**XCLUSIVE OR GATE** :-

Any ODD Number of Inputs gives Q.

That is if odd number of inputs then output is one.

**EXCLUSIVE NOR GATE** :-

Complement of Exclusive OR gate.

Any EVEN Number of Inputs gives Q.

That is if even number of inputs then output is one.

**UNIVERSAL GATE:-**

A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates

**EXCITATION TABLE FORMULAE**:-

SR Flip Flop :- Q(next) = S + R'Q

D Flip Flop :- Q(next) = DQ’ + DQ = D

JK Flip Flop :- Q(next) = JQ' + K'Q

T Flip Flop :- TQ' + T'Q

**CHARACTERISTIC TABLE:-**

D Flip Flop:- Qn+1 = D

The main difference between JK and RS flip-flop is that:

JK flip-flop accepts both inputs as 1

**Synchronous Counter :-**

Ring Counter

Johnson Counter

**Asynchronous Counter :-**

Ripple Counter

Up Down Counter

**RIPPLE COUNTER :-**

A ripple counter is an asynchronous counter where only the first flip-flop is clocked by an external clock. All subsequent flip-flops are clocked by the output of the preceding flip-flop. Asynchronous counters are also called ripple-counters because of the way the clock pulse ripples it way through the flip-flops.

**TTL :-** High power dissipation, Offer highest noise margin.

**ECL :-**Low Propagation delay, Fastest Logic Family

**CMOS :-**Highest Fan Out, Least power dissipation, Highest noise margin, Low cost per gate

**RTL:-** Low Noise Margin

**SSI, MSI, LSI, VLSI**:-

The number of gates for the different classification are as follows.

Small Scale Integration(SSI) - <10

Medium Scale Integration(MSI) - between 10 to 1000

Large Scale Integration(LSI) - >1000

Very Large Scale Integration(VLSI) - >100000

So the answer is option B

**Polling**:-

---------

In early years of computing processor has to wait for the signal for processing, so processor has to check each and every hardware and software program in the system if it has any signal to process. This method of checking the signal in the system for processing is called polling method.

**Hardware Interrupts:** If the signal for the processor is from external device or hardware is called hardware interrupts. Example: from keyboard we will press the key to do some action this pressing of key in keyboard will generate a signal which is given to the processor to do action, such interrupts are called hardware interrupts. Hardware interrupts can be classified into two types they are

**Maskable Interrupt**: The hardware interrupts which can be delayed when a much highest priority interrupt has occurred to the processor.

**Non Maskable Interrupt:** The hardware which cannot be delayed and should process by the processor immediately.

**Software Interrupts**: Software interrupt can also divided in to two types. They are

**Normal Interrupts**: the interrupts which are caused by the software instructions are called software instructions.

Exception: unplanned interrupts while executing a program is called Exception. For example: while executing a program if we got a value which should be divided by zero is called a exception.

**Classification of Interrupts According to Periodicity of Occurrence:**

**Periodic Interrupt:** If the interrupts occurred at fixed interval in timeline then that interrupts are called periodic interrupts

**Aperiodic Interrupt**: If the occurrence of interrupt cannot be predicted then that interrupt is called aperiodic interrupt.

**Classification of Interrupts According to the Temporal Relationship with System Clock:**

**Synchronous Interrupt**: The source of interrupt is in phase to the system clock is called synchronous interrupt. In other words interrupts which are dependent on the system clock. Example: timer service that uses the system clock.

**Asynchronous Interrupts**: If the interrupts are independent or not in phase to the system clock is called asynchronous interrupt.

**TRAP** :- It is an software generated interrupt like division by zero, invalid memory access.

**Boolean Algebra:-**

Complement Laws x + ~x = 1 x \* ~x = 0

Law of the Double Complement ~(~x) = x

Idempotent Laws x + x = x x \* x = x

Identity Laws x + 0 = x x \* 1 = x

Dominance Laws x + 1 = 1 x \* 0 = 0

Commutative Laws x + y = y + x x \* y = y \* x

Associative Laws x + (y + z) = (x + y) + z

x \* (y \* z) = (x \* y) \* z

Distributive Laws x + (y \* z) = (x + y) \* (x + z)

x \* (y + z) = (x \* y)+(x \* z)

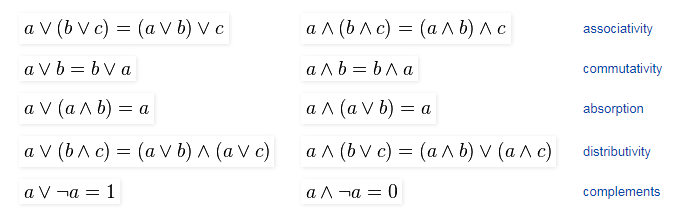
DeMorgan's Laws:- ~(x \* y) = ~x + ~y

~(x + y) = ~x \* ~y

Absorbtion Law:- x \* (x + y) = x

x + (x \* y) = x

Other Boolean Algebra Laws:-



**1’s Complement and 2’s Complement :-**

1’s complement of a binary number is another binary number obtained by toggling all bits in it, i.e., transforming the 0 bit to 1 and the 1 bit to 0.

Examples:

Let numbers be stored using 4 bits

1's complement of 7 (0111) is 8 (1000)

1's complement of 12 (1100) is 3 (0011)

2’s complement of a binary number is 1 added to the 1’s complement of the binary number.

Examples:

Let numbers be stored using 4 bits

2's complement of 7 (0111) is 9 (1001)

2's complement of 12 (1100) is 4 (0100)

These representations are used for signed numbers.

The main difference between 1′ s complement and 2′ s complement is that 1′ s complement has two representations of 0 (zero) – 00000000, which is positive zero (+0) and 11111111, which is negative zero (-0); whereas in 2′ s complement, there is only one representation for zero – 00000000 (+0) because if we add 1 to 11111111 (-1), we get 00000000 (+0) which is the same as positive zero. This is the reason why 2′ s complement is generally used.

**FRACTION BINARY INTO DECIMAL:-**

Let's take an example for n = 110.101

**Step 1: Conversion of 110 to decimal**

=> 1102 = (1\*22) + (1\*21) + (0\*20)

=> 1102 = 4 + 2 + 0

=> 1102 = 6

So equivalent decimal of binary integral is 6.

**Step 2: Conversion of .0101 to decimal**

=> 0.1012 = (1\*1/2) + (0\*1/22) + (1\*1/23)

=> 0.1012 = 1\*0.5 + 0\*0.25 + 1\*0.125

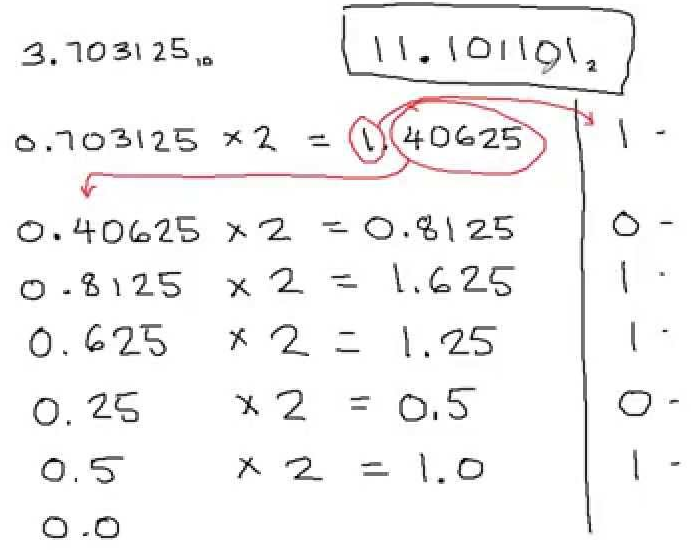
=> 0.1012 = 0.625

So equivalent decimal of binary fractional is 0.625

**Step 3: Add result of step 1 and 2.**

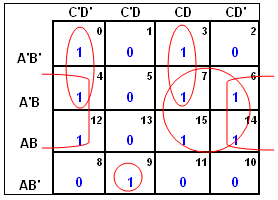
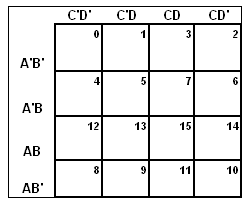
=> 6 + 0.625 = 6.625

**DECIMAL TO BINARY CONVERSION:-**

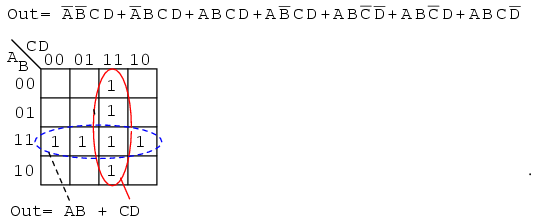
****

**KARNAUGH MAP:-**

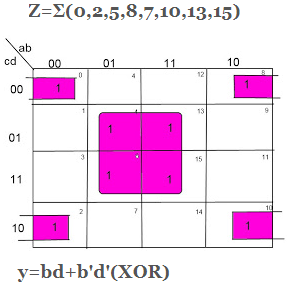
**General Structure Sample Joinings of 1’s**

****

**Example 1 :-**

****

**Example 2 :-**

****

**8085 ARCHITECTURE**

1. **Carry flag(CY):-**

If an operation performed in ALU generates the carry from D7 to next stage then CY flag is set, else it is reset.

2. **Auxiliary carry(AC):-**

If an operation performed in ALU generates the carry from lower nibble (D0 to D3) to upper nibble (D4 to D7) AC flag is set, else it resets.

3. **Zero flag(z):-**

If an operation performed in ALU results 0value of entire 8-bits then zero flag is set, else it resets.This flag is set, when the result of operation is zero, else it is reset.

1-zero result 0-non-zero result

4. **Sign flag(s):-**

If MSB bit =0 then the number is positive, else it is negative.

This flag is set, when MSB (Most Significant Bit) of the result is 1. Since negative binary numbers are represented in the 8085 CPU in standard two’s complement notation, SF indicates sign of the result.

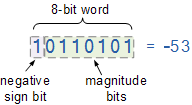
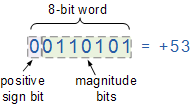
1-MSB is 1 (negative)

0-MSB is 0 (positive)

5. **Parity flag(p):-**

If the result contains even no. of ones this flag is set and for odd no. of ones this flag is reset.

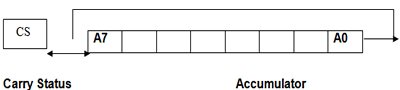
**PositiveAnd Negative Signed Binary Numbers:-**

****

**8085 INSTRUCTIONS:-**

**1. RLC: -** Each binary bit of the accumulator is rotated left by one position. Bit D7 is placed in the position of D0 as well as in the Carry flag. CY is modified according to bit D7. Any other bit is not affected.

**2. RRC: -** Each binary bit of the accumulator is rotated right by one position. Bit D0 is placed in the position of D7 as well as in the Carry flag. CY is modified according to bit D0. Any other bit is not affected.



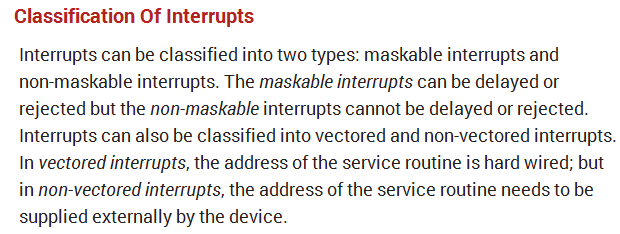
**3. RAL: -** Each binary bit of the accumulator is rotated left by one position through the Carry flag. Bit D7 is placed in the Carry flag, and the Carry flag is placed in the least significant position D0. CY is modified according to bit D7.

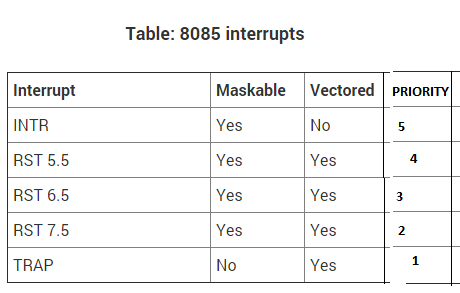
**4. RAR: -** Each binary bit of the accumulator is rotated right by one position through the Carry flag. Bit D0 is placed in the Carry flag, and the Carry flag is placed in the most significant position D7. CY is modified according to bit D0.

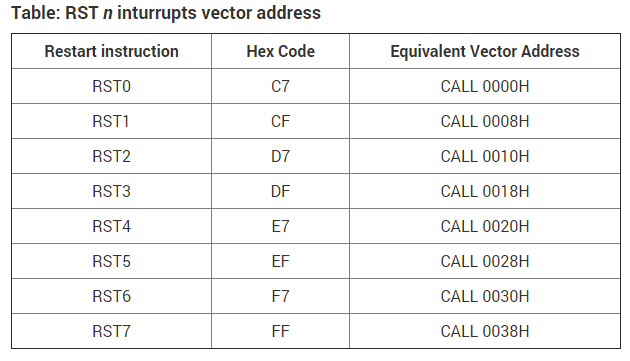
**5. CMA: -** The content of accumulator is complemented.

**6. CMC: -** The carry flag is complemented.

**7. STC: -** This instruction sets the carry flag.

****

****

****

**One-byte instructions:** A 1 byte instruction include the opcode and the operand in the 8 bits only which is one byte.

Ex: 1. MOV C, A Hex code = 4FH (one byte)

2. ADD B Hex code = 80H (one byte)

3. CMA Hex code = 2FH (one byte)

**Two-byte instructions:** The two byte instruction is one which contains an 8-bit op-code and 8-bit operand (Data).

Ex: 1. MVI A, 09 Hex code = 3E, 09 (two bytes)

2. ADD B, 07 Hex code = 80, 07 (two bytes)

3. SUB A, 05 Hex code = 97, 05 (two bytes)

**Three-byte instructions:** In a three byte instruction the first byte is opcode and second and third bytes are operands i.e. 16-bit data or 16-bit address.

1. LDA 8509 Hex code = 3A, 09, 85 (Three bytes)

2. LXI 2500 Hex code = 21, 00, 25 (Three bytes)

3. STA 2600 Hex code = 32, 00, 26 (Three bytes)

**The different types of instructions are as follows:**

– **Immediate Mode**: As the name suggests the instruction in itself contains the operand.

Example: MOV AL, 35H (move the data 35H into AL register)

– **Register Mode**: In this mode the operands of an instruction are placed in the registers which themselves are placed inside the CPU.

Example: MOV AX,CX (move the contents of CX register to AX register)

**– Direct address mode:** The address part of an instruction in this mode is the effective address.

Example:ADD AL,[0301] //add the contents of offset address 0301 to AL

– **Indexed addressing mode:** In this mode in order to obtain the effective address the contents of the index register is added to the instructions address part.

Example:-MOV AX, [SI +05] (Index Register SI or DI)

**– Relative address mode**: In this mode in order to find out the effective address the contents of the program counter are added to the address part of the instruction.

**-Register Indirect Addressing**: Register indirect addressing is similar to indirect addressing, except that the address field refers to a register instead of a memory location. It requires only one memory reference and no special calculation.

Example:-MOV AX, [BX](move the contents of memory location s addressed by the register BX to the register AX)

**Base addressing:** The operand’s offset is sum of an 8 bit or 16 bit displacement and the contents of the base register BX or BP.BX is used as a base register for data segment ,and BP is used as a base register for stack segment.

Example:MOV AL,[BX+05] //suppose the register BX contain 0301.The offset will be 0301+05=0306.Content of the memory location 0306 will move to AL.

**Indexed addressing mode:** The operand’s offset is the sum of the content of an index register SI or DI and an 8 bit or 16 bit displacement. Example:MOV AX, [SI +05]

**Based Indexed Addressing:** The operand’s offset is sum of the content of a base register BX or BP and an index register SI or DI. Example: ADD AX, [BX+SI]

**Based Indexed plus displacement addressing mode:** In this mode of addressing the operand’s offset is given by offset=[BX or BP]+[SI or DI]+8 bit or 16 bit displacement. Example:MOV AX, [BX+SI+05]

**GRID COMPUTING**

In M \* N grid the worst case delays ( in hops ) is n -1 + m – 1.

A multicomputer with 256 CPUs is organized as 16 × 16 grid. What is the worst case delay (in hops) that a message might have to take?

Ans : 16 + 16 -2 = 30