DIGITAL LOGIC AND MICROPROCESSOR

SYLLABUS

BTES405: Digital Logic Design & Microprocessor

[Unit1] Introduction **[7 Hours]**

Digital signals, digital circuits, AND, OR, NOT, NAND, NOR and Exclusive-OR operations, Boolean algebra, examples of IC gates, Number Systems: binary, signed binary, octal hexadecimal number, binary arithmetic, one's and two's complements arithmetic, codes, error detecting and correcting codes.

[Unit 2] Combinational Digital Circuits **[7 Hours]**

Standard representation for logic functions, K-map representation, simplification of logic functions using K-map, minimization of logical functions, Don't care conditions, Multiplexer, De-Multiplexer / Decoders,Adders, Subtractors, BCD arithmetic, carry look ahead adder, serial adder, ALU, elementary ALU design, parity checker / generator.

[Unit 3] Sequential circuits and systems **[7 Hours]**

1-bit memory, the circuit properties of Bistable latch, the clocked SR flip flop, J-K-T and D-types flip flops, applications of flip flops, shift registers, applications of shift registers, serial to parallel converter, parallel to serial converter, ring counter, sequence generator, ripple(Asynchronous) counters, synchronous counters, counters design using flip flops, special counter IC's, asynchronous sequential counters, applications of counters.

[Unit 4] Fundamentals of Microprocessors **[7 Hours]**

Fundamentals of Microprocessor, Comparison of 8-bit, (8085) 16-bit (8086), and 32-bit microprocessors (80386), The 8086 Architecture: Internal Block Diagram, CPU, ALU, address, data and control bus, Working registers, SFRs, Clock and RESET circuits, Stack and Stack Pointer, Program Counter, I/O ports, Memory Structures, Data and Program Memory, Timing diagrams and Execution Cycles.

[Unit 5] 8086 Instruction Set and Programming **[7 Hours]**

Memory Interfacing, I/O Interfacing, Direct Memory Access (DMA), Interrupts in 8086, 8086 Instruction Set and Programming: Addressing modes: Introduction, Instruction syntax, Data types, Subroutines Immediate addressing, Register addressing, Direct addressing, Indirect addressing, Relative addressing, Indexed addressing, Bit inherent addressing, bit direct addressing, Instruction timings, Data transfer instructions, Arithmetic instructions, Logical instructions, Branch instructions, Subroutine instructions, Bit manipulation instruction, Assembly language programs, C language programs, Assemblers and compilers, Programming and debugging tools.

Text Book:

1. R. P. Jain, Modern Digital Electronics, McGraw Hill Education, 2009.

NOTES

Unit 1: Introduction

Syllabus Focus: Digital signals, digital circuits (AND, OR, NOT, NAND, NOR, XOR), Boolean algebra, number systems (binary, signed binary, octal, hexadecimal), binary arithmetic, one's and two's complements, codes, error detecting and correcting codes.

Key 6-Mark Questions & How to Answer:

1. **What is a Signal? Differentiate between Analog vs. Digital Signals. Write Characteristics of Digital Signals. (6 Marks)**
    - **Answer Structure:**
        - **Definition of Signal:** Start with a general definition of a signal (e.g., a physical quantity that conveys information).
        - **Analog Signal:**
            - **Definition:** Continuous in both time and amplitude. Can take any value within a range.
            - **Example:** Voice, temperature, sound waves.
            - **Diagram:** Draw a continuous, wavy line.
        - **Digital Signal:**
            - **Definition:** Discrete in time and amplitude. Typically represented by two distinct voltage levels (HIGH/LOW, 1/0).
            - **Example:** Data in computers, ON/OFF states.
            - **Diagram:** Draw a square wave showing clear HIGH and LOW levels.
        - **Differentiation (Table Recommended):** | Feature | Analog Signal | Digital Signal | | :---------- | :---------- | :---------- | | Values | Continuous | Discrete (typically 0s and 1s) | | Noise Immunity | Low (more susceptible) | High (more robust) | | Bandwidth | Requires more bandwidth | Requires less bandwidth | | Storage | Difficult to store precisely | Easy to store and reproduce accurately | | Processing | Difficult, requires analog circuits | Easy, uses logic gates/microprocessors |
        - **Characteristics of Digital Signals (Elaborate on 4-5 points):**
            - **Noise Immunity:** Due to distinct voltage levels, small fluctuations don't alter the logic state.
            - **Accuracy & Reproducibility:** Can be copied and transmitted without degradation, maintaining original information integrity.
            - **Ease of Design & Processing:** Digital circuits are easier to design and can process information using simple logic operations.

        - **Integration & Miniaturization:** Digital circuits can be highly integrated (ICs), leading to smaller, more complex systems.
        - **Programmability:** Digital systems are easily controlled and reconfigured through software.

2. **Which gates are known as universal gates? Justify using examples. (6 Marks)**
    - **Answer Structure:**
        - **Definition of Universal Gate:** A universal gate is a logic gate that can implement any other Boolean function or logic gate (AND, OR, NOT) without the need for other gate types.
        - **Identify Universal Gates:** NAND and NOR gates are universal gates.
        - **Justification for NAND Gate:**
            - **NOT gate from NAND:** Connect all inputs of a NAND gate together. Output is A.
                - *Diagram:* [NAND gate with inputs tied to A, output A]
            - **AND gate from NAND:** Connect the output of a NAND gate (A NAND B) to the input of a NOT gate (using another NAND). Output is A·B.
                - *Diagram:* [NAND gate followed by another NAND used as NOT]
            - **OR gate from NAND:** Connect the inputs A and B to two separate NAND gates (each acting as NOT), then feed their outputs to a third NAND gate. Output is A+B. (Using De Morgan's: (A·B)=A+B)
                - *Diagram:* [Two NAND-NOT gates feeding into a third NAND]
        - **Justification for NOR Gate (briefly but clearly):**
            - **NOT gate from NOR:** Connect all inputs of a NOR gate together. Output is A.
                - *Diagram:* [NOR gate with inputs tied to A, output A]
            - **OR gate from NOR:** Connect the output of a NOR gate (A NOR B) to the input of a NOT gate (using another NOR). Output is A+B.
                - *Diagram:* [NOR gate followed by another NOR used as NOT]
            - **AND gate from NOR:** Connect inputs A and B to two separate NOR gates (each acting as NOT), then feed their outputs to a third NOR gate. Output is A·B. (Using De Morgan's: (A+B)=A·B)
                - *Diagram:* [Two NOR-NOT gates feeding into a third NOR]
        - **Clarity on Diagrams:** Ensure all diagrams are clear, correctly labeled, and show the logical derivation.

3. **State and prove any two theorems of Boolean algebra. (6 Marks)**

- **Answer Structure:**
  - **Introduction:** Briefly explain Boolean algebra as a mathematical system for analyzing and simplifying digital circuits using logical operations (AND, OR, NOT).
  - **Theorem 1: De Morgan's First Theorem**
    - **Statement:** $A \cdot B = A + B$ (The complement of a product is equal to the sum of the complements).
    - **Proof (Truth Table):** | A | B | A.B | A·B (LHS) | A | B | A+B (RHS) | |---|---|-----|-------------------|---------------|---------------|---------------------| | 0 | 0 | 0 | 1 | 1 | 1 | | 0 | 1 | 0 | 1 | 1 | 0 | 1 | | 1 | 0 | 0 | 1 | 0 | 1 | 1 | | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
      - *Conclusion:* Since LHS = RHS for all combinations, the theorem is proven.
    - **Alternative Proof (Algebraic):** (Less common for 6-marks, but good to know)
  - **Theorem 2: Absorption Law**
    - **Statement:** A+(A·B)=A**Error! Filename not specified.**
    - **Proof (Truth Table):** | A | B | A.B | A+(A·B) (LHS) | |---|---|-----|-----------------| | 0 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | | 1 | 0 | 0 | 1 | | 1 | 1 | 1 | 1 | 1 |
      - *Conclusion:* Since LHS = A for all combinations, the theorem is proven.
    - **Alternative Proof (Algebraic):** $A + (A \cdot B) = A \cdot 1 + A \cdot B$ (Identity Law: $A \cdot 1 = A$) $= A \cdot (1 + B)$ (Distributive Law) $= A \cdot 1$ (Identity Law: $1 + B = 1$) $= A$ (Identity Law: $A \cdot 1 = A$)
  - **Choice of Theorems:** You can choose any two, but De Morgan's and Absorption are very common. Other options include Associative, Commutative, Distributive, Idempotent laws, etc.

Unit 2: Combinational Digital Circuits

Syllabus Focus: Standard representation for logic functions, K-map, simplification, minimization, Don't care conditions, Multiplexer, De-Multiplexer/Decoders, Adders, Subtractors, BCD arithmetic, carry look ahead adder, serial adder, ALU, parity checker/generator.

Key 6-Mark Questions & How to Answer:

1. **Minimize the four-variable logic function using K-map (e.g.,** f(A,B,C,D)=Σm(0,1,2,3,5,7,8,9,11,14)**). (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** Briefly explain Karnaugh Maps (K-maps) as a graphical method to simplify Boolean expressions. Mention its advantage over algebraic methods for up to 5 variables.
        - **Draw K-map Grid:** Draw a 4-variable K-map grid (16 cells). Clearly label the rows and columns with Gray code (e.g., AB on top: 00, 01, 11, 10; CD on side: 00, 01, 11, 10).
        - **Plot Min/Maxterms:** For Σm, place '1's in the cells corresponding to the given minterms.
        - **Grouping Rules & Strategy:**
            - **Largest Groups First:** Always try to form the largest possible groups (powers of 2: 16, 8, 4, 2, 1).
            - **Adjacent Cells:** Cells are adjacent horizontally, vertically, and *wrap around* (top-bottom, left-right).
            - **Overlap Allowed:** Groups can overlap to include more '1's, but each '1' must be covered by at least one group.
            - **Minimal Groups:** Aim for the fewest number of groups that cover all '1's.
        - **Example (for** f(A,B,C,D)=Σm(0,1,2,3,5,7,8,9,11,14)**):**

1. **K-Map Plotting:**
2. CD
3. AB 00 01 11 10
4. 00 1 1 1 1 (0,1,3,2)
5. 01 1 1 0 1 (4,5,7,6) -> (5,7) are 1
6. 11 0 1 0 0 (12,13,15,14) -> (14) is 1
7. 10 1 0 1 0 (8,9,11,10) -> (8,9,11) are 1

8. **Grouping:**

- Group 1 (Quad): m(0,1,2,3) ->
  AB**Error! Filename not specified.**
- Group 2 (Quad): m(0,8,9,1) (wrap-around) ->
  BD**Error! Filename not specified.**
- Group 3 (Pair): m(5,7) -> ABD**Error! Filename not specified.**
- Group 4 (Pair): m(11,3) (wrap-around diagonally if allowed, but no, this needs 11,14) -> m(11,14) is not a simple group, it's m(11) and m(14). Recheck. Let's re-plot the given minterms carefully:
  f(A,B,C,D)=Σm(0,1,2,3,5,7,8,9,11,14)
  Error! Filename not specified.

9. CD

10. AB 00 01 11 10

11. 00 1 1 1 1 (m0, m1, m3, m2)

12. 01 0 1 1 0 (m4, m5, m7, m6) <-- m5, m7 are 1s

13. 11 0 0 0 1 (m12, m13, m15, m14) <-- m14 is 1

14. 10 1 1 1 0 (m8, m9, m11, m10) <-- m8, m9, m11 are 1s

- **Groups:**
  - Quad: m(0,1,2,3) ->
    AB**Error! Filename not specified.**
  - Quad: m(0,8,1,9) ->
    BD**Error! Filename not specified.**
  - Pair: m(5,7) -> ABD**Error! Filename not specified.**
  - Pair: m(9,11) -> BCD**Error! Filename not specified.**
  - Isolated '1': m(14) cannot be grouped larger than a pair. Can it be grouped with anything else? No. So m(14) -> ABCD**Error! Filename not specified.**
- *Revised Groups (check for essential prime implicants):*
  - Essential Quad: m(0,1,2,3) -> AB (covers m2, m3, which nothing else covers fully)
  - Essential Quad: m(0,8,9,1) -> BD (covers m8, m9 which no other large group covers efficiently)
  - Pair: m(5,7) -> ABD (covers m5, m7)

- What about m11 and m14? m11 can be paired with m3.
- Let's redraw and re-group carefully:
- CD
- AB 00 01 11 10
- 00 1 1 1 1 <-- (0,1,3,2)
- 01 0 1 1 0 <-- (5,7)
- 11 0 0 0 1 <-- (14)
- 10 1 1 1 0 <-- (8,9,11)

  - Group 1 (Quad): m(0,1,3,2) -> AB**Error! Filename not specified.**
  - Group 2 (Quad): m(0,8,9,1) -> BD**Error! Filename not specified.**
  - Group 3 (Pair): m(5,7) -> ABD**Error! Filename not specified.**
  - Group 4 (Pair): m(3,11) -> ACD (covers m3 again, but m11 needs covering, and it's adj to m3)
  - Group 5 (Isolated/Pair): m(14) - This can't be grouped with m10 (0101) or m12 (1100). Let's re-examine m14: it's 1110. It's adjacent to m12 (1100) and m15 (1111). In the current K-map: m14 is isolated. The given solution in your document is f(A,B,C,D)=Σm(0,1,2,3,5,7,8,9,11,14) Result for this specific K-map (using essential prime implicants first):

6. AB (covers 0,1,2,3)
7. BD (covers 0,1,8,9)
8. ACD (covers 3,7. covers 3 already. so only need to cover 7 efficiently. m(5,7) -> ABD)
9. BCD (covers 9,11. covers 9 already. so need to cover 11 efficiently. m(11) adjacent to m(3).)
10. ABCD (covers 14. m(14) is 1110. Adjacent to m(10), which is 1010.) Wait, m(11) is 1011. m(14) is 1110. Let's re-evaluate the terms and groups. m0(0000), m1(0001), m2(0010), m3(0011) m5(0101), m7(0111) m8(1000), m9(1001), m11(1011) m14(1110)

Optimal Grouping:

11. Quad: m(0,1,8,9) -> CD (No, this is BD) (0000, 0001, 1000, 1001) -> BD**Error! Filename not specified.**

12. Quad: m(0,2,8,10) -> BC (No, m10 is not 1. m(0,2) is ABC') Let's use the standard K-map format:

13. CD

14. AB 00 01 11 10

15. 00 1 1 1 1 (0,1,3,2)

16. 01 0 1 1 0 (4,5,7,6) -> m5,m7

17. 11 0 0 0 1 (12,13,15,14) -> m14

18. 10 1 1 1 0 (8,9,11,10) -> m8,m9,m11

- **Group A (Quad):** m(0,1,2,3) -> AB**Error! Filename not specified.**
- **Group B (Quad):** m(0,8,9,1) -> BD (covers 0,1,8,9. m(0) and m(1) are already covered by Group A)
- **Group C (Pair):** m(5,7) -> ABD**Error! Filename not specified.**
- **Group D (Pair):** m(3,11) -> BCD (Covers m3 which is already covered by Group A)
- **Group E (Pair):** m(14,10) - m10 is not 1. No, m14 is only adjacent to m10 (1010) and m15 (1111).
- Let's ensure *all* 1s are covered with *minimal* groups.

Revised Optimal Grouping for Σm(0,1,2,3,5,7,8,9,11,14)**Error! Filename not specified.**

7. **Quad 1 (Essential):** m(0,1,2,3) => AB (covers m0, m1, m2, m3)

8. **Quad 2 (Essential):** m(0,8,9,1) => BD (covers m0, m1, m8, m9. m0,m1 are redundant coverage, but m8,m9 are uniquely covered by this large group).

9. **Pair 1:** m(5,7) => ABD (covers m5, m7)

10. **Pair 2:** m(11,9) => ACD (covers m11. m9 is already covered by Quad 2, but it helps make a larger group for m11. If m11 is isolated, it would be ABCD. However, it pairs with m9.)

11. **Isolated:** m(14) => ABCD (m14 has no other groups of 2 or 4.)

Final simplified expression: F=AB+BD+ABD+ACD+ABCD**Error! Filename not specified.**

        - **Write Minimized Expression:** Write the sum of the products (SOP) terms derived from your groups.
        - **Show All Steps:** Clearly indicate the K-map, the groups drawn, and the derivation of each product term.

2. **Explain the working of Multiplexer and De-Multiplexer. (6 Marks)**
    - **Answer Structure:**
        - **Multiplexer (MUX):**
            - **Definition:** A combinational logic circuit that selects one of 'n' input data lines and routes it to a single output line. The selection is controlled by 'm' select lines, where 2m=n.
            - **Working Principle:** The decoder within the MUX activates one specific input gate based on the binary code on the select lines. Only the data from the selected input line passes through to the output.
            - **Block Diagram:** Draw a general block diagram of an N×1 MUX (e.g., 4:1 MUX).
                - Inputs: I0,I1,...,IN−1**Error! Filename not specified.**
                - Select Lines: S0,S1,...,Sm−1**Error! Filename not specified.**
                - Output: Y
                - Enable (Optional but good): E**Error! Filename not specified.**
                - *Example:* For a 4:1 MUX, if S1S0=00, I0 is selected; if S1S0 =01, I1 is selected, and so on.
            - **Truth Table (Brief Example):** For a 2:1 MUX, show how the select line determines the output.
        - **De-Multiplexer (DEMUX):**
            - **Definition:** A combinational logic circuit that takes a single input data line and routes it to one of 'n' output lines. The specific output line is controlled by 'm' select lines, where 2m=n. It's the inverse of a MUX.
            - **Working Principle:** The input data is passed to the output determined by the select lines. All other output lines remain OFF or at a default state. Essentially, it acts as a decoder with an enable input.
            - **Block Diagram:** Draw a general block diagram of a 1×N DEMUX (e.g., 1:4 DEMUX).
                - Input: D_in
                - Select Lines: S0,S1,...,Sm−1**Error! Filename not specified.**
                - Outputs: O0,O1,...,ON−1**Error! Filename not specified.**

        - Enable (often the data input itself)
        - *Example:* For a 1:4 DEMUX, if S1S0=10, Din is routed to O2.
    - **Truth Table (Brief Example):** For a 1:2 DEMUX, show how the select line routes the input.

3. **Design a half-adder and full-adder circuits. (6 Marks)**
    - **Answer Structure:**
        - **Half-Adder:**
            - **Definition:** A combinational circuit that adds two single binary bits (A and B) and produces a Sum (S) and a Carry-out (Cout).
            - **Truth Table:** | A | B | Sum (S) | Carry Out (Cout) | |---|---|---------|---------------| | 0 | 0 | 0 | 0 | | 0 | 1 | 1 | 0 | | 1 | 0 | 1 | 0 | | 1 | 1 | 0 | 1 |
            - **Boolean Expressions:**
                - $S=A \oplus B$ (Exclusive-OR)
                - $Cout=A \cdot B$ (AND)
            - **Logic Circuit Diagram:** Draw the circuit using one XOR gate and one AND gate. Clearly label inputs A, B and outputs S, Cout.
        - **Full-Adder:**
            - **Definition:** A combinational circuit that adds three binary bits (two input bits A and B, and a Carry-in Cin) and produces a Sum (S) and a Carry-out (Cout). Used for adding multi-bit numbers.
            - **Truth Table:** | A | B | Cin | Sum (S) | Carry Out (Cout) | |---|---|-----|---------|------------------| | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 1 | 1 | 0 | | 0 | 1 | 0 | 1 | 0 | | 0 | 1 | 1 | 0 | 1 | | 1 | 0 | 0 | 1 | 0 | | 1 | 0 | 1 | 0 | 1 | | 1 | 1 | 0 | 0 | 1 | | 1 | 1 | 1 | 1 | 1 |
            - **Boolean Expressions (Derived from K-maps for S and Cout):**
                - $S=A \oplus B \oplus Cin$**Error! Filename not specified.**
                - $Cout=(A \cdot B)+(Cin \cdot (A \oplus B))$**Error! Filename not specified.**
            - **Logic Circuit Diagram:** Draw the circuit. The most common way is to show it constructed using **two Half-Adders and one OR gate**. Clearly label inputs A, B, Cin and outputs S, Cout.

Unit 3: Sequential Circuits and Systems

Syllabus Focus: 1-bit memory, Bistable latch, clocked SR flip flop, J-K-T and D-types flip flops, applications of flip flops, shift registers, serial to parallel converter, parallel to serial converter, ring counter, sequence generator, ripple (Asynchronous) counters, synchronous counters, counter design using flip flops.

Key 6-Mark Questions & How to Answer:

1. **Explain SR Flip flop in detail. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** Define a flip-flop as a fundamental 1-bit memory element in digital electronics, capable of storing a binary '1' or '0'. The SR (Set-Reset) flip-flop is one of the most basic types.
        - **Logic Diagram:** Draw the **NOR gate based SR Latch** (usually the starting point) or a **clocked SR flip-flop**.
            - *For NOR Latch:* Two cross-coupled NOR gates. Inputs S, R. Outputs Q, Q.
            - *For Clocked SR FF:* Add two NAND gates at the input to control S and R inputs of the latch using a Clock pulse.
        - **Working Principle (for Clocked SR FF):**
            - **Clock (CLK) = 0:** No change in output (FF retains its previous state), as inputs to the latch are always 1.
            - **Clock (CLK) = 1:**
                - **S=0, R=0:** No change. (Inputs to latch are 1,1; previous state maintained)
                - **S=0, R=1:** Reset. Q becomes 0, Q becomes 1. (Inputs to latch are 1,0; FF resets)
                - **S=1, R=0:** Set. Q becomes 1, Q becomes 0. (Inputs to latch are 0,1; FF sets)
                - **S=1, R=1:** Invalid/Forbidden state. Both Q and Q try to go to 0, which is a contradictory state. This state must be avoided.
        - **Truth Table (or Characteristic Table for Clocked SR FF):** | CLK | S | R | Qn+1 (Next State) | Qn+1 | Remarks | |-----|---|---|-----------------------|-----------------------|--------------| | 0 | X | X | Qn | Qn | No change | | 1 | 0 | 0 | Qn | Qn | No change | | 1 | 0 | 1 | 0 | 1 | Reset | | 1 | 1 | 0 | 1 | 0 | Set | | 1 | 1 | 1 | Invalid | Invalid | Forbidden |
        - **Disadvantage:** The main disadvantage is the "Forbidden" or "Invalid" state (S=1, R=1) where the output is unpredictable if S and R return to 0 simultaneously. This makes it impractical for many applications.
2. **Write and explain any two applications of flip-flop. (6 Marks)**

- **Answer Structure:**
    - **Introduction:** Define flip-flops as fundamental building blocks for digital memory and sequential logic circuits.
    - **Application 1: Data Storage / Registers:**
        - **Explanation:** A single flip-flop can store one bit of binary data. A collection of 'N' flip-flops, connected together, forms an N-bit register, capable of storing an N-bit binary word. Data is loaded into the register synchronously with a clock pulse.
        - **Diagram:** Draw a simple 4-bit parallel-in, parallel-out register (four D-flip-flops connected with a common clock and data inputs).
        - **Importance:** Forms the basis of CPU registers, memory units (like RAM), and temporary data storage.
    - **Application 2: Frequency Division:**
        - **Explanation:** A T-flip-flop (Toggle flip-flop) or a JK flip-flop configured in toggle mode (J=K=1) can act as a frequency divider. When clocked, its output toggles with each clock pulse, effectively dividing the input clock frequency by 2.
        - **Diagram:** Draw a single T-flip-flop or JK-flip-flop (with J and K tied to 1) and show an input clock waveform and the output waveform, demonstrating the division by 2.
        - **Importance:** Essential in clock generation circuits, timing circuits, and creating lower frequencies from a higher master clock.
    - **Application 3 (Optional - choose if you have more detail): Counters:**
        - **Explanation:** Flip-flops are the core components of counters. By cascading flip-flops, you can create circuits that count in a specific sequence (e.g., binary up/down counters, BCD counters). Each flip-flop represents a bit in the count.
        - **Importance:** Used in digital clocks, event counters, timers, and sequence generators.

3. **Design 3-bit synchronous up counter using JK flip-flops. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** A synchronous counter is one where all flip-flops are clocked simultaneously by the same clock signal, leading to faster operation and avoiding propagation delays. We will design a 3-bit up counter, which counts from 000 to 111 and then rolls over.
        - **Steps:**

1. **State Diagram/Table:**
    - Draw the state diagram (000 -> 001 -> ... -> 111 -> 000).

- Create the **State Table:** | Present State (Q2Q1Q0) | Next State (Q2+Q1+Q0+) | | :------------------------- | :------------------------- | | 000 | 001 | | 001 | 010 | | 010 | 011 | | 011 | 100 | | 100 | 101 | | 101 | 110 | | 110 | 111 | | 111 | 000 |

2. **Excitation Table of JK Flip-Flop:** (Recall the input required for a given state transition) | Qn | Qn+1 | J | K | | :---- | :-------- | :-| :-| | 0 | 0 | 0 | X | | 0 | 1 | 1 | X | | 1 | 0 | X | 1 | | 1 | 1 | X | 0 |

3. **Derive Flip-Flop Input Equations:** Use the State Table and JK Excitation Table to find the J and K inputs for each flip-flop (J2,K2,J1,K1,J0,K0).

- Extend the State Table by adding columns for J2,K2,J1,K1,J0 ,K0.
- *Example derivation for* J0,K0*:*
  - Q0:0→1⟹J0=1,K0=X**Error! Filename not specified.**
  - Q0:1→0⟹J0=X,K0=1**Error! Filename not specified.**
  - Observe the pattern: Q0 toggles on every clock pulse. So, J0=1,K0=1.
- *Example derivation for* J1,K1*:*
  - Q1:0→0 if Q0=0⟹J1=0,K1 =X**Error! Filename not specified.**
  - Q1:0→1 if Q0=1⟹J1=1,K1 =X**Error! Filename not specified.**
  - So J1=Q0.
  - Similarly, Q1:1→1 if Q0=0⟹J1=X,K1 =0**Error! Filename not specified.**
  - Q1:1→0 if Q0=1⟹J1=X,K1 =1**Error! Filename not specified.**
  - So K1=Q0.
- Therefore, for J1,K1: J1=Q0, K1=Q0.
- *Derivation for* J2,K2*:*
  - Q2:0→0 if Q1Q0=00,01⟹J2=0,K2 =X**Error! Filename not specified.**
  - Q2:0→1 if Q1Q0=11⟹J2=1,K2 =X**Error! Filename not specified.**
  - So J2=Q1Q0.
  - Similarly, Q2:1→1 if Q1Q0=00,01⟹J2=X,K2 =0**Error! Filename not specified.**

- Q2:1→0 if Q1Q0=11⟹J2=X,K2 =1**Error! Filename not specified.**
- So K2=Q1Q0.

- Therefore, for J2,K2: J2=Q1Q0, K2=Q1Q0.

4. **Logic Circuit Diagram:** Draw the circuit using three JK flip-flops (FF0,FF1,FF2).
    - Connect the Clock to all FF's clock inputs.
    - Connect J0,K0 to HIGH (logic 1).
    - Connect J1,K1 to Q0.
    - Connect J2,K2 to an AND gate whose inputs are Q1 and Q0.
    - Label outputs Q2,Q1,Q0.
    - **Clarity:** Ensure the diagram is neat, connections are clear, and all components are labeled.

4. **Draw and explain serial in serial out shift register in detail. (6 Marks)**
    - **Answer Structure:**
        - **Definition:** A shift register is a sequential logic circuit that stores and shifts binary data. A Serial-In Serial-Out (SISO) shift register accepts data one bit at a time serially and outputs data one bit at a time serially.
        - **Working Principle:** Data is loaded bit by bit into the leftmost flip-flop, shifting the existing bits one position to the right with each clock pulse. After 'N' clock pulses (for an N-bit register), the first bit entered will appear at the rightmost output.
        - **Logic Circuit Diagram:** Draw an N-bit SISO shift register using N D-flip-flops (e.g., a 4-bit SISO).
            - Connect the output (Q) of each D-FF to the input (D) of the next D-FF in sequence.
            - All flip-flops share a common clock (CLK) input.
            - The data input (Serial Data In) goes to the D input of the first flip-flop.
            - The final output (Serial Data Out) is the Q output of the last flip-flop.
            - *Diagram Example (4-bit SISO):*
            - Serial In --D-> FF0 --Q-> D-- FF1 --Q-> D-- FF2 --Q-> D-- FF3 --Q-> Serial Out
            - ^ ^ ^ ^
            - |---------|---------|---------|-- CLK (common for all FFs)

- **Operation Example (Step-by-step for a 4-bit SISO with input data 1011):**
    - Assume initial state: Q3 Q2 Q1 Q0 = 0000
    - **CLK 1 (Input 1):** Q3 Q2 Q1 Q0 = 1000 (1 enters FF3)
    - **CLK 2 (Input 1):** Q3 Q2 Q1 Q0 = 1100 (1 enters FF3, previous 1 shifts to FF2)
    - **CLK 3 (Input 0):** Q3 Q2 Q1 Q0 = 0110 (0 enters FF3, 1 shifts to FF2, 1 shifts to FF1)
    - **CLK 4 (Input 1):** Q3 Q2 Q1 Q0 = 1011 (1 enters FF3, 0 shifts to FF2, 1 shifts to FF1, 1 shifts to FF0)
    - Now data 1011 is loaded. To read:
    - **CLK 5:** Q3 Q2 Q1 Q0 = 0101 (Serial Out = 1)
    - **CLK 6:** Q3 Q2 Q1 Q0 = 0010 (Serial Out = 1)
    - **CLK 7:** Q3 Q2 Q1 Q0 = 0001 (Serial Out = 0)
    - **CLK 8:** Q3 Q2 Q1 Q0 = 0000 (Serial Out = 1)
- **Timing Diagram (Optional but highly recommended):** Show CLK, Serial Data In, and outputs Q3, Q2, Q1, Q0 waveforms over several clock cycles to illustrate the shifting process.

Unit 4: Fundamentals of Microprocessors

Syllabus Focus: Fundamentals of Microprocessor, Comparison of 8-bit (8085), 16-bit (8086), and 32-bit (80386) microprocessors, The 8086 Architecture: Internal Block Diagram, CPU, ALU, address, data and control bus, Working registers, SFRs, Clock and RESET circuits, Stack and Stack Pointer, Program Counter, I/O ports, Memory Structures, Data and Program Memory, Timing diagrams and Execution Cycles.

Key 6-Mark Questions & How to Answer:

1. **Comparison of 8-bit (8085), 16-bit (8086), and 32-bit (80386) microprocessors. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** Briefly state that microprocessors are classified by their data bus width, which indicates the amount of data they can process at once. This comparison highlights the evolution of microprocessor technology.
        - **Comparison Table:** Create a comprehensive table focusing on key differentiating features. Aim for 7-8 points. | Feature | 8085 (8-bit) | 8086 (16-bit) | 80386 (32-bit) | | :---------- | :---------- | :---------- | | **Data Bus Width** | 8 bits | 16 bits | 32 bits | | **Address Bus Width** | 16 bits | 20 bits | 32 bits | | **Memory Capacity** | 64 KB (2^16) | 1 MB (2^20) | 4 GB (2^32) | | **Architecture** | Von Neumann | Segmented (BIU & EU) | Segmented & Paged | | **Operating Modes** | Single mode | Minimum/Maximum modes | Real, Protected, Virtual 8086 | | **Pipelining** | No | Yes (Instruction Queue) | Yes (6-stage) | | **Memory Management** | None | Segmentation | Segmentation & Paging Unit (MMU)| | **Cache** | No | No | Yes (On-chip, 8KB) | | **Floating Point Unit** | No (external 8087) | No (external 8087) | Yes (integrated 80387DX or external)| | **Complexity** | Simpler | More complex than 8085 | Highly complex |
        - **Conclusion:** Briefly summarize how each generation offered significant improvements in processing power, memory addressing, and feature sets, laying the groundwork for modern computing.
2. **Draw and explain 8086 Internal Block Diagram. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** The 8086 is a 16-bit microprocessor with a pipelined architecture, internally divided into two main functional units to improve performance: the Bus Interface Unit (BIU) and the Execution Unit (EU).
        - **Block Diagram:** Draw a clear and well-labeled diagram of the 8086 architecture.
            - **Major Sections:** Clearly demarcate BIU and EU.
            - **BIU Components:**
                - Segment Registers (CS, DS, SS, ES): 16-bit, used for memory segmentation.

        - Instruction Pointer (IP): 16-bit, holds offset of next instruction.
        - Address Adder: Combines segment and offset to generate 20-bit physical address.
        - Instruction Queue (Prefetch Queue): 6-byte FIFO buffer.
        - Bus Control Logic: Manages bus cycles for fetching, reading, writing.
    - **EU Components:**
        - General Purpose Registers (AX, BX, CX, DX): 16-bit, can be accessed as 8-bit.
        - Pointer and Index Registers (SP, BP, SI, DI): 16-bit, used for memory addressing.
        - ALU (Arithmetic Logic Unit): Performs arithmetic and logical operations.
        - Flag Register (Status Register): 16-bit, stores status flags (Carry, Zero, Sign, Overflow, etc.) and control flags.
        - Control Unit: Decodes and executes instructions, generates control signals.
- **Explanation of Units and Key Components:**
    - **Bus Interface Unit (BIU):** Responsible for all external bus operations like fetching instructions, reading/writing data from/to memory or I/O ports. It manages the 20-bit address bus and 16-bit data bus. The instruction queue helps in pipelining (fetching next instructions while current one executes).
    - **Execution Unit (EU):** Responsible for executing instructions. It contains the ALU, general-purpose registers, and the control unit. It receives instructions from the BIU's queue, decodes them, and performs the necessary operations. It does not directly interact with the system buses.
    - **Registers:** Briefly explain the categories:
        - **General Purpose (AX, BX, CX, DX):** For data manipulation.
        - **Segment (CS, DS, SS, ES):** For memory segmentation.
        - **Pointer/Index (SP, BP, SI, DI, IP):** For addressing memory locations, stack operations, and instruction sequencing.
    - **ALU:** Performs arithmetic (+,-,*,/) and logical (AND, OR, NOT, XOR) operations.
    - **Flag Register:** Stores status of arithmetic/logical operations and controls certain CPU functions.

3. **Draw the pin diagram of 8086 and explain in brief. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** The 8086 is a 40-pin Integrated Circuit (IC) capable of operating in two modes: Minimum Mode (for single-processor systems) and Maximum Mode (for multi-processor systems). Its pins can be grouped based on their function.
        - **Pin Diagram:** Draw a clear 40-pin outline, numbering the pins from 1 to 40. Show common pin assignments and multiplexed pins.
            - *Example Sketch:*
            - (1) GND VCC (40)
            - (2) AD14 AD15 (39)
            - ... (A/D lines) ...
            - (16) AD0 A19/S6 (35)
            - ... (Control lines) ...
            - (18) CLK READY (22)
            - (21) RESET TEST (23)
            - (24) MN/MX INTR (18)
            - ... (Mode specific/Interrupts) ...
        - **Explanation of Key Pins (Select 10-12 crucial ones and briefly describe their function):**
            - **AD0-AD15 (Pins 16-2):** Multiplexed Address/Data lines. During the first clock cycle, they carry lower 16-bit address; then data.
            - **A16/S3 - A19/S6 (Pins 35-38):** Multiplexed Higher Address/Status lines. Carry upper 4-bit address during the first clock cycle, then status information (segment register used, etc.).
            - RD **(Pin 32):** Read signal. Active LOW, indicates data is being read from memory/IO.
            - WR **(Pin 29):** Write signal. Active LOW, indicates data is being written to memory/IO.
            - **ALE (Pin 25):** Address Latch Enable. Indicates that the multiplexed AD lines carry valid address information.
            - **CLK (Pin 19):** Clock input. Provides timing for all internal and external operations.
            - **RESET (Pin 21):** Active HIGH. Resets the processor, setting IP to FFFF0H.

        - **INTR (Pin 18):** Maskable Interrupt Request input.
        - **NMI (Pin 17):** Non-Maskable Interrupt input.
        - **READY (Pin 22):** Input from external devices indicating they are ready for data transfer. CPU enters wait states if not ready.
        - **MN/**MX **(Pin 33):** Minimum/Maximum mode selection. HIGH for Minimum, LOW for Maximum.
        - **VCC (Pin 40), GND (Pin 1, 20):** Power supply pins (+5V) and Ground.
        - **Specific for Max Mode (briefly):**
            - S0, S1, S2 (Pins 26-28): Status signals used by 8288 bus controller.
            - RQ/GT0, RQ/GT1 (Pins 30-31): Request/Grant lines for bus mastership in multi-processor systems.
    - **Clarity:** Use clear labels and concise descriptions for each pin. Grouping pins (e.g., Address/Data, Control, Power) helps in readability.

Unit 5: 8086 Instruction Set and Programming

Syllabus Focus: Memory Interfacing, I/O Interfacing, Direct Memory Access (DMA), Interrupts in 8086, 8086 Instruction Set and Programming: Addressing modes, Instruction syntax, Data types, Subroutines, Immediate addressing, Register addressing, Direct addressing, Indirect addressing, Relative addressing, Indexed addressing, Bit inherent addressing, bit direct addressing, Instruction timings, Data transfer instructions, Arithmetic instructions, Logical instructions, Branch instructions, Subroutine instructions, Bit manipulation instruction, Assembly language programs, C language programs, Assemblers and compilers, Programming and debugging tools.

Key 6-Mark Questions & How to Answer:

1. **Explain different type of Addressing modes of 8086 with examples. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** Addressing modes are ways of specifying the location of an operand (data to be operated upon) for an instruction. The 8086 has a powerful set of addressing modes to access memory locations and registers efficiently.
        - **Explain 4-5 diverse and important modes with clear examples:**
            1. **Immediate Addressing Mode:**
                - **Definition:** The operand is a part of the instruction itself. No memory access is needed for the operand.
                - **Syntax:** Opcode Destination, Immediate_Value
                - **Example:** MOV AX, 1234H (Moves the immediate value 1234H into register AX)
                - **Explanation:** The value 1234H is directly encoded within the instruction.
            2. **Register Addressing Mode:**
                - **Definition:** The operand is located in one of the 8086's internal registers.
                - **Syntax:** Opcode Destination_Reg, Source_Reg
                - **Example:** MOV BX, CX (Copies the content of CX register into BX register)
                - **Explanation:** Both operands are directly available in CPU registers, making this a very fast mode.
            3. **Direct Addressing Mode:**
                - **Definition:** The effective address of the operand in memory is directly specified as a 16-bit displacement within the instruction. The physical address is calculated as DS:EA.
                - **Syntax:** Opcode Destination, [Memory_Address]

        - **Example:** MOV AX, [2000H] (Copies the word from memory location 2000H (offset) in the data segment into AX)
        - **Explanation:** The constant offset (2000H) is part of the instruction.
    4. **Register Indirect Addressing Mode:**
        - **Definition:** The effective address of the operand is held in a base register (BX or BP) or an index register (SI or DI).
        - **Syntax:** Opcode Destination, [Register] (where Register is BX, BP, SI, or DI)
        - **Example:** MOV AX, [BX] (Copies the word from the memory location pointed to by BX in the data segment into AX)
        - **Explanation:** The content of BX acts as the offset. This is useful for accessing data arrays.
    5. **Based Indexed Addressing Mode:**
        - **Definition:** The effective address is calculated by summing a base register (BX or BP), an index register (SI or DI), and an optional 8-bit or 16-bit displacement.
        - **Syntax:** Opcode Destination, [Base_Reg + Index_Reg + Displacement]
        - **Example:** MOV AX, [BX + SI + 10H] (Copies the word from memory location (DS:BX+SI+10H) into AX)
        - **Explanation:** Provides flexible access to elements within arrays or structures.
- **Optimization:** Focus on distinct modes with clear, simple examples. Avoid complex derivations unless specifically asked.

2. **Explain classification of instruction set of 8086. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** The 8086 instruction set is a collection of commands that the microprocessor understands and executes. These instructions are grouped into various categories based on their function, enabling programmers to perform diverse operations.
        - **Categories (List and explain 5-6 major ones with examples):**
            1. **Data Transfer Instructions:**
                - **Purpose:** Move data between registers, memory, and I/O ports. They do not affect flag bits.
                - **Examples:**
                    - MOV AX, BX (Move content of BX to AX)

        - PUSH AX (Push content of AX onto stack)
        - POP BX (Pop content from stack to BX)
        - XCHG AX, BX (Exchange content of AX and BX)
        - IN AL, 08H (Input byte from port 08H to AL)
        - OUT 0AH, AL (Output byte from AL to port 0AH)
2. **Arithmetic Instructions:**
    - **Purpose:** Perform arithmetic operations like addition, subtraction, multiplication, and division on binary or BCD numbers. These instructions affect flags.
    - **Examples:**
        - ADD AX, BX (AX = AX + BX)
        - SUB CL, 05H (CL = CL - 5)
        - MUL BX (AX = AL * BX for byte, DX:AX = AX * BX for word)
        - DIV CL (Divide AX by CL)
        - INC CX (Increment CX by 1)
        - DEC DL (Decrement DL by 1)
3. **Logical Instructions:**
    - **Purpose:** Perform bitwise logical operations (AND, OR, NOT, XOR, TEST) and bit shifting/rotation. These instructions affect flags.
    - **Examples:**
        - AND AL, 0FH (Bitwise AND AL with 0FH)
        - OR BX, CX (Bitwise OR BX with CX)
        - NOT DH (Invert bits of DH)
        - XOR AX, AX (Clear AX to 0)
        - SHL CL, 1 (Shift CL left by 1 bit)
        - ROR BH, 2 (Rotate BH right by 2 bits)
4. **Control Transfer (Branching) Instructions:**
    - **Purpose:** Change the flow of program execution by modifying the Instruction Pointer (IP) or both IP and CS. Includes jumps, calls, returns, and loops.
    - **Examples:**

        - JMP LABEL (Unconditional jump to LABEL)
        - JE LABEL (Jump if Equal)
        - CALL PROCEDURE (Call a subroutine)
        - RET (Return from subroutine)
        - LOOP LABEL (Loop until CX is zero)
    5. **String Manipulation Instructions:**
        - **Purpose:** Operate on blocks of data (strings) stored in memory, performing byte-by-byte or word-by-word operations.
        - **Examples:**
            - MOVSB (Move byte from DS:SI to ES:DI)
            - CMPSB (Compare byte from DS:SI with ES:DI)
            - LODSW (Load word from DS:SI to AX)
            - STOSB (Store byte from AL to ES:DI)
    6. **Processor Control Instructions:**
        - **Purpose:** Control the operation of the processor itself, affecting flag bits or processor states.
        - **Examples:**
            - HLT (Halt processor execution)
            - NOP (No operation)
            - STC (Set Carry Flag)
            - CLC (Clear Carry Flag)
            - STI (Set Interrupt Flag - enable interrupts)
            - CLI (Clear Interrupt Flag - disable interrupts)
- **Clarity:** Provide concise explanations and simple, representative examples for each category.

3. **Write short note on assembler and compiler. (6 Marks)**
    - **Answer Structure:**
        - **Introduction:** Both assemblers and compilers are essential language translators used in software development to convert human-readable source code into machine-executable code. They differ primarily in the level of the input language they process.
        - **Assembler:**

        - **Definition:** An assembler is a software tool that translates assembly language programs into machine code. Assembly language uses mnemonics (symbolic codes) for machine instructions (e.g., MOV, ADD).
        - **Input:** Assembly language source code (e.g., .asm file).
        - **Output:** Machine code (object code, e.g., .obj file), which is directly executable by the CPU.
        - **Working Principle:** Performs a one-to-one translation (one assembly instruction typically translates to one machine instruction). It resolves symbolic addresses and labels into actual memory locations.
        - **Characteristics:**
            - **Low-level:** Closely tied to the specific hardware architecture of the CPU.
            - **Speed:** Generally faster execution as it provides direct hardware control.
            - **Memory Efficiency:** Produces compact and efficient code.
            - **Machine-dependent:** Code written for one processor's assembly language cannot run on another.
        - **Use Cases:** System programming, device drivers, embedded systems, performance-critical applications.
- **Compiler:**
        - **Definition:** A compiler is a software tool that translates programs written in a high-level programming language (HLL) into machine code or an intermediate code. HLLs are more abstract and human-friendly (e.g., C, C++, Java, Python).
        - **Input:** High-level language source code (e.g., .c, .cpp file).
        - **Output:** Machine code, assembly code, or an intermediate bytecode/object code.
        - **Working Principle:** Involves multiple phases like lexical analysis, parsing, semantic analysis, optimization, and code generation. It often translates one HLL statement into multiple machine instructions.
        - **Characteristics:**
            - **High-level:** More abstract, provides stronger abstractions from hardware details.
            - **Portability:** Code can often be compiled and run on different hardware platforms (with appropriate compilers).

        - **Productivity:** Easier to write and maintain complex programs.
        - **Machine-independent:** Focuses on algorithms rather than hardware specifics.
    - **Use Cases:** Application development, web development, data science, general-purpose programming.
- **Key Differences (Summarize briefly):**
    - **Input Language:** Assembler -> Assembly, Compiler -> HLL
    - **Output:** Both -> Machine code (Assembler more direct, Compiler can produce intermediate)
    - **Abstraction Level:** Assembler -> Low, Compiler -> High
    - **Portability:** Assembler -> Low, Compiler -> High
    - **Translation Ratio:** Assembler -> 1:1, Compiler -> 1:Many
    - **Error Reporting:** Assembler -> Simpler, Compiler -> More complex/detailed.

General Exam Writing Tips for 6-Mark Questions:

- **Structure is Key:** Always use clear headings, subheadings, bullet points, and numbered lists. This makes your answer organized and easy for the examiner to follow.
- **Draw Clear Diagrams:** For any question involving circuits (gates, adders, flip-flops, counters, shift registers) or architecture (8086 block/pin diagram), **draw large, neat, and clearly labeled diagrams.** Diagrams can account for a significant portion of the marks. Use a pencil for diagrams so you can erase and redraw if needed.
- **Define Terms First:** Start every answer with a concise definition of the core concept.
- **Use Examples:** For concepts like addressing modes, instruction sets, or universal gates, providing relevant examples is crucial for demonstrating understanding.
- **Explain "How" and "Why":** Don't just list facts. Explain the working principle ("how it works") and the significance or advantages ("why it's used").
- **Truth Tables/Tables:** For comparative questions or logical functions, presenting information in tables (truth tables, comparison tables) is highly effective and readable.
- **Conciseness vs. Completeness:** For 6 marks, aim for a balanced answer. Provide enough detail to cover the necessary points without adding irrelevant information. A well-structured answer typically fits within half to one full page.
- **Problem-Solving Steps:** For design or minimization problems (like K-maps, counter design), clearly show all steps involved from the initial problem statement to the final solution.
- **Practice Drawing:** Regularly practice drawing the important diagrams (8086 block/pin, various circuits) to ensure you can replicate them quickly and accurately in the exam.

- **Time Management:** Be mindful of the time allocated for each 6-mark question. Don't overspend time on one question at the expense of others.

This expanded set of notes should provide you with a much more comprehensive and effective resource for your exam preparation. Focus on understanding these core concepts deeply, practicing the designs/problems, and mastering the art of clear, concise, and diagram-rich answers.