KARAPAGAM ACADEMY OF HIGHER EDUCATION, COIMBATORE-21

Semester–VI

L T P C

15PHU601 DIGITAL ELECTRONICS AND MICROPROCESSOR 5 - - 5

Scope: Digital electronics has a lot of applications in daily life as most of the electonics instruments which we use are digital. Some basic aspects of digital electronics are explained in this paper.

Objectives: Digital electronics is the heart of computers. In the modern world, everything is digital. So, it is important to give the students an idea of electronics.

UNIT - I

Decimal, binary, octal, hexadecimal - Conversion of number system - Conversion of decimal to binary, binary to decimal- decimal to octal - Octal to decimal - Octal to binary - Binary to octal - Decimal to hexadecimal - Hexadecimal to decimal, hexadecimal to binary - Binary to hexadecimal.

Binary coded decimal - 8421 code - Alphanumeric codes ASCII code - EBCDIC code - Error detecting code – Parity - Even parity and odd parity method.

UNIT - II

Logic gates - AND, OR, NOT, NAND, NOR gates - Construction of circuit only I/C - Action truth table - Logic symbol.

Boolean operators - Logic expressions – Demorgan's theorems - Laws and rules of Boolean algebra - Truth table - Reducing Boolean expressions, K maps; logic diagrams of Boolean algebra expressions - Converting logic circuits to expressions.

UNIT - III

Flip-flop definitions; clocked flip-flop; S-R flip-flop: JK flip flop: T-flip flop; D flip flop; master slave J-K flip flop: construction circuits.

Ring counter; Ripple counter and mod counters

UNIT - IV

XOR gates half adder - Full adder - Full subtracter - Parallel binary adder - Parallel binary subtracter - Construction, action and truth table.

Magnetic tape - Magnetic disc - Floppy disc - Magnetic cores - Magnetic core logic, coincident memory - Memory addressing. MOS - random access memory - MOD read only memory PROM-EPROM.

UNIT - V

Brief history, organization of 8085 - Data and address bus, addressing the I/O devices, registers in the 8085, instruction set - Instruction types, and classification of instruction, simple programs.

Text Book

1. Floyd, 2003, Digital Fundamentals, 8th Edition, Pearson education, New Delhi.

2. Ramesh Gaonkar 6th edition 2013 Microprocessor Architecture, Programming and Applications with 8085 ,PENRAM International P Ltd

REFERENCES

1. Malvino and Leach, 1983, Digital Principles and Applications, 3rd Edition, Tata McGrawHill, New Delhi.
2. Aditya P. Mathur, 1995, Introduction to Microprocessor, 3rd Edition, Tata McGrawHill, New Delhi.
3. Morris Mano. M, 1st2002, Digital Logic and Computer Design, Prentice Hall, New Delhi.

KARPAGAM ACADEMY OF HIGHER EDUCATION

(Deemed to be University Established Under Section 3 of UGC Act 1956)

Coimbatore – 641 021.

LECTURE PLAN
DEPARTMENT OF PHYSICS

STAFF NAME: Mrs. A. SAHANA FATHIMA
SUBJECT NAME: DIGITAL ELECTRONICS & MICROPROCESSOR SUB.CODE: 15PHU601
SEMESTER: VI CLASS: III B.Sc (PHY)

S.No	Lecture Duration Period	Topics to be Covered	Support Material/Page Nos
		UNIT-I	
1	1 hr	Decimal, binary, octal, hexadecimal	T1, 42-44, 75,69
2	1 hr	Conversion of number system- Conversion of decimal to binary	T1- 47- 48
3	1 hr	Binary to decimal	T1 46-47
4	1 hr	Decimal to octal	T1 76
5	1 hr	Octal to decimal	T1 76
6	1 hr	Octal to binary	T1 77
7	1 hr	Binary to octal	T1 77
8	1 hr	Decimal to hexadecimal	T1 72
9	1 hr	Hexadecimal to decimal Hexadecimal to binary	T1 71, 70
10	1 hr	Binary to hexadecimal	T1 70
11	1 hr	Binary coded decimal -8421 code	T1 78
12	1 hr	Alphanumeric codes ASCII code	T1 84-87
13	1 hr	EBCDIC code	T1 83
14	1 hr	Error detecting code	T1 87
15	1 hr	Parity	T1 87
16	1 hr	Even parity and odd parity method	T1 87-91

17	1 hr	Revision	
	Total No of Hours Planned For Unit 1=17		
		UNIT-II	
1	1 hr	Logic gates	T1-105-108
2	1 hr	AND, OR, NOT, NAND, NOR gates	T1-109-131
3	1 hr	Construction of circuit only I/C	Hyperphysics
4	1 hr	Action truth table	T1-109
5	1 hr	Logic symbol	T1-109, 107, 106
6	1 hr	Boolean operators	T1-170
7	1 hr	Logic expressions	T1-181-183
8	1 hr	Demorgan's theorems	T1-49-54
9	1 hr	Laws and rules of Boolean algebra	T1-171-172
10	1 hr	Truth table	T1-192-195
11	1 hr	Reducing Boolean expressions	T1-196-197
12	1 hr	K maps	T1-196-197
13	1 hr	Logic diagrams of Boolean algebra expressions	T1-181-183
14	1 hr	Converting logic circuits to expressions	T1-182-184
15	1 hr	Revision	
	Total No of Hours Planned For Unit II=15		
		UNIT-III	
1	1 hr	Flip-flop definitions	T1-393
2	1 hr	Clocked flip-flop	T1-394
3	1 hr	S-R flip-flop	T1-394-395
4	1 hr	JK flip flop	T1-406
5	1 hr	Continuation	
6	1 hr	T-flip flop	T1-407

7	1 hr	D flip flop	T1-398
8	1 hr	Master slave J-K flip flop	T1-412-413
9	1 hr	Construction Circuits	
10	1 hr	Ring counter	T1-550
11	1 hr	Continuation	
12	1 hr	Ripple counter	T1-462
13	1 hr	Continuation	
14	1 hr	Mod counters	T1-461
15	1 hr	Continuation	
16	1 hr	Revision	
	Total No of Hours Planned For Unit III=16		
		UNIT-IV	
1	1 hr	XOR gates, Half adder	T1- 290
2	1 hr	Full adder,	T1-291
3	1 hr	Full subtractor	T1-301
4	1 hr	Parallel binary adder,	T1-293-294
5	1 hr	Parallel binary subtractor	T1-295-296
6	1 hr	Construction, action and truth table	T1-659
7	1 hr	Magnetic tape - Magnetic disc	T1-659
8	1 hr	Floppy disc	T1-661
9	2 hr	Magnetic cores	T1-663
10	1 hr	Magnetic core logic, Coincident memory	T1-663
11	1 hr	Memory addressing- MOS-	T1-663
12	1 hr	random access memory	T1-621
13	1 hr	MOD read only memory	T1-634-639
14	1 hr	PROM-EPROM	T1-641

15	1 hr	Revision	
	Total No of Hours Planned For Unit IV=15		
		UNIT-V	
1	1 hr	Brief history	T1 (165-167)
2	1 hr	Organization of 8085	T1 (168-172)
3	1 hr	Data	T1 (173-186)
4	1 hr	Address bus	T1 (186-192)
5	1 hr	Addressing the I/O devices, Registers in the 8085	TI (193-199)
6	1 hr	Instruction set	T1 (201-214)
7	1 hr	Instruction types	T1 (215-242)
8	1 hr	Classification of instruction	T1 (243-247)
9	1 hr	**Revision**	
10	1 hr	Old Question Paper Revision	
11	1 hr	Old Question Paper Revision	
12	1 hr	Old Question Paper Revision	
	Total No of Hours Planned for unit V=12		
Total Planned Hours	**75**		

TEXT BOOK

1. T1 : Floyd 2003, Digital Fundamentals, 8th edition Pearsaon education, New Delhi
2. T2- Microprocessor Architecture, Programming and application with the 8085, Gaonkar..

WEBSITE:

1. W1 : www. Hyperphysics.com

UNIT-I
SYLLABUS

Decimal, binary, octal, hexadecimal - Conversion of number system - Conversion of decimal to binary, binary to decimal- decimal to octal - Octal to decimal - Octal to binary - Binary to octal - Decimal to hexadecimal - Hexadecimal to decimal, hexadecimal to binary - Binary to hexadecimal. Binary coded decimal - 8421 code - Alphanumeric codes ASCII code - EBCDIC code - Error detecting code – Parity - Even parity and odd parity method.

NUMBER SYSTEMS

There are infinite ways to represent a number. The four commonly associated with modern computers and digital electronics are: decimal, binary, octal, and hexadecimal.

Decimal (base 10) is the way most human beings represent numbers. Decimal is sometimes abbreviated as dec.

Decimal counting goes:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, and so on.

Binary (base 2) is the natural way most digital circuits represent and manipulate numbers. (Common misspellings are “bianary”, “bienary”, or “binery”.) Binary numbers are sometimes represented by preceding the value with '0b', as in 0b1011. Binary is sometimes abbreviated as bin.

Binary counting goes:

0, 1, 10, 11, 100, 101, 110, 111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111, 10000, 10001, and so on.

Octal (base 8) was previously a popular choice for representing digital circuit numbers in a form that is more compact than binary. Octal is sometimes abbreviated as oct.

Octal counting goes:

0, 1, 2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21, and so on.

Hexadecimal (base 16) is currently the most popular choice for representing digital circuit numbers in a form that is more compact than binary. (Common misspellings are “hexdecimal”,

"hexidecimal", "hexedecimal", or "hexodecimal".) Hexadecimal numbers are sometimes represented by preceding the value with '0x', as in 0x1B84. Hexadecimal is sometimes abbreviated as hex.

Hexadecimal counting goes:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, 10, 11, and so on.

All four number systems are equally capable of representing any number. Furthermore, a number can be perfectly converted between the various number systems without any loss of numeric value.

Decimal Number System

In decimal number system we use ten digits 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9. Decimal implies base 10.

Binary Number System

In Binary number system we use two digits 0 and 1. Binary implies base 2.

Octal Number System

In Octal number system we use eight digits 0, 1, 2, 3, 4, 5, 6 and 7. Octal implies base 8.

Hexadecimal Number System

In hexadecimal number system we use ten digits and six english alphabet letters 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

10 is denoted as A

11 is denoted as B

12 is denoted as C

13 is denoted as D

14 is denoted as E

15 is denoted as F

Hexadecimal implies base 16.

Conversion Table

Following is the conversion table that we will use as a reference to perform conversion operations.

DECIMAL	BINARY	OCTAL	HEXADECIMAL
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

1. Binary to Octal conversion

To convert a binary number into octal we follow the given steps

1. Divide the binary digits into groups of 3 digits, starting from the right.

2. Convert each group of 3 binary digits into 1 octal digit.

Convert Binary number 100101_2 into Octal form

Step 1. Make groups of 3 digits from right

100101_2

Groups: 100_2 101_2

Step 2. Convert each 3 digits group into 1 octal digit

$101_2 = 58$

$100_2 = 48$

so, 1001012 = 458

2. Octal to Binary conversion

To convert an Octal number into Binary we follow the following steps

1. Convert each octal digits into 3 digits binary group

2. Combine the groups

Convert Octal number 45_8 into Binary form

Step 1. Convert each octal digit into 3 digits binary group

45_8

Groups: 4_8 5_8

$5_8 = 1012$

$4_8 = 1002$

Step 2. Combine the groups

so, $45_8 = 100101_2$

3. Binary to Hexadecimal conversion

To convert a binary number into hexadecimal we follow the given steps

1. Divide the binary digits into groups of 4 digits, starting from the right

2. Convert each group of 4 binary digits into 1 hexadecimal digit

Convert Binary number 10100101_2 into Hexadecimal form

Step 1. Make groups of 4 digits from right

10100101_2

Groups: 1010_2 0101_2

$0101_2 = 5_{16}$

$1010_2 = 4_{16}$

Step 2. Combine the groups

so, $10100101_2 = 45_{16}$

4. Hexadecimal to Binary conversion

To convert a hexadecimal number into binary we follow the given steps

1. Convert each hexadecimal digit into group of 4 digits binary

2. Combine the groups

Convert Hexadecimal number A516 into Binary form

Step 1. Convert each hexadecimal digit into group of 4 digits binary

$A5_{16}$

Groups: A_{16} 5_{16}

$5_{16} = 0101_2$

$A_{16} = 1010_2$

Step 2. Combine the groups

so, $A5_{16} = 10100101_2$

5. Octal to Hexadecimal conversion

To convert an octal number into hexadecimal we follow the given steps

1. Convert each octal digit into groups of 3 digits binary.

2. Combine the groups from step 1.

3. Divide the binary digits from step 2 into groups of 4 digits, starting from the right

4. Convert each group of 4 binary digits into 1 hexadecimal digit

Convert Octal number 25_8 into Hexadecimal form

Step 1. Convert each octal digit into groups of 3 digits binary

25_8

Groups: 2_8 5_8

$5_8 = 101_2$

$2_8 = 010_2$

Step 2. Combine the groups

so, $25_8 = 010101_2$

Step 3. Divide the binary digits from step 2 into groups of 4 digits, starting from the right

Groups: 0001_2 0101_2

Step 4. Convert each group of 4 binary digits into 1 hexadecimal digit

$0101_2 = 5_{16}$

$0001_2 = 1_{16}$

so, $25_8 = 15_{16}$

6. Hexadecimal to Octal conversion

To convert a hexadecimal number into octal we follow the given steps

1. Convert each hexadecimal digit into groups of 4 digits binary.

2. Combine the groups from step 1.

3. Divide the binary digits from step 2 into groups of 3 digits, starting from the right

4. Convert each group of 3 binary digits into 1 octal digit

Convert Hexadecimal number 15_{16} into Octal form

Step 1. Convert each hexadecimal digit into groups of 4 digits binary

15_{16}

Groups: 1_{16} 5_{16}

$5_{16} = 0101_2$

$1_{16} = 0001_2$

Step 2. Combine the groups

so, $15_{16} = 00010101_2$

Step 3. Divide the binary digits from step 2 into groups of 3 digits, starting from the right

Groups: 000_2 010_2 101_2

Step 4. Convert each group of 4 binary digits into 1 hexadecimal digit

$101_2 = 5_8$

$010_2 = 2_8$

$000_2 = 0_8$

so, $15_{16} = 025_8 = 25_8$

Fundamental Information Element of Digital Circuits

Almost all modern digital circuits are based on two-state switches. The switches are either on or off. It doesn't matter if the switches are actually physical switches, vacuum tubes, relays, or transistors. And, it doesn't matter if the 'on' state is represented by 1.8 V on a cutting-edge CPU core, -12 V on a RS-232 interface chip, or 5 V on a classic TTL logic chip.

Because the fundamental information element of digital circuits has two states, it is most naturally represented by a number system where each individual digit has two states: binary.

For example, switches that are 'on' are represented by '1' and switches that are 'off' are represented by '0'. It is easy to instantly comprehend the values of 8 switches represented in binary as 10001101. It is also easy to build a circuit to display each switch state in binary, by having an LED (lit or unlit) for each binary digit.

Making Values More Compact

"Binary digit" is a little unwieldy to say over and over, so the term was contracted to "bit". Not only is the term "binary digit" a little unwieldy, but so is the length of a binary number, since each digit can only represent one switch. As digital circuits grew more complex, a more compact form of representing circuit information became necessary.

An octal number (base 8) can be up to 1/3 the length of a binary number (base 2). 8 is a whole power of 2 (2^3=8). That means three binary digits convert neatly into one octal digit.

A hexadecimal number (base 16) can be up to 1/4 the length of a binary number. 16 is a whole power of 2 (2^4=16). That means four binary digits convert neatly into one hexadecimal digit.

Unfortunately, decimal (base 10) is not a whole power of 2. So, it is not possible to simply chunk groups of binary digits to convert the raw state of a digital circuit into the human-centric format.

Decimal Number Conversion

A repeated division and remainder algorithm can convert decimal to binary, octal, or hexadecimal.

Divide the decimal number by the desired target radix (2, 8, or 16).

Append the remainder as the next most significant digit.

Repeat until the decimal number has reached zero.

Decimal to Binary

Here is an example of using repeated division to convert 1792 decimal to binary:

Decimal Number	Operation	Quotient	Remainder	Binary Result
1792	÷ 2 =	896	0	0
896	÷ 2 =	448	0	00
448	÷ 2 =	224	0	000
224	÷ 2 =	112	0	0000
112	÷ 2 =	56	0	00000
56	÷ 2 =	28	0	000000
28	÷ 2 =	14	0	0000000
14	÷ 2 =	7	0	00000000
7	÷ 2 =	3	1	100000000
3	÷ 2 =	1	1	1100000000
1	÷ 2 =	0	1	11100000000
0	done.			

Decimal to Octal

Here is an example of using repeated division to convert 1792 decimal to octal:

Decimal Number	Operation	Quotient	Remainder	Octal Result
1792	÷ 8 =	224	0	0
224	÷ 8 =	28	0	00
28	÷ 8 =	3	4	400
3	÷ 8 =	0	3	3400
0	done.			

Decimal to Hexadecimal

Here is an example of using repeated division to convert 1792 decimal to hexadecimal:

Decimal Number	Operation	Quotient	Remainder	Hexadecimal Result
1792	÷ 16 =	112	0	0
112	÷ 16 =	7	0	00
7	÷ 16 =	0	7	700
0	done.			

The only addition to the algorithm when converting from decimal to hexadecimal is that a table must be used to obtain the hexadecimal digit if the remainder is greater than decimal 9.

Decimal:	0	1	2	3	4	5	6	7
Hexadecimal:	0	1	2	3	4	5	6	7

Decimal:	8	9	10	11	12	13	14	15
Hexadecimal:	8	9	A	B	C	D	E	F

The addition of letters can make for funny hexadecimal values. For example, 48879 decimal converted to hex is:

Decimal Number	Operation	Quotient	Remainder	Hexadecimal Result
48879	÷ 16 =	3054	15	F
3054	÷ 16 =	190	14	EF
190	÷ 16 =	11	14	EEF
11	÷ 16 =	0	11	BEEF
0	done.			

Other fun hexadecimal numbers include: AD, BE, FAD, FADE, ADD, BED, BEE, BEAD, DEAF, FEE, ODD, BOD, DEAD, DEED, BABE, CAFE, C0FFEE, FED, FEED, FACE, BAD, F00D, and my initials DAC.

Binary Number Conversion

Binary to Octal

An easy way to convert from binary to octal is to group binary digits into sets of three, starting with the least significant (rightmost) digits.

Binary: 11100101 =	11 100 101	
	011 100 101	Pad the most significant digits with zeros if necessary to complete a group of three.

Then, look up each group in a table:

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

Binary = 011 100 101

Octal = 3 4 5 = 345 oct

Binary to Hexadecimal

An equally easy way to convert from binary to hexadecimal is to group binary digits into sets of four, starting with the least significant (rightmost) digits.

Binary: 11100101 = 1110 0101

Then, look up each group in a table:

Binary:	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal:	0	1	2	3	4	5	6	7

Binary:	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal:	8	9	A	B	C	D	E	F

Binary = 1110 0101

Hexadecimal = E 5 = E5 hex

Binary to Decimal

They say there are only 10 people in this world: those that understand binary and those that don't. One method involves addition and multiplication.

Start the decimal result at 0.

Remove the most significant binary digit (leftmost) and add it to the result.

If all binary digits have been removed, you're done. Stop.

Otherwise, multiply the result by 2.

Go to step 2.

Here is an example of converting 11100000000 binary to decimal:

Binary Digits	Operation	Decimal Result	Operation	Decimal Result
11100000000	+1	1	× 2	2
1100000000	+1	3	× 2	6
100000000	+1	7	× 2	14
00000000	+0	14	× 2	28
0000000	+0	28	× 2	56
000000	+0	56	× 2	112
00000	+0	112	× 2	224
0000	+0	224	× 2	448
000	+0	448	× 2	896
00	+0	896	× 2	1792
0	+0	1792	done.	

4. Octal Number Conversion

Octal to Binary

Converting from octal to binary is as easy as converting from binary to octal. Simply look up each octal digit to obtain the equivalent group of three binary digits.

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

Octal =	3	4	5	
Binary =	011	100	101	= 011100101 binary

Octal to Hexadecimal

When converting from octal to hexadecimal, it is often easier to first convert the octal number into binary and then from binary into hexadecimal. For example, to convert 345 octal into hex:

(from the previous example)

Octal =	3	4	5	
Binary =	011	100	101	= 011100101 binary

Drop any leading zeros or pad with leading zeros to get groups of four binary digits (bits):

Binary 011100101 = 1110 0101

Then, look up the groups in a table to convert to hexadecimal digits.

Binary:	0000	0001	0010	0011	0100	0101	0110	0111
Hexadecimal:	0	1	2	3	4	5	6	7

Binary:	1000	1001	1010	1011	1100	1101	1110	1111
Hexadecimal:	8	9	A	B	C	D	E	F

Binary =	1110	0101	
Hexadecimal =	E	5	= E5 hex

Therefore, through a two-step conversion process, octal 345 equals binary 011100101 equals hexadecimal E5.

Octal to Decimal

Converting octal to decimal can be done with repeated division.

Start the decimal result at 0.

Remove the most significant octal digit (leftmost) and add it to the result.

If all octal digits have been removed, you're done. Stop.

Otherwise, multiply the result by 8.

Go to step 2.

Octal Digits	Operation	Decimal Result	Operation	Decimal Result
345	+3	3	× 8	24
45	+4	28	× 8	224
5	+5	229	done.	

The conversion can also be performed in the conventional mathematical way, by showing each digit place as an increasing power of 8.

345 octal = $(3 * 8^2) + (4 * 8^1) + (5 * 8^0) = (3 * 6^4) + (4 * 8) + (5 * 1) = 229$ decimal

Hexadecimal Number Conversion

Hexadecimal to Binary

Converting from hexadecimal to binary is as easy as converting from binary to hexadecimal. Simply look up each hexadecimal digit to obtain the equivalent group of four binary digits.

Hexadecimal:	0	1	2	3	4	5	6	7
Binary:	0000	0001	0010	0011	0100	0101	0110	0111

Hexadecimal:	8	9	A	B	C	D	E	F
Binary:	1000	1001	1010	1011	1100	1101	1110	1111

Hexadecimal =	A	2	D	E	
Binary =	1010	0010	1101	1110	= 1010001011011110 binary

Hexadecimal to Octal

When converting from hexadecimal to octal, it is often easier to first convert the hexadecimal number into binary and then from binary into octal. For example, to convert A2DE hex into octal:

(from the previous example)

Hexadecimal =	A	2	D	E	
Binary =	1010	0010	1101	1110	= 1010001011011110 binary

Add leading zeros or remove leading zeros to group into sets of three binary digits.

Binary: 1010001011011110 = 001 010 001 011 011 110

Then, look up each group in a table:

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

Binary = 001 010 001 011 011 110

Octal = 1 2 1 3 3 6 = 121336 octal

Therefore, through a two-step conversion process, hexadecimal A2DE equals binary 1010001011011110 equals octal 121336.

Hexadecimal to Decimal

Converting hexadecimal to decimal can be performed in the conventional mathematical way, by showing each digit place as an increasing power of 16. Of course, hexadecimal letter values need to be converted to decimal values before performing the math.

Hexadecimal:	0	1	2	3	4	5	6	7
Decimal:	0	1	2	3	4	5	6	7
Hexadecimal:	8	9	A	B	C	D	E	F
Decimal:	8	9	10	11	12	13	14	15

A2DE hexadecimal:

$= ((A) * 16^3) + (2 * 16^2) + ((D) * 16^1) + ((E) * 16^0)$

$= (10 * 16^3) + (2 * 16^2) + (13 * 16^1) + (14 * 16^0)$

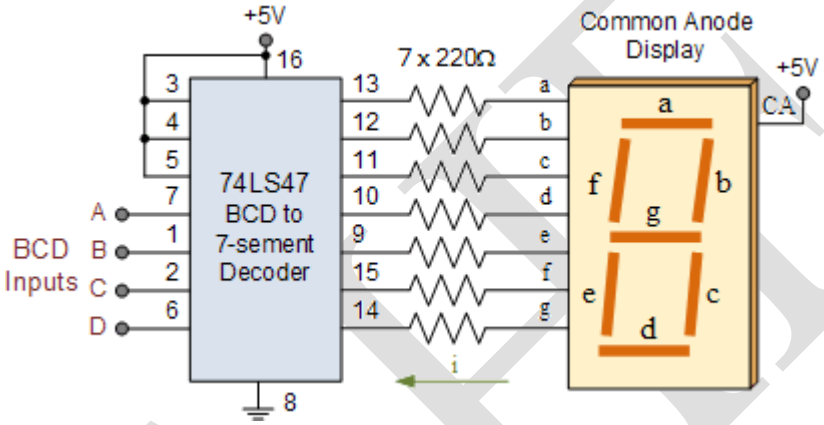
$= (10 * 4096) + (2 * 256) + (13 * 16) + (14 * 1)$

= 40960 + 512 + 208 + 14

= 41694 decimal

Binary Coded Decimal

Binary Coded Decimal or BCD as it is more commonly called, is another process for converting decimal numbers into their binary equivalents.



As we have seen in this Binary Numbers section of tutorials, there are many different binary codes used in digital and electronic circuits, each with its own specific use.

As we naturally live in a decimal (base-10) world we need some way of converting these decimal numbers into a binary (base-2) environment that computers and digital electronic devices understand, and binary coded decimal code allows us to do that.

We have seen previously that an n-bit binary code is a group of "n" bits that assume up to 2n distinct combinations of 1's and 0's. The advantage of the Binary Coded Decimal system is that each decimal digit is represented by a group of 4 binary digits or bits in much the same way as Hexadecimal. So for the 10 decimal digits (0-to-9) we need a 4-bit binary code.

But do not get confused, binary coded decimal is not the same as hexadecimal. Whereas a 4-bit hexadecimal number is valid up to F16 representing binary11112, (decimal 15), binary coded decimal numbers stop at 9 binary 10012. This means that although 16 numbers (24) can be represented using four binary digits, in the BCD numbering system the six binary code combinations of: 1010 (decimal 10), 1011 (decimal 11), 1100 (decimal 12), 1101(decimal 13), 1110 (decimal 14), and 1111 (decimal 15) are classed as forbidden numbers and can not be used.

The main advantage of binary coded decimal is that it allows easy conversion between decimal (base-10) and binary (base-2) form. However, the disadvantage is that BCD code is wasteful as the states between 1010(decimal 10), and 1111 (decimal 15) are not used. Nevertheless, binary coded decimal has many important applications especially using digital displays.

In the BCD numbering system, a decimal number is separated into four bits for each decimal digit within the number. Each decimal digit is represented by its weighted binary value performing a direct translation of the number. So a 4-bit group represents each displayed decimal digit from 0000 for a zero to 1001 for a nine.

So for example, 35710 (Three Hundred and Fifty Seven) in decimal would be presented in Binary Coded Decimal as:

35710 = 0011 0101 0111 (BCD)

Then we can see that BCD uses weighted codification, because the binary bit of each 4-bit group represents a given weight of the final value. In other words, the BCD is a weighted code and the weights used in binary coded decimal code are 8, 4, 2, 1, commonly called the 8421 code as it forms the 4-bit binary representation of the relevant decimal digit.

Binary Coded Decimal Representation of a Decimal Number

Binary Power	23	22	21	20
Binary Weight:	8	4	2	1

The decimal weight of each decimal digit to the left increases by a factor of 10. In the BCD number system, the binary weight of each digit increases by a factor of 2 as shown. Then the first digit has a weight of 1 (20), the second digit has a weight of 2 (21), the third a weight of 4 (22), the fourth a weight of 8 (23).

Then the relationship between decimal (denary) numbers and weighted binary coded decimal digits is given below.

Truth Table for Binary Coded Decimal

Decimal Number	BCD 8421 Code
0	0000 0000

1	0000 0001
2	0000 0010
3	0000 0011
4	0000 0100
5	0000 0101
6	0000 0110
7	0000 0111
8	0000 1000
9	0000 1001
10 (1+0)	0001 0000
11 (1+1)	0001 0001
12 (1+2)	0001 0010
…	…
20 (2+0)	0010 0000
21 (2+1)	0010 0001
22 (2+2)	0010 0010
etc, continuing upwards in groups of four	

Then we can see that 8421 BCD code is nothing more than the weights of each binary digit, with each decimal (denary) number expressed as its four-bit pure binary equivalent.

Decimal-to-BCD Conversion

As we have seen above, the conversion of decimal to binary coded decimal is very similar to the conversion of hexadecimal to binary. Firstly, separate the decimal number into its weighted digits and then write down the equivalent 4-bit 8421 BCD code representing each decimal digit as shown.

Binary Coded Decimal Example No1

Using the above table, convert the following decimal (denary) numbers: 85_{10}, 572_{10} and 8579_{10} into their 8421 BCD equivalents.

85_{10} = 1000 0101 (BCD)

572_{10} = 0101 0111 0010 (BCD)

8579_{10} = 1000 0101 0111 1001 (BCD)

Note that the resulting binary number after the conversion will be a true binary translation of decimal digits. This is because the binary code translates as a true binary count.

BCD-to-Decimal Conversion

The conversion from binary coded decimal to decimal is the exact opposite of the above. Simply divide the binary number into groups of four digits, starting with the least significant digit and then write the decimal digit represented by each 4-bit group. Add additional zero's at the end if required to produce a complete 4-bit grouping. So for example, 1101012 would become: 0011 01012 or 3510 in decimal.

Binary Coded Decimal Example No: 2

Convert the following binary numbers: 1001_2, 1010_2, 1000111_2 and 10100111000.101_2 into their decimal equivalents.

1001_2 = 1001BCD = 9_{10}

1010_2 = this will produce an error as it is decimal 1010 and not a valid BCD number

1000111_2 = 0100 0111BCD = 47_{10}

10100111000.101_2 = 0101 0011 0001.1010BCD = 538.625_{10}

The conversion of BCD-to-decimal or decimal-to-BCD is a relatively straight forward task but we need to remember that BCD numbers are decimal numbers and not binary numbers, even though they are represented using bits. The BCD representation of a decimal number is important to understand, because microprocessor based systems used by most people needs to be in the decimal system.

However, while BCD is easy to code and decode, it is not an efficient way to store numbers. In the standard 8421 BCD encoding of decimal numbers, the number of individual data bits needed to represent a given decimal number will always be greater than the number of bits required for an equivalent binary encoding.

For example, in binary a three digit decimal number from 0-to-999 requires only 10-bits (1111100111_2), whereas in binary coded decimal, the same number requires a minimum of 12-bits (0011 1110 0111BCD) for the same representation.

Also, performing arithmetic tasks using binary coded decimal numbers can be a bit awkward since each digit can not exceed 9. The addition of two decimal digits in BCD, will create a possible carry bit of 1 which needs to be added to the next group of 4-bits.

If the binary sum with the added carry bit is equal to or less than 9 (1001), the corresponding BCD digit is correct. But when the binary sum is greater than 9 the result is an invalid BCD digit. Therefore it is better to convert BCD numbers into pure binary, perform the required addition, and then convert the back to BCD before displaying the results.

Nevertheless, the use of a BCD coding system in both microelectronics and computer systems is particularly useful in situations where the binary coded decimal is intended to be displayed on one or more 7-segment LED or LCD displays and there are many popular integrated circuits available that are configured to give a BCD output or outputs.

One common IC is the 74LS90 asynchronous counter/divider that contains independent divide-by-2 and divide-by-5 counters that can be used together to produce a divide-by-10 decade counter with BCD outputs. Another is the 74LS390 which is a dual version of the basic 74LS90, and can also be configured to produce a BCD output.

But the most commonly used BCD encoded IC's are the 74LS47 and the 74LS48 BCD to 7-segment decoder/driver, which converts a 4-bit BCD code of a counter, etc. and converts it into the required display code to drive the individual segments of a 7-segment LED display. While both IC's are functionally the same, the 74LS47 has active-low outputs for driving common-anode displays, while the 74LS48 has active-high outputs for driving common-cathode displays.

Addition with BCD

To perform addition in BCD, you can first add-up in binary format, and then perform the conversion to BCD afterwards. This conversion involves adding 6 to each group of four digits that has a value of greater-than 9. For example: 9+5=14 = [1001] + [0101] = [1110] in binary.

However, in BCD, we cannot have a value greater-than 9 (1001) per-nibble. To correct this, one adds 6 to that group: [0000 1110] + [0000 0110] = [0001 0100]

which gives us two nibbles, [0001] and [0100] which correspond to "1" and "4" respectively. This gives us the 14 in BCD which is the correct result.

BCD Code or 8421 Code

The full form of BCD is 'Binary-Coded Decimal.' Since this is a coding scheme relating decimal and binary numbers, four bits are required to code each decimal number. For example, (35)10 is represented as 0011 0101 using BCD code, rather than (100011)2. From the example it is clear that it requires more number of bits to code a decimal number using BCD code than using the straight binary code. However, inspite of this disadvantage it is convenient to use BCD code for input and output operations in digital systems.

The code is also known as 8-4-2-1 code. This is because 8, 4, 2, and 1 are the weights of the four bits of the BCD code. The weight of the LSB is 20 or 1, that of the next higher order bit is 21 or 2, that of the next higher order bit is 22 or 4, and that of the MSB is 23 or 8. Therefore, this is a weighted code and arithmetic operations can be performed using this code, which will be discussed later on. The bit assignment 0101, for example, can be interpreted by the weights to represent the decimal digit 5 because $0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 = 5$. Since four binary bits are used the maximum decimal equivalent that may be coded is 1510 (i.e., 11112). But the maximum decimal digit available is 910. Hence the binary codes 1010, 1011, 1100,

Decimal digit	*(BCD) 8421*
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

1101, 1110, 1111, representing 10, 11, 12, 13, 14, and 15 in decimal are never being used in BCD code. So these six codes are called forbidden codes and the group of these codes is called the forbidden group in BCD code. BCD code for decimal digits 0 to 9 is shown in the following table.

Example 1. Give the BCD equivalent for the decimal number 589.

Solution. The decimal number is 5 8 9

BCD code is 0101 1000 1001 Hence, (589)10 = (010110001001)BCD

Example 2. Give the BCD equivalent for the decimal number 69.27.

Solution. The decimal number 6 9 2 7

BCD code is s0110 1001 0010 0111 Hence, (69.27)10 = (01101001.00100111)BCD

Alphanumeric codes | ASCII code | EBCDIC code | UNICODE

Alphanumeric codes are sometimes called character codes due to their certain properties. Now these codes are basically binary codes. We can write alphanumeric data, including data, letters of the alphabet, numbers, mathematical symbols and punctuation marks by this code which can be easily understandable and can be processed by the computers. Input output devices such as keyboards, monitors, mouse can be interfaced using these codes. 12-bit Hollerith code is the better known and perhaps the first effective code in the days of evolving computers in early days. During this period punch cards were used as the inputting and outputting data. But nowadays these codes are termed obsolete as many other modern codes have evolved. The most common alphanumeric

codesused these days are ASCII code, EBCDIC code and Unicode. Now we will discuss about them briefly.

ASCII code

The full form of ASCII code is American Standard Code for Information Interchange. It is a seven bit code based on the English alphabet. In 1967 this code was first published and since then it is being modified and updated. ASCII code has 128 characters some of which are enlisted below to get familiar with the code.

DEC	OCT	HEX	BIN	Symbol	Description
0	000	00	00000000	NUL	Null char
1	001	01	00000001	SOH	Start of Heading
2	002	02	00000010	STX	Start of Text
3	003	03	00000011	ETX	End of Text
4	004	04	00000100	EOT	End of Transmission
5	005	05	00000101	ENQ	Enquiry
6	006	06	00000110	ACK	Acknowledgment
7	007	07	00000111	BEL	Bell
8	010	08	00001000	BS	Back Space
9	011	09	00001001	HT	Horizontal Tab
10	012	0A	00001010	LF	Line Feed
11	013	0B	00001011	VT	Vertical Tab
12	014	0C	00001100	FF	Form Feed
13	015	0D	00001101	CR	Carriage Return
14	016	0E	00001110	SO	Shift Out / X-On
15	017	0F	00001111	SI	Shift In / X-O

There are many more codes which are not included here.

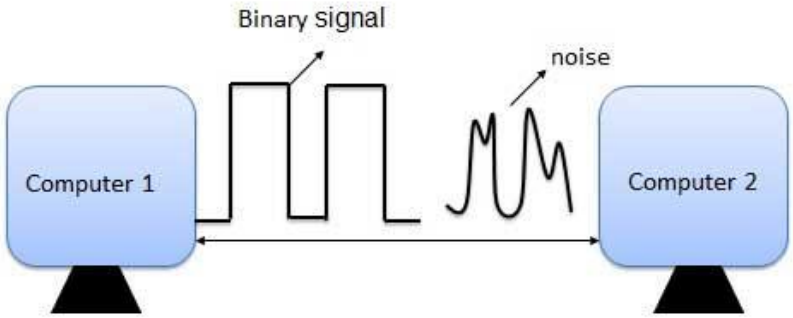
EBCDIC code

The EBCDIC stands for Extended Binary Coded Decimal Interchange Code. IBM invented this code to extend the Binary Coded Decimal which existed at that time. All the IBM computers and peripherals use this code. It is an 8 bit code and therefore can accommodate 256 characters. Below is given some characters of EBCDIC code to get familiar with it.

Char	EBCDIC	HEX	Char	EBCDIC	HEX	Char	EBCDIC	HEX
A	1100 0001	C1	P	1101 0111	D7	4	1111 0100	F4
B	1100 0010	C2	Q	1101 1000	D8	5	1111 0101	F5
C	1100 0011	C3	R	1101 1001	D9	6	1111 0110	F6
D	1100 0100	C4	S	1110 0010	E2	7	1111 0111	F7
E	1100 0101	C5	T	1110 0011	E3	8	1111 1000	F8
F	1100 0110	C6	U	1110 0100	E4	9	1111 1001	F9
G	1100 0111	C7	V	1110 0101	E5	blank	...	...
H	1100 1000	C8	W	1110 0110	E6	.	...	...
I	1100 1001	C9	X	1110 0111	E7	(	...	...
J	1101 0001	D1	Y	1110 1000	E8	+	...	...
K	1101 0010	D2	Z	1110 1001	E9	$	...	...
L	1101 0011	D3	0	1111 0000	F0	*	...	...
M	1101 0100	D4	1	1111 0001	F1	)	...	...
N	1101 0101	D5	2	1111 0010	F2	-	...	...
O	1101 0110	D6	3	1111 0011	F3	/		

What is Error?

Error is a condition when the output information does not match with the input information. During transmission, digital signals suffer from noise that can introduce errors in the binary bits travelling from one system to other. That means a 0 bit may change to 1 or a 1 bit may change to 0.



Error-Detecting codes

Whenever a message is transmitted, it may get scrambled by noise or data may get corrupted. To avoid this, we use error-detecting codes which are additional data added to a given digital message to help us detect if an error occurred during transmission of the message. A simple example of error-detecting code is parity check.

Error-Correcting codes

Along with error-detecting code, we can also pass some data to figure out the original message from the corrupt message that we received. This type of code is called an error-correcting code. Error-correcting codes also deploy the same strategy as error-detecting codes but additionally, such codes also detect the exact location of the corrupt bit.

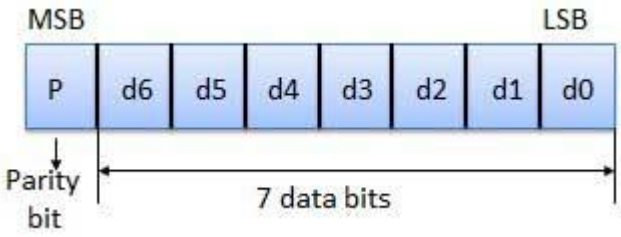
In error-correcting codes, parity check has a simple way to detect errors along with a sophisticated mechanism to determine the corrupt bit location. Once the corrupt bit is located, its value is reverted (from 0 to 1 or 1 to 0) to get the original message.

How to Detect and Correct Errors?

- To detect and correct the errors, additional bits are added to the data bits at the time of transmission.
- The additional bits are called parity bits. They allow detection or correction of the errors.
- The data bits along with the parity bits form a code word.

Parity Checking of Error Detection

It is the simplest technique for detecting and correcting errors. The MSB of an 8-bits word is used as the parity bit and the remaining 7 bits are used as data or message bits. The parity of 8-bits transmitted word can be either even parity or odd parity.



- Even parity -- Even parity means the number of 1's in the given word including the parity bit should be even (2,4,6,....).
- Odd parity -- Odd parity means the number of 1's in the given word including the parity bit should be odd (1,3,5,....).
- Use of Parity Bit
- The parity bit can be set to 0 and 1 depending on the type of the parity required.
- For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).
- For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in fig. (b).

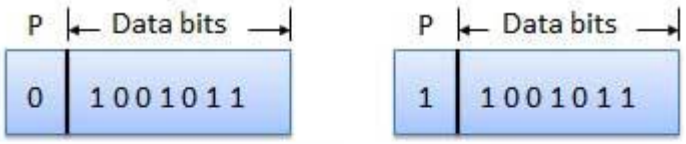


Fig. (a)

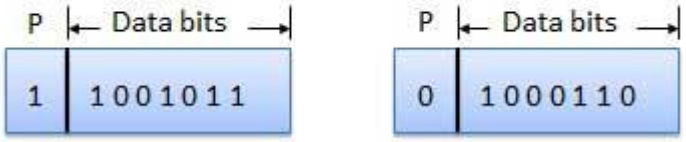
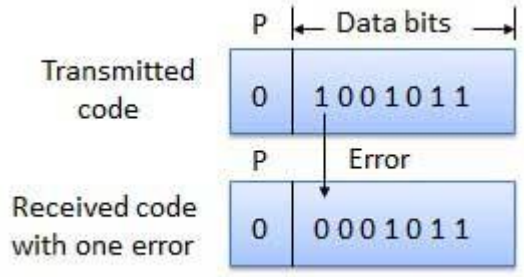


Fig. (b)

How Does Error Detection Take Place?

Parity checking at the receiver can detect the presence of an error if the parity of the receiver signal is different from the expected parity. That means, if it is known that the parity of the transmitted signal is always going to be "even" and if the received signal has an odd parity, then the receiver can conclude that the received signal is not correct. If an error is detected, then the receiver will ignore the received byte and request for retransmission of the same byte to the transmitter.



POSSIBLE QUESTIONS

EIGHT MARK QUESTIONS

1. $(125)_{10} = (x)_8 = (y)_{16} = (z)_2$ find x, y,z.
2. Write $(25)_{10}$ $(123)_{10}$ $(37)_{10}$ in binary and BCD.
3. Convert the following octal system into decimal system

   (i) $(46)_8$ (ii) $(12.6)_8$ (iii) $(23.625)_8$ (iv) $(36.77)_8$
4. What is the binary number that follows 101111.
5. Convert the following hexadecimal system into binary and decimal systems

   i) $(FF)_{16}$ (ii) $(3EF)_{16}$ (iii) $(56.48)_{16}$
6. Explain the following terms in details (i) BCD 8421 code (ii) ASCII code
7. Convert the following system into decimal, hexa and octal system

   (i) $(11101)_2$ (ii) $(111.11)_2$
8. Explain the procedure for BCD addition
9. $(29A.8)_{16} = (x)_{10} = (y)_2 = (z)_8$ find x,y,z.
10. $(11001001.101)_2 = (x)_{10} = (y)_2 = (z)_8$ find x,y,z.

UNIT-II

SYLLABUS

Logic gates - AND, OR, NOT, NAND, NOR gates - Construction of circuit only I/C - Action truth table - Logic symbol.

Boolean operators - Logic expressions – Demorgan's theorems - Laws and rules of Boolean algebra - Truth table - Reducing Boolean expressions, K maps; logic diagrams of Boolean algebra expressions - Converting logic circuits to expressions.

LOGIC GATE (AND, OR, XOR, NOT, NAND, NOR AND XNOR)

Definition:

A logic gate is an elementary building block of a digital circuit. Most logic gates have two inputs and one output. At any given moment, every terminal is in one of the two binaryconditions low (0) or high (1), represented by different voltage levels. The logic state of a terminal can, and generally does, change often, as the circuit processes data. In most logic gates, the low state is approximately zero volts (0 V), while the high state is approximately five volts positive (+5 V).

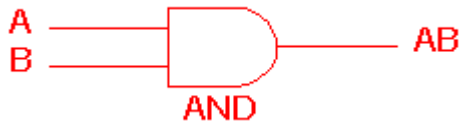
Type of Logic gate:

There are seven basic logic gates:

- AND
- OR
- XOR
- NOT
- NAND
- NOR and
- XNOR

AND gate:

The AND gate is so named because, if 0 is called "false" and 1 is called "true," the gate acts in the same way as the logical "and" operator. The following illustration and table show the circuit symbol and logic combinations for an AND gate. (In the symbol, the input terminals are at left and the output terminal is at right.) The output is "true" when both inputs are "true." Otherwise, the output is "false."



AND gate

Input 1	Input 2	Output
0	0	0
0	1	0
1	0	0
1	1	1

OR gate:

The OR gate gets its name from the fact that it behaves after the fashion of the logical inclusive "or." The output is "true" if either or both of the inputs are "true." If both inputs are "false," then the output is "false."

OR gate

Input 1	Input 2	Output
0	0	0
0	1	1

1	0	1
1	1	1

XOR gate:

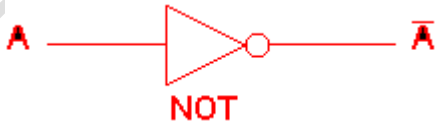
The XOR (exclusive-OR) gate acts in the same way as the logical "either/or." The output is "true" if either, but not both, of the inputs are "true." The output is "false" if both inputs are "false" or if both inputs are "true." Another way of looking at this circuit is to observe that the output is 1 if the inputs are different, but 0 if the inputs are the same.

XOR gate

Input 1	Input 2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Inverter or NOT gate:

A logical inverter , sometimes called a NOT gate to differentiate it from other types of electronic inverter devices, has only one input. It reverses the logic state.



Inverter or NOT gate

Input	Output

1	0
0	1

NAND gate:

The NAND gate operates as an AND gate followed by a NOT gate. It acts in the manner of the logical operation "and" followed by negation. The output is "false" if both inputs are "true." Otherwise, the output is "true."

NAND gate

Input 1	Input 2	Output
0	0	1
0	1	1
1	0	1
1	1	0

NOR gate:

The NOR gate is a combination OR gate followed by an inverter. Its output is "true" if both inputs are "false." Otherwise, the output is "false."

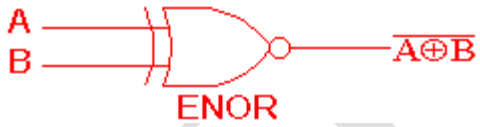
NOR gate

Input 1	Input 2	Output
0	0	1
0	1	0

1	0	0
1	1	0

XNOR gate:

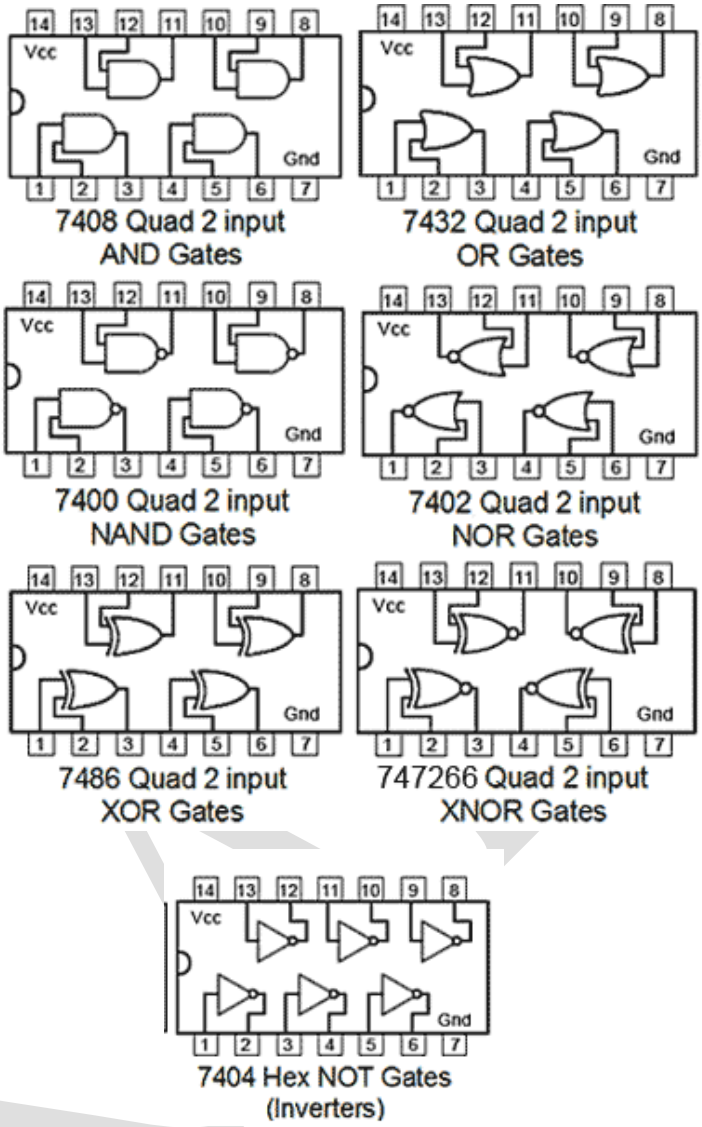
The XNOR (exclusive-NOR) gate is a combination XOR gate followed by an inverter. Its output is "true" if the inputs are the same, and"false" if the inputs are different.



XNOR gate

Input 1	Input 2	Output
0	0	1
0	1	0
1	0	0
1	1	10

Using combinations of logic gates, complex operations can be performed. In theory, there is no limit to the number of gates that can be arrayed together in a single device. But in practice, there is a limit to the number of gates that can be packed into a given physical space. Arrays of logic gates are found in digital integrated circuits (ICs). As IC technology advances, the required physical volume for each individual logic gate decreases and digital devices of the same or smaller size become capable of performing ever-more-complicated operations at ever-increasing speeds.



De Morgan's Theorems

De Morgan has suggested two theorems which are extremely useful in Boolean Algebra. The two theorems are discussed below.

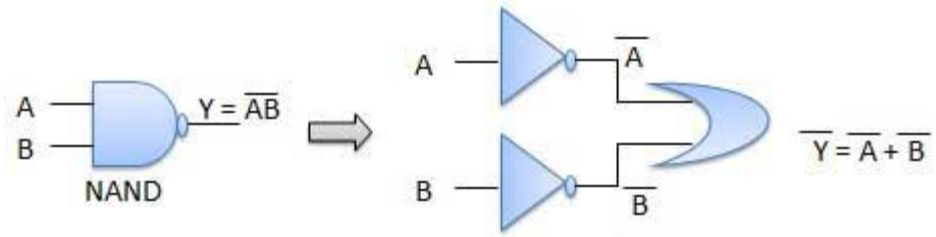
Theorem 1

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

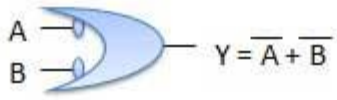
NAND = Bubbled OR

The left hand side (LHS) of this theorem represents a NAND gate with inputs A and B, whereas the right hand side (RHS) of the theorem represents an OR gate with inverted inputs.

This OR gate is called as Bubbled OR.



NAND ≡ Bubbled OR



Bubbled OR

Table showing verification of the De Morgan's first theorem −

A	B	$\overline{AB}$	$\overline{A}$	$\overline{B}$	$\overline{A} + \overline{B}$
0	0	1	1	1	1
0	1	1	1	0	1
1	0	1	0	1	1
1	1	0	0	0	0

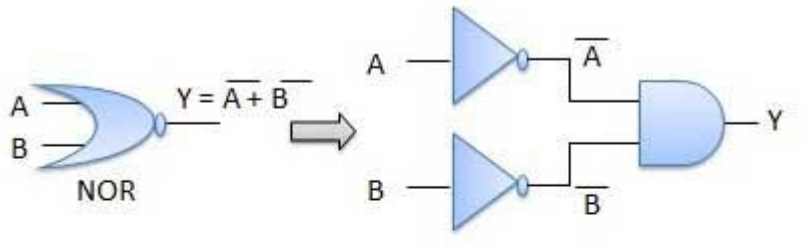
Theorem 2

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

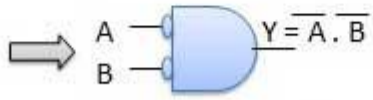
NOR = Bubbled AND

The LHS of this theorem represents a NOR gate with inputs A and B, whereas the RHS represents an AND gate with inverted inputs.

This AND gate is called as Bubbled AND.



NOR ☰ Bubbled AND



Bubbled AND

Table showing verification of the De Morgan's second theorem −

A	B	$\overline{A+B}$	$\overline{A}$	$\overline{B}$	$\overline{A}$. $\overline{B}$
0	0	1	1	1	1
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	0

K- MAPS

We have simplified the Boolean functions using Boolean postulates and theorems. It is a time consuming process and we have to re-write the simplified expressions after each step.

To overcome this difficulty, Karnaugh introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map

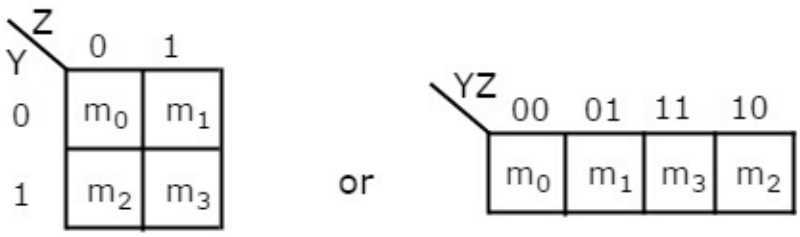
method. It is a graphical method, which consists of 2n cells for 'n' variables. The adjacent cells are differed only in single bit position.

K-Maps for 2 to 5 Variables

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

2 Variable K-Map

The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows 2 variable K-Map.

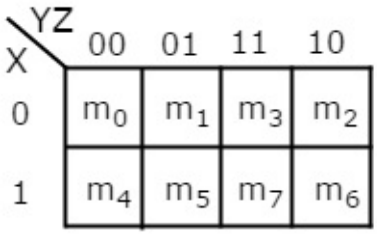


There is only one possibility of grouping 4 adjacent min terms.

The possible combinations of grouping 2 adjacent min terms are {(m0, m1), (m2, m3), (m0, m2) and (m1, m3)}.

3 Variable K-Map

The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows 3 variable K-Map.



There is only one possibility of grouping 8 adjacent min terms.

The possible combinations of grouping 4 adjacent min terms are {(m0, m1, m3, m2), (m4, m5, m7, m6), (m0, m1, m4, m5), (m1, m3, m5, m7), (m3, m2, m7, m6) and (m2, m0, m6, m4)}.

The possible combinations of grouping 2 adjacent min terms are {(m0, m1), (m1, m3), (m3, m2), (m2, m0), (m4, m5), (m5, m7), (m7, m6), (m6, m4), (m0, m4), (m1, m5), (m3, m7) and (m2, m6)}.

If x=0, then 3 variable K-map becomes 2 variable K-map.

4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows 4 variable K-Map.

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

There is only one possibility of grouping 16 adjacent min terms.

Let R1, R2, R3 and R4 represents the min terms of first row, second row, third row and fourth row respectively. Similarly, C1, C2, C3 and C4 represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are {(R1, R2), (R2, R3), (R3, R4), (R4, R1), (C1, C2), (C2, C3), (C3, C4), (C4, C1)}.

If w=0, then 4 variable K-map becomes 3 variable K-map.

5 Variable K-Map

The number of cells in 5 variable K-map is thirty-two, since the number of variables is 5. The following figure shows 5 variable K-Map.

V=0

WX \ YZ	00	01	11	10
00	m_0	m_1	m_3	m_2
01	m_4	m_5	m_7	m_6
11	m_{12}	m_{13}	m_{15}	m_{14}
10	m_8	m_9	m_{11}	m_{10}

V=1

WX \ YZ	00	01	11	10
00	m_{16}	m_{17}	m_{19}	m_{18}
01	m_{20}	m_{21}	m_{23}	m_{22}
11	m_{28}	m_{29}	m_{31}	m_{30}
10	m_{24}	m_{25}	m_{27}	m_{26}

There is only one possibility of grouping 32 adjacent min terms.

There are two possibilities of grouping 16 adjacent min terms. i.e., grouping of min terms from m0 to m15 and m16 to m31.

If v=0, then 5 variable K-map becomes 4 variable K-map.

In the above all K-maps, we used exclusively the min terms notation. Similarly, you can use exclusively the Max terms notation.

Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is '1', then we will get the Boolean function, which is instandard sum of products form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is '0', then we will get the Boolean function, which is in standard product of sums form after simplifying the K-map.

Follow these rules for simplifying K-maps in order to get standard sum of products form.

Select the respective K-map based on the number of variables present in the Boolean function.

If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.

Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.

Each grouping will give either a literal or one product term. It is known as prime implicant. The prime implicant is said to be essential prime implicant, if atleast single '1' is not covered with any other groupings but only that grouping covers.

Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

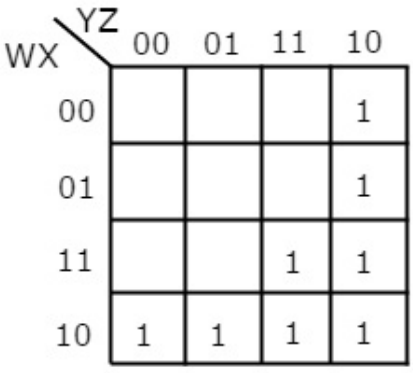
Note 1 − If outputs are not defined for some combination of inputs, then those output values will be represented with don't care symbol 'x'. That means, we can consider them as either '0' or '1'.

Note 2 − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as '1'.

Example

Let us simplify the following Boolean function, f(W, X, Y, Z)= WX'Y' + WY + W'YZ' using K-map.

The given Boolean function is in sum of products form. It is having 4 variables W, X, Y & Z. So, we require 4 variable K-map. The 4 variable K-map with ones corresponding to the given product terms is shown in the following figure.



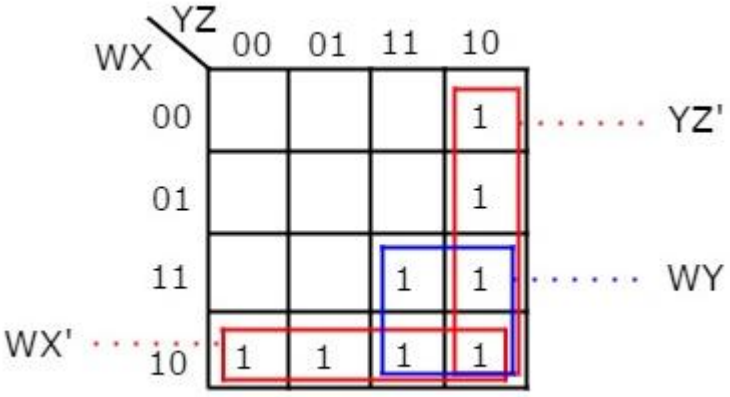
Here, 1s are placed in the following cells of K-map.

The cells, which are common to the intersection of Row 4 and columns 1 & 2 are corresponding to the product term, WX'Y'.

The cells, which are common to the intersection of Rows 3 & 4 and columns 3 & 4 are corresponding to the product term, WY.

The cells, which are common to the intersection of Rows 1 & 2 and column 4 are corresponding to the product term, W'YZ'.

There are no possibilities of grouping either 16 adjacent ones or 8 adjacent ones. There are three possibilities of grouping 4 adjacent ones. After these three groupings, there is no single one left as ungrouped. So, we no need to check for grouping of 2 adjacent ones. The 4 variable K-map with these three groupings is shown in the following figure.



Here, we got three prime implicants WX', WY & YZ'. All these prime implicants are essential because of following reasons.

Two ones (m8 & m9) of fourth row grouping are not covered by any other groupings. Only fourth row grouping covers those two ones.

Single one (m15) of square shape grouping is not covered by any other groupings. Only the square shape grouping covers that one.

Two ones (m2 & m6) of fourth column grouping are not covered by any other groupings. Only fourth column grouping covers those two ones.

Therefore, the simplified Boolean function is

f= WX' + WY + YZ'

Follow these rules for simplifying K-maps in order to get standard product of sums form.

Select the respective K-map based on the number of variables present in the Boolean function.

If the Boolean function is given as product of Max terms form, then place the zeroes at respective Max term cells in the K-map. If the Boolean function is given as product of sums form, then place the zeroes in all possible cells of K-map for which the given sum terms are valid.

Check for the possibilities of grouping maximum number of adjacent zeroes. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.

Each grouping will give either a literal or one sum term. It is known as prime implicant. The prime implicant is said to be essential prime implicant, if atleast single '0' is not covered with any other groupings but only that grouping covers.

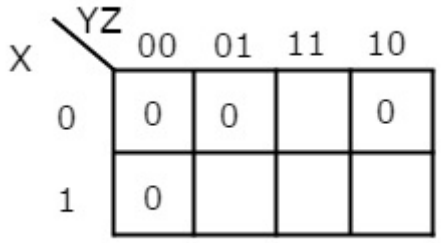
Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

Note − If don't care terms also present, then place don't cares 'x' in the respective cells of K-map. Consider only the don't cares 'x' that are helpful for grouping maximum number of adjacent zeroes. In those cases, treat the don't care value as '0'.

Example

Let us simplify the following Boolean function, f(X,Y,Z)=∏M(0,1,2,4)f(X,Y,Z)=∏M(0,1,2,4) using K-map.

The given Boolean function is in product of Max terms form. It is having 3 variables X, Y & Z. So, we require 3 variable K-map. The given Max terms are M0, M1, M2 & M4. The 3 variable K-mapwith zeroes corresponding to the given Max terms is shown in the following figure.



There are no possibilities of grouping either 8 adjacent zeroes or 4 adjacent zeroes. There are three possibilities of grouping 2 adjacent zeroes. After these three groupings, there is no single zero left as ungrouped. The 3 variable K-map with these threegroupings is shown in the following figure.

Here, we got three prime implicants X+Y, Y+Z & Z+X. All these prime implicants are essential because one zero in each grouping is not covered by any other groupings except with their individual groupings.

Therefore, the simplified Boolean function is

f=(X+Y).(Y+Z).(Z+X)

In this way, we can easily simplify the Boolean functions up to 5 variables using K-map method. For more than 5 variables, it is difficult to simplify the functions using K-Maps. Because, the number of cells in K-map gets doubled by including a new variable.

POSSIBLE QUESTIONS

EIGHT MARK QUESTIONS

1. Prove that (i) A(B+C) = AB + AC (ii) A+BC = (A+B) (A+C) .
2. Solve the following expression using 4 variable K map method
   (i) F(A,B,C,D) = Σ (1,2,4,7,11,13)
3. Explain NAND and NOR gates with neat diagram and their truth table.
4. Explain Demorgan's theorem
5. Prove that NAND and NOR universal building blocks.
6. simplify using K Maps
   (i) Y = F (A,B,C) = Σ (1,6,7)
   (ii) Y = F (A,B,C,D) = Σ (2,3,12,13,14,15)
7. NOR as an universal gate, explain with neat diagram and their truth table.
8. State the rules of Boolean algebra and explain?
9. Explain the Various logic gate with their truth table.
10. Simplify using K Maps
    (i) Y=F (A,B,C,D) = Σ (2,3,4,5,10,11,12,13,14,15)

UNIT-III
SYLLABUS

Flip-flop definitions; clocked flip-flop; S-R flip-flop: JK flip flop: T-flip flop D flip flop; master slave J-K flip flop: construction circuits.

Ring counter; Ripple counter and mod counters

FLIP FLOPS

A digital computer needs devices which can store information. flip flop is a binary storage device. It can store binary bit either 0 or 1. It has two stable states HIGH and LOW i.e. 1 and 0. It has the property to remain in one state indefinitely until it is directed by an input signal to switch over to the other state. It is also called bistable multivibrator.

The basic formation of flip flop is to store data. They can be used to keep a record or what value of variable (input, output or intermediate). Flip flop are also used to exercise control over the functionality of a digital circuit i.e. change the operation of a circuit depending on the state of one or more flip flops. These devices are mainly used in situations which require one or more of these three.

OPERATIONS, STORAGE AND SEQUENCING.

Flip flop is formed using logic gates, which are in turn made of transistors. Flip flop are basic building blocks in the memory of electronic devices. Each flip flop can store one bit of data. These are also called as sequential logic circuits. Also know these before learning about fliplfops.

Sequential Logic circuits

LATCHES

- Flip – flops have two stable states and hence they are bistable multivibrators. The two stable states are High (logic 1) and Low (logic 0).
- The term flip – flop is used as they can switch between the states under the influence of a control signal (clock or enable) i.e. they can ‘flip’ to one state and ‘flop’ back to other state.
- Flip – flops are a binary storage device because they can store binary data (0 or 1).

- Flip – flops are edge sensitive or edge triggered devices i.e. they are sensitive to the transition rather than the duration or width of the clock signal.
- They are also known as signal change sensitive devices which mean that the change in the level of clock signal will bring change in output of the flip flop.
- A Flip – flop works depending on clock pulses.
- Flip flops are also used to control the digital circuit's functionality. They can change the operation of a digital circuit depending on the state.
- Some of the most common flip – flops are SR Flip – flop (Set – Reset), D Flip – flop (Data or Delay), JK Flip – flop and T Flip – flop.

Latches vs Flip-Flops

Latches and flip – flops are both 1 – bit binary data storage devices. The main difference between a latch and a flip – flop is the triggering mechanism. Latches are transparent when enabled , whereas flip – flops are dependent on the transition of the clock signal i.e. either positive edge or negative edge.

The modern usage of the term flip – flop is reserved to clocked devices and term latch is to describe much simpler devices. Some of the other differences between latches and flip – flops are listed in below table.

LATCH	FLIP – FLOP
Latches do not require clock signal.	Flip – flops have clock signals
A latch is an asynchronous device.	A flip – flop is a synchronous device.
Latches are transparent devices i.e. when they are enabled, the output changes immediately if the input changes.	A transition from low to high or high to low of the clock signal will cause the flip – flop to either change its output or retain it depending on the input signal.
A latch is a Level Sensitive device (Level Triggering is involved).	A flip – flop is an edge sensitive device (Edge Triggering is involved).
Latches are simpler to design as there is no clock signal (no careful routing of clock signal is required).	When compare to latches, flip – flops are more complex to design as they have clock signal and it has to be carefully routed. This is because all the flip – flops in a design should have a clock signal and the delay in the clock reaching each flip – flop must be minimum or negligible.
The operation of a latch is faster as they do not have to wait for any clock signal.	Flip - flops are comparatively slower than latches due to clock signal.
The power requirement of a latch is less.	Power requirement of a flip – flop is more.
A latch works based on the enable signal.	A flip – flop works based on the clock signal.

TYPES OF FLIP FLOPS

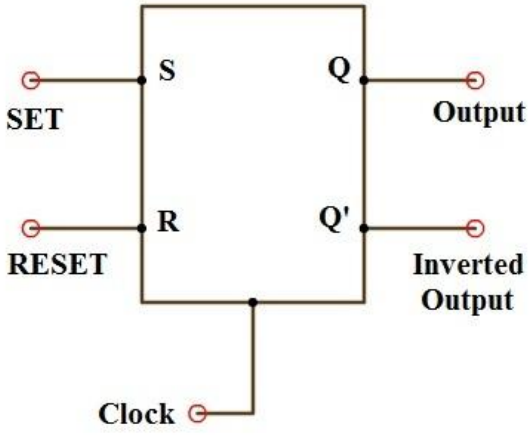
Based on their operations, flip flops are basically 4 types. They are

1. R-S flip flop
2. D flip flop
3. J-K flip flop
4. T flip flop

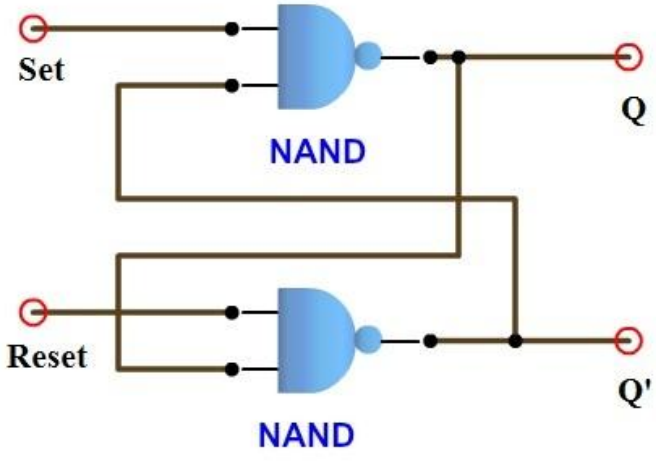
S-R Flip Flop

The S-R flip-flop is basic flip-flop among all the flip-flops. All the other flip flops are developed after SR-flip-flop.

SR flip flop is represented as shown below.



- S-R stands for SET and RESET. This can also be called RS flip-flop. Difference is RS is inverted SR flip-flop.
- Any flip flop can be build using logic gates. NAND and NOR gates were used as they are universal gates.
- Here is the SR flip-flop using NAND gates.

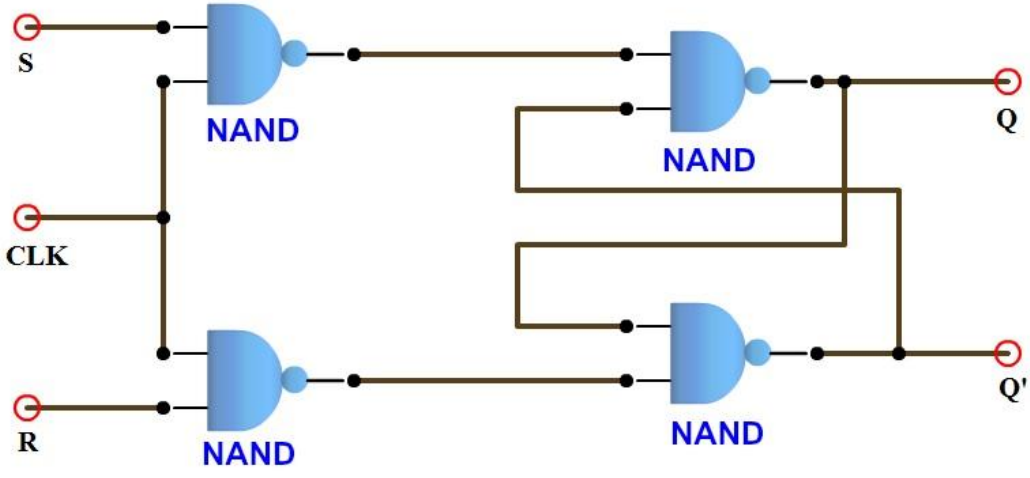


Truth Table of SR Flip Flop

Sno	S	R	Q	Q'	State
1	1	0	1	0	Q is set to 1
2	1	1	1	0	No change
3	0	1	0	1	Q' is set to 1
4	1	1	0	1	No change
5	0	0	1	1	Invalid

Working

- From the above truth table it is clear that SR flip flop will be set or reset for four conditions.
- For last condition it will be in invalid state.
- SR Flip-flop will be set when S=1 and R=0, if S=1 and R=1 then previous state is remembered by the flip flop.
- Flip-flop will be reset when S=0 and R=1, if S=1 and R=1, then it will remember the previous state.
- But when both the inputs are zeros, SR Flip flop will be in an uncertain state where both Q and Q' will be same. This is not same allowed..
- This is indeterminate state is avoided by adding gates extra gates to the existing flip flop. This is called clocked or gated SR Flip flop. This produces the output only for the High clock pulse.
- The circuit of a clocked SR flip – flop using NAND gates is shown below.



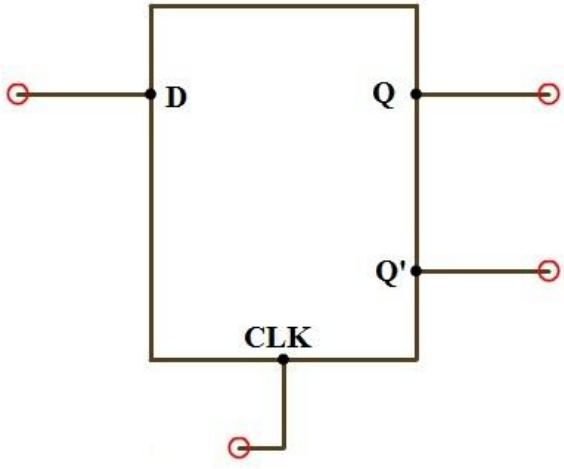
Know in detail about SR Flip FlopD flip flop

D FLIP FLOP

In the SR flip flop an uncertain state occurred. This can be avoided by using D flip flop. Here D stands for "Data".

It is constructed from SR flip flop. The two inputs (S &R) of the clocked SR flip flop are connected to an inverter.

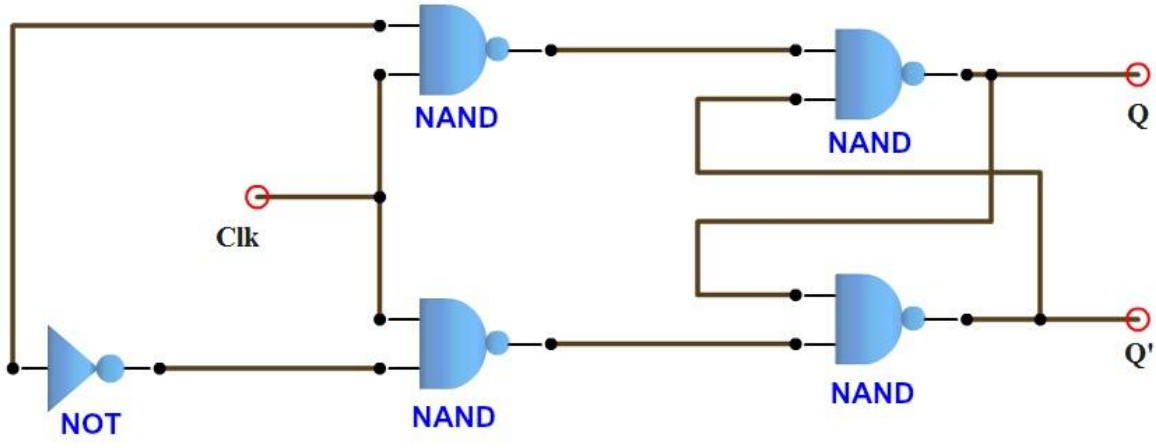
It is one of the most widely used flip – flops. It has a clock signal (Clk) as one input and Data (D) as other. There are two outputs and these outputs are complement to each other. The symbol of D flip – flop is shown below.



Truth table

Clk	D	Q	Q'	State
0	0	Q	Q'	No change in state
1	0	0	1	Resets Q to 0
1	1	1	1	Sets Q to 1

D flip – flop using NAND gates is shown below.



WORKING

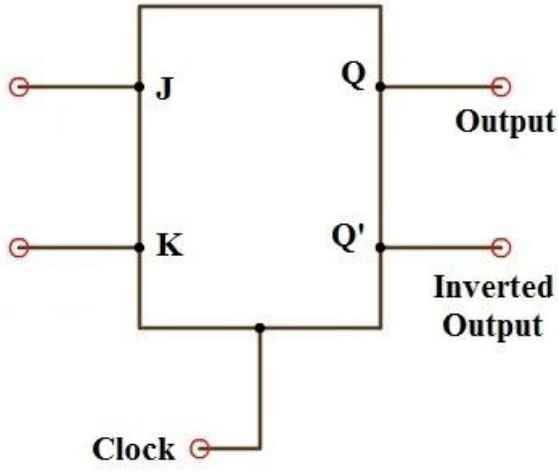
- D flip flop will work depending on the clock signal.
- When the clock is low there will be no change in the output of the flip flop i.e. it remembers the previous state.
- When the clock signal is high and if it receives any data on its data pin, it Changes the state of output.
- When data is high Q reset to 0,while Q is set to 0 if data is low.
- A master slave D flip flop can be constructed using D-flip flop.
- Know in detail about D-flip flop.

J-K FLIP FLOP

- JK flip – flop is named after Jack Kilby, an electrical engineer who invented IC.
- A JK flip – flop is a modification of SR flip – flop. In this the J input is similar to the set input of SR flip – flop and the K input is similar to the reset input of SR flip – flop. The condition J = K = 1 which is not allowed in SR flip – flop (S = R = 1) is interpreted as a toggle command.
- The JK flip flop has

- Two data inputs J and K.
- One clock signal input (CLK).
- Two outputs Q and Q'.

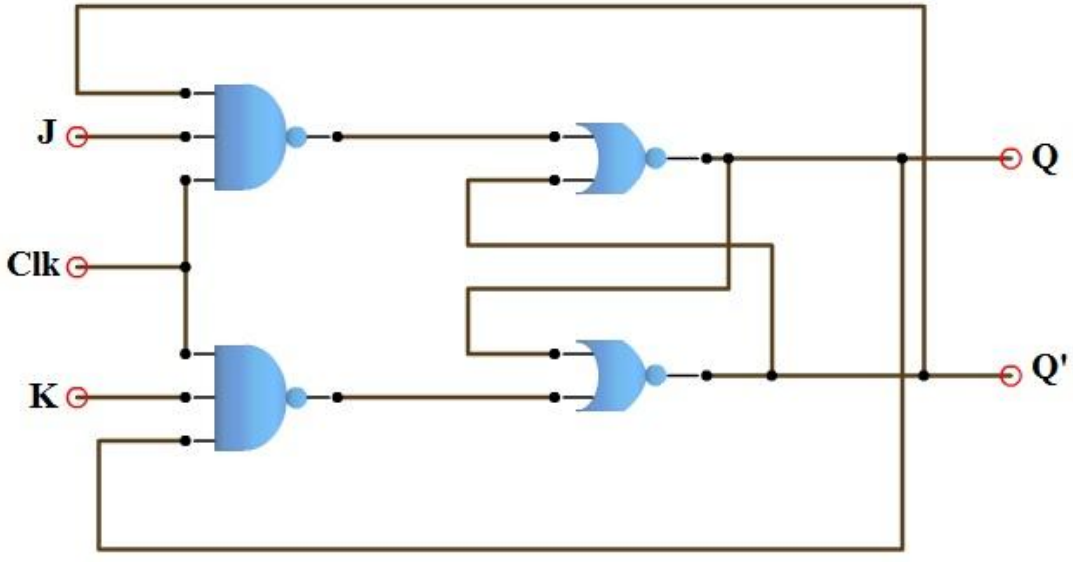
The symbol of a JK flip – flop is shown below.



Truth Table

Clk	J	K	Q	Q'	State
1	0	0	Q	Q'	No change in state
1	0	1	0	1	Resets Q to 0
1	1	0	1	0	Sets Q to 1
1	1	1	-	-	Toggles

The circuit of a JK flip – flop using gates is shown below. It is similar to a modified NAND SR flip – flop.

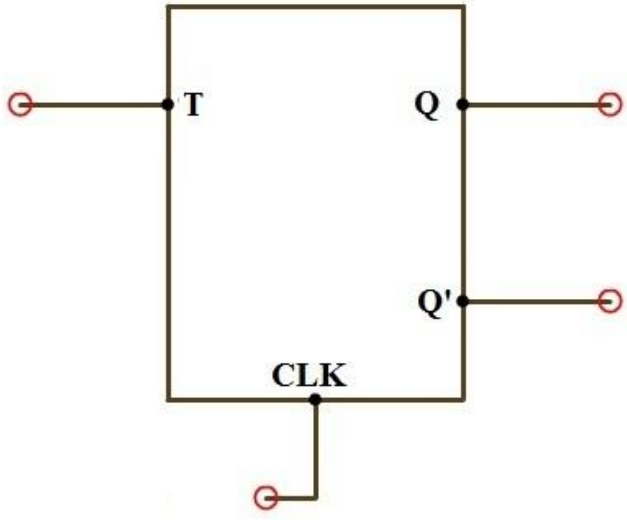


WORKING

- When J is low and K is low, then Q returns its previous state value i.e. it holds the current state.
- When J is low and K is high, then flip – flop will be in reset state i.e. Q = 0, Q' =1.
- When J is high and K is low then flip – flop will be in set state i.e. Q = 1, Q' =0.
- When J is high and K is high then flip – flop will be in Toggle state or flip state. This means that the output will complement to the previous state value.

T FLIP FLOP

- T flip flop is also known as "Toggle Flip – flop". Toggle is to change the output to complement of the previous state in the presence of clock input signal.
- The T flip flop has
- T input.
- One clock signal input (CLK).
- Two outputs Q and Q'.
- The symbol of a T flip – flop is shown below.

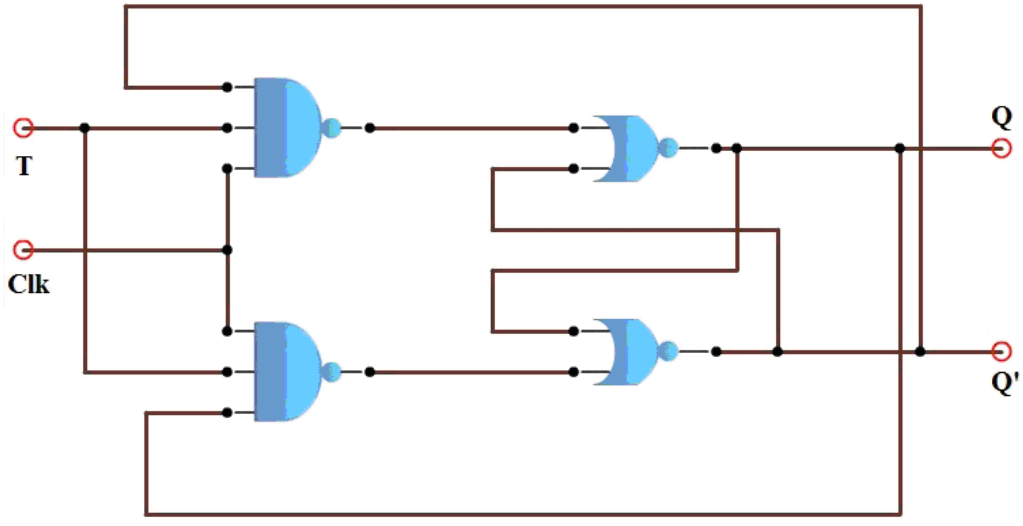


- We can construct a T flip – flop by using any other flip – flops.
- SR flip – flop: By connecting the feedback of outputs of SR flip – flop to the inputs (S & R).
- D flip – flop: Connecting the Q' to its Data input of D flip – flop as feedback path.
- J K flip – flop: By combing the J & K inputs of JK flip – flop, to make as single input, we can design the T flip – flop.

Truth Table

T	Q	Q'
0	0	0
1	0	1
0	1	0
1	1	0

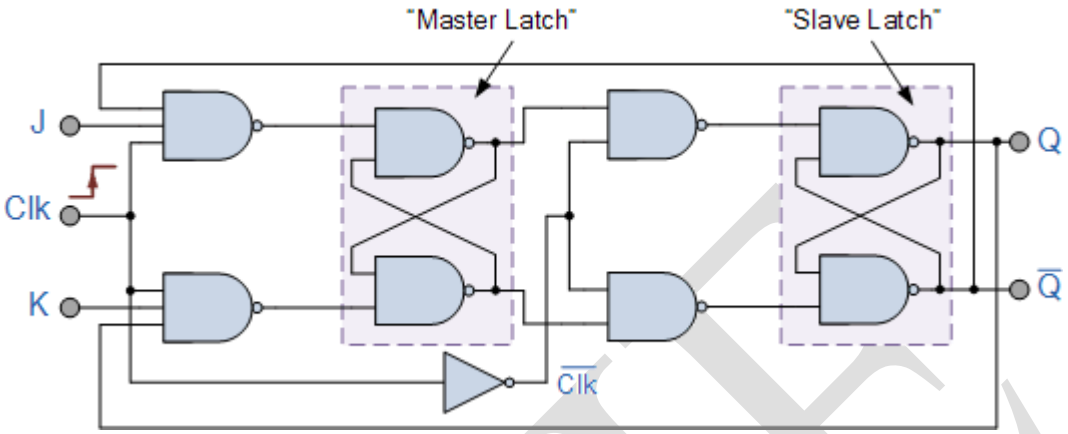
The circuit of a T flip – flop made from NAND JK flip – flop is shown below.



WORKING

- The operation of the T flip flop is explained below.
- When the T input is low, then the next sate of the T flip – flop is same as the present state i.e. it holds the current state.
- T = 0 and present state = 0 then the next state = 0.
- T = 0 and present state = 1 then the next state = 1.
- When the T input is high, then the next sate of the T flip – flop is toggled i.e. it is same as the complement of present state on clock transition.
- T = 1 and present state = 0 then the next state = 1.
- T = 1 and present state = 1 then the next state = 0.

The Master-Slave JK Flip Flop



The input signals J and K are connected to the gated "master" SR flip flop which "locks" the input condition while the clock (Clk) input is "HIGH" at logic level "1". As the clock input of the "slave" flip flop is the inverse (complement) of the "master" clock input, the "slave" SR flip flop does not toggle. The outputs from the "master" flip flop are only "seen" by the gated "slave" flip flop when the clock input goes "LOW" to logic level "0".

When the clock is "LOW", the outputs from the "master" flip flop are latched and any additional changes to its inputs are ignored. The gated "slave" flip flop now responds to the state of its inputs passed over by the "master" section.

Then on the "Low-to-High" transition of the clock pulse the inputs of the "master" flip flop are fed through to the gated inputs of the "slave" flip flop and on the "High-to-Low" transition the same inputs are reflected on the output of the "slave" making this type of flip flop edge or pulse-triggered.

Then, the circuit accepts input data when the clock signal is "HIGH", and passes the data to the output on the falling-edge of the clock signal. In other words, the Master-Slave JK Flip flop is a "Synchronous" device as it only passes data with the timing of the clock signal.

In the next tutorial about Sequential Logic Circuits, we will look atMultivibrators that are used as waveform generators to produce the clock signals to switch sequential circuits.

Ring Counter

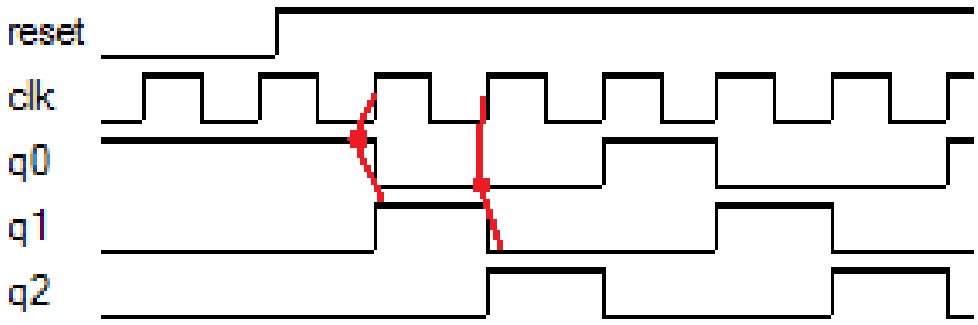
A ring counter is a Shift Register (a cascade connection of flip-flops) with the output of the last flip flop connected to the input of the first. It is initialised such that only one of the flip flop output is 1 while the remander is 0. The 1 bit is circulated so the state repeats every n clock cycles if n flip-flops are used. The "MOD" or "MODULUS" of a counter is the number of unique states. The MOD of the n flip flop ring counter is n.

It can be implemented using D-type flip-flops (or JK-type flip-flops).

010000

Notes:

- Enable the flips flops by clicking on the RESET (Green) switch. The RESET switch is a on/off switch (similar to a room light switch)
- Click on CLK (Red) switch and observe the changes in the outputs of the flip flops. The CLK switch is a momentary switch (similar to a door bell switch - normally off).
- The D flip flop clock has a rising edge CLK input. For example Q1 behaves as follows:
- The D input value just before the CLK rising edge is noted (Q0).
- When CLK rising edge occurs, Q1 is assigned the previously noted D value (Q0).



The MOD or number of unique states of this 3 flip flop ring counter is 3.

Truth Table

State	Q0	Q1	Q2

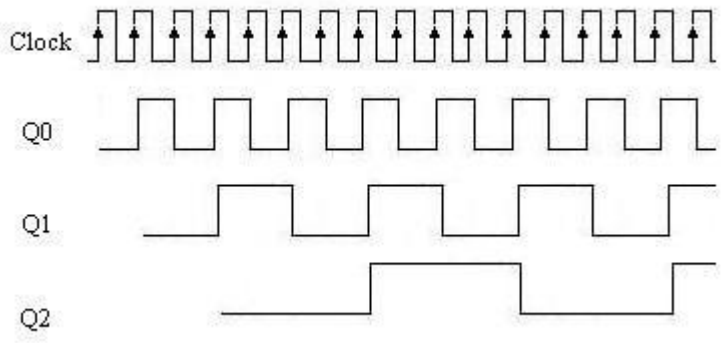
0	1	0	0
1	0	1	0
2	0	0	1

Simulate and Breadboard the Ring Counter circuit.

Ripple Counter

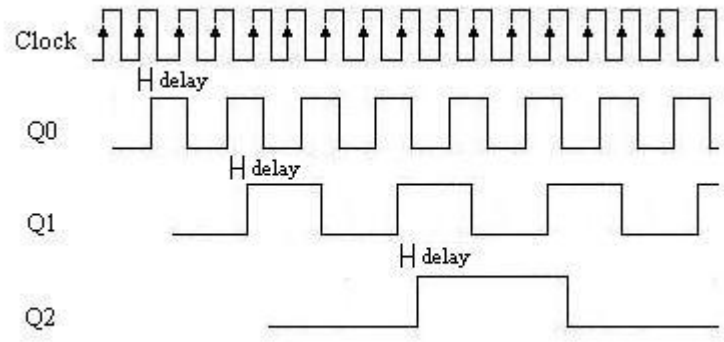
A good first thought for making counters that can count higher is to chain Divide-by-2 counters together. We can feed the Q out of one flop into the CLK of the next stage. The result looks something like this:

The ripple counter is easy to understand. Each stage acts as a Divide-by-2 counter on the previous stage's signal. The Q out of each stage acts as both an output bit, and as the clock signal for the next stage.



We can chain as many ripple counters together as we like. A three bit ripple counter will count 23=8 numbers, and an n-bit ripple counter will cound 2n numbers.

The problem with ripple counters is that each new stage put on the counter adds a delay. This propagation delay is seen when we look at a less idealized timing diagram:



Now we can see that the propagation delay does not only slow down the counter, but it actually introduces errors into the system. These errors increase as we add additional stages to the ripple counter.

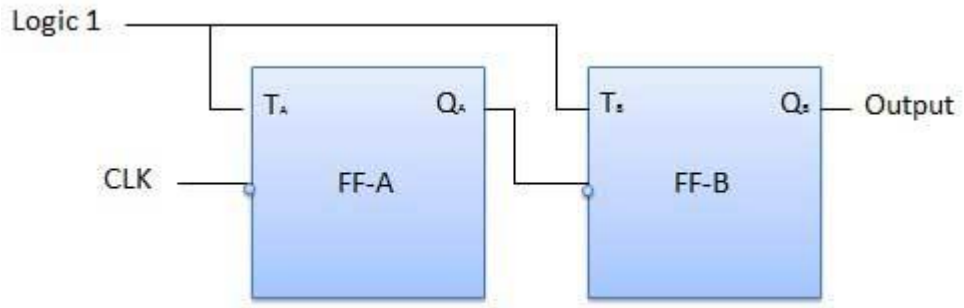
Counter is a sequential circuit. A digital circuit which is used for a counting pulses is known counter. Counter is the widest application of flip-flops. It is a group of flip-flops with a clock signal applied. Counters are of two types.

- Asynchronous or ripple counters.
- Synchronous counters.

Asynchronous or ripple counters

The logic diagram of a 2-bit ripple up counter is shown in figure. The toggle (T) flip-flop are being used. But we can use the JK flip-flop also with J and K connected permanently to logic 1. External clock is applied to the clock input of flip-flop A and QA output is applied to the clock input of the next flip-flop i.e. FF-B.

Logical Diagram



Operation

S.N.	Condition	Operation
1	Initially let both the FFs be in the reset state	QBQA = 00 initially
2	After 1st negative clock edge	As soon as the first negative clock edge is applied, FF-A will toggle and QA will be equal to 1. QA is connected to clock input of FF-B. Since QA has changed from 0 to 1, it is treated as the positive clock edge by FF-B. There is no change in QB because FF-B is a negative edge triggered FF. QBQA = 01 after the first clock pulse.
3	After 2nd negative clock edge	On the arrival of second negative clock edge, FF-A toggles again and QA = 0. The change in QA acts as a negative clock edge for FF-B. So it

		will also toggle, and QB will be 1. QBQA = 10 after the second clock pulse.
4	After 3rd negative clock edge	On the arrival of 3rd negative clock edge, FF-A toggles again and QA become 1 from 0. Since this is a positive going change, FF-B does not respond to it and remains inactive. So QB does not change and continues to be equal to 1. QBQA = 11 after the third clock pulse.
5	After 4th negative clock edge	On the arrival of 4th negative clock edge, FF-A toggles again and QA becomes 1 from 0. This negative change in QA acts as clock pulse for FF-B. Hence it toggles to change QBfrom 1 to 0. QBQA = 00 after the fourth clock pulse.

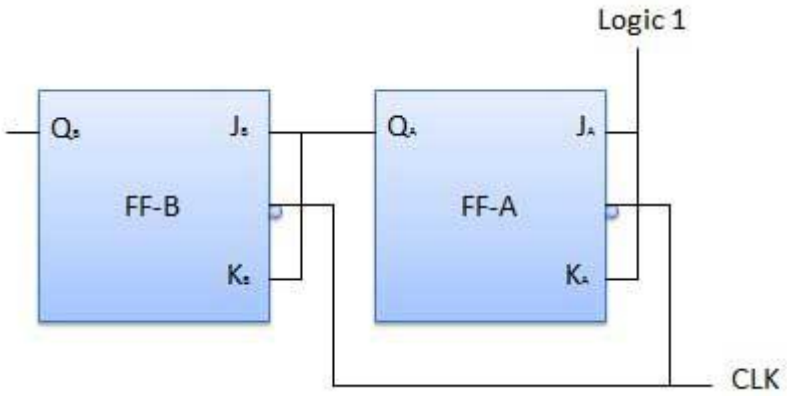
Truth Table

SYNCHRONOUS COUNTERS

- If the "clock" pulses are applied to all the flip-flops in a counter simultaneously, then such a counter is called as synchronous counter.
- 2-bit Synchronous up counter

- The JA and KA inputs of FF-A are tied to logic 1. So FF-A will work as a toggle flip-flop. The JB and KB inputs are connected to QA.

Logical Diagram



Operation

S.N.	Condition	Operation
1	Initially let both the FFs be in the reset state	QBQA = 00 initially.
2	After 1st negative clock edge	As soon as the first negative clock edge is applied, FF-A will toggle and QA will change from 0 to 1. But at the instant of application of negative clock edge, QA , JB = KB = 0. Hence FF-B will not change its state. So QB will remain 0. QBQA = 01 after the

		first clock pulse.
3	After 2nd negative clock edge	On the arrival of second negative clock edge, FF-A toggles again and QA changes from 1 to 0. But at this instant QAwas 1. So JB = KB= 1 and FF-B will toggle. Hence QB changes from 0 to 1. QBQA = 10 after the second clock pulse.
4	After 3rd negative clock edge	On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B. QBQA = 11 after the third clock pulse.
5	After 4th negative clock edge	On application of the next clock pulse, QA will change from 1 to 0 as QB will also change from 1 to 0. QBQA = 00 after the

		fourth clock pulse.

Classification of counters

Depending on the way in which the counting progresses, the synchronous or asynchronous counters are classified as follows −

- Up counters
- Down counters
- Up/Down counters

UP/DOWN Counter

Up counter and down counter is combined together to obtain an UP/DOWN counter. A mode control (M) input is also provided to select either up or down mode. A combinational circuit is required to be designed and used between each pair of flip-flop in order to achieve the up/down operation.

Type of up/down counters

- UP/DOWN ripple counters
- UP/DOWN synchronous counter

UP/DOWN Ripple Counters

In the UP/DOWN ripple counter all the FFs operate in the toggle mode. So either T flip-flops or JK flip-flops are to be used. The LSB flip-flop receives clock directly. But the clock to every other FF is obtained from (Q = Q bar) output of the previous FF.

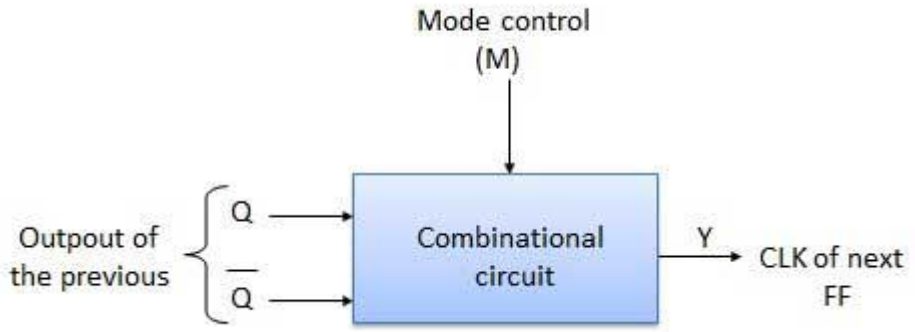
UP counting mode (M=0) − The Q output of the preceding FF is connected to the clock of the next stage if up counting is to be achieved. For this mode, the mode select input M is at logic 0 (M=0). DOWN counting mode (M=1) − If M = 1, then the Q bar output of the preceding FF is connected to the next FF. This will operate the counter in the counting mode.

Example

- 3-bit binary up/down ripple counter.
- 3-bit − hence three FFs are required.
- UP/DOWN − So a mode control input is essential.

- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple up counter, the Q output of preceding FF is connected to the clock input of the next one.
- For a ripple down counter, the Q bar output of preceding FF is connected to the clock input of the next one.
- Let the selection of Q and Q bar output of the preceding FF be controlled by the mode control input M such that, If M = 0, UP counting. So connect Q to CLK. If M = 1, DOWN counting. So connect Q bar to CLK.

Block Diagram



Truth Table

Inputs			Outputs	
M	Q	$\overline{Q}$	Y	
0	0	0	0	Y = Q for up counter
0	0	1	0	
0	1	0	1	
0	1	1	1	
1	0	0	0	$Y = \overline{Q}$ for up counter
1	0	1	1	
1	1	0	0	
1	1	1	1	

Operation

S.N.	Condition	Operation
1	Case 1 − With M = 0 (Up counting mode)	If M = 0 and M bar = 1, then the AND gates 1 and 3 in fig. will be enabled whereas the AND gates 2 and 4 will be disabled. Hence QA gets connected to the clock input of FF-B and QBgets connected to the clock input of FF-C. These connections are same as those for the normal up counter. Thus with M = 0 the circuit work as an up counter.
2	Case 2: With M = 1 (Down counting mode)	If M = 1, then AND gates 2 and 4 in fig. are enabled whereas the AND gates 1 and 3 are disabled. Hence QA bar gets connected to the clock input of FF-B and QBbar gets connected to the clock input of FF-C. These connections will produce a down counter. Thus with M = 1 the circuit works as a down counter.

MODULUS COUNTER (MOD-N COUNTER)

The 2-bit ripple counter is called as MOD-4 counter and 3-bit ripple counter is called as MOD-8 counter. So in general, an n-bit ripple counter is called as modulo-N counter. Where, MOD number = 2n.

Type of modulus

- 2-bit up or down (MOD-4)
- 3-bit up or down (MOD-8)
- 4-bit up or down (MOD-16)

APPLICATION OF COUNTERS

- Frequency counters
- Digital clock
- Time measurement
- A to D converter
- Frequency divider circuits
- Digital triangular wave generator.

POSSIBLE QUESTIONS

EIGHT MARK QUESTIONS

1. Explain the working a T-Filp flop and give its truth table.
2. Explain the working of JK master slave filp flop
3. Explain the working of SR filp flop using NOR gate with neat diagram.
4. Explain the Mod 8 counter.
5. With NAND gates, construct a D-filp flop and discuss its working.
6. Write about the counters and explain mod-2 counter with neat diagram.
7. Explain the working of a clocked SR filp flop using NAND gate
8. Discuss about the mod-16 counter.
9. Explain the working of JK filp flop in detail manner with neat diagram and truth table.
10. Draw the circuit of ring counter and discuss its working.

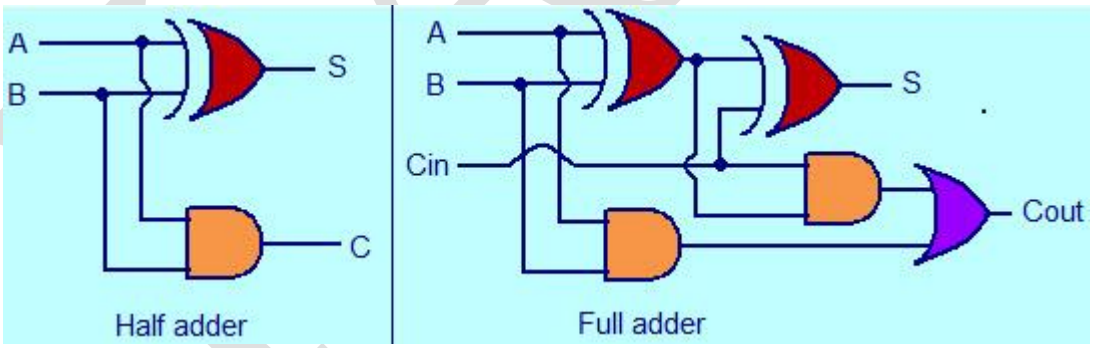
UNIT-IV
SYLLABUS

XOR gates half adder - Full adder - Full subtracter - Parallel binary adder - Parallel binary subtracter - Construction, action and truth table.

Magnetic tape - Magnetic disc - Floppy disc - Magnetic cores - Magnetic core logic, coincident memory - Memory addressing. MOS - random access memory - MOD read only memory PROM-EPROM.

Half Adder and Full Adder with Truth Table

An adder is a digital logic circuit in electronics that implements addition of numbers. In many computers and other types of processors, adders are used to calculate addresses, similar operations and table indices in the ALU and also in other parts of the processors. These can be built for many numerical representations like excess-3 or binary coded decimal. Adders are classified into two types: half adder and full adder. The half adder circuit has two inputs: A and B, which add two input digits and generate a carry and sum. The full adder circuit has three inputs: A and C, which add the three input numbers and generate a carry and sum. This article gives brief information about half adder and full adder in tabular forms and circuit diagrams.



Half Adder and Full Adder

Half Adder and Full Adder Circuit

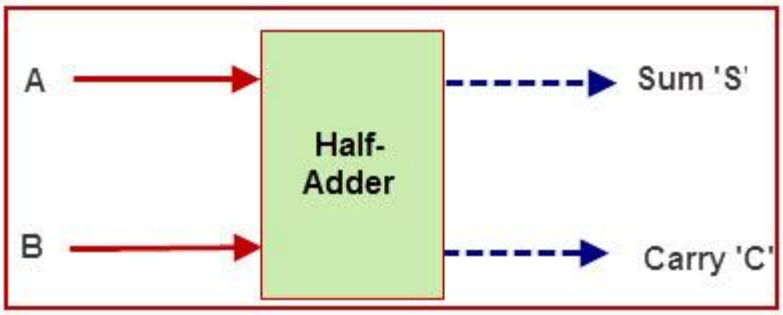
An adder is a digital circuit that performs addition of numbers. The half adder adds two binary digits called as augend and addend and produces two outputs as sum and carry; XOR is applied to both inputs to produce sum and AND gate is applied to both inputs to produce carry. The full adder adds 3

one bit numbers, where two can be referred to as operands and one can be referred to as bit carried in. And produces 2-bit output, and these can be referred to as output carry and sum.

Half Adder

By using half adder, you can design simple addition with the help of logic gates.

Let's see an addition of single bits.



Half Adder

0+0 = 0

0+1 = 1

1+0 = 1

1+1 = 10

These are the least possible single-bit combinations. But the result for 1+1 is 10, the sum result must be re-written as a 2-bit output. Thus, the equations can be written as

0+0 = 00

0+1 = 01

1+0 = 01

1+1 = 10

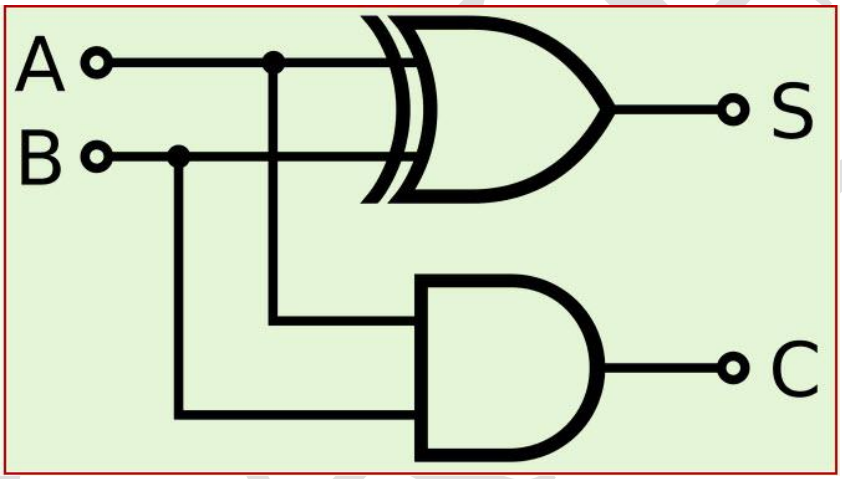
The output '1'of '10' is carry-out. 'SUM' is the normal output and 'CARRY' is the carry-out.

Half Adder Truth Table

INPUTS		OUTPUTS	
A	B	SUM	CARRY
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Half Adder Truth Table

Now it has been cleared that 1-bit adder can be easily implemented with the help of the XOR Gate for the output 'SUM' and an AND Gate for the 'Carry'. When we need to add, two 8-bit bytes together, we can be done with the help of a full-adder logic. The half-adder is useful when you want to add one binary digit quantities. A way to develop a two-binary digit adders would be to make a truth table and reduce it. When you want to make a three binary digit adder, do it again. When you decide to make a four digit adder, do it again. The circuits would be fast, but development time is slow.

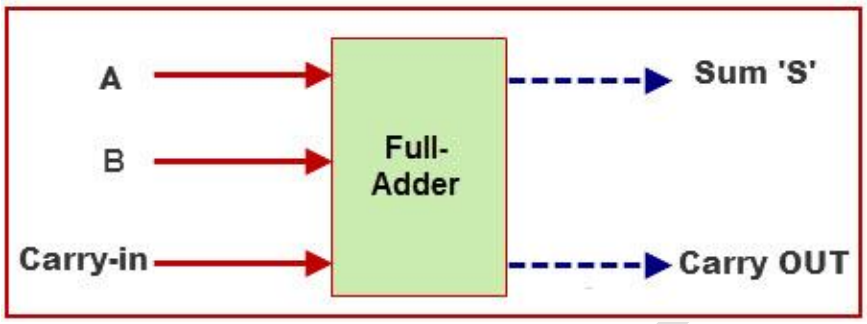


Half Adder Logic Circuit

The simplest expression uses the exclusive OR function: Sum=AÅB. An equivalent expression in terms of the basic AND, OR, and NOT is: SUM=A|.B+A.B'

Full Adder

This adder is difficult to implement than a half-adder. The difference between a half-adder and a full-adder is that the full-adder has three inputs and two outputs, whereas half adder has only two inputs and two outputs. The first two inputs are A and B and the third input is an input carry as C-IN. When a full-adder logic is designed, you string eight of them together to create a byte-wide adder and cascade the carry bit from one adder to the next.



Full Adder

The output carry is designated as C-OUT and the normal output is designated as S.

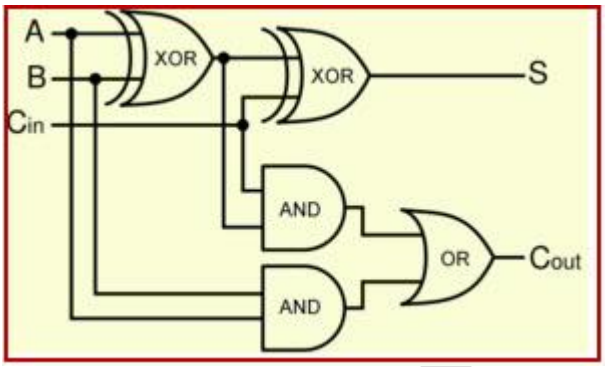
Full Adder Truth Table:

INPUTS			OUTPUT	
A	B	C-IN	C-OUT	S
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

Full Adder Truth Table

With the truth-table, the full adder logic can be implemented. You can see that the output S is an XOR between the input A and the half-adder, SUM output with B and C-IN inputs. We take C-OUT will only be true if any of the two inputs out of the three are HIGH.

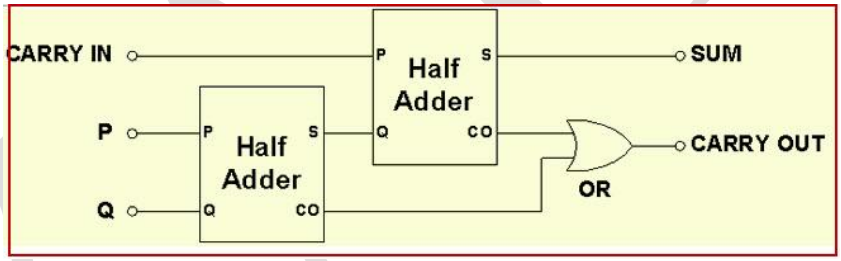
So, we can implement a full adder circuit with the help of two half adder circuits. At first, half adder will be used to add A and B to produce a partial Sum and a second half adder logic can be used to add C-IN to the Sum produced by the first half adder to get the final S output.



Full Adder Logic Circuit

If any of the half adder logic produces a carry, there will be an output carry. So, COUT will be an OR function of the half-adder Carry outputs. Take a look at the implementation of the full adder circuit shown below.

The implementation of larger logic diagrams is possible with the above full adder logic a simpler symbol is mostly used to represent the operation. Given below is a simpler schematic representation of a one-bit full adder.



Full Adder Design Using Half Adders

With this type of symbol, we can add two bits together, taking a carry from the next lower order of magnitude, and sending a carry to the next higher order of magnitude. In a computer, for a multi-bit operation, each bit must be represented by a full adder and must be added simultaneously. Thus, to add two 8-bit numbers, you will need 8 full adders which can be formed by cascading two of the 4-bit blocks.

Combinational circuit combines the different gates in the circuit for example encoder, decoder, multiplexer and demultiplexer. Characteristics of combinational circuits are as follows.

The output at any instant of time, depends only on the levels present at input terminals.

It does not use any memory. The previous state of input does not have any effect on the present state of the circuit.

It can have a number of inputs and m number of outputs.

The relationship between the Full-Adder and the Half-Adder is half adder produces results and full adder uses half adder to produce some other result. Similarly, while the Full-Adder is of two Half-Adders, the Full-Adder is the actual block that we use to create the arithmetic circuits.

INTRODUCTION TO MEMORY DEVICES

Computers and many electronic gadgets usually rely on stored information which is mainly data which can be used to direct circuit actions. The digital information is stored in memory devices. Memories can be divided into 2 categories based on what memory cells can be accessed at a given instant.

SAM (Sequentially Access Memory) is accessed by stepping through each memory location until the desired location is reached. Magnetic tape is an example of SAM.

The second category of memory devices is called RAM (Random Access Memory) where the memory can be randomly accessed at any instant, without having to step through each memory location. It is generally faster to access a RAM compared to SAM. Most of the electronics gadgets memory are of

RAM TYPE.

Random Access Memory (RAM) Memory Device

RAM memory is "volatile" which means that the information stored in the RAM will be lost once the power to it is removed. Two common types of RAM are DRAM (Dynamic RAM) and SRAM (Static RAM). DRAM stored a bit as the presence or absence of charge on MOSFET gate substrate capacitance.

As the capacitance has leakage, it must be refreshed every few miliseconds. SRAM is an array of flip flops of which the bit stored in the flip flop will remain until power is removed or another bit replaces it. SRAM does not need to be refreshed. DRAM is usually 1.5 to 4 times as dense as SRAM and hence cheaper. However, SRAM has faster access times than DRAM.

Read Only Memory (ROM) Memory Device

ROM is non volatile in that its contents are not lost when power to it is removed. All ROMs can be programmed at least once. Mask ROMs are programmed by having "1"s and "0"s etched into their semiconductors at the time of manufacturing.

Programmable ROM (PROM) can be written after manufacturing by electrically burning specific transistors or diodes in the memory array. EPROM can be erased and reprogrammed by using ultraviolet light.

EEPROM (electronically erasable PROM) data can be erased electronically but it takes longer time compared to RAM. Read and write time for RAM is almost the same but PROM has slower write times. PROM must be erased before they can be reprogrammed and it often needs a higher programming voltage than its operating voltage.

ROM is usually used to store data or programs that do not change frequently and must still be there when power supply cuts off.

Non Volatile RAM (NVRAM) Memory Device

A NVRAM chip consists of both RAM and ROM. During power on reset, the contents of the ROM are copied to RAM. Before the power turns off, the system will copy the entire contents of the RAM into ROM for non volatile storage. The RAM in an NVRAM is called shadow RAM. NVRAM fills the gap between easily written memory and non volatile memory.

POSSIBLE QUESTIONS

EIGHT MARK

1. Write short note on Half adder and Full subtractor
2. Explain Parallel binary adder with neat diagram.
3. Explain briefly about magnetic core logic and magnetic cores.
4. Write short note on Full adder and Full subtractor
5. write short note on RAM, EPROM and ROM
6. Explain briefly about half adder and full adder
7. Explain briefly about Magnetic Disc and Floppy Disc
8. Write short note on RAM ,PROM, EPRM
9. Write short note on Parallel binary adder and Parallel subtractor

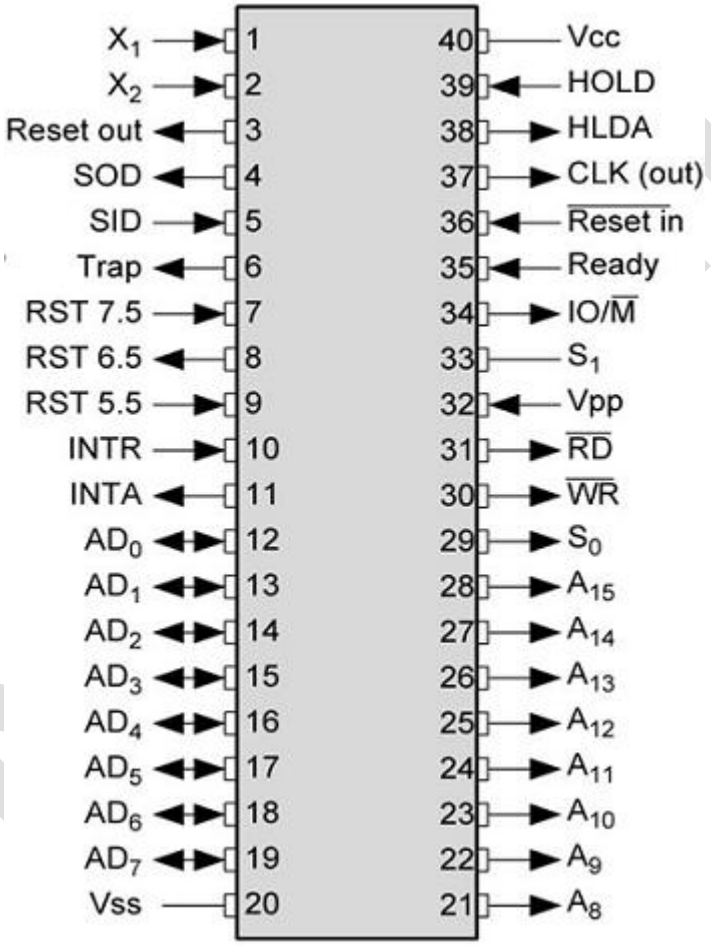
UNIT-V

SYLLABUS

Brief history, organization of 8085 - Data and address bus, addressing the I/O devices, registers in the 8085, instruction set - Instruction types, and classification of instruction, simple programs.

BRIEF HISTORY, ORGANIZATION OF 8085

The following image depicts the pin diagram of 8085 Microprocessor −



The pins of a 8085 microprocessor can be classified into seven groups −

ADDRESS BUS

A15-A8, it carries the most significant 8-bits of memory/IO address.

DATA BUS

AD7-AD0, it carries the least significant 8-bit address and data bus.

CONTROL AND STATUS SIGNALS

These signals are used to identify the nature of operation. There are 3 control signal and 3 status signals.

Three control signals are RD, WR & ALE.

- **RD** − This signal indicates that the selected IO or memory device is to be read and is ready for accepting data available on the data bus.
- **WR** − This signal indicates that the data on the data bus is to be written into a selected memory or IO location.
- **ALE** − It is a positive going pulse generated when a new operation is started by the microprocessor. When the pulse goes high, it indicates address. When the pulse goes down it indicates data.

THREE STATUS SIGNALS ARE IO/M, S0 & S1.

IO/M

This signal is used to differentiate between IO and Memory operations, i.e. when it is high indicates IO operation and when it is low then it indicates memory operation.

S1 & S0

These signals are used to identify the type of current operation.

POWER SUPPLY

There are 2 power supply signals − VCC & VSS. VCC indicates +5v power supply and VSS indicates ground signal.

CLOCK SIGNALS

There are 3 clock signals, i.e. X1, X2, CLK OUT.

- **X1, X2** − A crystal (RC, LC N/W) is connected at these two pins and is used to set frequency of the internal clock generator. This frequency is internally divided by 2.
- **CLK OUT** − This signal is used as the system clock for devices connected with the microprocessor.

Interrupts & externally initiated signals

Interrupts are the signals generated by external devices to request the microprocessor to perform a task. There are 5 interrupt signals, i.e. TRAP, RST 7.5, RST 6.5, RST 5.5, and INTR. We will discuss interrupts in detail in interrupts section.

- **INTA** − It is an interrupt acknowledgment signal.
- **RESET IN** − This signal is used to reset the microprocessor by setting the program counter to zero.
- **RESET OUT** − This signal is used to reset all the connected devices when the microprocessor is reset.
- **READY** − This signal indicates that the device is ready to send or receive data. If READY is low, then the CPU has to wait for READY to go high.
- **HOLD** − This signal indicates that another master is requesting the use of the address and data buses.
- **HLDA (HOLD Acknowledge)** − It indicates that the CPU has received the HOLD request and it will relinquish the bus in the next clock cycle. HLDA is set to low after the HOLD signal is removed.

SERIAL I/O SIGNALS

There are 2 serial signals, i.e. SID and SOD and these signals are used for serial communication.

- **SOD** (Serial output data line) − The output SOD is set/reset as specified by the SIM instruction.
- **SID** (Serial input data line) − The data on this line is loaded into accumulator whenever a RIM instruction is executed.

8085 is pronounced as "eighty-eighty-five" microprocessor. It is an 8-bit microprocessor designed by Intel in 1977 using NMOS technology.

It has the following configuration −

- 8-bit data bus
- 16-bit address bus, which can address upto 64KB
- A 16-bit program counter
- A 16-bit stack pointer

- Six 8-bit registers arranged in pairs: BC, DE, HL
- Requires +5V supply to operate at 3.2 MHZ single phase clock

It is used in washing machines, microwave ovens, mobile phones, etc.

8085 MICROPROCESSOR – FUNCTIONAL UNITS

8085 consists of the following functional units −

Accumulator

It is an 8-bit register used to perform arithmetic, logical, I/O & LOAD/STORE operations. It is connected to internal data bus & ALU.

Arithmetic and logic unit

As the name suggests, it performs arithmetic and logical operations like Addition, Subtraction, AND, OR, etc. on 8-bit data.

General purpose register

There are 6 general purpose registers in 8085 processor, i.e. B, C, D, E, H & L. Each register can hold 8-bit data.

These registers can work in pair to hold 16-bit data and their pairing combination is like B-C, D-E & H-L.

Program counter

It is a 16-bit register used to store the memory address location of the next instruction to be executed. Microprocessor increments the program whenever an instruction is being executed, so that the program counter points to the memory address of the next instruction that is going to be executed.

Stack pointer

It is also a 16-bit register works like stack, which is always incremented/decremented by 2 during push & pop operations.

Temporary register

It is an 8-bit register, which holds the temporary data of arithmetic and logical operations.

Flag register

It is an 8-bit register having five 1-bit flip-flops, which holds either 0 or 1 depending upon the result stored in the accumulator.

These are the set of 5 flip-flops −

- Sign (S)
- Zero (Z)
- Auxiliary Carry (AC)
- Parity (P)
- Carry (C)

Its bit position is shown in the following table −

D7	D6	D5	D4	D3	D2	D1	D0
S	Z		AC		P		CY

Instruction register and decoder

It is an 8-bit register. When an instruction is fetched from memory then it is stored in the Instruction register. Instruction decoder decodes the information present in the Instruction register.

Timing and control unit

It provides timing and control signal to the microprocessor to perform operations. Following are the timing and control signals, which control external and internal circuits −

- Control Signals: READY, RD', WR', ALE
- Status Signals: S0, S1, IO/M'
- DMA Signals: HOLD, HLDA
- RESET Signals: RESET IN, RESET OUT

Interrupt control

As the name suggests it controls the interrupts during a process. When a microprocessor is executing a main program and whenever an interrupt occurs, the microprocessor shifts the control from the main program to process the incoming request. After the request is completed, the control goes back to the main program.

There are 5 interrupt signals in 8085 microprocessor: INTR, RST 7.5, RST 6.5, RST 5.5, TRAP.

Serial Input/output control

It controls the serial data communication by using these two instructions: SID (Serial input data) and SOD (Serial output data).

Address buffer and address-data buffer

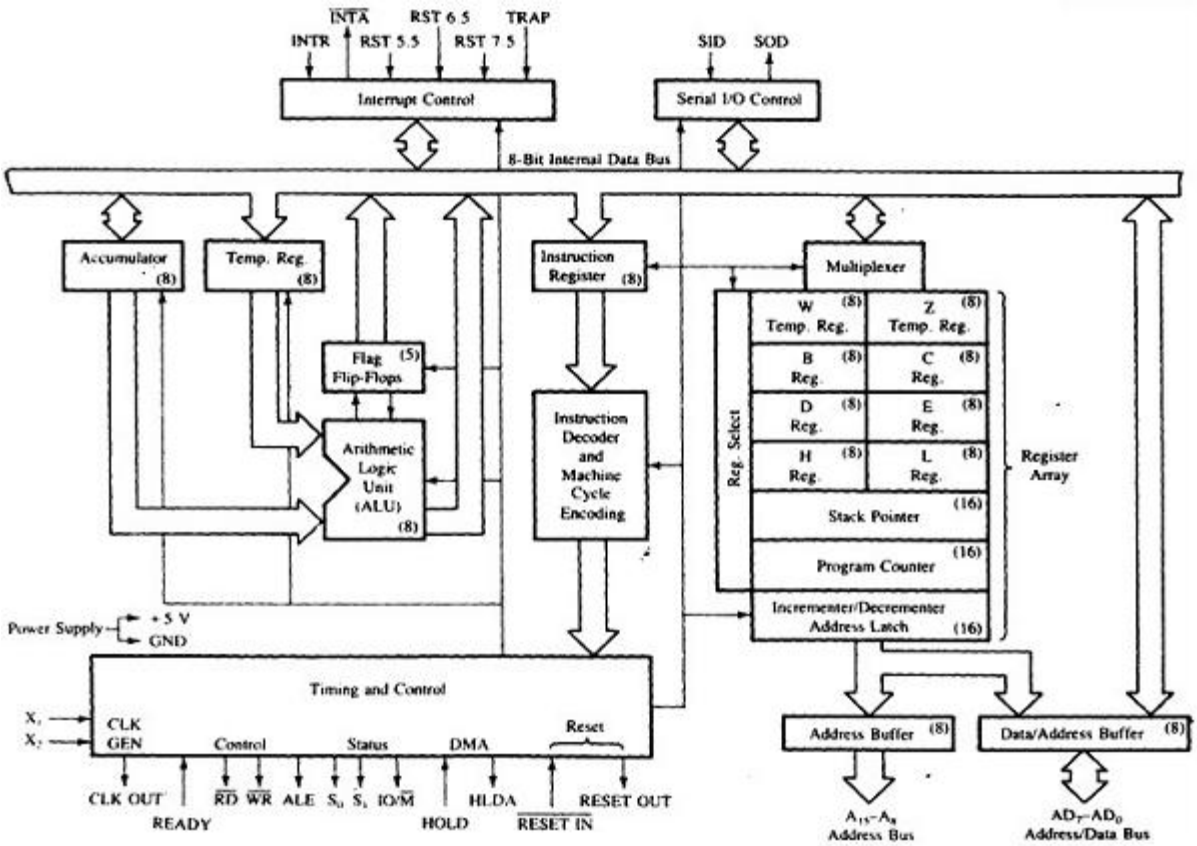
The content stored in the stack pointer and program counter is loaded into the address buffer and address-data buffer to communicate with the CPU. The memory and I/O chips are connected to these buses; the CPU can exchange the desired data with the memory and I/O chips.

Address bus and data bus

Data bus carries the data to be stored. It is bidirectional, whereas address bus carries the location to where it should be stored and it is unidirectional. It is used to transfer the data & Address I/O devices.

8085 Architecture

We have tried to depict the architecture of 8085 with this following image −



POSSIBLE QUESTIONS

EIGHT MARKS

1. Explain the architecture of 8085 microprocessor.
2. Explain the arithmetic instructions of 8085.
3. Explain the addressing modes of 8085
4. Write short note on address bus and data bus
5. Explain the addressing modes of 8085
6. Explain the Pin configuration of 8085
7. Explain the Pin configuration of 8085.
8. Explain the instruction set of 8085