oplus C$$

$$\text{CARRY} = A\ \ B + (A \oplus B)\ C$$

The first Half adder HA1 is used to add A and B. Its SUM output and C are added by the second Half adder HA2. The SUM output of HA2 is the final SUM. The carry outputs of HA1 and HA2 are ORed to get the final CARRY output.

2.4 Half subtractor

Half subtractor subtracts one binary bit from another binary bit at a time. It has two inputs A and B, and two outputs DIFFERENCE and BORROW.

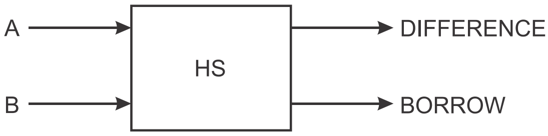


Figure : Block diagram of half subtractor

As per the laws of binary subtraction, we get,

When A= 0 and B = 0
0 - 0 ===> Borrow = 0, Difference = 0

When A= 0 and B = 1
0 - 1 ===> Borrow = 1, Difference = 1

When A= 1 and B = 0
1 - 0 ===> Borrow = 0, Difference = 1

When A= 1 and B = 1
1 - 1 ===> Borrow = 0, Difference = 0

The logic diagram, truth table and logic equations for DIFFERENCE and BORROW are given in figure.

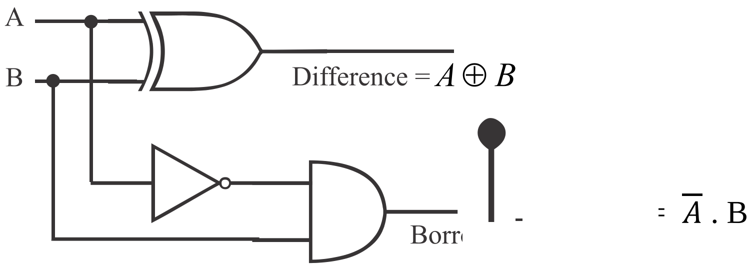


Figure : Logic diagram of half subtractor

Inputs		Outputs	
A	B	BORROW	DIFFERENCE
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

$$\text{DIFFERENCE} = \overline{A}B + A\overline{B} = A \oplus B$$

$$\text{BORROW} = \overline{A}\,B$$

From the truth table (A, B and DIFFERENCE), we find that an EX-OR gate can be used to produce the DIFFERENCE. From the truth table (A, B and BORROW), we find that A’.B will give the BORROW. Hence, a NOT gate and an AND gate are used to produce the BORROW.

2.5 Full subtractor

When we subtract two bits, we may get a borrow bit. Hence, a subtractor circuit needs to subtract three bits (ie. two input bits A and B and one borrow bit C). The logic circuit which subtracts three bits is called Full subtractor. The truth table of full subtractor is given in table.

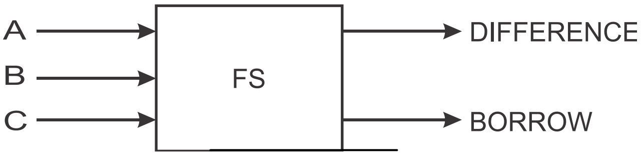


Figure : Block diagram of full subtractor

Inputs			Outputs	
A	B	C	BORROW	DIFFERENCE
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

A full subtractor can be implemented using two half subtractors and an OR gate as shown in the figure.

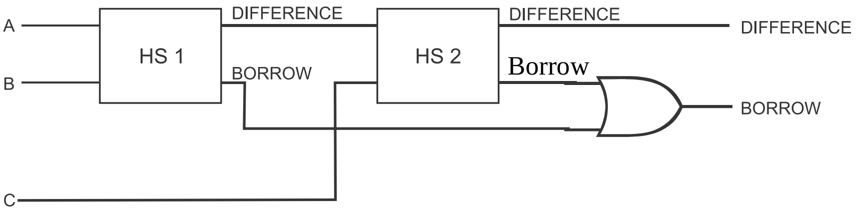


Figure : Block diagram of full subtractor using two half subtractors

The logic diagram and logic equations for DIFFERENCE and BORROW are given in figure.

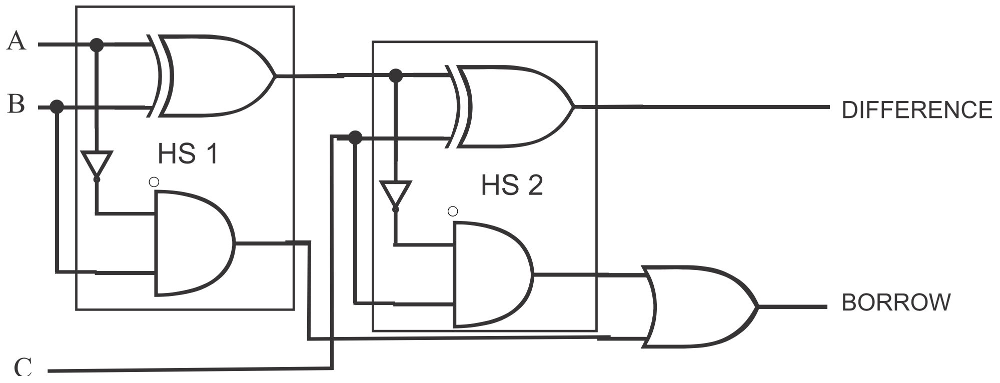


Figure : Logic diagram of full subtractor

$$\text{DIFFERENCE} = A \oplus B \oplus C$$

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

The first Halfsubtractor HS1 is used for (A - B). Then C is subtracted from (A-B) using the second Half subtractor HS 2. The DIFFERENCE output of HA2 is the final DIFFERENCE. The BORROW outputs of HS1 and HS2 are ORed to get the final BORROW output.

2.6Parallel Adder

Parallel adders are digital circuits that add ‘n’ bits in parallel. The symbol and logic diagram of parallel adder are shown in Figure.

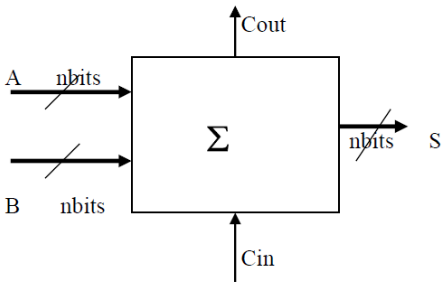


Figure : Symbol of Parallel adder

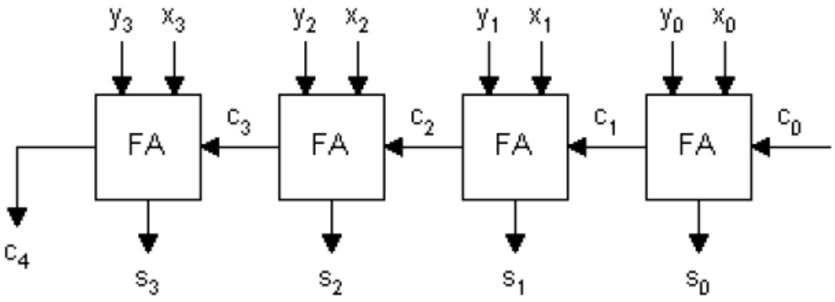


Figure : Logic diagram of Parallel adder

It is constructed by cascading full adders one after another. Each full adder stage is responsible for the addition of two binary digits and carry from the previous stage. The carryout of one stage is fed directly to the carry-in of the next stage. $x_3x_2x_1x_0$ and $y_3y_2y_1y_0$ are the data inputs, c_0 is the carry input, $s_3s_2s_1s_0$ is the sum output and c_4 is the carry output.

The augend's bits of 'x' are added to the addend bits of 'y' respectively of their binary position. Each bit addition creates a sum s_0, s_1, s_2, $s_3$and a carry out c_1, c_2, c_3, c_4. The carry out is then transmitted to the carry in of the next stage. The final result creates a sum of four bits $s_3s_2s_1s_0$plus a carry out c_4.

2.7 Serial adder

Parallel adders are digital circuits that add 'n' bits in serial, ie., one bit at a time. The logic diagram of serial adder is shown in Figure.

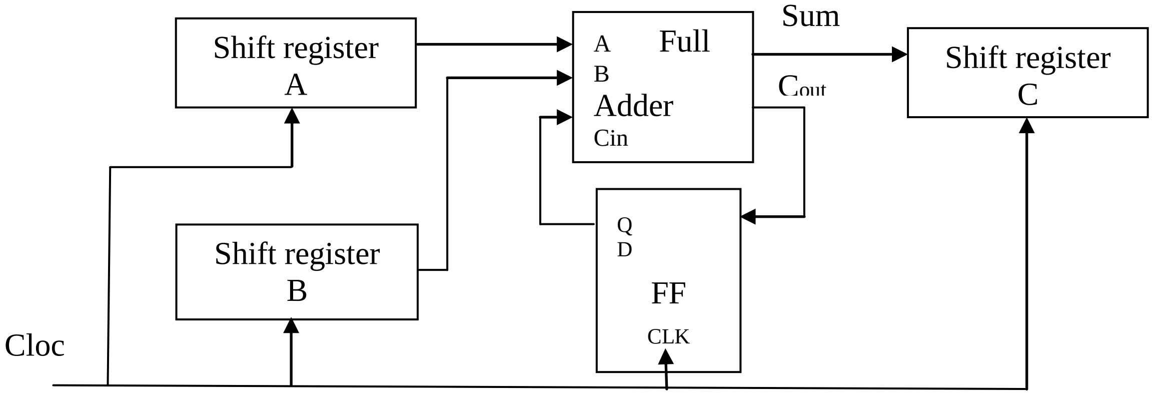


Figure : Logic diagram of Serial adder

The shift register (right shift) A holds the augend bits (first numer) and shift register B holds the addend bits (second number), after addition, the result will be avaialble in shift register C. All the shift registers are right shift registers.The full adder performs the bit by bit addition. The D-flip-flop is used to store the carry output generated after addition.

Initially, the D-flip-flip is cleared and addition starts with the least significant bits (LSBs) of both shift registers. After each clock pulse, data within the shift registers A and B are shifted right by 1-bit. The full adder adds the bits along with the carry of the previous state (from the D flip-flop) and the sum is stored in the output shift register C.

2.8 BCD adder

A BCD adder is a circuit that adds two BCD digits and produces a sum digit in BCD format. BCD numbers use 10 digits, 0 to 9 which are represented in the binary form 0000 to 1001, (i.e.) each BCD digit is represented as a 4-bit binary number.

For adding two BCD numbers, the following procedure should be followed:

1. Add two BCD numbers using ordinary binary addition.
2. If the 4-bit sum is equal to or less than 9, no correction is needed. The sum is in proper BCD form.

3. If the 4-bit sum is greater than 9 or if a carry is generated from the 4-bit sum, the sum is invalid.
4. To correct the invalid sum, add 0110_2 (ie. decimal 6) to the 4-bit sum. If a carry results from this addition, add it to the next higher-order BCD digit.

Thus to implement BCD adder we require:

- A 4-bit binary adder for initial addition
- Logic circuit to detect sum greater than 9 and
- One more 4-bit adder to add 0110_2 if the sum is greater than 0 or carry is 1.

Figure shows the logic diagram of a BCD adder.

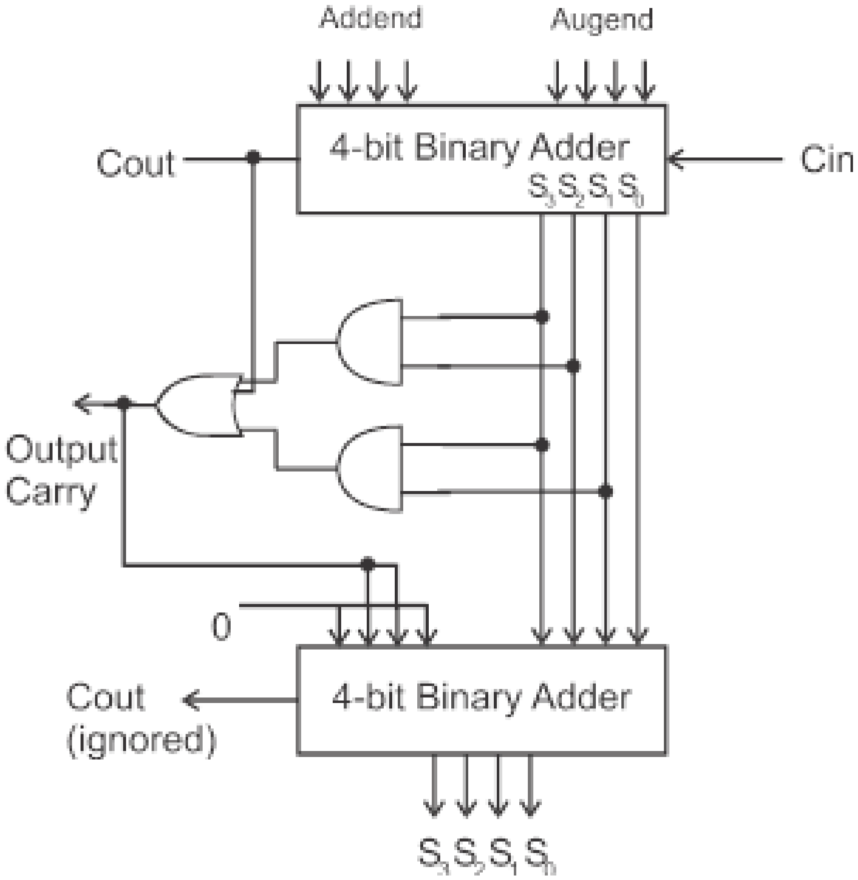


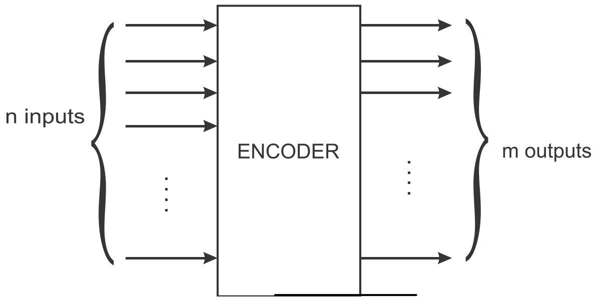
Figure : BCD adder

Output Carry = $S_1.S_3 + S_2.S_3 + C_{out}$

As shown in Figure, the two BCD numbers, together with input carry, are first added in the top 4-bit binary adder to produce a binary sum. When the output carry is equal to zero (i.e. when sum <=9 and Cout=0) nothing (zero) is added to the binary sum. When it is equal to one (i.e. when sum>9 or Cout=1), binary 0110 is added to the binary sum through the bottom 4-bit binary adder. The output carry generated from the bottom binary adder can be ignored.

2.9 Encoder

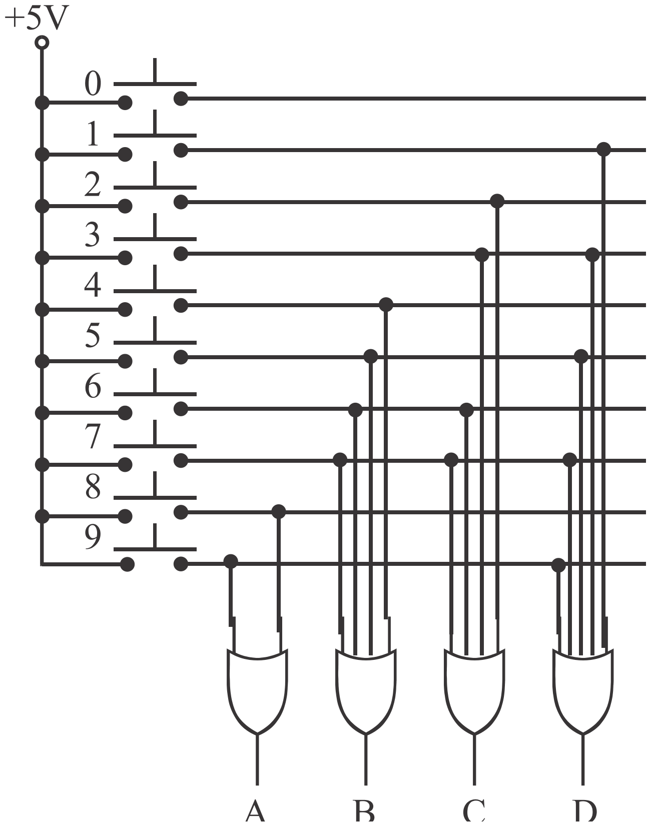
An encoder is a code converter circuit which is used to convert an active input signal to a coded output signal. This is shown in figure. It should be noted that encoders have more number of input lines and less number of output lines.



There are ‘n’ input signals, only one of which is active. The encoder circuit converts this active input to ‘m’ bit coded output (n > m). Decimal to BCD converter is a good example for encoder.

Decimal to BCD encoder

The logic diagram of Decimal to BCD encoder is shown in figure.



There are 10 input switches corresponding to ten decimal numbers 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. The BCD outputs are taken from the OR gates. When any one of the input switches is pressed, the corresponding output will be generated in BCD form. The truth table is shown in figure.

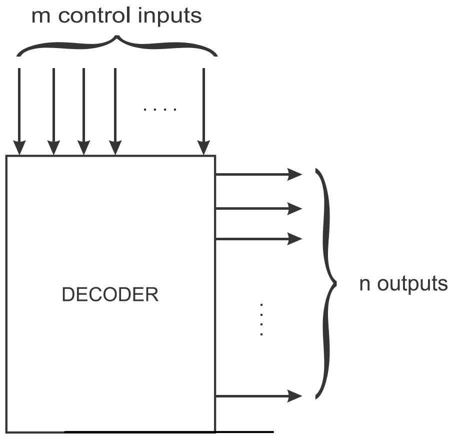
Decimal input switch pressed	BCD outputs			
	A	B	C	D
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

For example, when switch ‘3’ is pressed, only the OR gates C and D get high input, and hence the output ABCD will be 0011. Similarly when switch ‘9’ is pressed, only the OR gates A and D get high input and the output will be 1001.

The Decimal to BCD encoder is also available in IC form. The IC number is 74147. It has 16 pins.

2.10 Decoder

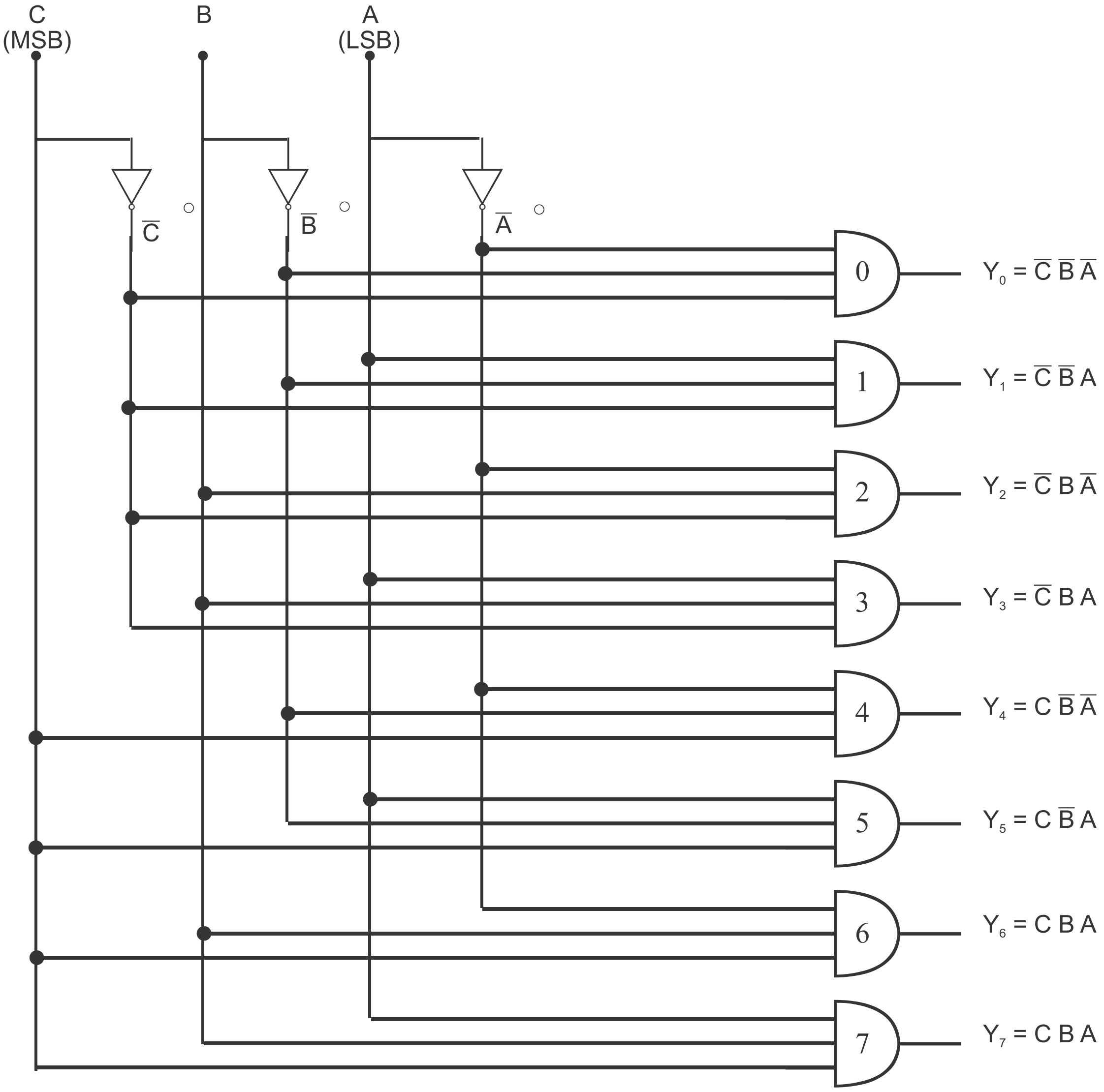
Decoder is a logic circuit which converts binary data in one coded form to another coded form. Decoders have less number of input lines and more number of output lines. Decoders have no data input but they have only control inputs.



The control input lines are also called address lines. Any one of the output lines will be enabled depends upon the selection made in the address lines.

3-to-8 Decoder

The 3-to-8 decoder has eight output lines and three control input lines. Any one of the output will go high (ie. logic ‘1’) depends on the conditions of the control lines. It is also called 1 of 8 decoder. The logic diagram and truth table of 3-to8 decoder are given in figure.


C
(MSB)
B
A
(LSB)
$\overline{C}$
$\overline{B}$
$\overline{A}$
0
1
2
3
4
5
6
7
$Y_0 = \overline{C}\,\overline{B}\,\overline{A}$
$Y_1 = \overline{C}\,\overline{B}\,A$
$Y_2 = \overline{C}\,B\,\overline{A}$
$Y_3 = \overline{C}\,B\,A$
$Y_4 = C\,\overline{B}\,\overline{A}$
$Y_5 = C\,\overline{B}\,A$
$Y_6 = C\,B\,A$
$Y_7 = C\,B\,A$

Control Input			Output							
C	B	A	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	1	0	0
0	1	1	0	0	0	0	1	0	0	0
1	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	1	0	0	0	0	0
1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	0	0	0	0	0	0	0

In the logic diagram, the AND gates are numbered from 0 to 9. When the control input lines CBA are 000, all the inputs $\overline{CBA}$ of 0th AND gate are 1. Hence, Y_0 will be ‘1’. But all other AND gates will have at least one input as ‘0’. Hence, their outputs are ‘0’. When the control input lines CBA are 001, all the inputs $\overline{CB}$A of 1st AND gate are 1. Hence, Y_1 will be ‘1’. But all other AND gates will have atleast one input as ‘0’. Hence, their outputs are ‘0’.

Similarly, the outputs for all other control input combinations can be found. The 3-to-8 decoder is available in IC form also. The IC number is 74138.

2.11 BCD to seven segment decoder

A 7-segment display is shown in figure. It has seven flat LEDs arranged in such a way to display decimal numbers from 0 to 9 and other symbols. The segments are labeled as a, b, c, d, e, f and g. When LED segments a, b, c, d and g are ON, we get the number 3 as shown in figure. The display patterns for numbers from0 to 9 are shown in figure.

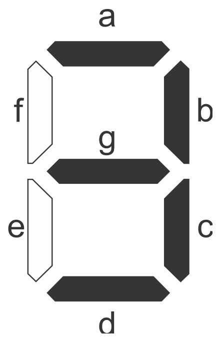


Figure : 7-segment display showing decimal number 3

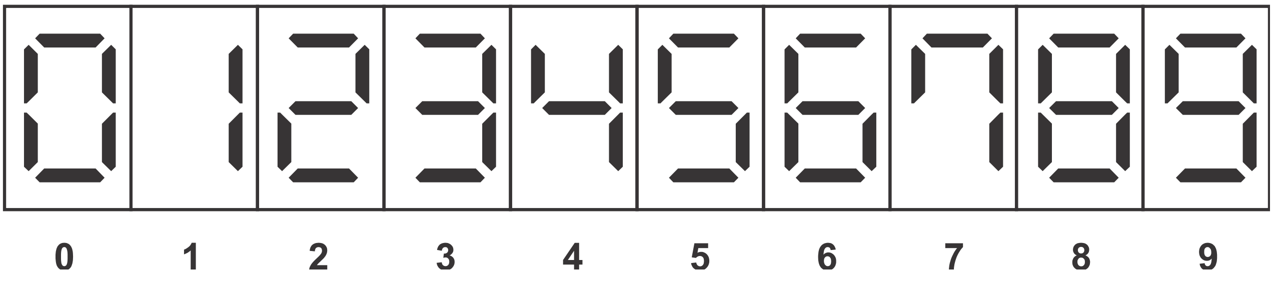


Figure : Display pattern for decimal numbers

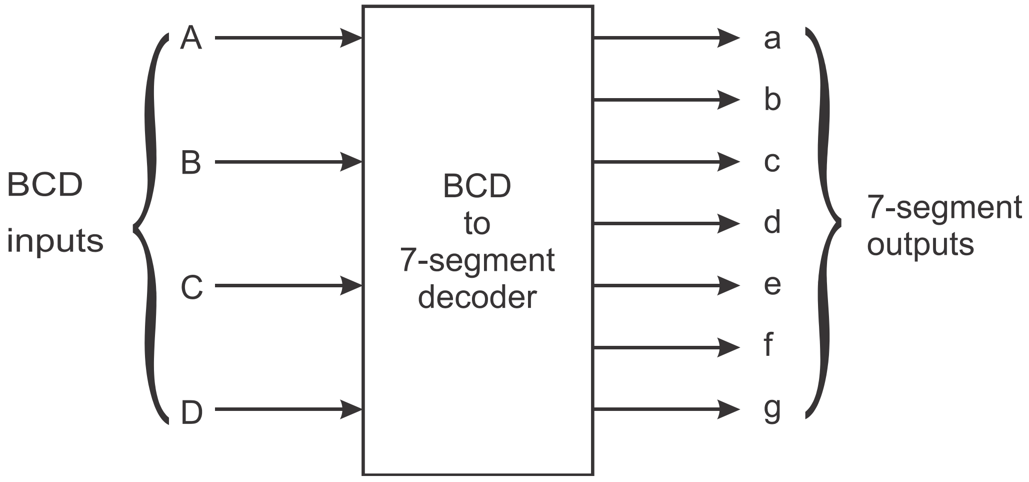


Figure : Block diagram of BCD to Seven segment decoder

BCD to 7-segment decoder is a logic circuit which converts the BCD (4 bits) code to 7-segment display code (7 bits). The functional diagram and truth table are shown in figure.

BCD input				7-segment output							Display
A	B	C	D	a	b	c	d	e	f	g	
0	0	0	0	1	1	1	1	1	1	0	0
0	0	0	1	0	1	1	0	0	0	0	1
0	0	1	0	1	1	0	1	1	0	1	2
0	0	1	1	1	1	1	1	0	0	1	3
0	1	0	0	0	1	1	0	0	1	1	4
0	1	0	1	1	0	1	1	0	1	1	5
0	1	1	0	1	0	1	1	1	1	1	6
0	1	1	1	1	1	1	0	0	0	0	7
1	0	0	0	1	1	1	1	1	1	1	8
1	0	0	1	1	1	1	1	0	1	1	9

Figure : Truth table of BCD to Seven segment decoder

From the truth table, we can find the logic equations for the outputs.

$$a = \overline{ABCD} + \overline{AB}\ C\ \overline{D} + \overline{AB}\ C\ D + \overline{A}\ \ B\ \overline{C}\ D + \overline{A}\ \ B\ C\ \overline{D} + \overline{A}\ \ B\ C\ D + A\ \overline{BCD} + A\ \overline{BC}\ D$$

$$b = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}\ D + \overline{A}\overline{B}\ C\ \overline{D} + \overline{A}\overline{B}\ C\ D + \overline{A}\ B\ \overline{C}\overline{D} + \overline{A}\ B\ C\ D + A\ \overline{B}\overline{C}\overline{D} + A\ \overline{B}\overline{C}\ D$$

$$c = \overline{ABCD} + \overline{ABC}\ D + \overline{AB}\ C\ D + \overline{A}\ B\ \overline{CD} + \overline{A}\ B\ \overline{C}\ D + \overline{A}\ B\ C\ \overline{D} + \overline{A}\ B\ C\ D + A\ \overline{BCD} + A\ \overline{BC}\ D$$

$$d = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{AB}\ C\ \overline{D} + \overline{AB}\ C\ D + \overline{A}\ \ B\ \overline{C}\ D + \overline{A}\ \ B\ C\ \overline{D} + A\ \overline{B}\overline{C}\overline{D} + A\ \overline{BC}\ D$$

$$e = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{AB}\ C\ \overline{D} + \overline{A}\ \ B\ C\ \overline{D} + A\ \overline{B}\overline{C}\overline{D}$$

$$f = \overline{ABCD} + \overline{A}\ B\ \overline{CD} + \overline{A}\ \ B\ \overline{C}\ D + \overline{A}\ \ B\ C\ \overline{D} + \overline{A}\ B\ C\ D + A\ \overline{BCD} + A\ \overline{BC}\ D$$

$$g = \overline{AB}\ C\ \overline{D} + \overline{AB}\ C\ D + \overline{A}\ B\ \overline{CD} + \overline{A}\ B\ \overline{C}\ D + \overline{A}\ \ B\ C\ \overline{D} + A\ \overline{BCD} + A\ \overline{BC}\ D$$

By simplifying the above equations using K-map, we get the following simplified logic equations for the seven segments :

$$a = A + C + B\ D + \overline{BD}$$

$$b = \overline{B} + \overline{CD} + C\ D$$

$$c = B + \overline{C} + D$$

$$d = \overline{BD} + \overline{B}\ C + B\ \overline{C}\ D + A$$

$$e = \overline{BD} + C\ \overline{D}$$

$$f = A + \overline{CD} + B\ \overline{C} + B\ \overline{D}$$

$$g = A + B\,\overline{C} + \overline{B}\,C + C\,\overline{D}$$

From the above equations, we can draw the logic diagram for all the outputs. NOT gates are used for complements, AND gates are used for '.' operations and OR gates are used for '+' operations. Hence, for segment 'a' we need two 2-input AND gates and one 4-input OR gate. The final logic diagram for the BCD to 7-segment decoder is shown in figure.

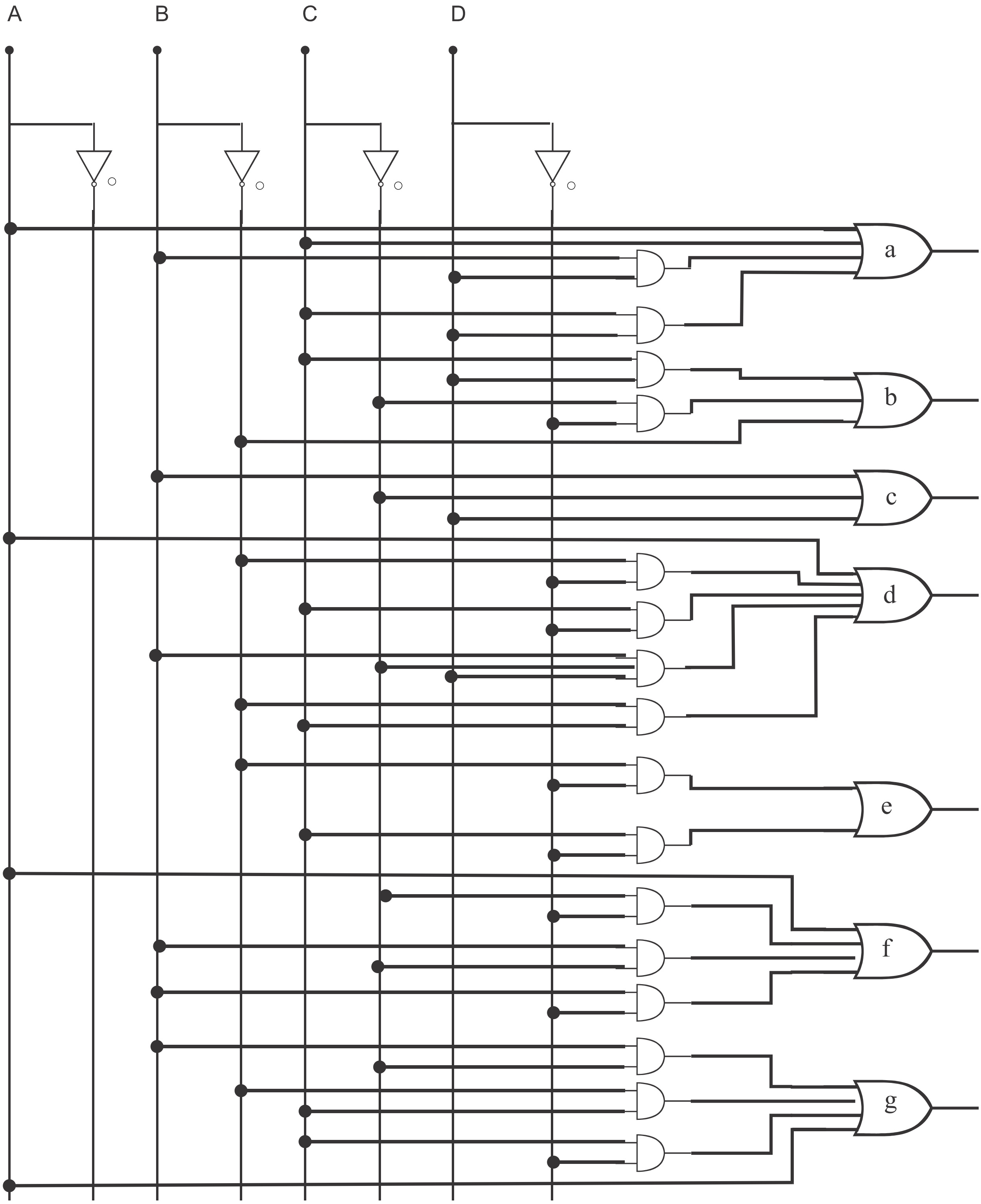
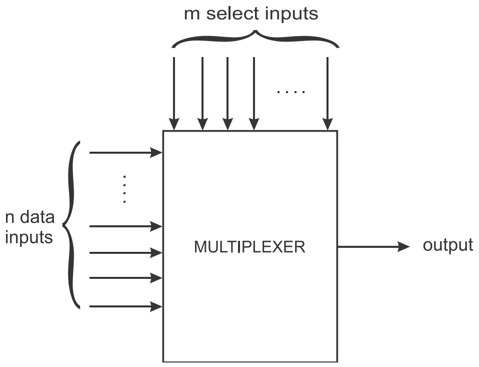


Figure :Logic diagram of BCD to 7-segment decoder

BCD to 7-segment display decoder logic is available in IC form also. IC 7447 is used for common anode displays and IC 7448 is used for common cathode displays.

2.12 Multiplexer (MUX)

Multiplex means ‘many to one’. A multiplexer is a logic circuit with many inputs and only one output. We can select from any one of the data inputs to the output by using the control input signals. Multiplexer is also called ‘data selector’ and the control inputs are called ‘select’ inputs.



If there are ‘m’ select lines, we can have a maximum of 2^m data input lines. Hence, $n \leq 2^m$.

2-to-1 Multiplexer

In a 2-to-1 multiplexer, there are 2 data input lines (D0, D1) and one output line (Y). We need at least 1 select line (A) to select any one of the 2 inputs to the output line. The logic diagram, truth table and logic equation of 2-to-1 multiplexer are shown in figure.

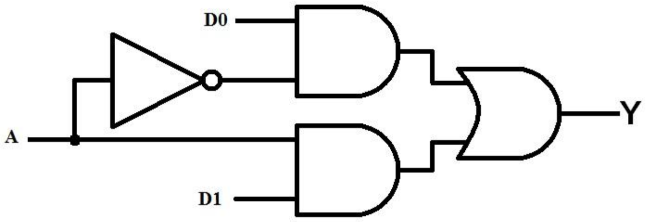


Figure : Logic diagram of 2-to-1 multiplexer

Select line	Output
A	Y
0	D_0
1	D_1

Figure : Truth table of 2-to-1 multiplexer

$$Y = \overline{A}\, D_0 + A\, D_1$$

When A = 0, the first AND gate is enabled. Hence, the input D_0 (0 or 1) will be available at the output. When A = 1, the second AND gate is enabled and the output Y will be D_1.
IC 74157 is quad 2-to-1 multiplexer. It has four sets of 2-to-1 multiplexer in a single package. There are 16 pins in this IC.

4-to-1 Multiplexer

In a 4-to-1 multiplexer, there are 4 data input lines and one output line. We need at least 2 select lines to select any one of the 4 inputs to the output line. The logic diagram, truth table and logic equation of 4-to-1 multiplexer are shown in figure.

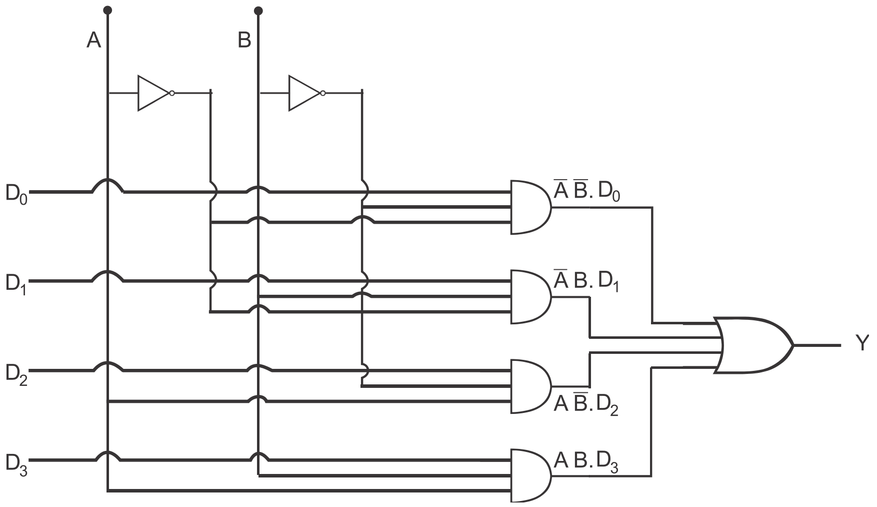


Figure : Logic diagram of 4-to-1 multiplexer

Select lines		Output
A	B	Y
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

Figure : Truth table of 4-to-1 multiplexer

$$Y = \overline{AB}\ D_0 + \overline{A}\ B\ D_1 + A\ \overline{B}\ D_2 + A\ B\ D_3$$

When A = 0 and B = 0, the first AND gate is enabled. Hence, the input D_0 (0 or 1) will be available at the output. When A = 0 and B = 1, the second AND gate is enabled and the output Y will be D_1. When A = 1 and B = 0, we will get D_2 at the output. Similarly, Y = D_3 when A = 1 and B = 1.

IC 74153 is a dual 4-to-1 multiplexer. It has two sets of 4-to-1 multiplexer in a single package. There are 16 pins in this IC.

8-to-1 Multiplexer

In a8-to-1 multiplexer, there are 8 data input lines and one output line. We need at least 3 select lines to select any one of the 8 inputs to the output line. The logic diagram, truth table and logic equation of 8-to-1 multiplexer are shown in figure.

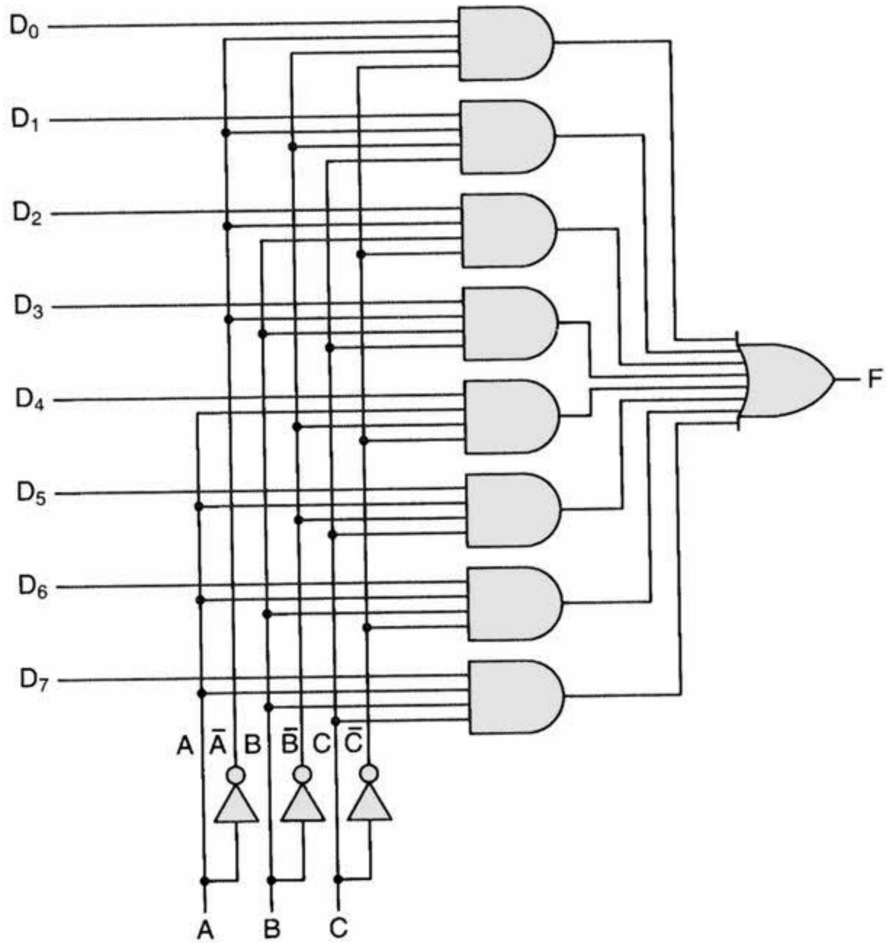


Figure : Logic diagram of 8-to-1 multiplexer

Select lines			Output
A	B	C	Y
0	0	0	D_0
0	0	1	D_1
0	1	0	D_2
0	1	1	D_3
1	0	0	D_4
1	0	1	D_5
1	1	0	D_6
1	1	1	D_7

Figure : Truth table of 8-to-1 multiplexer

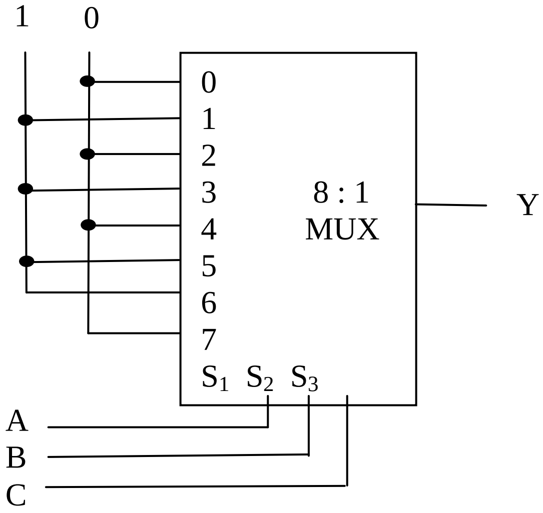
$$Y = \overline{ABC}\ D_0 + \overline{AB}\ C\ D_1 + \overline{A}\ B\ \overline{C}\ D_2 + \overline{A}\ B\ C\ D_3 + A\ \overline{BC}\ D_4 + A\ \overline{B}\ C\ D_5 + A\ B\ \overline{C}\ D_6 + A\ B\ C\ D_7$$

When A = 0, B = 0 and C = 0, the first AND gate is enabled. Hence, the input D_0 (0 or 1) will be available at the output. When A = 1, B = 1 and C = 1, the last (eighth) AND gate is enabled and the output Y will be D_7. Similarly, the output is D2, D3, D4, D5 and D6 when the inputs ABC are 001, 010, 011, 100, 101 and 110 respectively. IC 74151 is the 8-to-1 multiplexer. There are 16 pins in this IC.

Applications of MUX

Multiplexers are used to implement combinational logic functions.

The function $F\ (A, B, C) = \sum m\ (1, 3, 5, 6)$ can be implemented using an 8-1 multiplexer, as shown in figure.



The variables A, B and C are applied to the select lines. The minterms to be included (1, 3, 5 and 6) are chosen and their corresponding input lines are connected to 1 (Vcc). The remaining input lies (0, 2, 4 and 7) are connected to 0 (GND). When the select lines ABC are 000, the input line 0 is selected and we get 0 at the output because the input line 0 is connected to GND. Similarly, when the select lines ABC are 101, the input line 5 is selected and we get 1 at the output because the input line 5 is connected to Vcc.

2.13 Demultiplexer (DEMUX)

Demultiplex means 'one to many'. A demultiplexer is a logic circuit with only one input and many outputs. We can send the data input to any one of the outputs by using the control input signals. Demultiplexer is also called 'data distributer'.

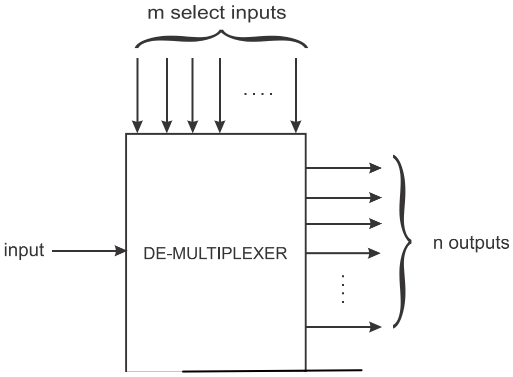


Figure :Demultiplexer

If there are 'm' select lines, we can have a maximum of 2^m output lines. Hence, $n \leq 2^m$.

1-to-2Demultiplexer

In a 1-to-2demultiplexer, there are 1 data input line and 2 output lines. We need at least 1 select line to select the output line. The logic diagram, truth table and logic equations of 1-to-2demultiplexer are shown in figure.

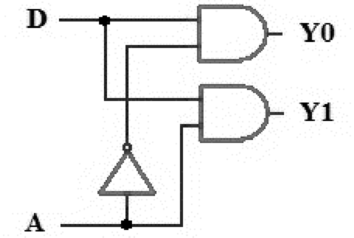


Figure : Logic diagram of 1-to-2demultiplexer

Select lines	Outputs	
A	Y_1	Y_0
0	0	D
1	D	0

Figure : Truth table of 1-to-2demultiplexer

$$Y_0 = \overline{A}\ D$$
$$Y_1 = A\ D$$

When A = 0, the first AND gate is enabled. Hence, the input D (0 or 1) will be available at the output Y_0. When A = 1, the second AND gate is enabled and the output Y_1will get the input D.

1-to-4 Demultiplexer

In a 1-to-4 demultiplexer, there are 1 data input line and 4 output lines. We need at least 2 select lines to select the output line. The logic diagram, truth table and logic equations of 1-to-4 demultiplexer are shown in figure.

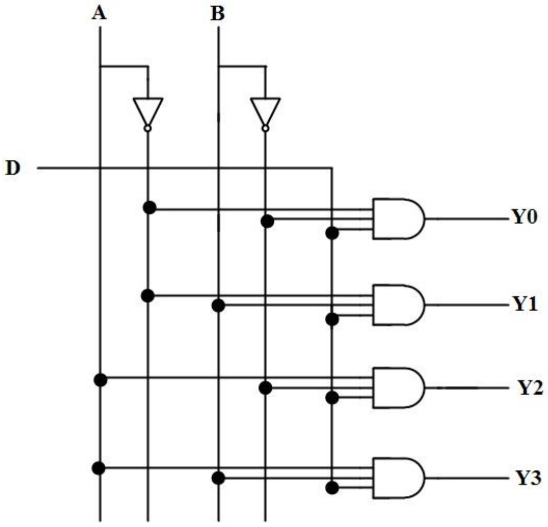


Figure : Logic diagram of 1-to-4 demultiplexer

Select lines		Outputs			
A	B	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

Figure : Truth table of 1-to-4 demultiplexer

$$Y_0 = \overline{AB}\ D$$
$$Y_1 = \overline{A}\ B\ D$$
$$Y_2 = A\ \overline{B}\ D$$
$$Y_3 = A\ B\ D$$

When A = 0 and B = 0, the first AND gate is enabled. Hence, the input D (0 or 1) will be available at the output Y_0. When A = 0 and B = 1, the second AND gate is enabled and the output Y_1will get the input D. When A = 1 and B = 0, we will get D at the output Y_2. Similarly, Y_3 = D when A = 1 and B = 1.

IC 74139 is a dual 1-to-4 demultiplexer. It has two sets of 1-to-4 demultiplexer in a single package. There are 16 pins in this IC.

1-to-8 Demultiplexer

In a 1-to-8demultiplexer, there are 1 data input line and 8 output lines. We need at least 3 select lines to select the output line. The logic diagram, truth table and logic equations of 1-to-8demultiplexer are shown in figure.

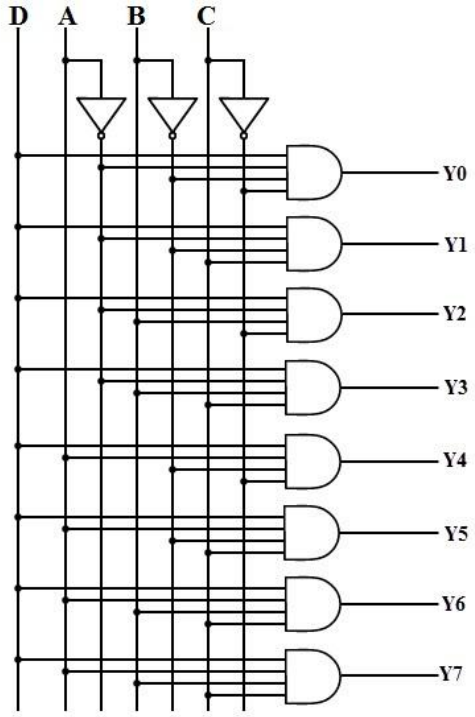


Figure : Logic diagram of 1-to-8demultiplexer

Select lines			Outputs							
A	B	C	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0

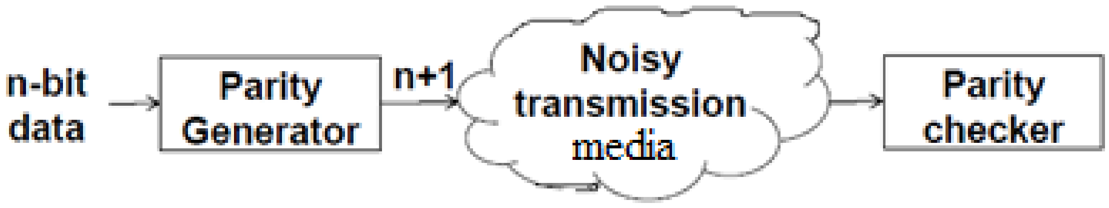
Figure : Truth table of 1-to-8demultiplexer

$$Y_0 = \overline{ABC}\ D$$
$$Y_1 = \overline{AB}\ C\ D$$
$$Y_2 = \overline{A}\ B\ \overline{C}\ D$$
$$Y_3 = \overline{A}\ B\ C\ D$$
$$Y_4 = A\ \overline{BC}\ D$$
$$Y_5 = A\ \overline{B}\ C\ D$$
$$Y_6 = A\ B\ \overline{C}\ D$$
$$Y_7 = A\ B\ C\ D$$

When A = 0, B = 0 and C = 0, the first AND gate is enabled. Hence, the input D (0 or 1) will be available at the output Y_0. When A = 1, B = 1 and C = 1, the last (eighth) AND gate is enabled and the input D will be available at the output Y_7.Similarly, the input D can reach Y_1, Y_2, Y_3, Y_4, Y_5 and Y_6 when the control inputs ABC are 001, 010, 011, 100, 101 and 110 respectively. IC 74138 is the 1-to-8demultiplexer. There are 16 pins in this IC.

2.14 Parity generator and checker

The most common error detection code used is the parity bit. A parity bit is an extra bit included with a binary message to make the total number of 1's either odd or even. In case of even parity, the parity bit is chosen so that the total number of 1's in the coded message is even including the parity bit. Alternatively, odd parity can be used in which the total number of 1's in the coded message is made odd including the parity bit.



During transfer of information, the message at the sending-end is applied to a parity generator where the parity bit is generated. At the receiving end a parity checker is used to detect single bit error in the received data.

Even parity generator

In an even parity system, the number of ones in data bits including the parity bit must be an even number. For example, the even parity bit for the data bits 0101 must be 0 because we must have an even number of ones in the data bits including the parity bit. EX-OR gates are used in parity generator and checker circuits. A 4-bit even parity generator circuit, its truth table and logic equation are shown in figure.

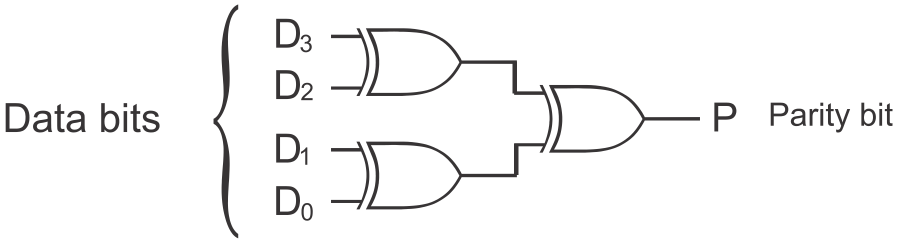


Figure : Logic diagram of 4-bit Even parity generator

Data input				Output (Even parity)
D_3	D_2	D_1	D_0	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Figure : Truth table of 4-bit Even parity generator

$$P = D_3 \oplus D_2 \oplus D_1 \oplus D_0$$

The four data bits along with the parity bit $D_3D_2D_1D_0P$ are transmitted to the receiver circuit.

Even parity Checker

An even-parity checker will produce an error (“1”) if the number of bits in the entire group of digits including the parity bit is notan even number. A 4-bit even parity checker logic diagram, truth table and its logic equation are given in figure.

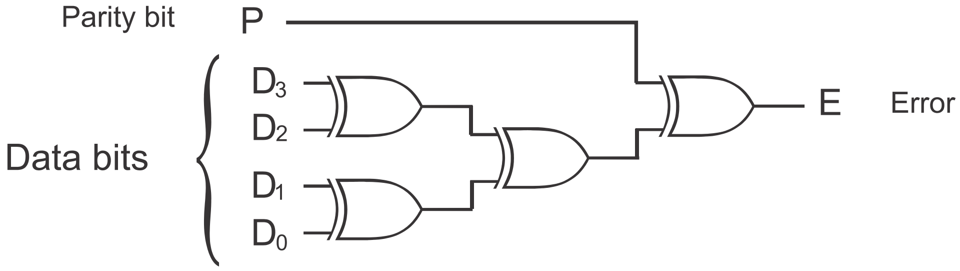


Figure : Logic diagram of 4-bit Even parity checker

Data input				Parity bit Input	Error Bit
D_3	D_2	D_1	D_0	P	E
0	0	0	0	0	0
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	1	0
0	1	0	1	0	0
0	1	1	0	0	0
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	1	0
1	1	0	0	0	0
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	0

Figure : Truth table

$$P = D_3 \oplus D_2 \oplus D_1 \oplus D_0 \oplus P$$

When data bits matches with the parity bit, then the Error E is zero. If any one of the data bits changes from 0 to 1 or 1 to 0, then the Error bit E will be 1 indicating the Parity Error.

IC 74280 is a 9-bit Parity generator / checker.

☺☺☺☺☺☺

UNIT - III

SEQUENTIAL CIRCUITS

Sequential Circuits

In sequential circuits, the output depends not only on the present input conditions but also on the previous output conditions i.e. the past history of the inputs. The past history is provided by the feedback from the output back to the input.

3.1 Flip-flops

The basic building block for sequential logic circuits is the flip-flop(FF). Flip-flop is a bi-stable logic element with one or more inputs and two outputs. The outputs are complement to each other. A flip-flop can store one bit of binary data ‘0’ or ‘1’. Flip-flops are used in counters, shift registers and memory devices. There are six types of flip-flops:

1. SR Flip-flop
2. Clocked SR flip-flop
3. JK flip-flop
4. JK Master Slave flip-flop
5. D flip-flop and
6. T flip-flop

Flip-flop is a one bit memory cell. It can store one bit of information.

3.1.1 SR (or RS) flip-flop

It is also called Set/Reset flip-flop. The logic symbol of SR flip-flop is shown in figure:

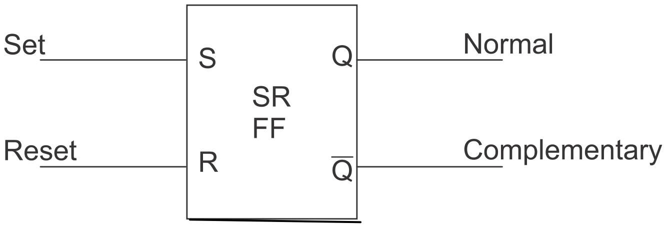


Figure : Symbol of SR flip-flop

The SR flip-flop has two inputs and two outputs. The inputs are S (Set) and R (Reset). The outputs Q and $\overline{Q}$ are complements to each other. i.e. when Q = 0, $\overline{Q}$ will be 1 and when Q = 1, $\overline{Q}$ will be 0. The logic diagram of SR flip-flop using NAND gates and the truth table are shown in figure.

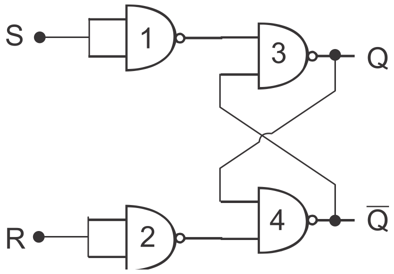


Figure : Logic diagram of SR flip-flop

Inputs		Outputs		Condition
S	R	Q	$\overline{Q}$	
0	0	Previous value	Previous value	No change
0	1	0	1	Reset
1	0	1	0	Set
1	1	Forbidden		Not used

Figure : Truth table of SR flip-flop

NAND gates 3 and 4 form the basic flip-flop circuit. The output of gate 3 (Q) is connected as one of the input to gate 4. Similarly, the output of gate 4 ($\overline{Q}$) is connected as one of the input to gate 3. This feedback type of connection is called cross coupled connection. NAND gates 1 and 2 are used as NOT gates for complementing S and R.

When S=1 and R=1, the outputs Q and $\overline{Q}$ will not change and the previous values are retained. When S=0 and R=1, the output Q will become 0 and $\overline{Q}$ will become 1. This condition is called RESET condition. i.e. the output Q is reset to zero. When S=1 and R=0, the output Q will become 1 and $\overline{Q}$ will become 0. This condition is called SET condition. i.e. the output Q is set to ONE.

But, when S=1 and R=1, both the outputs Q and $\overline{Q}$ will become 1. This is not allowed in digital circuits because the outputs Q and $\overline{Q}$ are complement to each other. Hence, this state is called forbidden state and we should not use the SR flip-flop with S=R=1.

3.1.2 Clocked SR (CSR) flip-flop

Normally, sequential logic circuits work in sequence with on the occurrence of clock signal. Clock signal is a train of pulses in the form of square wave. Clocked SR flip-flop is similar to SR flip-flop with an additional clock input. The output of the FF changes only on the arrival of the clock signal. The logic symbol, logic circuit diagram and truth table of CSR FF are shown in figure.

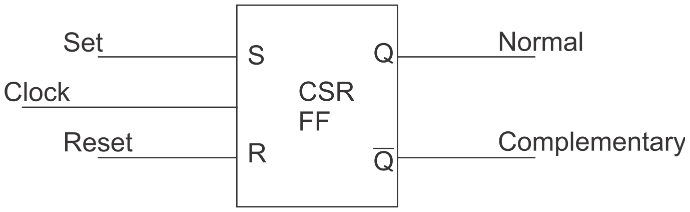


Figure : Logic symbol of CSR FF

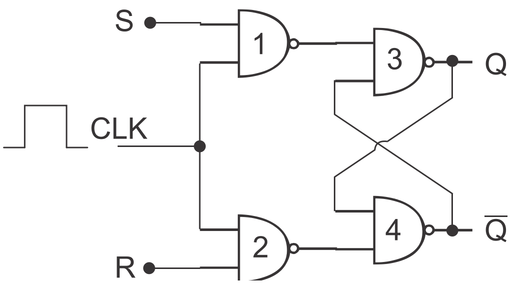


Figure : Logic circuit of CSR flip-flop

Inputs			Outputs		Condition
CLK	S	R	Q	$\overline{Q}$	
⎍	0	0	Previous value	Previous value	No change
⎍	0	1	0	1	Reset
⎍	1	0	1	0	Set
⎍	1	1	Forbidden		Not used

Figure : Truth table of CSR flip-flop

The inputs S and R will be allowed to enter the circuit only when the CLK input is present i.e. at logic ‘1’. When there is no clock pulse (i.e. at logic ‘0’), the flip-flop will retain the previous state. The operation of CSR flip-flop is similar to SR flip-flop when the clock signal is ‘1’.

3.1.3 JK Flip-flop

The NOT USED condition of SRFF i.e. S=1, R=1 condition is eliminated in the JK flip-flop. The condition J=1, K=1 is used to toggle the flip-flop. Toggling means, when the previous output is ‘0’, the present output will be ‘1’. Similarly, when the previous output is ‘1’, the present output will be’0’.

JK FF has three inputs J, K and CLK and two outputs Q and $\overline{Q}$. Preset (Pr) and Clear (Cr) inputs are also provided in the JK FF. The logic symbol, logic circuit diagram and truth table of JK FF are shown in figure.

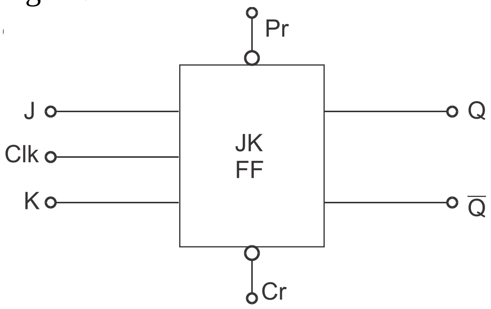


Figure : Logic symbol of JK FF

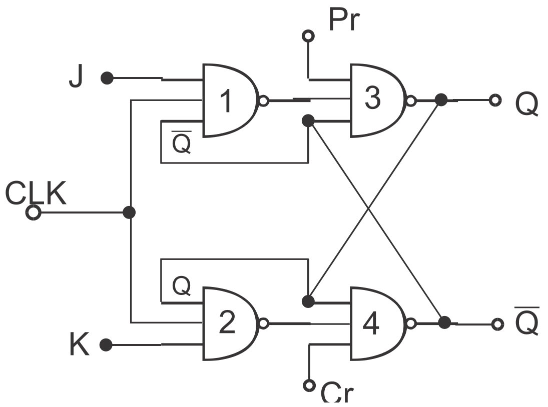


Figure : Logic circuit diagram of JK FF

Inputs			Outputs		Condition
CLK	J	K	Q	$\overline{Q}$	
⎍	0	0	Previous value	Previous value	No change
⎍	0	1	0	1	Reset
⎍	1	0	1	0	Set
⎍	1	1	Complement of Previous value	Complement of Previous value	Toggle

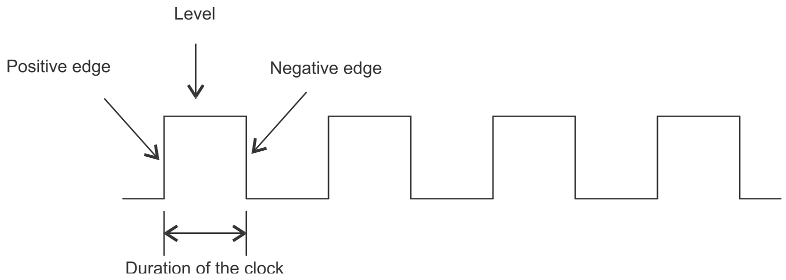
Figure : Truth table of JK FF

When J=0 and K=0, there will be no change in the output. Q and $\overline{Q}$ will retain the previous state. When J=0 & K = 1 and CLK = 1, Q will become ‘0’ and $\overline{Q}$ will become ‘1’, This condition is called RESET condition. When J = 1 & K = 0 and CLK = 1, Q will become ‘1’ and $\overline{Q}$will become ‘0’. This condition is called SET condition. When J =1 and K =1, the output will toggle repeatedly on the arrival of the successive clock signal.

The PRESET and CLEAR inputs are used to set and clear the FF irrespective of the application of clock pulse. The bubbles shown in Pr and Cr inputs represent the active low inputs. That means the signal is active when it is ‘0’. The FF will SET when Pr = 0 and CLEAR when Cr = 0.

Racing problem

In JK flip-flop, when J = 1, K = 1 and when the clock pulse duration is more, the FF will toggle many times (more than one time). Hence, we cannot estimate the final output.



This problem in JK FF is called racing problem and this condition is called race-around condition. The racing problem can be avoided by using JK Master Slave (JK MS) flip-flop or by using edge triggering techniques.

3.1.4 JK Master Slave (JKMS) flip-flop

The racing problem in JK FF can be avoided by using JKMS FF. The logic symbol of JKMS FF is show in figure.

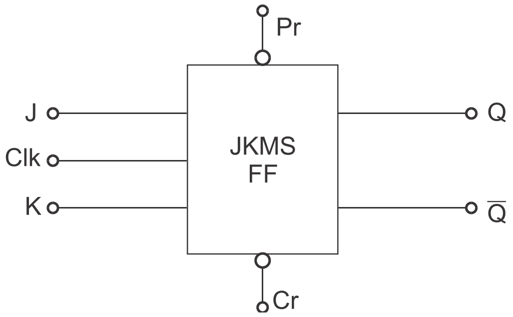


Figure : Logic symbol of JKMS FF

In JKMS FF there are two JKFFs, one Master JKFF and one Slave JKFF. The CLK pulse of the master section is inverted and then given to the CLK input of the slave section.

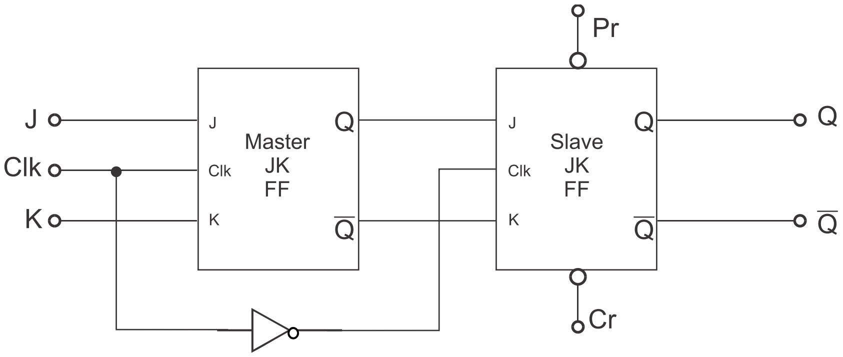


Figure : JKMS FF using two JK FF

The logic circuit diagram and truth table of JKMS FF are shown in figure.

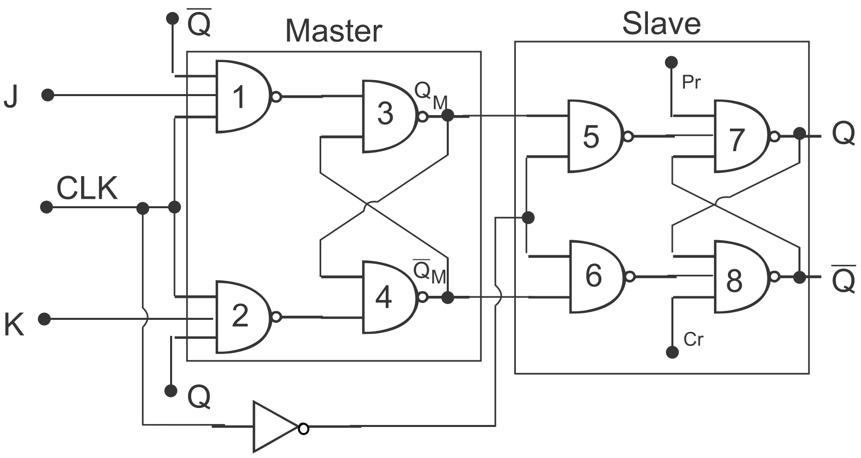


Figure : Logic circuit diagram of JKMS FF

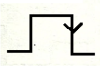
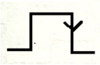
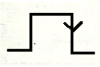
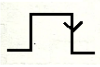
Inputs			Outputs		Condition
CLK	J	K	Q	$\overline{Q}$	
	0	0	Previous value	Previous value	No change
	0	1	0	1	Reset
	1	0	1	0	Set
	1	1	Complement of Previous value	Complement of Previous value	Toggle

Figure : Truth table of JKMS FF

NAND gates 1, 2, 3 and 4 form the Master section and NAND gates 5, 6, 7 and 8 form the slave section. NOT gate is used to generate the inverted clock for the slave section.

When CLK = 1, the Master section in enabled and the outputs Q_M and $\overline{Q}_M$ respond to the inputs J and K. At this time, the Slave section is inhibited (not enabled) because the CLK to the slave section is 0. When CLK goes LOW, the Master section is inhibited and the Slave section is enabled, because its CLK input is HIGH. Therefore, the outputs Q and $\overline{Q}$ follow Q_M and $\overline{Q}_M$ respectively. Hence, the slave section follows the master section.

The input to the gates 3 and 4 do not change during the clock pulse, therefore the race-around condition does not exist. The state of the JKMS FF changes at the negative transition (trailing edge) of the clock pulse. The Pr and Cr inputs are used to SET and CLEAR the FF irrespective of the clock input.

3.1.5 T Flip-flop

The T (Toggle) Flip-flop is formed by connecting the J and K inputs of JKMS FF together. (or) In a JKMS FF, when we make J = K, we will get a T FF. This is shown in figure.

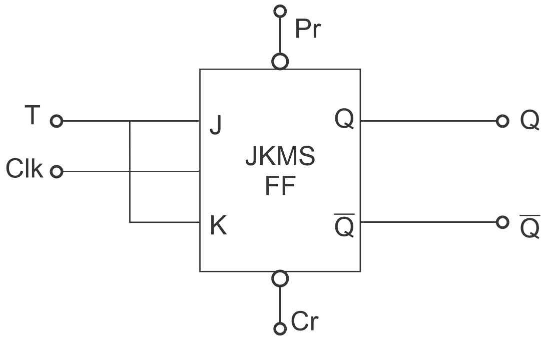


Figure : Logic diagram of T FF

The truth table of T FF is shown in figure.

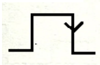
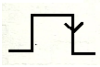
Inputs		Outputs		Condition
CLK	T	Q	$\overline{Q}$	
	0	Previous value	Previous value	No change
	1	Complement of Previous value	Complement of Previous value	Toggle

Figure : Truth table of T FF

The T FF has only one input, called T input. When T = 1 (J =1 and K = 1), the output Q toggles i.e. the complement of the previous output. When T = 0 (J = 0 and K = 0), the output will remain unchanged. Pr and Cr inputs are used to SET and CLEAR the FF irrespective of the clock signal.

T FF is also called 'divide by 2' counter because the output signal frequency is half of the clock signal frequency.

3.1.6 D Flip-flop

In a JK FF, when K is the complement of J by connecting a NOT gate between them (K = $\overline{J}$), we will get the D flip-flop. D FF has only one input D and two outputs Q and $\overline{Q}$. The logic circuit and truth table of D FF are shown the figure.

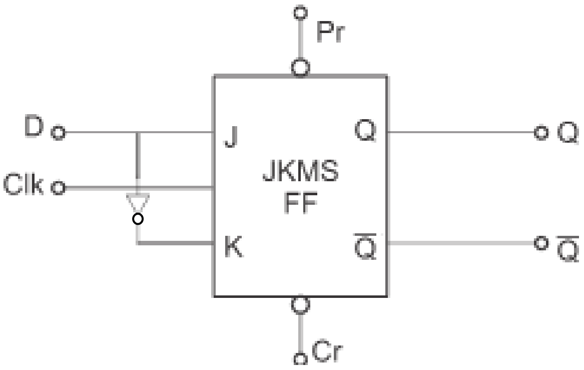


Figure : Logic diagram of D FF

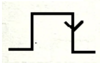
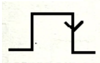
Inputs		Outputs		Condition
CLK	D	Q	$\overline{Q}$	
	0	0	1	Reset
	1	1	0	Set

Figure : Truth table of D FF

When D = 0, J will become 0 and K will become 1, and after the clock pulse is arrived, the output will be in RESET condition i.e. Q = 0 and $\overline{Q}$ = 1. When D = 1, J will become 1 and K will become 0, and after the clock pulse is arrived, the output will be in SET condition i.e. Q = 1 and $\overline{Q}$ = 0. It is clear that the FF stores the input value D.

D FF is called Data FF because this flip-flop can be used to store one bit. D FF is also called Delay FF because the input is transferred to the output only after the arrival of the clock pulse. . Pr and Cr inputs are used to SET and CLEAR the FF irrespective of the clock signal.

3.1.7 Triggering of Flip-flop

The condition of the output changes from one state to another is called triggering. The triggering is happening in FFs only due to the clock pulse. Basically there are two types of triggering the flip-flop using clock signal:

1 Level triggering
2 Edge triggering

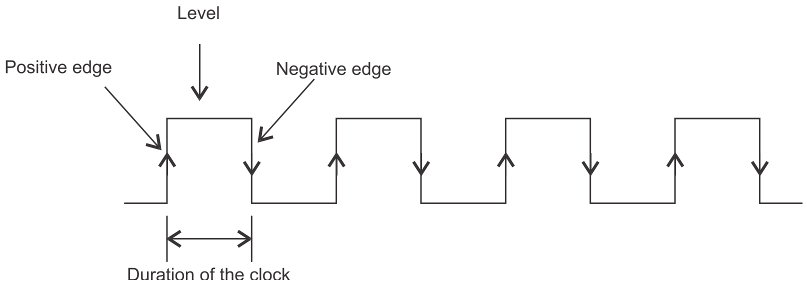


Figure : Clock pulse

Level triggering

In this triggering, the output of the FF changes during the presence of the clock signal.

When the inputs to the FF change during the presence of the clock pulse, uncertainty in the output occurs. This problem can be avoided by using Master-Slave FFs or edge triggered FFs.

Edge triggering

There are two types of edge triggering:

1. Positive edge triggering
2. Negative edge triggering

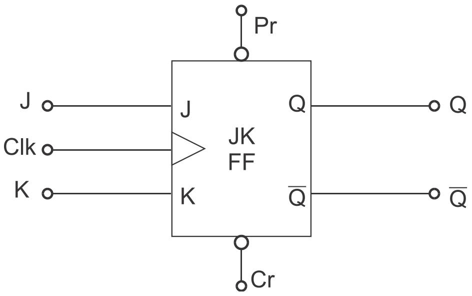


Figure : Positive edge triggered FF

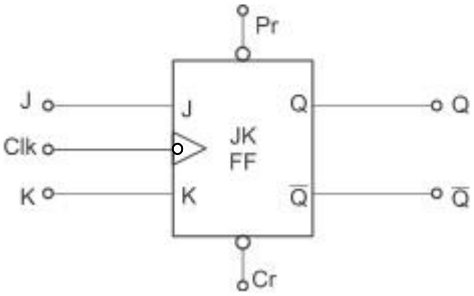


Figure : Negative edge triggered FF

In Positive edge triggering, the output of the FF changes only during the positive edge (leading edge) of the clock pulse. In negative edge triggering, the output of the FF changes only during the negative edge (trailing edge) of the clock pulse. The type of triggering of FF is represented in the clock input as shown in figure.

3.2 Counters

The most important sequential circuits used in digital systems are

1) Counters and
2) Registers

A sequential logic circuit used for counting the number of pulses is known as a counter. Counters are also used for measuring time and frequency.Flip-flops are the basic elements used for designing the counter circuits. Basically there are two types of counters. They are,

1. Asynchronous counter
2. Synchronous counter

3.2.1 Asynchronous counter

The asynchronous counter (ripple counter) is simple and straightforward in operation and construction and requires minimum hardware. These counters are slow in operation. Each flip-flop is triggered by the previous flip-flop and hence the counter has a cumulative settling time. i.e. the flip-flops are connected in serial and hence these counters are also called serial counters. In this type of counters, the triggers move through the flip-flop like a ripple in water. Hence, these counters are also known as ripple counters.

3.2.1.1 Four bit binary asynchronous (ripple)UP counter

The logic diagram of 4-bit binary asynchronous UP counter is shown in figure. The UP counter counts from 0000 to 1111.

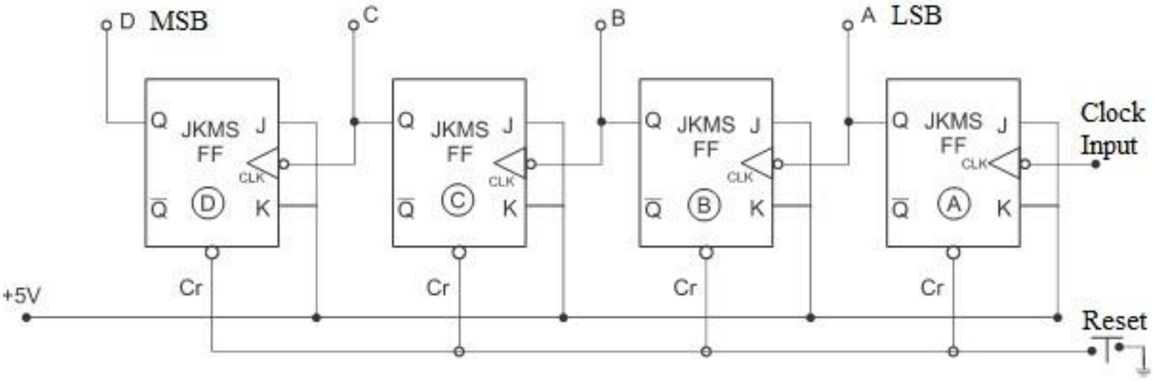


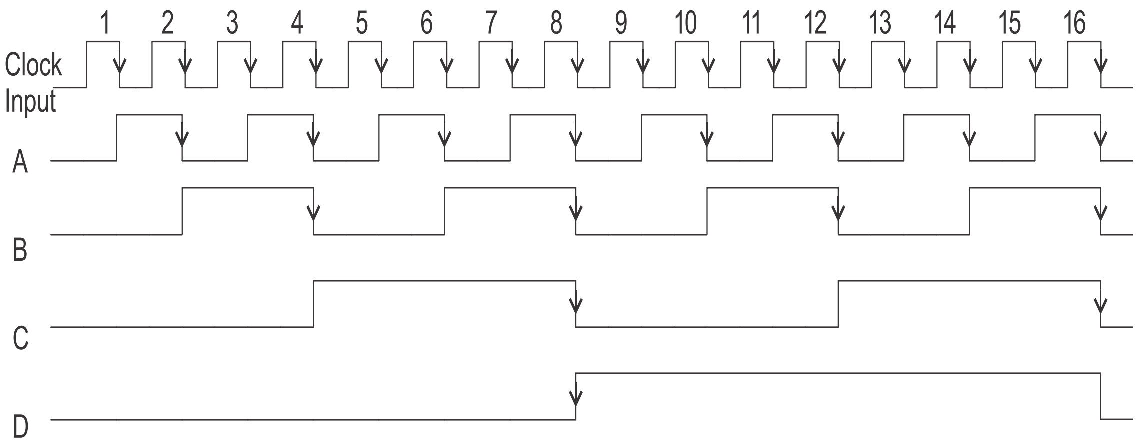
Figure :Four bit binary asynchronous (ripple) UP counter

Four negative edge triggered JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to the first flip-flop A. The Q output of the FF A is given as clock input to the second flip-flop B. The Q output of FF B is given as clock input to the third flip-flop C. The Q output of flip-flop C is given as clock input to the forth flip-flop D. The Q output of all the flip-flops are taken as the counter outputs DCBA. The output A is called the

Least Significant Bit (LSB) and the output D is called the Most Significant Bit (MSB). The CLEAR (Cr) input of all the FFs are connected to ground through the Master Reset switch.

When the Master Reset switch is pressed, all the FFs are cleared and the counter output DCBA is 0000. During the negative edge of the first clock pulse, FF A will be toggled i.e. the output A changes from 0 to 1. At this time, the outputs of all other flip-flops will not change.Hence, the counter output DCBA is 0001.

Input	Output			
Clock	D	C	B	A
Reset	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	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0



During the application of the second clock pulse, FF A will be toggled once again from 1 to 0. This will give a negative edge triggering pulse to FF B and hence FF B also toggles from 0 to 1. The counter output DCBA will become 0010.

Similarly, FF C will toggle when the output of FF B toggles from 1 to 0 and FF D will toggle when the output of FF C toggles from 1 to 0. It should be noted that FF A toggles for every clock pulse, FF B toggles for every two clock pulses, FF C toggles for every 4 clock pulses and FF D toggles for every eight clock pulses. The frequency of output A is ½ of the clock frequency, output B is ¼ of clock, output C is 1/8 of clock and output D is 1/16 of clock frequency. Hence, the four bit counter acts as a 'divided by 16' counter.

For the 15th clock pulse, the output is 1111. When the next (16th) clock pulse is applied, all the flip-flops will toggle from 1 to 0 at the same time and hence the output is 0000.The outputs of the counter during the application of each clock pulse are shown in the truth table and also in the waveforms.

3.2.1.2 Four bit binary asynchronous (ripple) DOWN counter

The logic diagram of 4-bit binary asynchronous DOWN counter is shown in figure. The DOWN counter counts from 1111 to 0000.

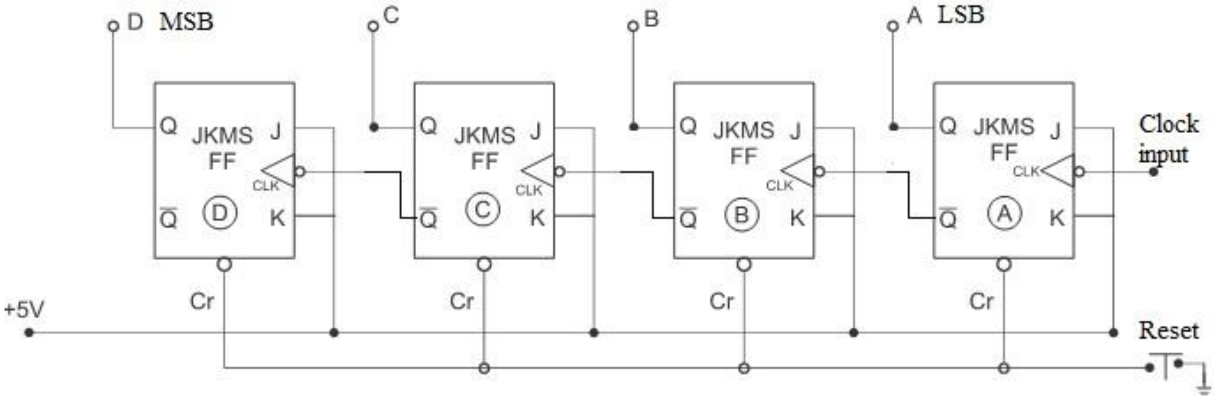
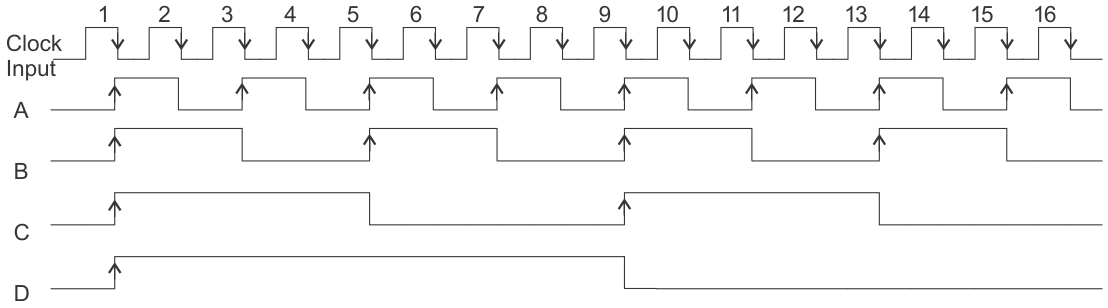


Figure :Four bit binary asynchronous (ripple) DOWN counter

Four negative edge triggered JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to the first flip-flop A. The $\overline{Q}$ output of the FF A is given as clock input to the second flip-flop B. The $\overline{Q}$ output of FF B is given as clock input to the third flip-flop C. The $\overline{Q}$ output of flip-flop C is given as clock input to the fourth flip-flop D. The Q output of all the flip-flops are taken as the counter outputs DCBA. The output A is called the Least Significant Bit LSB) and the output D is called the Most Significant Bit (MSB). The CLEAR (Cr) input of all the FFs are connected to ground through the Master Reset switch.

When the Master Reset switch is pressed, all the FFs are cleared and the counter output DCBA is 0000. During the negative edge of the first clock pulse, FF A will be toggled i.e. the Q output of A changes from 0 to 1 and $\overline{Q}$ output of A changes from 1 to 0. Hence, flip-flop B also toggles. Similarly flip-flops C and D also toggle. Hence, the counter output DCBA is 1111.

Input	Output			
Clock	D	C	B	A
Reset	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0



During the application of the second clock pulse, the Q output of FF A will toggled once again from 1 to 0 and $\overline{Q}$ from 0 to 1. Hence, FF B will not toggle this time. The counter output DCBA will become 1110.

During the third clock pulse, the Q output of FF A will toggle once again from 0 to 1 and $\overline{Q}$ from 1 to 0. Hence, FF B will toggle this time. The counter output DCBA will become 1101. Similarly, FF C will toggle when the $\overline{Q}$ output of FF B toggles from 1 to 0 and FF D will toggle when the $\overline{Q}$ output of FF C toggles from 1 to 0. It should be noted that FF A toggles for every clock pulse, FF B toggles for every two clock pulses, FF C toggles for every 4 clock pulses and FF D toggles for every eight clock pulses. The frequency of output A is ½ of the clock frequency, output B is ¼ of clock, output C is 1/8 of clock and output D is 1/16 of clock frequency. Hence, the four bit counter acts as a 'divided by 16' counter.The outputs of the counter during the application of each clock pulse are shown in the truth tableand also in the waveforms.

3.2.1.3 Four bit binary asynchronous (ripple) UP / DOWN counter

The logic circuit diagram of 4-bit binary asynchronous UP / DOWN counter is shown in figure.

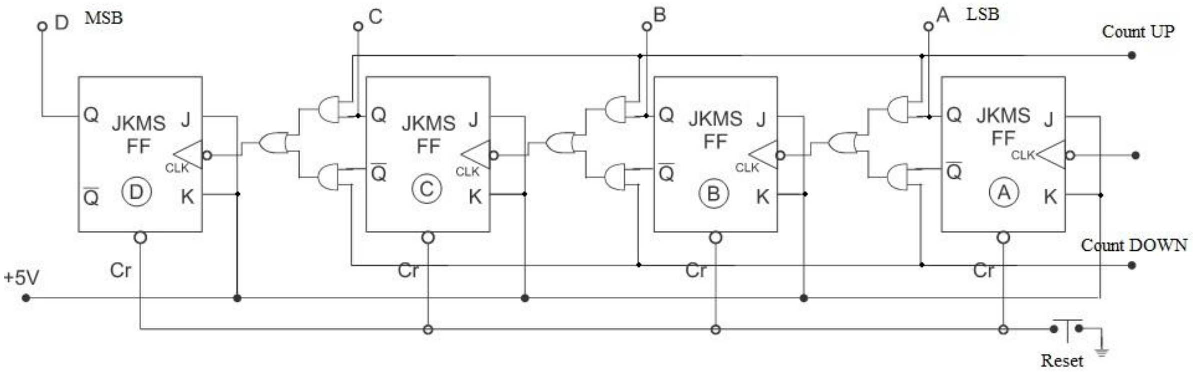


Figure :Four bit binary asynchronous (ripple) UP / DOWN counter

We know that in UP counter, the Q output of each flip-flop is given as clock input for the next FF. For the DOWN counter, the $\overline{Q}$ output each flip-flop is given as clock input for the next FF. A combinational circuit using AND and OR gates is used to select the Q or $\overline{Q}$ output. When the COUNT UP line is made 1, the upper AND gate is enabled and Q will go as clock input to the next FF. When the COUNT DOWN line is made 1, the lower AND gate is enabled and $\overline{Q}$ will go as clock input to the next flip-flops.

3.2.2 Decade counter

The logic diagram of decade counter is shown in figure. It counts from 0 to 9 and at the 10th clock pulse the counter will reset and starts counting again. The other name for decade counter is Mod-10 counter.

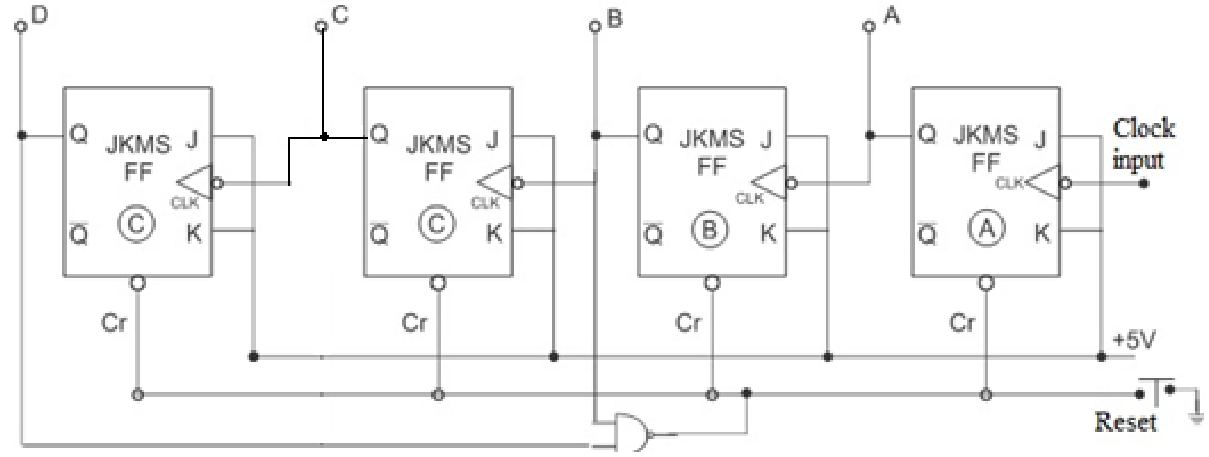
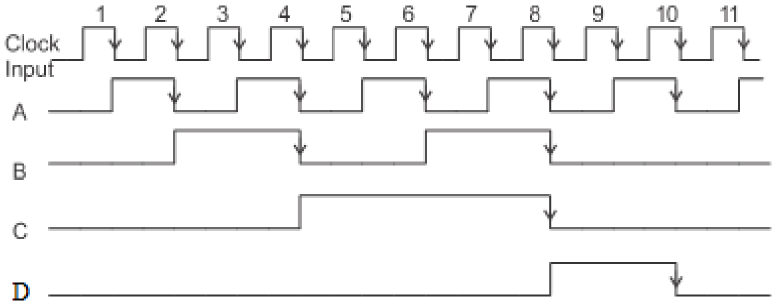


Figure : Decade counter

The counter has to count from 0000 to 1001. Hence, four negative edge triggered JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to the first flip-flop A. The Q output of the FF A is given as clock input to the second flip-flop B. The Q output of FF B is given as clock input to the third flip-flop C. The Q output of FF C is given as clock input to the fourth flip-flop D. The Q output of all the flip-flops are taken as the counter outputs DCBA. The output A is called the Least Significant Bit LSB) and the output D is called the Most Significant Bit (MSB). The CLEAR (Cr) input of all the FFs are connected to ground through the Master Reset switch.

We need to reset the counter at 10th clock pulse i.e. at DCBA = 1010. Hence, we have to reset the counter when D = 1 and B = 1. The NAND gate is used to apply RESET signal. The inputs for the NAND gate are taken from D and B.

Input	Output			
Clock	D	C	B	A
Reset	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	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	0	0	0	0
11	0	0	0	1



When the Master Reset switch is pressed, all the FFs are cleared and the counter output DCBA is 0000. During the negative edge of the first clock pulse, FF A will be toggled i.e. the output A changes from 0 to 1. At this time, the outputs of all other flip-flops will not change. Hence, the counter output DCBA is 0001. During the application of the second clock pulse, FF A will be toggled once again from 1 to 0. This will give a negative edge triggering pulse to FF B and hence FF B also toggles from 0 to 1. The counter output DCBA will become 0010. Similarly the counting continues.

At the 10th clock pulse, the output DCBA will try to become 1010 (D = 1 and B = 1). As the NAND gate inputs, D and B are 11, the 0 in the gate output RESET the counter. The frequency of output D is 1/10 of clock signal frequency. Hence, the mod-10 counter acts as a 'divided by 10' counter. The outputs of the counter during the application of each clock pulse are shown in the truth table and also in the waveforms.

3.2.3 Modulo–N counter

We know that the 4-bit binary asynchronous as well as synchronous counters resets to 0000 and starts counting again in every 16th clock pulse automatically. This is called 'divide by 16' counter or 'mod-16' counter. i.e. a mod-N counter resets in every Nth clock pulse. We can design any mod-N counter using an additional NAND gate in the counter circuit to reset the counter in every Nthclock pulse. In simply says, a MOD-N counter resets at Nth clock pulse.

3.2.3.1 Mod-3 counter

The logic diagram of Mod-3 counter is shown in figure. It counts from 0 to 2 and at the 3rdclock pulse the counter will reset and starts counting again.

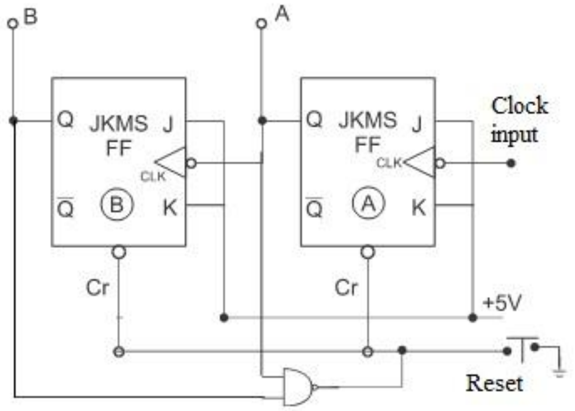
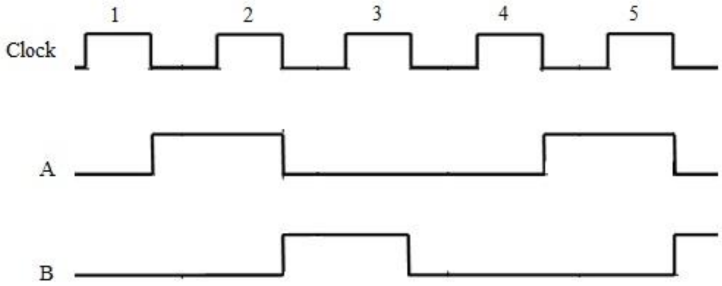


Figure : Mod-3 counter

The counter has to count from 00 to 10. Hence, two negative edge triggered JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to the first flip-flop A. The Q output of the FF A is given as clock input to the second flip-flop B. The Q output of all the flip-flops are taken as the counter outputs BA. The output A is called the Least Significant Bit (LSB) and the output B is called the Most Significant Bit (MSB). The CLEAR (Cr) input of all the FFs are connected to ground through the Master Reset switch.

We need to reset the counter at 3rdclock pulse i.e. at BA = 11. Hence, we have to reset the counter when B = 1 and A = 1. The NAND gate is used to apply RESET signal. The inputs for the NAND gate are taken from B and A.

Input	Output	
Clock	B	A
Reset	0	0
1	0	1
2	1	0
3	0	0
4	0	1



When the Master Reset switch is pressed, all the FFs are cleared and the counter output BA is 00. During the negative edge of the first clock pulse, FF A will be toggled i.e. the output A changes from 0 to 1. At this time, the outputs of all other flip-flops will not change. Hence, the counter output BA is 01. During the application of the second clock pulse, FF A will be toggled once again from 1 to 0. This will give a negative edge triggering pulse to FF B and hence FF B also toggles from 0 to 1. The counter output BA will become 10.

At the 3rdclock pulse, the output BA will try to become 11 (B = 1 and A = 1). As the NAND gate inputs, B and A are 11, the 0 in the gate output RESET the counter. The frequency of output B is 1/3 of clock signal frequency. Hence, the mod-3 counter acts as a 'divided by 3' counter. The outputs of the counter during the application of each clock pulse are shown in the truth table and also in the waveforms.

3.2.3.2 Mod-7 counter

The logic diagram of Mod-7 counter is shown in figure. It counts from 0 to 6 and at the 7thclock pulse the counter will reset and starts counting again.

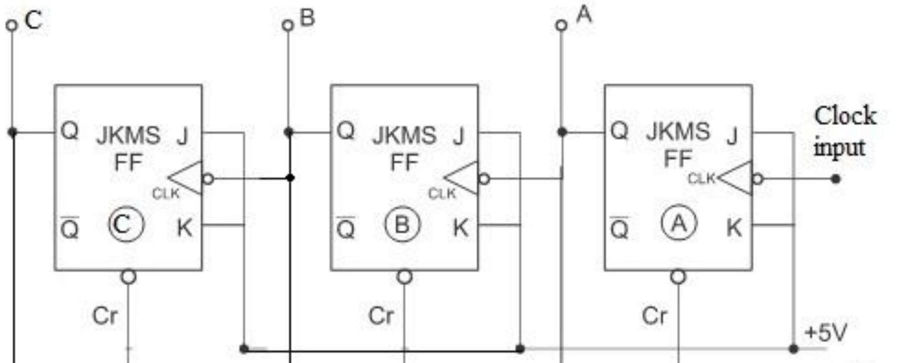


Figure : Mod-7 counter

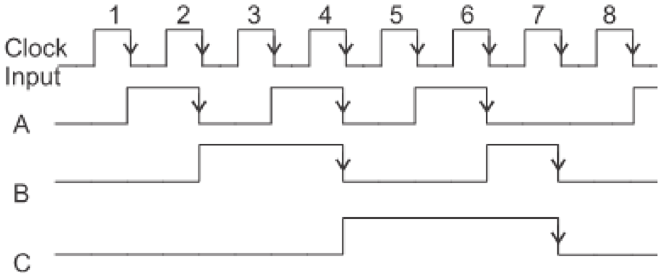
The counter has to count from 000 to 110. Hence, three negative edge triggered JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to the first flip-flop A. The Q output of the FF A is given as clock input to the second flip-flop B. The Q output of FF B is given as clock input to the third flip-flop C. The Q output of all the flip-flops are taken as the counter outputs CBA. The output A is called the Least Significant Bit

LSB) and the output C is called the Most Significant Bit (MSB). The CLEAR (Cr) input of all the FFs are connected to ground through the Master Reset switch.

We need to reset the counter at 7thclock pulse i.e. at CBA = 111. Hence, we have to reset the counter when C = 1, B = 1 and A = 1. The NAND gate is used to apply RESET signal. The inputs for the NAND gate are taken from C, B and A.

Input	Output		
Clock	C	B	A
Reset	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	0	0	0
8	0	0	1

When the Master Reset switch is pressed, all the FFs are cleared and the counter output CBA is 000. During the negative edge of the first clock pulse, FF A will be toggled i.e. the output A changes from 0 to 1. At this time, the outputs of all other flip-flops will not change. Hence, the counter output CBA is 001. During the application of the second clock pulse, FF A will be toggled once again from 1 to 0. This will give a negative edge triggering pulse to FF B and hence FF B also toggles from 0 to 1. The counter output CBA will become 010. Similarly the counting continues.



At the 7thclock pulse, the output CBA will try to become 111 (C = 1, B = 1 and A = 1). As the NAND gate inputs, C, B and A are 111, the 0 in the gate output RESET the counter. The frequency of output C is 1/7 of clock signal frequency. Hence, the mod-7 counter acts as a ‘divided by 7’ counter. The outputs of the counter during the application of each clock pulse are shown in the truth table and also in the waveforms in figure.

3.2.4 Synchronous counter

The speed of operation can be improved by using parallel or synchronous counter. Here, every flip-flop is triggered by the clock pulse directly (in synchronism), and thus the settling time is equal to the delay time of single FF. These counters require more hardware.

3.2.4.1 Four bit binary synchronous UP counter

The logic diagram of 4-bit binary synchronous UP counter is shown in figure. The UP counter counts from 0000 to 1111. The synchronous counters are fast in operation but require more hardware.

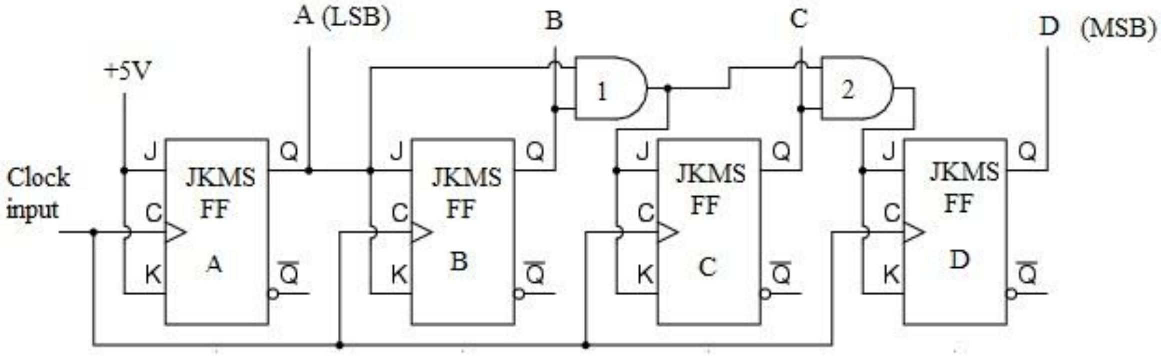
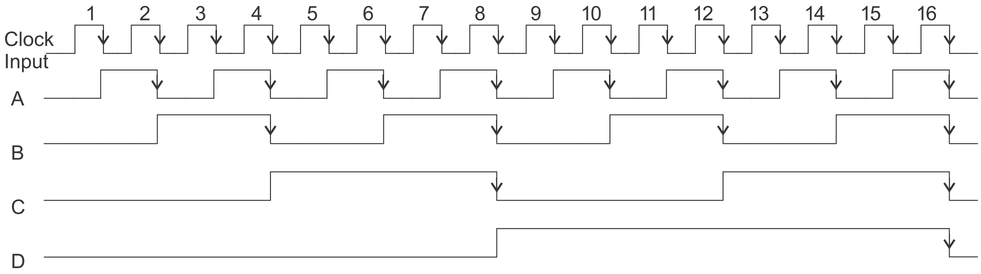


Figure : Four bit binary synchronous UP counter

Four JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to all the flip-flops. The gate circuits are arranged in such a way that FF A toggles for every clock pulse, FF B toggles for every two clock pulses, FF C toggles for every four clock pulses and FF D toggles for every eight clock pulses. The Q output of all the flip-flops are taken as the counter outputs ABCD.The output A is called the Least Significant Bit (LSB) and the output D is called the Most Significant Bit (MSB). The truth table of the counter is shown in figure.

Input	Output			
Clock	D	C	B	A
Reset	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	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1
16	0	0	0	0

From the truth table we may observe that the output A toggles in every clock pulse, output B toggles when output A is 1, output C toggles when both A and B are 1, output C toggles when A, B and C are 1. This can be achieved by using AND gates as shown in the figure. Toggle input of FF A is directly connected to +5v, hence, FF A toggles for every clock pulse. The Q output of the FF A is connected to the Toggle input of FF B, hence, FF B toggles only when A is 0. The output of AND gate 1, whose inputs are A and B, is connected to the Toggle input of FF C, hence, FF C toggles only when Aand B are 1. Similarly, the output of AND gate 2, whose inputs are the output of AND gate 1 and C, is connected to the Toggle input of FF D, hence, FF D toggles only when A, B and C are 1.



The frequency of output A is ½ of the clock frequency, output B is ¼ of clock, output C is 1/8 of clock and output D is 1/16 of clock frequency. Hence, the four bit counter acts as a ‘divided by 16’ counter.For the 15th clock pulse, the output is 1111. When the next (16th) clock pulse is applied, all the flip-flops will toggle from 1 to 0 at the same time and hence the output is 0000. The outputs of the counter during the application of each clock pulse are shown in the waveforms.

3.2.4.2 Four bit binary synchronous DOWN counter

The logic diagram of 4-bit binary synchronous UP counter is shown in figure. The DOWN counter counts from 1111 to 0000. The synchronous counters are fast in operation but require more hardware.

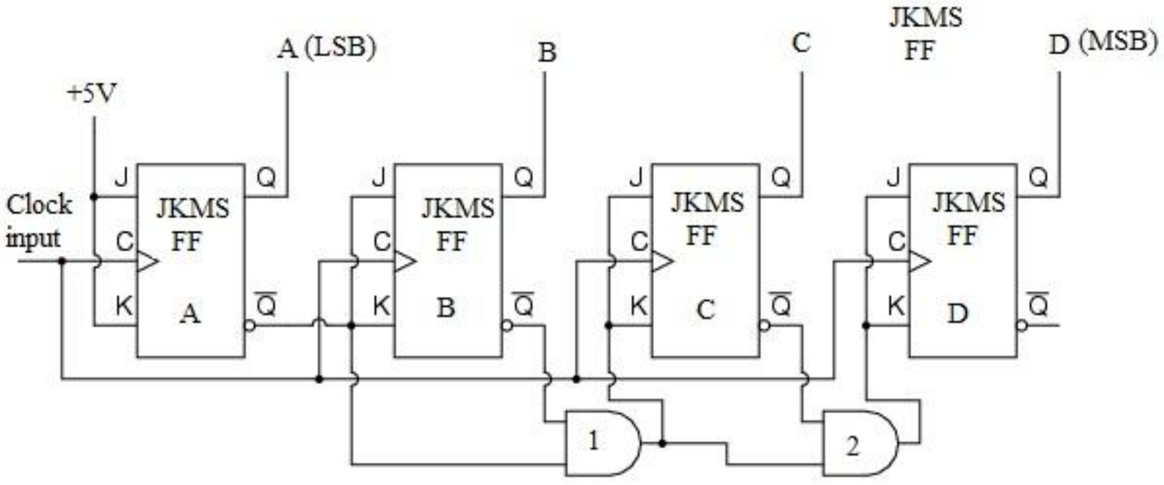
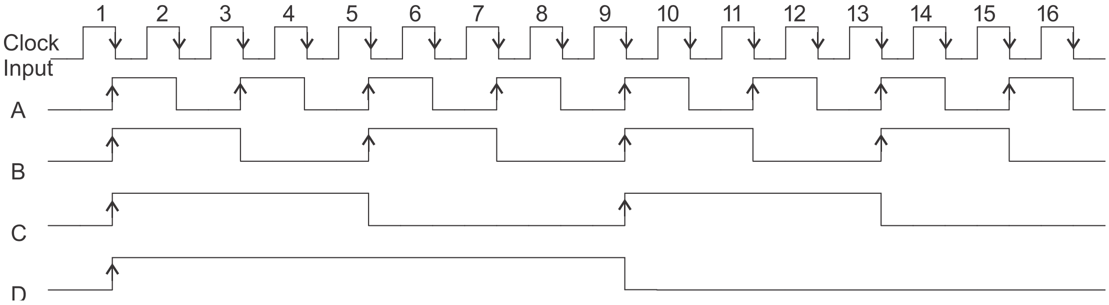


Figure : Four bit binary synchronous DOWN counter

Four JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to all the flip-flops. The gate circuits are arranged in such a way that FF A toggles for every clock pulse, FF B toggles for every two clock pulses, FF C toggles for every four clock pulses and FF D toggles for every eight clock pulses. The Q output of all the flip-flops are taken as the counter outputs ABCD. The output A is called the Least Significant Bit (LSB) and the output D is called the Most Significant Bit (MSB). The truth table of the counter is shown in figure.

Input	Output			
Clock	D	C	B	A
Reset	0	0	0	0
1	1	1	1	1
2	1	1	1	0
3	1	1	0	1
4	1	1	0	0
5	1	0	1	1
6	1	0	1	0
7	1	0	0	1
8	1	0	0	0
9	0	1	1	1
10	0	1	1	0
11	0	1	0	1
12	0	1	0	0
13	0	0	1	1
14	0	0	1	0
15	0	0	0	1
16	0	0	0	0

From the truth table we may observe that the output A toggles in every clock pulse, output B toggles when output A is 0, output C toggles when both A and B are 0, output C toggles when A, B and C are 0. This can be achieved by using AND gates as shown in the figure. Toggle input of FF A is directly connected to +5v, hence, FF A toggles for every clock pulse. The $\overline{Q}_A$ output of the FF A is connected to the Toggle input of FF B, hence, FF B toggles only when A is 0. The output of AND gate 1, whose inputs are $\overline{Q}_A$ and$\overline{Q}_B$, is connected to the Toggle input of FF C, hence, FF C toggles only when A and B are 0. Similarly, the output of AND gate 2, whose inputs are the output of AND gate 1 and $\overline{Q}_C$, is connected to the Toggle input of FF D, hence, FF D toggles only when A, B and C are 0.



The frequency of output A is ½ of the clock frequency, output B is ¼ of clock, output C is 1/8 of clock and output D is 1/16 of clock frequency. Hence, the four bit counter acts as a ‘divided by 16’ counter.For the 15th clock pulse, the output is 0000. When the next (16th) clock pulse is applied, all the flip-flops will toggle from 0 to 1 at the same time and hence the output is 1111. The outputs of the counter during the application of each clock pulse are shown in the waveforms.

3.2.4.3 Four bit binary synchronous UP/DOWN counter

The logic diagram of 4-bit binary synchronous UP/DOWN counter is shown in figure. The UP counter counts from 0000 to 1111. The DOWN counter counts from 1111 to 0000. The synchronous counters are fast in operation but require more hardware.

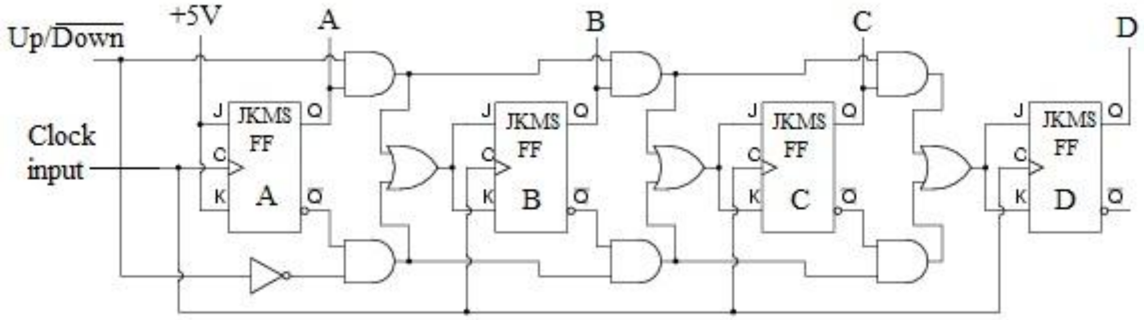


Figure : Four bit binary synchronous UP/DOWN counter

Four JKMS flip-flops are used in this counter. J and K inputs of all the FFs are connected to +5v (J =1, K = 1). This makes the FFs to operate as T (Toggle) flip-flop. The T FF changes its state (i.e. from 0 to 1 or 1 to 0) for every input clock pulse. The clock input is applied to all the flip-flops. The gate circuits are arranged in such a way that FF A toggles for every clock pulse, FF B toggles for every two clock pulses, FF C toggles for every four clock pulses and FF D toggles for every eight clock pulses.

UP function :

For UP function, the Q output of previous FF must be connected to the Toggle input of next FF. This is achieved by the AND gates 1, 2 and 3 and the OR gates. The UP function is enabled by connecting Up/$\overline{Down}$ input to +5V (logic 1). Now, the counter functions as UP counter.

DOWN function :

For DOWN function, the $\overline{Q}$ output of the previous FF must be connected to the Toggle input of next FF using the AND gates 4, 5 and 6 and the OR gates. The DOWN function is selected by connecting the Up/$\overline{Down}$ input to GND (logic 0). Now, the counter functions as DOWN counter.

3.2.5 Johnson counter

The logic circuit of Johnson counter is shown in figure. It is similar to ring counter except one change. Instead of the Q output, $\overline{Q}$ output of the last flip-flop is given as the input for the first flip-flop. Therefore, the Johnson counter is also called Twisted-ring counter. In Johnson counter, there is no need for the START button.

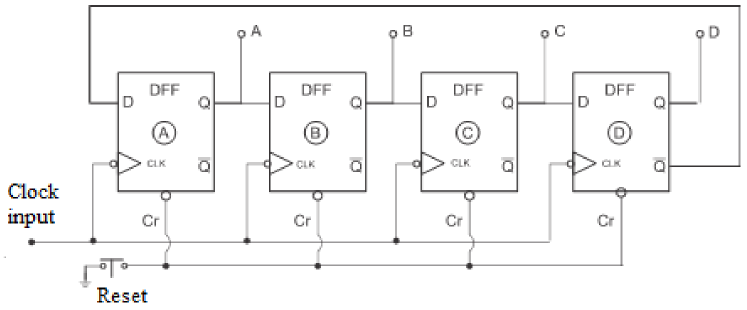
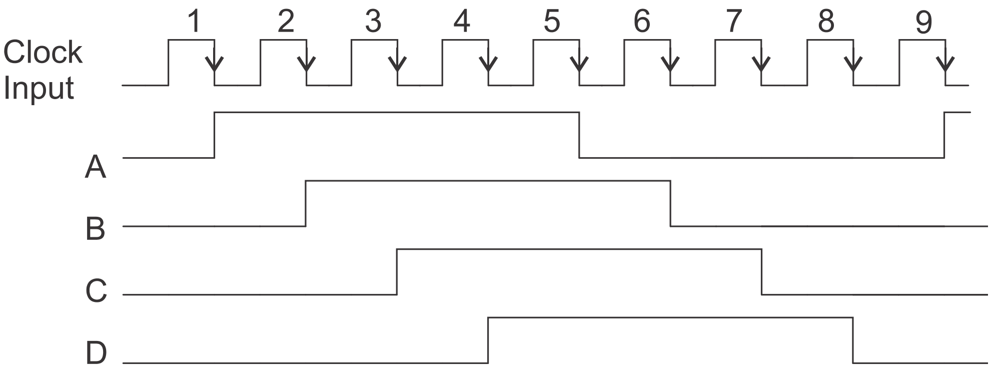


Figure : Johnson counter

The flip-flops are connected in synchronous mode. i.e. clock pulse is applied to all the flip-flops in parallel. The output of the first flip-flop A is connected as the input to the second flip-flop. The output of the second flip-flop B is connected as the input to the third flip-flop. The output of the third flip-flop C is connected as the input to the fourth flip-flop. The $\overline{Q}$output of the fourth flip-flop D is connected as the input to the first flip-flop. The CLEAR (Cr) input of all the FFs are connected to ground through the Master Reset switch.

Initially, the master reset switch is pressed to clear all the FFs so that ABCD is 0000. When the first clock pulse is applied, $\overline{Q}$ of the fourth FF D (1) is moved to FF A, Q of FF A (0) to FF B, Q of FF B (0) to FF C and Q of FF C (0) to FF D and the counter output is 1000. When the second clock pulse is applied, $\overline{Q}$ of the fourth FF D (1) is moved to FF A, Q of FF A (1) to FF B, Q of FF B (0) to FF C and Q of FF C (0) to FF D and the counter output is 1100. Similarly the circuit operation continues. The outputs of the counter during the application of each clock pulse are shown in the truth table and also in the waveforms.

Input	Output			
Clock	A	B	C	D
Reset	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0
9	1	0	0	0



3.2.6 Ring counter

The logic diagram of 4-bit ring counter is shown in figure. We use D FF in the ring counter circuit. This counter is called ring counter because a ‘1’ is passed to the next flip-flop is a circular ring format.

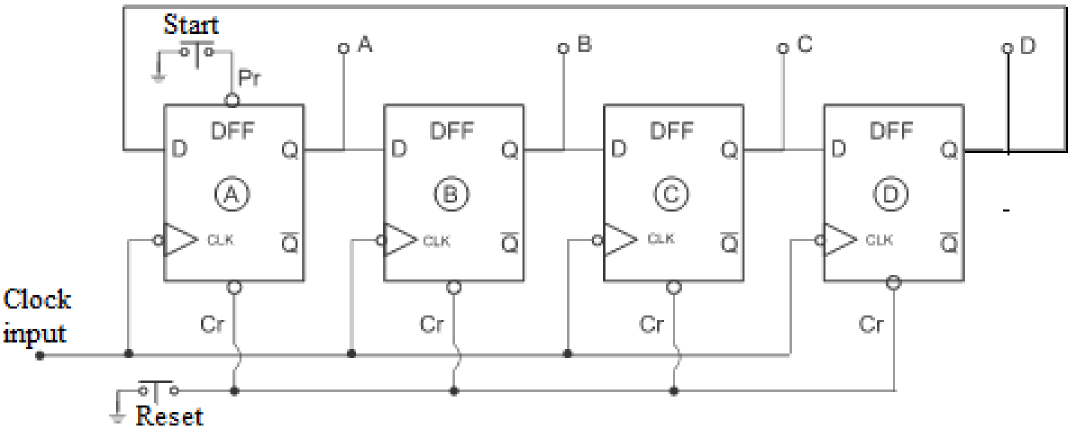
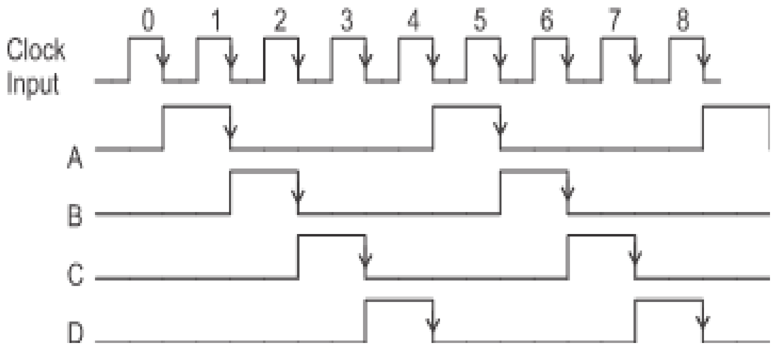


Figure : Ring counter

The flip-flops are connected in synchronous mode. i.e. clock pulse is applied to all the flip-flops in parallel. The output of the first flip-flop A is connected as the input to the second flip-flop. The output of the second flip-flop B is connected as the input to the third flip-flop. The output of the third flip-flop C is connected as the input to the fourth flip-flop. The output of the fourth flip-flop D is connected as the input to the first flip-flop. The CLEAR (Cr) input of all the FFs are connected to ground through the Master Reset switch.

Initially, the master reset switch is pressed to clear all the FFs. When the start button (PRESET) is pressed a ‘1’ will be stored in the first flip-flop A and the output ABCD is 1000. When the first clock pulse is applied, A is moved to B, B to C, C to D and D to A and the counter output is 0100. Similarly the circuit operation continues. The outputs of the counter during the application of each clock pulse are shown in the truth table and also in the waveforms.

Input	Output			
Clock	A	B	C	D
Reset	0	0	0	0
Start	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0
5	0	1	0	0
6	0	0	1	0
7	0	0	0	1
8	1	0	0	0



3.3 Registers

A flip-flop can store only one binary digit. Register is a combination of N flip-flops to store N bits. For example, an 8-bit register can be used to store a data of 8-bit information.

3.3.1 4-bit Shift register

Shit register is a type of register in which data is shifted from one flip-flop to another flip-flop for storing and retrieving the information. Depending upon how we enter the data into the shift register (input) and how we take the data from the shift register (output), shift registers are classified into four types. They are,

1. Serial IN - Serial OUT (SISO) Shift register
2. Serial IN - Parallel OUT (SIPO) Shift register
3. Parallel IN - Serial OUT (PISO) Shift register
4. Parallel IN - Parallel OUT (PIPO) Shift register

The logic diagram of shift register with all the four functions SISO, SIPO, PISO and PIPO is shown in figure.

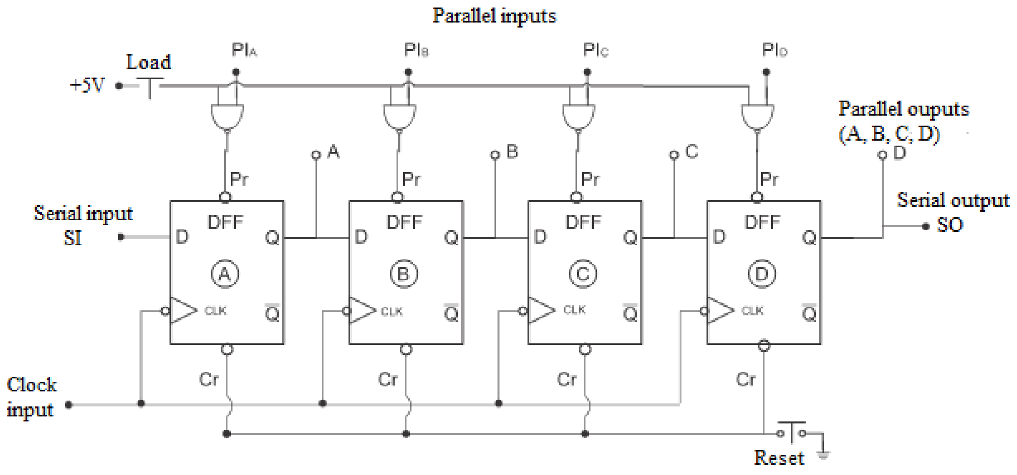
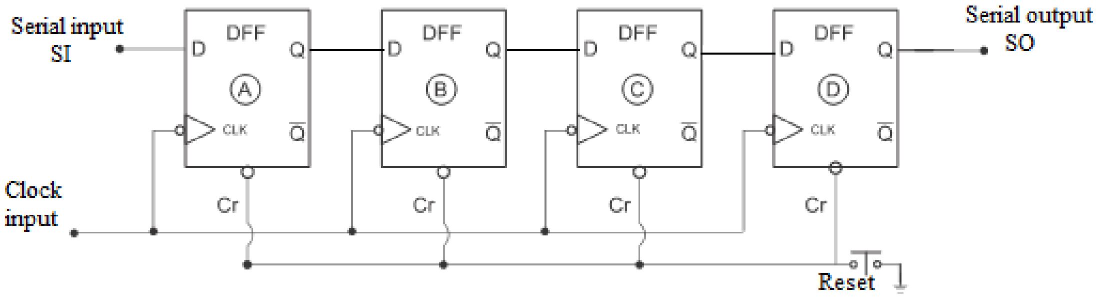


Figure : Shift register

The operation of SISO, SIPO, PISO and PIPO are explained in the following sections.

3.3.2 Serial IN – Serial OUT (SISO) Shift register

In SISO shift register the input is given in serial form (i.e. one bit by one bit) and the output is taken in serial form. The logic diagram of SISO shift register is shown in figure. Four D FFs A, B, C and D are used is synchronous mode. Clock pulse is applied to all the FFs simultaneously. The Serial Input Data (SI) is given to the D input of the first FF. The Q output of FF A is given to the D input of FF B. The Q output of FF B is given to the D input of FF C. The Q output of FF C is given to the D input of FF D. The Serial Output data (SO) is taken from the Q output of FF D. The CLEAR inputs of all the FFs are connected to ground through the Master Reset switch.

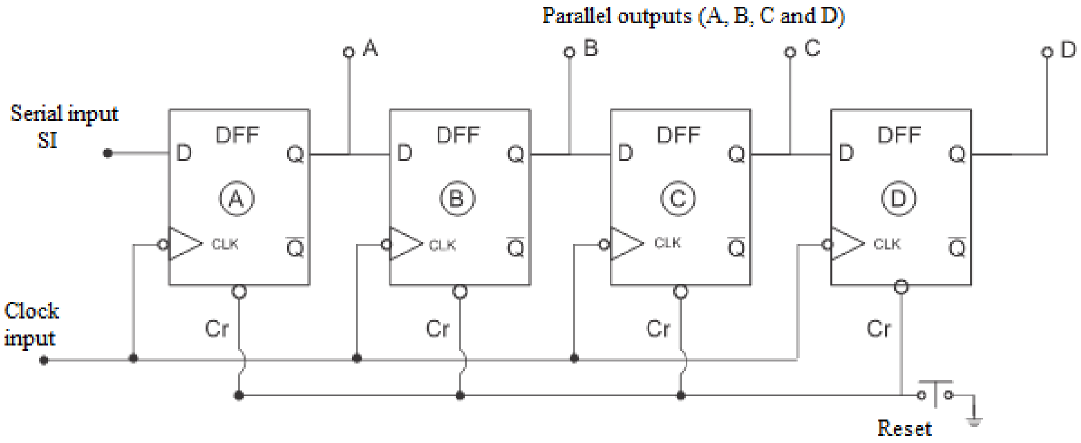


The sequence for input and output operations is given in table.

Input (Write) operation		Output (Read) operation	
S.No	Sequence (Serial Input)	S.No	Sequence (Serial Output)
1.	Press the Master RESET switch to clear all the FFs.	1.	The first bit is available in the Serial Output (SO) terminal.
2.	Give the serial data to the Serial Input terminal SI, one by one (4 bits).	2.	Give first clock pulse to get the second bit from SO terminal.
3.	For each input, one clock pulse must be applied (4 clock pulses).	3.	Give second clock pulse to get the third bit from SO terminal.
4.	The data (4 bits) are stored in the four FFs.	4.	Give third clock pulse to get the fourth bit from SO terminal.

3.3.3 Serial IN – Parallel OUT (SIPO) Shift register

In SIPO shift register the input is given in serial form (i.e. one bit by one bit) and the output is taken in parallel form (i.e. all the bits at the same time). The logic diagram of SIPO shift register is shown in figure. Four D FFs A, B, C and D are used is synchronous mode. Clock pulse is applied to all the FFs simultaneously. The Serial Input Data (SI) is given to the D input of the first FF. The Q output of FF A is given to the D input of FF B. The Q output of FF B is given to the D input of FF C. The Q output of FF C is given to the D input of FF D. The Parallel Output data (ABCD) are taken from the Q outputs of each FF. The CLEAR inputs of all the FFs are connected to ground through the Master Reset switch.

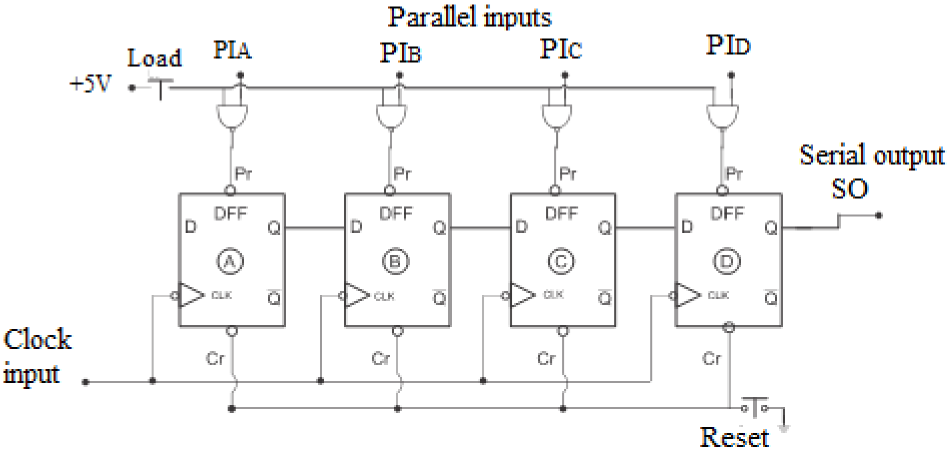


The sequence for input and output operations is given in table.

Input (Write) operation		Output (Read) operation	
S.No	Sequence (Sequence Input)	S.No	Sequence (Parallel Output)
1.	Press the Master RESET switch to clear all the FFs.	1.	The 4-bits parallel data are taken out from ABCD terminals.
2.	Give the serial data to the Serial Input terminal SI, one by one (4 bits).		
3.	For each input, one clock pulse must be applied (4 clock pulses).		
4.	The data (4 bits) are stored in the four FFs.		

3.3.4 Parallel IN – Serial OUT (PISO) Shift register

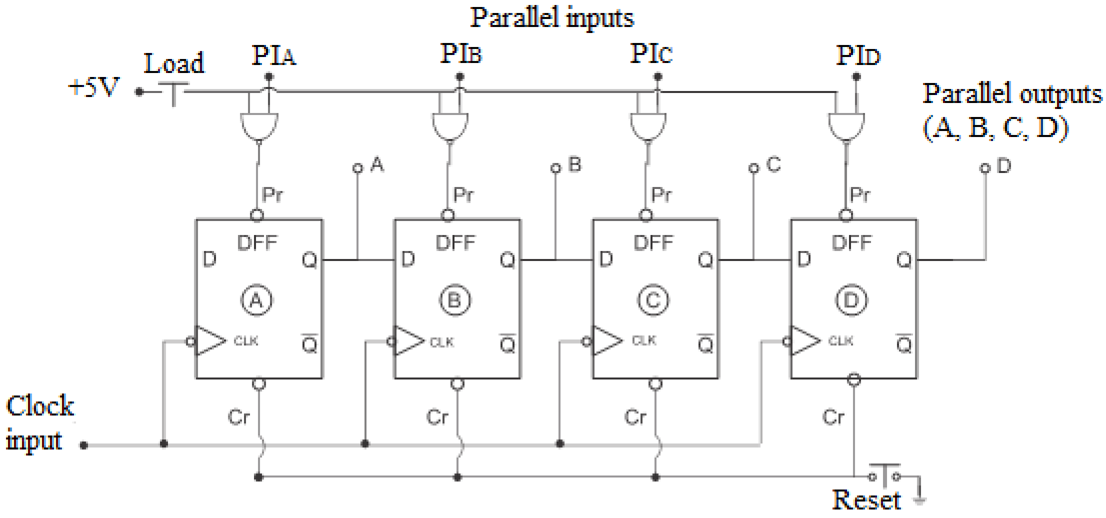
In PISO shift register the input is given in parallel form (i.e. all the bits at the same time) and the output is taken in serial form (i.e. one bit by one bit). The logic diagram of SIPO shift register is shown in figure. Four D FFs A, B, C and D are used is synchronous mode. Clock pulse is applied to all the FFs simultaneously. The Parallel Input data PI_A, PI_B, PI_C and PI_D are given to the PRESET terminals of the FFs through NAND gates. The NAND gates are enabled by the LOAD switch. The Q output of FF A is given to the D input of FF B. The Q output of FF B is given to the D input of FF C. The Q output of FF C is given to the D input of FF D. The Serial Output data (SO) is taken from the Q output of FF D. The CLEAR inputs of all the FFs are connected to ground through the Master Reset switch.



Input (Write) operation		Output (Read) operation	
S.No	Sequence (Parallel Input)	S.No	Sequence (Serial Output)
1.	Press the Master RESET switch to clear all the FFs.	1.	The 4-bits parallel data are taken out from ABCD terminals.
2.	Give the 4-bit input data to the Parallel Input terminals PI_A, PI_B, PI_C and PI_D.	2.	Give first clock pulse to get the second bit from SO terminal.
3.	Press the LOAD switch.	3.	Give second clock pulse to get the third bit from SO terminal.
4.	The data (4 bits) are stored in the four FFs.	4.	Give third clock pulse to get the fourth bit from SO terminal.

3.3.5 Parallel IN – Parallel OUT (PIPO) Shift register

In PIPO shift register the input is given in parallel form (i.e. all the bits at the same time) and the output is taken in parallel form. The logic diagram of PIPO shift register is shown in figure. Four D FFs A, B, C and D are used is synchronous mode. Clock pulse is applied to all the FFs simultaneously. The Parallel Input data PI_A, PI_B, PI_C and PI_D are given to the PRESET terminals of the FFs through NAND gates. The NAND gates are enabled by the LOAD switch. The Q output of FF A is given to the D input of FF B. The Q output of FF B is given to the D input of FF C. The Q output of FF C is given to the D input of FF D. The Parallel Output data (ABCD) are taken from the Q outputs of each FF. The CLEAR inputs of all the FFs are connected to ground through the Master Reset switch.



Input (Write) operation		Output (Read) operation	
S.No	Sequence (Parallel Input)	S.No	Sequence (Parallel output)
1.	Press the Master RESET switch to clear all the FFs.	1.	The 4-bits parallel data are taken out from ABCD terminals.
2.	Give the 4-bit input data to the Parallel Input terminals PI_A, PI_B, PI_C and $PI_{D.}$		
3.	Press the LOAD switch.		
4.	The data (4 bits) are stored in the four FFs.		

☺☺☺☺☺

UNIT - IV

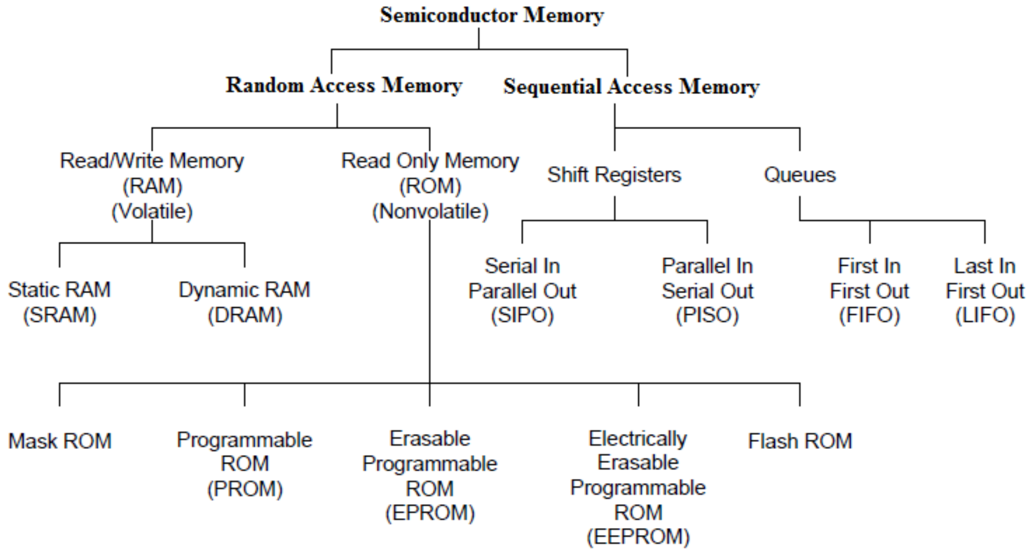
MEMORY DEVICES

Memory devices

Modern digital systems require the capability of storing and retrieving large amounts of information at high speeds. Semiconductor memories are digital circuits that store digital information (binary data) in large quantity.

4.1 Classification of Semiconductor memories

The various classifications of semiconductor memories are shown in figure.



Semiconductor Memory
Random Access Memory
Sequential Access Memory
Read/Write Memory (RAM) (Volatile)
Read Only Memory (ROM) (Nonvolatile)
Shift Registers
Queues
Static RAM (SRAM)
Dynamic RAM (DRAM)
Serial In Parallel Out (SIPO)
Parallel In Serial Out (PISO)
First In First Out (FIFO)
Last In First Out (LIFO)
Mask ROM
Programmable ROM (PROM)
Erasable Programmable ROM (EPROM)
Electrically Erasable Programmable ROM (EEPROM)
Flash ROM

Figure : Classification of Semiconductor memory

Basically, Semiconductor memory is classified into Random Access Memory and Sequential memory. In Random Access Memory, the data can be stored and retrieved in any order but in Sequential Memory the data is stored and retrieved in serial order.

The Random Access Memory is further divided into Read and Write memory (RAM) and Read Only Memory (ROM). Data can be read and written in RAM but in ROM the user can only read the data and cannot write or change the data. RAM can retain the memory only when power is present but ROM can retain the data even when the power is absent. So, RAM is called volatile memory and ROM is called non-volatile memory.

RAM is further divided into Static RAM and Dynamic RAM. Static RAM can retain the data indefinite time as long as the power is present. But, in Dynamic RAM the stored data will gradually disappear even when the power is present. Hence, we need to refresh the Dynamic RAM periodically.

ROM is further classified into Mask ROM, PROM, EPROM, EEPROM and Flash ROM according to the process of entering the data into the memory (programming) and erasing the data.

Shift registers and Queues are classified under Sequential Access Memory. There are two types of shift registers namely Serial In Parallel Out (SIPO) and Parallel In Serial Out (PISO). Queues are classified into First In Last Out (FILO) and Last In First Out (LIFO).

4.2 RAM organization

The block diagram of a RAM chip is shown in figure.

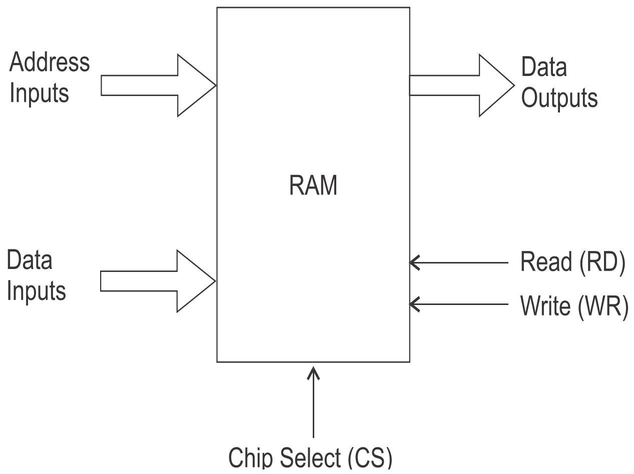


Figure : Block diagram of RAM

Address signals:

The RAM chip needs address inputs to select the required memory location.

The following table shows the number of address inputs (or lines) required for addressing the memory locations. If there are ‘n’ address lines, then the memory have 2^n locations.

Address lines (n)	Memory Size (No. of locations) (2^n)
4	16
8	256
10	1024 (1 Kilo)
11	2048 (2 Kilo)
12	4096 (4 Kilo)
13	8192 (8 Kilo)
14	16384 (16 Kilo)
15	32768 (32 Kilo)
16	65536 (64 Kilo)

Table : Address lines and Memory size

Data signals:

The data is given through the Data Inputs lines and the stored data is taken from the Data Outputs lines.

Control signals:

In addition to the address and data lines, there are some control signals like Chip Select (CS), Read (RD) and Write (WR). CS is used to select the memory chip, only when CS is active (CS = 1), the RAM chip will work. WR line used while writing the data into the memory locations and RD line is used while reading the data from the memory location selected by the address inputs.

The organization of 16 x 4 bits RAM chip is shown in figure.

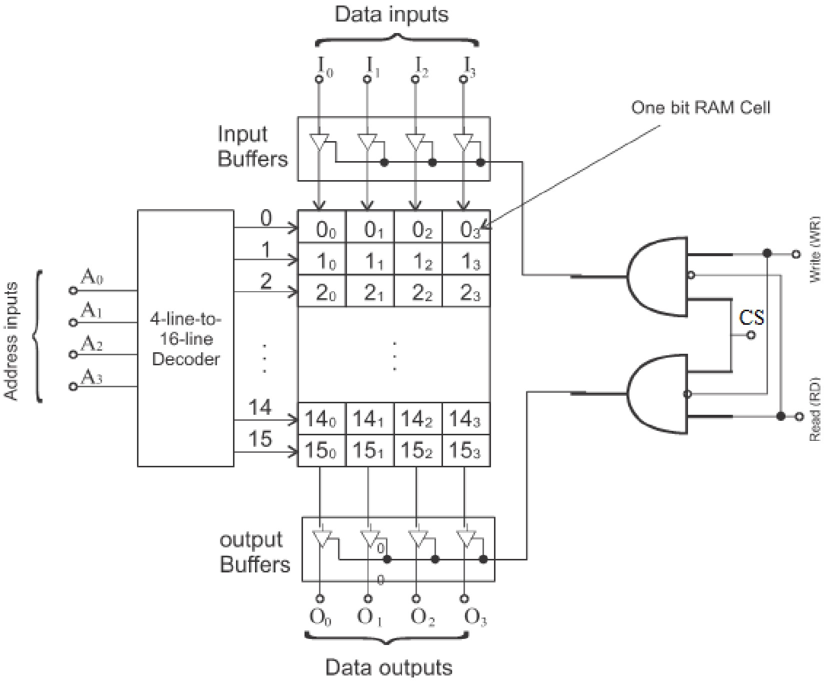


Figure: Organization of 16 x 4 bits RAM chip

It has 16 memory locations numbered from 0 to 15. Each memory location can store 4 bits numbered from 0 to 3. So, there are 64 (16 multiplied by 4) one bit memory cells. The particular memory location is selected by the 4-line to 16-line decoder. Hence, there are 4 address inputs lines labeled as A0 to A3.The data inputs I_0 to I_3 are given to the memory cells through the Input Buffers. The Input buffers are enabled by WR signal. The data outputs from the selected memory cells are taken from Output Buffers. The output buffers are enabled by RD signal. Both the Input and Output buffers are enabled only when CS input is HIGH. In recent memory chips, instead of using separate Input and Output Buffers, Bidirectional buffers are used.

Write operation:

The following sequences are required to write (store) a data into a particular memory location.

1. The Chip Select signal is applied to the CS pin (CS = 1).
2. The word (data) to be stored is applied to the Data Input pins I_0 to I_3.
3. The address of the desired memory location is applied to the Address Input pins A_0 to A_3.
4. The write command is applied by making WR pin to HIGH (1) level and RD pin to low (0) level.

Read operation:

The following sequences are required to read the data from a particular memory location.

1. The Chip Select signal is applied to the CS pin (CS = 1).
2. The address of the desired memory location is applied to the Address Input pins A_0 to A_3.
3. The read command is applied by making RD pin to HIGH (1) level and WR pin to low (0) level.
4. Data will be available at the Data output pins O_1 to O_3.

4.3 Static RAM (SRAM)

RAM is classified into Static RAM and Dynamic RAM. Static RAM can retain the data indefinite time as long as the power is present. But, in Dynamic RAM the stored data will gradually disappear even when the power is present. Hence, we need to refresh the Dynamic RAM periodically.

Static RAM circuits can be constructed using Bipolar Junction Transistor (BJT) or Metal Oxide Semiconductor Field Effect Transistor (MOSFET). Based on the type of transistor used, SRAM is classified into:

1. Bipolar Static RAM
2. MOS Static RAM

4.3.1 Bipolar Static RAM Cell

The simplified circuit diagram of Bipolar Static RAM cell is shown in figure. It stores 1 bit of information. The circuit is nothing but a flip-flop. It can store either 0 or 1 as long as the power is applied.

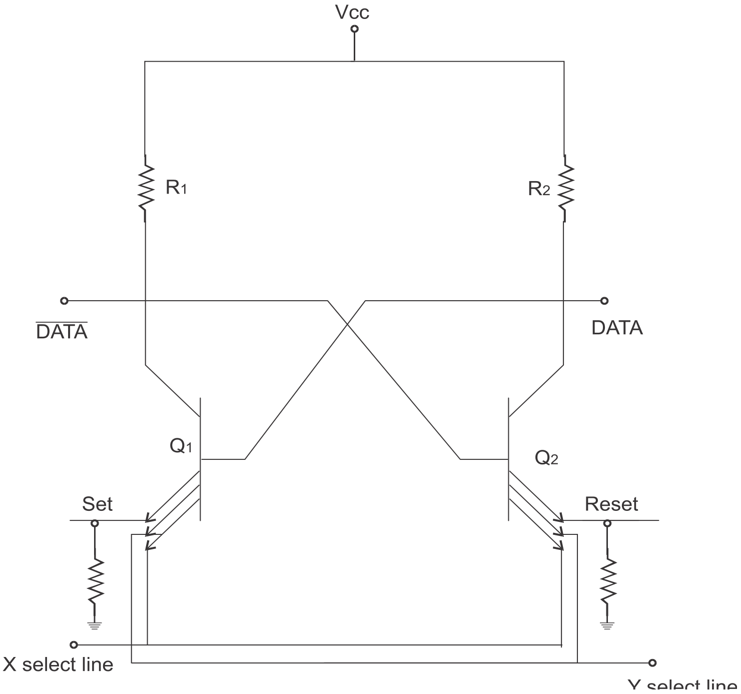


Figure :Bipolar Static RAM Cell

The Bipolar Static RAM cell is implemented using two BJTs with multiple emitters. The two transistors are cross-coupled together to get the bi-stable multi-vibrator (flip-flop) operation. Large number of similar cells is connected in matrix form. The X-select (ROW select) line and Y-select (COLUMN select) line are used to select a particular cell from the matrix.The Set input is used to store a '1' in the memory cell and Reset input is used to store a '0' in the memory cell. The output is taken either from Data line or from $\overline{Data}$line. The Q1 and Q2 are cross coupled inverters, hence one is always OFF while the other is ON.

Read operation: When the cell is selected by the X-select and Y-select lines, the data stored in the cell is available at Data and $\overline{Data}$ output lines.

Write operation: The cell is selected by the X-select and Y-select lines. By pulsing a HIGH on the SET input line, a '1' is stored in the cell. Similarly, by pulsing a HIGH in the RESET input line, a '1' is stored in the cell.

4.3.2 MOS Static RAM Cell

The simplified circuit diagram of MOS Static RAM cell is shown in figure. It stores 1 bit of information. The circuit is nothing but a flip-flop. It can store either 0 or 1 as long as the power is applied.

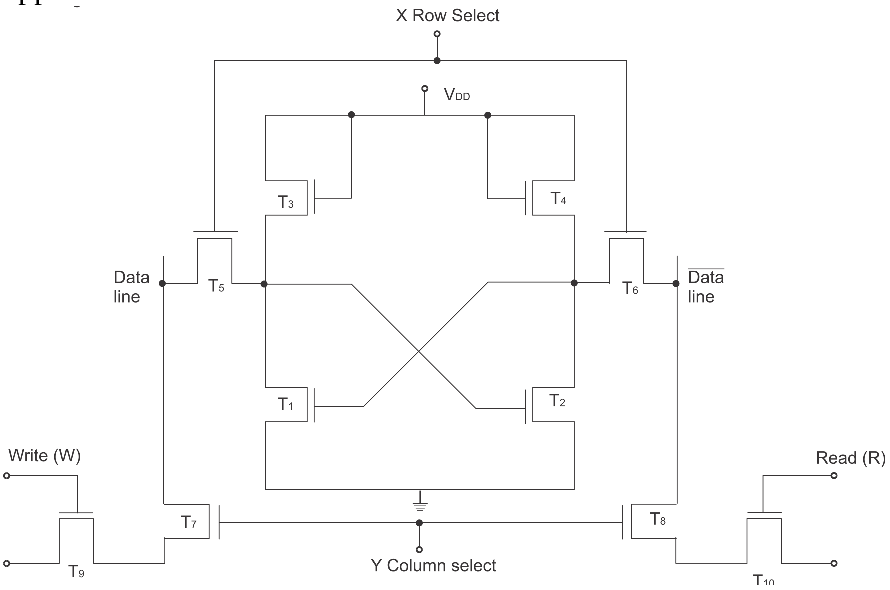


Figure : MOS Static RAM Cell

The MOS Static RAM cell is implemented using ten Enhancement mode MOSFETs. T1 and T2 form the basic cross-coupled inverters and T3 and T4 act as load resistors.T5 and T6 are used for taking the outputs. T7 and T9 are for write input and T8 and T10 are used for read input. X and Y lines are used for selecting the cell.

Write operation: Write operation is enabled by making W signal HIGH. The data input either '0' or '1' is given through the DATA IN terminal. When DATA IN is '1' T2 is turned ON and T1 is CUTOFF. When DATA IN is '0' T2 is turned CUTOFF and T1 is ON
Read operation: Read operation is enabled by making R signal HIGH. The cell is selected by the X-select and Y-select lines. The X-select line enables T6, Y-select line enables T8. R input selects T10. Hence, $\overline{D}$is available at the output terminal.

4.4 Dynamic RAM

Dynamic RAM stores the data as a charge on the capacitor. Figure shows the concept of dynamic RAM.

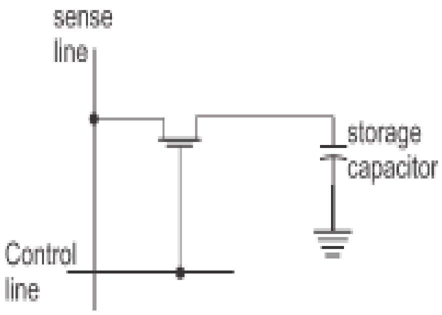


Figure : Concept of dynamic RAM

When ROW (Control) and COLUMN (Sense) lines go HIGH, the MOSFET conducts and charges the capacitor. When the ROW and COLUMN lines go low, the MOSFET opens and the capacitor retains the charge. The transistor acts like a switch only. In this way, it stores 1 bit. Since only a single MOSFET and capacitor are needed, the dynamic RAM occupies very less space when compared with static RAM.

The charge stored in the capacitor will discharge slowly and hence the data stored in the cell will lost in the course of time even when the power is present. This is the main disadvantage of dynamic RAM. Extra hardware is used to retain the charge in the capacitor. The extra hardware is called ‘Refresh logic’ and the operation is called ‘Refreshing’.

The Write, Read and Refresh operations are illustrated in figure. Three buffers are used for Write, Read and Refresh operations.

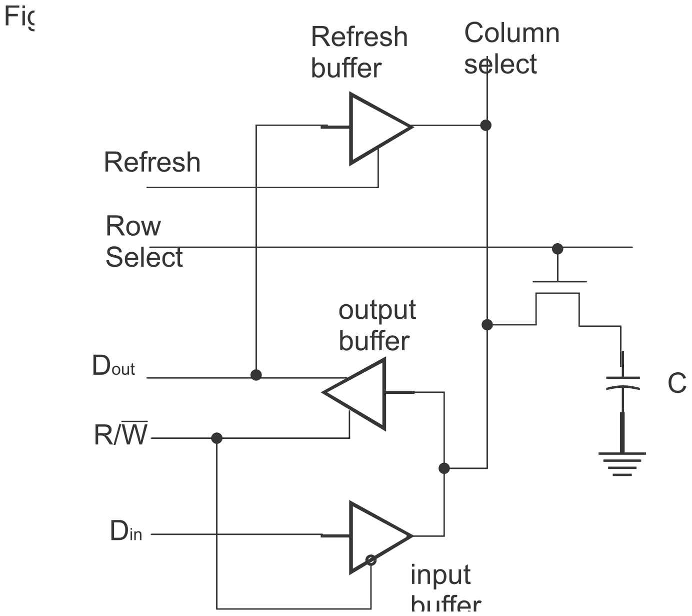


Figure : Dynamic RAM Cell

When R/$\overline{W}$ line is LOW, input buffer is enabled and output buffer is disabled. When R/$\overline{W}$ line is HIGH, input buffer is disabled and output buffer is enabled.

Write operation:

1. By making the R/$\overline{W}$ line LOW, the input buffer is enabled and the output buffer is disabled.
2. To write a '1'
    a. DIN line is made HIGH.
    b. ROW Select is made HIGH and the transistor is turned ON.
    c. Now, the capacitor is charged and stores a '1'.
3. To write a '0'
    a. DIN line is made LOW.
    b. ROW Select is made HIGH and the transistor is turned ON.
    c. Now, the capacitor is discharged and stores a '0'.
4. When the ROW Select is made LOW, the transistor is switched OFF and the charge on the capacitor is not disturbed.

Read operation:

1. By making the R/$\overline{W}$ line HIGH, the output buffer is enabled and input buffer is disabled.
2. ROW Select line is made HIGH. This turns ON the transistor and connects the capacitor to the DOUT line through the output buffer.

Refresh operation:

1. To enable the Refresh operation, R/$\overline{W}$ line, ROW line and REFRESH line are made HIGH.
2. The transistor is turned ON and the capacitor is connected to COLUMN line.
3. As R/$\overline{W}$ is HIGH, the output buffer is enabled and the stored data bit is applied to the input of the refresh buffer.
4. The output of the refresh buffer either '0' or '1' is applied to the COLUMN line and this maintains the charge on the capacitor.

Comparison between Static RAM (SRAM) and Dynamic RAM (DRAM)

S. No	Static RAM (SRAM)	Dynamic RAM (DRAM)
1.	Static RAM contains less memory cells per unit area. (Occupies more space)	Dynamic RAM contains more memory cells per unit area as compared to SRAM. (Occupies less space)
2.	Faster	Slower
3.	Data is stored in flip-flops.	Data is stored as a charge on the capacitor.
4.	Refreshing circuitry is not required. (Less hardware)	Refreshing circuitry is required to maintain the charge on the capacitor. Refreshing should be done in every few milliseconds. (More hardware)
5.	Cost is less.	Cost is more.

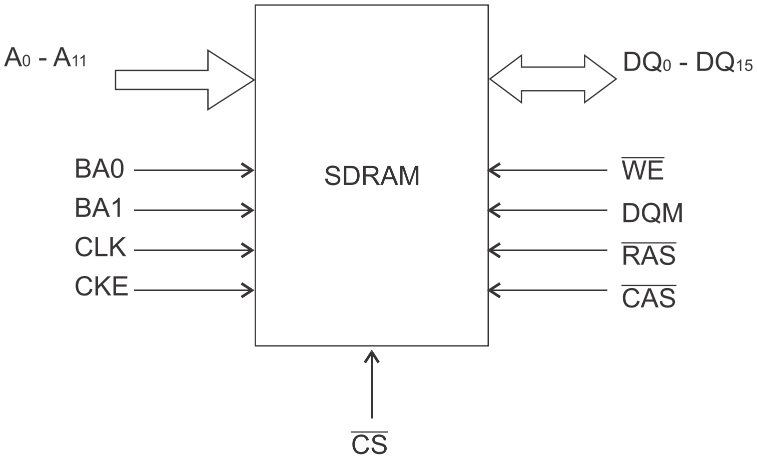
Table : SRAM Vs DRAM

4.5 SDRAM

Synchronous Dynamic Random Access Memory (SDRAM) is a faster memory than ordinary Dynamic RAM (DRAM). SDRAM is mainly used in computers. Ordinary DRAM operates in an asynchronous manner. They react to changes as the control inputs change, and also they are only able to operate as the requests are presented to them, dealing with one at a time. SDRAM is able to operate more efficiently. It is synchronized to the clock of the processor.

With SDRAM having a synchronous interface, it has an internal finite state machine (FSM) that pipelines incoming instructions. This enables the SDRAM to operate in a more complex fashion than an asynchronous DRAM. This enables it to operate at much higher speeds. As a result of this, SDRAM is capable of keeping two sets of memory addresses open simultaneously. By transferring data alternately from one set of addresses, and then the other, SDRAM reduces the delays associated with asynchronous DRAM.

<u>SDRAM Pins</u>



A0 – A11	Address Inputs
DQ0 – DQ15	Data Inputs / Outputs
RAS	Row Address Strobe Input
CAS	Column Address Strobe Input
CLK	Clock Input
CKE	Clock Enable Input
WE	Write Enable Input
BA0 & BA1	Bank Selection Inputs
DQM	Data Mask Input

Figure : Pins of a typical SDRAM Chip

Pipelining

Pipelining means that SDRAM can accept a new instruction before it has finished processing the previous one. In other words, it can effectively process two instructions at once. One write command can be immediately followed by another write command without waiting for the original data to be stored.

SDRAM architecture

The SDRAM architecture can be split into two main areas:

- ***Array:*** This is the area of the chip where the memory cells are implemented. It is normally divided into a number of banks, which in turn is split into smaller areas called segments.
- ***Periphery:*** This is the area of the chip where control and addressing circuitry is located as well as items such as line drivers and sense amplifiers.

SDRAM devices are internally divided into 2, 4 or 8 independent internal data banks. SDRAM can accept one command and transfer one word of data per clock cycle. Typical clock frequencies are 100 and 133 MHz. Chips are made with a variety of data bus sizes (most commonly 4, 8 or 16 bits).

SDRAM chips are generally assembled into 168-pin DIMMs (Dual In-line Memory Modules) that read or write 64 or 72 bits at a time.

4.6 DDR SDRAM

Double Data Rate Synchronous Dynamic Random Access Memory (DDR SDRAM) is a type of memory IC used in computers. Compared to Single Data Rate (SDR) SDRAM, the DDR SDRAM interface makes higher speeds possible by more strict control of the timing of the clock signals. Phase Locked Loops (PLLs) are used to meet the required timing accuracy. The interface uses double pumping (transferring data on both the rising and falling edges of the clock signal) to lower the clock frequency.

The name "double data rate" refers to the fact that a DDR SDRAM achieves nearly twice the bandwidth of a SDR SDRAM running at the same clock frequency, due to this double pumping.

With data being transferred 64 bits at a time, DDR SDRAM gives a transfer rate of (memory bus clock rate) × 2 (for dual rate) × 64 (number of bits transferred) / 8 (number of bits per byte). Thus, with a bus frequency of 100 MHz, DDR SDRAM gives a maximum transfer rate of 1600 MB/s (Mega Bits per second).

DDR SDRAM banks

DDR SDRAM memory has multiple banks. This enables the memory to provide multiple interleaved memory access, and this enables the overall memory bandwidth to be increased. DDR SDRAMs access multiple memory locations in a single read or write command.

DDR SDRAM power

DDR SDRAM provides an improvement in speed, but, the power dissipation is high. The power required by a DDR SDRAM is related to the number of rows that are open at any one time. Thus to gain the fastest operation, it is necessary to open a number of rows together, but this consumes more power.

4.7 Read only memory

ROM (Read Only Memory) is a form of semiconductor memory that retainsits contents even when the power supply is switched off. So, ROM is called Non-volatile memory. In ROM, we can read the data any number of times but data can be written to once during the manufacturing process only. ROM is used to store the "boot" or start-up program (so called firmware) that acomputer executes when powered on. ROM is also used in systems with fixed functionalities, e.g.controllers in cars, household appliances etc.

There are some special varieties of ROM such as PROM, EPROM, EEPROM and Flash memory.

PROM (Programmable Read Only Memory) : PROM is like ROM but allows end-users to write their own programs anddata. It requires special PROM writing equipment. The users can only write-once to PROM.

EPROM (Erasable Programmable Read Only Memory) : With EPROM we can erase (using strong ultra-violet light) the contentsof the chip and rewrite it with new contents, typically several thousand times. It is commonly used tostore the "boot" program of a computer, known as the firmware.

EEPROM (Electrically Erasable Programmable Read Only Memory)or E^{2}PROM: As the name implies the contents of EEPROMs are erasedelectrically. EEPROMs are also limited to the number of erase-writes that can be performed (e.g,1,00,000) but support updates (erase-writes) to individual bytes whereas EPROM updates the wholememory and only supports around 10,000 erase-write cycles.

FLASH memory: It is a cheaper form of EEPROM where updates (erase-writes) can only be performedon blocks of memory, not on individual bytes. Flash memories are found in USB Pen drive, memory cardsand typically range in size from 1GB to 32GB. The number of erase/write cycles to a block istypically several hundred thousand times.

Types of ROM

- ROM
    - Written during manufacture
    - Very expensive for small volumes
- Programmable ROM (PROM)
    - Read-only
    - Write-once
    - Needs special equipment to program
    - Convenience
- Erasable Programmable (EPROM)
    - R/W
    - Have to erase before write
    - Erased by UV

- Electrically Erasable (EEPROM)
  - R/W
  - Takes much longer to write than read
  - Individual bytes programmable
- Flash memory
  - Faster erase (block erase)
  - Higher density than EEPROM

4.8 ROM organization

ROM is a read only memory. We can't write data in this memory. It is a non-volatile memory i.e. it can retain the data even when the power is switched off. Generally ROM is used to store permanent data like computer programs and look-up tables. Figure shows the organization of simple ROM using diode matrix. Diodes are physically fabricated in the required positions during manufacturing of the memory chip. ROMs are cheap when produced in large volumes.

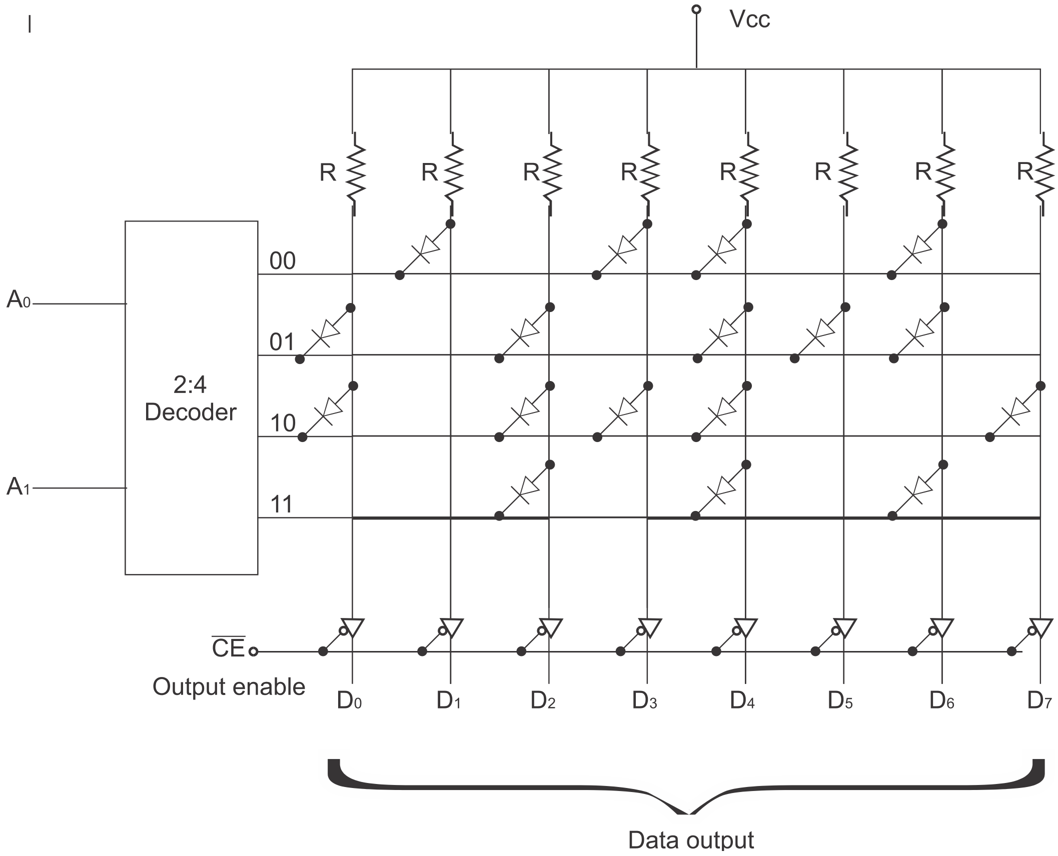


Figure : Four-byte Diode Matrix ROM

The address lines A0 and A1 are decoded by 2:4 decoder and used to select one of the four rows. The active low output of the decoder produces a '0' on the selected row. When the row line is '0' and if the diode is present in the column, we get '0' at the particular column. Data is available on the output lines only when the output enable ($\overline{OE}$) signal is low. Table shows the contents of ROM at four locations.

Address in Binary	Binary data								Data in Hex
	D0	D1	D2	D3	D4	D5	D6	D7	
00	1	0	1	0	0	1	0	1	A5
01	0	1	0	1	0	0	0	1	51
10	0	1	0	0	0	1	1	0	46
11	1	1	0	1	0	1	0	1	D5

Table : Data stored in ROM

4.9 Expanding memory

We need to expand the memory for the following two reasons.

1. To increase the word size (number of bits in each memory location).
2. To increase the memory capacity (number of locations).

4.9.1 Expanding word size

The word size i.e. the number of bits in the data lines can be expanded by connecting two or more ICs together. The connection diagram is shown in figure.

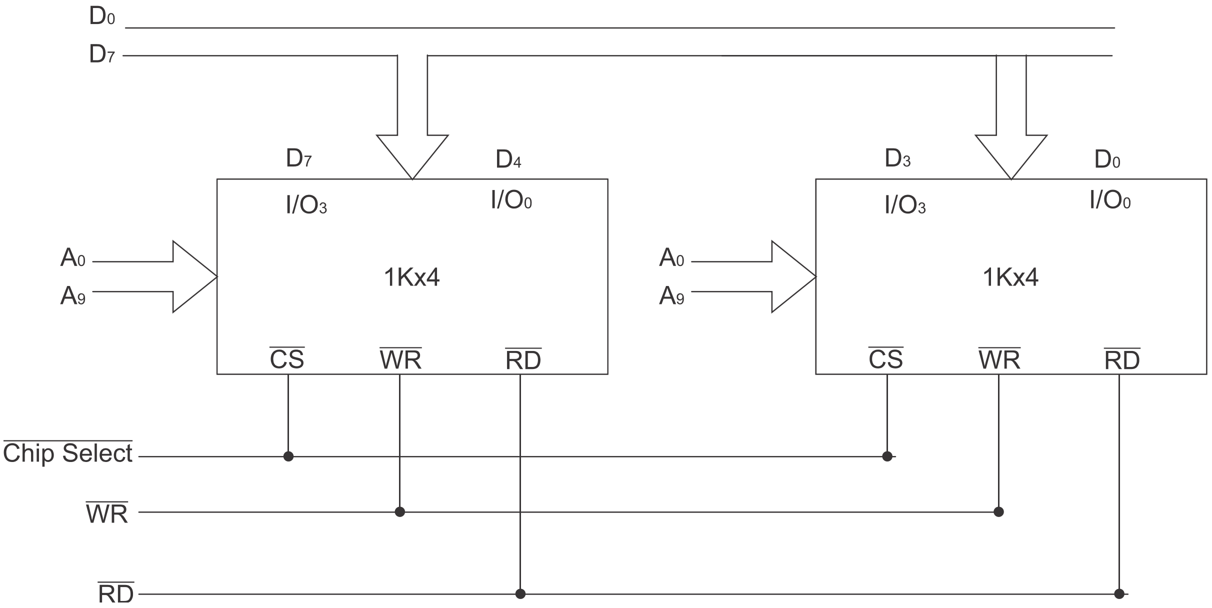


Figure : Expanding word size of memory

The ICs are connected in such a way that the data lines are connected in series and the address lines are connected in parallel. There are two 1K x 4 (1024 locations, each location has 4 bits) memory ICs are used to get 1K x 8 memory. For 1024 locations, there are 10 address lines A0 to A9. Both memory ICs are selected simultaneously by common chip select signal. When a particular address is given, both ICs are selected. Out of 8-bits, the first 4-bits are taken from one IC and the next 4-bits are taken from another IC.

4.9.2 Expanding memory capacity

The memory capacity (i.e. the number of locations) can be increased by connecting two or more ICs in parallel. The figure shows the connection diagram for 16 K x 8 memory using four 4K x 8 memory chips.

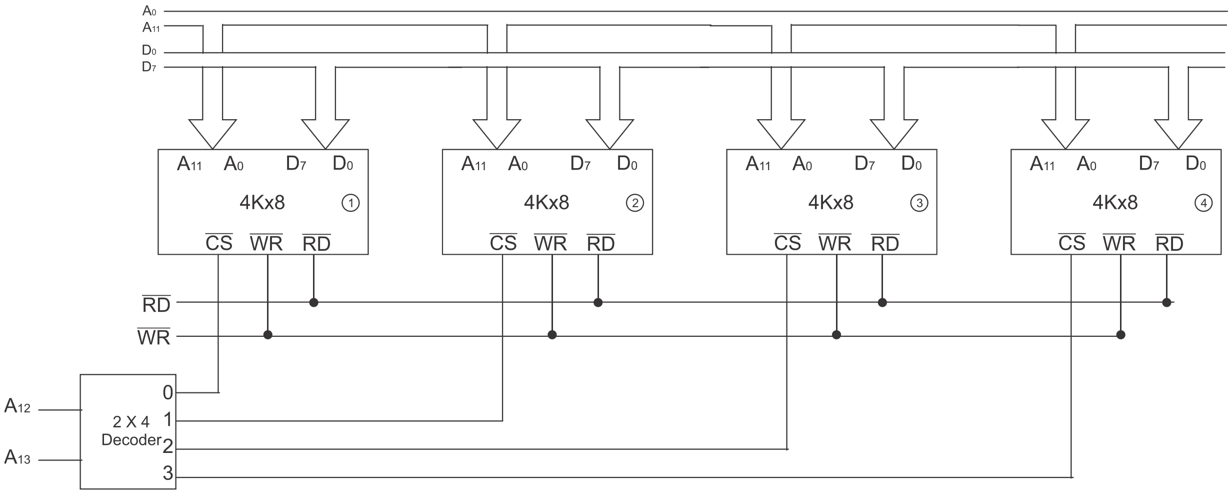


Figure : 16K x 8 Memory expansion using four 4k x 8 memory chips

Address lines A12 and A13 are decoded by the 2:4 decoder to select any one of the memory chip. When A12 and A13 are 00, chip 1 will be selected. Similarly when A12 and A13 are 11, chip 4 will be selected. Address lines A0 to A11 are used to select a particular memory location in the selected chip.

4.10 PROM (Programmable Read Only Memory)

PROM is like ROM but allows end-users to write their own programs and data. It requires special PROM writing equipment. The users can only write-once to PROM. Figure shows four byte PROM.

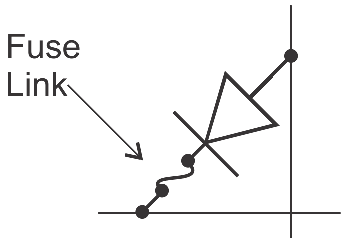


Figure : Fuse link used in PROM

It has diodes in every bit position; therefore the output is initially all 0s. Each diode has a fusible link in series with it. We can blow the fuse by selecting the particular row and column and applying some high current (20 to 50 mA for a period of 5 to 20 µs) at the corresponding output. This can be done by a special device called PROM programmer.

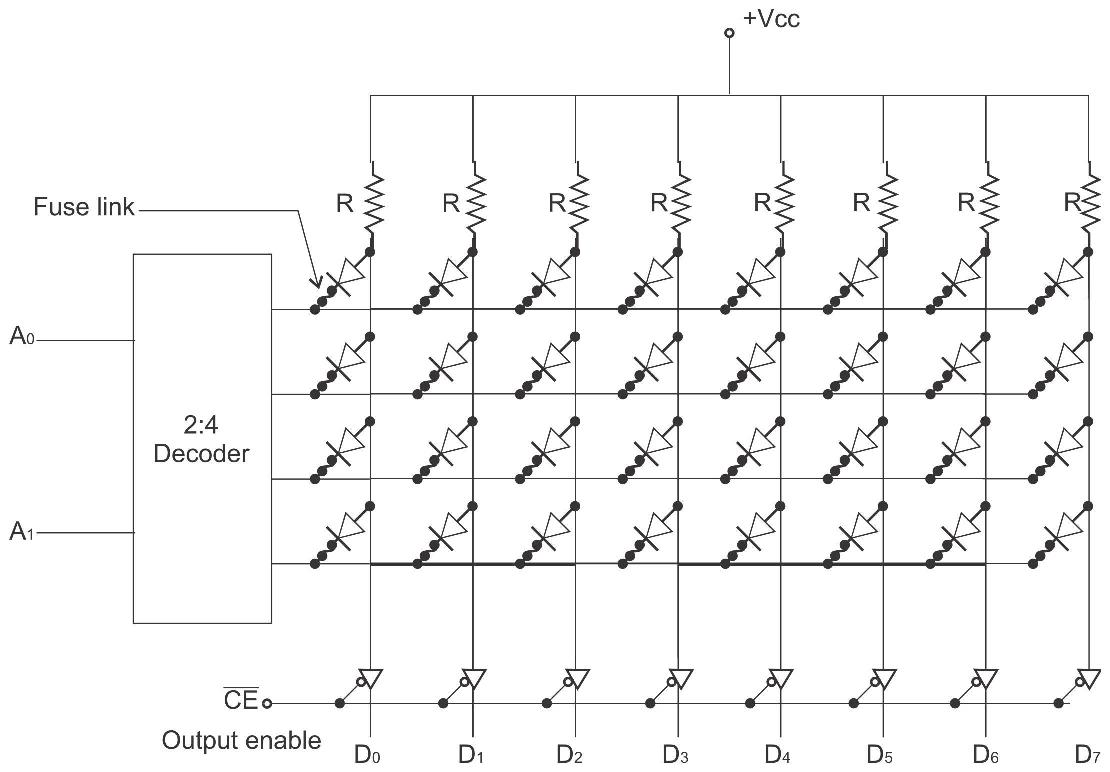


Figure : Organization of four byte PROM

This process is also known as burning of PROM. The PROMs are one-time programmable. Once programmed, the information stored is permanent. The blown fuse stores a ‘1’. The fuse uses materials like nichrome and polycrystalline.

Comparison between ROM and PROM

S. No.	ROM	PROM
1.	Non-programmable.	Programmable.
2.	Not flexible (because data can't be changed.	Flexible (Field-Programmable i.e. can be programmed in any place of work).
3.	It is comparatively slower.	It is comparatively faster.
4.	Economical only when produced in large volumes.	Economical even when produced in small volume.
5.	Used in PC's.	Mainly used for research and development purpose.
6.	Writing of cells done by 'masking’ mechanisms.	Mainly done electrically.

Table : ROM vs PROM

4.11 EPROM (Erasable Programmable Read Only Memory)

EPROM (Erasable Programmable Read Only Memory) is a type of memory chip. It retains its data even when the power supply is switched off. So, it is called non-volatile memory. It is an array of floating-gate transistors individually programmed by a special electronic device. The device supplies higher voltages than the normal operating voltage. Once programmed, an EPROM can be erased by exposing it to strong ultraviolet light source (such as from a mercury-vapor light).

Figure : A typical EPROM chip

EPROMs are easily identified by the transparent quartz window in the top of the package, through which the silicon chip is visible. The window permits exposure to UV light during erasing.

Operation

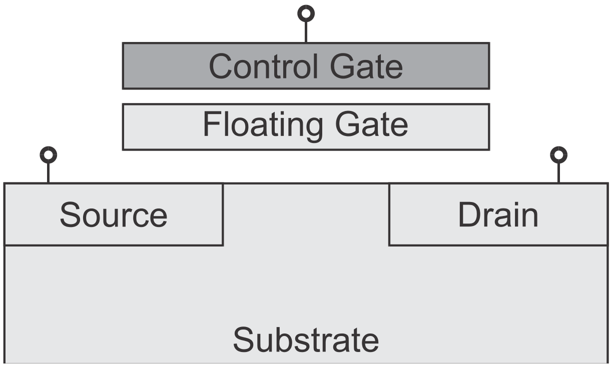


Figure : Cross-section of floating-gate transistor used in EPROM

Each storage location of an EPROM consists of a single field-effect transistor. Each field-effect transistor consists of a channel in the semiconductor body of the device. Source and drain contacts are made to regions at the end of the channel. An insulating layer of oxide is grown over the channel, then a conductive (silicon or aluminum) gate electrode is deposited, and a further thick layer of oxide is deposited over the gate electrode. The floating gate electrode has no connections to other parts of the integrated circuit and is completely insulated by the surrounding layers of oxide. A control gate electrode is deposited and further oxide covers it.

Programming

Storing data in the memory requires selecting a given address and applying a higher voltage to the transistors. This creates an avalanche discharge of electrons, which have enough energy to pass through the insulating oxide layer and accumulate on the gate electrode. When the high voltage is removed, the electrons are trapped on the electrode. Because of the high insulation value of the silicon oxide surrounding the gate, the stored charge cannot leak away and the data can be retained for many years.

Erasing

To erase the data stored in the array of transistors, ultraviolet light is directed onto the die. Photons of the UV light cause ionization within the silicon oxide, which allow the stored charge on the floating gate to dissipate. Since the whole memory array is exposed, all the memory is erased at the same time. The process takes several minutes for UV lamps of convenient sizes; sunlight would erase a chip in few weeks.

The erasing window must be kept covered with an opaque label to prevent accidental erasure by the UV found in sunlight or camera flashes.

4.12 EEPROM (Electrically Erasable Programmable Read Only Memory)

EEPROM (Electrically Erasable Programmable ROM) is a ROM chip which can be erased and re-programmed for unlimited number of times, without expensive or time-consuming erasing processes. EEPROM is also called E^{2}PROM (e-squared PROM) and EAPROM (Electrically Alterable PROM).To erase the data, a relatively high voltage is required. Only one external power supply is required since the high voltage for program/erase is internally generated. In this way the memory device could run from a single supply, thereby considerably reducing the cost of an overall circuit using an EEPROM and simplifying the design. Write and erase operations are performed on a byte per byte basis.

EEPROM memory uses the same basic principle that is used by EPROM memory technology. The memory cell has two field effect transistors. One of these is the floating gate storage transistor. Electrons can be made to become trapped in this gate and the presence or absence of electrons is referred as a ‘0’or a ‘1’. The other transistor is known as the access transistor and it is required for the operational aspects of the EEPROM memory cell.

Comparison between EPROM and EEPROM

S. No.	EPROM	EEPROM
1.	Erasing by UV light.	Electrically erased.
2.	Single byte erase is not possible. When UV light is passed, the complete data will be erased.	Single byte erase is possible.
3.	Quartz window is provided for UV erasing.	Quartz window is not required.

Table : EPROM vs EEPROM

4.13 Flash memory

Flash memory is a form of non-volatile memory (EEPROM) that can be electrically erased and reprogrammed. It is erased and programmed in blocks consisting of multiple locations (usually512 bytes in size). Flash memory costs far less than EEPROM and therefore has become the dominant technology wherever a significant amount of non-volatile, solid-state storage is needed.

Examples of Flash memory:

- Computer's BIOS (Basic Input Output System) chip
- USB flash drives
- Compact Flash (digital cameras)
- Smart Media (digital cameras)
- Memory Stick (digital cameras and cell phones)

There are two types of flash memory.

1. NOR flash
2. NAND flash

The characteristics of flash memory vary according to its type.

NOR flash

NOR-based flash has long erase and write times, but has a full address/data (memory) interface that allows random access to any location. This makes it suitable for storage of program code that needs to be infrequently updated, such as a computer's BIOS or the firmware of set-top boxes. Its endurance is 10,000 to 1,000,000 erase cycles.

NAND flash

NAND flash has faster erase and write times, higher density, and lower cost per bit than NOR flash, and ten times the endurance. However its I/O interface allows only sequential access to data. This makes it suitable for mass-storage devices such as PC cards and various memory cards, and somewhat less useful for computer memory.

A blank flash memory has all cells as 1's.It can be read or programmed a byte or word at a time in a random fashion, but it can only be erased a block at a time. Once a byte has been programmed it cannot be changed again until the entire block is erased. Erasing is applied to one or more blocks by the application of a high voltage that returns all cells to a 1 state.

Flash Memory – Program Operation

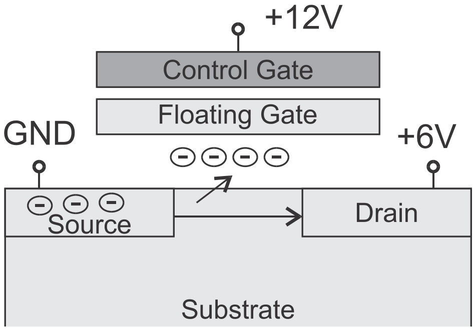


Figure : Flash memory – Program operation

- Apply 6V between drain and source
- Generates hot electrons that are swept across the channel from source to drain
- Apply 12 V between source and control gate
- The high voltage on the control gate overcomes the oxide energy barrier, and attracts the electrons across the thin oxide, where they accumulate on the floating gate
- Called channel hot-electron injection (HEI)

Flash Memory – Erase Operation

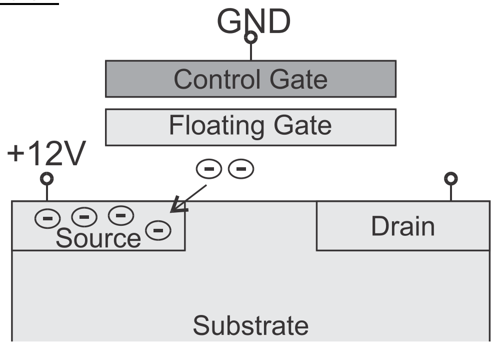


Figure : Flash memory – Erase operation

- Floating the drain, grounding the control gate, and applying 12V to the source
- A high electric field pulls electrons off the floating gate
- Called Fowler-Nordheim (FN) tunneling

Flash Memory – Read Operation

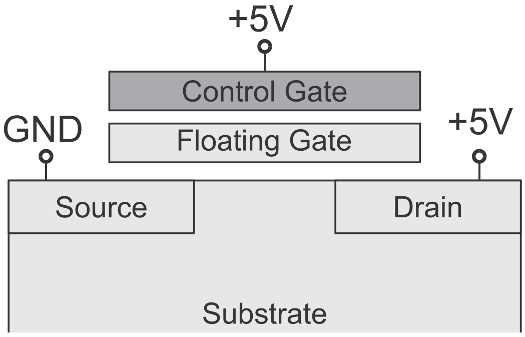


Figure : Flash memory – Read operation

- Apply 5V on the control gate and drain, and source is grounded
- The drain to source current is detected by the sense amplifier
- The applied voltage on the control gate is not sufficient to turn it on. The absence of current results in a 0 at the corresponding flash memory output

4.14 Anti fuse technology

In memory devices like ROM, we need to store a permanent ‘0’ or ‘1’ in memory cells. Anti fuse is a technology used for storing one bit in memory cell.

An **anti fuse**is the opposite of a regular fuse. Anti fuse is normally an open circuit until a **programming current**is passed through it (about 5 mA). In a poly–diffusion anti fuse the high current density causes a large power dissipation in a small area, which melts a thin insulating dielectric between polysilicon and diffusion electrodes. This forms a thin (about 20 nm in diameter), permanent, and resistive silicon **link**.

Figure shows a poly–diffusion anti fuse with an **oxide–nitride–oxide** (**ONO**) dielectric sandwich of silicon dioxide grown over the n-type anti fuse diffusion, a silicon nitride layer, and another thin silicon dioxide layer.

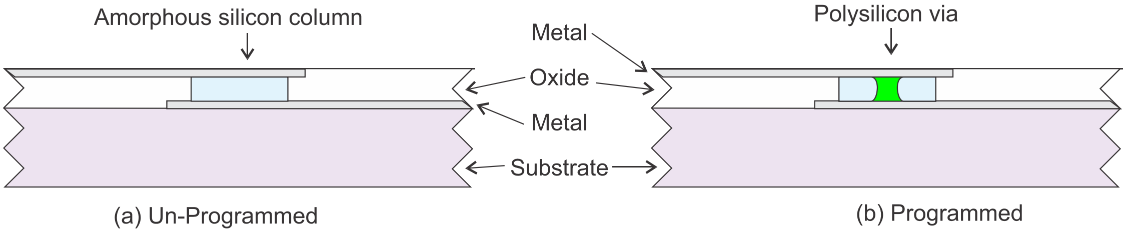


Figure : Anti fuse technology

In its un-programmed state, the amorphous silicon acts as an insulator with a very high resistance in excess of one billion ohms. The act of programming this particular element effectively "grows" a link by converting the insulating amorphous silicon into conducting polysilicon.

Comparison between Flash memory and Anti fuse technology

S. No	Parameter	Flash memory	Anti fuse technology
1.	Technology	Floating Gate Transistor	Oxide breakdown
2.	Endurance (Erasing – Programming cycle)	Many	Less than 5
3.	Power consumption	Medium	Medium
4.	Programming	Can be done in the field itself	Can be done in the field itself
5.	System performance	Low	High
6.	Design security	No	Very high

Table : Flash vs Anti fuse

☺☺☺☺☺☺

UNIT – V

MICROPROCESSOR – 8085

5.1 Microprocessor - Introduction

Microprocessor is a multipurpose programmable integrated circuit (IC) chip. It has computing and decision-making capabilities similar to the central processing unit (CPU) of a computer. Figure 5.1 shows the parts of a microprocessor based system. The microprocessor works as per the program stored in memory. Data from the external world enters the microprocessor through the input unit. Data may be sent to the external world through the output input.

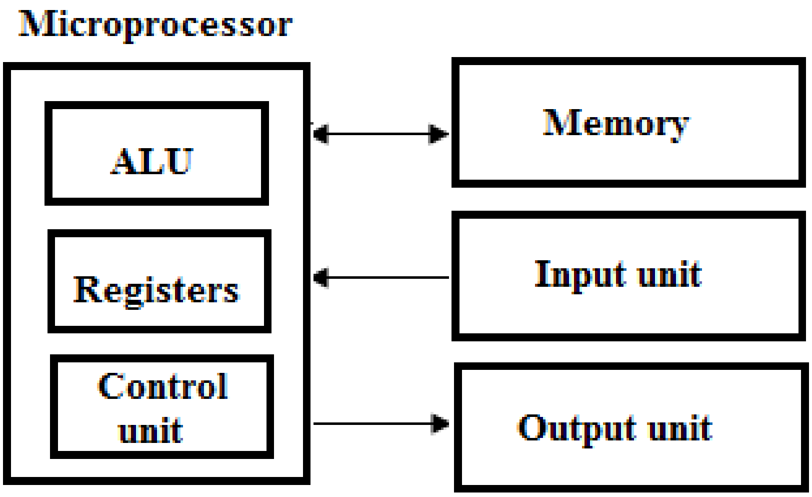


Figure 5.1 : Microprocessor based system

Some of the important features of microprocessor are :

The microprocessor IC consists of ALU, Registers and Control unit.

1. **ALU** – Arithmetic and Logic Unit. ALUperforms the computing and decision making operations.
2. **Registers**–Registers are used for storing the internal temporary data.
3. **Control unit** – The control unit controls the operation of the microprocessor and the devices connected to the microprocessor.
4. The microprocessor can understand a set of basic commands **(instruction set)**.
5. The microprocessor has several pins for transmitting address signals to the memory and I/O (Input / Output) devices. These pins are known as **address bus**.
6. The microprocessor has several pins for transmitting data signals to the memory and I/O devices. These pins are known as **data bus**.
7. The microprocessor has few pins for controlling the memory and I/O devices. These pins are known as **control bus**.

5.1.1 Evolution of microprocessor

The history of the development of microprocessor is given below:

4-bit microprocessors:

- 4004 was the first microprocessor introduced in 1971 by Intel Corporation, USA.
- Operating on 4-bits of data at a time.
- Has the capabilities for addition, subtraction, comparison and logical (AND and OR) operations.
- Examples: Intel's 4004, Intel's 4040, Rockwell International's PPS4, Toshiba's T3472

Examples

4-bit data : 0110

8-bit data : 1011 0111

16-bit data : 1100 1111 0101 0111

8-bit microprocessors:

- 8008 was the first 8-bit microprocessor introduced in 1973 by Intel Corporation, USA.
- Perform arithmetic and logical operations on 8-bit data.
- Examples: Intel's 8008, Intel's 8080, Intel's 8085, Motorola's M6800, National Semiconductor's NSC 800, Zilog Corporation's Z80, Fairchild's F8, Hitachi's 6809.

12-bit microprocessors:

- Performs arithmetic and logical operations on 12-bit data
- Examples : Intersil's IM6100, Toshiba's T3190

16-bit microprocessors:

- Performs arithmetic and logical operations on 16-bit data
- Examples : Intel's 8086, Intel's 8088, Intel's 80286, Fairchild's 9440, Data General's mN601, Texas Instrument's TMS9900, Motorola's M68000, Zilog's Z8000

32-bit microprocessors:

- Performs arithmetic and logical operations on 32-bit data
- Examples: Intel's 80386, Intel's 80486, Intel's iAPX432, Motorola's 68020, Motorola's 68030, National's 32032, National's 32523, Inmos' T414, Inmos' T800

64-bit microprocessors:

- Performs arithmetic and logical operations on 64-bit data
- Intel's Pentium microprocessor executes 100 million instructions per second (MIPS).
- Examples: Intel's Pentium (80586), Intel's Pentium Pro, Intel's Pentium II, Celeron, Intel's Pentium III and Intel's Pentium IV

5.1.2 Introduction to 8085 Microprocessor

8085 Microprocessor was first introduced by Intel Corporation, USA in the year 1976. It has 6,500 transistors. The important features of 8085 are given below:

- 8-bit microprocessor. i.e. data bus width is 8 bits.
- 16-bit address bus. Hence, the maximum memory range is 64 Kilo Bytes (2^{16}). The lower order address lines A0 to A7 is multiplexed with the data lines D0 to D7.
- Available in 3 MHz, 5 MHz and 6 MHz clock frequencies
- Built-in clock generator circuit
- Serial In and Serial Out data port for serial data communication
- Three maskable and one non-maskable vectored interrupts
- Decimal, binary and double precision arithmetic operations
- Single +5V power supply
- Available in 40 pin Dual Inline Package (DIP)

Pin diagram of 8085 Microprocessor

There are 40 pins in Intel 8085 microprocessor. The photo image is shown in figure 5.2 and pin diagram is shown in figure 5.3.

Figure 5.2 : Intel 8085 microprocessor

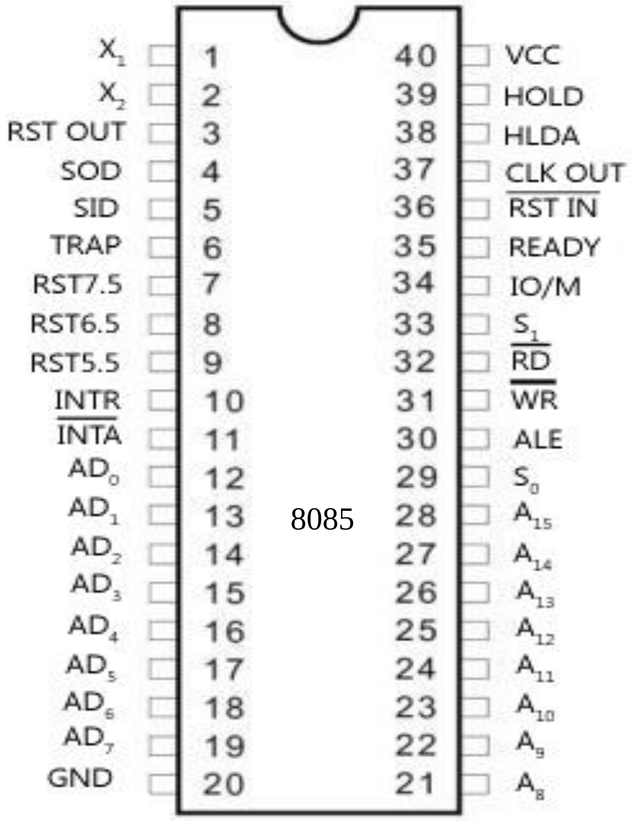


Figure 5.3 : Pin diagram of Intel 8085 microprocessor

The signals are classified into the following categories.

S.No.	Group
1.	Address bus
2.	Multiplexed Address / Data bus
3.	Control and Status signals
4.	Peripheral initiated signals
5.	Clock signals
6.	Reset signals
7.	Interrupt signals
8.	Serial I/O signals
9.	Power supply and Ground signals

The signal diagram is shown in figure 5.4.

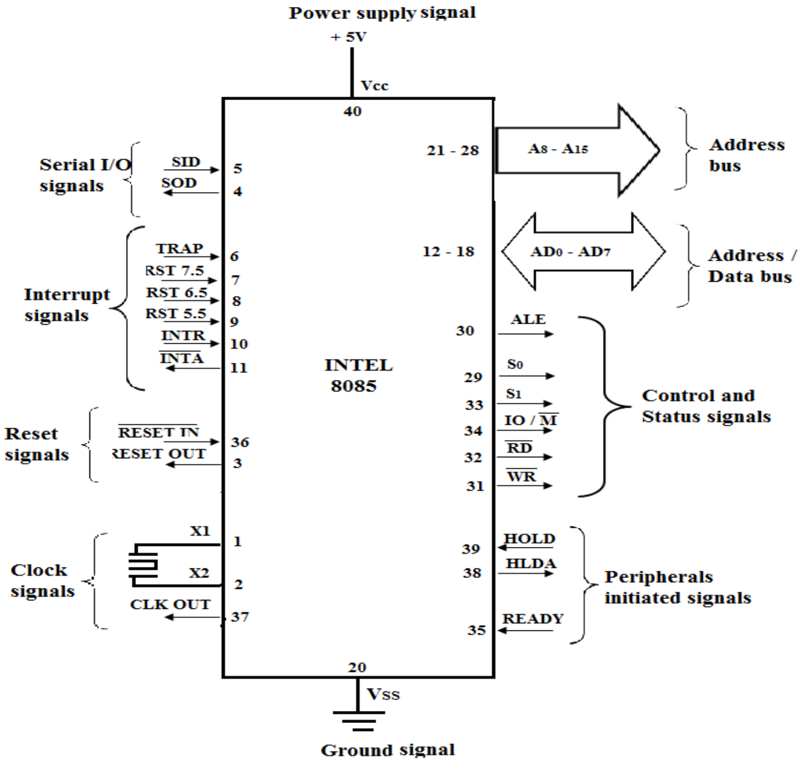


Figure 5.4 : Signal diagram of Intel 8085 microprocessor

Pin number	Signal name	Description
Address Bus		
21 – 28	$A_8 – A_{15}$	Higher order byte (8 bits) of address bus. The address bus is unidirectional (one-way, ie. microprocessor to memory).
Multiplexed Address / Data bus		
12 -19	$AD_0 – AD_7$	Lower order byte (8 bits) address bus (A_0 - A_7). And Data bus (8 bits) ($D_0 – D_7$).The data bus is bidirectional (two-wayie. microprocessor to memory and memory to microprocessor).
Control and Status signals		
30	ALE	ALE - Address Latch Enable. Used for de-multiplexing the Address / Data bus. i.e. separating the lower order address lines ($A_0 – A_7$) and data lines ($D_0 – D_7$) from ($AD_0 – AD_7$).
29	S_0	The microprocessor indicates its status (memory write, memory read, I/O write, I/O read, Opcode fetch, Halt, Hold, Reset) through these three output pins. IO/$\overline{M}$ will be ‘1’ for I/O operation and ‘0’ for memory operation.
33	S_1	
34	IO / $\overline{M}$	

32	$\overline{RD}$	This is an active low signal. It will be 0 during memory read and I/O read operation.
31	$\overline{WR}$	This is an active low signal. It will be 0 during memory write and I/O write operation.
Peripheral initiated signals		
35	READY	This pin is used for interfacing slow devices with the microprocessor.
39	HOLD	DMA (Direct Memory Access) signals: HOLD (hold request) HLDA (hold acknowledge)
38	HLDA	
Clock signals		
1, 2	X_1, X_2	Used for connecting the crystal for generating the clock signal.
37	CLK OUT	Clock signal for the peripheral devices.
Reset signals		
36	$\overline{RESETIN}$	Used to apply reset signal to the microprocessor. When this pin goes low, the microprocessor is reset and begins executing the instruction from the memory location 0000. All the registers inside the microprocessor is reset to zero.
3	RESET OUT	Used to reset the peripheral devices.
Interrupt signals		
6	TRAP	Interrupt request input signals to the microprocessor.
7	RST 7.5	
8	RST 6.5	
9	RST 5.5	
10	INTR	
11	$\overline{INTA}$	Interrupt Acknowledgesignal from the microprocessor to the peripheral devices.
Serial I/O signals		
5	SID	Serial Input Data. Used to receive serial data bits.
4	SOD	Serial Output Data. Used to transmit serial data bits.
Power supply and Ground signals		
40	V_{CC}	Power supply pins. +5V to V_{CC} GND to V_{SS}
20	V_{SS}	

5.1.3 Architecture of 8085 Microprocessor

The internal architecture (block diagram) of 8085 Microprocessor is shown in figure 5.5.

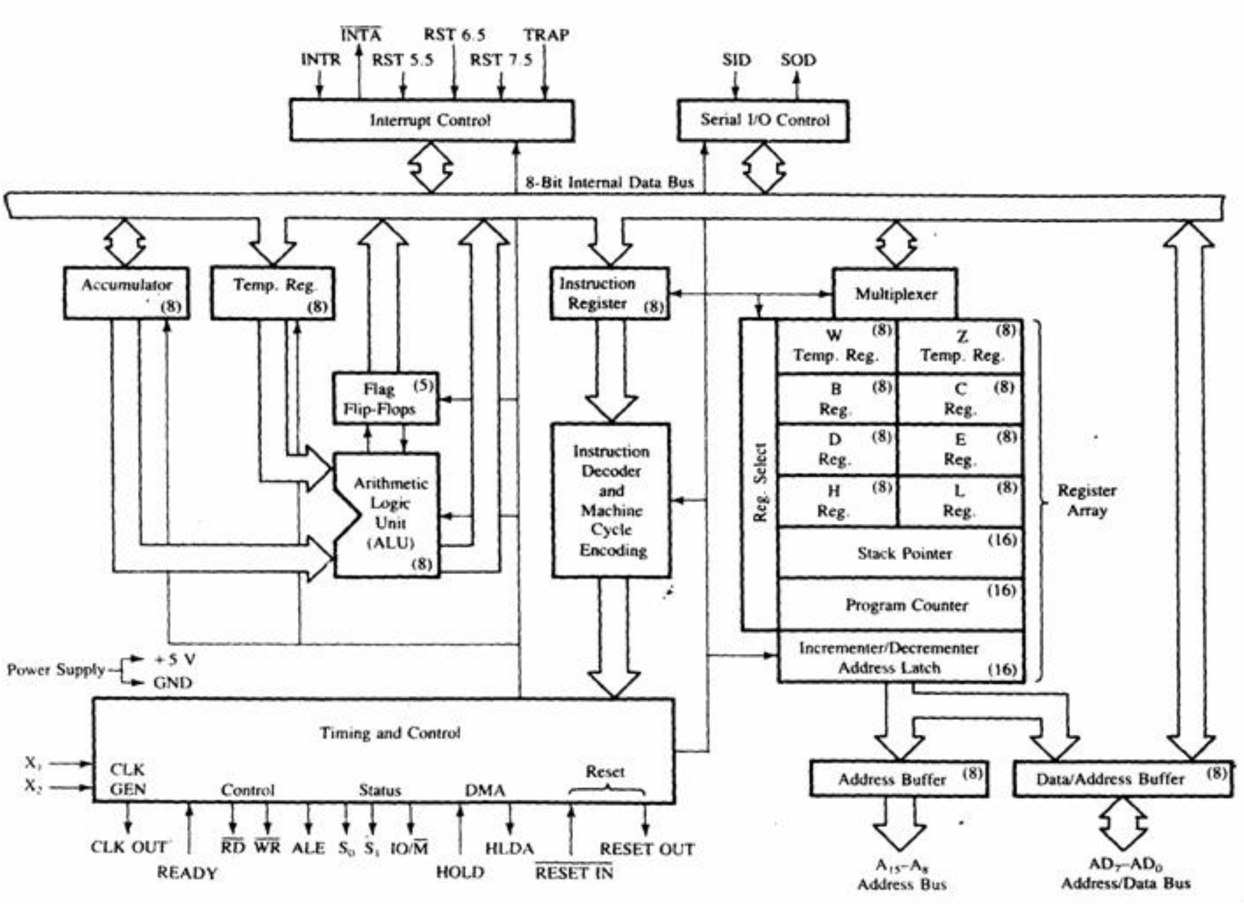


Figure 5.5 : Architecture of 8085 Microprocessor

The following are the functional blocks in the 8085Microprocessor.

1. Accumulator
2. Temporary register
3. Arithmetic and Logic Unit (ALU)
4. Flag register
5. Instruction Register
6. Instruction Decoder and Machine cycle encoder
7. General purpose registers
8. Stack Pointer
9. Program Counter
10. Incrementer / Decrementer
11. Timing and Control unit
12. Interrupt control
13. Serial I/O control
14. Address buffer and Address / Data buffer

1. Accumulator (A-register)

It is an 8-bit register. It is associated with ALU. The accumulator is also called A-register. During the arithmetic / logic operations, one of the operand is available in Accumulator. The result of the arithmetic / logic operations is also stored in the Accumulator.

2. Temporary (TEMP) register

It is an 8-bit register. It is also associated with ALU. This register is used to hold one of the data (from memory or general purpose registers) during an arithmetic / logic operation.

3. Arithmetic and Logic Unit (ALU)

The Arithmetic and Logic Unit includes Accumulator, Temporary register, arithmetic and logic circuits and flag register. The ALU can perform arithmetic (such as addition and subtraction) and logic operations (such as AND, OR and EX-OR) on 8-bit data. It receives the data from accumulator and or TEMP register. The result is stored in the accumulator. The conditions of the result (such as carry, zero) are indicated in the flags.

4. Flag register

It is an 8-bit register. But only five bits are used. The flag positions in the flag register are shown in figure 5.6.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
S	Z	-	AC	-	P	-	CY

Figure 5.6 : Flag register of 8085

The flags are affected by the arithmetic and logic operations in the ALU. The flag register is also known as Status register or Condition code register. There are five flags namely Sign (S) flag, Zero (Z) flag, Auxiliary Carry (AC) flag, Parity (P) flag and Carry (CY) flag.

- Sign (S) flag: Sign flag is set (1) if the bit D_7 of the result in the accumulator is 1, otherwise it is reset (0). This flag is set when the result is negative. This flag is used only for signed numbers.
- Zero (Z) flag: Zero flag is set (1) if the result in the accumulator is zero, otherwise it is reset (0).

- Auxiliary Carry (AC): Auxiliary Carry flag is set (1) if there is a carry from bit position $D_3$of result in the accumulator, otherwise it is reset (0). This flag is used for BCD operations.

- Parity (P) flag: Parity flag is set (1) if the result in the accumulator has even number of 1s, otherwise it is reset (0).

- Carry (CY) flag: Carry flag is set (1) if the result of an arithmetic operation results in a carry from bit position D_7, otherwise it is reset (0). This flag is also used to indicate a borrow condition during subtraction operations.

5. Instruction register

When an instruction is fetched from memory, it is stored in the Instruction register. It is an 8-bit register. This resister cannot be used in the programs.

6. Instruction Decoder and Machine cycle encoding

This unit decodes the instruction stored in the Instruction register. It determines the nature of the instruction and establishes the sequence of events to be followed by the Timing and control unit.

7. General purpose registers

There are six 8-bit general purpose registers namely B, C, D, E, H and L registers. B and C registers are combined together as BC register pair for 16-bit operations. Similarly D and E registers can be used as DE resister pair and H and L as HL register pair. The HL register pair is also used as memory pointer (M-register) for storing 16-bit address in some instructions.There are two more 8-bit temporary registers W and Z. These registers are used to hold data during the execution of some instructions. W and Z registers cannot be used in programs.

8. Stack Pointer (SP)

Stack is a portion of memory (RAM) used as FILO (First In Last Out) buffer. This is mainly used during subroutine operations. Stack Pointer is a 16-bit register used as a memory pointer (16-bit address) for denoting the stack position in memory. The Stack pointer is decremented each time when data is loaded into the stack and incremented when data is retrieved from the stack. Stack pointer always points to the top of the stack memory.

9. Program Counter (PC)

The Program Counter (PC) is a 16-bit register. It is used to point the address of the next instruction to be fetched from the memory. When one instruction is fetched from memory, PC is automatically incremented to point out the next instruction.

10.Incrementer / Decrementer

This unit is used to increment or decrement the contents of the 16-bit registers.

11.Timing and Control unit

This unit synchronizes all the microprocessor operations with the clock and generates the control signals necessary for communication between microprocessor and peripherals. The internal clock generator is available in this unit.This unit has the micro programs for all the instructions to carry out the micro steps required in completing the instructions. This unit receives signals from the Instruction decoder and Machine cycle encoding unit and generates control signals according to the micro-program for the instruction.

12.Interrupt control

There are five hardware interrupts available in 8085 Microprocessor namely TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR for interfacing the peripherals with the microprocessor. These interrupts are handled by the Interrupt control unit. $\overline{INTA}$ signal is generated by the Interrupt control unit as an acknowledgement for an interrupting device. If two or more interrupts occur at the same time, service is given according to the priority basis.

13.Serial I/O control

Serial data is transmitted to the peripherals through SOD pin and received through the SID pin. The SOD and SID pins are handled by the Serial I/O control unit using the SIM and RIM instructions.

14.Address buffer and Address / Data buffer

The Address buffer is an 8-bit unidirectional buffer from which the higher order address bits A_8 – A_{15} leaves the microprocessor to the memory and peripherals. The Address / Data buffer is an 8-bit bidirectional buffer used for sending the lower order address bits A_0 – A_7 and sending and receiving the data bits D_0 – D_7 to the memory and peripherals.

5.2 Instruction set and Addressing modes

5.2.1 Instruction format

The format of 8085 microprocessor instructions is shown in figure.

Opcode	Operand	
	Operand-1	Operand-2
8-bits	8-bits	8-bits

Figure : 5.7 Instruction format of 8085 microprocessor

The instruction has two parts, Opcode and Operand.

Opcode : Represents the operation to be performed on the operand. It is also called mnemonic.
Operand : Data or address is given in this part. If the operand is an 8-bit data, only Operand-1 is present in the instruction. If the operand is a 16-bit data or address, Operand-1 and Operand-2 are specified in the instruction. Both Operand-1 and Operand-2 are optional.

5.2.2 Classification of instructions based on size

There are three groups of instructions in 8085 microprocessor based on the length or size of the instruction. They are,

1. Single byte (or 1 byte) instructions
2. Two byte instructions
3. Three byte instructions

Single byte instructions

Byte 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Opcode

This type of instruction has only Opcode and the operand is specified within the Opcode itself.

Example : 1) MOV B, C ii) ADD B

Two byte instructions

Byte 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Opcode

Byte 2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Operand-1

This type of instruction has Opcode and one operand. The first byte represents the Opcode and the second byte represents the 8-bit operand data or 8-bit port address.

Example : 1) MVI A, 50H ii) OUT 50H

Three byte instructions

Byte 1

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Opcode

Byte 2

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Operand-1

Byte 3

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0

Operand-2

This type of instruction has Opcode and two operands. The first byte represents the Opcode, the second byte presents the lower order 8-bits of data or address and the third byte represents the higher order 8-bits of data or address.

Example : 1) STA 5000H ii) LXI B, 5000H

5.2.3 Classification of instructions based on function

There are 246 instructions (74 types) in the 8085 microprocessor. Based on the function of the instruction, the instructions are classified into the following five types.

1. Data transfer instructions
2. Arithmetic instructions
3. Logic and bit manipulation instructions
4. Branch instructions
5. Machine control instructions

Data transfer instructions

These instructions move (or copy) data from source to destination. The source and destination are registers and memory. Memory to memory transfer is not possible. After the data transfer, the content of the source is not modified and the earlier content of the destination is altered. No flags are affected.

Examples : 1) MOV A, B 2) MOV A, M

Arithmetic instructions

Arithmetic operations like addition, subtraction, increment and decrement are performed by this category of instructions. One of the operand is taken from the Accumulator and the other operand may be from registers or memory. The result of the arithmetic operations is stored in the Accumulator. All the flags are affected.

Examples : 1) ADD B ii) INR A

Logic and bit manipulation instructions

Logical functions like AND, OR and EX-OR are performed by this instructions. All logic functions are performed in relation with the contents of the Accumulator. All the flags are affected.

Examples : 1) ANA B ii) CMA

Branch instructions

Branch instructions change the sequence of the program execution unconditionally or conditionally. The condition of flags is used to take the decision for conditional branches. No flags are affected.

Examples : i) JMP 5000H ii) JNZ 5000H

Machine control instructions

The instructions dealing with interrupt handling and system operations are classified into this category. No flags are affected.

Examples : 1) HLT ii) EI

5.2.4 Instruction set

5.2.4.1 Data Transfer Instructions

S.No	Instruction	Example
1.	**Move - Copy from source to destination** MOV Rd, Rs MOV M, Rs MOV Rd, M	MOV B, C MOV M, A MOV B, M
	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered.	

2.	**Move immediate 8-bit** MVI Rd, data MVI M, data	MVI B, 50H MVI M, 50H
	The 8-bit data is stored in the destination register or memory.	
3.	**Load accumulator direct** LDA 16-bit address	LDA 5000H
	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator.	
4.	**Load accumulator indirect** LDAX Reg. pair	LDAX B LDAX D
	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator.	
5.	**Load register pair immediate** LXI Reg. pair, 16-bit data	LXI B, 5000H LXI D, 5000H LXI H, 5000H LXI SP, 5000H
	The instruction loads 16-bit data in the register pair designated in the operand.	
6.	**Load H and L registers direct** LHLD 16-bit address	LHLD 5000H
	The instruction copies the contents of the memory location pointed by the 16-bit address into register L and copies the contents of the next memory location into register H.	
7.	**Store accumulator direct** STA 16-bit address	STA 5000H
	The contents of the accumulator are copied into the memory location specified by the operand.	
8.	**Store accumulator indirect** STAX Rx	STAX B STAX D
	The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair).	
9.	**Store H and L registers direct** SHLD 16-bit address	SHLD 5000H
	The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location.	
10.	**Exchange H and L with D and E** XCHG	XCHG
	The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.	
11.	**Copy H and L registers to the stack pointer** SPHL	SPHL
	Loads the contents of the H and L registers into the stack pointer register.	

12.	**Exchange H and L with top of stack pointer** XTHL	XTHL
	The contents of the L register are exchanged with the stack location pointed by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1).	
13.	**Push register pair onto stack** PUSH Reg. pair (PSW 'Processor Status Word' means Accumulator and Flag register)	PUSH B PUSH D PUSH H PUSH PSW
	The contents of the register pair designated in the operand are copied onto the stack.	
14.	**Pop off stack to register pair** POP Reg. pair	POP B POP D POP H POP PSW
	The contents of the memory location pointed out by the stack pointer register are copied to registers specified.	
15.	**Output data from accumulator to a port with 8-bit address** OUT 8-bit port address	OUT 50H
	The contents of the accumulator are copied into the I/O port specified by the operand.	
16.	**Input data to accumulator from a port with 8-bit address** IN 8-bit port address	IN 50H
	The contents of the input port designated in the operand are read and loaded into the accumulator.	

5.2.4.2 Arithmetic Instructions

S.No	Instruction	Example
1.	**Add register or memory to accumulator** ADD R ADD M	ADD B ADD M
	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator.	
2.	**Add register to accumulator with carry** ADC R ADC M	ADC B ADC M
	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator.	

3.	**Add immediate to accumulator** ADI 8-bit data	ADI 45H
	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator.	
4.	**Add immediate to accumulator with carry** ACI 8-bit data	ACI 45H
	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator.	
5.	**Add register pair to H and L registers** DAD Reg. pair	DAD H
	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register.	
6.	**Subtract register or memory from accumulator** SUB R SUB M	SUB B SUB M
	The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator.	
7.	**Subtract source and borrow from accumulator** SBB R SBB M	SBB B SBB M
	The contents of the operand (register or memory) and the Borrow (carry flag) are subtracted from the contents of the accumulator and the result is placed in the accumulator.	
8.	**Subtract immediate from accumulator** SUI 8-bit data	SUI 45H
	The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator.	
9.	**Subtract immediate from accumulator with borrow** SBI 8-bit data	SBI 45H
	The 8-bit data (operand) and the Borrow (carry flag) are subtracted from the contents of the accumulator and the result is stored in the accumulator.	
10.	**Increment register or memory by 1** INR R INR M	INR B INR M
	The content of the designated register or memory is incremented by 1 and the result is stored in the same place.	
11.	**Increment register pair by 1** INX Reg. pair	INX B INX D INX H
	The contents of the designated register pair are incremented by 1 and the result is	

	stored in the same place.	
12.	**Decrement register or memory by 1** DCR R DCR M	DCR B DCR M
	The content of the designated register or memory is decremented by 1 and the result is stored in the same place.	
13.	**Decrement register pair by 1** DCX Reg. pair	DCX B DCX D DCX H
	The contents of the designated register pair are decremented by 1 and the result is stored in the same place.	
14.	**Decimal adjust accumulator** DAA	DAA
	The contents of the accumulator are changed from a binary value to two 4-bit binary coded decimal (BCD) digits. If the value of the low-order 4-bits in the accumulator is greater than 9 or if AC flag is set, the instruction adds 6 to the low-order four bits If the value of the high-order 4-bits in the accumulator is greater than 9 or if the Carry flag is set, the instruction adds 6to the high-order four bits.	

5.2.4.3 Logic and bit manipulation instructions

S.No	Instruction	Example
1.	**Compare register or memory with accumulator** CMP R CMP M	CMP B CMP M
	The contents of the operand (register or memory) are compared with the contents of the accumulator. Both contents are preserved. The result of the comparison is shown by setting the flags of the PSW as follows: if (A) < (reg/mem): carry flag is set if (A) = (reg/mem): zero flag is set if (A) > (reg/mem): carry and zero flags are reset	
2.	**Compare immediate with accumulator** CPI 8-bit data	CPI 50H
	The second byte (8-bit data) is compared with the contents of the accumulator. The values being compared remain unchanged. The result of the comparison is shown by setting the flags of the PSW as follows: if (A) < data: carry flag is set if (A) = data: zero flag is set if (A) > data: carry and zero flags are reset	

3.	**Logical AND register or memory with accumulator** ANA R ANA M	ANA B ANA M
	The contents of the accumulator are logically ANDed (bitwise) with the contents of the operand (register or memory), and the result is placed in the accumulator.	
4.	**Logical AND immediate with accumulator** ANI 8-bit data	ANI 50H
	The contents of the accumulator are logically ANDed with the 8-bit data (operand) and the result is placed in the accumulator.	
5.	**Exclusive OR register or memory with accumulator** XRA R XRA M	XRA B XRA M
	The contents of the accumulator are Exclusive ORed with the contents of the operand (register or memory), and the result is placed in the accumulator.	
6.	**Exclusive OR immediate with accumulator** XRI 8-bit data	XRI 50H
	The contents of the accumulator are Exclusive ORed with the 8-bit data (operand) and the result is placed in the accumulator.	
7.	**Logical OR register or memory with accumulator** ORA R ORA M	ORA B ORA M
	The contents of the accumulator are logically ORed with the contents of the operand (register or memory), and the result is placed in the accumulator.	
8.	**Logical OR immediate with accumulator** ORI 8-bit data	ORI 50H
	The contents of the accumulator are logically ORed with the 8-bit data (operand) and the result is placed in the accumulator.	
9.	**Rotate accumulator left** RLC	RLC
	Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag.	
10.	**Rotate accumulator right** RRC	RRC
	Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag.	
11.	**Rotate accumulator left through carry** RAL	RAL

	Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0.	
12.	**Rotate accumulator right through carry** RAR	RAR
	Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7.	
13.	**Complement accumulator** CMA	CMA
	The contents of the accumulator are complemented.	
17.	**Complement carry** CMC	CMC
	The Carry flag is complemented.	
18.	**Set Carry** STC	STC
	The Carry flag is set to 1.	

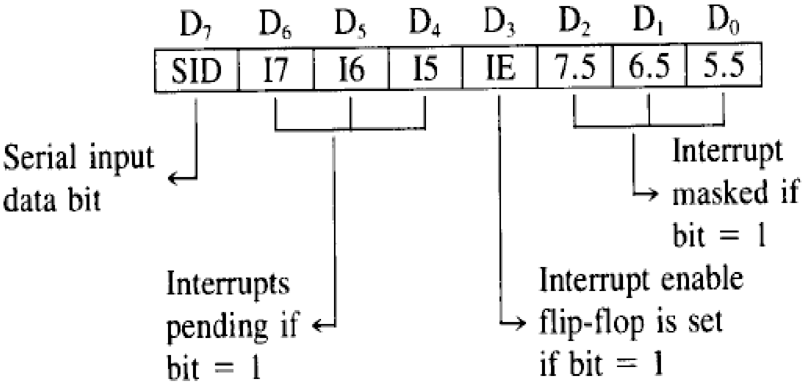
5.2.4.4 Branch instructions

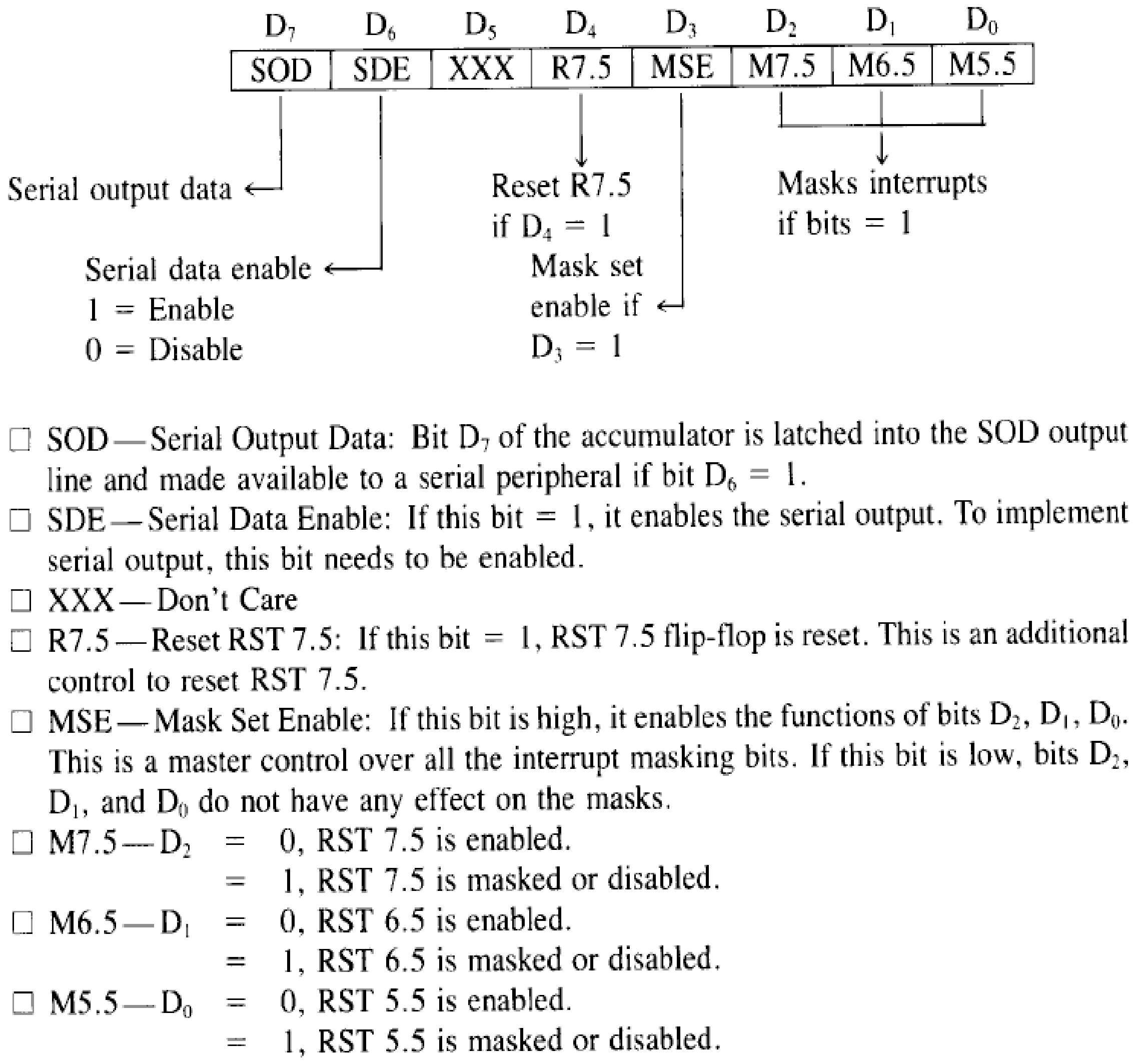
S.No	Instruction	Example
1.	**Jump unconditionally** JMP 16-bit address	JMP 5000H
	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand.	
2.	**Jump conditionally** **J** condition 16-bit address	JC 5000H JNC 5000H JP 5000H JM 5000H JZ 5000H JNZ 5000H JPE 5000H JPO 5000H
	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag. Opcode Description Flag Status *JC Jump on Carry CY = 1* *JNC Jump on No Carry CY = 0* *JP Jump on Positive S = 0* *JM Jump on Minus S = 1* *JZ Jump on Zero Z = 1* *JNZ Jump on No Zero Z = 0* *JPE Jump on Parity Even P = 1* *JPO Jump on Parity Odd P = 0*	

3.	**Unconditional subroutine call** CALL 16-bit address	CALL 5000H
	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand. Before the transfer, the address of the next instruction after CALL (the contents of the program counter) is pushed onto the stack.	
4.	**Call conditionally** **C**condition 16-bit address	CC 5000H CNC 5000H CP 5000H CM 5000H CZ 5000H CNZ 5000H CPE 5000H CPO 5000H
	The program sequence is transferred to the memory location specified by the 16-bit address given in the operand based on the specified flag of the PSW as described below. Before the transfer, the address of the next instruction after the call (the contents of the program counter) is pushed onto the stack. Opcode Description Flag Status *CC* *Call on Carry* *CY = 1* *CNC* *Call on No Carry* *CY = 0* *CP* *Call on Positive* *S = 0* *CM* *Call on Minus* *S = 1* *CZ* *Call on Zero* *Z = 1* *CNZ* *Call on No Zero* *Z = 0* *CPE* *Call on Parity Even* *P = 1* *CPO* *Call on Parity Odd* *P = 0*	
5.	**Return from subroutine unconditionally** RET	RET
	The program sequence is transferred from the subroutine to the calling program. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.	
6.	**Return from subroutine conditionally** **R**condition 16-bit address	RC 5000H RNC 5000H RP 5000H RM 5000H RZ 5000H RNZ 5000H RPE 5000H RPO 5000H

<table>
<tr><td></td><td colspan="2">The program sequence is transferred from the subroutine to the calling program based on the specified flag of the PSW as described below. The two bytes from the top of the stack are copied into the program counter, and program execution begins at the new address.
Opcode | Description | Flag Status
<i>RC</i> | <i>Return on Carry</i> | <i>CY = 1</i>
<i>RNC</i> | <i>Return on No Carry</i> | <i>CY = 0</i>
<i>RP</i> | <i>Return on Positive</i> | <i>S = 0</i>
<i>RM</i> | <i>Return on Minus</i> | <i>S = 1</i>
<i>RZ</i> | <i>Return on Zero</i> | <i>Z = 1</i>
<i>RNZ</i> | <i>Return on No Zero</i> | <i>Z = 0</i>
<i>RPE</i> | <i>Return on Parity Even</i> | <i>P = 1</i>
<i>RPO</i> | <i>Return on Parity Odd</i> | <i>P = 0</i></td></tr>
<tr><td rowspan="2">7.</td><td><b>Load program counter with HL contents</b>
PCHL</td><td>PCHL</td></tr>
<tr><td colspan="2">The contents of registers H and L are copied into the program counter. The contents of H are placed as the high-order byte and the contents of L as the low-order byte.</td></tr>
<tr><td rowspan="2">8.</td><td><b>Restart</b>
RST 0-7</td><td>RST 0
RS T1
RST 2
RST 3
RST 4
RST 5
RST 6
RST 7</td></tr>
<tr><td colspan="2">The RST instruction is equivalent to a 1-byte call instruction to one of eight memory locations depending upon the number. The addresses are:
Instruction | Restart Address
RST 0 | 0000H
RST 1 | 0008H
RST 2 | 0010H
RST 3 | 0018H
RST 4 | 0020H
RST 5 | 0028H
RST 6 | 0030H
RST 7 | 0038H</td></tr>
</table>

5.2.4.5 Machine control instructions

S.No	Instruction	Example
1.	**No operation** NOP	NOP
	No operation is performed. The instruction is fetched and decoded. However no operation is executed.	
2.	**Halt and enter wait state** HLT	HLT
	The CPU finishes executing the current instruction and halts any further execution. An interrupt or reset is necessary to exit from the halt state.	
3.	**Disable interrupts** DI	DI
	The interrupt enable flip-flop is reset and all the interrupts except the TRAP are disabled.	
4.	**Enable interrupt** EI	EI
	The interrupt enable flip-flop is set and all interrupts are enabled. No flags are affected.	
5.	**Read interrupt mask** RIM	RIM
	This is a multipurpose instruction used to read the status of interrupts 7.5, 6.5, 5.5 and read serial data input bit. The instruction loads eight bits in the accumulator with the following interpretations.  D7 D6 D5 D4 D3 D2 D1 D0 SID I7 I6 I5 IE 7.5 6.5 5.5 Serial input data bit Interrupts pending if bit = 1 Interrupt enable flip-flop is set if bit = 1 Interrupt masked if bit = 1	
7.	**Set interrupt mask** SIM	SIM
	This is a multipurpose instruction and used to implement the8085 interrupts 7.5, 6.5, 5.5, and serial data output. The instruction interprets the accumulator contents as follows.	



D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SOD	SDE	XXX	R7.5	MSE	M7.5	M6.5	M5.5

Serial output data

Serial data enable
1 = Enable
0 = Disable

Reset R7.5
if D_4 = 1

Mask set
enable if
D_3 = 1

Masks interrupts
if bits = 1

□ SOD—Serial Output Data: Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D_6 = 1.

□ SDE—Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.

□ XXX—Don't Care

□ R7.5—Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.

□ MSE—Mask Set Enable: If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.

□ M7.5—D_2 = 0, RST 7.5 is enabled.
= 1, RST 7.5 is masked or disabled.

□ M6.5—D_1 = 0, RST 6.5 is enabled.
= 1, RST 6.5 is masked or disabled.

□ M5.5—D_0 = 0, RST 5.5 is enabled.
= 1, RST 5.5 is masked or disabled.

5.2.5 Addressing modes

The method of specifying the location of operand in an instruction is called addressing mode. There are five types of addressing modes in 8085 microprocessor.

1. Direct addressing mode
2. Immediate addressing mode
3. Register addressing mode
4. Register indirect addressing mode
5. Implicit (or) implied addressing mode

Direct addressing mode

In direct addressing mode, **the address of the operand is directly specified in the instruction**. In this addressing mode, the instruction is two or three bytes long. The first byte is the Opcode. The operand may be a 16-bit (2 bytes) memory address or an 8-bit (1 byte) port address.
Examples:

S. No	Instruction	Remarks
1.	STA 5000	This instruction stores the content of the accumulator in memory location 5000. Here, the memory address is given directly in the instruction.
2.	LDA 5000	This instruction loads the data from memory location 5000 accumulator. Here, the memory address is given directly in the instruction.
3.	IN 80	This instruction loads the data from input port 80. Here, the port address is directly given in the instruction.

Immediate addressing mode

In immediate addressing mode, **the operand itself is immediately given after the Opcode**. The instruction is two or three bytes long. The first byte is the Opcode. The operand may be a 16-bit (2 bytes) immediate data or an 8-bit (1 byte) immediate data.

Examples:

S. No	Instruction	Remarks
1.	MVI A, 50	This instruction immediately moves the data 50 into the accumulator. Here, the data is given immediately after the Opcode.
2.	LXI B, 2050	This instruction immediately moves the data 2050 into the register pair BC. 20 to B register and 50 to C register. Here, the data is given immediately after the Opcode.

Register addressing mode

In register addressing mode, a **register is specified as the operand** in the instruction. The instruction is one byte long. The register name is specified in the Opcode itself.

Examples:

S. No	Instruction	Remarks
1.	ADD B	This instruction adds the content of B register with the accumulator. Here, the data is in the register B.
2.	MOV C, D	This instruction moves the D register value to C register. Here, C and D registers are specified as the operands.

Register indirect addressing mode

In register indirect addressing mode, **the content of the register pair is used as the address of the operand** in the instruction. The instruction is one byte long. The register pair contains the 16-bit address of the memory location where the actual operand is stored.

Examples:

S. No	Instruction	Remarks
1.	STAX B	This instruction stores the accumulator value in memory location whose address is specified by the BC register pair. Here, the address is indirectly specified in the register pair.
2.	MOV A, M	This instruction moves the data from memory to accumulator. M means memory whose address is specified in HL register pair. Here, address of the operand is indirectly specified in the HL register pair.

Implicit or Implied addressing mode

In implied addressing mode, **a particular register is implicitly specified as the operand** in the instruction. The instruction is one byte long. This addressing mode is also known as implied addressing mode and inherent addressing mode.

Examples:

S. No	Instruction	Remarks
1.	CMA	This instruction complements the contents of the accumulator. Here, Accumulator is implicitly specified in the instruction.
2.	RLC	This instruction rotates the contents of the accumulator left one time. Here, Accumulator is implicitly specified in the instruction.

5.3 Machine cycle and Instruction cycle

5.3.1 Machine cycle

Machine cycle is defined as the time required for completing one operation of accessing memory, I/O or acknowledging an external request. Machine cycle is comprised of T-states. T-state is defined as one subdivision of the operation performed in one clock period. The following are the various machine cycles of 8085 microprocessor.

1. Opcode Fetch (OF)
2. Memory Read (MR)
3. Memory Write (MW)
4. I/O Read (IOR)
5. I/O Write (IOW)
6. Interrupt Acknowledge (IA)
7. Bus Idle (BI)

All instructions have at least one Opcode Fetch machine cycle. Depending on the type of instruction one or more other machine cycles are required to complete the execution of the instruction. The number and type of machine cycles for different instructions are shown in table.

S.No	Instruction	Number of machine cycles	Machine cycle – 1	Machine cycle - 2	Machine cycle - 3	Machine cycle - 4
1.	MOV A, B	1	OF	-	-	-
2.	MVI A, 50H	2	OF	MR	-	-
3.	LDA 5000H	4	OF	MR	MR	MR
4.	STA 5000H	4	OF	MR	MR	MW
5.	IN 80H	3	OF	MR	IOR	-
6.	OUT 80H	3	OF	MR	IOW	-

5.3.1.1 Opcode Fetch (OF) machine cycle of 8085

Each instruction of the microprocessor has one byte Opcode. The Opcode is stored in memory. So, the processor executes the Opcode Fetch machine cycle to fetch the Opcode from memory. Hence, every instruction starts with Opcode Fetch machine cycle. The time taken by the microprocessor to execute the Opcode Fetch cycle is 4T (T- states). The first 3 T-states are used for fetching the Opcode from memory and the remaining T-state is used for internal operations by the microprocessor.

The timing diagram for Opcode Fetch machine cycle is shown in figure 5.10.

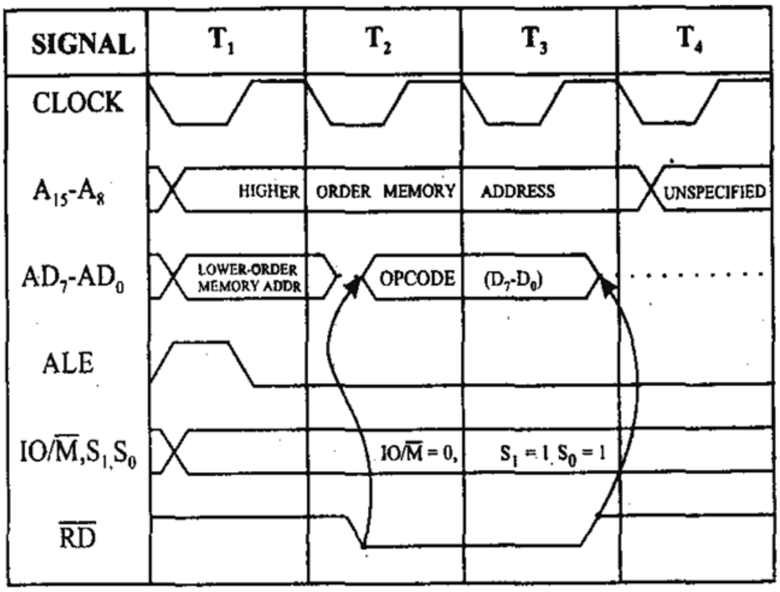


Figure 5.10 : Timing diagram of Opcode Fetch machine cycle

The steps in Opcode Fetch machine cycle are given in table.

S. No	T state	Operation
1.	T_1	The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus.
2		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3.		The status signals are changed as IO/$\overline{M}$ = 0, S1 =1 and S0 = 1. These status signals do not change throughout the OF machine cycle.
4.	T_2	The microprocessor makes the $\overline{RD}$ line LOW to enable memory read and increments the Program Counter.
5.		The contents on D7 – D0 (i.e. the Opcode) are placed on the address / data bus.
6.	T_3	The microprocessor transfers the Opcode on the address / data bus to Instruction Register (IR).
7.		The microprocessor makes the $\overline{RD}$ line HIGH to disable memory read.
8.	T_4	The microprocessor decodes the instruction.

5.3.1.2Memory Read Machine Cycle of 8085

Single byte instructions require only Opcode Fetch machine cycles. But, 2-byte and 3-byte instructions require additional machine cycles to read the operands from memory. The additional machine cycle is called Memory Read machine cycle. For example, the instruction MVI A, 50H requires one OF machine cycle to fetch the operand from memory and one MR machine cycle to read the operand (50H) from memory. The MR machine cycle takes 3 T-states.

The timing diagram for Memory Read machine cycle is shown in figure 5.11.

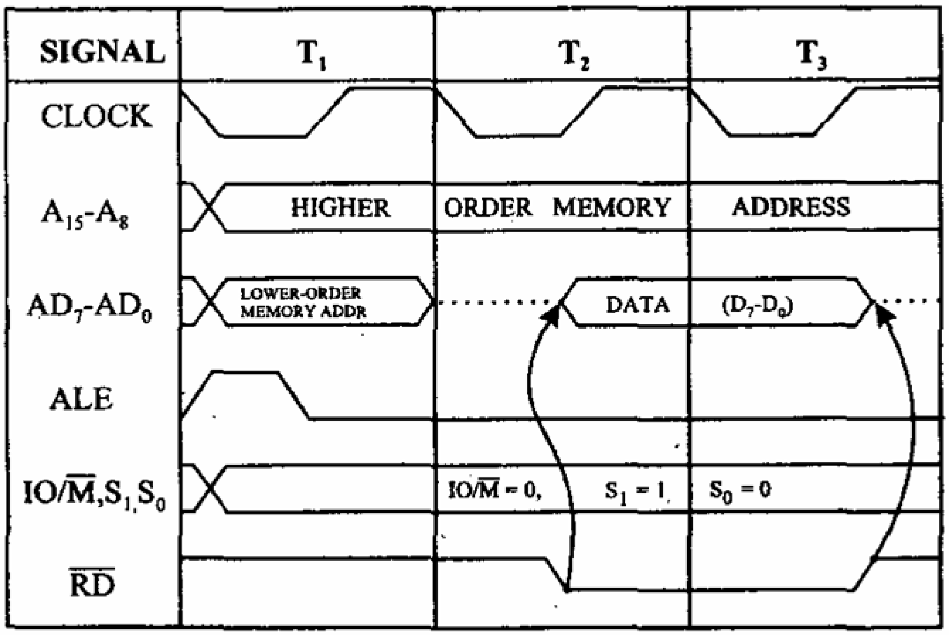


Figure 5.11 : Timing Diagram for Memory Read Machine Cycle

The steps in Memory Read machine cycle are given in table.

S.No	T state	Operation
1.	T_1	The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus.
2.		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3.		The status signals are changed as IO/$\overline{M}$ = 0, S1 =1 and S0 = 0. These status signals do not change throughout the memory read machine cycle.

4.	T_2	The microprocessor makes the $\overline{RD}$ line LOW to enable memory read and increments the Program Counter.
5.		The contents on D7 – D0 (i.e. the data) are placed on the address / data bus.
6.	T_3	The data loaded on the address / data bus is moved to the microprocessor.
7.		The microprocessor makes the $\overline{RD}$ line HIGH to disable the memory read operation.

5.3.1.3Memory Write Machine Cycle of 8085

Microprocessor uses the Memory Write machine cycle for sending the data in one of the registers to memory. For example, the instruction STA 5000H writes the data in accumulator to the memory location 5000H. The MW machine cycle takes 3 T-states.

The timing diagram for Memory Write machine cycle is shown in figure 5.12.

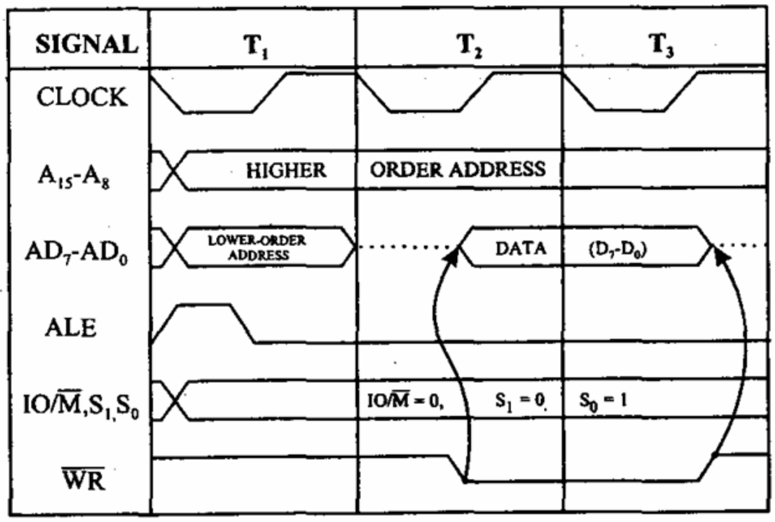


Figure 5.12 : Timing Diagram for Memory Write Machine Cycle

The steps in Memory Write machine cycle are given in table.

S.No	T State	Operation
1.	T_1	The microprocessor places the higher order 8-bits of the memory address on A15 – A8 address bus and the lower order 8-bits of the memory address on AD7 – AD0 address / data bus.
2.		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.

3.		The status signals are changed as IO/$\overline{M}$ = 0, S1 =0 and S0 = 1. These status signals do not change throughout the memory write machine cycle.
4.	T_2	The microprocessor makes the $\overline{WR}$ line LOW to enable memory write.
5.		The contents of the specified register are placed on the address / data bus.
6.	T_3	The data placed on the address / data bus is transferred to the specified memory location.
7.		The microprocessor makes the $\overline{WR}$ line HIGH to disable the memory write operation.

5.3.1.4I/O Read Machine Cycle of 8085

Microprocessor uses the I/O Read machine cycle for receiving a data byte from the I/O port or from the peripheral in I/O mapped I/O systems. The IN instruction uses this machine cycle during execution. The IOR machine cycle takes 3 T-states.

The timing diagram for I/O Read machine cycle is shown in figure 5.13.

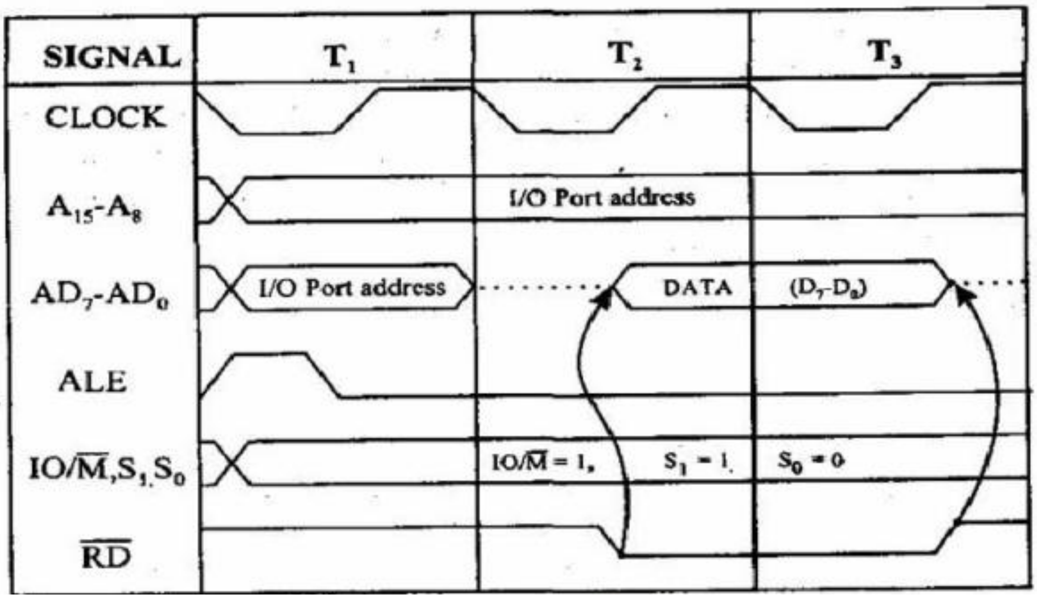


Figure 5.13 Timing Diagram for I/O Read Machine Cycle

The steps in I/O Read machine cycle are given in table.

S.No	T State	Operation
1.	T_1	The microprocessor places the address of the I/O port specified in the instruction on A15 – A8 address bus and also on AD7 – AD0 address / data bus.

2.		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3.		The status signals are changed as IO/$\overline{M}$ = 0, S1 =1 and S0 = 0. These status signals do not change throughout the I/O read machine cycle.
4.	T_2	The microprocessor makes the $\overline{RD}$ line LOW to enable I/O read.
5.		The contents on D7 – D0 (i.e. the data) are placed on the address / data bus.
6.	T_3	The data loaded on the address / data bus is moved to the microprocessor ie., to the accumulator.
7.		The microprocessor makes the $\overline{RD}$ line HIGH to disable the I/O read operation.

5.3.1.5 I/O Write Machine Cycle of 8085

Microprocessor uses the I/O Write machine cycle for sending a data byte to the I/O port or to the peripheral in I/O mapped I/O systems. The OUT instruction uses this machine cycle during execution. The IOR machine cycle takes 3 T-states.

The timing diagram for I/O Write machine cycle is shown in figure 5.14.

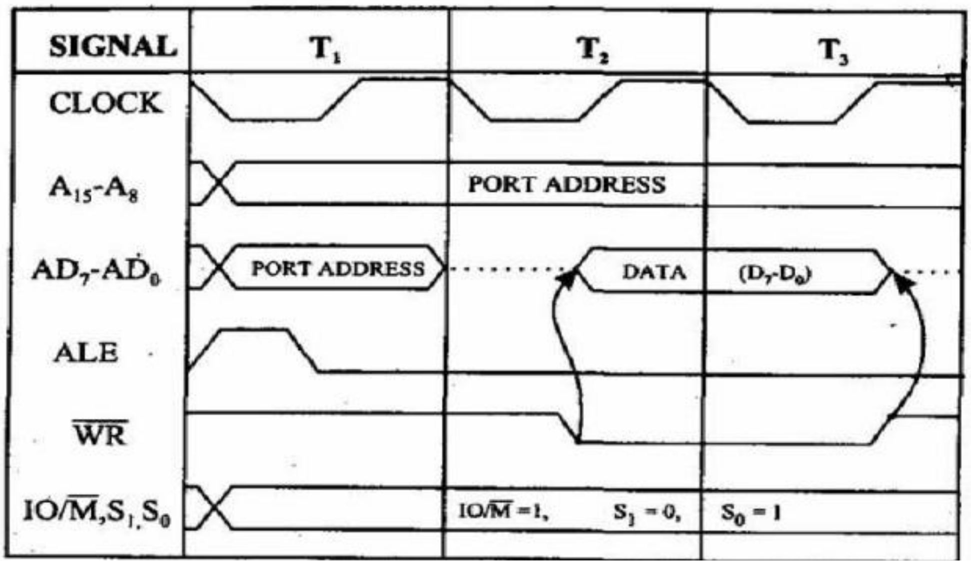


Figure 5.14 : Timing Diagram for I/O Write Machine Cycle

The steps in I/O Read machine cycle are given in table.

S.No	T State	Operation
1.	T_1	The microprocessor places the address of the I/O port specified in the instruction on A15 – A8 address bus and also on AD7 – AD0 address / data bus.
2.		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3.		The status signals are changed as IO/$\overline{M}$ = 0, S1 =0 and S0 = 1. These status signals do not change throughout the I/O write machine cycle.
4.	T_2	The microprocessor makes the $\overline{WR}$ line LOW to enable I/O write.
5.		The contents of the Accumulator are placed on the address / data bus.
6.	T_3	The data placed on the address / data bus is transferred to the specified I/O port.
7.		The microprocessor makes the $\overline{WR}$ line HIGH to disable the I/O write operation.

5.3.2 Instruction cycle

Timing diagram for MOV Rd, Rs (or MOV r1, r2) instruction

MOV Rd, Rs instruction moves (copies) the contents of the source register (Rs) into the destination register (Rd). It is a single byte instruction. It has only Opcode Fetch machine cycle.
Some examples for MOV Rd, Rs instruction:

1. MOV A, B
2. MOV C, L

The time taken by the processor to execute the Opcode Fetch cycle is 4T (T-states). The first 3 T-states are used for fetching the Opcode from memory and the remaining T-state is used for internal operations by the microprocessor. The timing diagram for MOV Rd, Rs (Opcode Fetch machine cycle) is shown in figure 5.14. It has 4 T states.

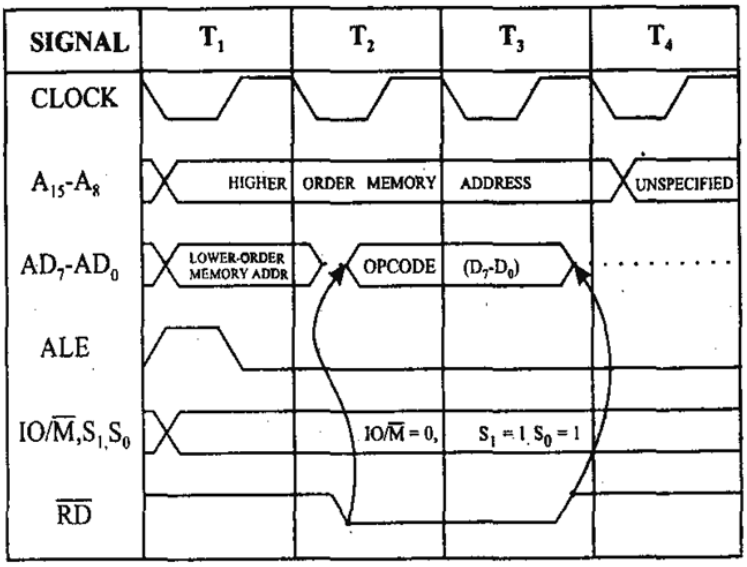


Figure 5.14 Timing Diagram for MOV Rd, Rs instruction
(Opcode Fetch Machine Cycle)

The steps for machine cycle of MOV Rd, Rs instruction are given in table.

S.No	T state	Operation
1.	T1	The microprocessor places the higher order 8-bits of the Program Counter on A15 – A8 address bus and the lower order 8-bits of the Program Counter on AD7 – AD0 address / data bus.
2.		The microprocessor makes the ALE signal HIGH and at the middle of T1 state, ALE signal goes LOW.
3.		The status signals are changed as IO/$\overline{M}$ = 0, S1 =1 and S0 = 1. These status signals do not change throughout the OF machine cycle.
4.	T2	The microprocessor makes the $\overline{RD}$ line LOW to enable memory read (opcode fetch) and increments the Program Counter.
5.		The contents on D7 – D0 (i.e. the Opcode) are placed on the address / data bus.
6.	T3	The microprocessor transfers the Opcode on the address / data bus to Instruction Register (IR).
7.		The microprocessor decodes the instruction.
8.	T4	The data in the register Rs (r_2) is moved to the register Rd (r_1).

5.4 I/O Mapping and Interrupts

5.4.1 I/O Mapping schemes

I/O interfacing

There are two methods of interfacing the Input / Output devices with the microprocessor. They are,1) Memory mapped I/O and 2) I/O mapped I/O.

5.4.1.1 Memory mapped I/O

In this method the I/O devices are treated like the memory. A part of the memory address space is used for the I/O devices. The memory mapped I/O scheme is shown in figure 5.8.

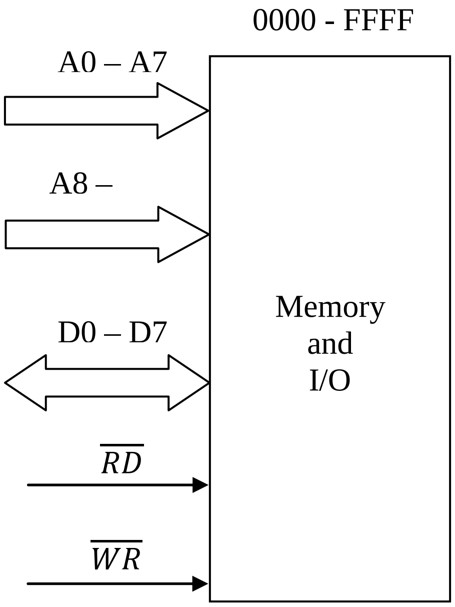


Figure 5.8 : Memory mapped I/O scheme

- In memory mapped I/O scheme, the same address space is used for both memory and I/O devices.
- The microprocessor uses the sixteen address line A0 – A7 and A8 – A15 for the memory as well as for the I/O devices.
- The I/O devices share the address space with the memory. All the memory related instructions are used for addressing I/O devices also.
- No separate IN and OUT instructions are required in memory mapped I/O scheme.
- IO/$\overline{M}$ pin is not required.

Steps for memory operations (memory read and memory write) :

1. When the memory related instructions like LDA and STA are used, the microprocessor places the 16-bit address on the address bus.
2. $\overline{RD}$ is activated for read operation and $\overline{WR}$ is activated for write operation.

Steps for I/O operations (I/O read and I/O write) :

The same steps used for memory operations are used for I/O operations also.

5.4.1.2 I/O mapped I/O

In this method, I/O devices are treated as I/O devices and memory is treated as memory. Separate address space is used for memory and I/O. The I/O mapped I/O scheme is shown in figure

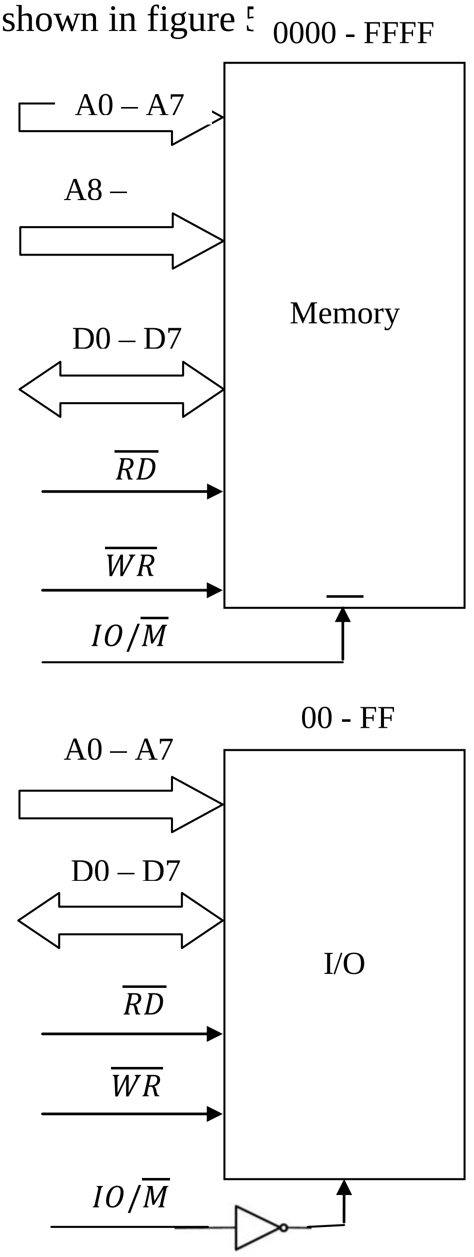


Figure 5.9 : I/O mapped I/O scheme

- In I/O mapped I/O scheme, the microprocessor uses the sixteen address lines A_0 – A_7 and A_8 – A_{15} for the memory and eight address lines A_0 to $A_7$to identify an input / output device.
- Here, the full address space 0000 – FFFF is used for the memory and a separate address space 00 – FF is used for the I/O devices.
- Hence, the microprocessor can address 65536 (2^{16}) memory locations 256 (2^8) input devices and 256 (2^8) output devices separately.
- IN and OUT instructions are used to activate the IO/$\overline{M}$ signal.
- When IO/$\overline{M}$ is low, the memory is selected for reading and writing operations.
- When IO/$\overline{M}$ is high, the I/O port is selected for reading and writing operations.

Steps for memory operations (memory read and memory write) :

1. When the memory related instructions like LDA and STA are used, the microprocessor places the 16-bit address on the address bus.
2. The microprocessor makes the IO/$\overline{M}$ linelow.
3. The microprocessor makes the $\overline{RD}$low for read operation and $\overline{WR}$low for write operation.

Steps for I/O operations (I/O read and I/O write) :

1. When the I/O related instructions like IN and OUT are used, the microprocessor places the 8-bit address on the address bus A_0 – A_7 as well as A_8 – A_{15}.
2. IO/$\overline{M}$ line is made high.
3. The microprocessor makes the $\overline{RD}$ low for read operation and $\overline{WR}$ low for write operation.

5.4.1.3 Differences between Memory mapped I/O ad I/O mapped I/O

S.No.	Memory mapped I/O	I/O mapped I/O
1.	16-bit device address.	8-bit device address.
2.	Data is transferred between any general-purpose register and I/O port.	Data is transferred only between accumulator and I/O port.
3.	The memory map (64K) is shared between I/O device and system memory.	The I/O map is independent of the memory map; 256 input devices and 256 output devices can be connected.
4.	More hardware is required to decode 16- bit address.	Less hardware is required to decode 8-bit address.

5.	Arithmetic or logic operation can be directly performed with I/O data.	Arithmetic or logical operation cannot be directly performed with I/O data.
6.	$IO/\overline{M}$ pin is not required.	$IO/\overline{M}$ pin is required.
7.	Instructions like LDA, STA, MOV R,M and ADD M are used.	IN and OUT instructions are used.

5.4.2 Interrupts

Interrupts are the signals send by an external device to the microprocessor to request the microprocessor to perform a particular task or work. Interrupts are used for data transfer between the peripheral and the microprocessor. The microprocessor will check the interrupts always at the 2nd T-state of last machine cycle. If there is any interrupt, it accepts the interrupt and sends the $\overline{INTA}$ signal to the peripheral. The microprocessor executes an interrupt service routine (ISR) stored in memory. It returns to the main program by RET instruction, after the ISR is executed. The interrupt process is shown in figure 5.15.

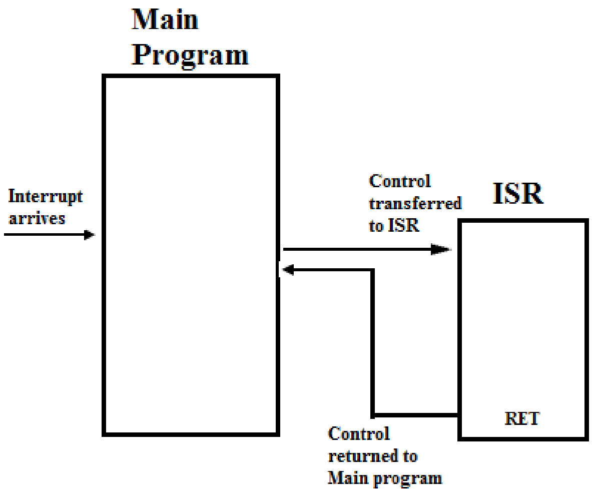


Figure 5.15 : Interrupt process

5.4.2.1 Types of interrupts

There are six types of interrupts. They are,

1. Hardware interrupts
2. Software interrupts
3. Maskable interrupts
4. Non-maskable interrupts
5. Vectored interrupts
6. Non-vectored interrupts

Hardware interrupts : These interrupts are given by the peripheral devices to the interrupt pin (hardware) of the microprocessor. Hardware interrupts are also called external interrupts.

Software interrupts : These interrupts are internally generated within the microprocessor using software instructions. Software interrupts are also called internal interrupts.

Maskable interrupts : These external interrupts can be delayed or rejected by the microprocessor.

Non-maskable interrupts : These external interrupts cannot be delayed or rejected by the microprocessor. Non-maskable interrupts are used for handling emergency situations.

Vectored interrupts : When the address of the Interrupt Service Routine (ISR) is fixed within the microprocessor itself, then the interrupt is called Vectored interrupt.

Non-vectored interrupts : When the address of the Interrupt Service Routine (ISR) is supplied by the peripheral device, then the interrupt is called Non-vectored interrupt.

5.4.2.2 8085 interrupts

In 8085 microprocessor, there are 5 interrupts as shown in figure 5.16.

1. TRAP
2. RST 5.5
3. RST 6.5
4. RST 7.5
5. INTR

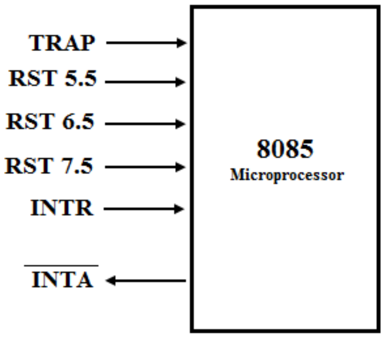


Figure 5.16 : 8085 Interrupts

In additional to these hardware interrupts, 8085 microprocessor has eight software interrupts. The RESTART instructions RST 0 to RST 7 are software interrupt instructions.

5.4.2.3 Interrupt priority

The microprocessor can respond to only one interrupt at one time. When multiple (more than one) interrupts occur simultaneously, the microprocessor will service the interrupts in their fixed priority order. Interrupt having the highest priority level will be serviced first. In 8085, TRAP interrupt has the highest priority and INTR has the lowest priority.

TRAP

- This interrupt is a non-maskable interrupt. It is unaffected by any mask or interrupt enable.
- It is a vectored interrupt. The interrupt vector address is 0024H.
- TRAP has the highest priority level.
- TRAP interrupt is edge and level triggered. This means that the TRAP must go high and remain high until it is acknowledged.
- In emergency situations like sudden power failure, it executes an ISR and sends the data from main memory to backup memory.

RST 7.5

- The RST 7.5 interrupt is a maskable interrupt.
- It is a vectored interrupt. The interrupt vector address is 003CH.
- It has the second highest priority.
- It is edge triggered. ie. Input goes to high and no need to maintain high state until it is recognized and acknowledged.

RST 6.5

- The RST 6.5 interrupt is a maskable interrupt.
- It is a vectored interrupt. The interrupt vector address is 0034H.
- It has the third highest priority.
- It is level triggered. ie. Input goes to high and stays high until it is recognized and acknowledged.

RST 5.5

- The RST 5.5 interrupt is a maskable interrupt.
- It is a vectored interrupt. The interrupt vector address is 002CH.
- It has the fourth highest priority.
- It is level triggered. ie. Input goes to high and stays high until it is recognized and acknowledged.

INTR

- INTR is a maskable interrupt.
- It is a non- vectored interrupt. After receiving $\overline{INTA}$, the peripheral has to supply the address of ISR.

- It has the lowest priority.
- It is a level triggered. ie. Input goes to high and it is necessary to maintain high state until it is recognized and acknowledged.

Process of INTR interrupt

1. The interrupt process should be enabled using the EI instruction.
2. The 8085 checks for an interrupt during the execution of every instruction.
3. If INTR is high, the microprocessor completes current instruction, disables the interrupt and sends $\overline{INTA}$ signal to the peripheral device.
4. $\overline{INTA}$ allows the peripheral device to send an RST instruction through data bus.
5. Upon receiving the $\overline{INTA}$ signal, the microprocessor saves the memory location of the next instruction on the stack and the program is transferred to 'call' location (ISR Call) specified by the RST instruction.
6. Microprocessor executes the ISR.
7. ISR must include the 'EI' instruction to enable the further interrupt within the program.
8. The RET instruction at the end of the ISR retrieves the return address from the stack and the program is transferred back to main program which was interrupted.

5.4.2.4 Instructions for Interrupts handling in 8085 microprocessor

There are four instructions available for interrupts handling. They are,

1. DI (Disable Interrupt)
2. EI (Enable Interrupt)
3. SIM (Set Interrupt Mask)
4. RIM (Read Interrupt Mask)

DI (Disable Interrupt)

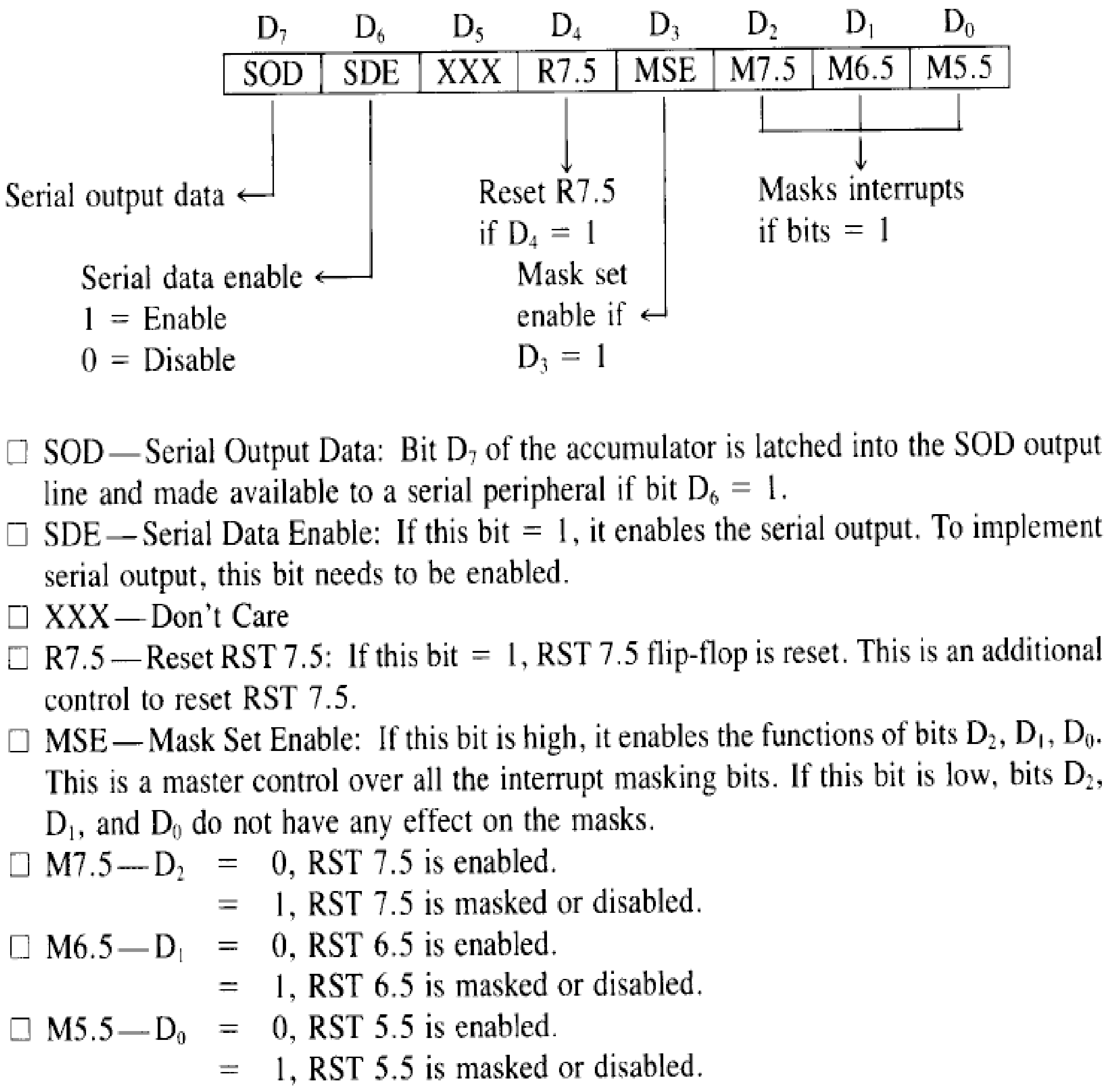
This instruction resets the Interrupt Enable Flip-flop inside the microprocessor. All the interrupts except the TRAP are disabled.

EI (Enable Interrupt)

This instruction sets the Interrupt Enable Flip-flop inside the microprocessor. All the interrupts are enabled.

SIM (Set Interrupt Mask)

This instruction is used to selectively mask (disable) and unmask (enable) RST 7.5, RST 6.5 and RST 5.5 interrupts. This instruction is also used for serial data output. The SIM the instruction uses the accumulator contents for masking and unmasking the interrupts.



- □ SOD—Serial Output Data: Bit D_7 of the accumulator is latched into the SOD output line and made available to a serial peripheral if bit D_6 = 1.
- □ SDE—Serial Data Enable: If this bit = 1, it enables the serial output. To implement serial output, this bit needs to be enabled.
- □ XXX—Don't Care
- □ R7.5—Reset RST 7.5: If this bit = 1, RST 7.5 flip-flop is reset. This is an additional control to reset RST 7.5.
- □ MSE—Mask Set Enable: If this bit is high, it enables the functions of bits D_2, D_1, D_0. This is a master control over all the interrupt masking bits. If this bit is low, bits D_2, D_1, and D_0 do not have any effect on the masks.
- □ M7.5—D_2 = 0, RST 7.5 is enabled.
  = 1, RST 7.5 is masked or disabled.
- □ M6.5—D_1 = 0, RST 6.5 is enabled.
  = 1, RST 6.5 is masked or disabled.
- □ M5.5—D_0 = 0, RST 5.5 is enabled.
  = 1, RST 5.5 is masked or disabled.

Figure 5.17 : Bits used in SIM instruction

Example :

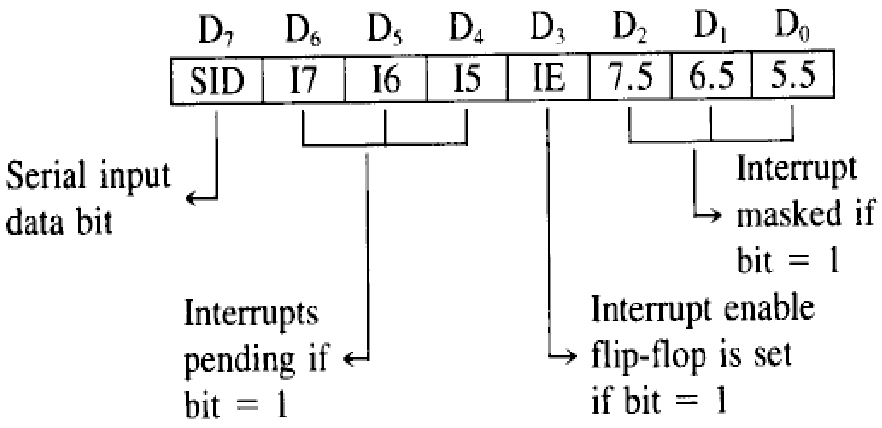
The following instructions are used to enable interrupt RST 5.5 and disable RST 7.5 and RST 6.5 :

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	1	1	1	0

```
MVI A, 0EH          ; Bits D3 = 1 and D0 = 0
SIM                 ; Enable RST 5.5
```

RIM (Read Interrupt Mask)

This instruction is used to read the status of RST 7.5, RST 6.5 and RST 5.5 interrupts like pending and enable / disable details. This instruction is also used for reading the serial data. When the RIM instruction is given, the microprocessor loads the details into the accumulator.



D7 D6 D5 D4 D3 D2 D1 D0
SID I7 I6 I5 IE 7.5 6.5 5.5
Serial input data bit
Interrupts pending if bit = 1
Interrupt enable flip-flop is set if bit = 1
Interrupt masked if bit = 1

Figure 5.18 : Bits used in RIM instruction

Bits D6, D5 and D4 give the pending details of RST 7.5, RST 6.5 and RST 5.5 interrupts respectively. Bits D2, D1 and D0 give the masked / unmasked details of RST 7.5, RST 6.5 and RST 5.5 interrupts respectively. Bit D3 gives the status of Interrupt enable Flip-flop.

Example :

Suppose after the execution of RIM instruction, the accumulator has 49H. The meanings are:

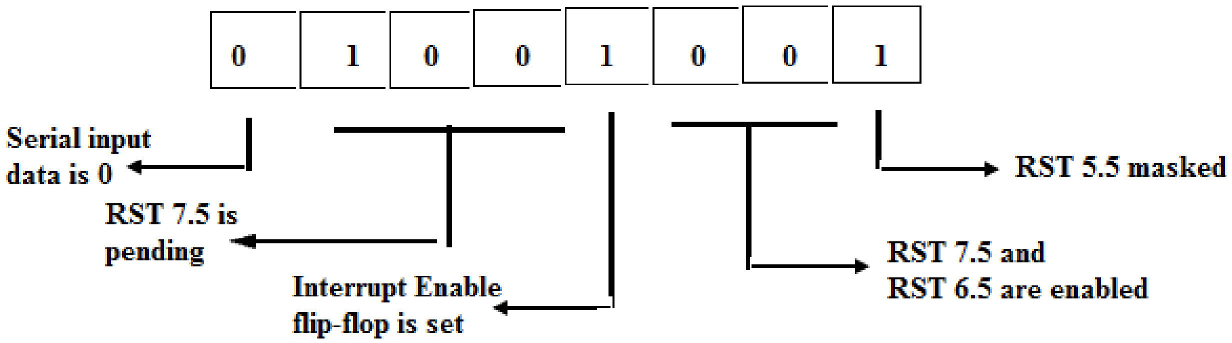


Figure 5.19 : Example for RIM instruction

5.4.2.5 Summary of 8085 interrupts

Interrupt	Vector address	Priority	Type
TRAP	0024_H	1 (Highest priority)	Hardware interrupt Vectored interrupt Non-maskable interrupt
RST7.5	$003C_H$	2	Hardware interrupt Vectored interrupt Maskable interrupt
RST6.5	0034_H	3	Hardware interrupt Vectored interrupt Maskable interrupt
RST5.5	$002C_H$	4	Hardware interrupt Vectored interrupt Maskable interrupt
INTR	--	5 (Lowest priority)	Hardware interrupt Non-vectored interrupt Maskable interrupt
RST instruction	RST 0 - 0000_H RST 1 - 0008_H RST 2 - 0010_H RST 3 - 0018_H RST 4 - 0020_H RST 5 - 0028_H RST 6 - 0030_H RST 7 - 0038_H	--	Software interrupt Vectored interrupt Maskable interrupt

☺☺☺☺☺☺☺