TEAM $514

Project: **SAP-1 Architecture-based 8-bit Computer Design and Implementation.**

COURSE NO. : EEE 4308 (Digital Electronics Lab)

TEAM NAME : $514

TEAM MEMBERS : 1. Md. Atiq Aziz Sadat (190021105)

2. Ahmed Jawad Rashid (190021107)

3. Nayeb Hasin (190021115)

Table of Contents

Project: SAP-1 Architecture-based 8-bit Computer Design and Implementation.

Objective:

The objective of the project is to implement an 8-bit computer based on the architecture of SAP-1 (Simple-As-Possible) using combinational and sequential circuits.

SAP-1

The SAP (Simple-as-Possible) computer has been designed for you, the beginner. The main purpose of SAP is to introduce all the crucial ideas behind computer operation. Because even a simple module such as an SAP will cover a large spectrum of very advanced concepts of Digital Electronics.

Purpose:

SAP-1 is the first stage in evolution towards modern computers. It is considered to be a big step for the beginners. (SAP)-1 computer is a very basic model of a microprocessor explained by Albert Paul Malvino. The SAP-1 design contains the basic necessities for a functional Microprocessor. Its primary purpose is to develop a basic understanding of how a microprocessor works, interacts with memory and other parts of the system like input and output. The instruction set is very limited and is simple.

Architecture:

Program Counter

It counts from 0 to F in BCD. It signals the memory address of next instruction to be fetched and executed.

Inputs and MAR (Memory Address Register)

When a program runs, the address of the RAM is latched into the MAR through the operations in the bus.

RAM

Random-access memory is a form of computer memory that can be read and changed in any order, typically used to store working data and machine code.

Instruction Register

The Instruction comes here and goes to the SAP in lower and upper nibble to be executed accordingly.

Controller-Sequencer

It generates the control signals for each block so that actions occur in desired sequence. A clock pulse is used to synchronize all the actions in an SAP. A 13-bit word comes out of the Controller-Sequencer block. This control word determines how the registers will react to the next positive CLK edge.

Accumulator
It is a 8-bit buffer register that stores intermediate results during a computer run.

It is always one of the operands of ADD, SUB and OUT instructions.

Adder/Subtractor
It is a 2's complement adder-subtractor.

This module is asynchronous (unclocked), which means that its contents can change as soon as the input words change.

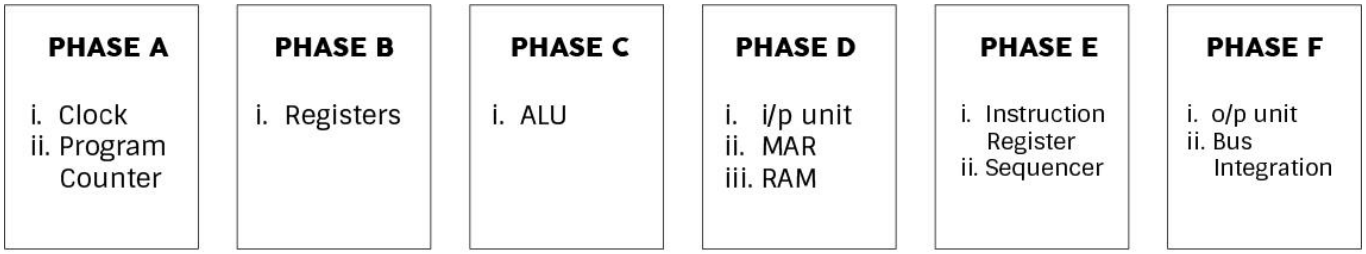
B-register
It is 8-bit buffer register which is primarily used to hold the other operand (one operand is always accumulator) of mathematical operations.

Output Register
This register holds the output of OUT instruction.

Binary Display
It is a row of eight LEDs to show the contents of output register.

Binary display unit is the output device for the SAP-1 microprocessor.

Design Phases:



PHASE A

Clock:

A clock signal is a specific sort of signal that oscillates between two states: high and low. The signal functions as a sequencer, allowing the digital circuit to coordinate its movements in real-time. Clock signals are used by digital circuits to determine when and how to perform the functions that have been programmed.

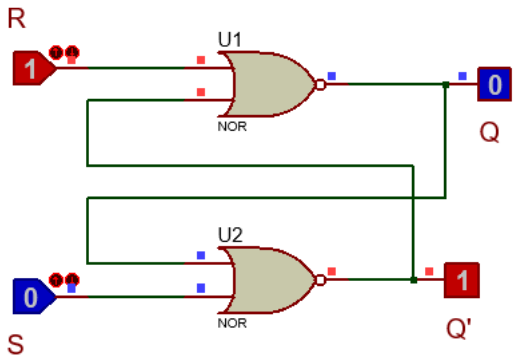
If the clock in a design is analogous to an animal's heart, then clock signals are the heartbeats that keep the system moving.

There is a lot of ways to design a clock for example using BJT/MOSFET, Op amp, Oscillators, etc. The most popular Microcontroller Arduino uses a Crystal oscillator to send its clock signal to ATMEGA 328 chip. But for our design, we are using 555 timer IC to generate clock signals. The reason for using it is this IC is very compact and very easy to use. By only changing the values of the RC network we can easily change the frequency of the clock signals.

Basic knowledge:

SR-Latch:

As suggested by the word bi in its name, a bistable multivibrator has two stable states. The first stage is usually referred to as the asset, whereas the second is referred to as reset. Set-reset, or S-R, latches are the simplest bistable devices. To make an S-R latch, we can connect two NOR gates so that the output of one feeds back into the input of the other, as seen below:



S	R	Q	Q’
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

The above circuit is a simple SR-Latch using NOR gate without enable pin. The outputs Q and not-Q should be in opposing states. Because setting both the S and R inputs to 1 results in both Q and not-Q being 0. As a result, having both S and R equal to 1 for the S-R multivibrator is referred to as an invalid or illegal state.

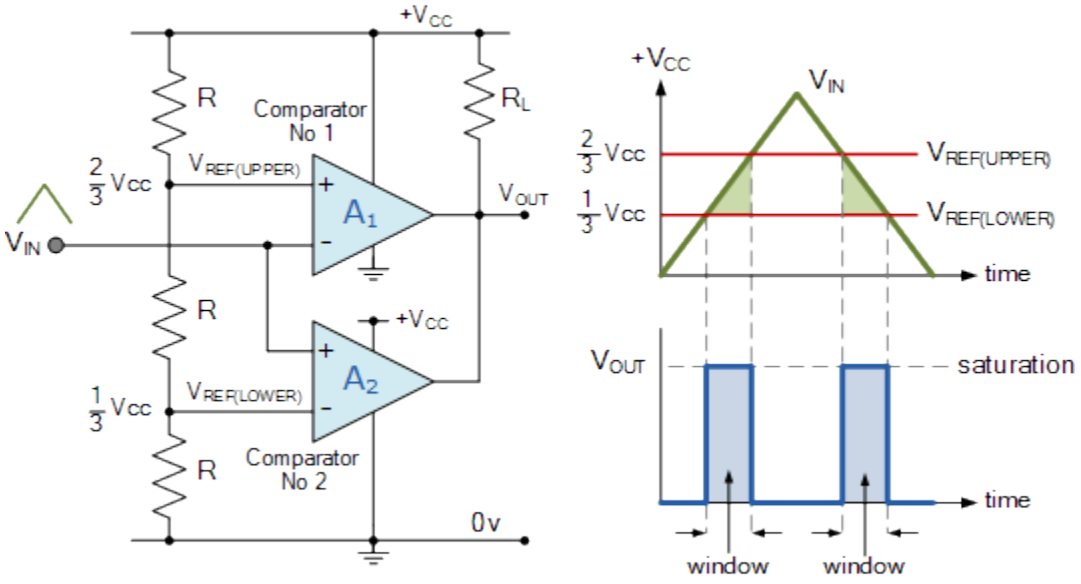
Making S=1 and R=0, on the other hand, "set" the multivibrator so that Q=1 and not-Q=0. Making R=1 and S=0, on the other hand, "reset" the multivibrator to the opposite state. When S and R are both zero, the outputs of the multivibrator "latch" in their previous states.

Comparator:

The Op-amp comparator compares two voltage levels or a predefined reference value, Reference voltage, and generates an output signal depending on the voltage comparison. To put it another way, the op-amp voltage comparator compares the magnitudes of two voltage inputs and determines which is the larger.

Because the amplifier's voltage gain is virtually equal to AVO in the open-loop mode, voltage comparators either employ positive feedback or no feedback at all (open-loop mode) to switch its output between two saturated states. On the application of a changing input signal that passes some preset threshold value, the comparator's output swings either fully to its positive input rail, +Vcc, or fully to its negative input rail, -Vcc, due to its high open-loop gain.

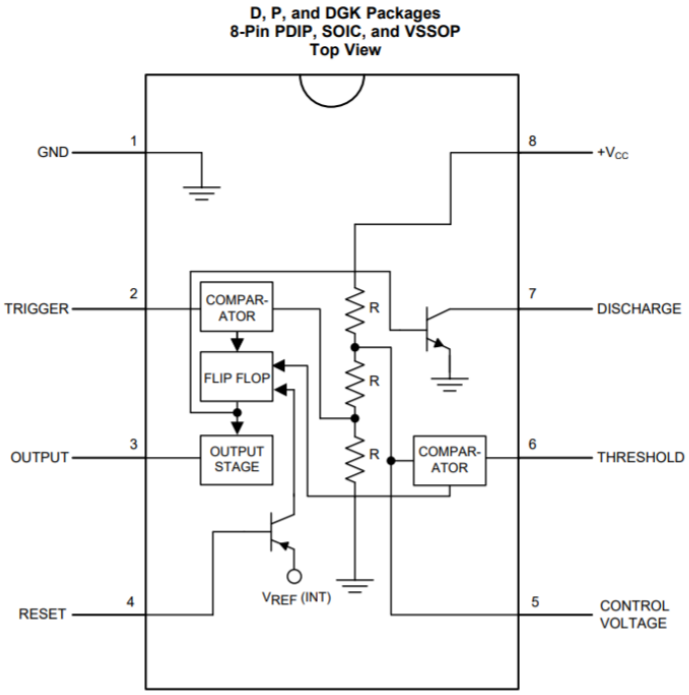
The open-loop op-amp comparator is a non-linear electronic circuit that behaves like a digital bistable device due to variations in the two analog inputs, V+ and V-, as triggering leads it to have two possible output states, +Vcc or -Vcc. The voltage comparator can then be described as a 1-bit analog to digital converter because the input signal is analog but the output is digital.



555 timer IC:

The 555 timer draws inspiration from the three internally linked 5k resistors it employs to generate the two comparator reference voltages. The 555 timer IC is a minimal, widely used precise timing device that can be used as a simple timer to generate single pulses or large time delays, or as a relaxation oscillator to generate a string of regulated waveforms with duty cycles ranging from 50 to 100%.

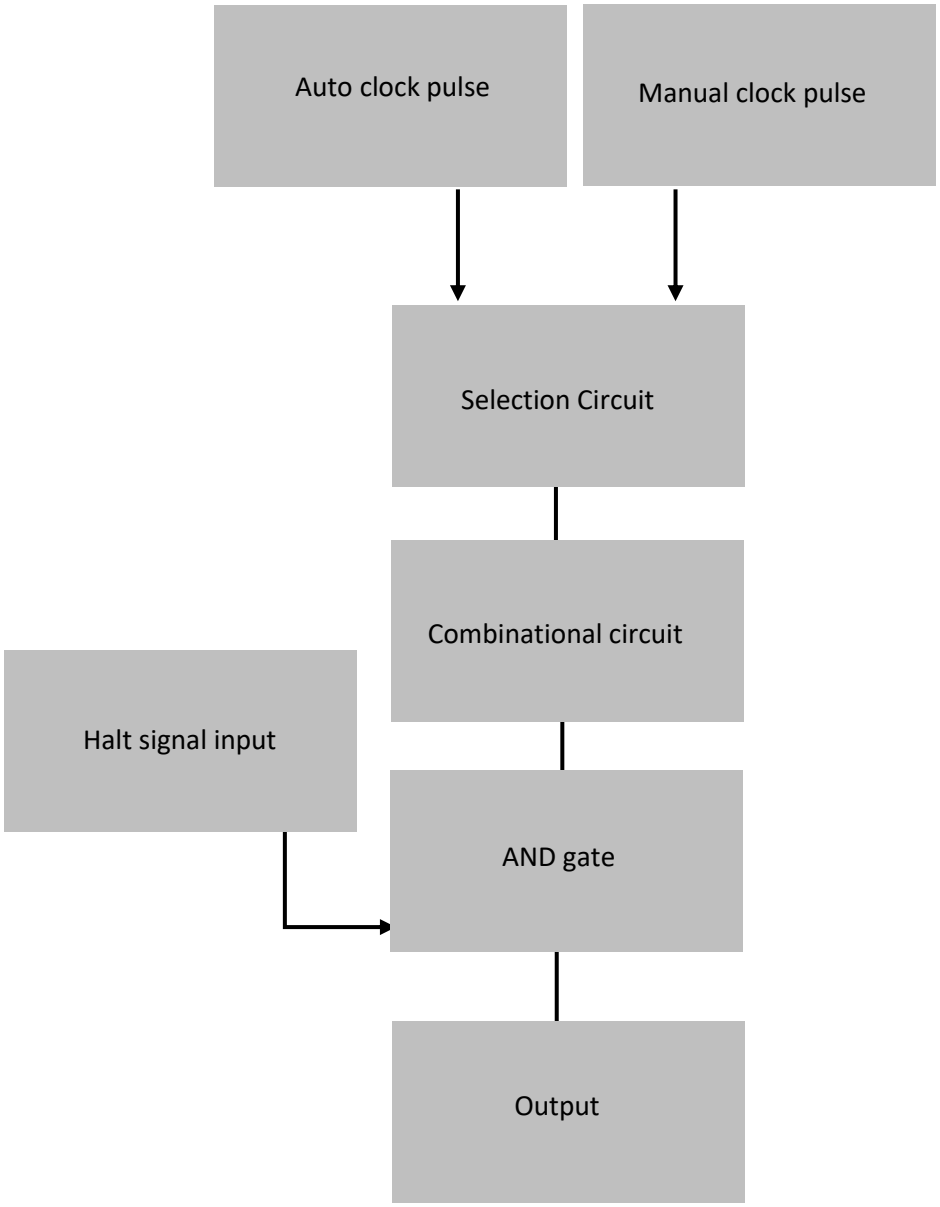
The 555 Timer chip is a Bipolar 8-pin small Dual-in-line Package (DIP) device with about 25 transistors, 2 diodes, and about 16 resistors grouped to make two comparators, a flip-flop, and a high current output stage. The NE556 Timer Oscillator, which combines 2 distinct 555s into a single 14-pin DIP package, and low-power CMOS variants of the single 555 timers, such as the 7555 and LMC555, which employ MOSFET transistors instead of 555s.



Pin name	PIN	I/O	Description
GND	1	O	Ground reference voltage
Trigger	2	I	The flip-transition flop's from set to reset is handled by this component. The amplitude of the external trigger pulse applied to this pin determines the timer's output.
Output	3	O	Output driven waveform
Reset	4	I	To disable or reset the timer, a negative pulse is supplied to this pin. It should be connected to VCC when not in use for reset reasons to avoid false triggering.
Control Voltage	5	I	Controls the trigger and threshold levels. It determines the output Voltage waveform's pulse width. This pin can also be used to modulate the output waveform with an external voltage.

Threshold	6	I	The voltage delivered to the terminal is compared to a 2/3 Vcc reference voltage. The set state of the flip-flop is determined by the amplitude of the voltage applied to this terminal.
Discharge	7	I	Between intervals, an open collector output drains a capacitor (in phase with output). When the voltage exceeds 2/3 of the supply voltage, it toggles the output from high to low.

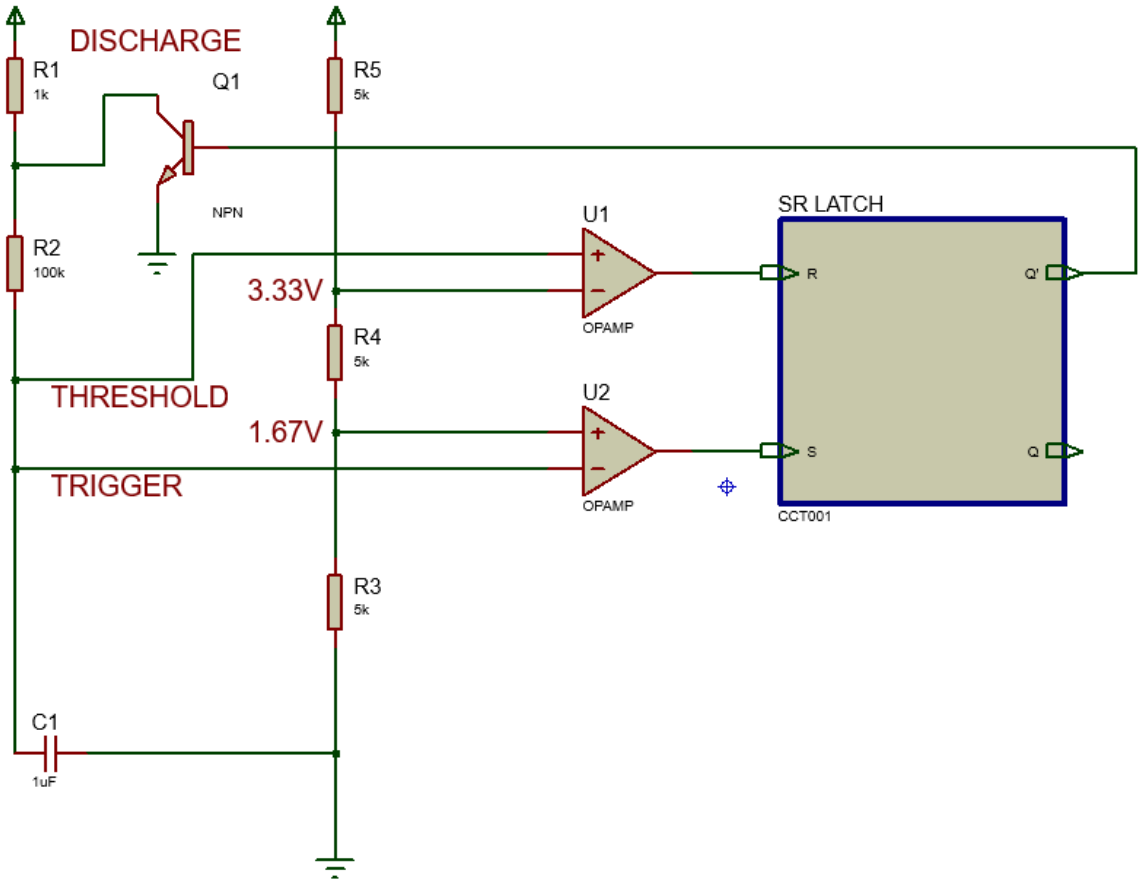
Block diagram:



Astable Clock pulse:

Free Running Multivibrator is another name for an astable multivibrator. It has no stable states and alternates between the two without any external stimulus. With the addition of three external components, the IC 555 can be turned into an astable multivibrator: two resistors and a capacitor.

Circuit diagram:



The above figure resembles the internal circuit diagram of the 555-timer circuit for Astable clock pulse generation.

Working Procedure:

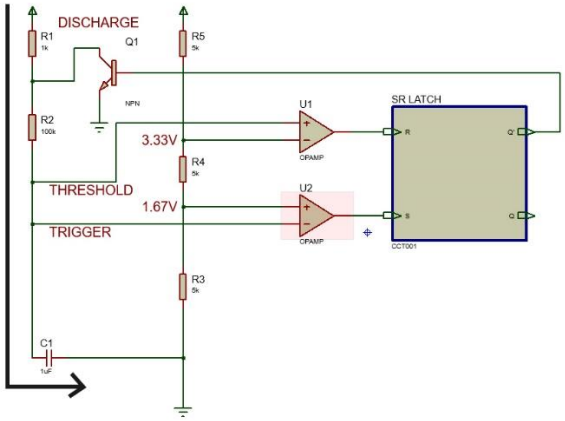
The 555-timer circuit has a built-in voltage divider network using three 5k ohm resistors. Resistor R1, R2, and C1 used in this circuit build up the RC network which is used to control the period of the timer. Here two comparators U1 and U2 are used as the input signals for the rs-latch. The comparator sets up its input from the voltage divider network and the threshold and trigger pin of the ic.

Step-01:

When the clock starts for the first time the capacitor is initially uncharged, the inverting terminal is connected with the RC network and the non-inverting terminal is connected with a 1.667v voltage divider network, as a result, the set pin of the RS-latch sets high and thus the output turns on.

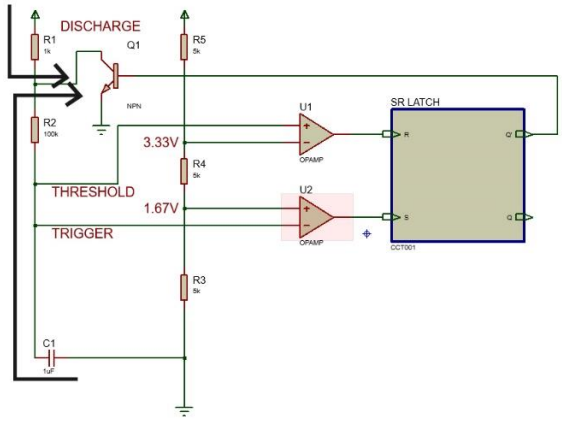
Step-02:

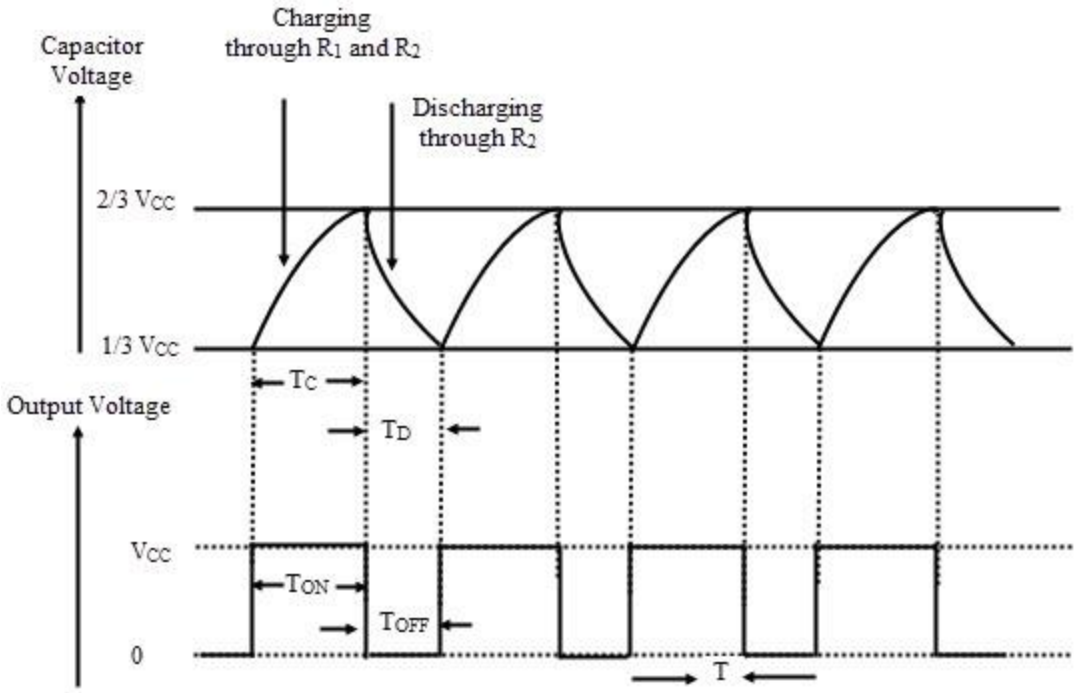
In the meanwhile, the capacitor gets charged and it supplies voltage to the inverting pin of the first Comparator. When the voltage rises above 1.667v the comparator outputs a low signal which is fed to the input of Set pin. On the other hand, the non-inverting pin of comparator U1 gets voltage higher than the 3.33v and thus it outputs a high signal to the reset pin which makes the RS-latch low (Q' high).



Step-03:

As the inverting output of the latch turns into a high state the transistor on pin 7 gets high and opens the discharge circuit. Now the voltage coming from the power source and also the capacitor gets discharged through that path. As the discharging continues the voltage drops below 1.667v the RS-latch turns on using the same manner as Step-01 and the cycle continues.

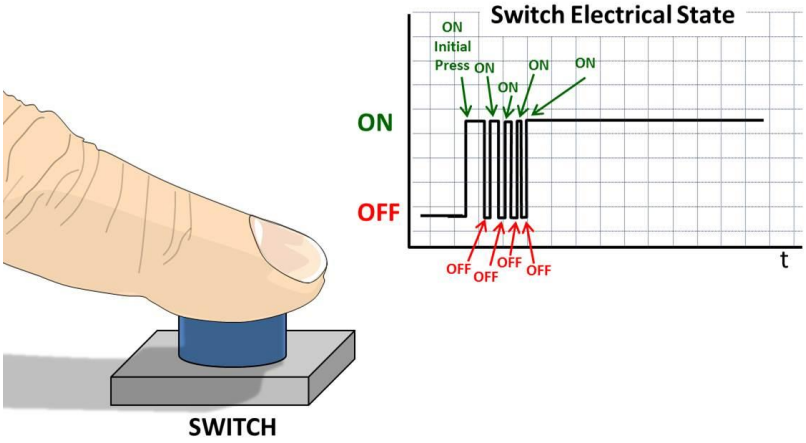




Monostable Clock pulse:

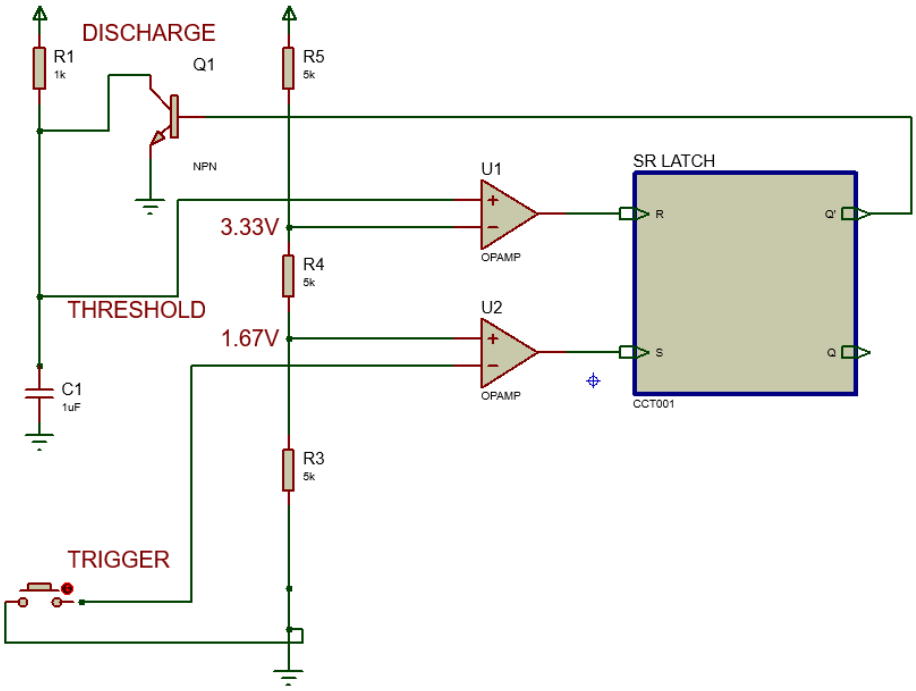
As the name specifies, a monostable multivibrator has only one stable state. When a trigger input is applied, a pulse is produced at the output and returns to the stable state after a time interval. The duration of time for which the pulse is high will depend on the timing circuit that comprises a resistor (R) and a capacitor (C). (https://www.electronicshub.org/monostable-multivibrator-using-555-timer/).

We will be using a 555 timer to avoid the switch bouncing. Two metal pieces come into contact when we press a pushbutton, toggle switch, or micro switch, shorting the supply. However, the metal plates do not attach quickly; instead, they link and disengage multiple times before forming a solid connection. When you release the button, the same thing happens. As a result, false or multiple triggering occurs, as though the button is hit many times. It's like dropping a bouncing ball from a great height, which continues to bounce on the surface until it comes to a stop.



https://microchipdeveloper.com/local--files/xpress:code-free-switch-debounce-using-tmr2-with-hlt/overview.jpg

Circuit Diagram:



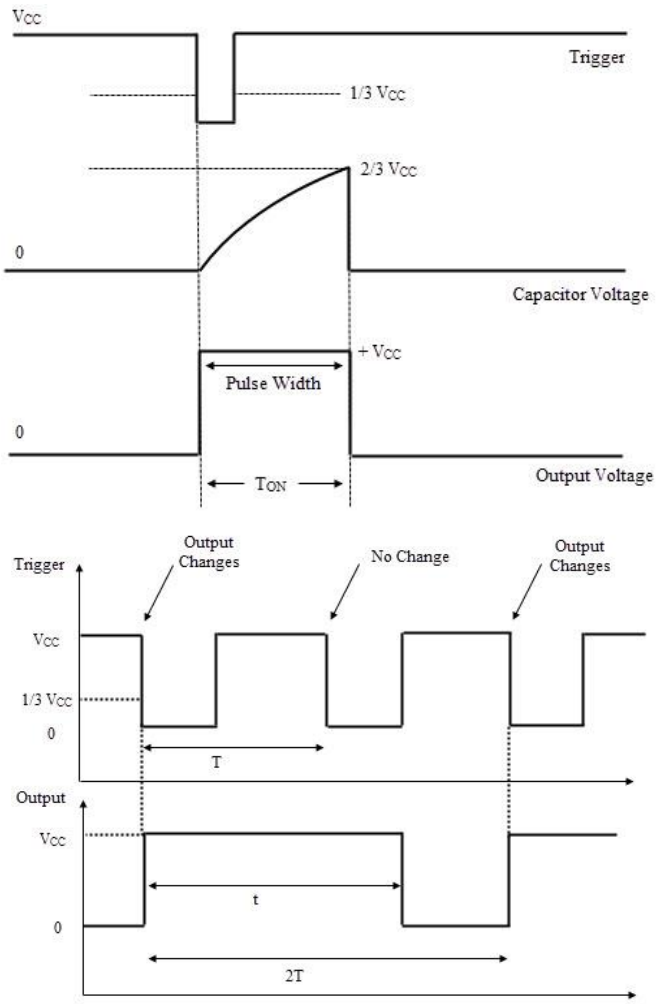
Internal Operation:

Step-01: The SR-latch was initially in the RESET position. A discharge path is provided for the capacitor which is connected to the transistor's open collector. As a result, the capacitor is entirely discharged, and the voltage across it is zero. Pin 3 has a low output (0).

Step-02: A trigger pulse input given from the push button is compared to a reference voltage of 1/3 VCC (1.667 V) when applied to trigger comparator 2. Until the trigger input surpasses the reference voltage, the output remains low. When the trigger voltage falls below 1/3 VCC (1.667 V), the comparator's output swings high, Activating the flip-flop. As a result, pin 3's output will become high.

Step-03: The discharge transistor is turned off and at the same moment the capacitor gets charged and the voltage across it increases exponentially. Thus, became more than the pin 6 threshold voltage. This is delivered to comparator 1 together with a 2/3 VCC (3.33 v) reference voltage. Until the voltage across the capacitor hits 2/3 VCC, the output at pin 3 will remain HIGH.

Step-04: The output of the first comparator turns high when the threshold voltage exceeds the reference voltage. This RESETS the SR-latch, causing the output at pin 3 to fall to logic 0), recovering the output's stable state. The discharge transistor is driven to saturation and the capacitor is discharged fully since the output is low.



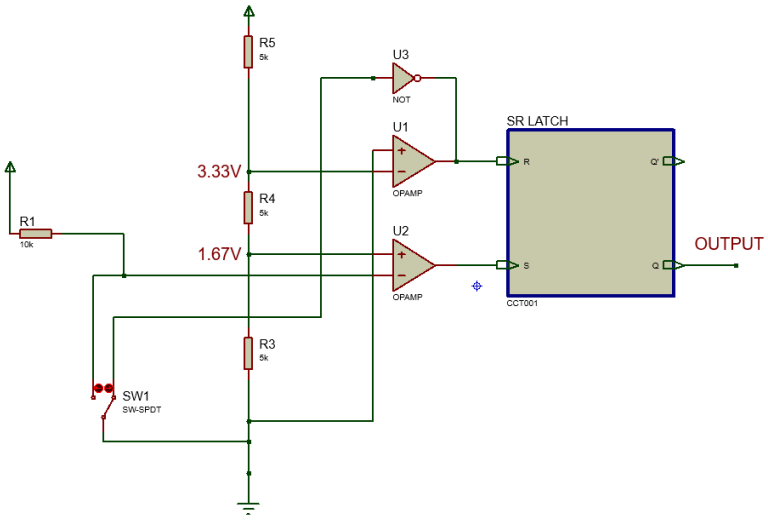
Source: https://www.electronicshub.org/

Selection circuit:

The selection circuit selects whether the user wants an astable or monostable clock pulse. This can be done using a normal push button but this will create the switch bouncing problem. To solve that problem, we will be using a 555 timer but this time we will be only using the RS-latch of it. This time we don't have to Work with the RC network of the 555 timers. The switching circuit can be imitated using a bistable clock pulse. So, we have to convert the 555 timers into a bistable multivibrator.

The term "bistable multivibrator" refers to a circuit that has two stable states (high and low). Until and unless an external trigger input is applied, it remains in the same state.

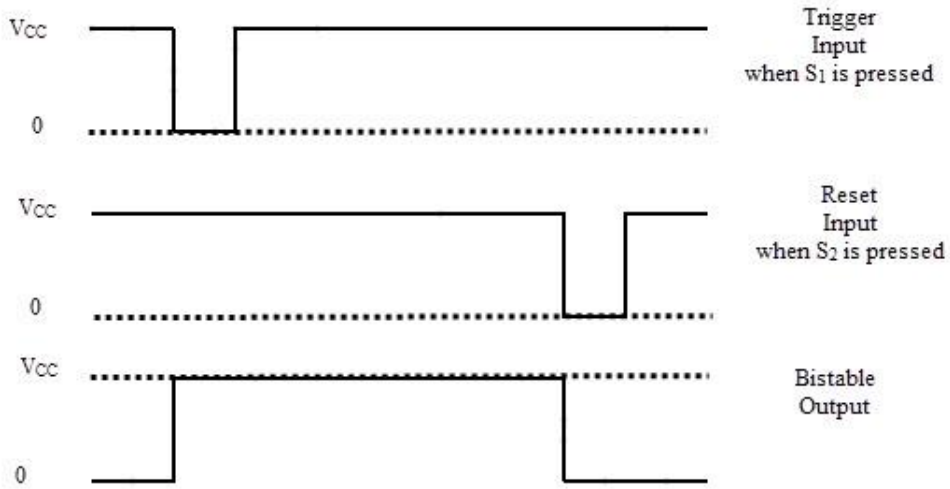
Circuit Diagram:



Internal Operations:

Set operation:
The non-inverting terminal of comparator-1 is connected with the 1.67 v of voltage divider network. So, if the inverting terminal goes below 1.67 v the comparator will switch on the set operation. There is also a pull-up resistor to supply 5 volts to the inverting pin so that the latch will be default off. An SPDT switch is used which is connected with the ground. Whenever the switch is connected with the comparator-2 the latch will be set.

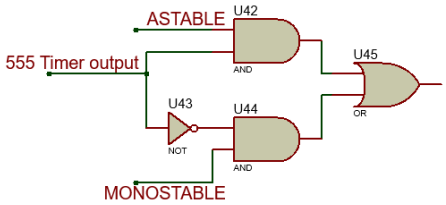
Reset operation:
We will not be using the comparator-2 for reset operation rather the reset pin of the timer. The non-inverting pin of the comparator is connected with the ground as a result the latch will always be off. The reset pin input is passed through a not gate, so whenever the switch connects with the reset pin it will make the reset pin of the latch high and thus the output will be low.



Source: https://www.electronicshub.org

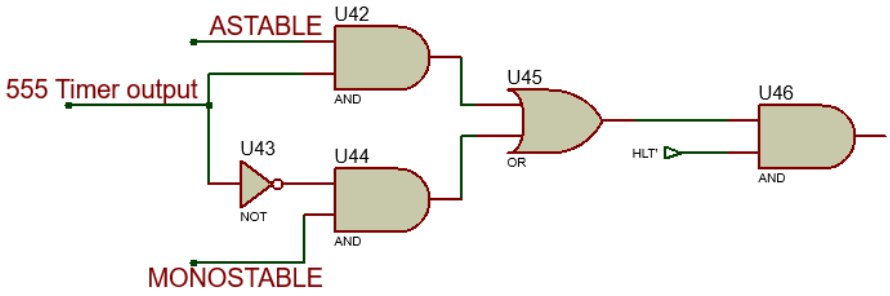
Combinational circuit:

Using the above 555 timers we were only getting high low pulses. Now we have to convert that clock pulses into a switching circuit. We will be using 2 and gates, a not and an Or gate for this operation.
The input of each of the AND gates is connected with the output of the 555 timers but the 2nd AND (U44) gate is passed through a not gate.
So, whenever we press the switch one AND gate will be high/low and the other will be inverted of it.
Thus, whenever the Astable switch is on the output will be what is there on the other pin of AND gate (U42).
Lastly, an OR gate is used to combine both the outputs and get the desired output.



Integrating halt operation:

The halt signal will be sent by the CONTROLLER SEQUENCER. Halt Operation is used to halt the master clock signal so the whole computer gets in Pause mode. We are using a simple AND gate for this task. Whenever the HLT' signal is sent by the controller it passes through a not gate and goes to one of the inputs of AND gate.

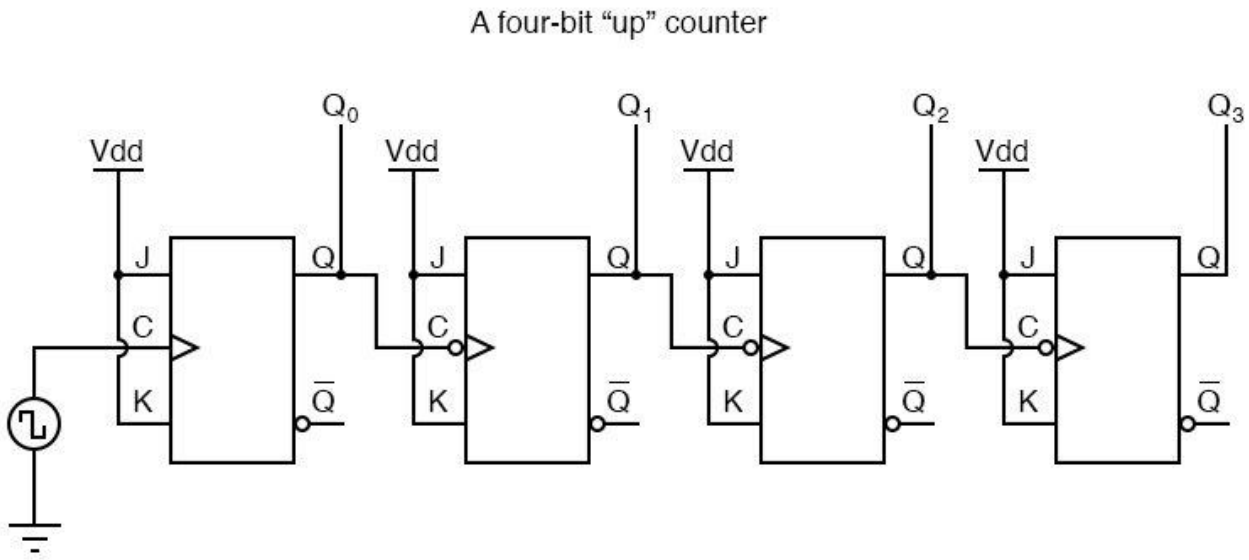


Program Counter

A program counter is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by 1. After each instruction is fetched, the program counter points to the next instruction in the sequence. When the computer restarts or is reset, the program counter normally reverts to 0.

Source: https://whatis.techtarget.com/definition/program-counter#:~:text=A%20program%20counter%20is%20a,its%20stored%20value%20by%201.&text=A%20register%20is%20one%20of,places%20that%20the%20processor%20uses.

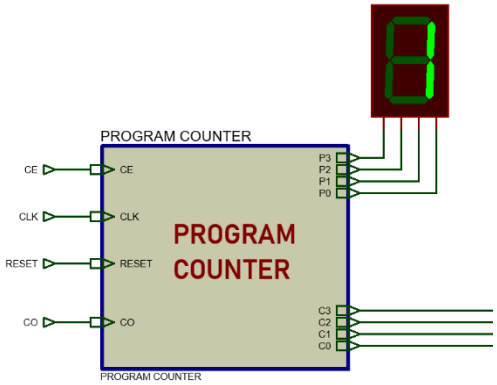
Block Diagram



Source: https://www.allaboutcircuits.com/textbook/digital/chpt-11/asynchronouscounters/?fbclid=IwAR14WeMoOkQLTmvGUyqbfXwl30RC7hgRwpjpt1FTJx4o8FD-1uRPFMcDrks

Specifications

ICs used	74LS107
Input pins	CE, CO, CLK, RESET
Output pin (Buffer)	C3(MSB)-C0(LSB)
Output pin (Display)	P3(MSB) -P0(LSB)

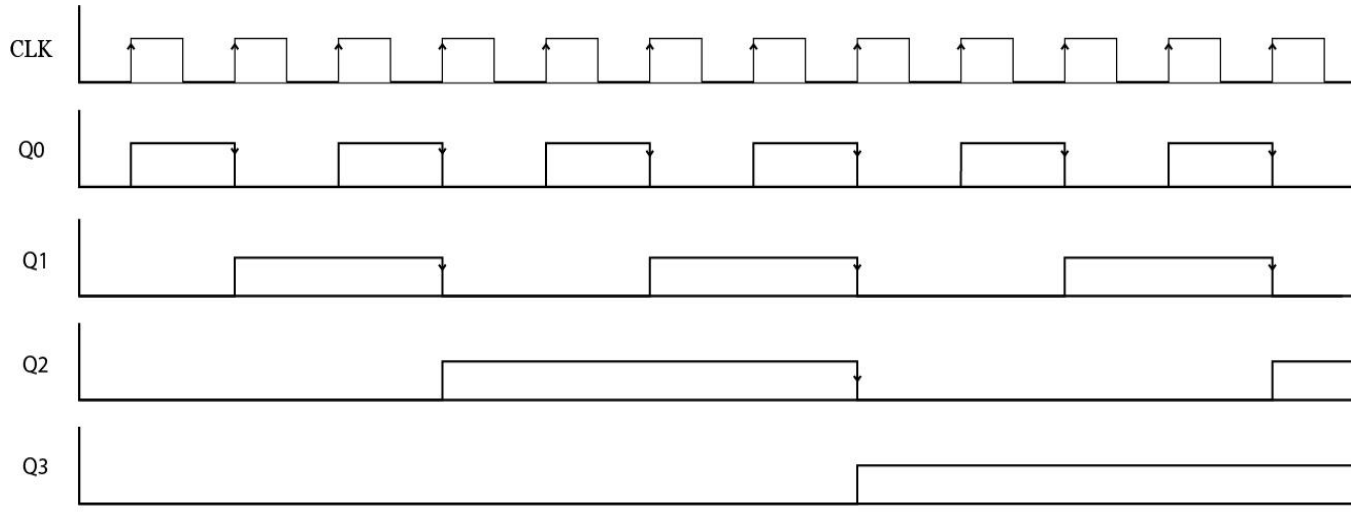


Algorithm

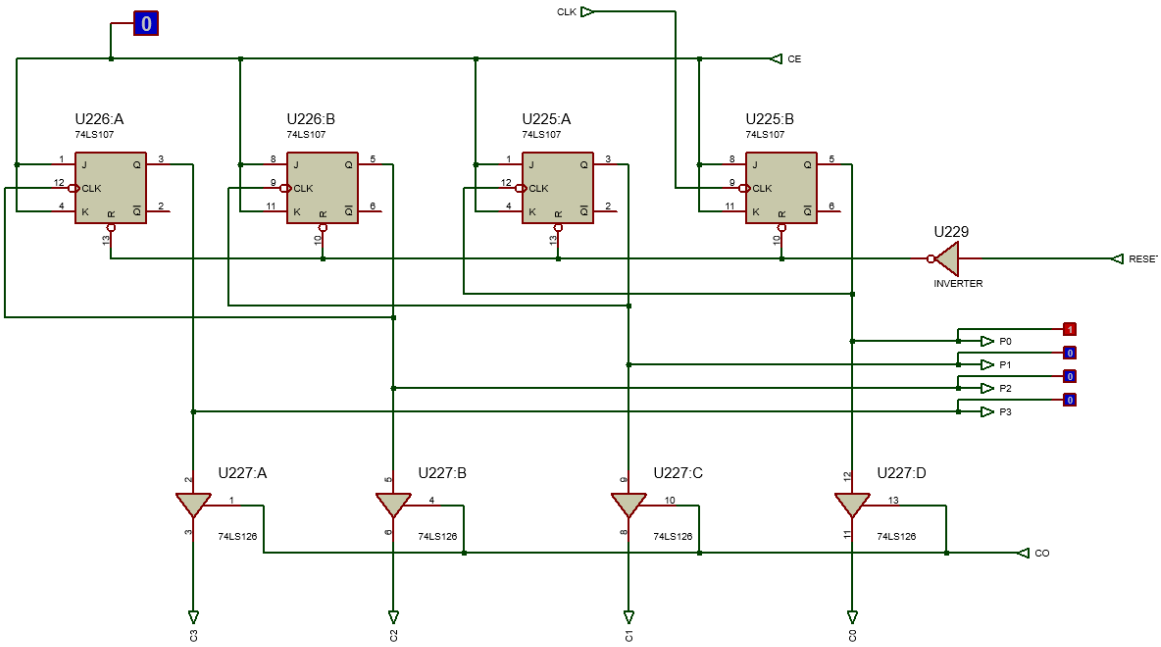
1. The rightmost flip flop has a positive edge triggered clock input, so it toggles with each rising edge of the clock signal.
2. Then we feed the output of this flip flop to the clock input of the next flip flop which is an ACTIVE LOW input.
3. Similarly, we connect the Q outputs to the clock input of the next flip flop
4. And then we short all the J and K input pins and connect it to CE input pins
5. Then the output of these flip flops is the output of the program counter
6. Then we add tri-state buffer to each of these output pins. And then the control pins of the buffer are the CO of the program counter.

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
1010
1011
1100
1101
1110
1111

Timing Diagram of Program Counter



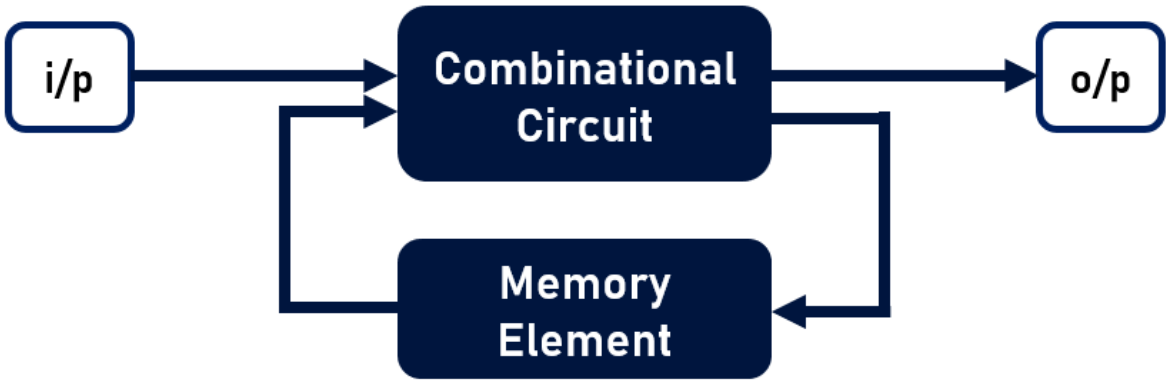
Proteus Implementation



PHASE B

Sequential Logic Circuits

Unlike combinational circuits, in sequential circuits outputs vary based on input. Here is a general block diagram designed for sequential circuits,



Latch

A latch is an electronic logic circuit that has two inputs and one output. One of the inputs is called the SET input; the other is called the RESET input.

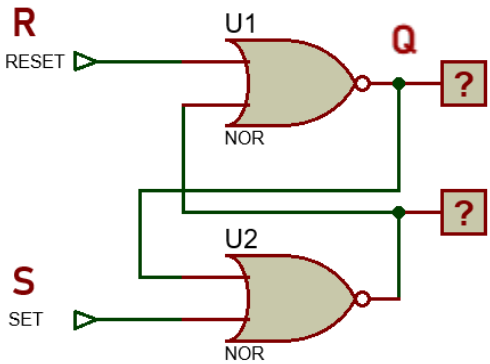
In Digital Electronics, a latch is a kind of circuit that has "Set" and "Reset" mode and we get a corresponding o/p based on whether it is a set or reset. Based on trigger of input units, circuits can be either ACTIVE HIGH or ACTIVE LOW. Generally, in Active high mode, Set refers to logic state 1 and reset refers to Logic state 0. But when the circuit is active low, with logic state 0, we get the set mode and with logic state 1 we will get reset mode.

Q is our output for the latch. And $\bar{Q}$ or Q bar is the complement of the output Q.

The basic S-R (Set-Reset) Latch

Active High Latch (w/ NOR Gate)

S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	*invalid*	

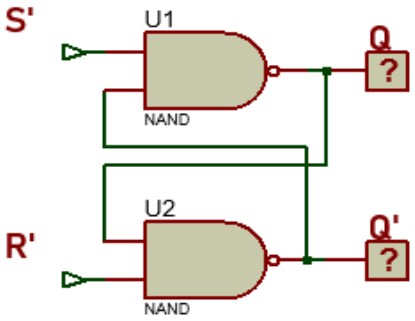


Here, S=1 means SET and R=1 means RESET.

And when S=0 and R=0, it will hold the previous memory. And we will get unpredictable results for when S=1 and R=1. So we label it's output as invalid.

Active Low Latch (w/ NAND Gate)

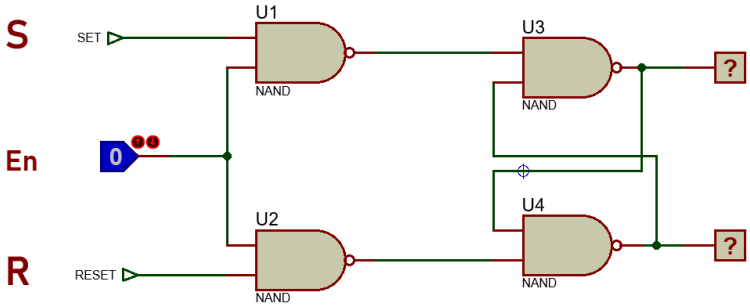
S'	R'	Q	Q'
0	1	1	0
1	1	1	0
1	0	0	1
1	1	0	1
0	0	*invalid*	



Here, S=0 means SET and R=0 means RESET.

And when S=1 and R=1, it will hold the previous memory. And we will get unpredictable results for when S=0 and R=0. So, we label its output as invalid.

Adding a Control Input to the S-R Latch

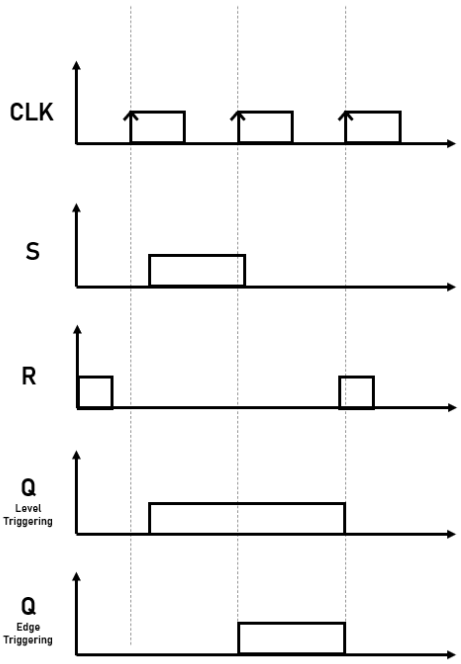


Here we simply added another level in our previous latch. And the new input pin is the enable pin here. Which when turns 0, disables the new values for S and R pin. It only holds memory during that time. But when En=1, the circuit functions like the S-R latch.

Clock Pulse

Clock pulses are continuous, precisely spaced changes in voltage. And we use them in two ways.

1. Level Triggering Clock Pulse
2. Edge Triggering Clock Pulse

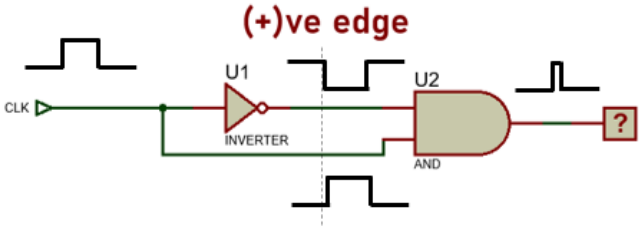


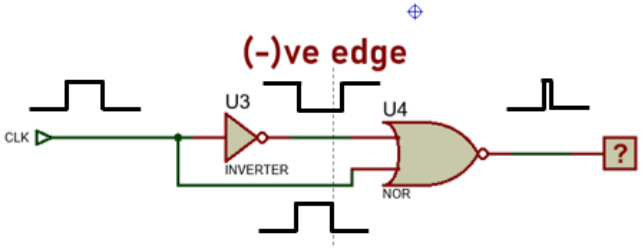
In edge triggering, we get outputs at the rising edge or falling edge.
In level triggering, we get outputs during the specific voltage state of the clock.

Added to that, it is important to mention that we are considering both the positive and negative edges of a clock pulse.

Pulse Transition Detector

This block will be able to find the rising and falling edges of a clock pulse. It is used in the edge triggering clock pulse modules. We require the following blocks for positive and negative edges respectively,

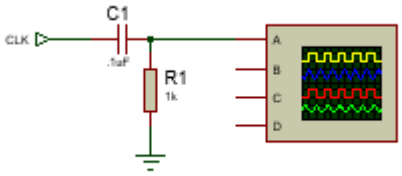




Here we had taken advantage of the delay through the NOT gate to get a specific trigger at the rising or falling edges.

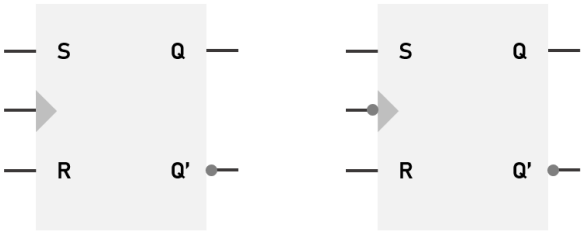
Alternative

We might also use a capacitor to get a trigger. Here at first when the pulse is turned on and it gets high, then the output current will flow in full. But at that time, the capacitor will start charging across the resistor, and as it does that we will get lesser current across the capacitor. Thus that will give us a spike or an edge. And we can control the transient time with $\zeta = RC$.



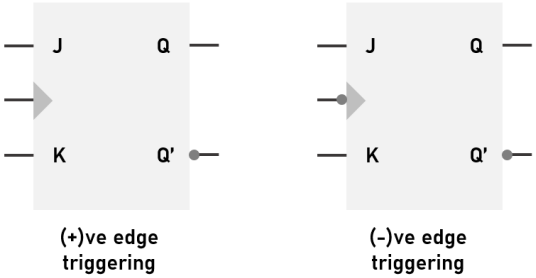
S-R Flip Flop

CP	S	R	Q
0	x	x	Memory
↑	0	0	Memory
↑	0	1	Reset
↑	1	0	SET
↑	1	1	*invalid*



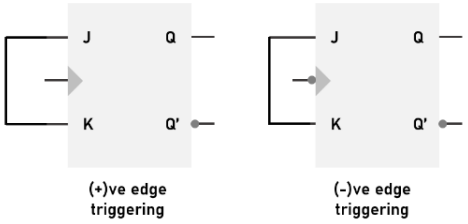
JK Flip Flop

CP	S	R	Q
0	X	X	Memory
↑	0	0	Memory
↑	0	1	Reset
↑	1	0	Set
↑	1	1	Toggle



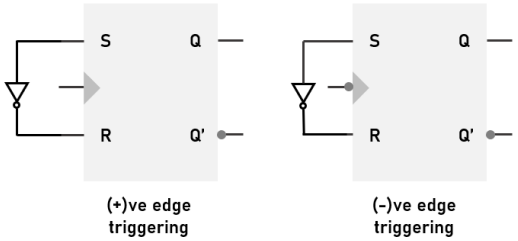
D Flip Flop

CP	D	Q
0	X	Memory
↑	0	0
↑	1	1



T Flip Flop

CP	T	Q
0	X	Memory
↑	0	Memory
↑	1	Toggle



Preset & Clear inputs for Flip Flop

S & R and J & K inputs are called synchronous inputs. They are synchronous to the clock pulse. But the Preset and Clear inputs are not dependent on the clock pulse. Rather when preset is activated then the Flip Flop goes in SET mode and when Clean is activated then the Flip Flop goes in RESET mode. And it is to be noted that these two pins are active low pins.

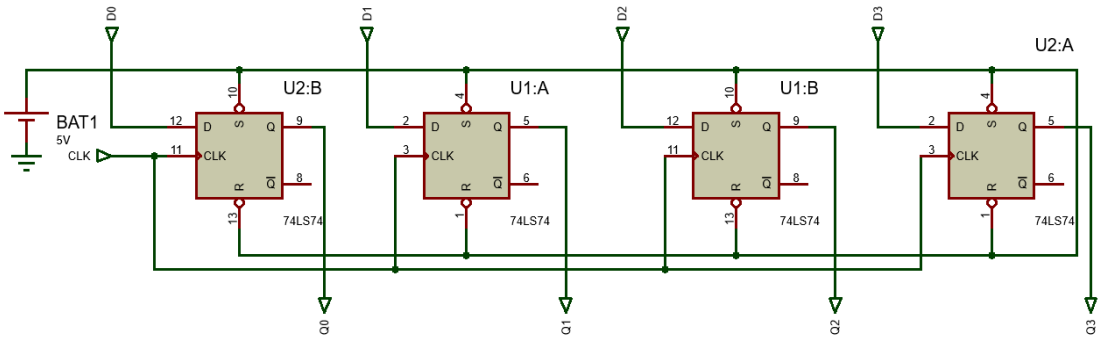
Register and bus architecture

Register

A **register** is a collection of flip flops. A flip flop is used to store single bit digital data. For storing a large number of bits, the storage capacity is increased by grouping more than one flip flops.

The register is used to perform different types of operations. For performing the operations, the CPU use these registers. The faded inputs to the system will store into the registers. The result returned by the system will store in the registers. There are the following operations which are performed by the registers.

A D Flip Flop is an example of a 1-bit data storage device. So, a single D flip flop can act like a one-bit register. Thus, if we want to store an n-bit word, we have to use an n-bit register containing n number of flip flops. There are various categories of registers. In the SAP-01 project, only the “Parallel In-Parallel Out” registers have been used.

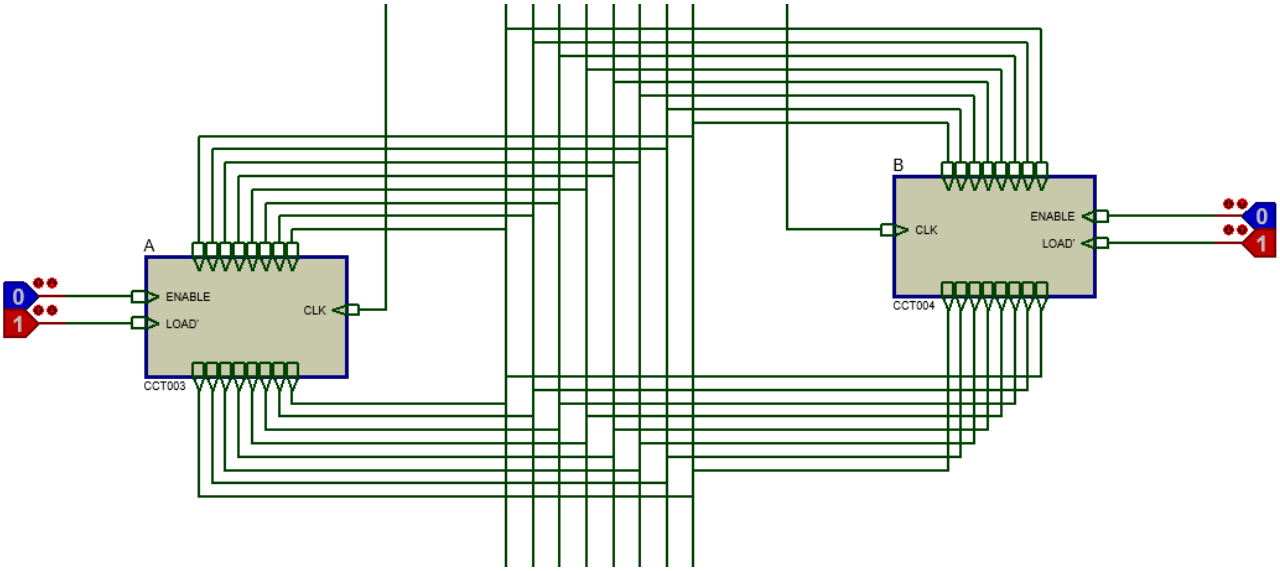


The above figure is a diagram of a 4-bit parallel in-parallel out register designed using 4 D Flip Flops. Here each of the flip flops are connected to separate D inputs and four separate outputs get out from the flip flops. Thus, as the inputs and outputs are inserted and found at the exact same time, it is called parallel in parallel out register. And in each of these blocks the same clock pulse is fed. And it is either positive or negative edge triggered, depending on the design and need of the register. And the values from these registers can come or go from or into the bus on each edge trigger of the clock input. And here in this

particular case data will go from D to Q on each positive edge trigger of the clock. And if we want to create an infrastructure which sends or receives some sort of data of particular bits, then we can make up our bus based on our need.

Bus Architecture

Bus is an abbreviation of *omnibus*. In the English bus is the abbreviation of "for all" and here in the modern use, the "omnis" part is not in use anymore and the "bus" part is kept which originally referred to "for". Nowadays in computer science a bus is usually a group of signal lines connecting multiple devices.

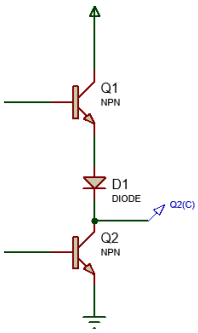


Here the above connection is a demonstration of the 8-bit bus. Here, each of the output and input connections of the modules are connected to the bus. But whether or not a module would input a value depends on the LOAD' pin. And usually for the SAP project the load pins of the registers are active low. So when the input of the LOAD' goes from 1 to 0, then the module will take inputs from the bus. And it will only give output when the ENABLE pin is activated. And usually ENABLE pins are active high. And if we try to send two data to the same bus at the same time, it would be Bus contention. In the proteus simulation we will get errors referring to logic contentions.

Bus Contention

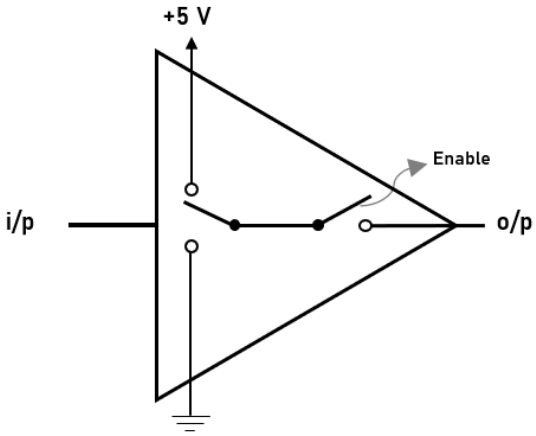
So, when we are going to connect the bus pins, then we need to make sure that only one of the modules are outputting values at the same time. The error or the problem that happens when two modules try to send values to the bus at the same time is called Bus Contention. This is an undesirable state for the system.

We need to really understand what does a logic 1 and 0 mean in a SAP module. In simple words, Logic State 1 is a representation of the 5 V and on the other hand the Logic State 0 is a representation of the GND. And it can be better understood using the source and the sink analogy.



Here when the Q1 is turned ON, the 5 V coming in from the source directly goes through the transistor and through the output taken across the voltage probe. And then we could've said that the gate is sourcing current. And inversely when the Q2 transistor is turned on the current goes to the ground from the o/p. And then it's kind of sinking the current. So, for the busses, the 0s indicate that the pin is going to sink current and the 1s are sourcing current.

Tri state Buffers



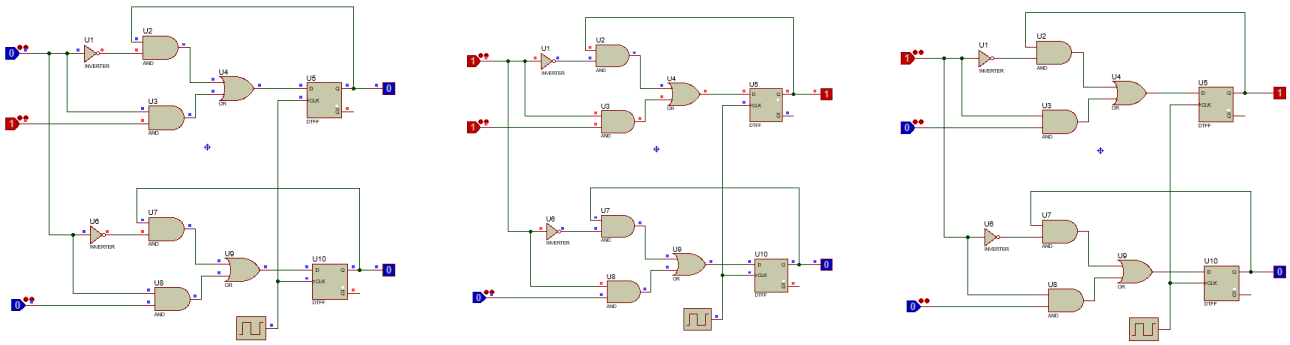
A tri-state buffer is a digital buffer circuit whose output can be electronically disconnected from the bus when required. Here these types of buffers are generally used in the SAP-01 project. Using these buffers, we can ignore the bus contention problem.

Building an 8-bit register

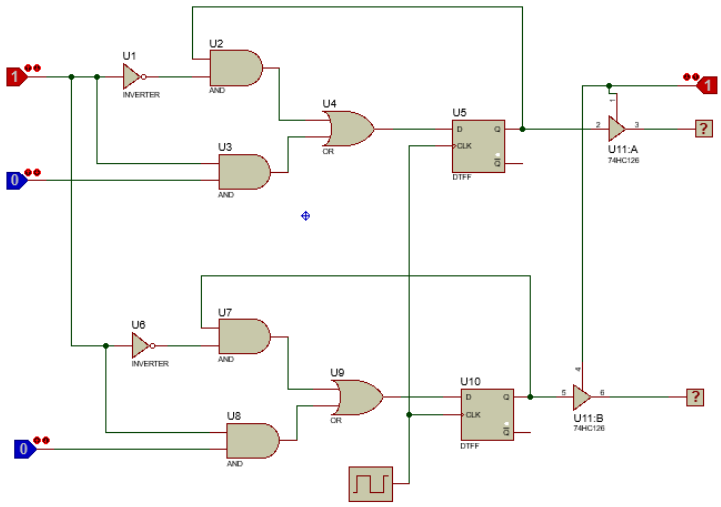
Goal: To load and output 8-bit data from/to the bus upon given instruction avoiding bus contention

Our initial thought would be to get 8 D flip flops and then upon each clock transition we would be able to input and output data but that would leave us with the door of facing bus contention.

So, our first step around our initial thought is to somehow make a combinational circuit design which would load data upon given instruction to do so.



Here our combinational circuit before the D flip flops was integrating the Load pins so that when Load pin is enabled then we can get subsequent values present from the D inputs of the Flip Flops. And then when the load is turned off, we can see from the logic probes that the values are stored after the clock pulse transition. Thus, we understand that the circuit is now able to store values and we have thus designed our register. But our goal is yet to be fulfilled. We still need to electrically disconnect the outputs so that it



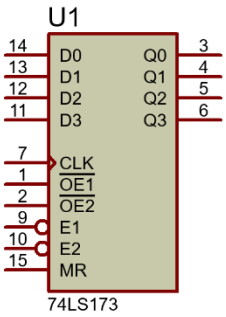
doesn't interfere with the values that is in the bus. And we must avoid bus contention in this case. Thus, we used the tri-state buffers at each of the outputs of the Flip Flops.

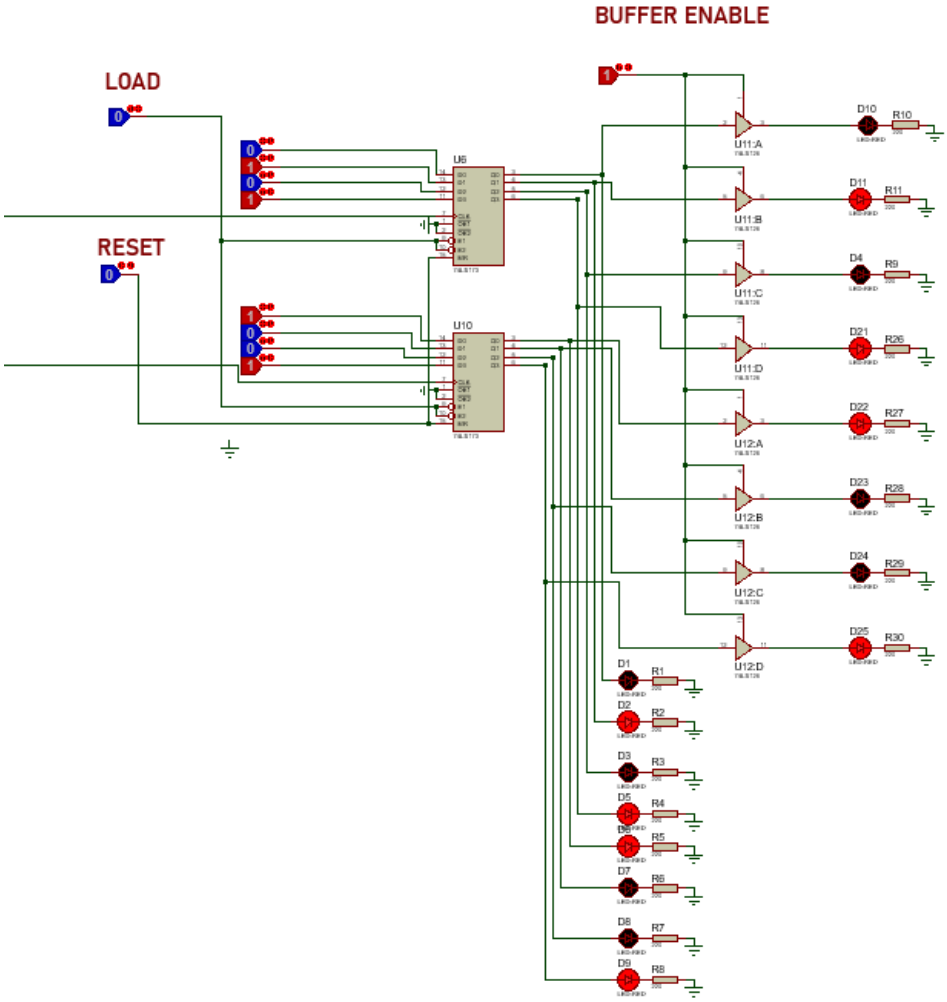
Now, that we know how the basic register is built, we can now move on to the ICs that were used in the registers of the SAP-01.

74LS1731

PINS

Pin No.	Name of Pin	Active Enable	Function
11,12,13,14	D0-D3	High	i/p pins
3,4,5,6	Q0-Q3	High	o/p pins
7	CLK	+ ve Edge	Clock Pulse
1,2	OE1'/OE2'	Low	o/p enable pin
9,10	E1/E2	High	Load pin
15	MR	High	Master Reset

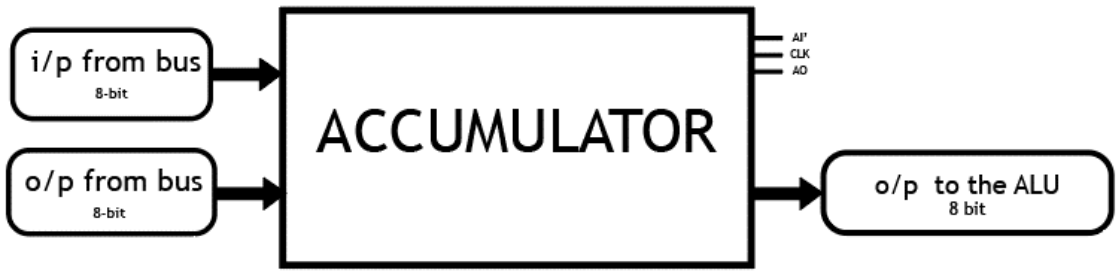




Here we used two 74LS173 IC which is basically used as an 8-bit register. Which has a load pin and reset pin which can be manipulated according to the user's demand. And we had used 8 tri-state buffers with the outputs of the registers. Where we had grounded the 1 and 2 pins i.e., the enable pins of the register. And our approach is going to be limiting the output values from going to the bus by using 8 tri-state buffers for 8 outputs. We had connected LEDs as required with pull up resistors in order to see if our register is working.

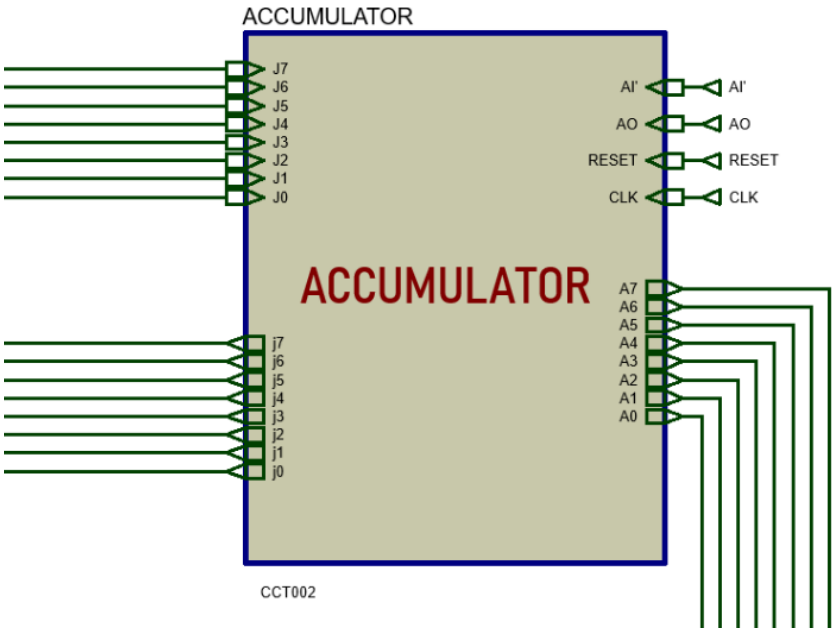
Accumulator

Block Diagram



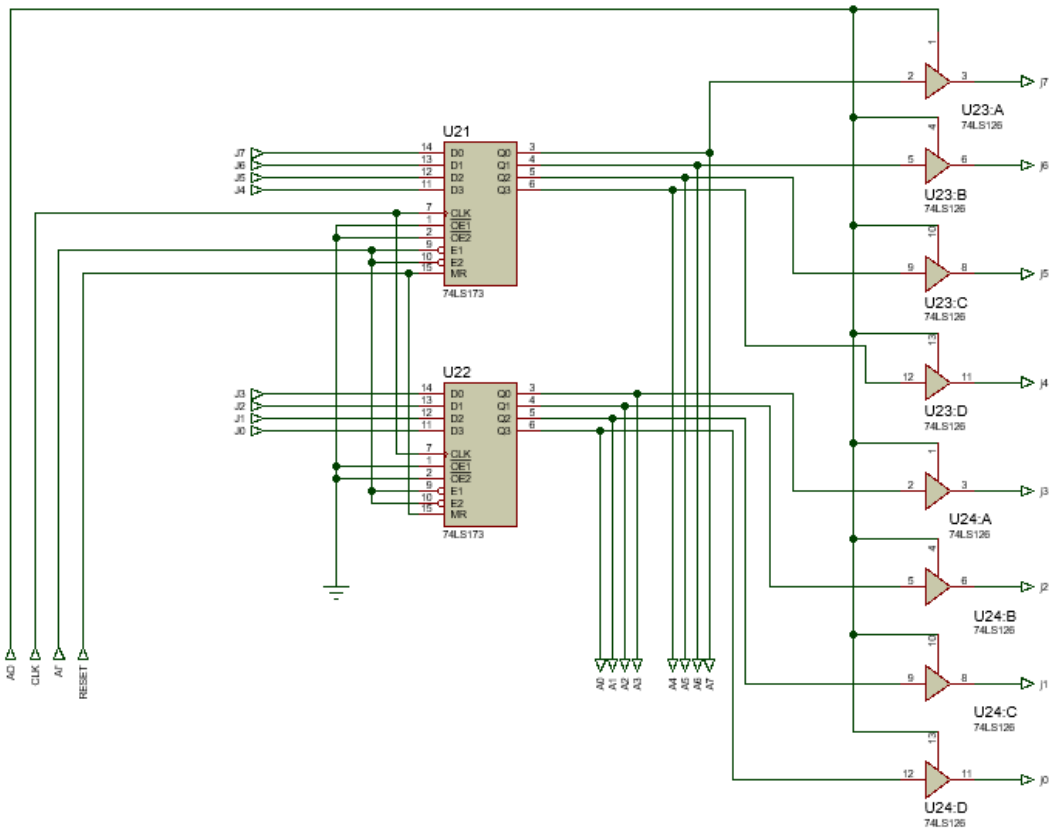
Specifications:

IC	Input pins	O/P pins (bus)	O/P pins (ALU)	Control Pins
74LS173 4-BIT D-TYPE REGISTERS	J7(MSB)-J0(LSB)	j7(MSB)-j0(LSB)	A7(MSB)-A0(LSB)	AI', AO, CLK, RESET



Algorithm

1. Here we have used two 74LS173 4-bit D type register.
2. We have used the 11,12,13,14 pins of the register as the inputs of the Accumulator.
3. And then the 7-no. pin is the Clock Pulse input of the registers. So, we short them together and then we take them as the inputs of the accumulator clock pulse coming in from our “Mode Switching”.
4. The no. 1 and 2 pins are active low. So, we make sure to keep them grounded so that the registers always store input value and gives output enabled.
5. 9 and 10 are the “Load” pins. Here, these pins are active low. So, when value transitions from 1 to 0, then the inputs pins are enabled to receive values from the bus.
6. 15 is the “RESET” pin. We short the 15 pins of both registers.
7. 3,4,5,6 are the output pins. And the direct values of these pins go to the ALU.
8. The values which come out of 3,4,5,6 goes through a tri-state buffer which has a singular input to enable/disable these values. So, this will help us, integrate the concept tri-state buffer in our accumulator to save us from Bus contention.

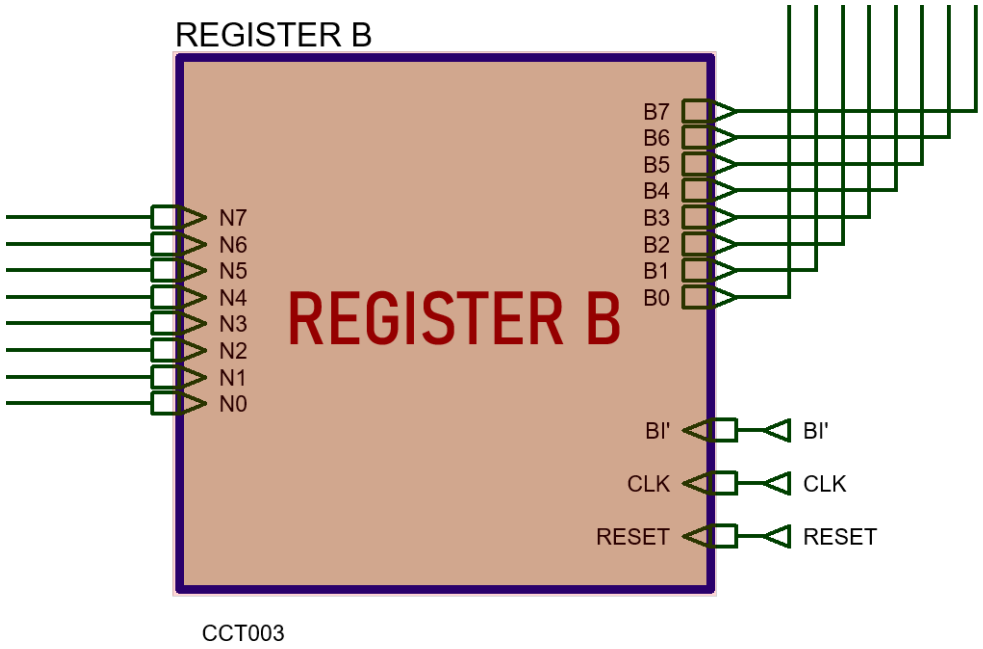


Register B

Block Diagram

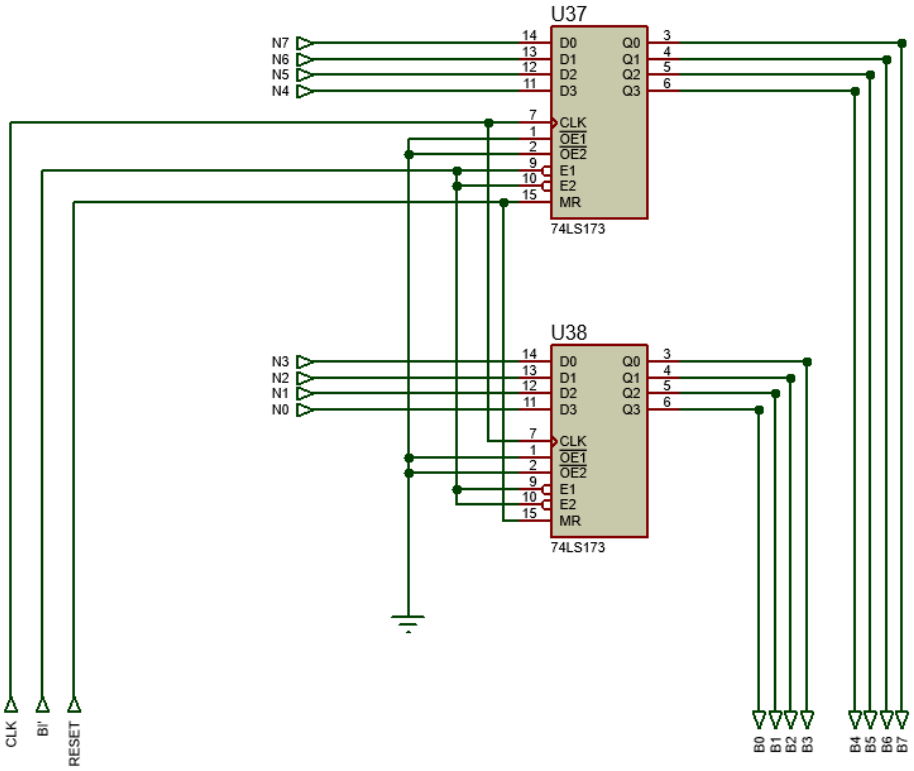
Specifications:

IC	Input pins	O/P pins (ALU)	Control Pins
74LS173 4-BIT D-TYPE REGISTERS	N7(MSB)-N0(LSB)	B7(MSB)-B0(LSB)	BI', CLK, RESET



Algorithm

1. Here we have used two 74LS173 4-bit D type register.
2. We have used the 11,12,13,14 pins of the register as the inputs of the Register B.
3. And then the 7-no. pin is the Clock Pulse input of the registers. So, we short them together and then we take them as the inputs of the accumulator clock pulse coming in from our “Mode Switching”.
4. The no. 1 and 2 pins are active low. So, we make sure to keep them grounded so that the registers always store input value and gives output enabled.
5. 9 and 10 are the “Load” pins. Here, these pins are active low. So, when value transitions from 1 to 0, then the inputs pins are enabled to receive values from the bus.
6. 15 is the “RESET” pin. We short the 15 pins of both registers.
7. 3,4,5,6 are the output pins. And the direct values of these pins go to the ALU.



PHASE C

ALU

Arithmetic Logic Unit or in short ALU will perform the task of addition and subtraction of 8-bit BCD numbers. At the basic level, a computer cannot perform subtraction, multiplication, and division operations rather it uses addition to do these operations. In our 8-bit computer design, we will be sticking only to subtraction and addition operations.

.

Addition

Computers generally use some logic mechanism to perform addition operation. This can be done using formulating a truth table and building equations using k-maps. At first, let us see how the 2bit binary addition works.

Half adder:

2bit basic add operation

0 + 0 = 00
0 + 1 = 01
1 + 0 = 01
1 + 1 = 10

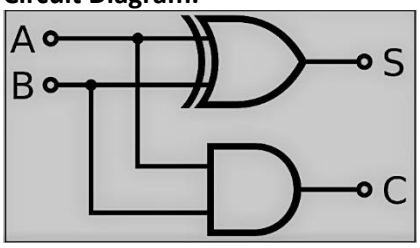
Truth Table:

A1	A0	C0	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Boolean Expression:

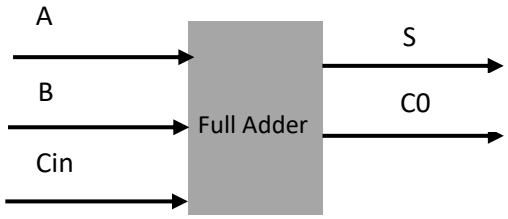
S = A1' A0 + A0 A1'
= A1 ⊕ A0
C0 = A1 A0

Circuit Diagram:



Full ADDER:

The full adder is quite similar to the Half adder but in this case, the adder circuit takes into account the carry input from the previous adder. So it adds 3 inputs (A, B, Cin) and outputs Co(Carry Out) and S(Sum).



Truth Table:

A	B	Cin	Co	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

S:

AB/Cin	0	1
00	0	1
01	1	0
10	0	1
11	1	0

Equation: A'B'Cin + A' B CCin' + A B'Cin'+ AB Cin

= Cin (A'B'+ AB) + Cin' (A'B +A B')

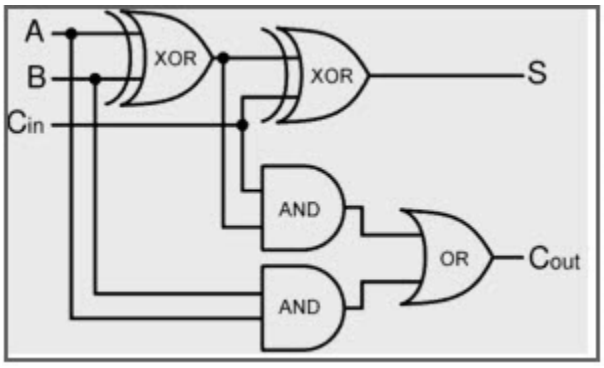
= A ⊕ B ⊕ Cin

C0:

AB/Cin	0	1
00	0	0
01	0	1
10	1	1
11	0	1

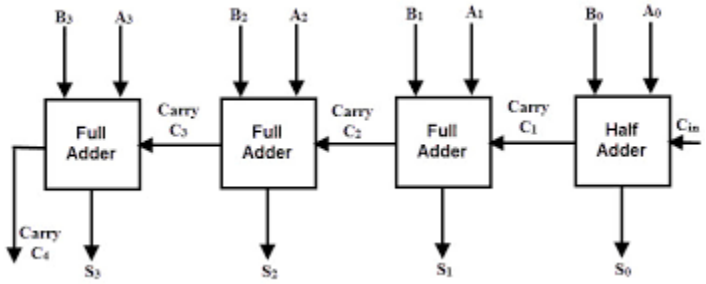
Equation: A'B Cin + AB'Cin + AB Cin' + ABCin

= AB + BCin + ACin

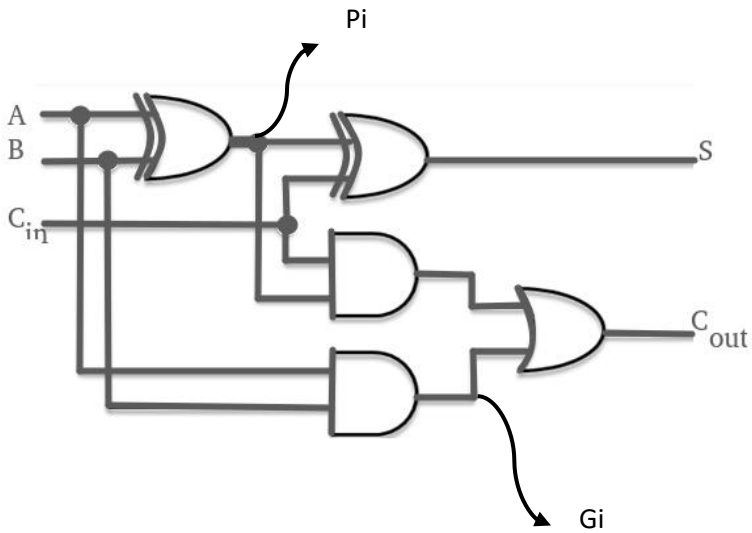


4-bit binary adder using full adder:

Till now we have formulated the combination circuit to perform 2-bit binary addition. We can now use that circuit to build a 4-bit binary adder by cascading the carry-out pin of one circuit with its preceding ones.

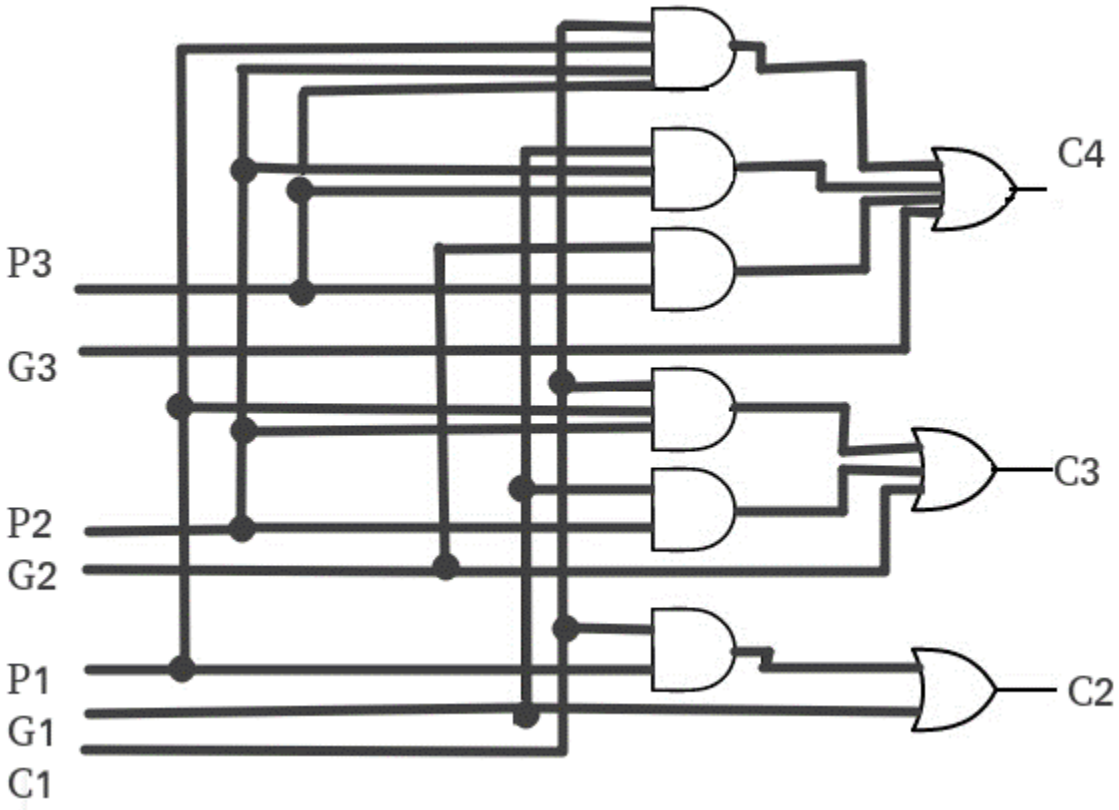


But there is a problem in this type of binary adder which is the carry propagation delay. In the case of the first adder block, the inputs (A0, B0, Cin) are instantly available. However, the remaining adder blocks will have to wait for the previous blocks to pass the carry input pin information. So if there are nth blocks of the adder circuit, it will have to wait for nth-1 blocks to do its operation. This is where the "Look ahead carry adders" came into action.



Here Cout is the carry output of the next carry.
So we can say that Pi = Ai ⊕ Bi and Gi = Ai Bi.
Pi is called the carry propagation and Gi is called the carry generation
Therefore, Si = Pi ⊕ Gi and Ci+1 = Gi + Pi Ci. So we can see that the next carry input and sum are directly dependent on the Inputs Ai and Bi which are readily available. Therefore by using look-ahead carry adders we can minimize the time propagation delay.
Putting the values of Pi and Gi we can now formulate equations for C1, C2, C3, and C4 :

C1 = C0.P0+G0.
C2 = C1.P1+G1 = (C0.P0+G0).P1+G1.
C3 = C2.P2+G2 = (C1.P1+G1).P2+G2.
C4 = C3.P3+G3 = C0.P0.P1.P2.P3 + P3.P2.P1.G0 + P3.P2.G1 + G2.P3 + G3.



Subtraction:

From previous discussions, we have known that a computer does its subtraction operation using the logic circuit used for the addition operation. But in this case, the number that needs to be subtracted should be negative. There are different ways to make a binary number negative. For example, Signed magnitude, 1's complement, 2's complement.

Signed Magnitude:

In general, mathematical numbers are made up of two things one is its sign (+/-) and the other is its magnitude. For example, positive and negative 48 in the decimal number system is represented as +48 and -48 respectively. So, each number should contain a digit that represents its sign. Similarly, in the case of binary numbers, the end binary value or MSB (Most Significant Bit) represents its sign bit. If '1' is used at the end it is treated as a negative number whereas a '0' is used for a positive number.

Positive number $(00101011)_2$ = $+(43)_{10}$

Negative number $(10101011)_2$ = $-(43)_{10}$

Sign Bit

4-bit Signed magnitude numbers:

Signed Bit	4	2	1	value
1	1	1	1	-7
1	1	1	0	-6
1	1	0	1	-5
1	1	0	0	-4
1	0	1	1	-3
1	0	1	0	-2
1	0	0	1	-1
1	0	0	0	-0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

A problem with this technique is that we are getting two representations of 0 in the binary number system. Again, if we add any two positive and negative numbers except 0 we are getting the wrong values.

Positive number $(00101011)_2$ = $+(43)_{10}$

Negative number $(10101011)_2$ = $-(43)_{10}$

Sum $(011010110)_2$ = $(214)_{10}$ which is not the expected value we are looking for. To solve this problem, we will try 1's complement method.

1's Complement:

1's complement of a binary number is the value obtained if we swap each 0 with 1 and vice versa. 1's complement is also a negative number representation technique. A table is drawn below which contains 4-bit 1s complement binary numbers.

8	4	2	1	value
1	0	0	0	-7
1	0	0	1	-6
1	0	1	0	-5
1	0	1	1	-4
1	1	0	0	-3
1	1	0	1	-2
1	1	1	0	-1
1	1	1	1	-0
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5

0	1	1	0	6
0	1	1	1	7

In this case, also we are getting two representations of decimal 0 in binary. Now let us add two positive and negative numbers in 1s complement form:
Positive number $(0011)_2$ = $+(3)_{10}$
Negative number $(1100)_2$ = $-(3)_{10}$
Sum $(1111)_2 = (-0)_{10}$. So, we are getting a negative of zero as an invalid result. So to solve the problems of Signed magnitude and 1's complement we will use 2's complement method.

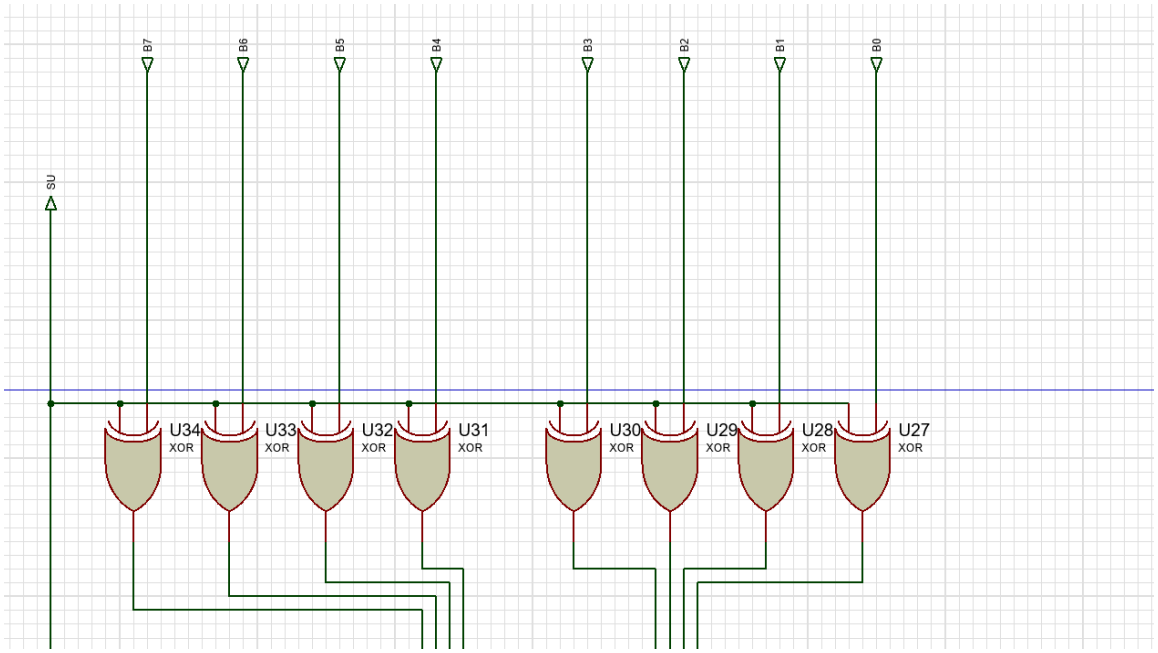
2's complement:
2's complement is the most versatile method used by computers to represent signed integer numbers. 2's complement is nothing but 1's complement + 1. For example, 2's complement of $(000)_2$ is equal to the 1's complement of $(000)_2 = (111)_2$ and then adding 1 with the result, $(111)_2 + (1)_2 = (1000)_2$
4bit 2's complement representation is given bellow:

8	4	2	1	value
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

In this case, we can see that the double zero representation of a binary number is removed. Only the value (0000) is representing decimal 0. Now let us add two numbers and check if it works or not:
Positive number $(0101)_2$ = $+(5)_{10}$
Negative number $(1011)_2$ = $-(5)_{10}$
Sum $(0000)_2 = (0)_{10}$. So, we are getting the desired result in this method.
2's Complement Logic diagram:
The first state of 2's Complement is to complement 8-bit binary numbers coming from the B registers. To do that we will take the help of X-or gate.
If we look into the truth table of X-or gate we will see that whenever the value of B is 1 the output becomes the inverse of A whereas if the value of B is zero the output will be the same as A.

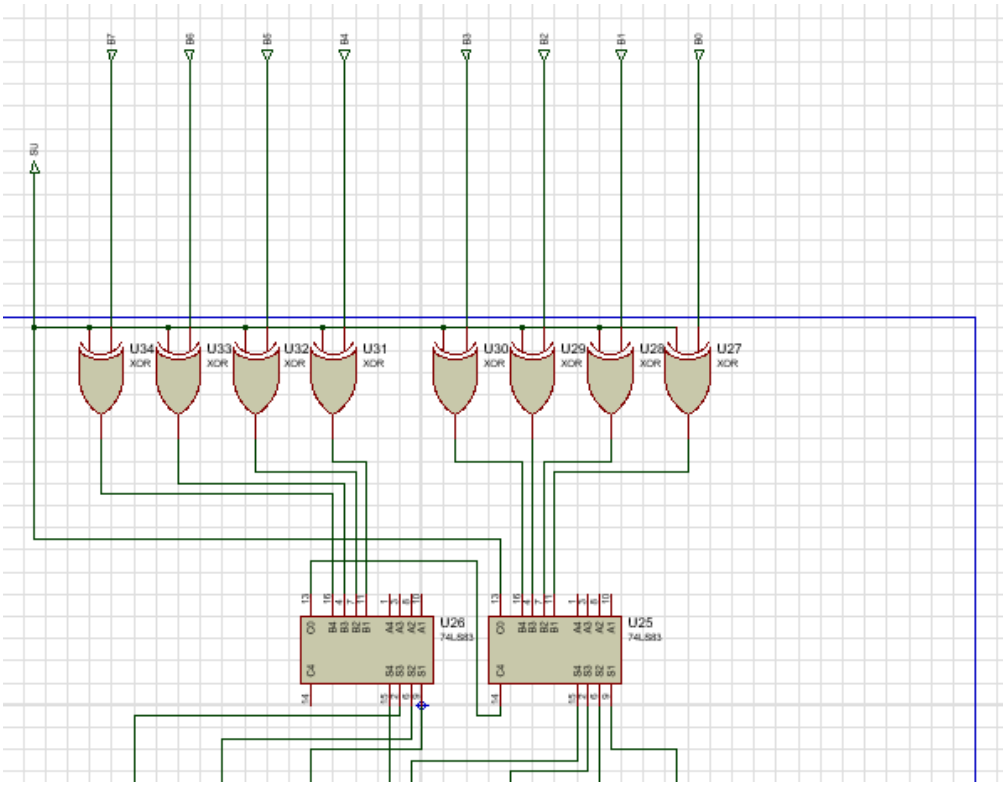
A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

So, we can complement a value by just controlling a single pin of the XOR gate and the remaining pin should be connected with the bit that we need to inverse.



So here If we make the SU pin high the input of the other pin of each X-or gate will be the inverse of it and if the SU is LOW the output of the X-or gate will be as same as the input bits.

The next thing that we need to consider is to add 1 with the complemented value of 8-bits. I order to do that we can connect the SU with the Carry in pin of the 1st adder. So, whenever a value came into the adder circuit, its lower nibble will be added with 1, and consecutively the carry out of the first adder will convey the carry bit for the next adder. Thus a 2's complement value can be found out.

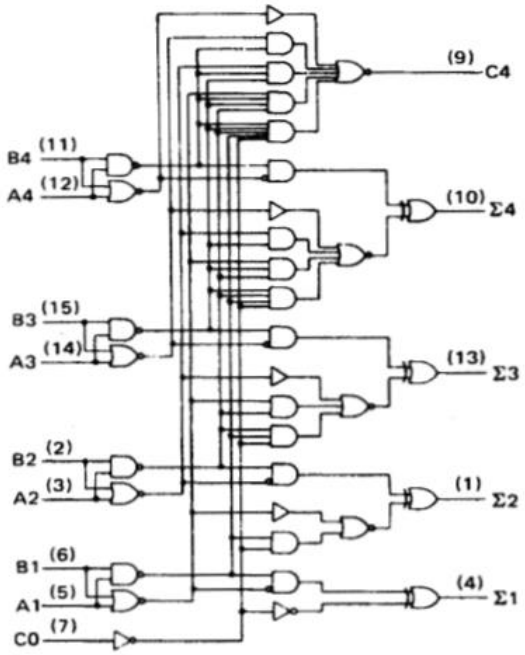


IC description:

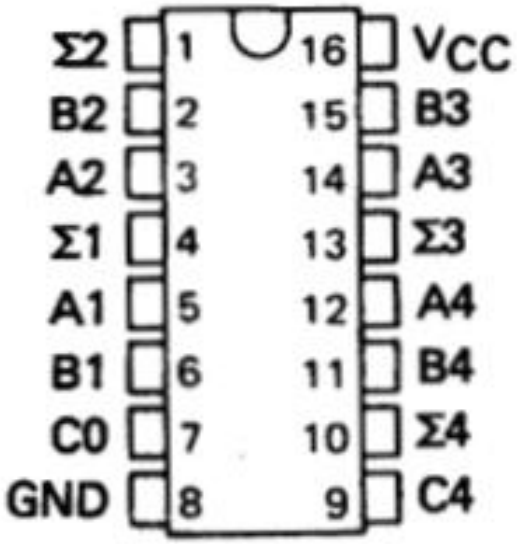
For our simulation, we will be using 74ls87 4-Bit Binary Full Adders with Fast Carry. We are using this IC as the number of IC required for building it with basic scratch logic gate increases exponentially with the number of bits. For example, for n bits, the total AND gates needed is n(n+1)/2, and or gate needed is n which is not feasible.

Pin	Specification
16	VCC
8	GND
5,3,14,12	Output from accumulator
6,2,15,11	Output from 2s complemented value of B register
4,1,13,10	The output of the Adder
C0(7)	Carry in
C4(9)	Carryout

logic diagram (positive logic)

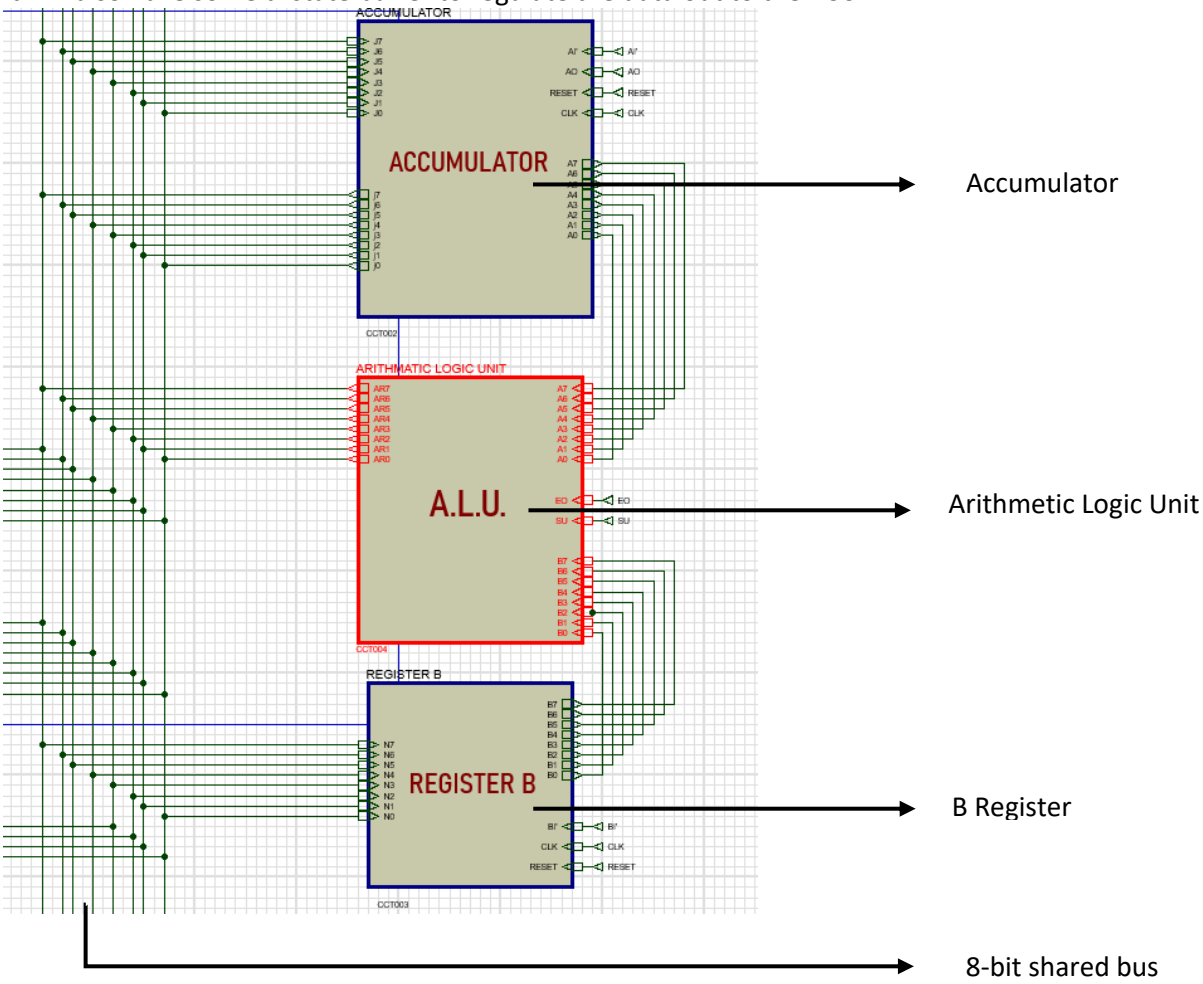


(TOP VIEW)



Block Diagram:

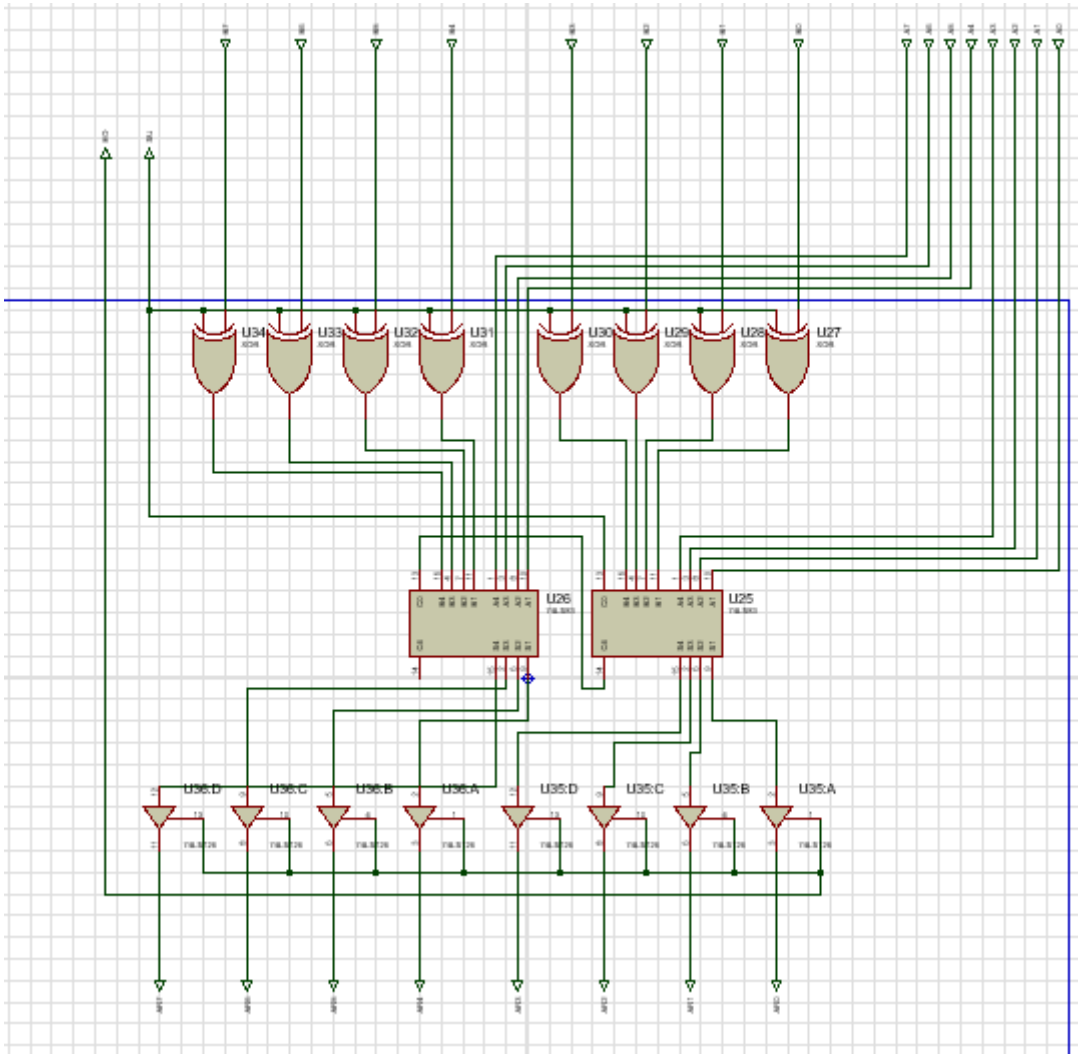
The arithmetic logic unit takes data from the Accumulator and B registers, which are directly connected without any buffer. As a result, any data stored in the A and B registers will go directly for sum operation. The arithmetic logic unit has some logic input to choose in between subtraction and addition operations. It will also have some tristate buffer to regulate the data out to the BUS.



Algorithm:

- Data input from Accumulator and B registers.
- Including combinational logic circuits to interchange between Addition and subtraction operations.
- Cascading 2 4bit 74ls83 IC to convert it to 8-bit Adder.
- Connecting tri-state buffer with the output of the adder circuit.

Circuit Diagram:

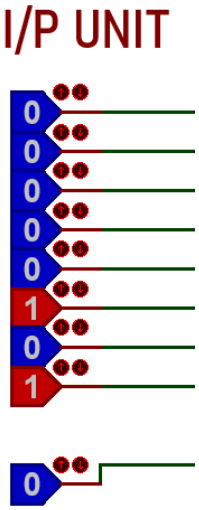


PHASE D

Input Unit & MAR

The input unit & Memory Address Register of our SAP works in co-ordination. When the SAP is in program state the RAM takes data from the input unit & address data from the input address switches. And when in run state the SAP does not take any input data it only takes address inputs from the memory address register(MAR). As the RAM takes address data from two different input modules in two different states so a multiplexer is needed in this case.

Input Unit

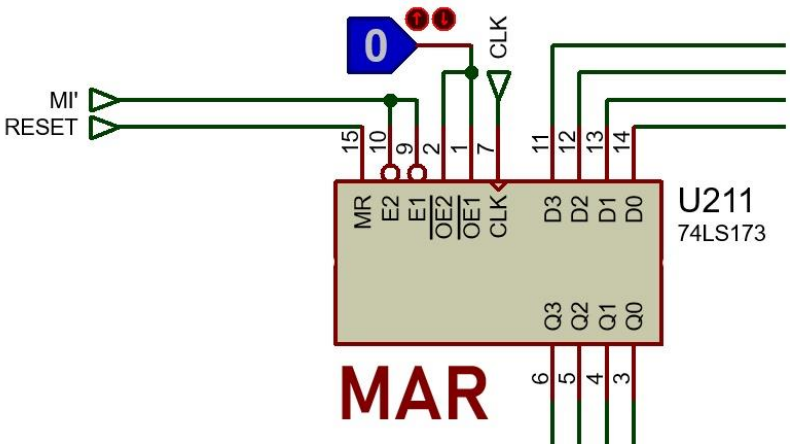


For building the data input unit of the RAM which is only active in the program state we used some logic states to give binary input to the SAP.

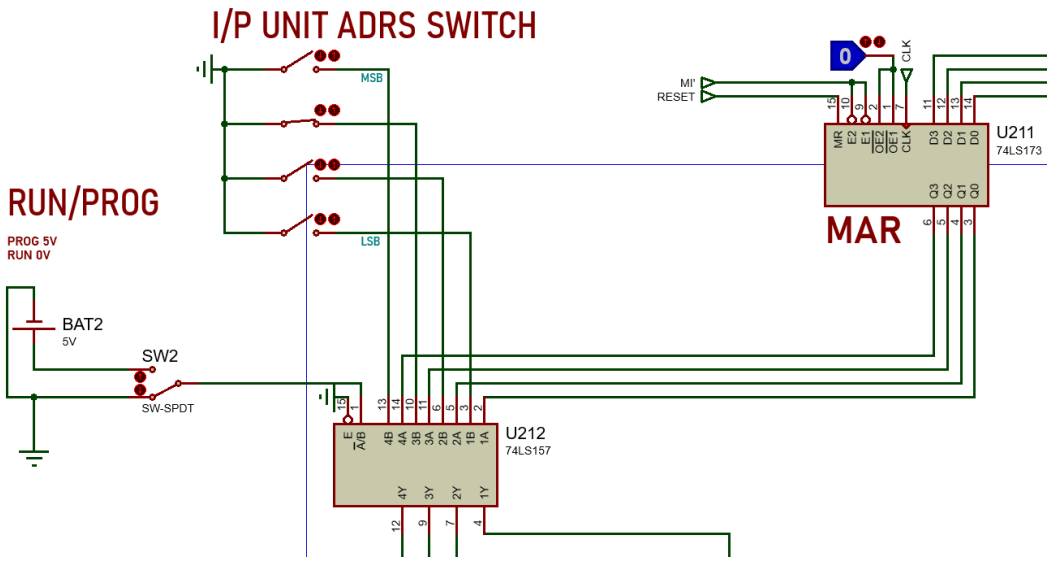
Here in the input unit we can see that there are 8 logic states to give 8 bit data input to the RAM. There is also an extra logic state which controls whether the RAM should be in Read mode or Write mode.

Memory Address Register(MAR)

For the memory address register we have used the 74LS173 IC as we used it for all of our register. Our memory address contains only 4 bits so we only need one 74LS173 IC for MAR. This IC is a 4-bit buffer register. We have grounded the input pin 1 & 2 because the MAR is not connected to the bus so there is no need to use the three-state output for MAR. This shorting of pin 1& 2 with the ground converts the three-state output to a two-state output.



Address Input & MAR

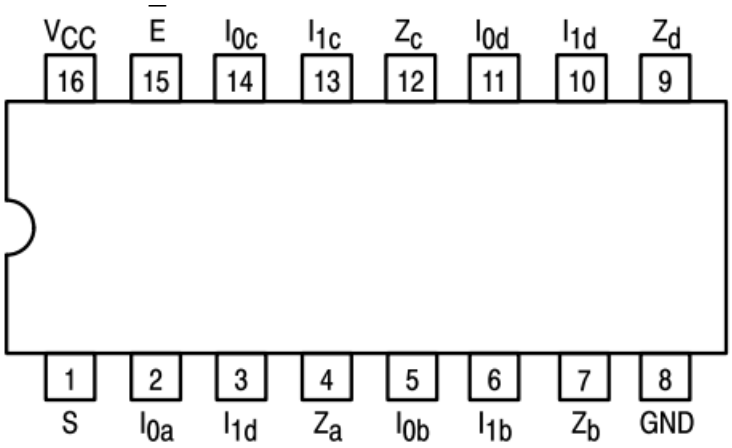


As the RAM takes address data from two different inputs. From the address switch while in the program state & from the MAR in RUN state. Two select the different input modules in different states we used a 2 to 1 multiplexer. For multiplexing the 74LS157 IC was used.

74LS157-Quad 2 input multiplexer

74LS157 is a high speed Quad 2-Input Multiplexer. Four bits of data from two sources can be selected using the common Select and Enable inputs. The four buffered outputs present the selected data in the true (non-inverted) form. The LS157 can also be used to generate any four of the 16 different functions of two variables.

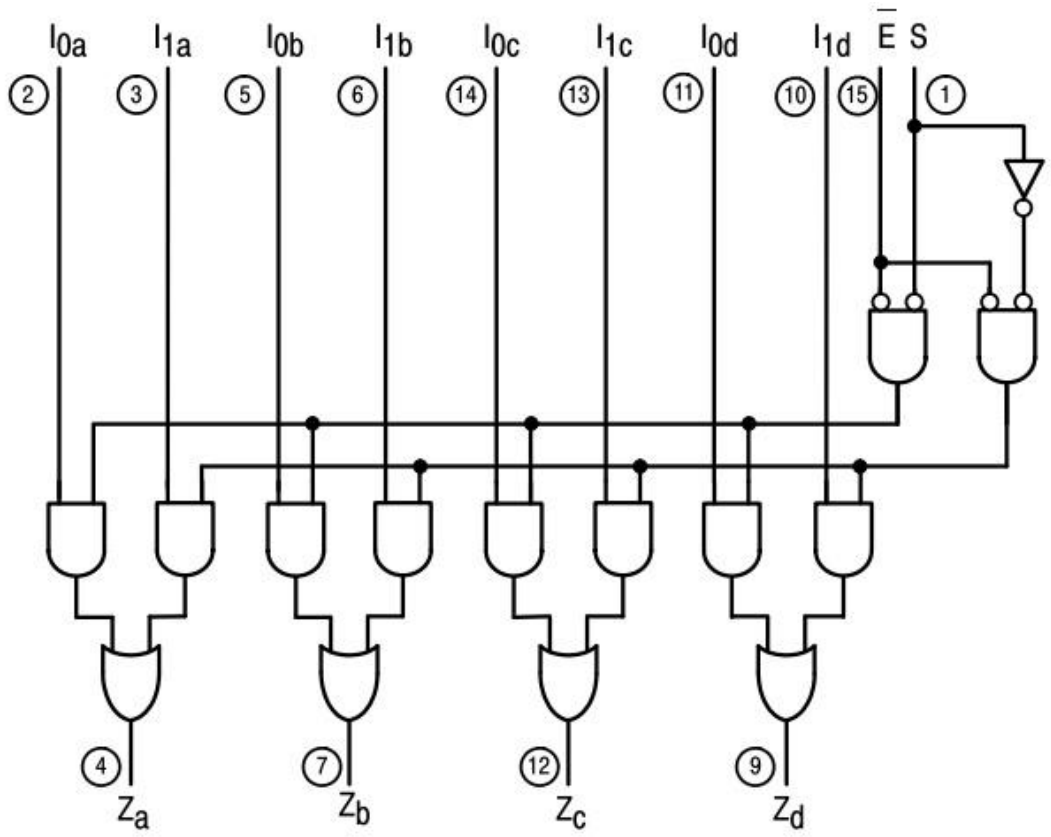
Block Diagram



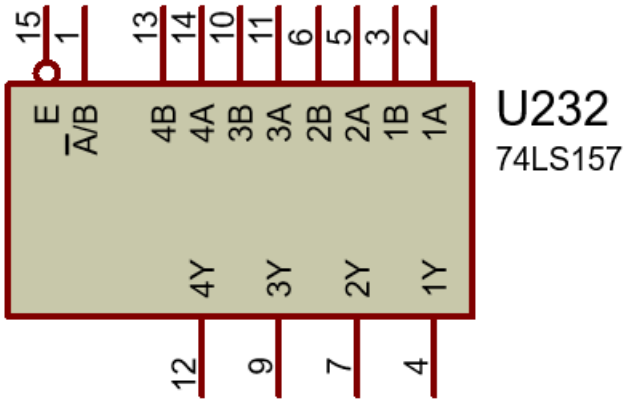
Truth Table

ENABLE	SELECT INPUT	INPUTS		OUTPUT
$\overline{E}$	S	I_0	I_1	Z
H	X	X	X	L
L	H	X	L	L
L	H	X	H	H
L	L	L	X	L
L	L	H	X	H

Combination Circuit Diagram



74LS157 In Proteus



Algorithm

1. The input pins 2,5,11,14 takes input from the MAR & pin 3,6,19,13 takes input from the address switches.
2. Pin 15 is for enabling the mux.
3. Pin 1 is the line selector. For active low(0) inputs from MAR will go to the RAMs address input in the RUN state of the SAP & for active high(1) the inputs from the address switches will go to the RAMs address input in the program state.

RAM

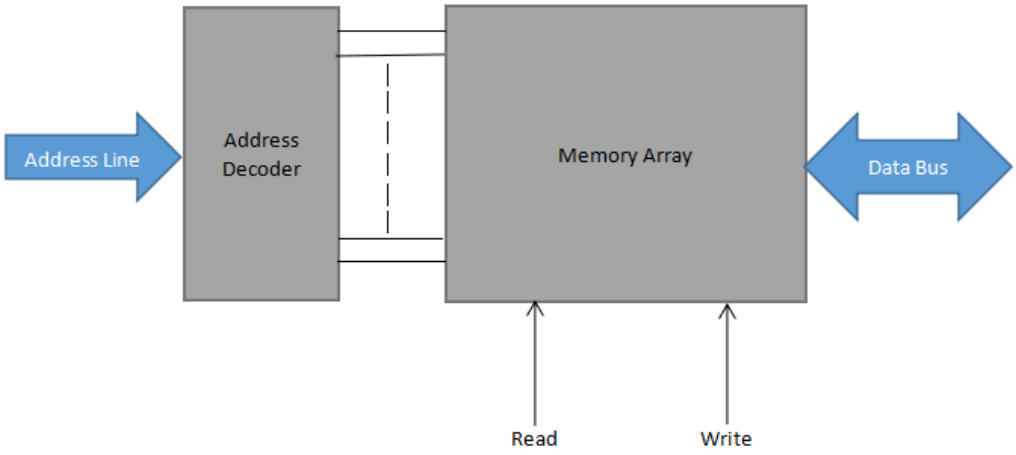
Random Access Memory (RAM) is the hardware of a computer that stores binary data of the currently running programs so that the data can be quickly reached by the computer's processing or logic unit. It is the main memory in any computer.

RAM can not store data permanently rather it stores data of the currently running programs only. It can be compared to the short-term memory of a person. Just a person's short term memory can be refreshed RAM can also store data after each Running of the computer.

Basic Structure of RAM

Inside the RAM each of the memory cells can be accessed for information transfer from any desired random location. Communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

So, a ram consists of address lines,an address decoder, memory array, read & write input and some output wires.



For n number of address line a ram can have 2^n number of address locations. The address decoder takes the input from the address lines and activates one memory location corresponding to that address.

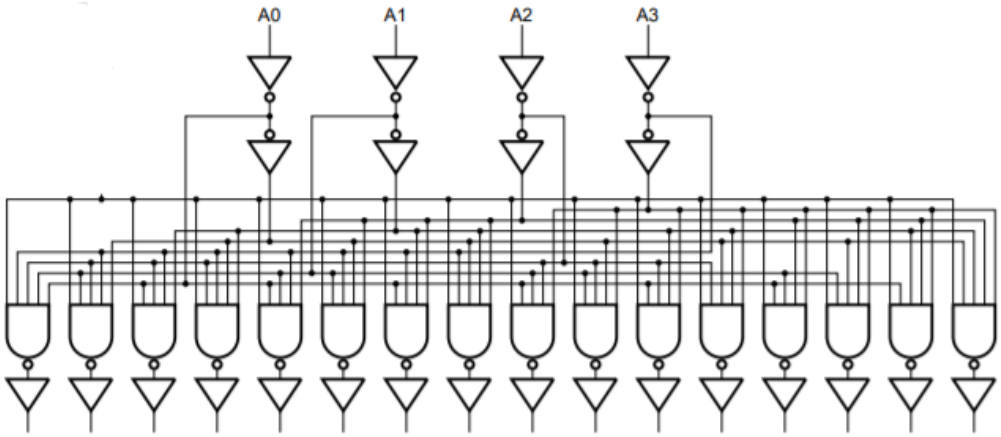
4x16 Bit Decoder

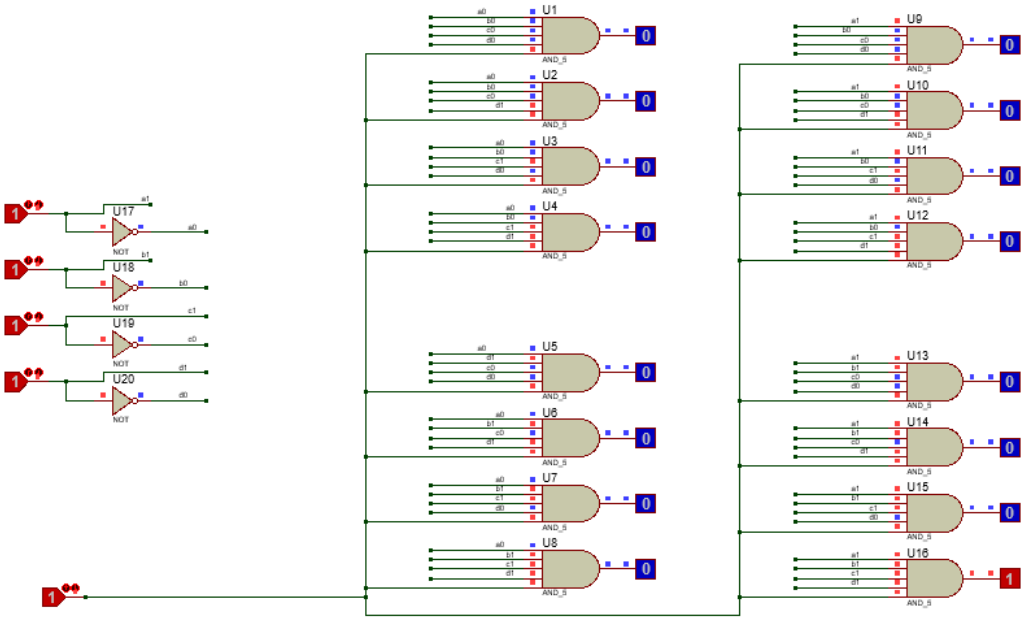
INPUTS				OUTPUTS															
A_3	A_2	A_1	A_0	Y_{15}	Y_{14}	Y_{13}	Y_{12}	Y_{11}	Y_{10}	Y_9	Y_8	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The combinational circuit that changes the binary information into 2^N output lines is known as **Decoders.** The binary information is passed in the form of N input lines. The output lines define the 2^N-bit code for the binary information.

As our ram is 16bytes so there are 16 address locations & our address data is of 4 bits. So, we need a 4x16 bit decoder for our project. The truth table for our Decoder is given.

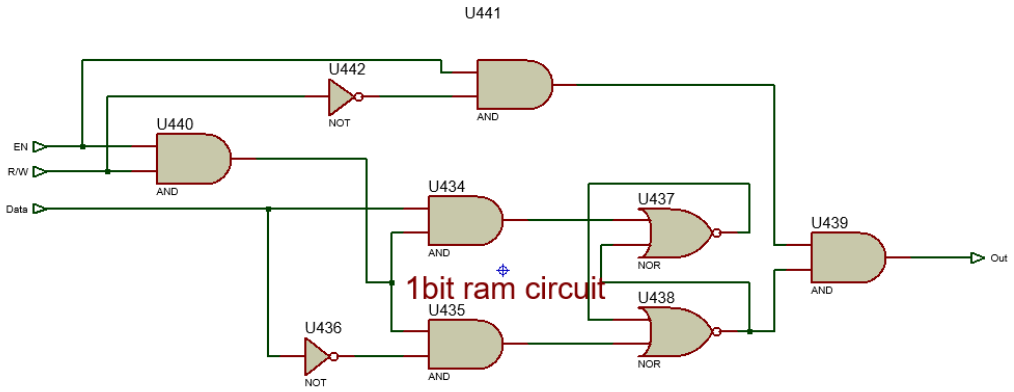
The circuit diagram for our 4x16 decoder is the following.





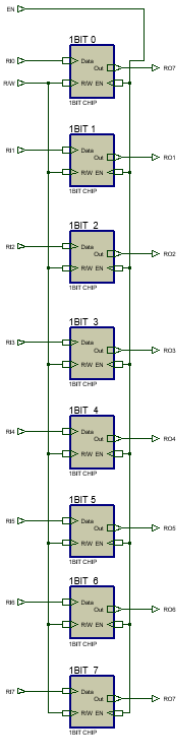
Proteus Simulation of 4x16 decoder

1 Bit Ram Unit



Here we have a 1 bit ram circuit which has 1 data input, 1 read/write selector and an enable input. All of these are incorporated with a SR latch and a couple of NAND gates to store the data.

Read & Write Operation of RAM

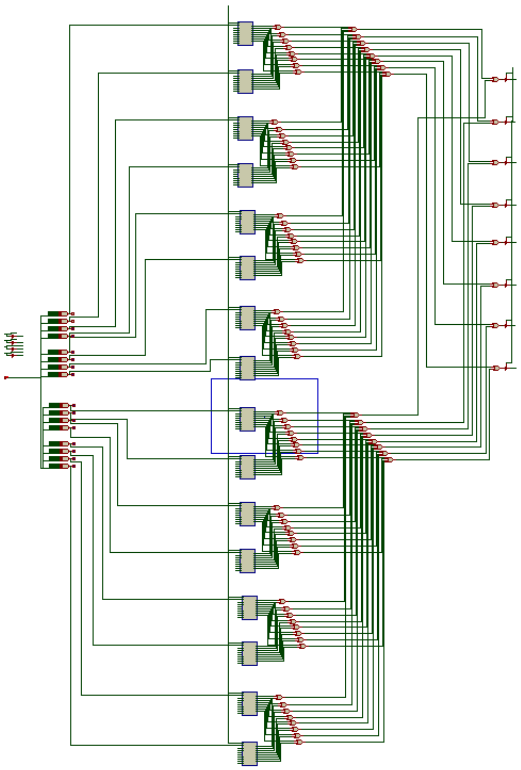


In the 1 bit ram circuit we can see that there are 3 inputs and 1 output. The inputs are enable, R/W & data . When the enable input is 1 the RAM is turned on and when 0 the RAM is off. When the write input is 1 the ram takes the data from the data pin and stores it. And when read input is 1 then the write option is inverted so the ram gives output of the data which was previously stored. **One Address line of the 16 byte RAM**

In this diagram we have created a sub-circuit of the previous 1 bit RAM and shorted all their enable pins & R/W pins so that the can be activated while one address line is selected.

16byte RAM

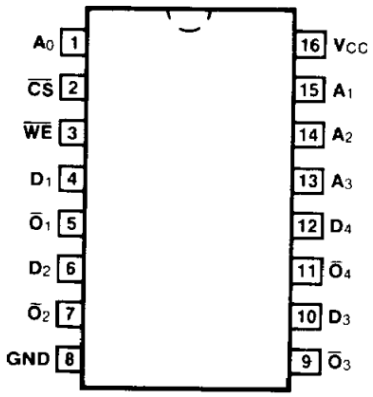
Now using 16 of the 8 bit data units we made a 16 byte RAM that is attached with a decoder.



Building RAM with 74LS189 IC

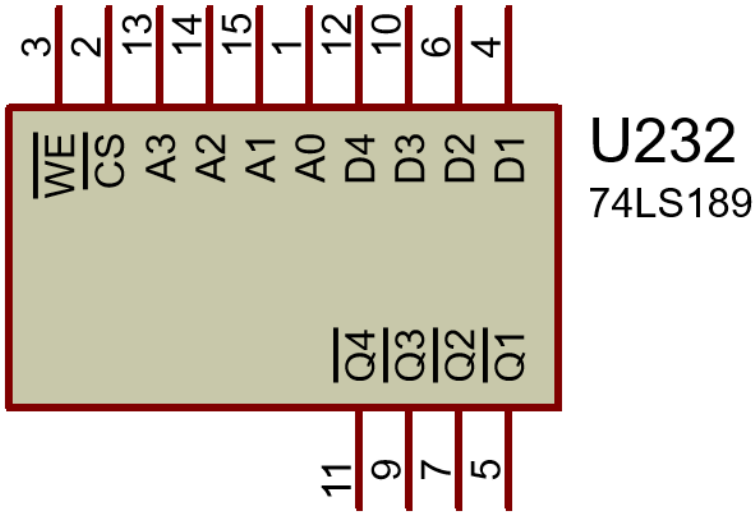
The 16x8 RAM that we built for the SAP has some problem while connecting to the whole computer. The proteus simulation has some problems because there are so many sub circuits and elements inside this RAM. So we have tried the RAM that Malvino used. We used 74LS189 IC which is a 64 bit memory array. So to build a 16 byte or 128 bit RAM we need to take two of this ICs.

The block diagram & truth table of the 74LS189 IC is taken from the datasheet of the IC.



	CE	WE	D_{IN}	DATA OUT		
Read	0	1	X	Stored data	Stored data	Stored data
Write "0"	0	0	0	1	1	Hi-Z
Write "1"	0	0	1	1	1	Hi-Z
Disable	1	X	X	1	1	Hi-Z

74LS189 IC in Proteus

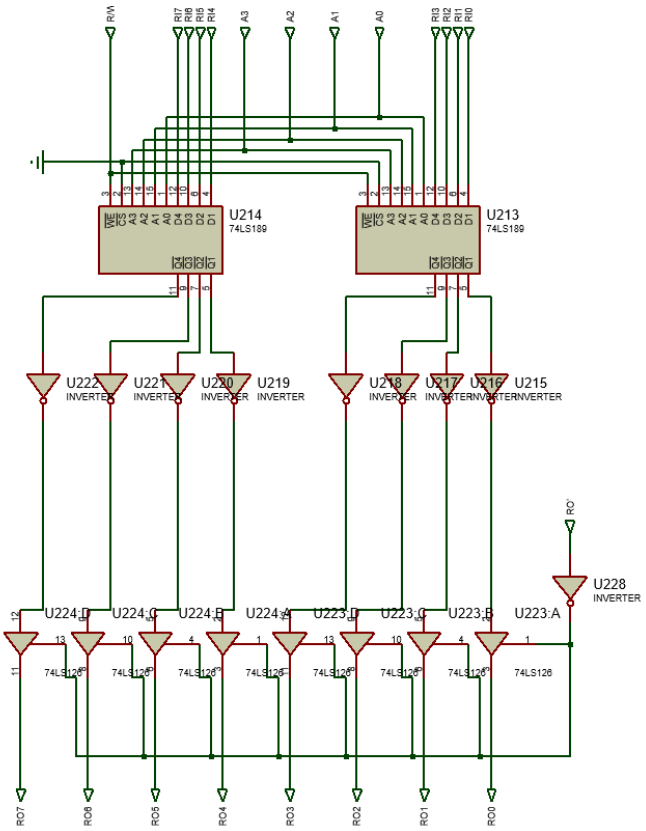


This the 74LS189 IC in proteus. We can see that the pin arrangements are based on the type of input & output rather than serial pin number arrangement.

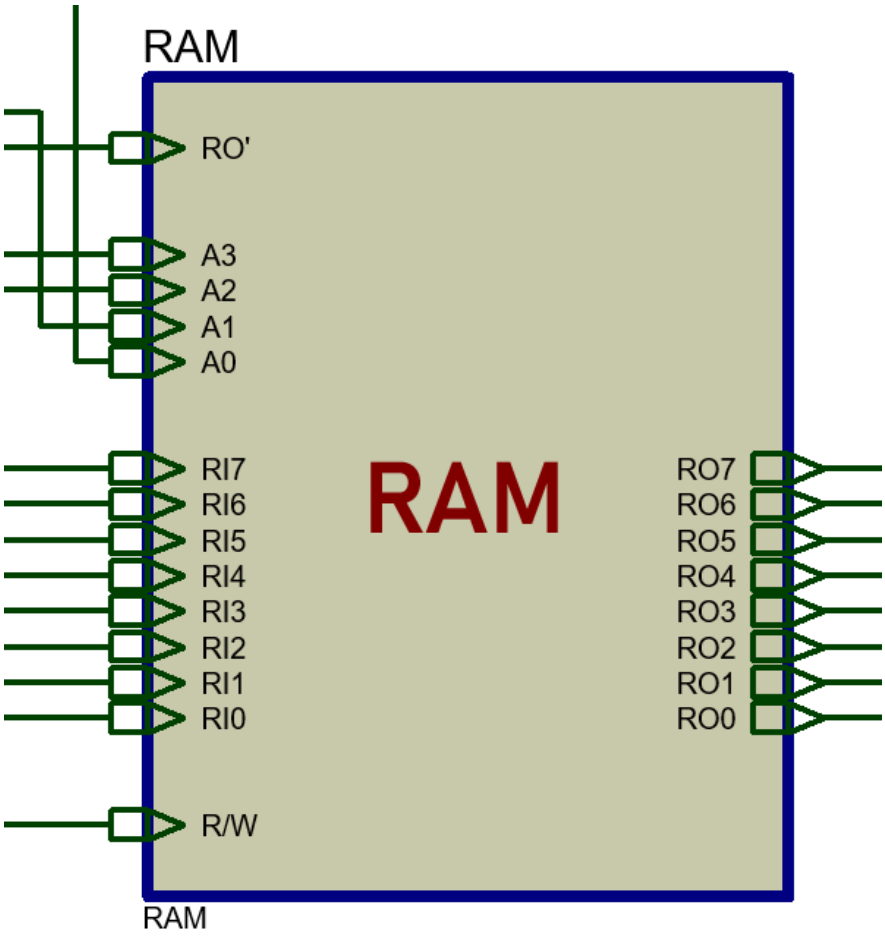
16byte RAM with Two

Pin No.	Name of Pin	Active Enable	Function
13,14,15,1	A3-A0	High	i/p pins
12,10,6,4	D4-D1	High	i/p pins
11,9,7,5	Q4'-Q1'	Low	o/p pins
2	CS'	Low	chip enable pin
3	R	High	Read
3	W	Low	Write

74LS189 IC in Proteus



Each 74LS189 IC is 64bit memory. So to make a 16 byte or 128 bit RAM we have connected two RAM ICs for building this. Then, we have shorted the address wires because it can take the address from the MAR. The output of this IC is by default inverted so we have used 8 inverters to invert the outputs back to their original value.



Algorithm

9. Here we have used two 74LS189 64-bit Memory Array..
10. We have used the 1,13,14,15 pins of the IC as the address inputs of RAM.
11. And then pin 12,10,6,4 is to take inputs of data from the input unit.
12. There is a multiplexer before the ram to decide to take address input from the MAR or the input unit in corresponding run state or program state.
13. The no. 3 pin is the R/W button of the RAM. When the input is low the RAM is in write mode and when the input is high the ram is in read mode.
14. Output pin 11,9,7,5 are inverted so we used inverters to get the original value.

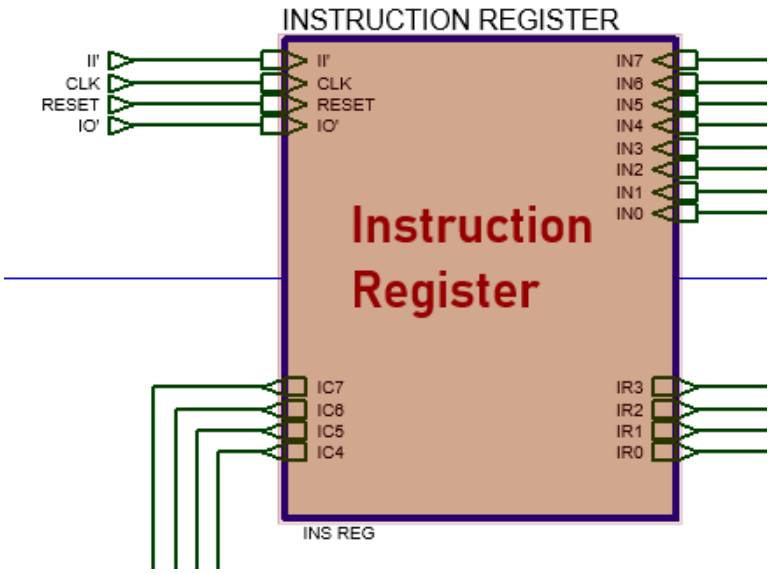
PHASE E

Instruction Register

Block Diagram

Specifications:

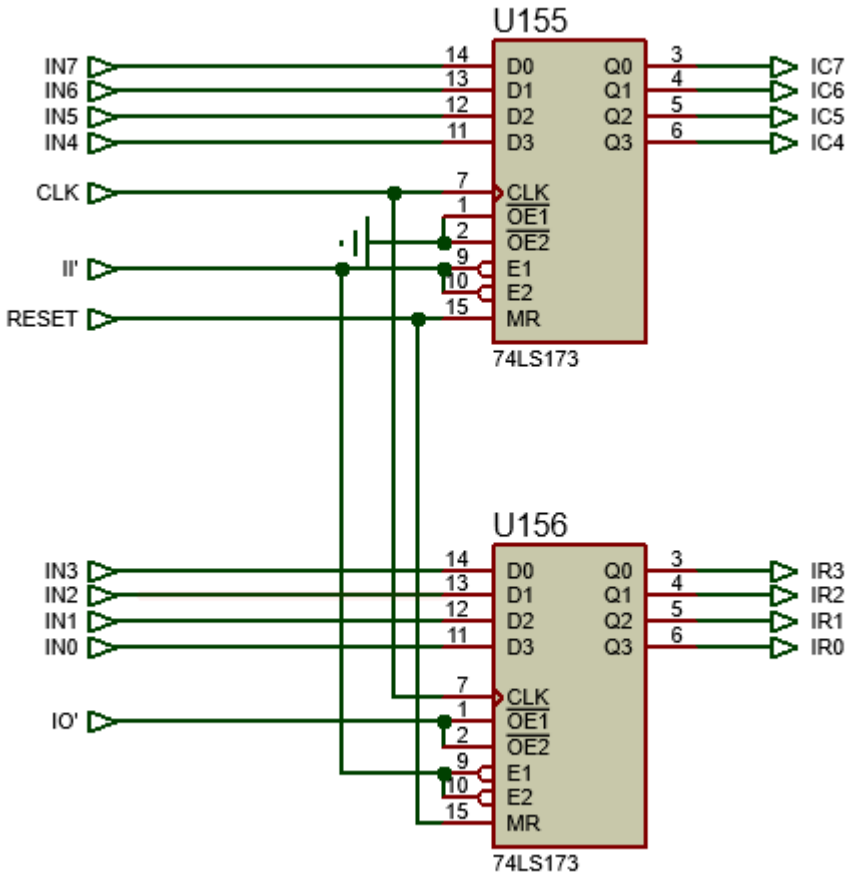
IC	Input pins	O/P pins (bus)	O/P pins (Sequencer)	Control Pins
74LS173 4-BIT D-TYPE REGISTERS	IN7(MSB)-IN0(LSB)	IR3(MSB)-IR0(LSB)	IC7(MSB)-IC4 (LSB)	II', IO', CLK, RESET



Algorithm

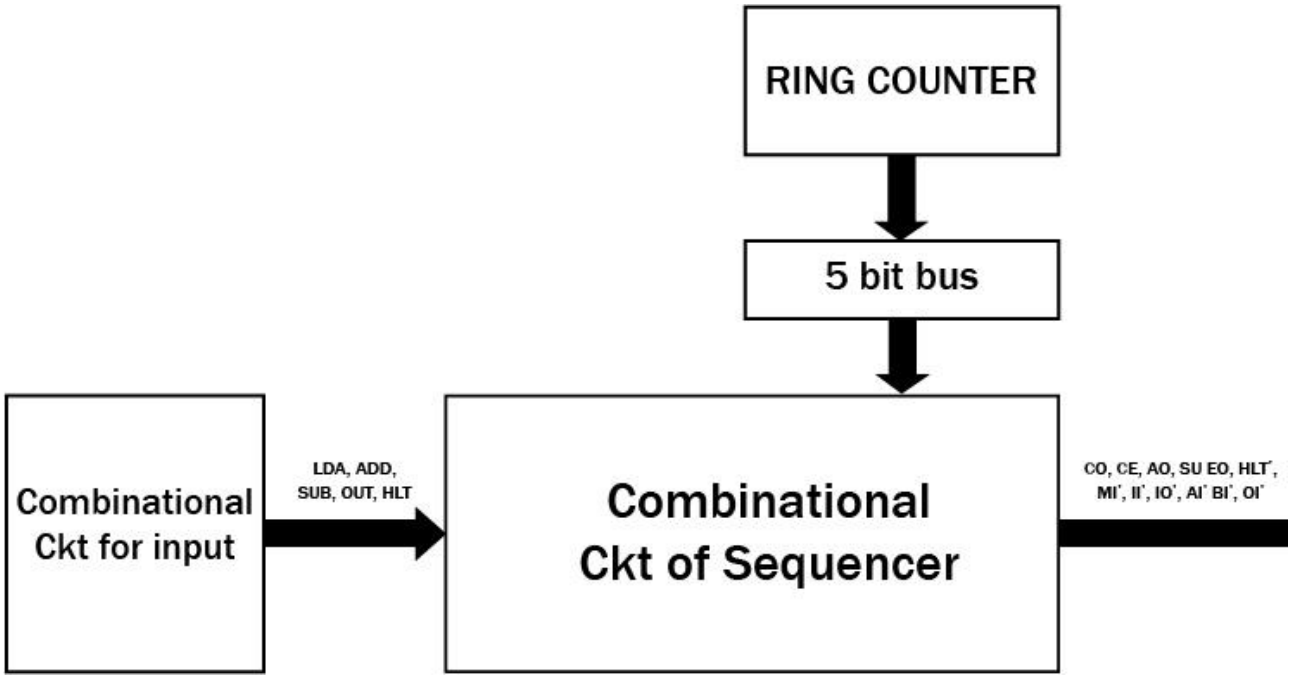
1. Here we have used two 74LS173 4-bit D type register.
2. We have used the 11,12,13,14 pins of the register as the inputs of the Instruction Register.
3. And then the 7-no. pin is the Clock Pulse input of the registers. So, we short them together and then we take them as the inputs of the accumulator clock pulse coming in from our "Mode Switching".

4. The no. 1 and 2 pins are active low. Here, the register which will take the upper nibble will have the pins 1 and 2 grounded always. And the register which will take the lower nibble will have an enable option through these pins as IO' input.
5. 9 and 10 are the "Load" pins. Here, these pins are active low. So, when value transitions from 1 to 0, then the inputs pins are enabled to receive values from the bus.
6. 15 is the "RESET" pin. We short the 15 pins of both registers.
7. 3,4,5,6 are the output pins. The register which will take the upper nibble will have the pins 1 and 2 grounded always. And the register which will take the lower nibble will have an enable option through these pins as IO' input.



Controller/Sequencer

Block Diagram



Purpose

Sequence control refers to user actions and computer logic that initiate or stop sequence. Sequencer controls the SAP and the values that goes in and out of the bus at each transition state. Methods of sequence control require explicit attention in interface design, and many published guidelines deal with this topic.

Specifications

Input pins	Output pins		Control Pins	Ring Counter	T State
	Active low	Active High			
IC7-IC4	CO, CE, AO, SU EO	HLT', MI', II', IO', AI' BI', OI'	CLK, RESET	Jump to "Ring Counter"	5

Programming SAP-1

Op-code and input combinational circuit design

LDA 0000
ADD 0001
SUB 0010
OUT 0011
HLT 1111

So, our idea is to use create a combinational circuit where each significant bits are designed in accordance to whether it is 0 or 1. For example, to get LDA activated we need to make sure that when inputs pins get 0 then LDA=1. So, to ensure this, we get a 4 input AND gate and use the inverted connection from the input pins. So, during this only the AND gate responsible for the LDA activation will be ON and the rest will be OFF. But when we have 1 in the input pins such as in ADD we have 1 in the LSB. So, we connect it directly and then connect it to the AND gate responsible for the ADD activation. Thus, we connect all the other combinational circuits in this manner.

Cycles and Instructions in SAP-1

The address and data switches allow you to program SAP-1. Here the op code will go to the upper nibble and the lower nibble is responsible for the operand. For example,

Address	Instruction
0H	LDA FH
1H	ADD EH
2H	HLT

1. Here at first convert each of these into binaries. For example, LDA will translate to 0000. And FH will translate to 1111. So LDA FH means 0000 1111.
2. Then 1H will translate to 0001 1110. And HLT to 1111 XXXX. And what these binary codes will translate to will be the core in understanding how the controller works.

There are generally **2 cycles** in our SAP-1.

1. Fetch Cycle
2. Execution Cycle

Fetch Cycle

Control unit generates the control words that fetch and execute each instruction and while this happens the computer will pass through various timing states (T STATES).

Address State (T1)

This is where the address of the program counter is transferred to the MAR. During this state CO and MI' are activated.

Increment and Memory State (T2)

This will increase a value in the program counter. Simultaneously, a value will be transferred from RAM to the instruction register

Execution Cycle

LDA

If LDA 9H comes then Instruction Register will receive 0000 1001. And the 0000 will go as input to the sequencer and then the lower nibble will act as operand. And after the T3 cycle, the AI' and RO' will be activated. So the 1001 value will go to the MAR and act as address to the RAM. And RAM will subsequently output whatever value is in that specific address. And accumulator will take this value and temporarily store it.

Summary:

T3	**IO'** **MI'**
T4	**AI'** **RO'**

ADD, SUB and OUT will follow a similar route as the LDA. Only in their cases slight modifications are required as per the op-code.

ADD

T3	**IO'** **MI'**
T4	**BI'** **RO'**
T5	**EO** **AI'**

SUB

T3	**IO'** **MI'**
T4	**BI'** **RO'**
T5	**EO** **AI'** **SU**

OUT

T3	**OI'** **AO**

HLT

When the IR will get 1111 XXXX then HLT' will get activated. It is done so that when 1111 is input in the sequencer then the NAND gate will give 0. And this 0 will go to the AND gate that is present in the Main CLOCK. And when 0 is an input in an AND gate, the output is automatically 0. Thus, we stop the clock and the entire SAP immediately.

Microinstructions

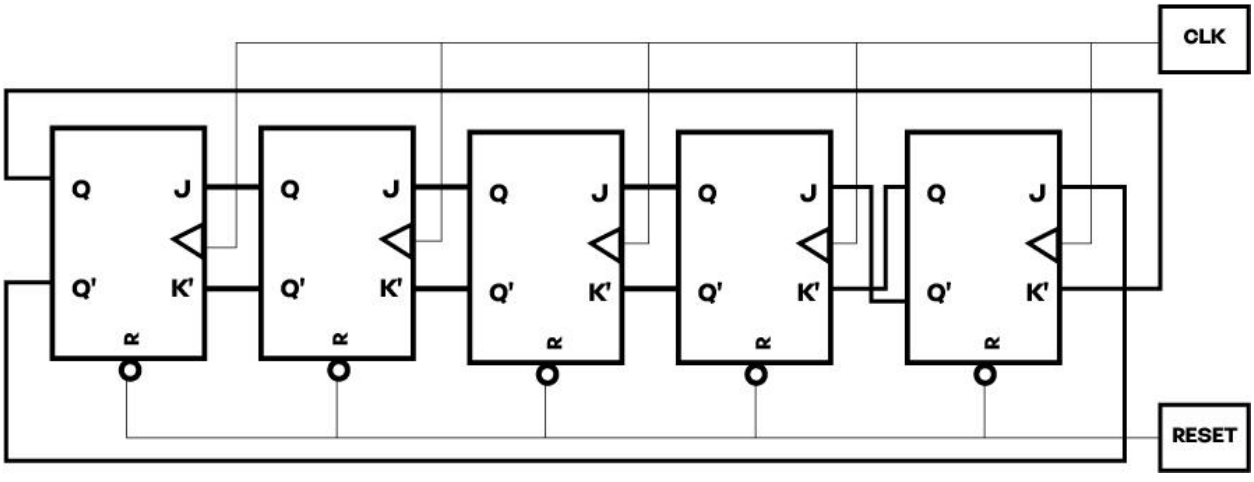
Macro	State	OUTPUT STATE CO CE MI' RO' II' IO' AI' AO EO SU BI' OI'
LDA	T3	0 0 0 1 1 0 1 0 0 0 1 1
	T4	0 0 1 0 1 1 0 0 0 0 1 1
ADD	T3	0 0 0 1 1 0 1 0 0 0 1 1
	T4	0 0 1 0 1 1 1 0 0 0 0 1
	T5	0 0 1 1 1 1 0 0 1 0 1 1
SUB	T3	0 0 0 1 1 0 1 0 0 0 1 1
	T4	0 0 1 0 1 1 1 0 0 0 0 1
	T5	0 0 1 1 1 1 0 0 1 1 1 1
OUT	T3	0 0 1 1 1 1 1 1 0 0 0 0

Ring Counter

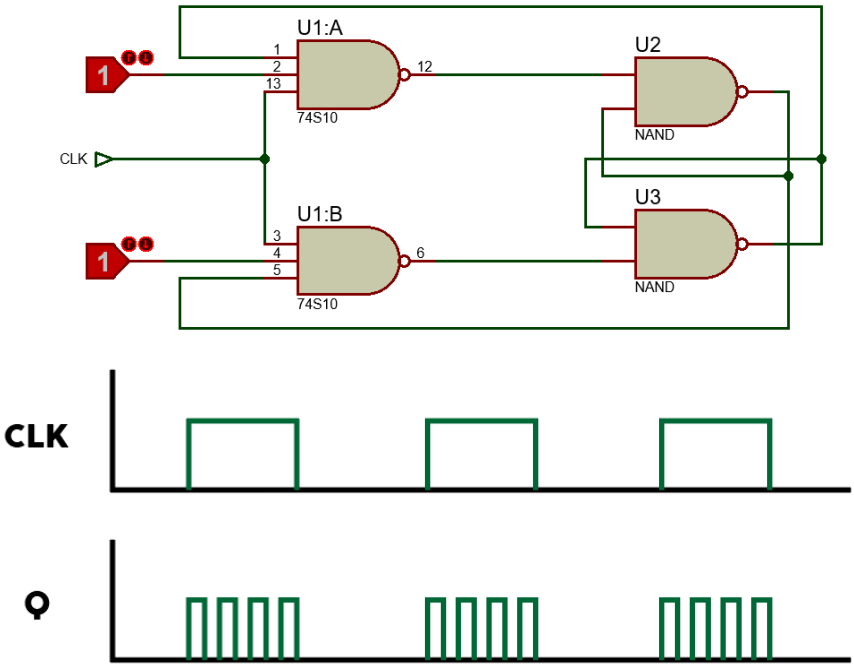
Ring counter is a typical application of Shift resister. Ring counter is almost same as the shift counter. The only change is that the output of the last flip-flop is connected to the input of the first flip-flop in case of ring counter but in case of shift resister it is taken as output. Except this all the other things are same.

https://www.geeksforgeeks.org/ring-counter-in-digital-logic/

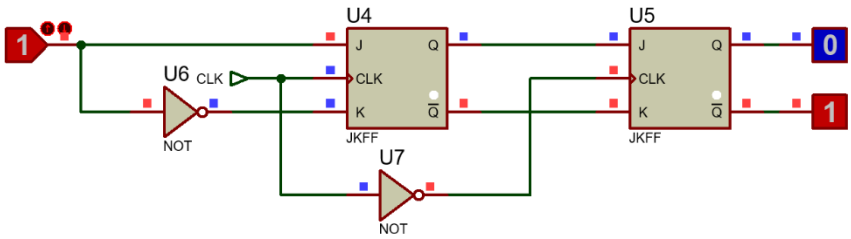
Block Diagram



Master-Slave JK Flip Flop



In the Level triggering mode of the clock pulse, we see that the value of Q keeps toggling during the clock pulse period. And this creates a phenomenon called “Race around Condition”. And now in order to go past this phenomenon, we need to understand the concept of Master Slave JK Flip Flop which will be critical in understanding the Ring Counter.



Here in the master slave JK flip flop we connect the master JK with the CLK and we invert it and connect it to the slave. So thus, here a particular value will be stored for one half of a clock pulse cycle. And then when the clock pulse hits again then the value will go from the output of master to the slave.

Algorithm:

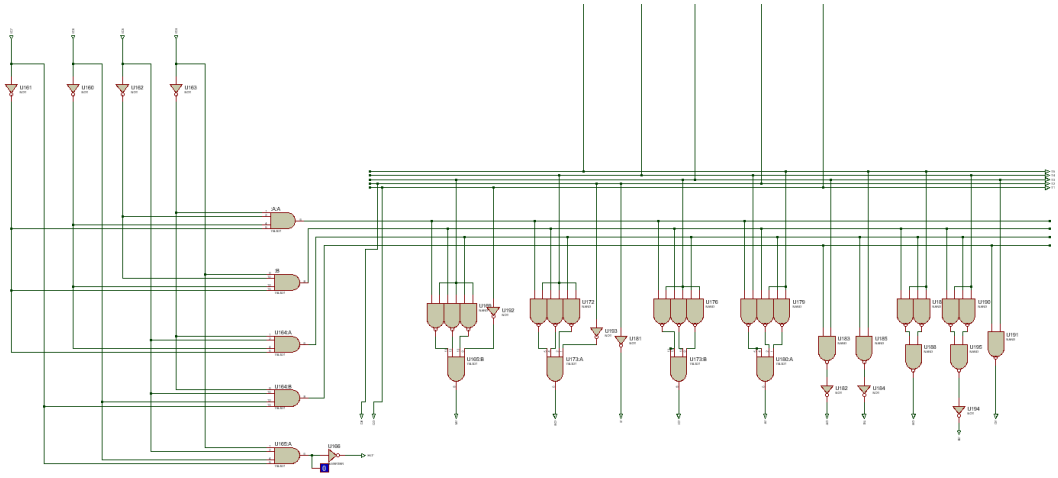
1. We have 5 T states in our SAP. So, we will need 5 Flip flops.
2. As we start in the reset condition first. So, the complemented output of the rightmost flip flop will be our first output in the ring counter.

3. Because it is fed to the “J” input of the next flip flop, so in the next clock pulse, using the master slave JK flip flop theory, the value will go the flip flop that is right next to it in the left direction. It is to be kept in mind, that the shifting of the values is going from right to left in our Controller.
4. Then when we have clock pulse transition, then the value will shift to the Flip Flop on the left. And thus, we complete shifting the value 5 times through each of the 5 flip flops.

RESET	CLK	Q0	Q1	Q2	Q3	Q4
0	0	1	0	0	0	0
	↑	0	1	0	0	0
		0	0	1	0	0
		0	0	0	1	0
		0	0	0	0	1
		1	0	0	0	0

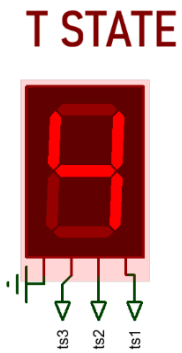
Algorithm

We must make combinational circuits so that we might get the required output. The combinational circuits are designed so that, when a specific T state is activated through the ring counter, then at that particular moment if either of the Macro is activated then it will give out certain output to the OUTPUT pins.



Display of T-States

Now, in our design we wanted to make a display to show the exact T state the SAP is in at a particular time, so that it is convenient for us to keep track. For this we will use the **7 seg BCD display**.



T states (Table)

T1	T2	T3	T4	T5	t3	t2	t1
1	0	0	0	0	0	0	1
0	1	0	0	0	0	1	0
0	0	1	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	0	0	1	1	0	1

Boolean Simplification

t3 = T1’.T2’.T3’.T4.T5’+ T1’.T2’.T3’.T4’.T5

= T1’.T2’.T3’ . (T4.T5’+ T4’.T5)

= (T1+T2+T3)’ . (T4⊕T5)

t2 = T1’.T2.T3’.T4’.T5’+ T1’.T2’.T3.T4’.T5’

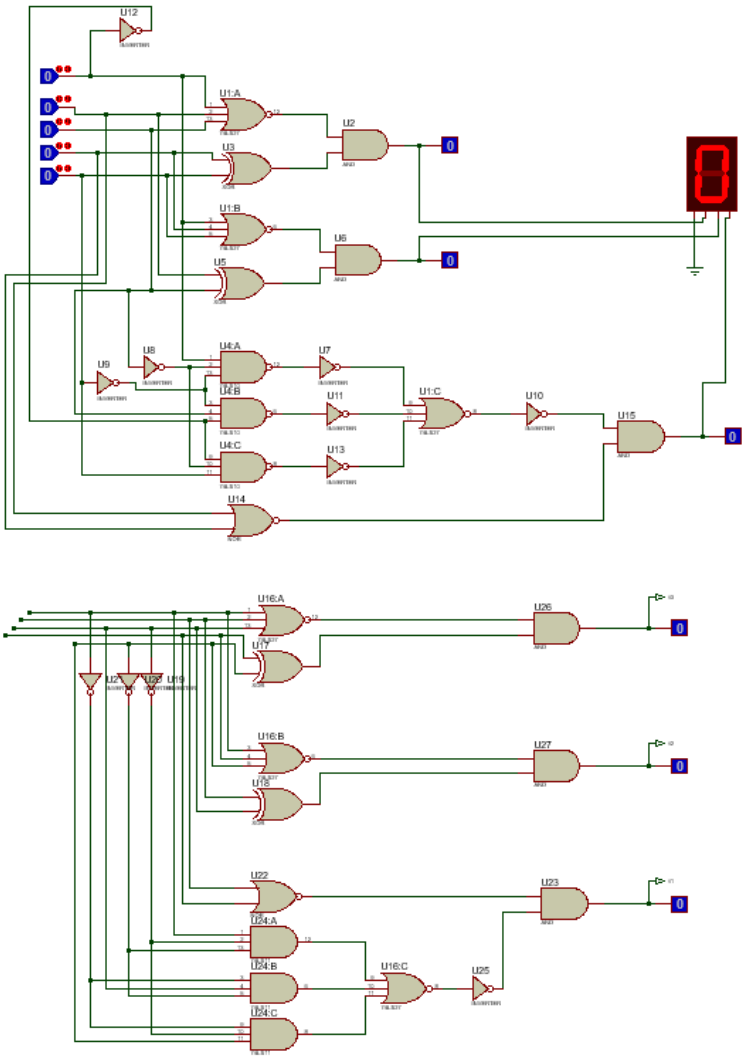
= T1’.T4’.T5’ . (T2.T3’+ T2’.T3)

= (T1+T4+T5)’ . (T2⊕T3)

t1 = T1.T2’.T3’.T4’.T5’+ T1’.T2’.T3.T4’.T5’+ T1’.T2’.T3’.T4’.T5

=T2’.T4’ . (T1.T3’.T5’+ T1’.T3.T5’+ T1’.T3’.T5)

Thus we make a combinational circuit using the above Boolean calculations. And we connect t1, t2 and t3 to the display. And the 4th pin of the display is to be grounded as we won’t be using it anyways.


U12
U1:A
U2
U3
U1:B
U6
U5
U4:A
U7
U9
U8
U4:B
U11
U1:C
U10
U15
U4:C
U13
U14
U16:A
U26
U17
U16:B
U27
U18
U22
U23
U24:A
U24:B
U16:C
U25
U24:C

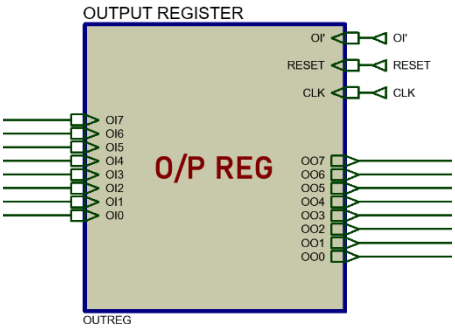
PHASE F

Output Register

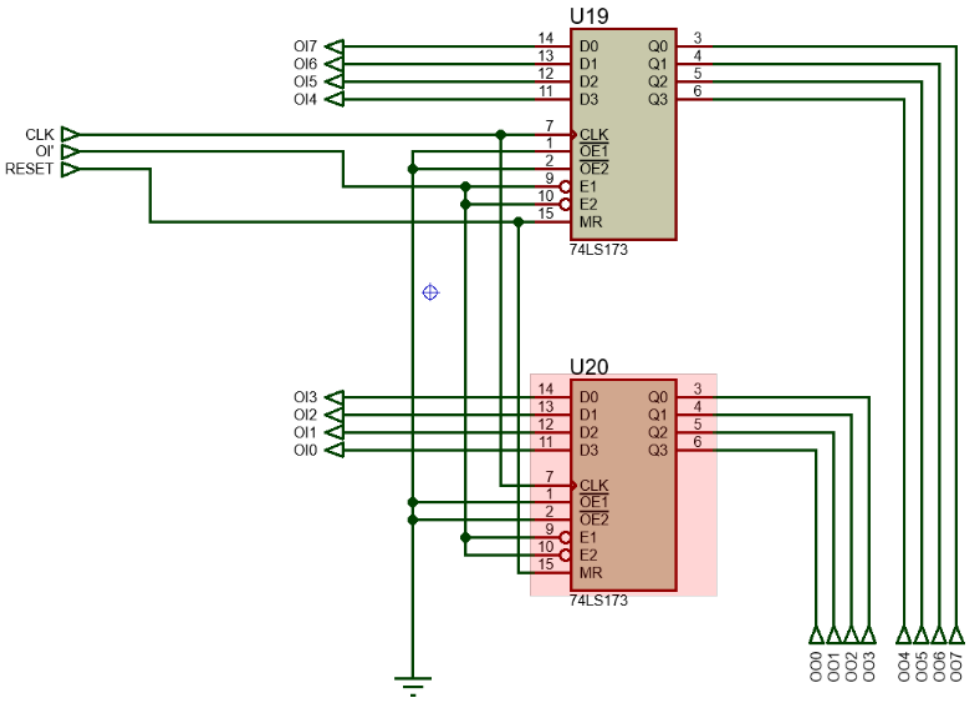
Block Diagram

Specifications:

IC	Input pins	O/P pins (ALU)	Control Pins
74LS173 4-BIT D-TYPE REGISTERS	OI7(MSB)-OI0(LSB)	OO7(MSB)-OO0(LSB)	OI', CLK, RESET



Algorithm of output register is the same as Register B



Binary To BCD Output Converter

To convert our binary output & show it in a seven segment display we need to convert 8-bit binary to BCD. For that purpose we used a technique called shift add 3 or double dabble.

The algorithm starts with the appropriate number of BCD 4-bit nibbles on the left (initialized to zero) and the binary value on the right. The whole binary representation (two 4-bit nibbles and a 6-bit

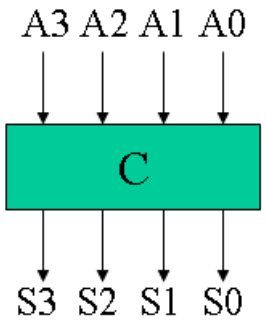
binary string in the present case, where 4210 i.e. 1010102 is the value to be converted) is regarded as a single shift register. Whenever shift left operations take place these have the effect of multiplying

the overall representation by 2 (hence the 'double' part of the name). If a prospective BCD digit is 5 or larger, then 3 is added before the next shift left. This is the 'dabble' part of the name — using the dictionary definition of this word as being: '... to make a small adjustment'.

Reference: Charles B. "Chuck" Falconer, "An Explanation of the Double-Dabble Bin-BCD Conversion Algorithm"

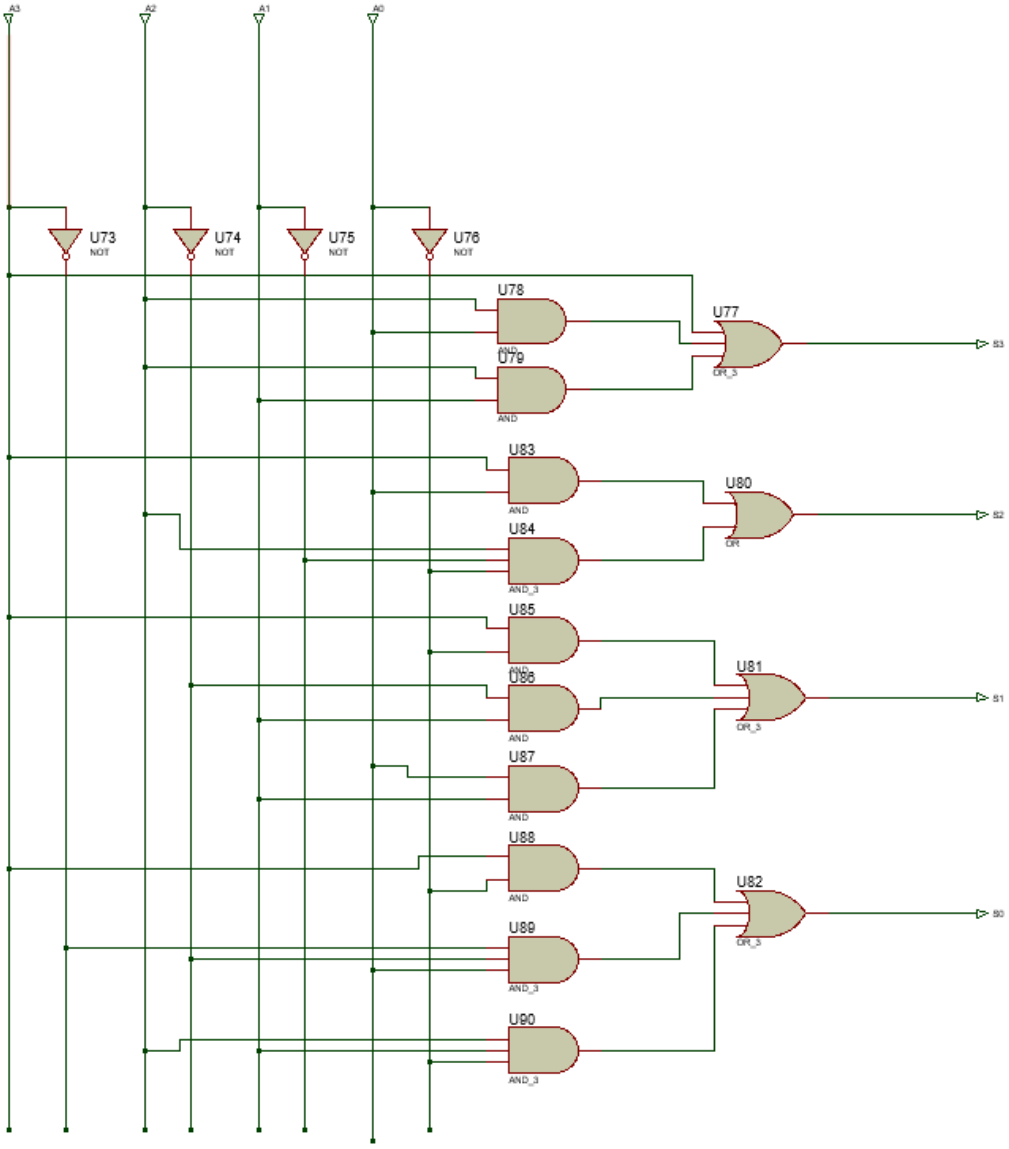
TENS	UNITS	BINARY	OPERATION
0000	0000	101010	Start
0000	0001	01010	Shift 1
0000	0010	1010	Shift 2
0000	0101	010	Shift 3
0000	1000	010	ADD-3 to UNITS
0001	0000	10	Shift 4
0010	0001	0	Shift 5

Truth table for Add-3 Module

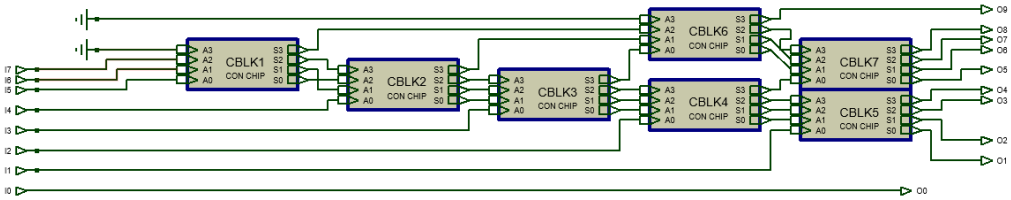


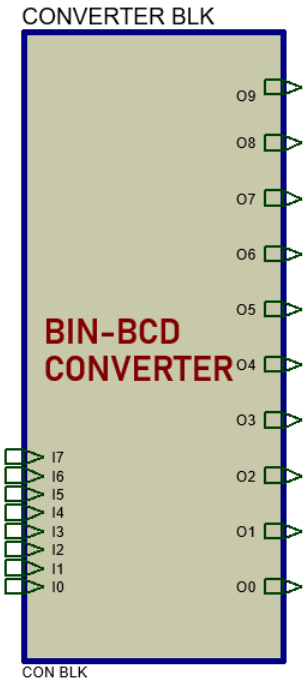
A3	A2	A1	A0	S3	S2	S1	S0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	0
0	0	1	1	0	0	1	1
0	1	0	0	0	1	0	0
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0
1	0	1	0	X	X	X	X
1	0	1	1	X	X	X	X
1	1	0	0	X	X	X	X
1	1	0	1	X	X	X	X
1	1	1	0	X	X	X	X
1	1	1	1	X	X	X	X

Combinational Circuit For Shift ADD-3 Module



Converter Circuit & Final Sub-Circuit



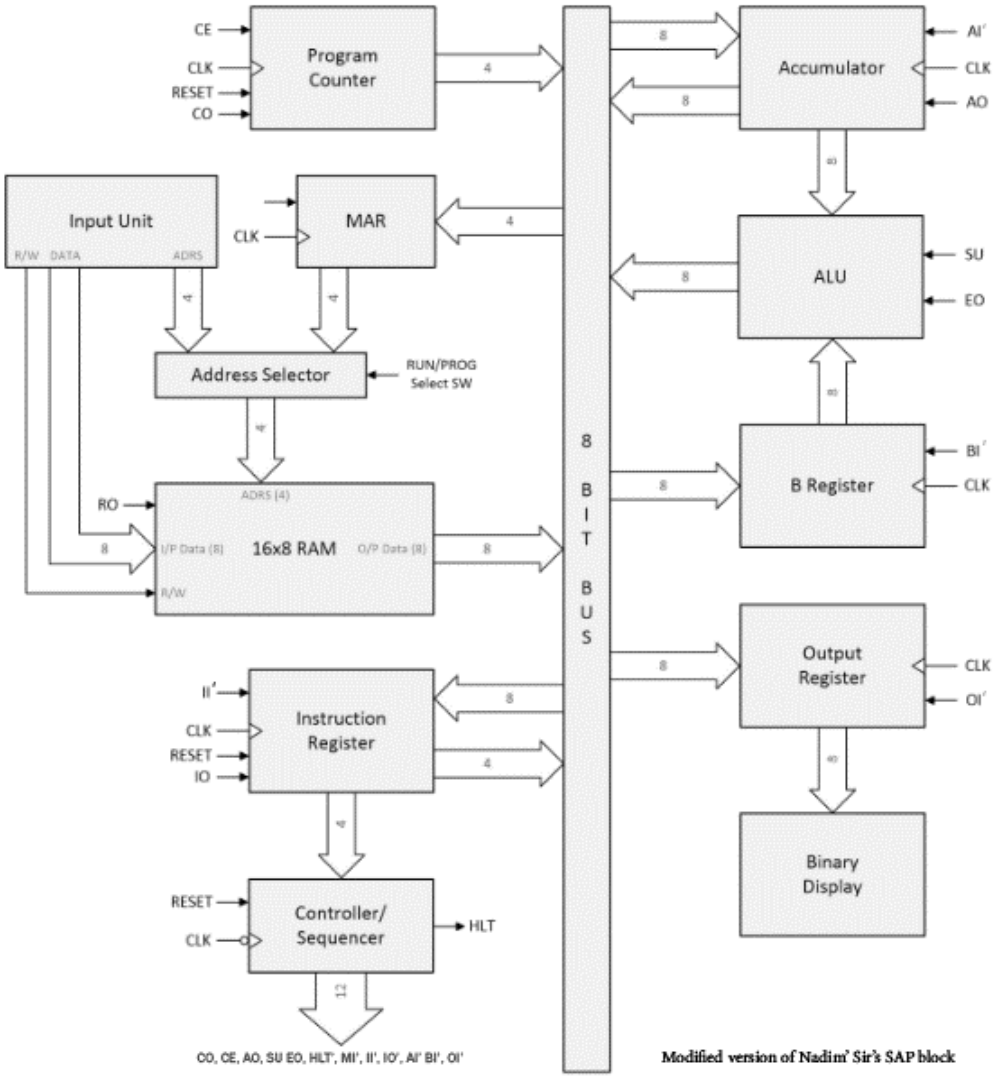


Algorithm

1. Pins I7-I0 will take inputs from the output register.
2. Inside the Bin-BCD converter block the conversion will take place using the double dabble algorithm.
3. The outputs will come out from pin O9-O0.
4. In the output there are 10 pins but, in the input, there are only two pins. It is because for 8bit binary input the highest number is 255. So, in the MSB of the output there is only three possible digits 0,1&2. And their corresponding binary numbers need only two digits for their representation.

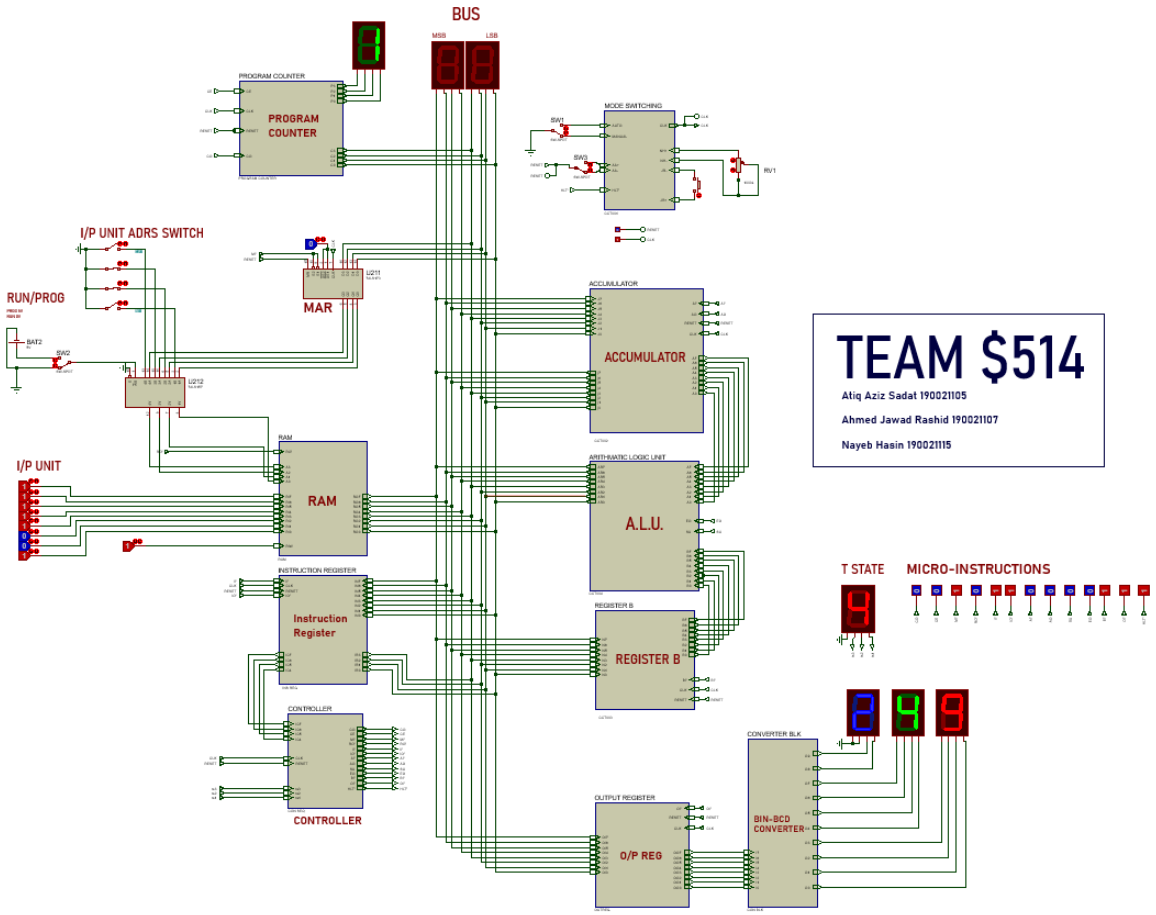
Complete SAP-1

Block Diagram



Modified version of Nadim' Sir's SAP block

Circuit Implementation using Proteus 8.12



Problems faced

Clock

1. **Clock Button and Potentiometer**: We wanted to create a sub-system for our clock module. But as it happened that the push button and the potentiometer was an internal component of the circuit. And there was a limitation in that regard. So, we made terminals at particular points and using inputs to the “Mode Switching” module that we made.
2. **Astable mode error in Proteus**: Because there is no initial value to the clock SPDT switch, so when we try to simulate the sap at any particular moment then error shows in Proteus.

Accumulator

1. When we ran accumulator separately then we couldn’t use the input and output pin names of upper case and lower case of the same alphabet. Otherwise, they were creating short circuit among themselves

Sequencer

1. We ran the sequencer using op codes that was created by us using required combinational circuits. Using those specific op codes, the sequencer was giving perfect values in isolation to the entire SAP. But when integrating Sequencer to the bus, we found logic contention issues in over 20 wires. And in that specific case we could only get to 4 T states. And we did not get outputs. And when we only programmed LDA we got output. So, as we were debugging the issue, we decided to revert back to another op code, which seemed to do the job at the time. And then we found out that the HLT’ was working as well, which was not functioning prior to changing the op codes.
2. We also faced difficulty in the creation of the combinational circuit of the display of T states. Because when we simulated it separately it was working but it didn’t give any result when we integrated it to the SAP. Then by changing some of the Boolean expressions we were able to execute our result.

RAM

1. The 16x8 RAM that we built for the SAP using 1 bit ram chip had some problem while running with the complete SAP. The proteus simulation has some problems because there are so many sub circuits and elements inside this RAM. This caused our software to crash and also, we faced a lot of bus contention while using this RAM

Discussions

Outcomes

By completing the project, we had a head start about how computers function in real life. In Digital electronics course 4307 we had an introductory familiarization with basic digital electronics, and by finishing this project we had an overview of how these concepts work in an integrated way. One of the significant outcomes of this project is how to use different IC's and read datasheets. Though we have completed the project online we had a brief overview of SMD components, PCB, and other core electronics concepts. As the computer is a versatile thing concept of other courses was also sharpened by this project.

Limitations

1. As this was an Online project, a lot of communication gaps between the teammates were present.
2. Hand-on experience about different IC's, boards, and electronics were lacking.
3. The software that was used in this project (PROTEUS) wasn't friendly while creating bigger circuits, as a result, we sometimes had to take the help of built-in IC's. Such as creating 16*8-bit Ram manually.
4. Sometimes circuits get misplaced in subcircuit mode due to similar subcircuit numbers.
5. Some built-in IC's in proteus don't give the proper output as expected. Thus, we had to use primitive versions of it or the ones without a PCB footprint.
6. Problems were faced when copying an IC in proteus, without copying one can use block copy for repetitive IC's.
7. Concepts like pullup and pulldown resistor, power distribution were left out of this project for simplicity though this was also included in this course.

Future Development:

1. We are working to do this project using Breadboard and PCB.
2. Research is going on to simplify the circuits further and reduce T-states.
3. We have a plan to upgrade our SAP-1 to Albert Paul Malvino designed SAP-2.
4. After upgrading our project, we will be writing a research paper about it to distribute the knowledge we have gathered.