DSM Endsem 2024 Solutions

November 30, 2024

TA Distribution

- ▷ **1A, 1B** – Abraham Elenjical
- ▷ **2A**, **2B** – Abhinav Reddy Boddu, Yash Bhutada
- ▷ **3A**, **3B** – Aniketh Parkala
- ▷ **3C**, **5A** – Chalasani Bhavani
- ▷ **4B**, **5B** – Autrio Das
- ▷ **5C** – Losetti Mourya
- ▷ **4A** – Malla Sailesh
- ▷ **1C**, **6C** – Srihari Bandarupalli
- ▷ **6A**, **6B** – Gowlapalli Rohit

1A

What is the need of microcontrollers over general purpose microprocessor? [2 M]

Microcontrollers act as a microcomputer without any external digital parts. **Memory, I/O ports, timers, interrupts are all integrated** inside the microcontroller chip. The higher integration inside a microcontroller **reduces cost and size** of the system.

Sr. No.	Microprocessors	Microcontroller
1	It is only a general-purpose computer CPU	It is a microcomputer itself
2	Higher accessing time required	Low accessing time

Table 1: Comparison of Microprocessors and Microcomputers

Grading scheme: Mentioning any 2 of the above points (those in the table or those in bold) will get you 2 marks. 1 mark for each point.

1B

Explain using the example of call instruction 'cd S B7' when [SP] = A9 and [PC]= 06. S is a Sign flag. [4 M]

Instruction Format: `cd <FL> xx`
Actions:

$$\begin{aligned} [\text{SP}] &\leftarrow [\text{SP}] - 1, \\ [[\text{SP}]] &\leftarrow [\text{PC}], \\ [\text{PC}] &\leftarrow \text{xx} \quad (\text{if } \langle \text{FL} \rangle = 1) \end{aligned}$$

Instruction	Control Signals	Select Signals
`cd<FL> xx`	Ck 3: E_{PC}, L_{MR}, I_{PC}, E_{FL}, End if $< FL >'$	$S_{FL} \leftarrow < FL >$
	Ck 4: RD, L_{OR}, D_{SP}	-
	Ck 5: E_{SP}, L_{MR}	-
	Ck 6: E_{PC}, WR	-
	Ck 7: E_{OR}, L_{PC}, End	-

Table 2: Control Signals for `cd<FL> xx` instruction.

Initial Conditions

- **[SP] (Stack Pointer):** $A9$
- **[PC] (Program Counter):** 06
- **Sign Flag (S):** The Sign flag copies the sign bit of the last arithmetic operation and becomes 1 if the result was negative. For the illustration of our example, we can assume it is 1.

Instruction Execution

1. **Check Condition (S Flag):**
   - If $S = 1$, proceed with the call. (In this case, the condition is satisfied.)
2. **Save Current [PC]:**

$$\begin{aligned} \text{Decrement [SP]:} \quad & [SP] = A9 - 1 = A8 \\ \text{Store [PC] at the new [SP] memory address:} \quad & [[SP]] = [PC] = 06 \end{aligned}$$

3. **Update [PC] to the Target Address (`xx`):**

$$[PC] = B7$$

Final Values

- **[SP]:** $A8$
- **[[SP]]:** 06
- **[PC]:** $B7$

Grading Scheme

- Mention the Actions/Control Signals required (simulate the working with the values mentioned in question) [2 M]
- Explain the working/different values of Sign Flag [1M]
- Correct final values of [SP], [[SP]] and [PC]. [1M]

Even if you have assumed S flag to be 0, actions/control signals should still be listed for the entire flow.

1C

How is ‘call‘ different from ‘jump‘? [2 M]

Call: The `call` instruction is used to invoke a subroutine (a function or procedure) and save the return address onto the stack so the program can return to the point it left off after the subroutine is executed.

Jump: The `jump` instruction simply changes the program's control flow to another location without saving any return address, making it a direct transfer of control.

2A

Architecture components

- **Registers:** Multipurpose registers (`R4`, `ACC` for accumulator).
- **ALU Operations:**
  - `ADD`: Adds two inputs:
    - Left input: From the bus.
    - Right input: From the Operand Register (`OR`).
  - The result of the ALU is written to the accumulator (`ACC`).
- **Control Signals:**
  - `E`: Enables a register to output data onto the bus.
  - `L`: Loads data from the bus into a register.
  - `SALUADD`: Sets the ALU operation to addition.

Steps for `ADD R4`

Ck 00: Load `R4` into Operand Register (`OR`)

- **Control Signals:** `E-R4`, `L-OR`
  - `E-R4`: Enable `R4` to output its value onto the bus.
  - `L-OR`: Load the value from the bus into the Operand Register (`OR`).

Ck 01: Perform Addition and Load Result into `ACC`

- **Control Signals:** `E-ACC`, `E-OR`, `L-ACC`, `S-ALU-ADD`
  - `E-ACC`: Enable `ACC` (current accumulator value) onto the bus.
  - `E-OR`: Enable `OR` to send its value to the ALU as the second operand.
  - `SALU-ADD`: Configure the ALU to perform addition.
  - `L-ACC`: Load the result from the ALU into the accumulator (`ACC`).

Timing Diagram

The following timing diagram represents the control signals:

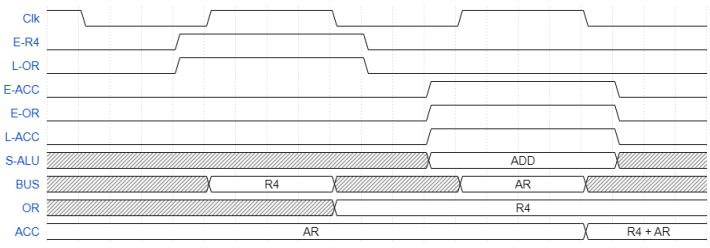


Figure 1: Timing Diagram

Grading Scheme

- `1 Mark`: Correctly explained and mentioned the operations in Ck 00.
- `2 Marks`: Correctly explained and mentioned the operations in Ck 01
- `2 Marks`: Correct timing diagram

Partial marks will be awarded in all other cases depending on the content.

2B

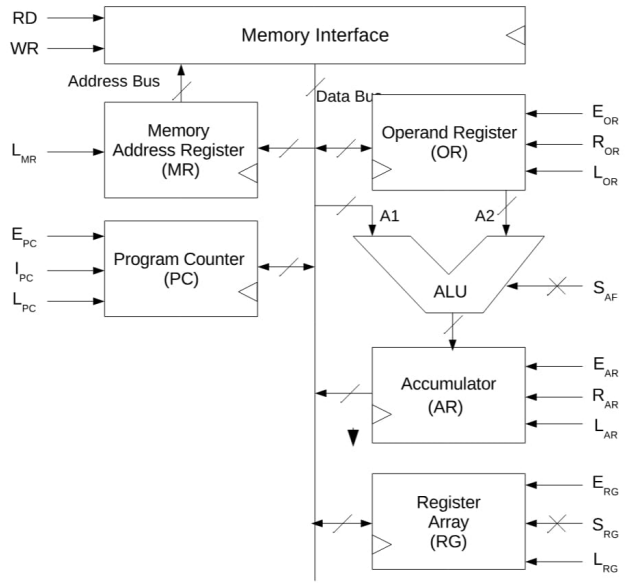


Figure 2: Complete Single bus architecture

1. Memory

The **Memory** block represents the Main Memory subsystem of the processor. The processor accesses main memory via the Memory Address Register (MR) and the Data Bus.

2. Address Bus

The **Address Bus** is responsible for transmitting memory addresses from the processor to the memory system. The Memory Address Register (MR) holds the address that the processor intends to access in memory.

3. Memory Address Register (MR)

The **Memory Address Register (MR)** stores the memory address to be accessed, either for reading data from or writing data to memory. It is connected to the Address Bus and specifies the memory location to be accessed

4. Data Bus

The **Data Bus** is used to transfer data between the processor and memory or I/O devices. Data can either be read from memory and placed on the Data Bus, or data can be written to memory from the Data Bus.

5. Operand Register (OR)

The **Operand Register (OR)** holds the operands (input data) for the current instruction being executed by the processor. These operands are passed to the Arithmetic Logic Unit (ALU) for computation.

6. Arithmetic Logic Unit (ALU)

The **Arithmetic Logic Unit (ALU)** is the core computational unit in the processor. It performs arithmetic and logical operations such as:

- **Arithmetic Operations**: Addition, subtraction, etc.
- **Logical Operations**: AND, OR, NOT, etc.

The results of these operations are stored in the Accumulator (AC) register.

7. Accumulator (AR)

The **Accumulator (AR)** is a register that stores the results of computations performed by the ALU. It holds intermediate and final results of the processor's calculations.

8. Program Counter (PC)

The **Program Counter (PC)** is a register that holds the address of the next instruction to be executed in the program. It increments after each instruction, ensuring that the processor fetches the next instruction from memory.

9. Register Array (RG)

The **Register Array (RG)** is a collection of general-purpose registers. These registers store intermediate data and addresses during program execution. The registers can be accessed and manipulated by the program instructions.

Grading Scheme

- `3.5 Marks:` for diagram - 0.5 for each component - Memory, MR, OR, ALU, AR, PC, RG. (no specific marks for Address bus and Memory bus)
- `4.5 Marks:` for explanation - 0.5 for each component.

3A

1. ROM is a type of non-volatile memory used to store fixed binary information.
2. ROM consists of a decoder and OR gates, forming a minterm generator within a single device.

Fusible Links in PROM (for reference)

In a **Programmable ROM (PROM)**, each intersection in the matrix contains a fusible link. These links are initially connected (representing a binary '1').

Fusing a connection involves applying a high current to "blow" the link, effectively disconnecting it:

- **Blown fuse** = binary '0'
- **Intact fuse** = binary '1'

How Fusing Helps

Any one of the points below:

Custom Data Storage

Fusing specific connections allows storing customized data patterns. Each address line combination corresponds to a pre-defined output word, set during the programming phase.

Permanence

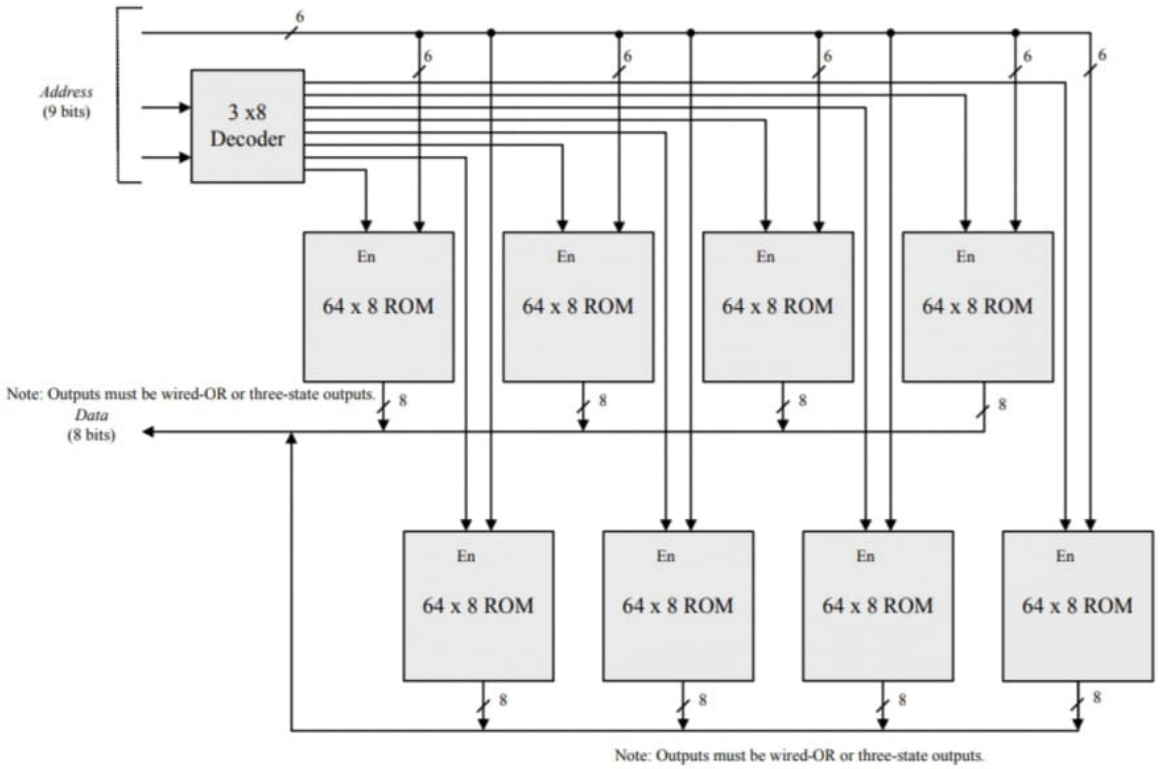
Once a fuse is blown, the ROM's data is fixed and cannot be changed during normal operation. This makes PROM ideal for storing information that does not require modification.

Reliability

Since data is hardwired through these fuses, it is very stable and immune to electrical noise or accidental changes, ensuring reliable operation.

3B

Address multiplexing is used to minimize the number of address pins by splitting an address into multiple parts, typically row and column addresses.



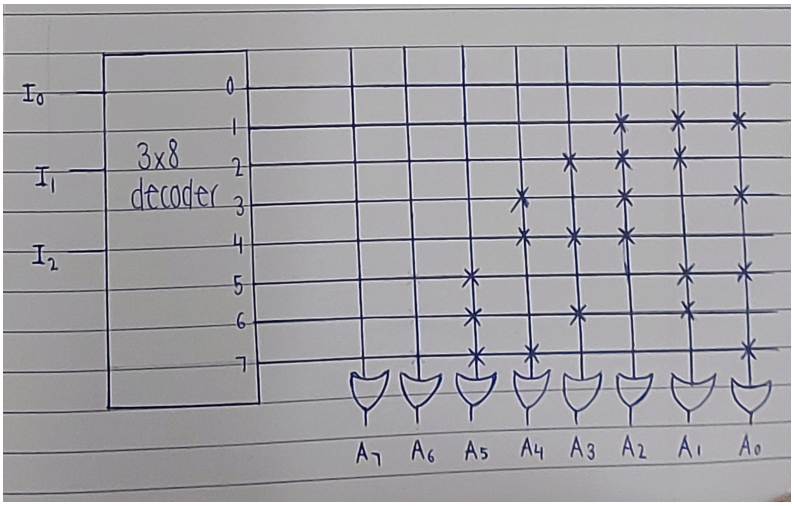
3C

The truth table for the required function, where the output is the input multiplied by 7, is as follows:

Inputs			Outputs							
I_2	I_1	I_0	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	1	1	1	0
0	1	1	0	0	0	1	0	1	0	1
1	0	0	0	0	0	1	1	1	0	0
1	0	1	0	0	1	0	0	0	1	1
1	1	0	0	0	1	0	1	0	1	0
1	1	1	0	0	1	1	0	0	0	1

By choosing connections for the minterms included in the function, the ROM outputs can be programmed to represent the function.

Here, we use a 3x8 decoder, and a connection marked with an x produces a minterm.



4A

Implement Boolean function F(A,B,C,D) =S(2, 4, 5, 7, 9, 11) using

Map

	$\overline{C}.\overline{D}$	$\overline{C}.D$	C.D	C.$\overline{D}$
$\overline{A}.\overline{B}$	0	0	0	1
$\overline{A}$.B	1	1	1	0
A.B	0	0	0	0
A.$\overline{B}$	0	1	1	0

Map Layout

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

Groups

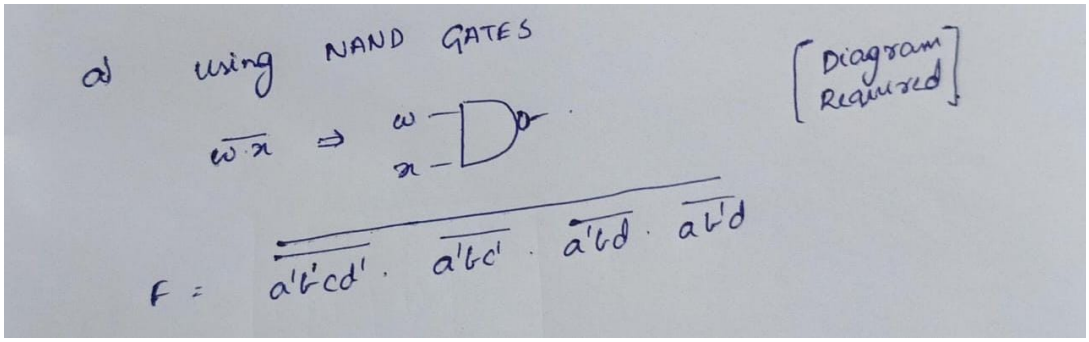
(4,5)	$\overline{A}$.B.$\overline{C}$
(5,7)	$\overline{A}$.B.D
(9,11)	A.$\overline{B}$.D
(2)	$\overline{A}.\overline{B}.C.\overline{D}$

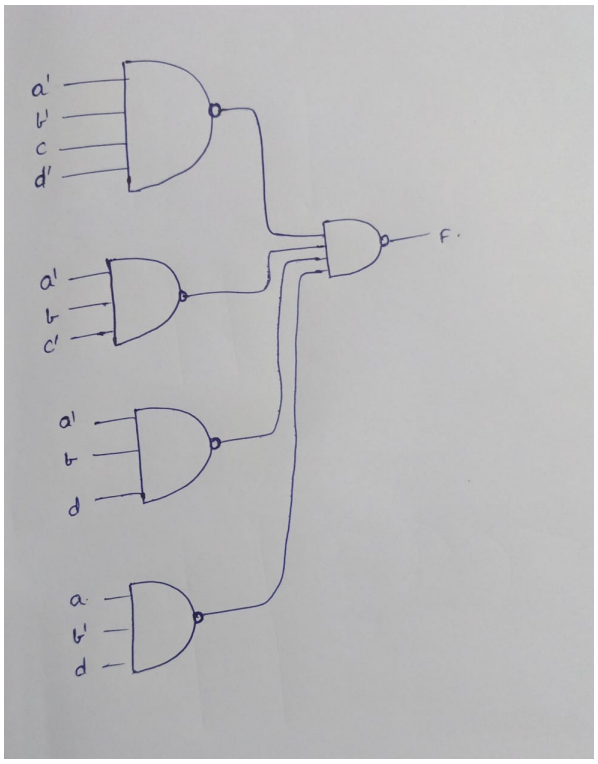
y = A'BC' + A'BD + AB'D + A'B'CD'

Truth Table

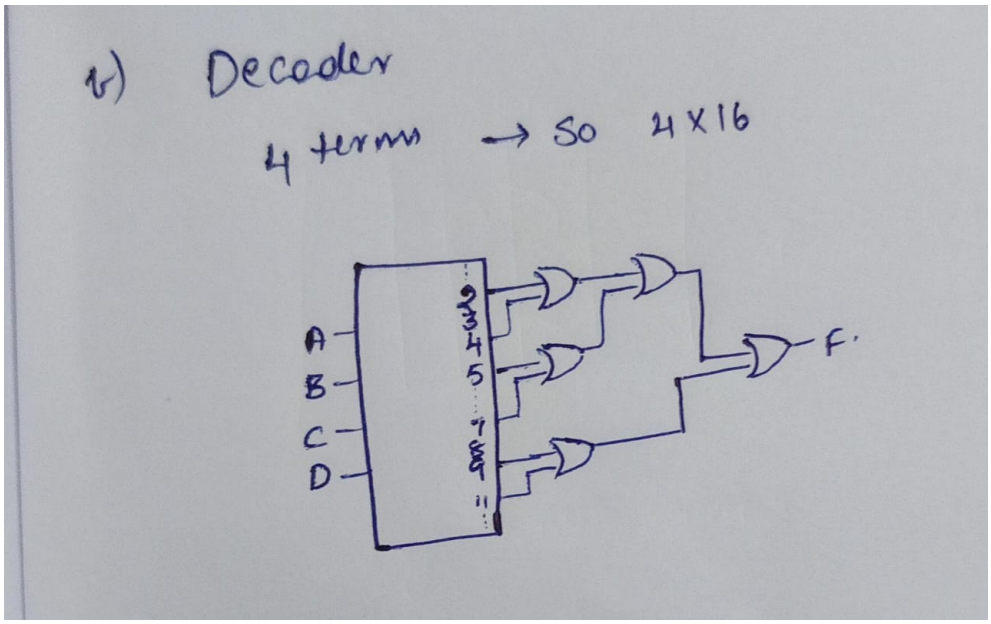
	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	1
3	0	0	1	1	0
4	0	1	0	0	1
5	0	1	0	1	1
6	0	1	1	0	0
7	0	1	1	1	1
8	1	0	0	0	0
9	1	0	0	1	1
10	1	0	1	0	0
11	1	0	1	1	1
12	1	1	0	0	0
13	1	1	0	1	0
14	1	1	1	0	0
15	1	1	1	1	0

(i) NAND Gates





(ii) Decoder



4B

How to convert SR latch into memory cell? [3 M]

Acceptable circuit diagrams :

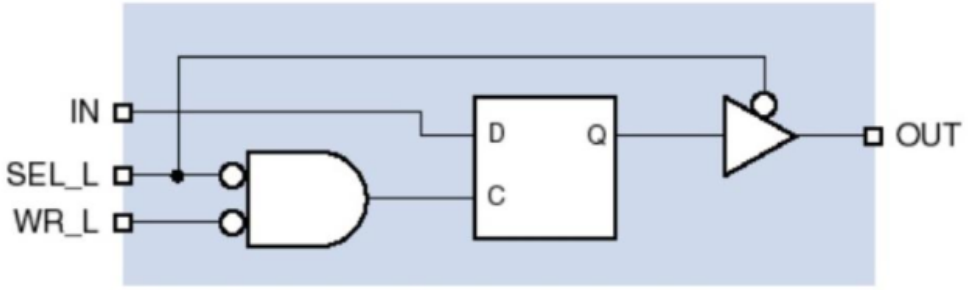


Figure 3: Memory cell using D latch

The binary storage cell (SR latch) is the basic building block of a memory unit The storage part of the cell is modeled by an SR latch with associated gates to form a D latch Note that this is not a D flip-flop

---

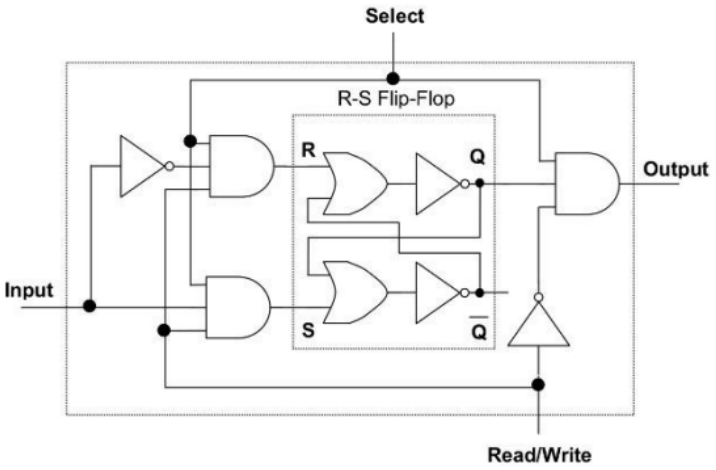


Figure 4: Memory cell using SR latch

The "select" input is used to access the cell, either for reading or writing. When the select line is high, "1", then a memory operation can be performed on this cell. When the select line of the binary cell is low, "0", then the cell is not being read from or written If the clock value on the "Read/write" line is low (which makes the "negated Read/write" high) indicating the cell contents are to be read. In this case, the value output by the cell will depend solely on the Q value of the flip-flop. When the cell is selected and the "Read/write" line is set to high, signifying a "write" operation, the value placed into the cell will depend solely on the state of the "Input" lin

Grading Scheme

1 Mark for correct SR-latch or D-latch diagram 2 Marks for working memory cell / correct conversion

5A

A Johnson counter is a circular shift register with the complemented output of the last flip-flop connected to the input of the first flip-flop (switch-tail ring counter) with decoding gates to provide outputs for timing signals.

5B

How many output states are possible for a 5 output Johnsson counter vs Binary counter? [2 M]

Johnsons Counter

An n-bit Johnsson counter is a synchronous counter that consists of n D-flip-flops connected output to input. The complemented output of the last flip-flop is connected to the input of the first flip-flop

in the case of a 5-bit Johnsons counter, we have the following circuit diagram and state table:

State	Q4	Q3	Q2	Q1	Q0
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
5	1	1	1	1	1
6	0	1	1	1	1
7	0	0	1	1	1
8	0	0	0	1	1
9	0	0	0	0	1
0	0	0	0	0	0

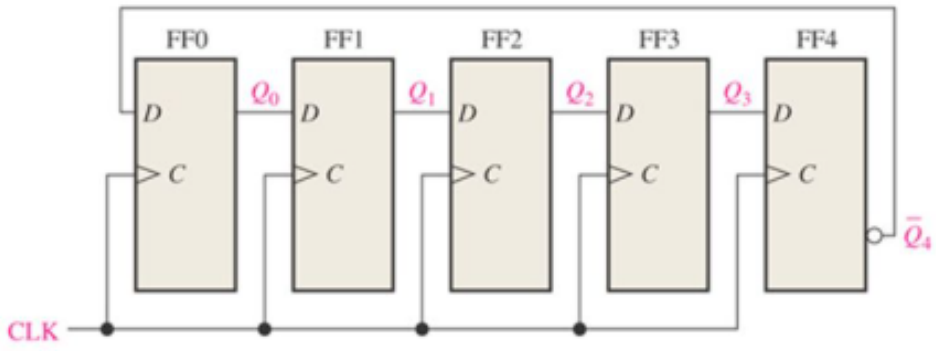


Figure 5: 5-bit Johnsons counter

As we can see from the circuit diagram, an n-bit johnsons counter will have $2 \times n$ states.

Binary counter

An n-bit Binary counter involves n J-K flip flops and has a regular pattern and can be constructed with complementing flip-flops and gates

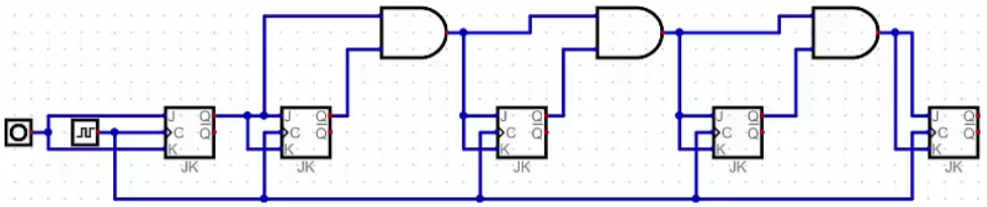


Figure 6: 5-bit Synchronous binary counter

Being a binary counter for n bits, it has 2^n distinct states. Thus a 5-bit Binary counter has 32 states

Grading Scheme

1 Mark for Johnsons counter explanation and number of states 1 marks for Binary counter explanation and number of states

5C

The given sequence is:

$$00 \rightarrow 11 \rightarrow 01 \rightarrow 10 \rightarrow 00$$

—

Step 1: State Transition Table The state transition table includes the current state (Q_1, Q_0), next state (Q_1^+, Q_0^+), and the required JK flip-flop inputs (J_1, K_1, J_0, K_0).

Q_1	Q_0	Q_1^+	Q_0^+	J_1	K_1	J_0	K_0
0	0	1	1	1	X	1	X
1	1	0	1	X	1	X	0
0	1	1	0	1	X	X	1
1	0	0	0	X	1	0	X

Flip-Flop Transition Rule: Each flip-flop input (J and K) is determined using the equation:

$$Q_{\text{new}} = JQ' + K'Q$$

Here, Q_{new} is the next state, Q is the current state, J is the set input, and K is the reset input. The values X denote "don't care" conditions.

—

Step 2: Karnaugh Maps for Flip-Flop Inputs Use the state transition table to derive Karnaugh maps for each JK flip-flop input.

J_1 **(Input for Q_1):**

$Q_1\,Q_0$	0	1
0	1	1
1	X	X

From the map:

$$J_1 = Q_1'$$

K_1 **(Input for Q_1):**

$Q_1\,Q_0$	0	1
0	X	X
1	1	1

From the map:

$$K_1 = Q_1$$

J_0 **(Input for Q_0):**

$Q_1\,Q_0$	0	1
0	1	X
1	0	X

From the map:

$$J_0 = Q_1'$$

K_0 **(Input for Q_0):**

$Q_1\,Q_0$	0	1
0	X	1
1	X	0

From the map:

$$K_0 = Q_1'$$

—

Step 3: Logic Circuit From the Karnaugh map derivations:

$$J_1 = Q_1', \quad K_1 = Q_1, \quad J_0 = Q_1', \quad K_0 = Q_1'$$

Step 4: Circuit Diagram

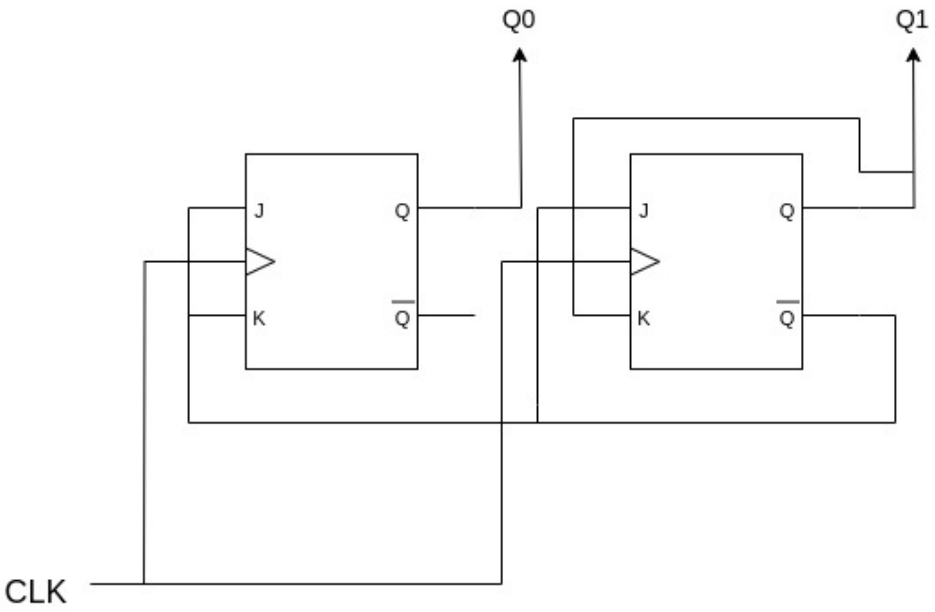


Figure 7: Synchronous Counter

Note: Step 2 is not compulsory

6A

Conversion to Base 10

Given $A69.8_{16}$,

$$A69.8_{16} = A \cdot 16^2 + 6 \cdot 16^1 + 9 \cdot 16^0 + 8 \cdot 16^{-1}$$

Substitute the values of hexadecimal digits:

$$= 10 \cdot 256 + 6 \cdot 16 + 9 \cdot 1 + 8 \cdot \frac{1}{16}$$

$$= 2560 + 96 + 9 + 0.5$$

$$= 2665.5_{10}$$

Conversion to Base 7

Step 1: Integer Part Conversion

$$2665_{10} \rightarrow \text{Divide by 7 repeatedly}$$

$$2665 \div 7 = 380 \quad \text{remainder } 5$$
$$380 \div 7 = 54 \quad \text{remainder } 2$$
$$54 \div 7 = 7 \quad \text{remainder } 5$$
$$7 \div 7 = 1 \quad \text{remainder } 0$$
$$1 \div 7 = 0 \quad \text{remainder } 1$$

Reading from bottom to top:

$$2665_{10} = 10525_7$$

Step 2: Fractional Part Conversion

$$0.5 \cdot 7 = 3.5 \quad \text{(integer part: 3)}$$

$$0.5 \cdot 7 = 3.5 \quad \text{(repeats)}$$

Thus, the fractional part is:

$$0.5_{10} \approx 0.3333_7$$

Final Answer

$$A69.8_{16} = 2665.5_{10} = 10525.3333_7$$

6B

i) Subtract 79 from 26

Step 1: Find the 9's Complement of 79

$$79 \quad \text{(Original Number)}$$

$$9\text{'s Complement of } 79 = 99 - 79 = 20$$

Step 2: Add the 9's Complement of 79 to 26

$$26 + 20 = 46$$

Step 3: Check for Carry - Since there is no carry, take the 9's complement of the result and place a negative sign.

$$9\text{'s Complement of } 46 = 99 - 46 = 53$$

$$26 - 79 = -53$$

—

ii) Subtract 748 from 983

Step 1: Find the 9's Complement of 748

$$748 \quad \text{(Original Number)}$$

$$9\text{'s Complement of } 748 = 999 - 748 = 251$$

Step 2: Add the 9's Complement of 748 to 983

$$983 + 251 = 1234$$

Step 3: Check for Carry - Since there is a carry (1), drop the carry and add 1 to the remaining result.

$$234 + 1 = 235$$

$$983 - 748 = 235$$

6C

Draw the logic diagram of a 4-to-1 line multiplexer with logic gates. [3 M]

A 4-to-1 multiplexer selects one of four input lines (I_0, I_1, I_2, I_3) based on the two select lines (S_0, S_1) and forwards the selected input to the output (Y). The logic diagram can be drawn using AND, OR, and NOT gates.

Logic Expression:

$$Y = (S_1' \cdot S_0' \cdot I_0) + (S_1' \cdot S_0 \cdot I_1) + (S_1 \cdot S_0' \cdot I_2) + (S_1 \cdot S_0 \cdot I_3)$$

Where: - S_1', S_0' are complements of S_1, S_0.

Logic Diagram: