# I - ASSIGNMENT

(Start Writing From Here)

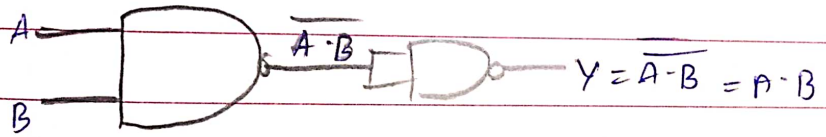
1) How to implement NOT, AND and OR Gate using NAND and NOR Gates.

A) A NAND gate is known as universal gate because it can be used to construct an AND Gate, OR Gate, NOT gate, Ex-OR Gate or any combination of functions.

a) NOT GATE by using NAND GATE

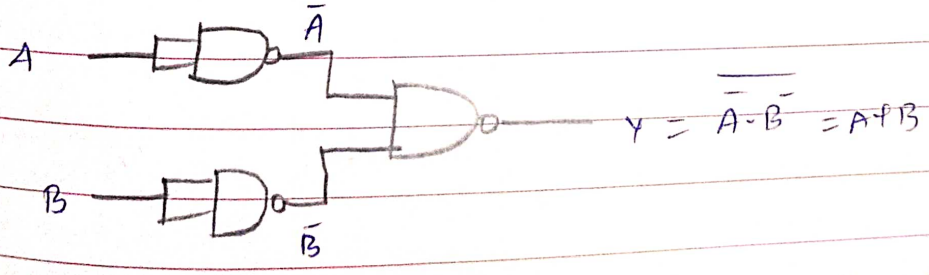

A NOT Gate maybe obtained from NAND gate by connecting all of its inputs of a NAND Gate together

b) AND Gate by using NAND GATE




* An AND gat can be obtained by connecting the out of a NAND gate to the input of NOT gate

c) OR Gate by using NAND Gates



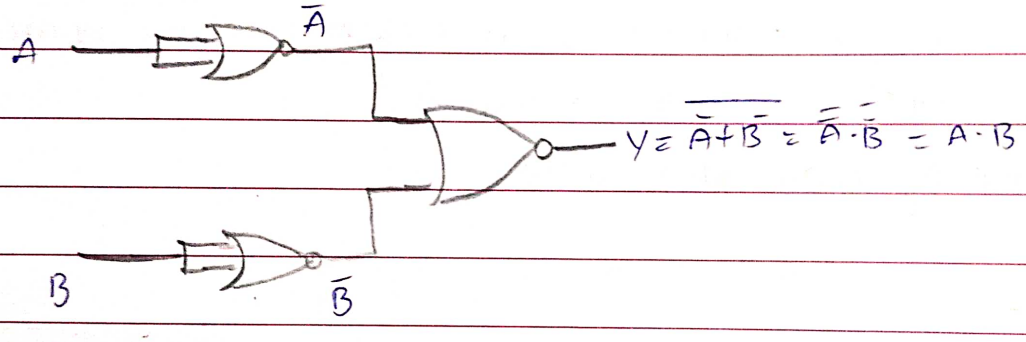

An OR Gate can be obtained by connecting the outputs of two NOT gates (using NAND) to the inputs of a NAND gate.

AND, OR, NOT using NOR Gate

a) NOT using NOR Gate

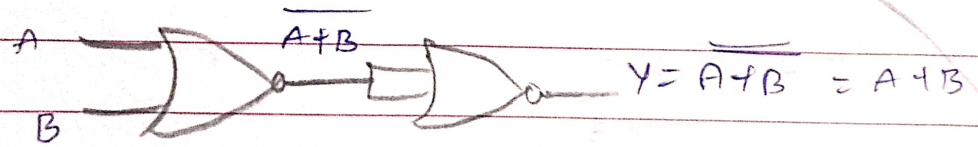

⇒ If the two inputs of a NOR gate are connected together, then we get NOT Gate.

b) AND Gate by using NOR Gates :-




* An AND gate can be produced by using 3 NOR gates.

c) OR Gate by using NOR Gates




* An OR Gate can be produced by using two NOR gates

2) Find out SOP for the following:-

A) $F(A,B,C) = AB + BC'$

$F(A,B,C,D) = A + B'CD'$

A) (i) $F(A,B,C) = AB + B\bar{C}$

$F(A,B,C) \rightarrow ABC(C+\bar{C}) + B\bar{C}(A+\bar{A})$

$\rightarrow ABC + AB\bar{C} + AB\bar{C} + \bar{A}B\bar{C}$ (∵ Neglect repeating terms)

$\rightarrow ABC + AB\bar{C} + \bar{A}B\bar{C}$

$ABC = 111 (7)$

$AB\bar{C} = 110 (6)$

$\bar{A}B\bar{C} = 010 = (2)$

$F(A,B,C) \rightarrow m_7 + m_6 + m_2$

$\rightarrow \Sigma m(7,6,2)$

(ii) $F(A,B,C,D) = A + B'CD'$

$\Rightarrow AB + A\bar{B} + \bar{B}C\bar{D}(A+\bar{A}) \Rightarrow AB + A\bar{B} + \bar{A}\bar{B}CD + \bar{A}BCD$

$\rightarrow ABC(C+\bar{C}) + AB(C+\bar{C}) + \bar{A}\bar{B}C\bar{D} + \bar{A}BCD \rightarrow ABC + ABC + AB\bar{C} + A\bar{B}C + A\bar{B}\bar{C} + \bar{A}\bar{B}CD + \bar{A}BCD$

$\rightarrow ABC(D+\bar{D}) + AB(C(D+\bar{D}) + AB\bar{C}(D+\bar{D}) + A\bar{B}\bar{C}(D+\bar{D}) + \bar{A}\bar{B}CD + \bar{A}BCD$

$\rightarrow ABCD + ABC\bar{D} + AB\bar{C}D + AB\bar{C}\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}BCD$

$\rightarrow ABCD + ABC\bar{D} + AB\bar{C}D + AB\bar{C}\bar{D} + A\bar{B}CD + A\bar{B}C\bar{D} + A\bar{B}\bar{C}D + A\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}CD + \bar{A}BCD$

(1111) (1110) (1011) (1010) (1101) (1100) (1001) (1000) (0011) (0111)

$\Rightarrow \Sigma m(15, 14, 11, 10, 13, 12, 9, 8, 3, 7)$

3) Demonstrate the design steps of synchronous counters with suitable examples?

A)

## 4 bit synchronous counter

In synchronous counter,

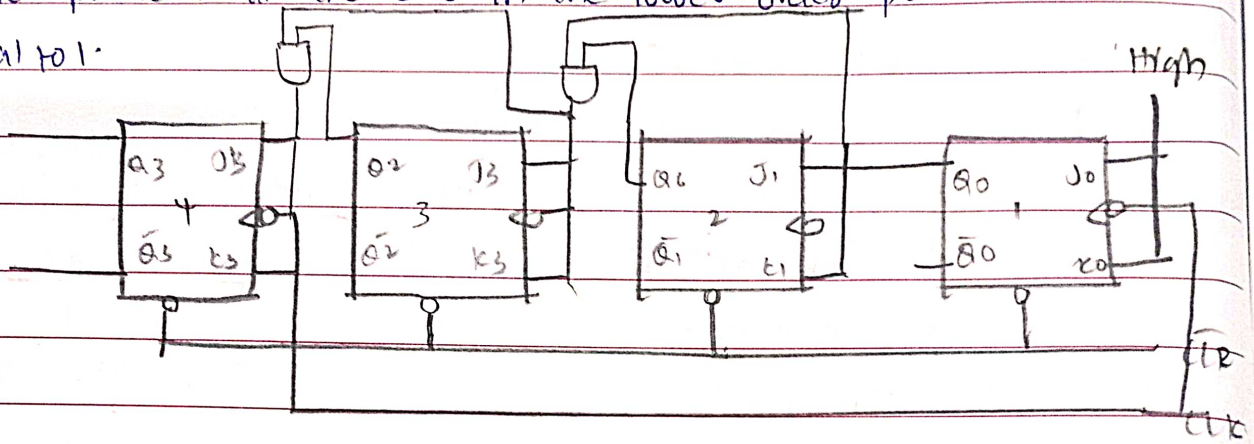
① The clock signal is applied to all the flip-flops simultaneously

② The J and K inputs of the first flip flop would toggle connected to high.

③ The first flip-flop would toggle for each clock pulse.

because it's J-K inputs are connected to high

4) A flip-flop in any other position is complemented with a pulse provided all the bits in the lower order positions are equal to 1.

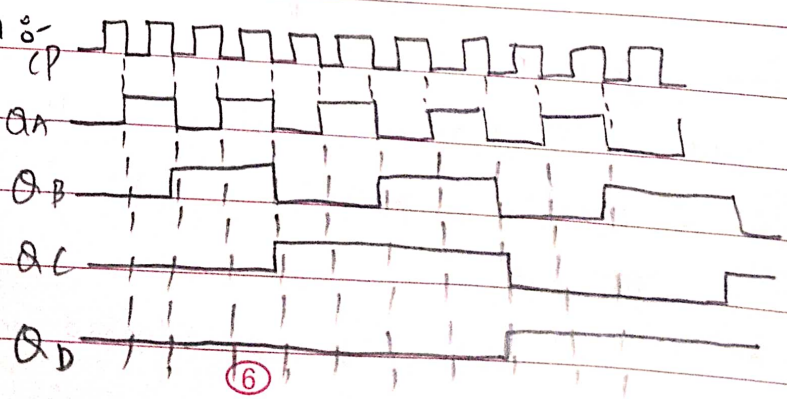



4-bit synchronous counter

operation

- The counting sequence shows that the flip-flop $Q_0$ toggoles during each negative going transition (NGT) of the clock pulse. For this reason its J0 and K0 inputs are permanently connected to high.
- The counting sequence shows that flip-flop $Q_1$ toggoles during each NGT clock pulse only when $Q_0 = 1$
- The counting sequence shows that the flip-flop $Q_2$ toggles during each NGT of the clock pulse only when $Q_0 = Q_1 = 1$.
- In a likemanner, we can see that the flip-flop $Q_3$ toggles on each NGT that occurs while $Q_0 = Q_1 = Q_2 = 1$.

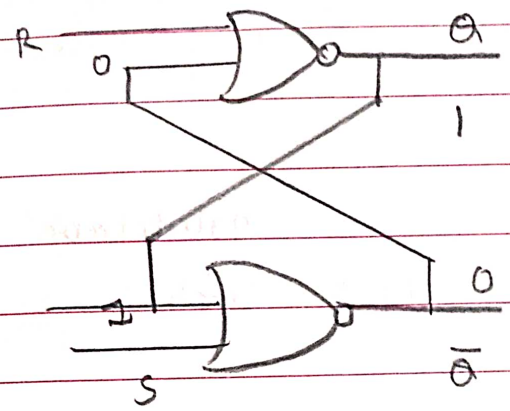
TIMING DIAGRAM :-



1) Explain about SR-latch using with a neat diagram using Nand and nor gates.

A) SR-latch using NOR gate

=) S and R stands for set and reset. It can be constructed from a pair of cross-coupled NOR logic gates.




| Inputs | | outputs | |
|---|---|---|---|
| S | R | Q | $\bar{Q}$ |
| 0 | 0 | $Q_0$ | $\bar{Q}_0$ |
| 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | x | x |

=) while the S and R inputs are both low, feedback maintains the Q and $\bar{Q}$ outputs in a constant state, with $\bar{Q}$ the complement of Q. If S(Set) is pulsed high while R(Reset) is low, then the Q output is forced high, and stays high when S returns to low; R is pulsed high, while S is held low, then the Q output is forced low, and stays low when R returns to low.

=) $Q_0$ is the previous state of Q and $\bar{Q}_0$ is the previous state of $\bar{Q}$.

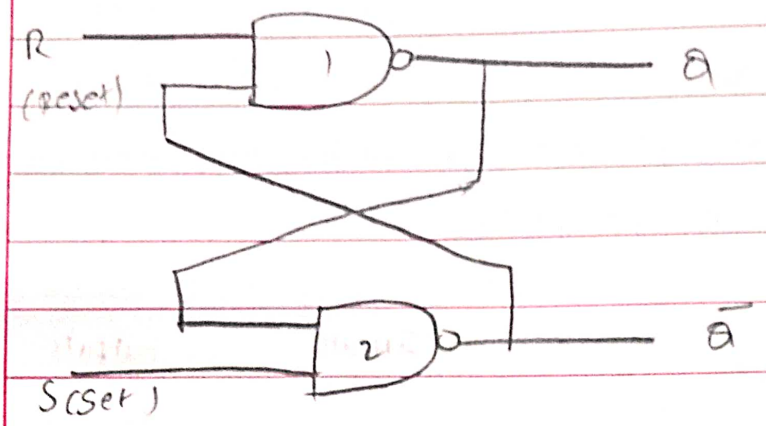
-) ~~R and S are~~ S sets the output to 1 and R resets the output to 0

=> Both R and S cannot be high at the same time. the ouput is undefined.

SR latch using NAND Gate

The two NAND gates are cross coupled so that the output of NAND gate 1 is connected to one of the inputs of NAND gate 2 and

vice versa. The




| Inputs | | outputs | |
|---|---|---|---|
| R | S | Q | $\bar{Q}$ |
| 0 | 0 | x | x |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 1 | $Q_0$ | $\bar{Q}_0$ |

SR latch using NAND Gate — Truth table

1) A high on the S input will set the R-S latch, i.e $Q = 1$, $\bar{Q} = 1$.

2) A high on R input will reset it, i.e, $Q = 0$, $\bar{Q} = 1$

3) If both S and R are high, the latch will remain in its previous state.

4) when both R and S are low, the output is invalid.

5) what are different types of instruction codes?

A) Instruction codes are bits that instruct the computer to execute a specific operation.

Types of instruction codes:-

1) One operand instructions:- These instructions have one operand and perform an operation on that operand.

2) Two-operand instructions:- These instructions have two operands and perform an operation involving both.

3) Three-operand instructions:- These instructions have three operands

and perform an operation involving both all the three operands.

4) Data transfer instructions:- These instructions move data between memory and registers or between registers.

5) Control transfer instructions: These instructions change the flow of program execution by modifying the program counter.

6) Arithmatic instructions:- These instructions perform mathematical operations on operands.

7) logical instructions: These instructions ~~compa~~ perform logical operations on operands.

8) Comparison instructions:- These instructions compare two operands and sets a flag based on the result.

9) Floating-point instructions:- These instructions perform arithmetic and other operations on floating-point numbers.