Switching Theory – CS225

Name: M. Maheeth Reddy **Roll No.**: 1801CS31 **Date**: 15-July-2020

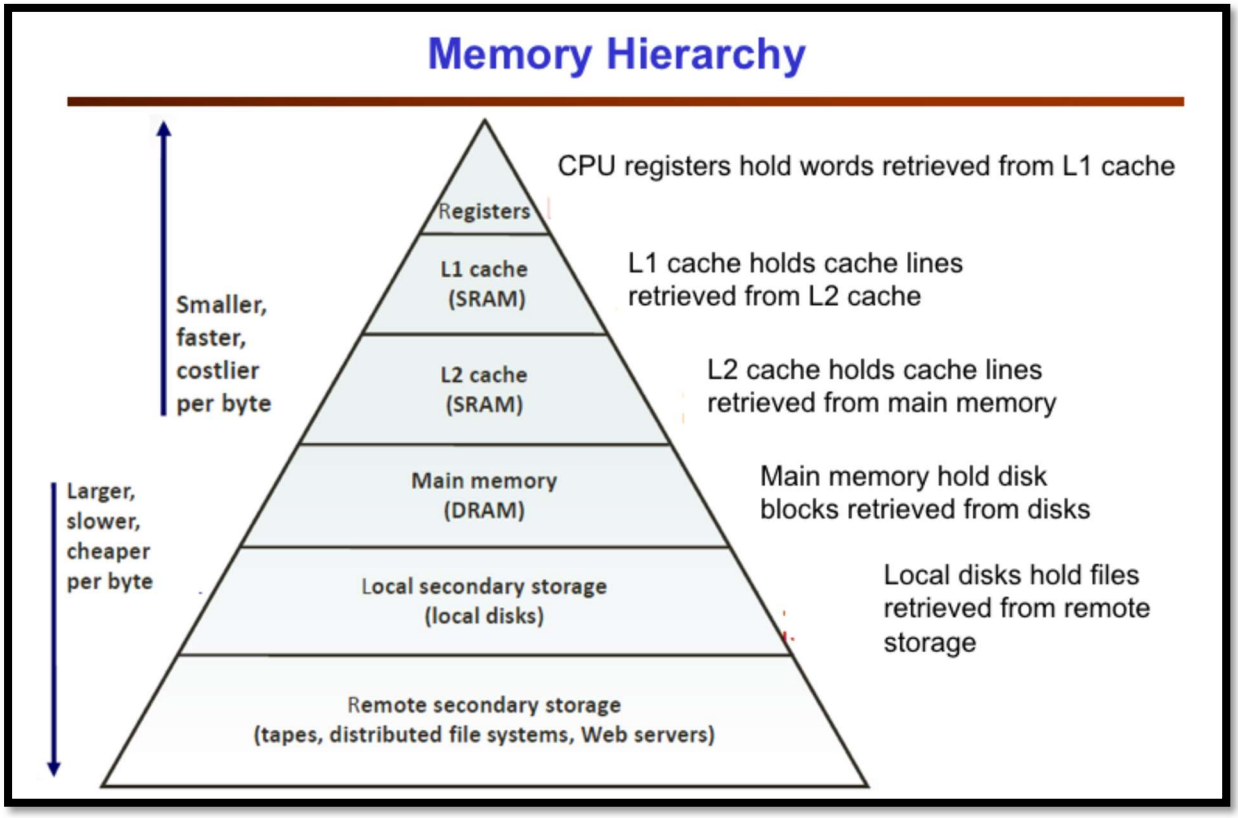
Spring 2020: End Semester Assignment

Ans 1:

Computer memory Architectures: Past and Present

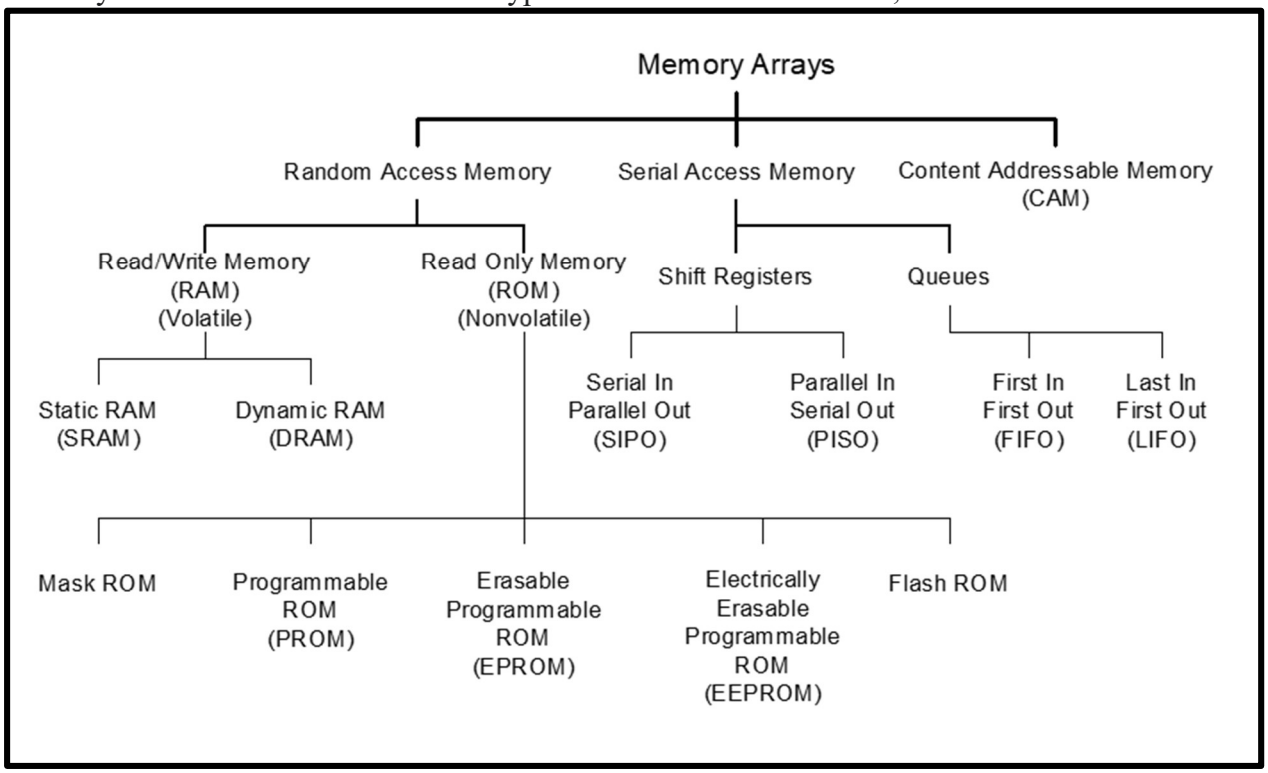
Computer memory refers to a device which stores information for immediate use in a computer or related computer hardware device.

In general, computer memory refers to semiconductor memory i.e., metal-oxide semiconductor memory. The MOS memory cells are fabricated on a silicon integrated circuit chip. The data is stored within the MOS memory cells.



Classification of Memory

Memory can be classified into various types based on the architecture, as shown below.



Random Access Memory

Random-access memory is a form of computer memory that can be accessed in any order. It is used to store working data and machine code. A random-access memory device allows data items to be read or written in almost the same amount of time irrespective of the physical location of data inside the memory.

Random access memory can be classified in to two types

1. Read / Write memory (RAM) or Volatile memory.
2. Read only memory (ROM) or Non-Volatile memory.

Volatile Memory:

It is the computer memory that requires power to retain the stored information.

This is classified in to two categories 1. Static RAM 2. Dynamic RAM

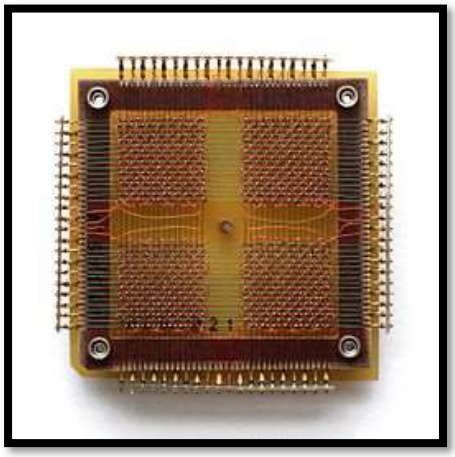
Static RAM uses six transistors to store one bit of data while Dynamic RAM uses one transistor and one capacitor to store one bit of data.

Evolution of RAM

The first form of RAM came into use in 1947 when Freddie Williams and Tom Kilburn developed the Williams-Kilburn tube. It used a cathode ray tube (similar to an analog TV picture tube) to store bits as dots on the screen's surface.

Manchester Mark I Williams-Kilburn tube

The second widely used form of RAM was magnetic-core memory, invented in 1947 by Frederick Viehe. It works through the use of tiny metal rings and wires connecting to each ring. One bit of data could be stored per ring and accessed at any time.



32 x 32 Magnetic core memory

Static random access memory (SRAM) chips built with the bipolar IC process became practical for high-speed computer applications in the mid-1960s. In 1970, Fairchild built memory systems for the ILLIAC IV supercomputer using 256-bit chips. These are used as Cache Memory in computers. In present day, computers have cache memory of 288 MB.

SRAM chip from a gaming console

However, RAM, as we know it today, as solid-state memory, was first invented in 1968 by Robert Dennard. Known specifically as dynamic random-access memory, or DRAM, transistors were used to store bits of data.

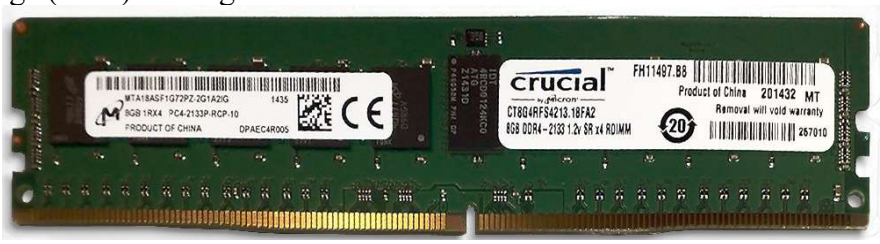
SDRAM (Synchronous Dynamic Random Access Memory): “Synchronous" tells about the behaviour of the DRAM type. In late 1996, SDRAM began to appear in systems. Unlike previous technologies, SDRAM is designed to synchronize itself with the timing of the CPU. This enables the memory controller to know the exact clock cycle when the requested data will be ready, so the CPU no longer has to wait between memory accesses.

DDR-SDRAM(Double Data Rate SDRAM):
The next generation of SDRAM is DDR, which achieves greater bandwidth than the preceding single data rate SDRAM by transferring data on the rising and falling edges of the clock signal. Effectively, it doubles the transfer rate without increasing the frequency of the clock. The transfer rate of DDR SDRAM is the double of SDR.

DDR2-SDRAM(Double Data Rate Two SDRAM):
Its primary benefit is the ability to operate the external data bus twice as fast as DDRSDRAM. This is achieved by improved bus signal.

DDR3 SDRAM (Double Data Rate Three SDRAM): DDR3 memory reduces 40% power consumption compared to current DDR2 modules, allowing for lower operating currents and voltages.

DDR4 SDRAM (Double Data Rate Fourth SDRAM): DDR4 SDRAM provides the lower operating voltage (1.2V) and higher transfer rate.



DDR4 SDRAM

Evolution of DRAM

DDR SDRAM Standard	Internal rate (MHz)	Bus clock (MHz)	Prefetch	Data rate (MT/s)	Transfer rate (GB/s)	Voltage (V)
SDRAM	100-166	100-166	1n	100-166	0.8-1.3	3.3
DDR	133-200	133-200	2n	266-400	2.1-3.2	2.5/2.6
DDR2	133-200	266-400	4n	533-800	4.2-6.4	1.8
DDR3	133-200	533-800	8n	1066-1600	8.5-14.9	1.35/1.5
DDR4	133-200	1066-1600	8n	2133-3200	17-21.3	1.2

Non-Volatile Memory (ROM): The computer memory that retains the stored information even after the power source is cut off, unlike volatile memory. It stores crucial information essential to operate the system, like the BIOS program which is essential to boot the computer. It is used in embedded systems or where the programming needs no change.

Evolution of ROM:

In 1974 The LaserDisc is introduced as “Discovision” by Philips.

In 1994 The Digital Video Disc (DVD) format is introduced, and its storage capacity is a huge increase over the common compact disc (CD).

Blu-ray disc: In 2003 the Blu-ray optical disc is released. It was designed to store high definition video at 1080p, while older DVDs were only capable of 480p resolution.

ROM is further classified into 4 types- Mask ROM, PROM, EPROM, and EEPROM.

Evolution of ROM

Mask ROM – This is the strictest form of ROM that contains the data permanently stored in it with no allowance for future modification.

PROM (Programmable read-only memory) – It can be programmed by user. Once programmed, the data and instructions in it cannot be changed.

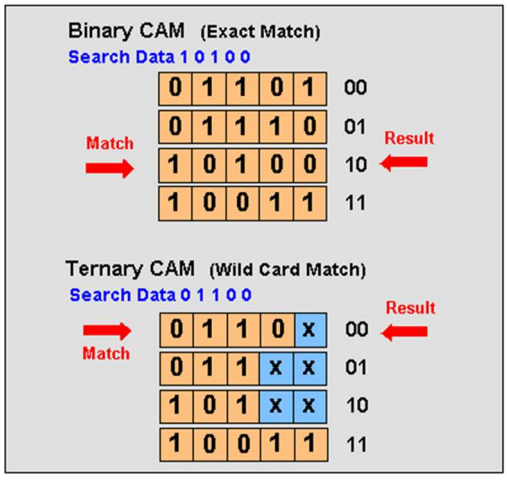
EPROM (Erasable Programmable read only memory) – It can be reprogrammed. To erase data from it, expose it to ultra violet light. To reprogram it, erase all the previous data.

EEPROM (Electrically erasable programmable read only memory) – The data can be erased by applying electric field, no need of ultra violet light. We can erase only portions of the chip.

Flash memory (or simply **flash**) is a modern type of EEPROM invented in 1984. Flash memory can be erased and rewritten faster than ordinary EEPROM, and newer designs feature very high endurance (exceeding 1,000,000 cycles). Modern NAND flash makes efficient use of silicon chip area, resulting in individual ICs with a capacity as high as 256 GB

Memory Cards use Flash Memory for storage

Content Addressable Memory (CAM) is also known as Associative Memory, in which the user supplies data word and associative memory searches its entire memory and if the data word is found, it returns the list of addresses where that data word was located.



Ans 2:

Voting Machine Controller

Overview:
A Voting Machine is used to record the votes during an election. It lets the voters cast their votes, stores the number of votes cast for each candidate in the election, displays the total votes cast and the votes cast for each candidate. After the result is declared, the machine is reset i.e., all memory is cleared.

The controller for a voting machine can be formulated as a Finite State Machine (FSM).

In the following design for the Voting Machine Controller, I have assumed that three candidates are contesting in an election.

Design:
States in the FSM:

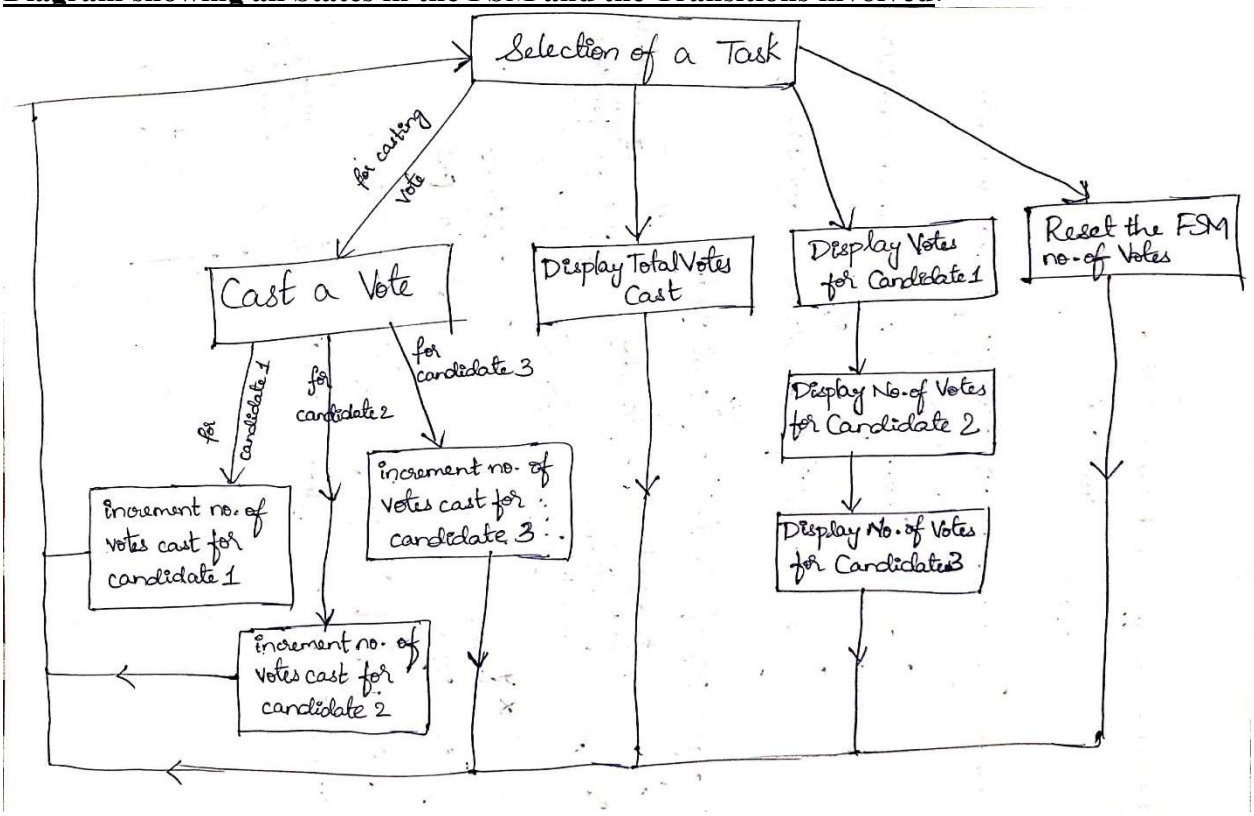
- Selection of Task :
    a. Cast a Vote
    b. Display Total No. of Votes Cast
    c. Display the No. of Votes Cast for each Candidate
    d. Reset the No. of Votes

- Cast a Vote
- Increment No. of Votes for Candidate 1
- Increment No. of Votes for Candidate 2
- Increment No. of Votes for Candidate 3
- Display Total No. of Votes Cast
- Display the No. of Votes Cast for Candidate 1
- Display the No. of Votes Cast for Candidate 2
- Display the No. of Votes Cast for Candidate 3
- Reset the No. of Votes

Hence, there should be Inputs for Selection of a Task and for Casting a Vote. The Voting Machine does not require any outputs.

Since there are ten states, the state register should be a 4-bit register.

Diagram showing all States in the FSM and the Transitions involved:



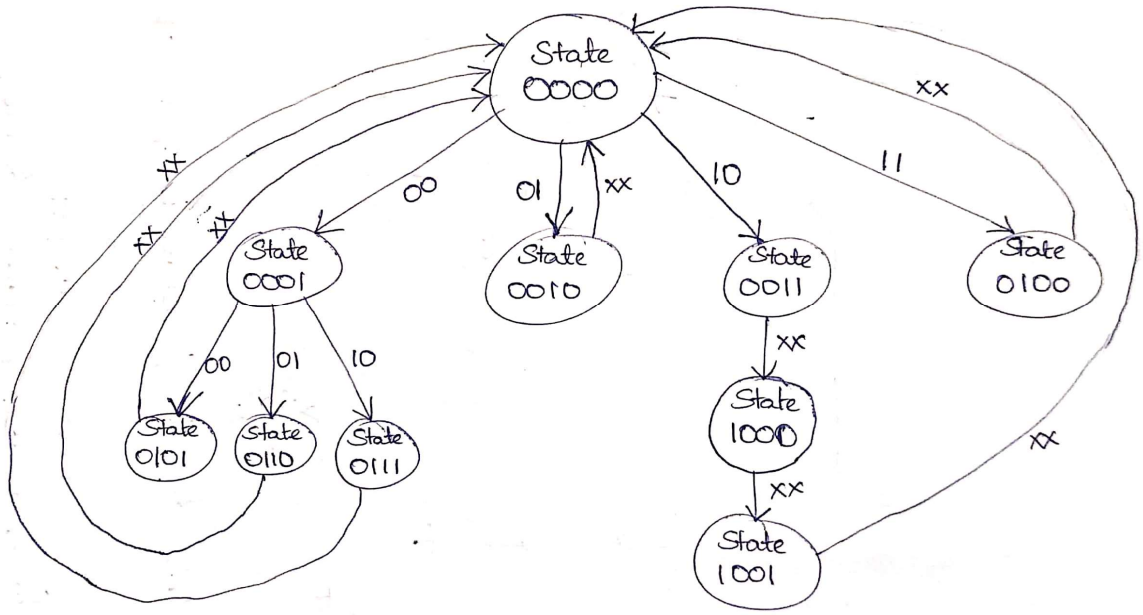
State Encoding Table

State	Encoding
Selection of Task	0000
Cast a Vote	0001
Increment No. of Votes for Candidate 1	0101
Increment No. of Votes for Candidate 2	0110
Increment No. of Votes for Candidate 3	0111
Display Total No. of Votes Cast	0010
Display the No. of Votes Cast for Candidate 1	0011
Display the No. of Votes Cast for Candidate 2	1000
Display the No. of Votes Cast for Candidate 3	1001
Reset the No. of Votes	0100

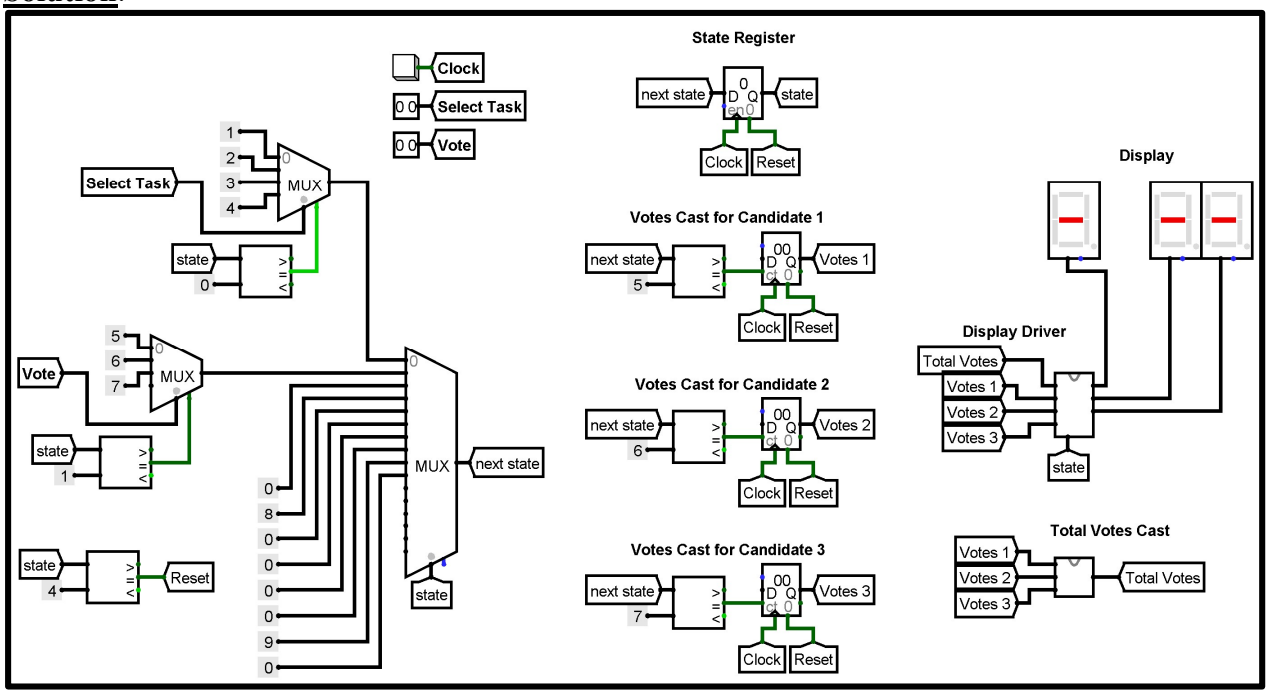
State Table

State	Input	Next State
0000	00	0001
0000	01	0010
0000	10	0011
0000	11	0100
0001	00	0101
0001	01	0110
0001	10	0111
0010	xx	0000
0011	xx	1000
0100	xx	0000
0101	xx	0000
0110	xx	0000
0111	xx	0000
1000	xx	1001
1001	xx	0000

State Diagram:



Solution:



Notations:

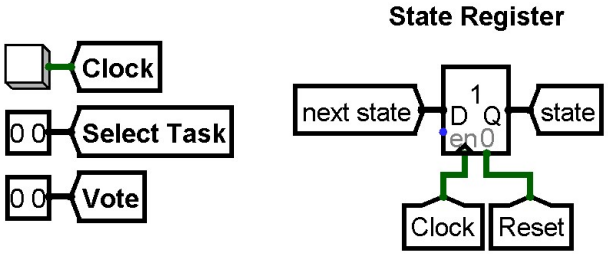
Clock	Button which acts as Clock for the FSM
Select Task	Input for Selection of Task (in State 0)
Vote	Input for Casting a Vote (in State 1)
Votes 1, Votes 2, Votes 3	No. of Votes cast for candidates 1,2,3 respectively
Total Votes	Total No. of Votes Cast
Reset	To clear memory
Display	To display **Total Votes**, **Votes 1**, **Votes 2**, **Votes 3**
State Register	Stores Current State of the FSM
state	Current State of the FSM
next state	Next State of the FSM

Simulation:

Step 1:

FSM is in State 0 as shown in the figure above.

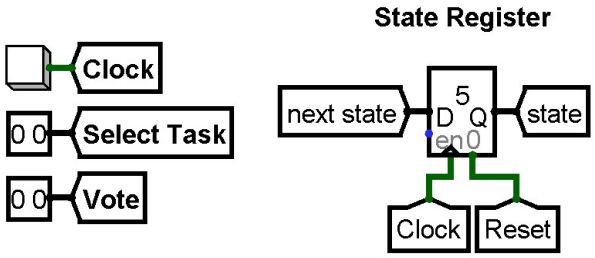
We choose to Cast a Vote. So, set **Select Task = 00**, and toggle the **Clock**.



For choosing other tasks	
Task	**Task Encoding**
Cast a Vote	00
Display Total No. of Votes Cast	01
Display Votes Cast for each Candidate	10
Reset / Clear Memory	11

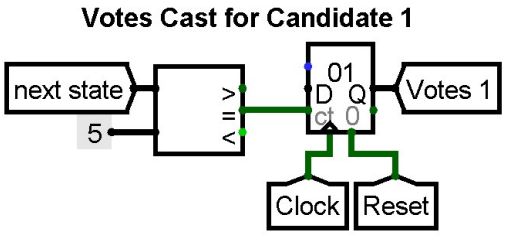
FSM goes to State 0001 (HEX 1).

Step 2:
Now, we choose to vote for Candidate 1. So, set **Vote = 00,** and toggle the **Clock**.



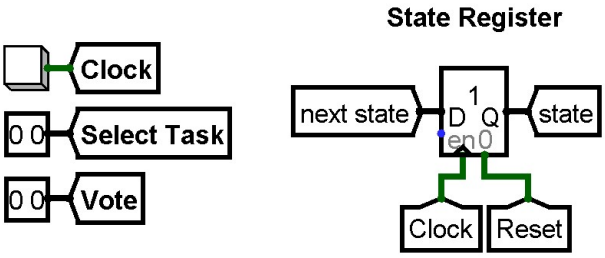
To vote another candidate	
Candidate	**Encoding**
1	00
2	01
3	10

The FSM goes to State 0101 (HEX 5). Notice that the No. of Votes cast for Candidate 1 is incremented. The no. of votes is an 8-bit number, displayed in hexadecimal system.



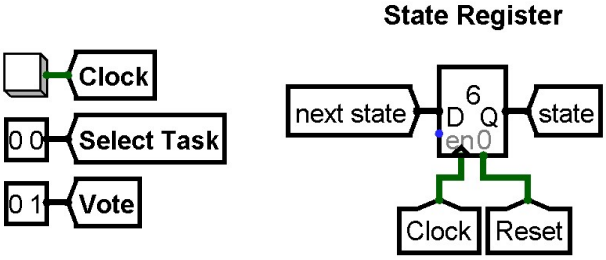
Toggle the **Clock** to go back to State 0.

Step 3:
We choose to Cast a Vote. So, set **Select Task = 00**, and toggle the **Clock**.

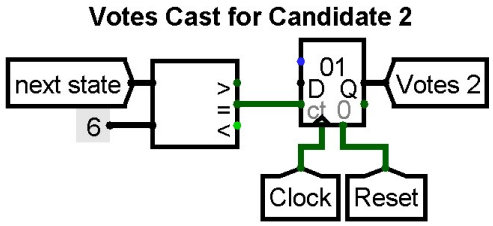


FSM goes to State 0001 (HEX 1).

Step 4:
Now, we choose to vote for Candidate 2. So, set **Vote = 01,** and toggle the **Clock**.

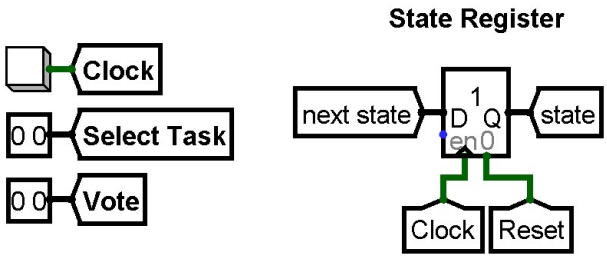


The FSM goes to State 0110 (HEX 6). Notice that the No. of Votes cast for Candidate 2 is incremented.



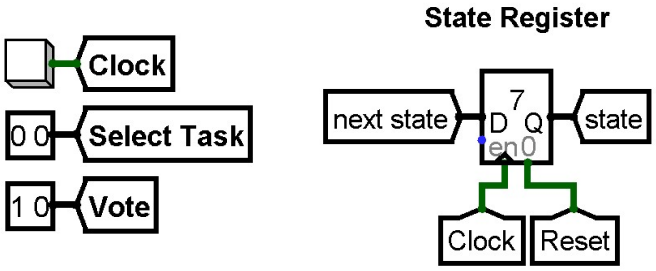
Toggle the **Clock** to go back to State 0.

Step 5:
We choose to Cast a Vote. So, set **Select Task = 00**, and toggle the **Clock**.

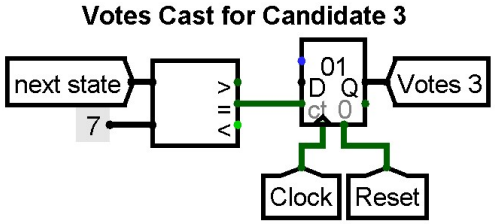


FSM goes to State 0001 (HEX 1).

Step 6:
Now, we choose to vote for Candidate 3. So, set **Vote = 10,** and toggle the **Clock**.

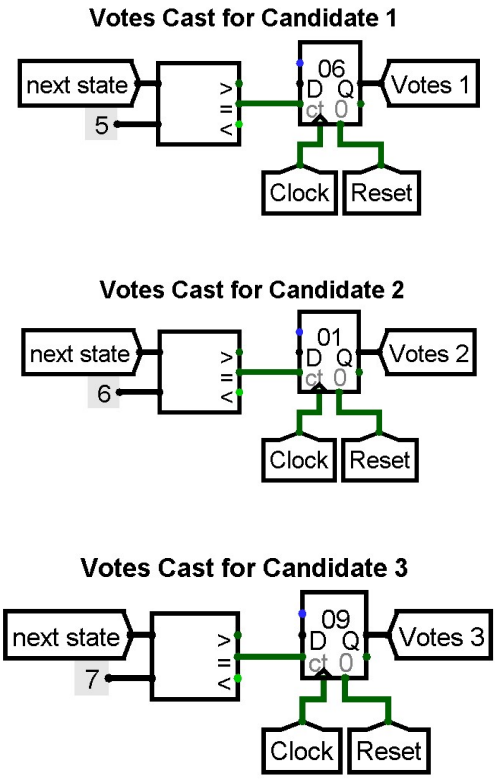


The FSM goes to State 0111 (HEX 7). Notice that the No. of Votes cast for Candidate 3 is incremented.

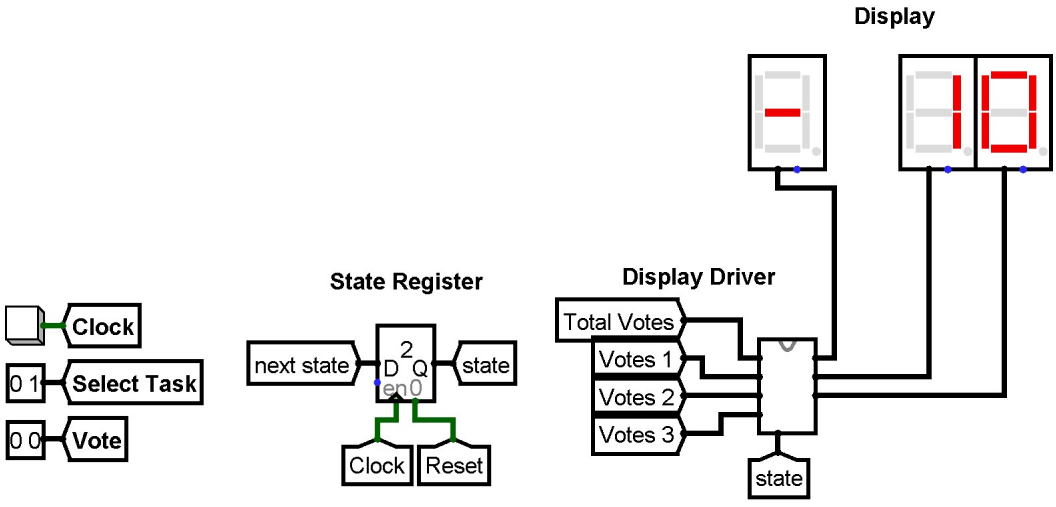


Toggle the **Clock** to go back to State 0000 (HEX 0).

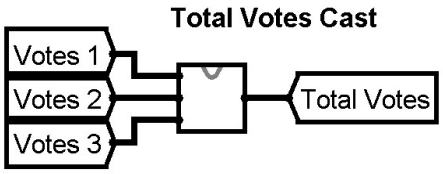
After all the voting process has been completed, assume the no. of votes each candidate got is as shown:



Step 7:
The FSM is in State 0000 (HEX 0). Now, we will try to display the Total Votes Cast. So, Set **Select Task = 01** and toggle the **Clock**.

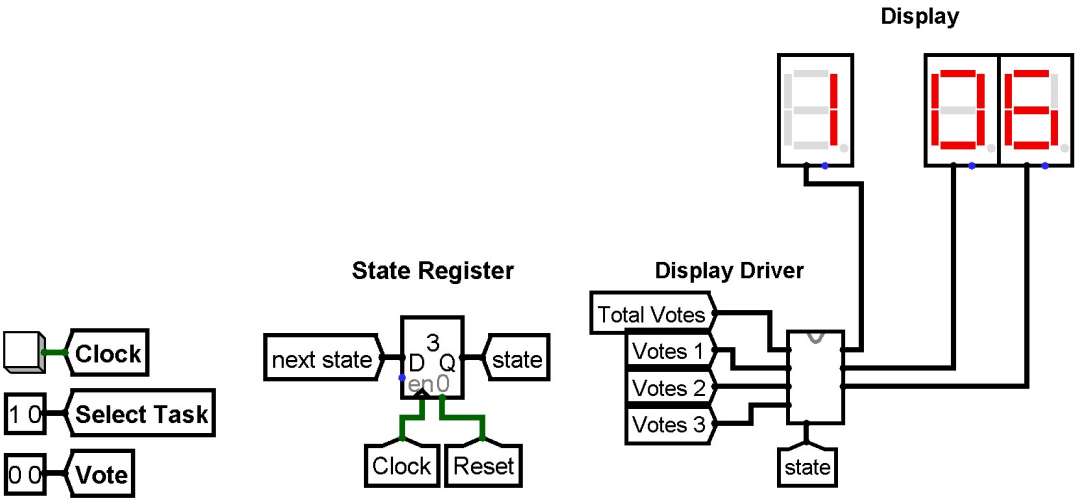


So, the FSM goes to state 0010 (HEX 2)

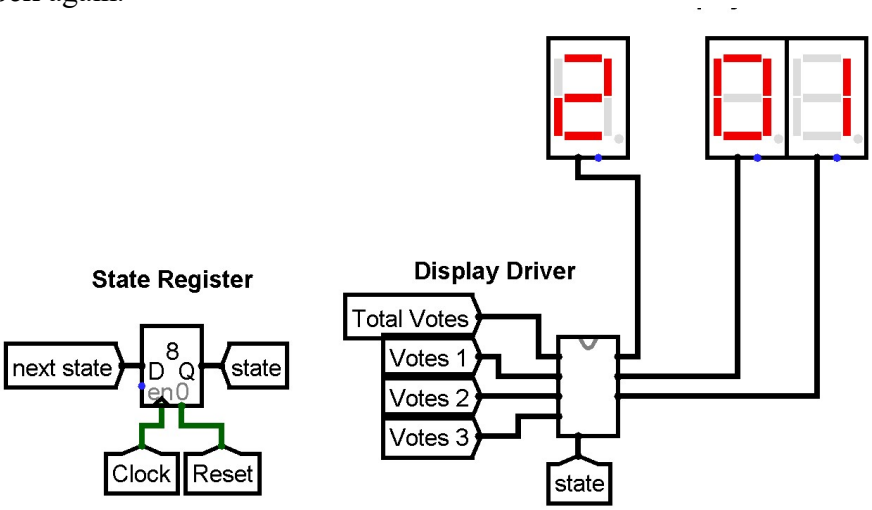


Sum of votes of Candidates 1,2,3 = HEX 6 + HEX 1 + HEX 9 = HEX 10 (16 in decimal)
The same value is being displayed as shown.
Toggle the **Clock** to go back to State 0000 (HEX 0).

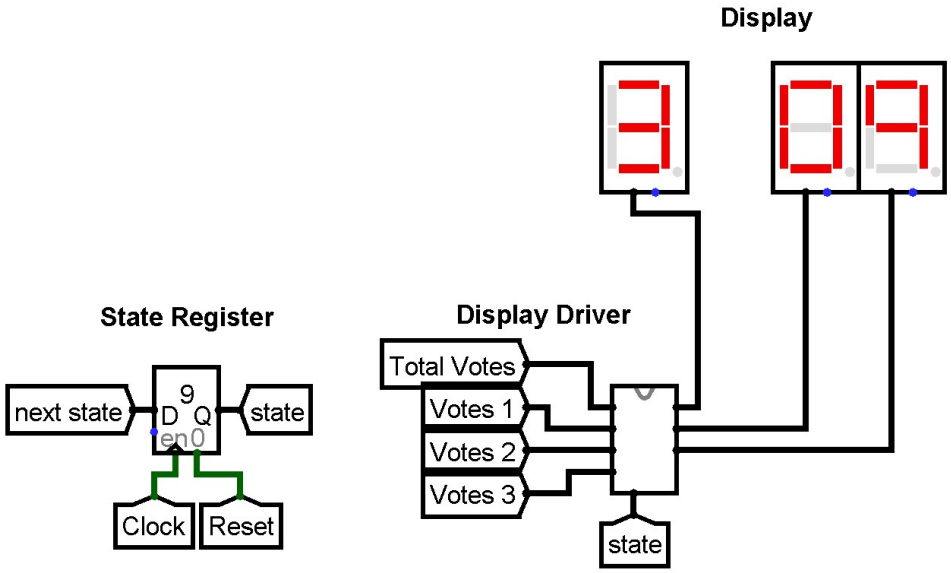
Step 8:
The FSM is in State 0000 (HEX 0). Now, we will try to display the Votes Cast for each Candidate.
So, Set **Select Task = 10** and toggle the **Clock**.



The FSM is in state 0011 (HEX 3). No. of Votes Cast for Candidate 1 (HEX 6) is displayed as shown.
Now toggle **Clock** again.

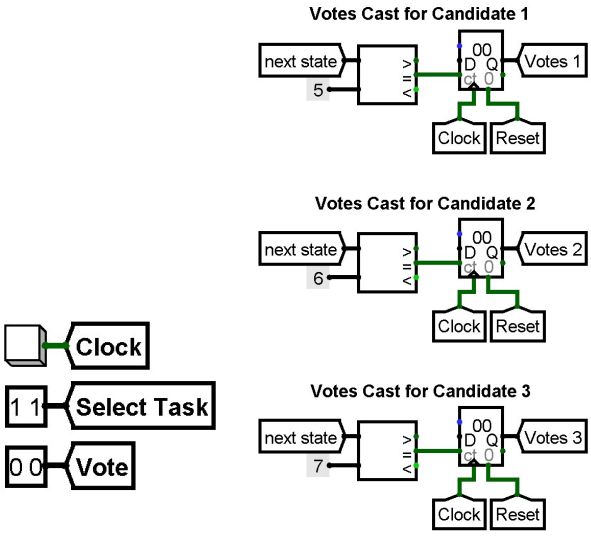


The FSM is in state 1000 (HEX 8). No. of Votes Cast for Candidate 2 (HEX 1) is displayed as shown.
Now toggle the **Clock** again.



The FSM is in state 1001 (HEX 9). No. of Votes Cast for Candidate 3 (HEX 9) is displayed as shown.
Now toggle the **Clock** again. The FSM goes back to state 0000 (HEX 0).

Step 9:
The FSM is in State 0000 (HEX 0). Now, we will Reset / Clear the memory. So, Set **Select Task = 11** and toggle the **Clock**.



Notice that the No. of Votes for all Candidates have become 0. Hence, Memory is cleared.

In this way, I have implemented a Voting Machine Controller and demonstrated all the features of the controller.

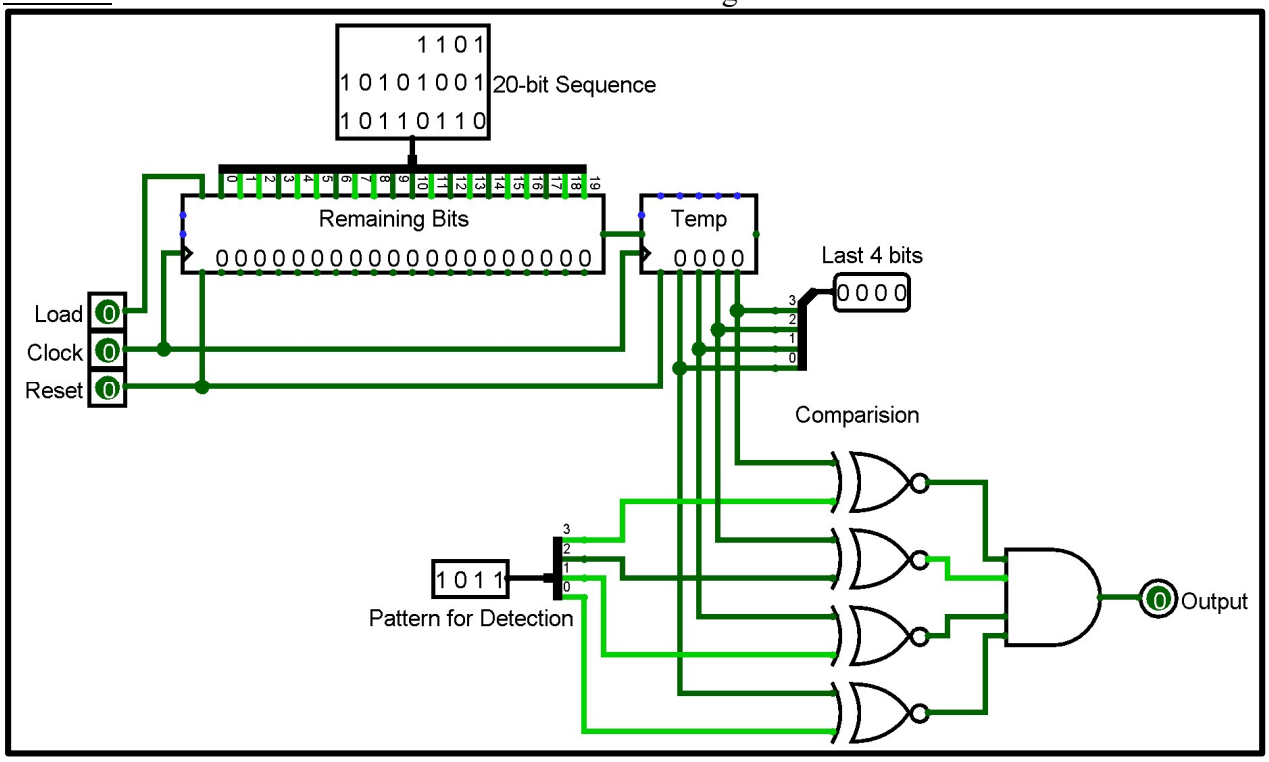
Ans 3:

Implementation of a 4-bit Pattern Detector using a Shift Register

Aim:
To implement a 4-bit Pattern Detector that detects the presence of a 4-bit pattern within a given 20-bit sequence, by using a Shift Register.

Method of Solving:
For example, **1101 1010 1001 1011 0110** is a 20-bit sequence. Our aim is to detect the presence of the pattern **1011**. For that purpose, we read the 20-bit sequence from left to right. The output is 1 when the last 4 bits that have been read, from left to right are 1101. Otherwise output is 0.

Solution: Circuit Diagram



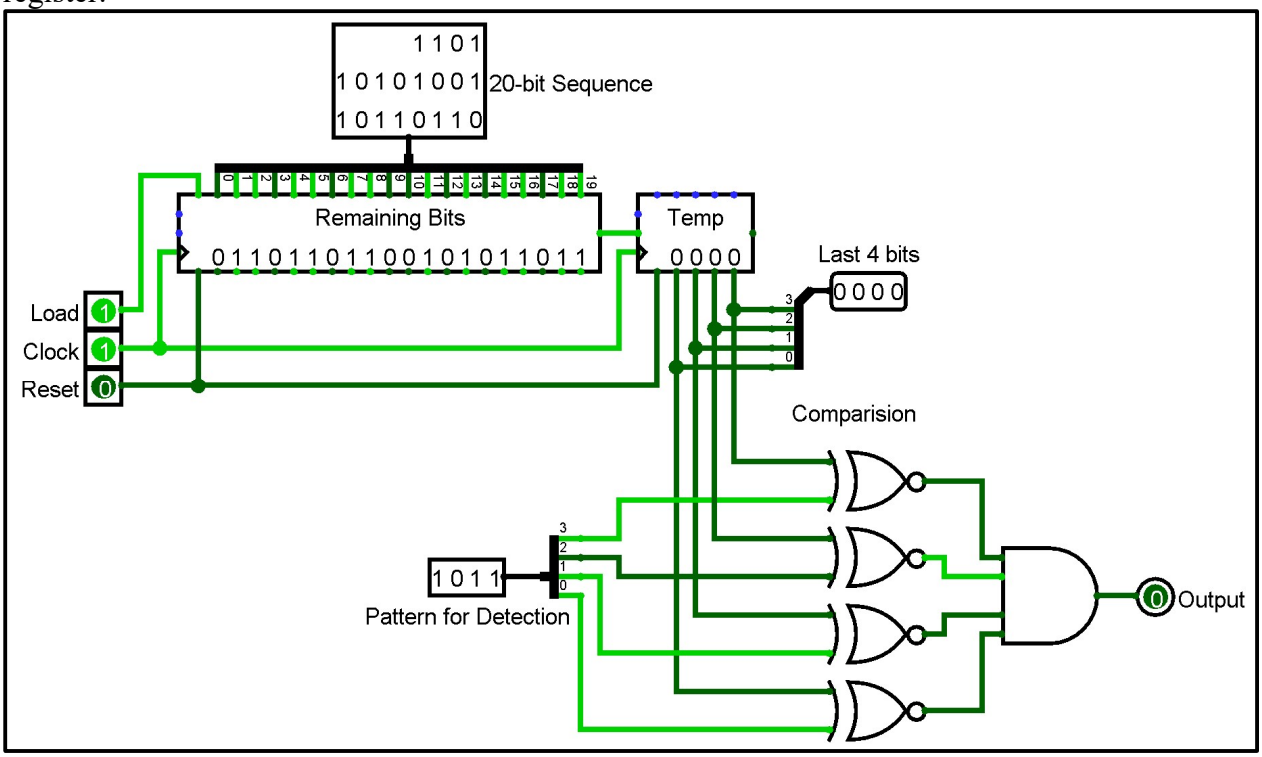
Notations:

Pattern for Detection	The pattern of bits we want to detect. Width is 4-bits.
20-bit sequence	Sequence of bits in which pattern is detected. Width is 20-bits.
Temp	Shift Register which stores the last 4 bits that have been read from 20-bit sequence.
Last 4 bits	Displays the last 4 bits that have been read from 20-bit sequence.
Remaining Bits	Shift Register which stores the bits that have to be read from 20-bit sequence.
Load	To load 20-bit sequence into shift register **Remaining Bits**
Clock	Acts as a clock for the whole circuit
Reset	Resets all bits of the shift registers to 0

Output	It is 1 when **Last 4 bits** is equal to **Pattern for Detection**
Comparison	The part of the circuit that checks whether **Last 4 bits** is equal to **Pattern for Detection**

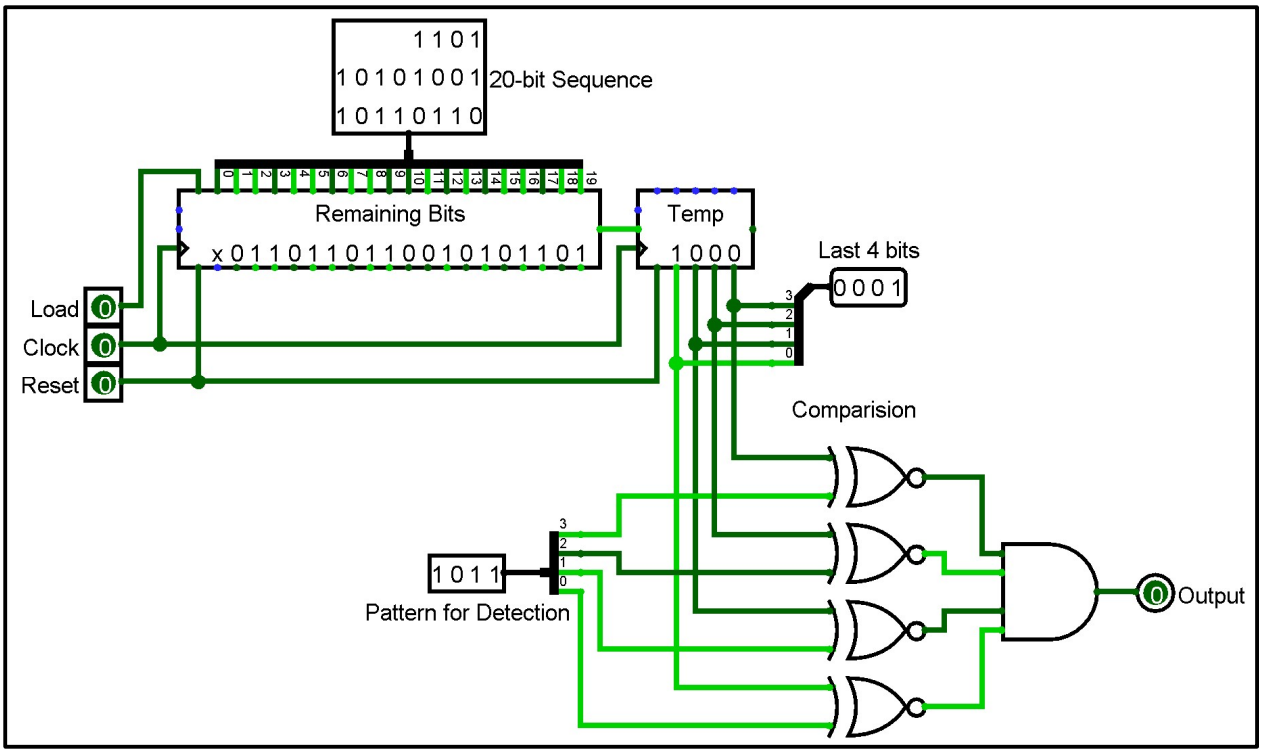
Simulation:
Step 1:
Input the **Pattern for Detection** (**1011**) and the **20-bit sequence** (**1101 1010 1001 1011 0110**) as shown above.

Step 2:
Set **Load = 1**, and toggle the **Clock**. This will load the **20-bit sequence** into the **Remaining Bits** register.



Step 3:
Now, toggle the **Clock** again (**Clock is 0** now). Set **Load = 0**.

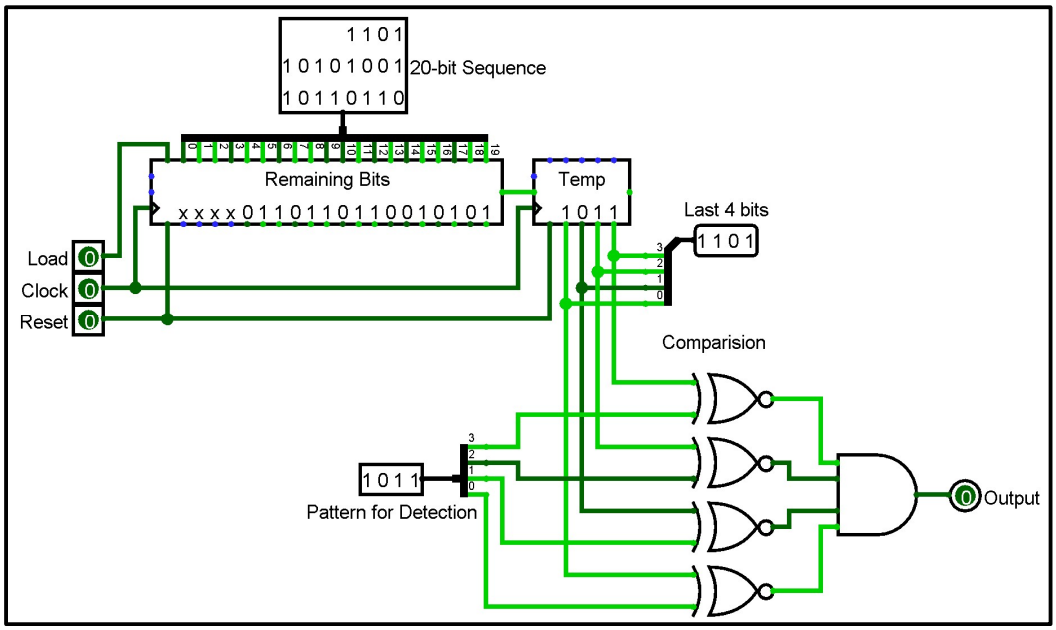
Step 4:
We start reading the bits now.
Toggle the **Clock**. **Clock is 1**.
The first bit that is loaded into the shift register **Temp** is the leftmost bit of **20-bit Sequence**.
Toggle the **Clock. Clock is 0**.



Step 5:
Toggle the **Clock. Clock is 1**.
2nd bit from the left of the **20-bit sequence** is loaded into the shift register **Temp**.
Toggle the **Clock. Clock is 0**.

Step 6:
Toggle the **Clock. Clock is 1**.
3rd bit from the left of the **20-bit sequence** is loaded into the shift register **Temp**.
Toggle the **Clock. Clock is 0**.

Step 7:
Toggle the **Clock. Clock is 1**.
4th bit from the left of the **20-bit sequence** is loaded into the shift register **Temp**.
Toggle the **Clock. Clock is 0**.



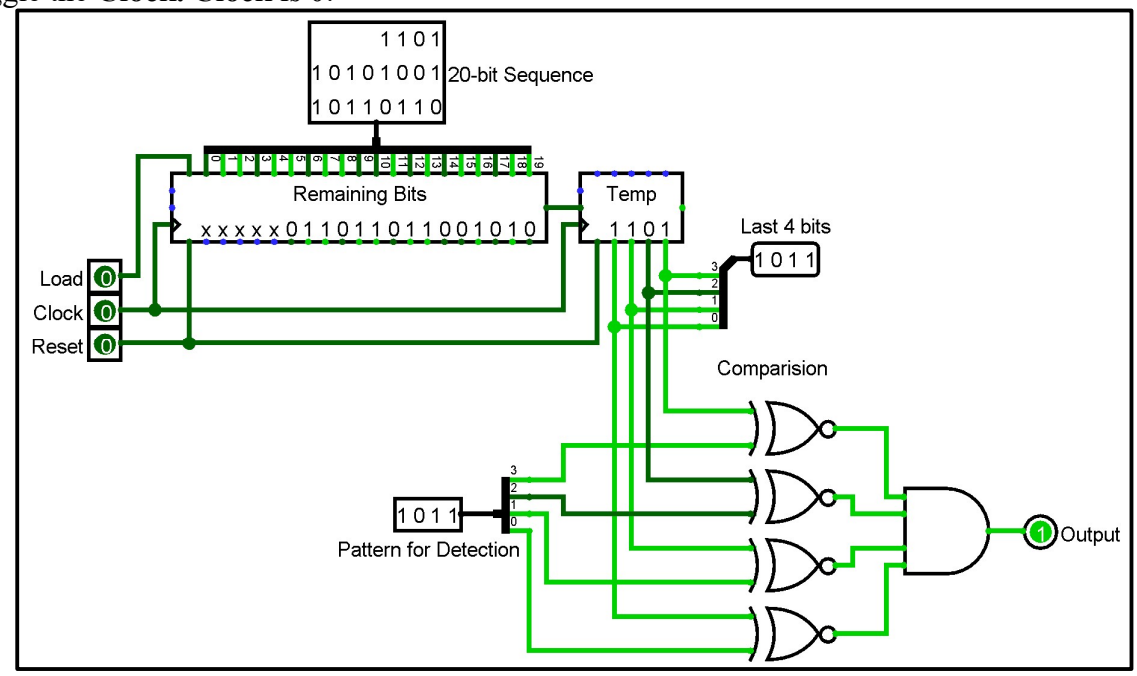
So, the last 4 bits that have been read are 1101, which clearly is not the pattern we desired. So, **output is 0**.

Step 8:

Toggle the **Clock. Clock is 1**.

5th bit from the left of the **20-bit sequence** is loaded into the shift register **Temp**. 1st bit is discarded. So, 5th,4th,3rd and 2nd bits present in the shift register **Temp**.

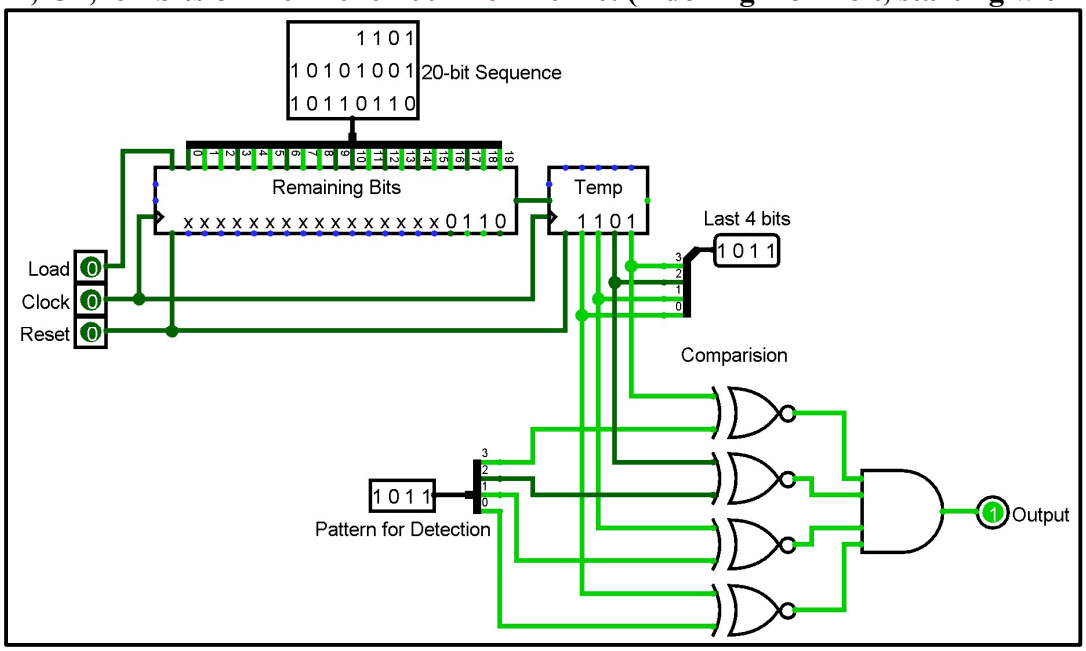
Toggle the **Clock. Clock is 0**.



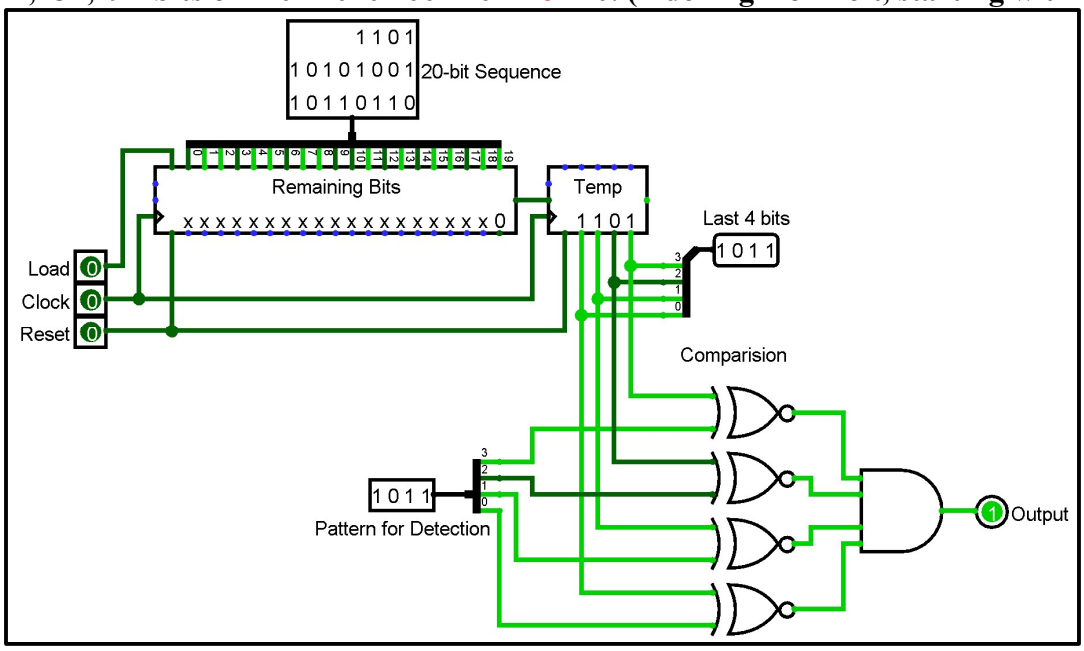
Since, the last 4 bits read are 1011 which is the same as the desired pattern 1011, the **output is 1**. The part of the circuit highlighted above is used for checking the equality between the last 4 bits read and the pattern for detection.

If we keep toggling the clock as mentioned in the above steps, we can detect the pattern at further positions of the 20-bit sequence. The screenshots below show other positions in the **20-bit sequence 1101 1010 1001 1011 0110**, where the **pattern 1011** is detected.

13th,14th,15th,16th bits of 1101 1010 1001 1011 0110. (indexing from left, starting with 1)



16th,17th,18th,19th bits of 1101 1010 1001 1011 0110. (indexing from left, starting with 1)



---------- THE END ----------