Building My Own 8-bit Computer.

The idea of this project came to me when I watched a video on YouTube by BenEater and it got me interested, can I build the same thing as he did? And even if I could is the knowledge that I have is enough? So, I take it as a challenge and started to watch his video series and made sure to understand each concept he talked about and of course document it.

You should note that this computer won't be able to play games or display any images but it's capability will be to do some basic addition, subtraction, multiplication, and division, it may seem a little but once we understand these small concepts and how to design the logic that can do this then we can perform more complex problem by dividing them into smaller subproblems that a computer can perform easily.

The steps taken to build this project was as follow:

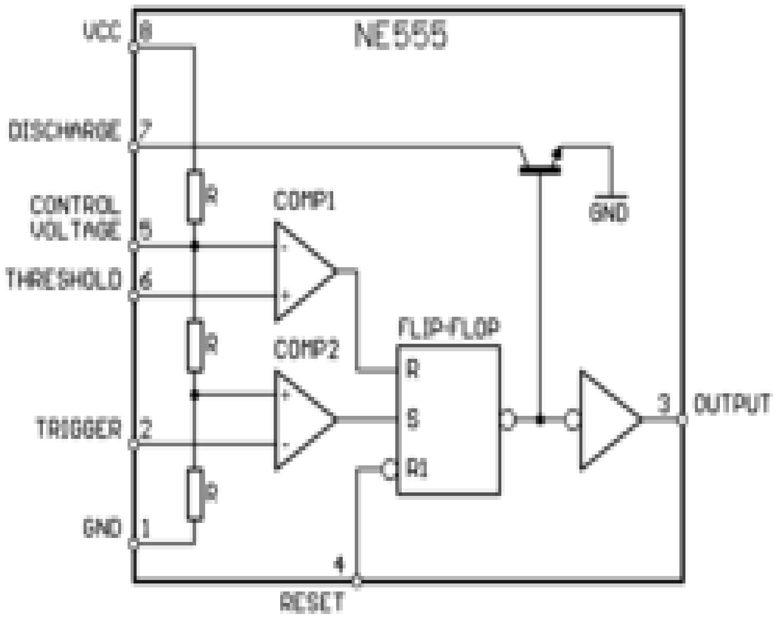
- Build the clock circuit based on N555 timer.
  - A-stable mode
  - Mono stable mode
  - Bi-stable mode
- Flip flops and what's inside them.
  - R-S flip flop
  - D latch
  - D-flip flop
- What is a bus and how it works?
  - How to design a bus connection.
  - How registers work and communicate to the bus
  - How to build a register
- ALUs
  - Subtraction in binary and the 2's complement method.
  - How adders works and how to design one.
  - How to re-design adders to do subtraction.
- Memories
  - The difference between static and dynamic memories
  - How a static memory looks like from the inside and how to build one.
  - Add Run and Edit modes to the ram.

- Build a 7-segment counter.
    - What is a J-K flip flop and master slave J-K flip flop
    - How to build a counter using master slave J-K flip flop
    - Convert binary reading to decimal on values on a seven segment.
        - Using logic and Gates
        - Using EPROMS
- CPU
    - Connect the bus.
    - Add indicators to each process.
    - How to program a computer to do something

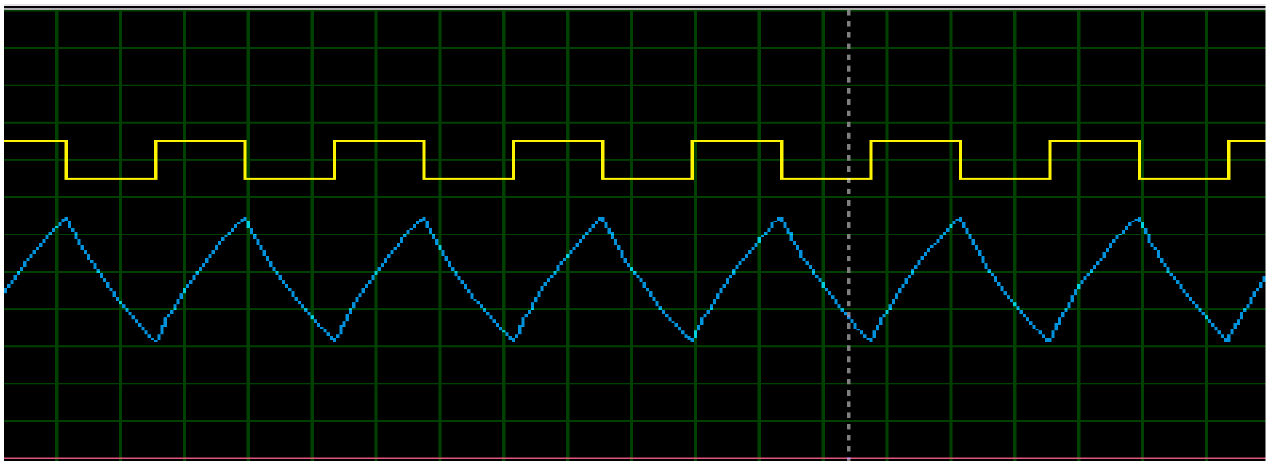
Clock circuit using N555 timer.

The 555 timer is one of the most popular ICs ever made and it is used in variety of timer, delay, pulse generation, and oscillator applications, in this part we will show how to use this IC to generate the needed pules signal and how to trouble shoot most of the common problems that may arise in A-stable mode.

If we looked at the internal circuit design of the 555 we will see that its consisted of two comparators , RS-flip flop and three voltage dividing resistors, and the idea is to make the output signal oscillate on and off, we need to change the supplied voltage across the comparator, so if the voltage on comparator 2 is higher that the threshold which equal to 1/3 VCC then the S pin will be high which drive the output pin high (Note the not gates), and the same with comparator 1, if the voltage is higher than 2/3 VCC then the reset pin will be high which mean the output should be low, this also mean that the transistor connected to the latch pin will turn on and connect pin 7 to the ground and her is where the 555 timer shine because if we used a configuration that cases a change of the voltage on pin 5 and 2 this will case the output signal to go high and low which is achieved as in the next circuit .



Here at the start the capacitor will be discharged which mean that pin 2 will be high casing the output of the flip flop to go high, and as current flow throw R2 the capacitor starts to charge up until the voltage on pin 2 is no longer smaller than threshold casing the output of comparator to go low, in the mean while the voltage on pin 6 is rising wish mean that at some point the output of the first connected to the reset pin of the flip flop will go high casing the led to go low and in the same time turning on the transistor which will case a discharge of the capacitor, the next figure will show the change of the output and the capacitor charging and discharging.



Now by controlling the values of R1, R2 and C we can change the pulse frequency and the formulas for the on and off time is given in the datasheet of N555 are:

- The output high time interval of each pulse is given by:

$$t_{\text{high}} = \ln(2) \cdot (R_1 + R_2) \cdot C$$

- The output low time interval of each pulse is given by:

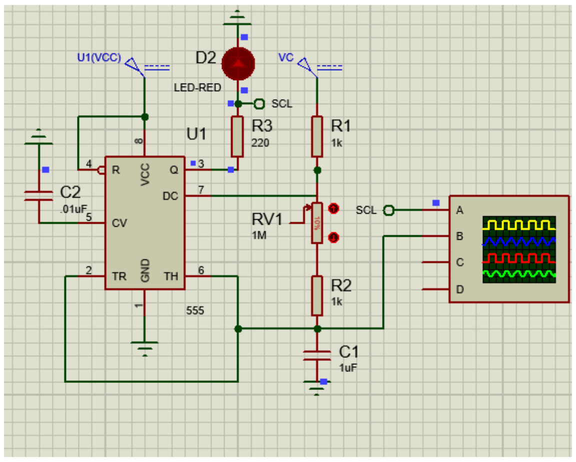
$$t_{\text{low}} = \ln(2) \cdot R_2 \cdot C$$

- Hence, the frequency f of the pulse is given by:

$$f = \frac{1}{t_{\text{high}} + t_{\text{low}}} = \frac{1}{\ln(2) \cdot (R_1 + 2R_2) \cdot C}$$

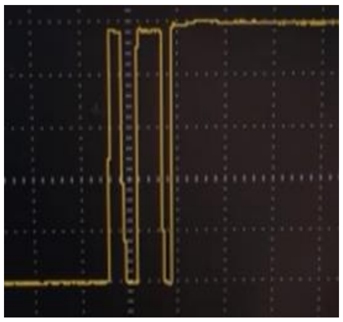
For the clock signal the high and low time should be as close as possible, and to able to control the timing we will change R2 to a variable 1MΩ potentiometer but If we are going to do that we need to add one extra 1KΩ resistor so that when the value of the potentiometer is 0 the capacitor doesn't discharge instantly.

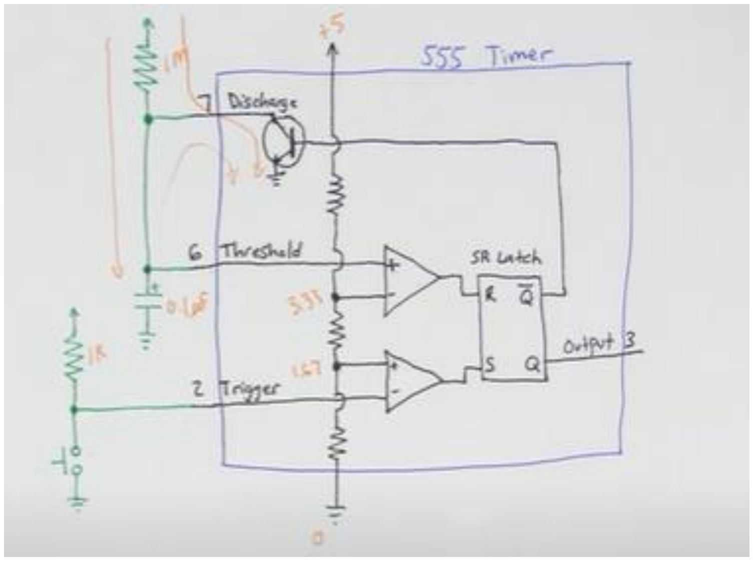
In the datasheet they also recommend adding to connect pin 4 (reset) to VCC and to connect a 0.01uF capacitor between pin 5 and VCC this to smooth out the transitions from one state to another at the nano second scale you should also add a capacitor on the power rails.

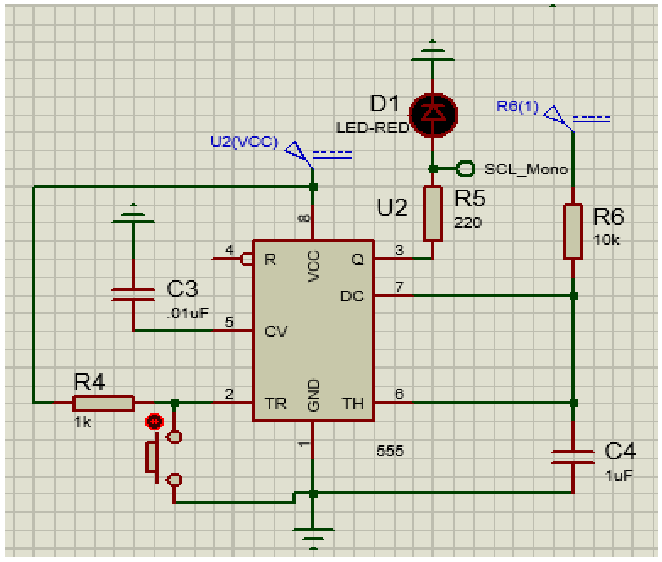


Mono Stable 555 Timer:

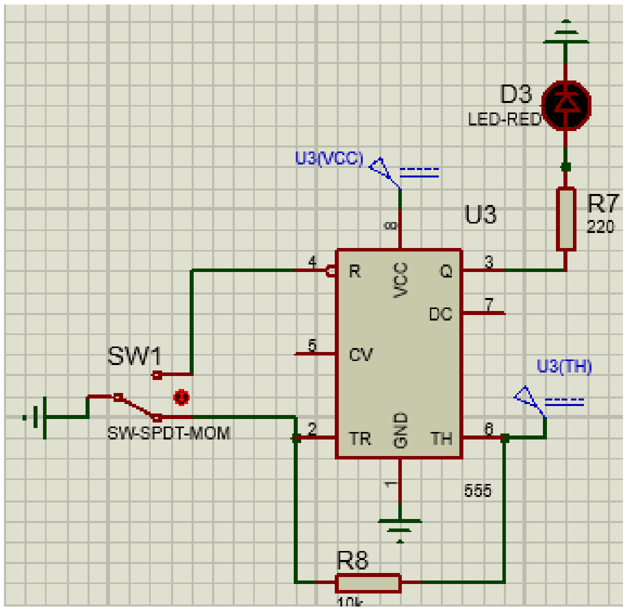
Sometimes we might want to trigger a signal manually and the answer might be to use a button and it would work but not perfectly, see the idea is that buttons are made of two plates and when these plates come to contact they conduct electricity but some times they start to bounce and this behavior may lead to triggering multiple signals so, to solve this problem we will need to use the 555 timer but in the mono stable mode, this configuration will case the output to go high only once and the delay of high time is controlled by a capacitor


1 bounce of metal plate in push button.

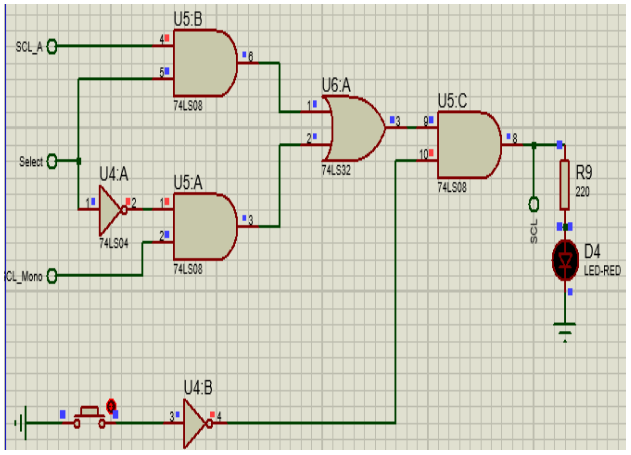




The last configuration that might need to use the 555 timer in is the bi stable mode, and in this mode the led will wither be on or off, this mode is exactly like a flip button but if we are going to use buttons then we need to work with the bouncing, so here we will use the 555 timer again to solve this problem but we only are interested in the flip flop inside it, and the idea is if we want the output high then we trigger the S pin and if we want it low we trigger the R pin and to access theses pins we will use the second differentiator trigger pin and pin 5 on the IC (reset pin) and note that its connected to a not gate.

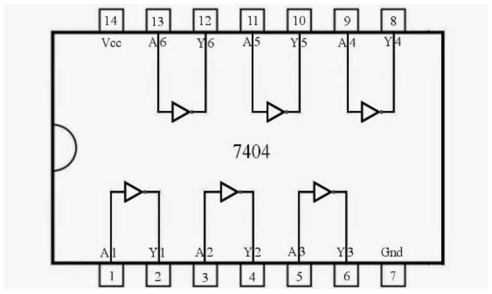


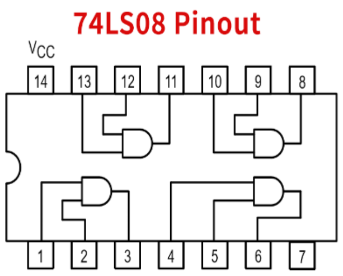
Once we finished with each stage now we need some logic to controle witch signal should pass throw the the output we also need to add a way for the computer to control these signals without the need of a mechanical switch, for this porpus we are going to use the next logic.

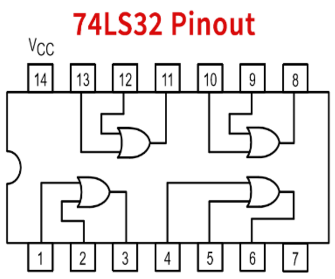


From the diagram the select input is the signal from the bistable 555 configuration and HLT sugnal will be the computer way to turn the clock on or off digitaly.

For the gates we will use the next ICs that contain these gates and they are as follow

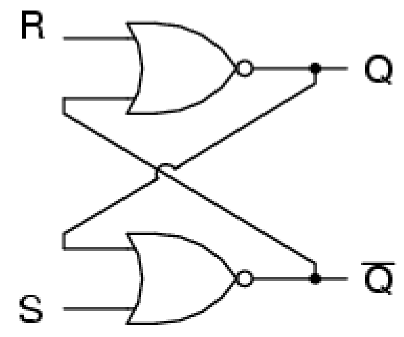




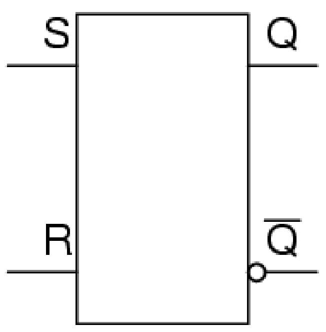


S-R latch circuit:

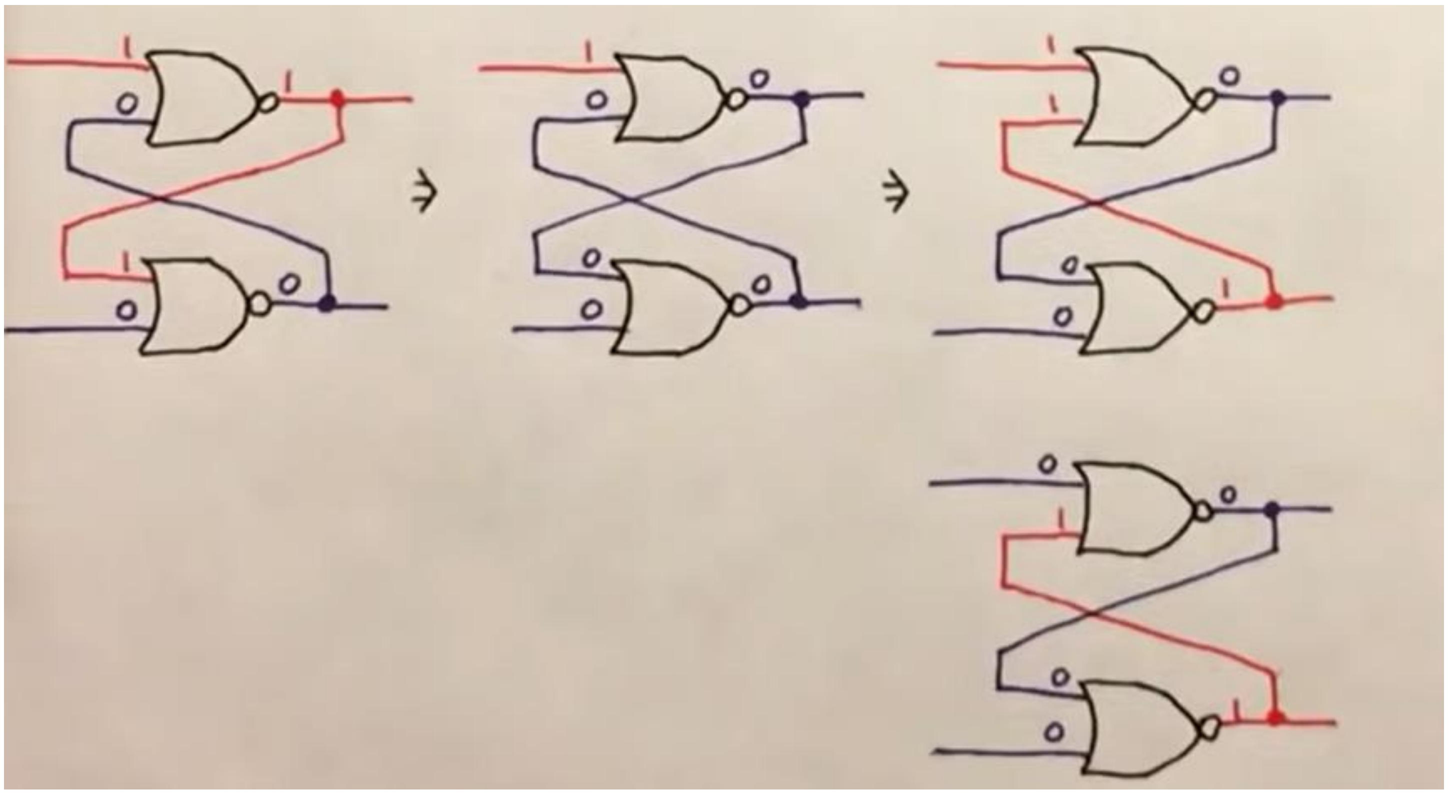
In this part we will show the diagram of S-R latch circuit and how it works, and basicly it's a compination of two NOR gates with the outputs of each one is used as an input for the other



S	R	Q	$\overline{Q}$
0	0	latch	latch
0	1	0	1
1	0	1	0
1	1	0	0

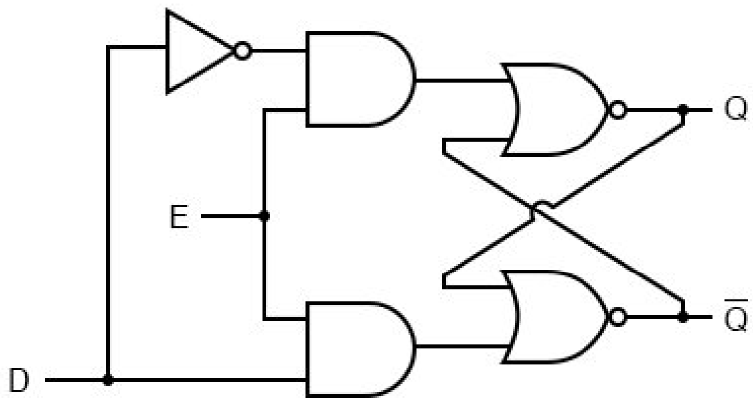


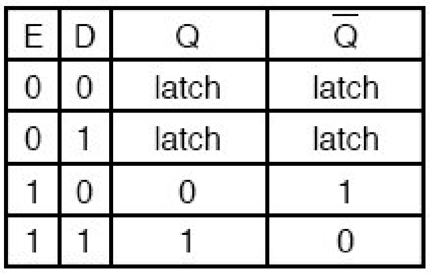
You can use an ic that with NOR gates to design this circuit like 74LS02, but there is some notes to consider the oeder of which pin will be high Q or $\bar{Q}$ is random it's a mater of wich one of the NOR gates will output a one first, and you can see the steps that a flip flop takes as folow:



D latch circuit:

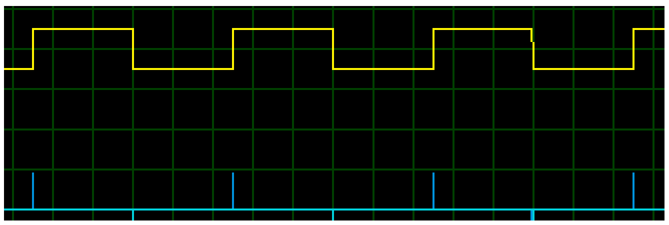
The idea of this crcuit is that we ant to control the filiping process with only one signal eather on or off, so this is what is a D-lactch circuit os for, but there is one extra pin we will add which is the enable pin, and basicly as long as this pin staies low then no matter input comes to the chip it will stay the same but if we set the enable pin to high then it will act as a S-R latch and here is the diagram for it:



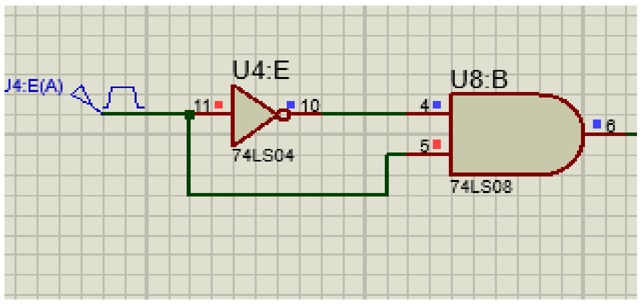


E	D	Q	$\overline{Q}$
0	0	latch	latch
0	1	latch	latch
1	0	0	1
1	1	1	0

Afer we finished the D latch circuit we want to to add a litle presigin to it in other words we want the enable signal to be on only at the rising edge of the clock signal, meaning we want to check the data either high or low only at the start of the rising edge of the clock signal, and the enable signal should take the folowing form

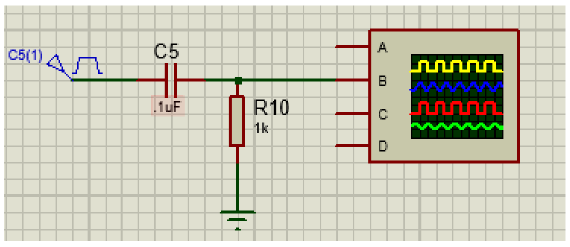


There are two ways to achive this behavior, the first approtche is to take advanige of the idea that gates are not perffect and they take some time to flip from one value to another, there for if we used the next logic: And by increasing the number of NOT gates we can add more time delay before the fall of the signal

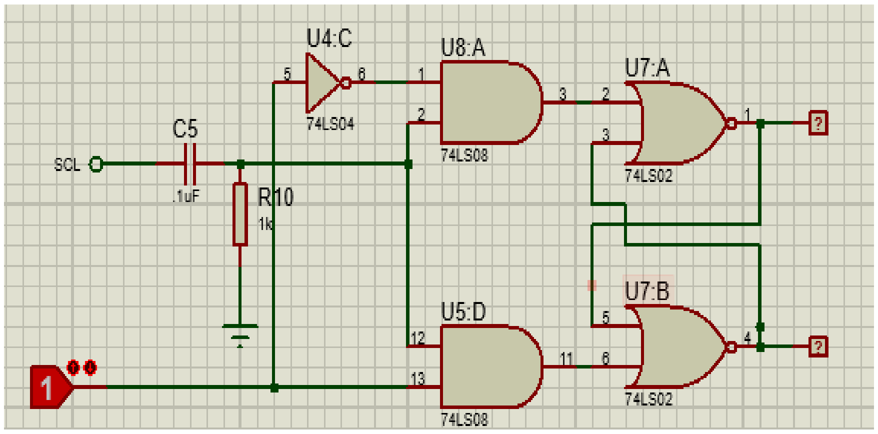


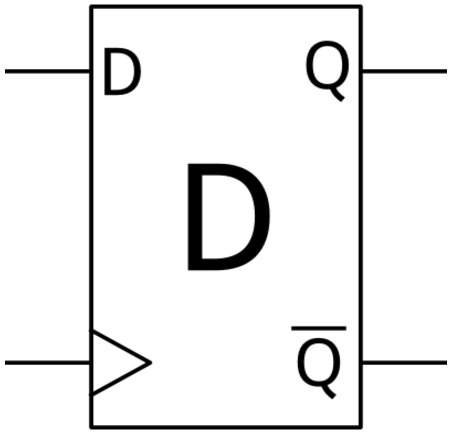
But this way is inefficient because it will deppend on the manufacturing of the gate and how ideal it is.

The other way is to breaty simple actuly all you need is a capacitor and a resistor and the delay time is $\tau = RC$ and the idea is the capacitor will case a lowering in the current flow signal until it becomes zero a ful charged.



If we coonected the delay circuit to a D latch circuit then we have a D flip flop configuration

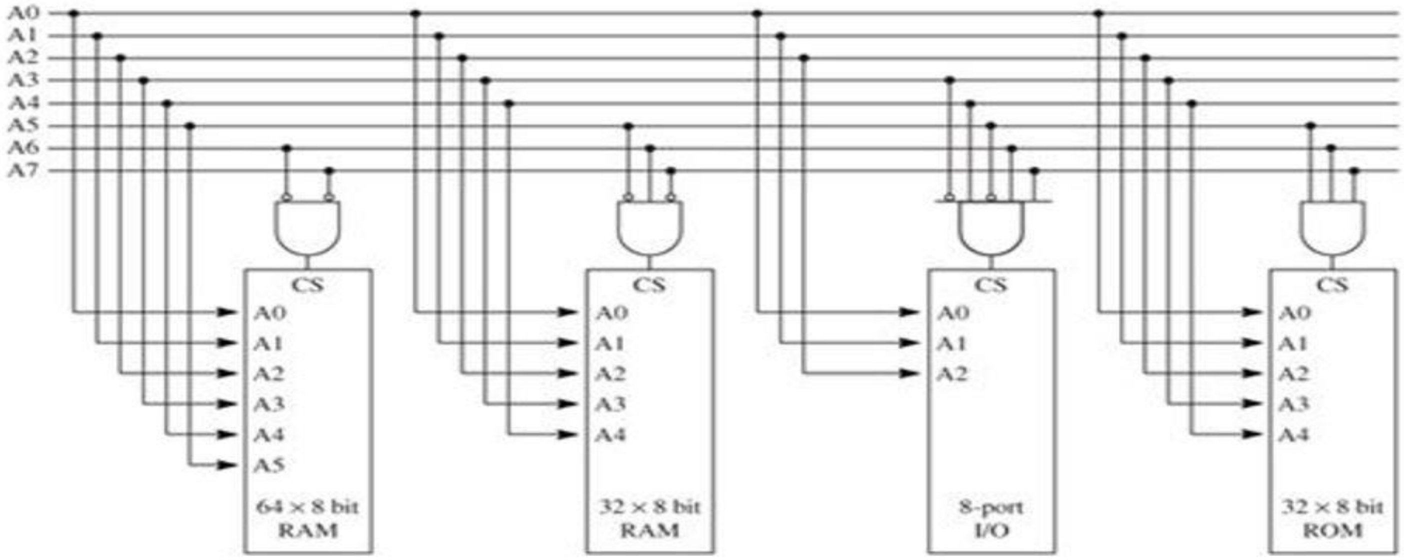




D-flip flop

What is a bus and how it works

A bus connection is colums of wires that conect multible stages of the circuit, each stage can load data to the wires (bus) so that the other stages can read it, so it's a way for register to read and upload data to each other, and these wires look like this:

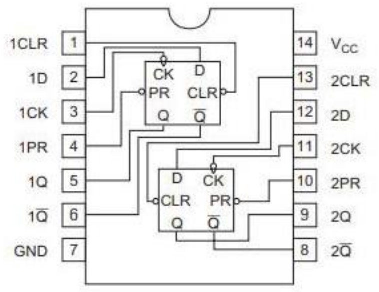


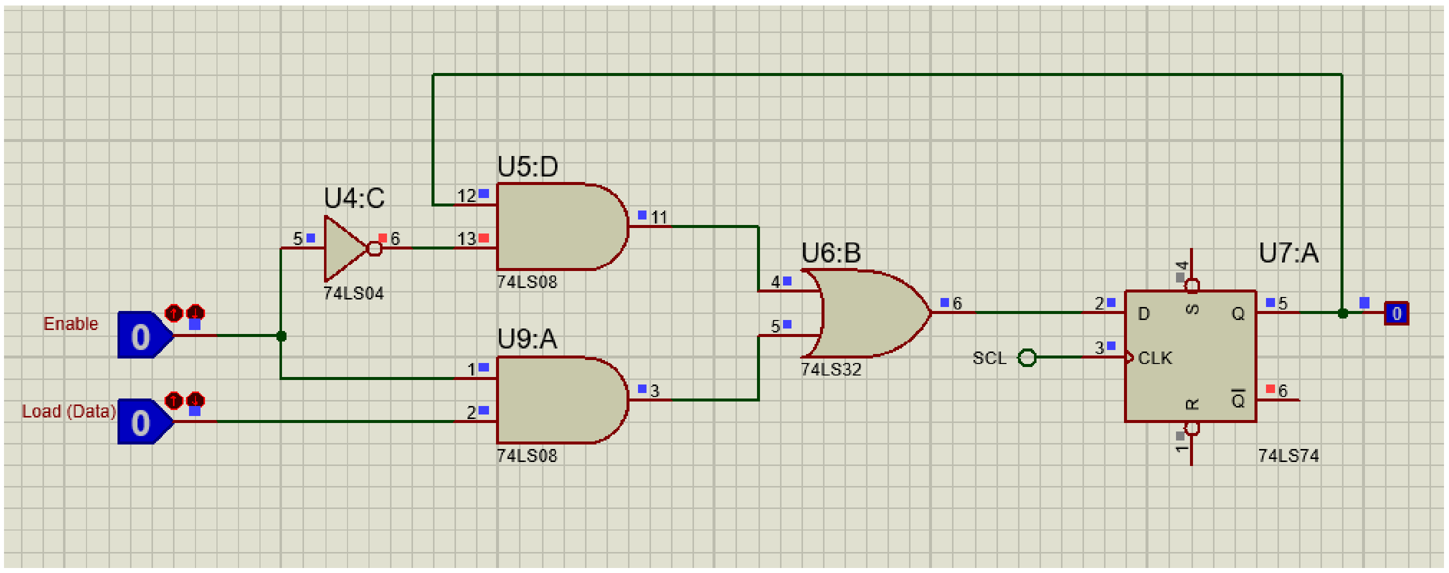
The following schematic shows an 8 bit address bus with 4 registers connected to it, each one can read or send data to another register

now we need to understand how a bus works and how to transfer data from one register to another, but first we need to understand how gates read 1 and 0, will if we applied high voltage to the input of a gate this mean that current will flow into it which represent a digital one and if the input voltage is low this will case the current to flow out of the gate which represent a digital zero, with this cleared out let's assume that we connected 4 register s to a bus and all of them has the same clock signal, what would happen if the one register outputted a one and another one outputted a zero to the same bus? also how we can prevent one register from over writing others when we don't want to? Will to solve the first problem we will need an enable pin that tells the register to read the data from the bus or not, in other words if we outputted some data to the bus and the enable pin was low the receiving register will simply ignore that data, so this is one problem solved.

The other problem can be solved by adding a buffer to the output of each stage, and its job will be to isolate the output of the stage until its said no so.

To understand this better let's build a register and see how we can ably this setup to it, and you can consider a register as a flip flop that store a bit inside it a one or zero and it's (74LS74) D-flip flop, and for the enabling trigger we will use the following configuration.

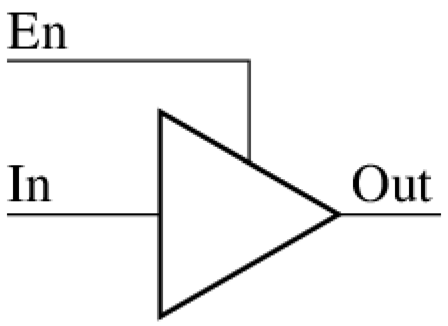


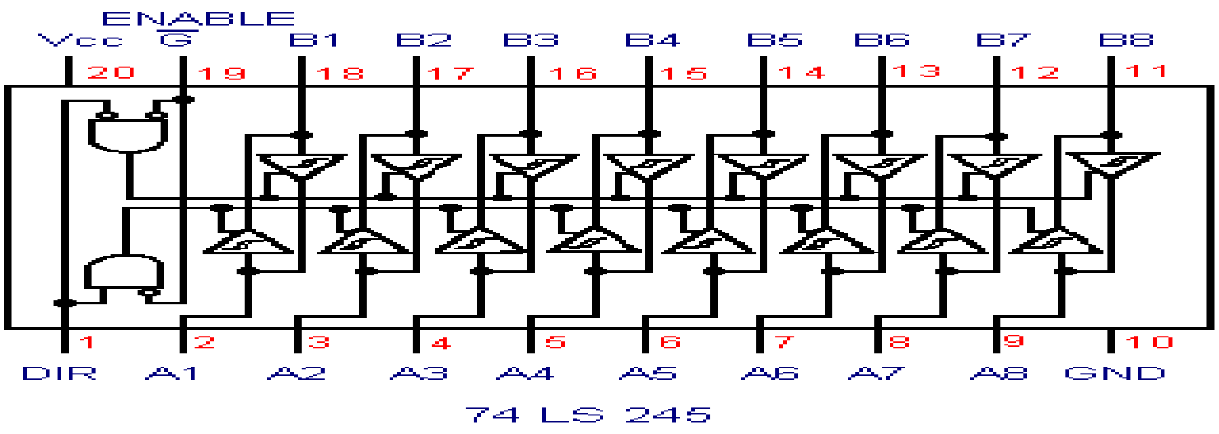


Adding the enable functionality.

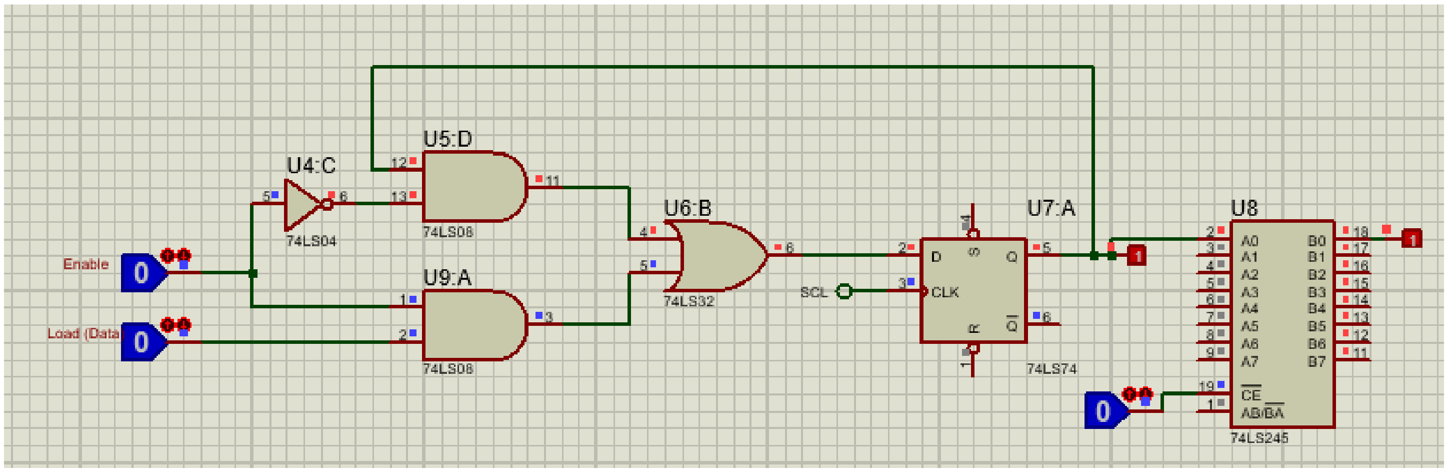
If we noted that this configuration need three inputs and they are: the clock signal to show when to read the data, the load pin which carry the data shared or transmitted throw the bus and the enable pin which tells whether to load the data or not, when the enable is high no data will pass .

The next part is to add a buffer the circuit, and a buffer is like a switch that can allow data (current) to flow or not depending on the Enable pin state, <u>if its high then it will not</u> allow it flow and vice versa, the ship we are going to use is called 74LS245 and it contain 16 buffer in it but we are only interested in using one of them.

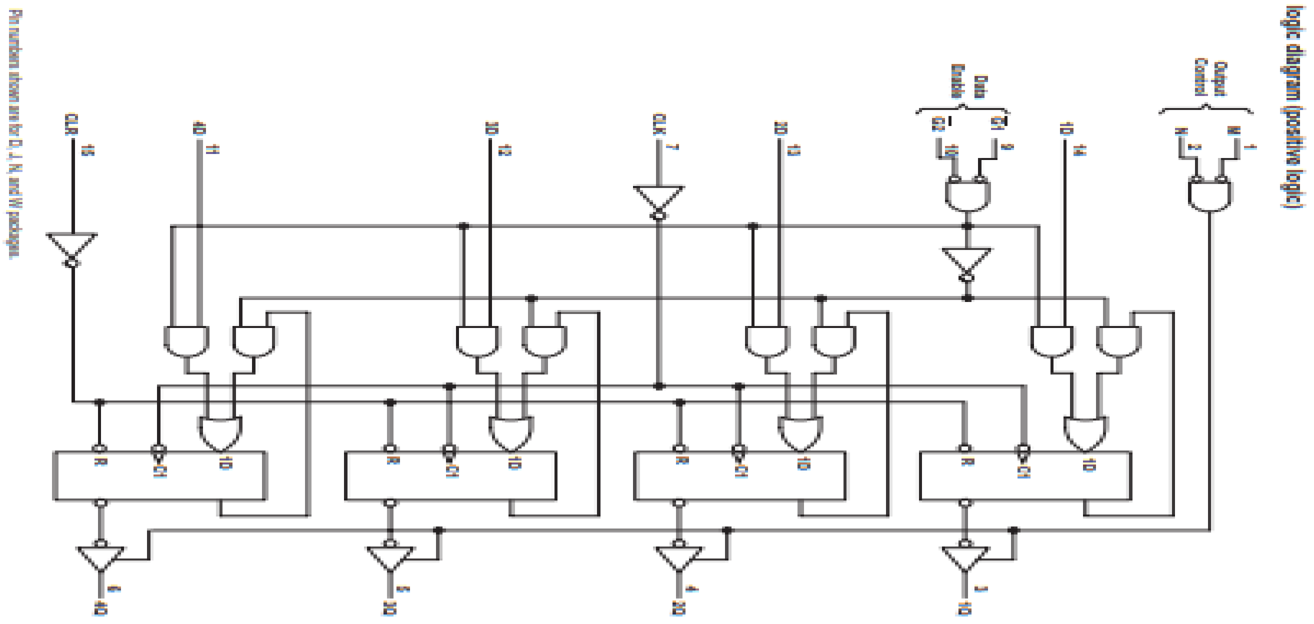




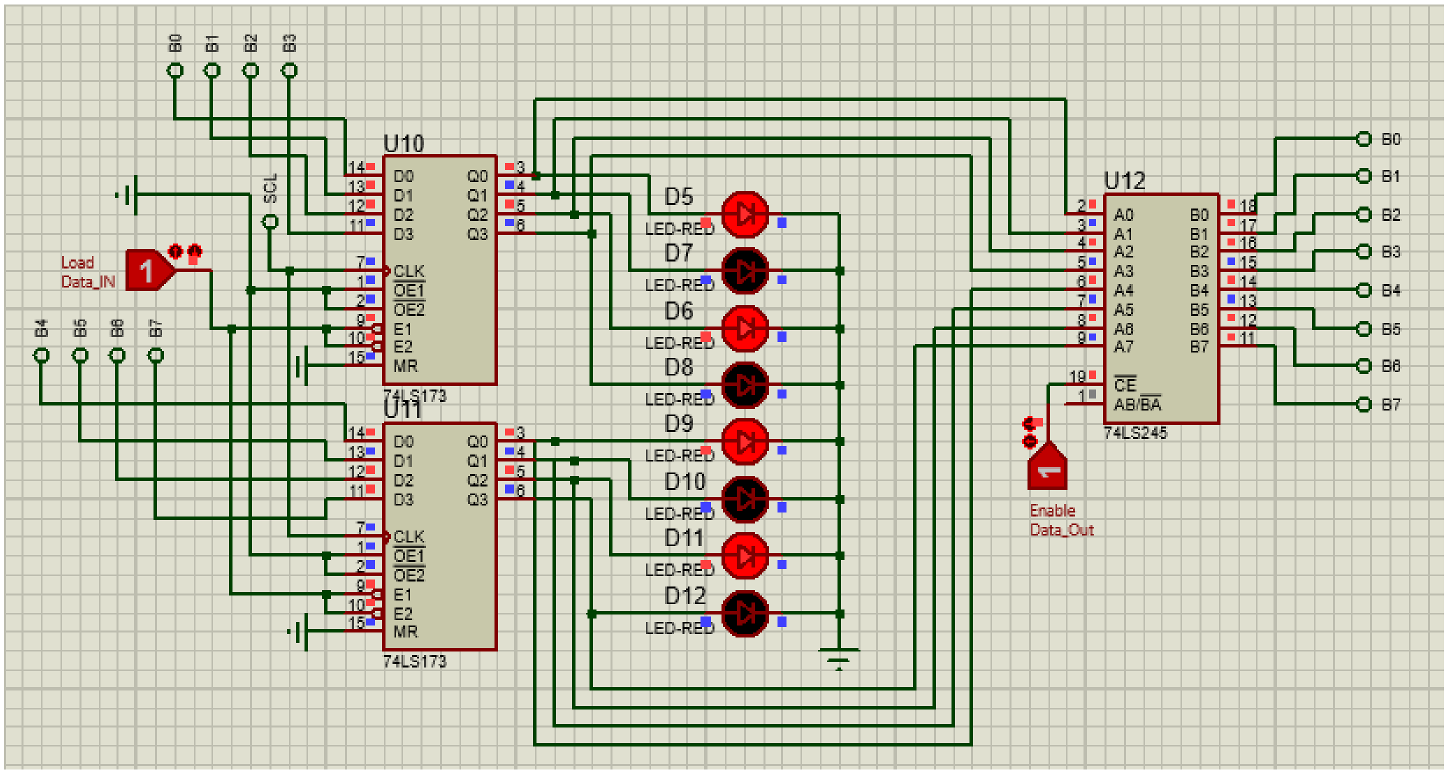
Now the final diagram of the register will like so:



Hence, we are going to build an 8 bit computer then we need 8 of this, and in simulation its easy as copy and paste but in real life it will be a real pain to do so, so what we can do is to use a ship that contain a pre-designed register on it like SN74LS173 4-bit register and its datasheet.

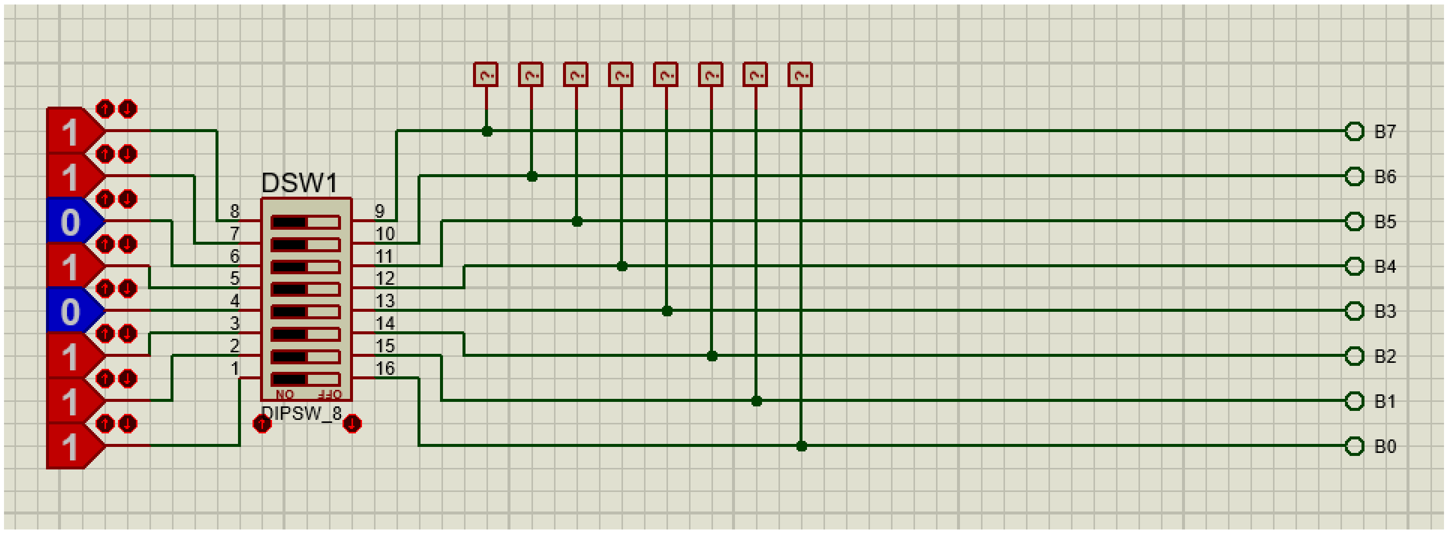

logic diagram (positive logic)
Pin numbers shown are for D, J, N, and W packages.
CLR
CLK
1D
2D
3D
4D
Data Enable
Output Control
1Q
2Q
3Q
4Q

The computer that we are build will contain three 8-bit registers connected to one bus, but hence the SN74LS173 is designed in such way that you can't see what is stored inside it, so we will add one extra stage (buffer) just to see what is inside it, but in general you don't need to do this step.



8-Bit register

After configuring the circuit as shown by connecting the output and input signal to the bus lines we should add a kind of switches to separate between the value we are generating and the value coming out of the register like so:



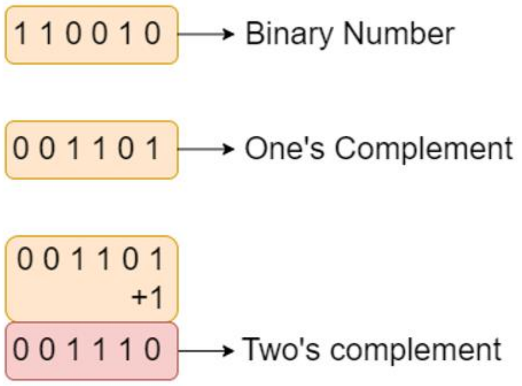
Note after sending the signal you should turn off the switches.

Negative number in Binary and the twos complement:

To add a sign to the binary number we use the last bit of the register to indicate its sign so if the last bit is a one then this number is negative in decimal and if the last bit is zero then its a positive number in decimal this mean that we have n-1 bits to store our values to it, but how to subtract in binary.

Will unlike addition subtraction in binary requires more steps and there is lots of ways to do so but what we are going to use is the twos complement method, and in this method:

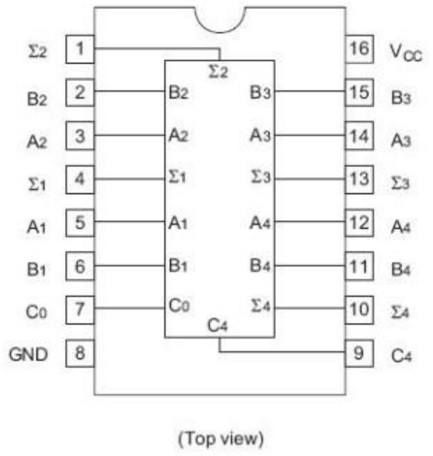
1. Represent the number in decimal.
2. Get the complement of the number 0 ➞ 1
3. Add 1 to the complement.
4. Do the subtraction.



ALUs and how they work?

ALUs stands for Arithmetic Logic Unit and basically you can consider it a place where we preformed some actions on the coming signals and generating an output this output might be addition or subtractions or any process that can be done on the inputted signals.

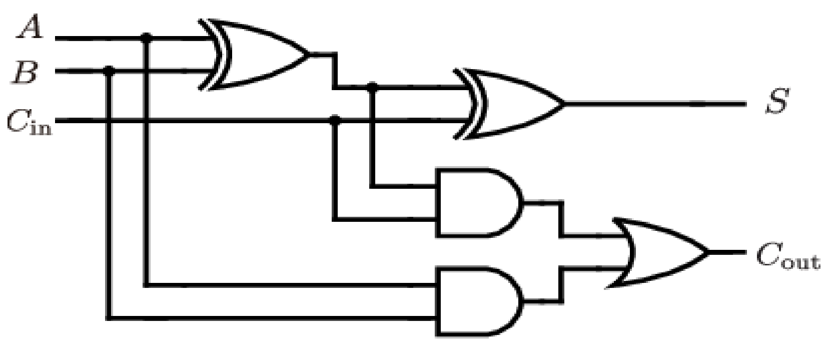
In this part we will use ALUs to do basic addition and subtraction with the help of 74LS283 which is a 4-biy adder IC, and also, we will need to know a little bit about XOR gates and how they work.



How Adders Works?

the adder logic is actually simple, and to understand it lets see all the cases of summation in binary which is shown by the next truth table and if you noted if we want to add two number and we don’t have a carry in the setup can be achieved by a combination of XOR and AND gate and this configuration is known as half adder the XOR gate will take care of the first column while the AND gate carry out the second column or the carry bit, but if the input signal comes with a carry in bit then we need to add an extra XOR gate to handle the first row, and another

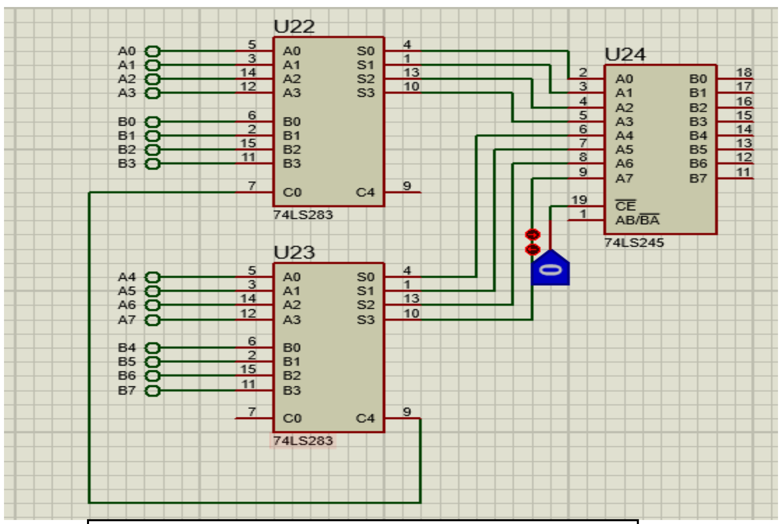
AND gate to handle the carry out bit, IF this seems difficult I recommend to watch this video that shows how adders work also made by ben and you can find it here.



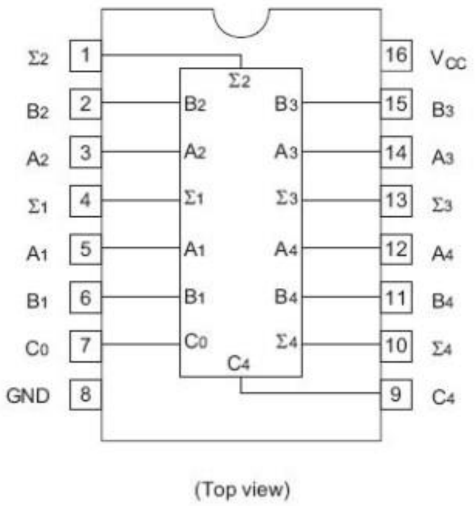
Inputs			Outputs	
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

The usual is to use a group of adders to make a 4-bit adder, 8-bit or larger, but for simplicity we can use a pre-designed one.

Now if we looked at the 74LS283 4-bit adder IC which has the shown diagram we will note that it has A1, A2 A3 and A4 as the pins for the first input and B1, B2, B3 and B4 as the pins for the second input, $\Sigma 1$, $\Sigma 2$, $\Sigma 3$ and $\Sigma 4$ which are the summation output pins, but we are really interested at is the carry in, carry out pins because this chip is capable of 4-bit addition while we are building an 8-bit computer so what we can do is cascading two ICs and using the carty out of the first adder as the carry in for the second adder and by so we can create an 8-bit adder, but want to add an enabling pin to tell the adder to output the results to the bus when we wants so as we did before we ‘ll add a buffer, and the final diagram of the buffer should be like this:

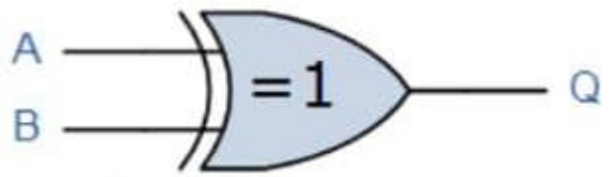


8-bit adder with an enable pin



74LS283 4-bit adder

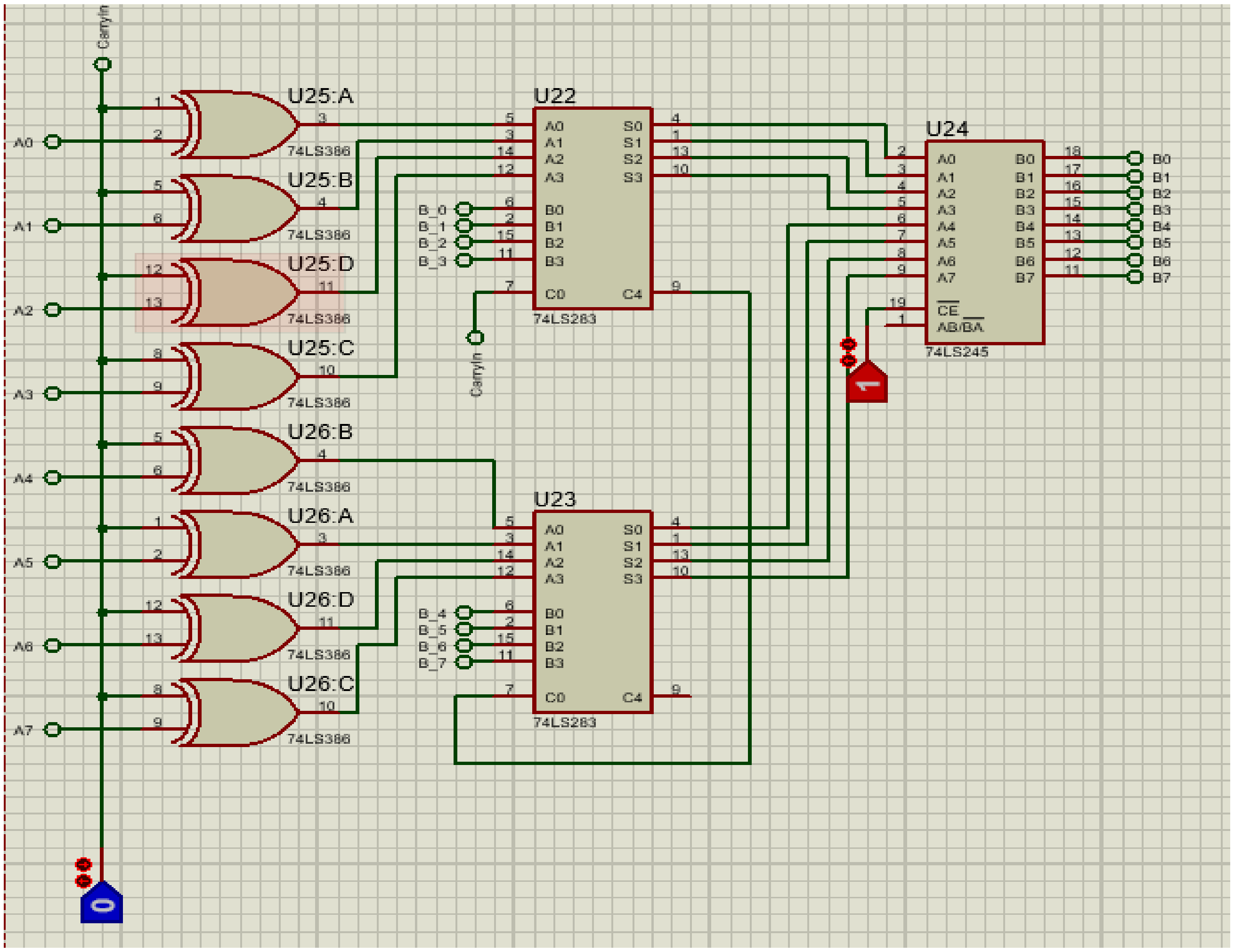
But what about subtraction, will to do subtraction you first need to understand the 2’s complement discussed in the last section, and the idea will be how to configure our adder to get the complement of the number, as we said before that the last bit of the binary number will indicate its sign 1➞ negative and 0 ➞ positive so the logic we should build will be as follow: if the last bit in A is a one then get the complement of the bit B, then after we finish add a one to this, the thing is the first part is the same logic of an XOR gate that if A is 1 then the value of B will be converted to the complement and if A is 0 then the value of B will stay the same, and this is what we want, thus the remaining part will be to add the one to the final step this also can be done as follow, if the last bit of the number is a one this mean that it’s a negative one so we till the adder to carry in a one and add to the input signal.



2-input Ex-OR Gate

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

Using this concept, we can design circuit that works as both adder and subtractor depending on the last bit of the number and it will be as follows:



Note in this configuration we used a switch to indicate the type of process we want to do, whether addition or subtraction and of course if you want you can use the last bit of the byte.

How to Design 128-bit Ram?

There are two types of Ram in term of how to store data, the first one a static Ram and other one is a dynamic Ram so what is the difference? Will a static Ram uses 1-bit registers to store its bit which means that the data will be stored inside it until we cut the power from the circuit, but the disadvantage is that registers as we saw requires lots of components to build it so a it will be too expensive to design a 16Gbit static Ram there for we use dynamic Rams to solve the cost problem that's because capacitors to represent the data inside it, so if the capacitor is charged this means that the stored bit is one otherwise it means zero, but the problem with this idea is that over time the capacitors will lose its charge do to some leak or something else, so we need to add an extra stage to recharge the capacitors which are holding ones inside them until we power the computer off, and hence this is a small 8-bit computer building a static memory won't be that expensive.

What’s inside a static memory and how to build one?

First let’s recall how a 1-bit register works: will after we connect the clock signal the input data can pass to the flip flop and overwrite the existing value only if the load pin is enabled and it can also send this data out, so if we connected 8-regestors to the same Row and connected all there Enable pins to one line we can trigger this line to release the data of the 8 registers once, and we can stack as many Rows as we ant and the final shape of this ram will look more like a matrix of registers and we will call each row a WORD and each column will represent a bit so a 16 word by 8 column static ram should contain a 64 registers and this is how it looks like:

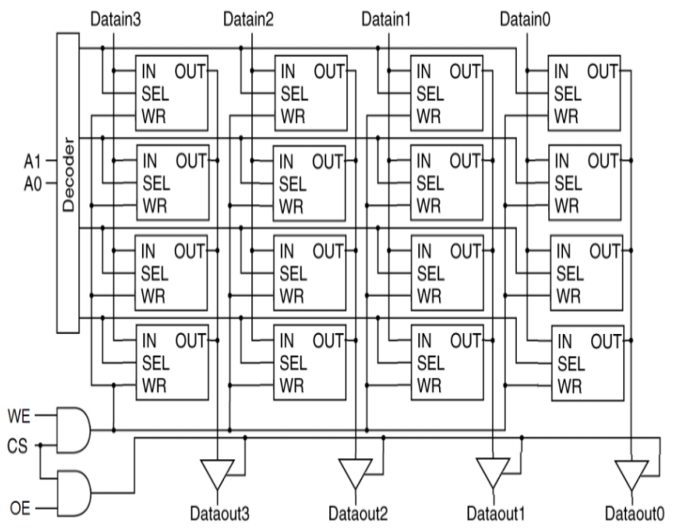
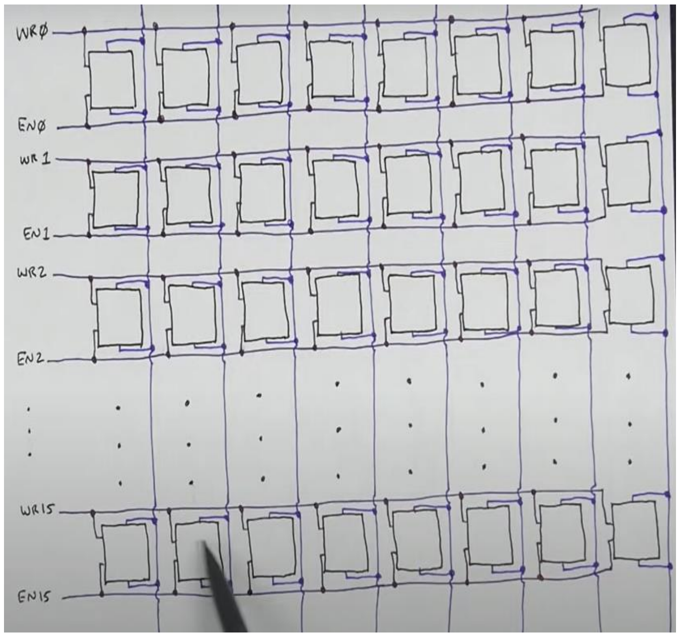
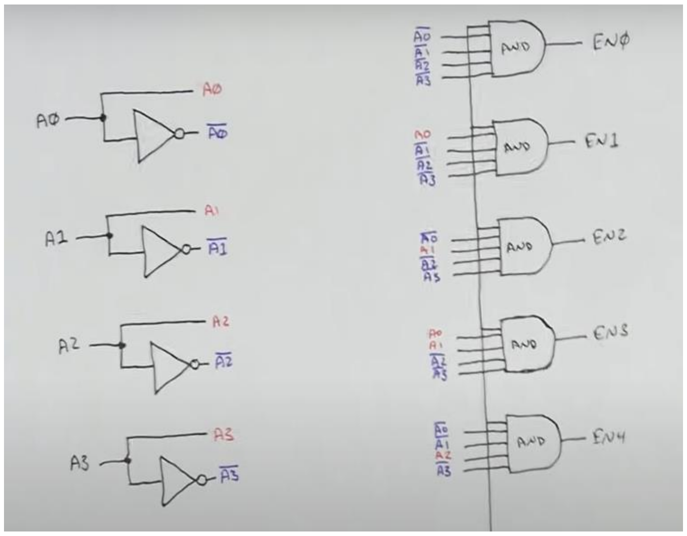


Figure 16. Internal structure of a 4x4 static RAM

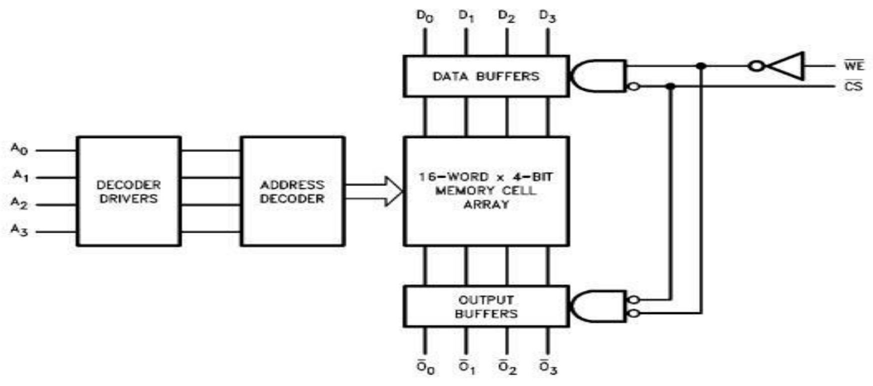


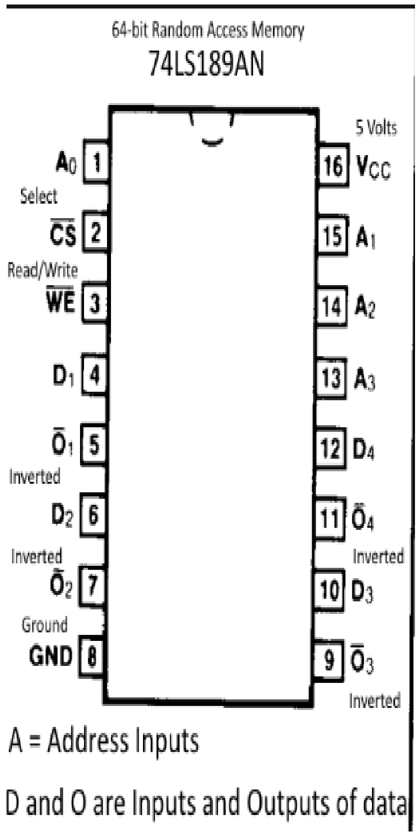
After we know the building unit of the static ram, the remaining question is how to program it? This will be done by first adding a way to select the row we want to store the data in then we set the load pin low and load the data into the row, and for our 16-word static ram, it would require a 4-bit address with a different combination of ones and zeros to control the 16 row and the way we will use to represent whether we want the 1st or the 2nd or the <u>nth bit will be in binary</u> like shown in the next figure.

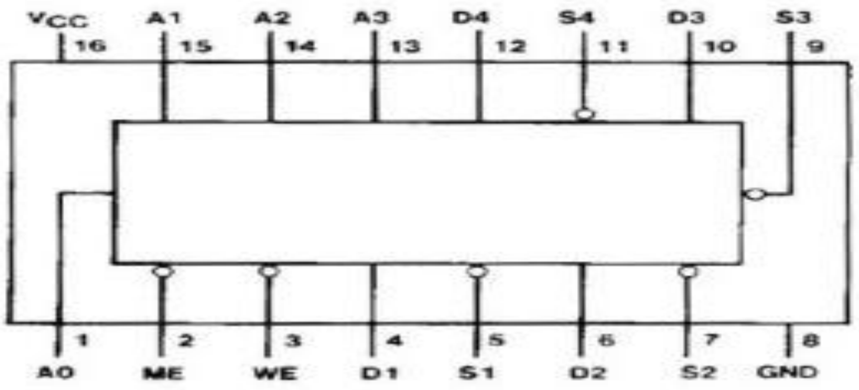


The ram we are going to build will be made by the 74ls189 which is a 64-bit (16 word x 4-bit) static ram so we will need to use 2 of them to get a total 128-bit static ram, also note that this chip has its output inverted so you will need to add one extra stage to get the complement of the outputs, for this we can use 74ls04 NOT gate chip.

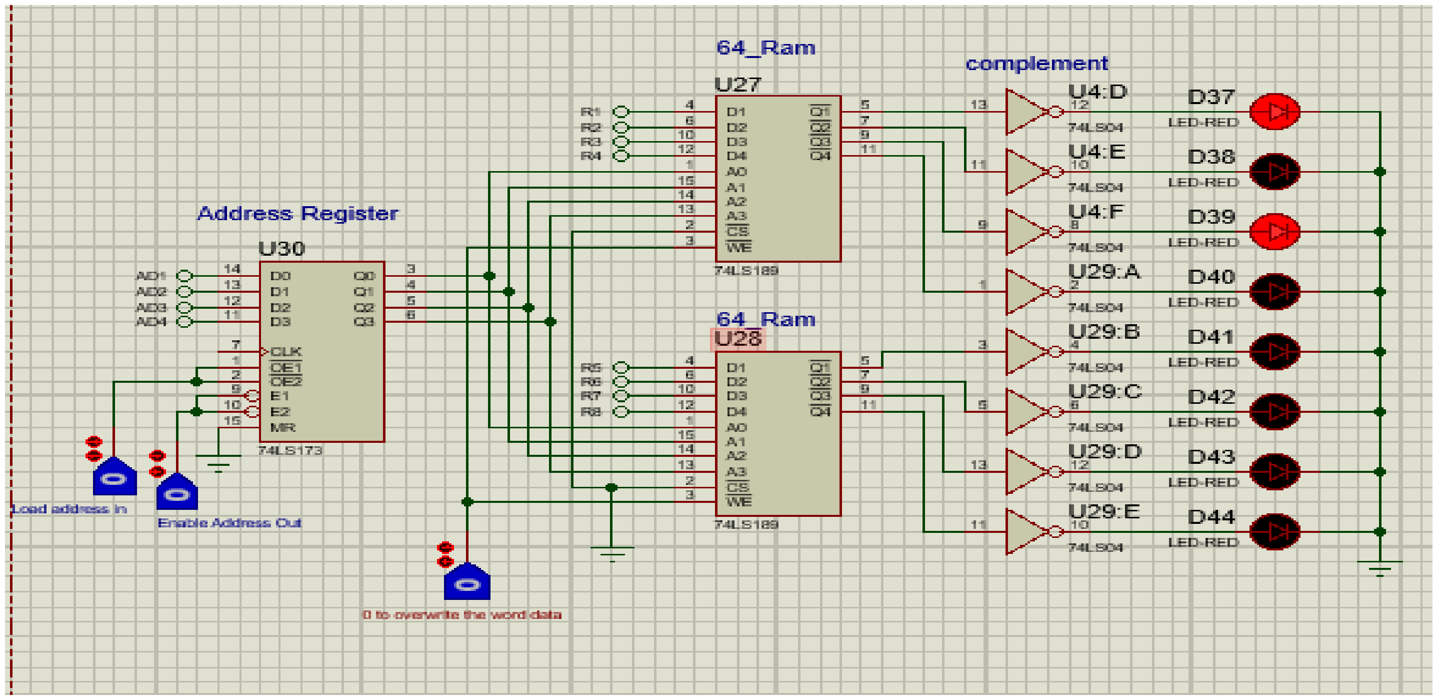
And you can fined all the pin layout, internal logic for the chip for better understanding.



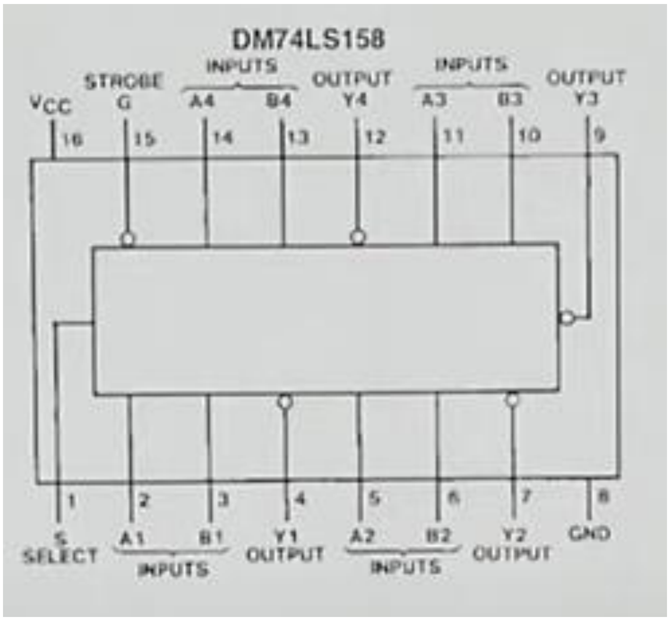


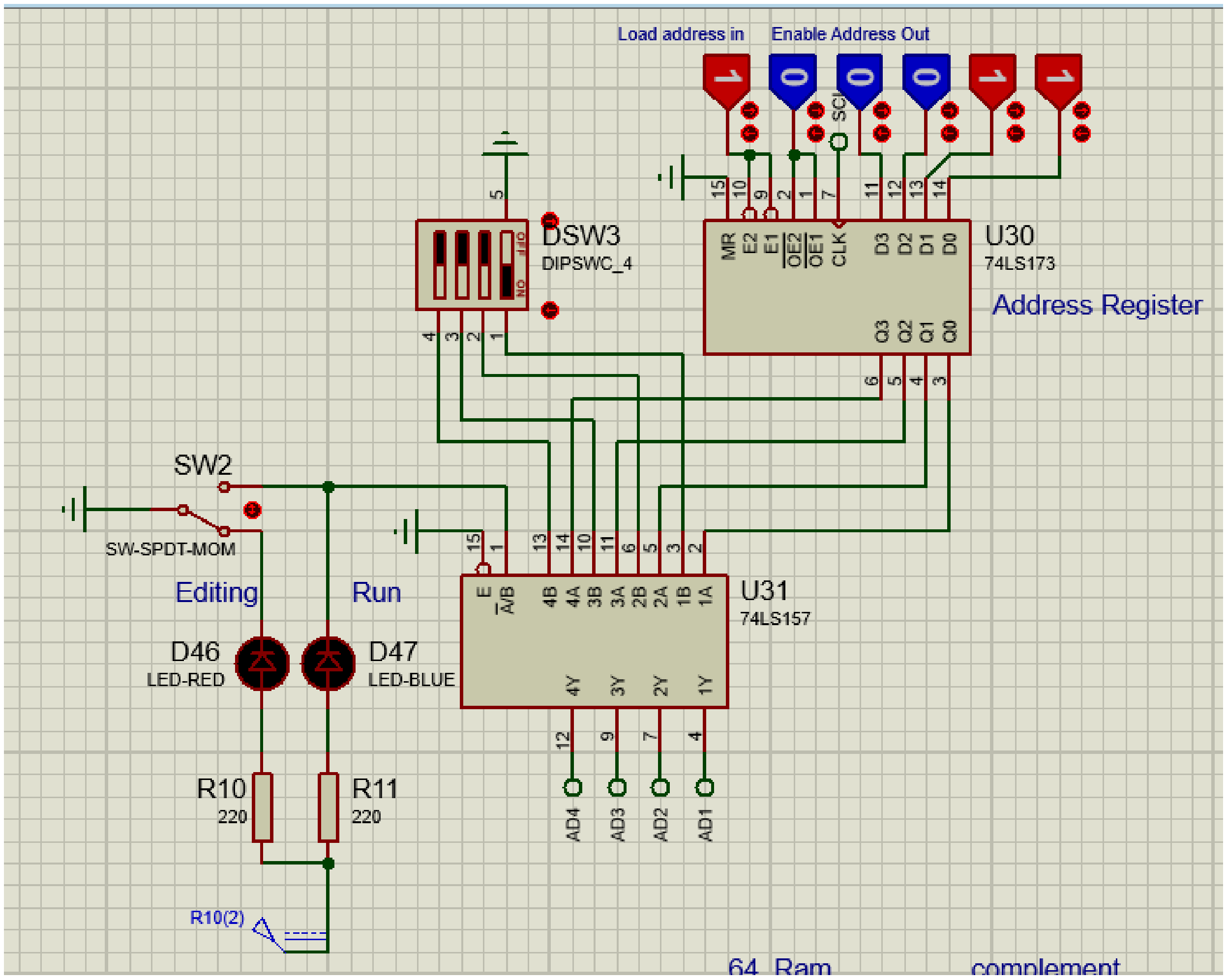


We will use an extra 8 bit register to transfer the bits from the bus to the ram, note also that pin 2 on the ram chip is the chip select pin it basically enable and disable this chip so we will connect it to GND as we always want the chips to be enabled, the next thing to do is to connect the address pins of the two chips together and the last thing is that pin 3 should also be connected for both chips and this pin is used to allow the data in and the final schematic will be as follows:

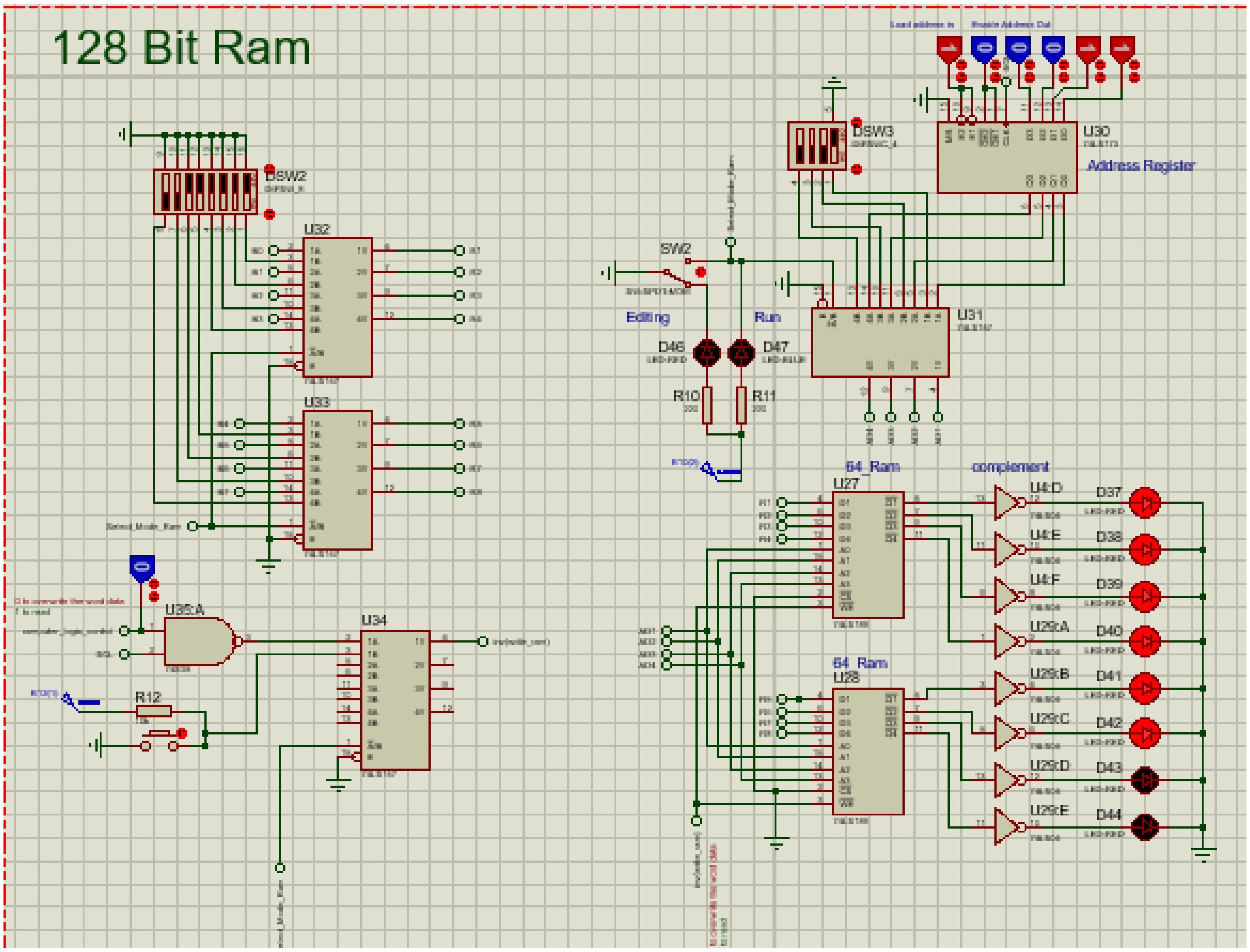


After we made the ram computer's ram it's time to program, and there are two ways to select the address of the word which we want to store the data in, either manually using some DIPS switches or by throw the bus and this will be the Run mode, and to do so we need a way to select which mode we are an edit or run and if you remember we did the same with the clock signal when we wanted to select which one to pass the A-stable or the mono-stable, we will use the same logic here but hence we will need 4 select circuit for each bit then we can use the 74LS158 which contain 4 logic select circuits built in it, in this IC if the select bin is high then the A value will be the one that passes to the output Y, you should also note that both A and B a pre-defined as pull-up so they hold the value of 1 and if we want to pass value of 0 we should ground it and this will be done using the DIPS switches.





There is one extra step we need to do for the ram to be ready which is to use some DIPS switches to manually add the zeros and ones to the ram, in this part will also use 74LS175 as a selector to toggle whether the 8-bits coming to the ram should inter from the DIP switches or should it inter from the bus and this setup will be missing a switch that tills which should pass A or B and this switch already exists it's the same toggle switch we added before, and till now we used a logic state 0 or 1 to decide whether we want to write the data to the ram or not (pin 3 on the ram) but in the future we want the computer to control the write functionality of the ram so we will add a push button that will allow the manual data to pass when we use the Edit mode and also a select gate that will the computer controlling logic and you don't need to understand it for now as it will be illustrated in later section.

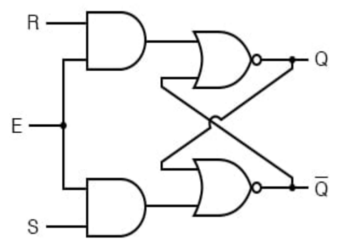


How Build a binary counter?

If you think of it we already build a binary counter before when we made the adder logic and we can modify it so at the rise of each clock signal we add one bit to the last register and that's it, but if you noted the number of compounds to build an adder and a register is a lot, so we need to think of something else, and that something is master slave J-k flip flops.

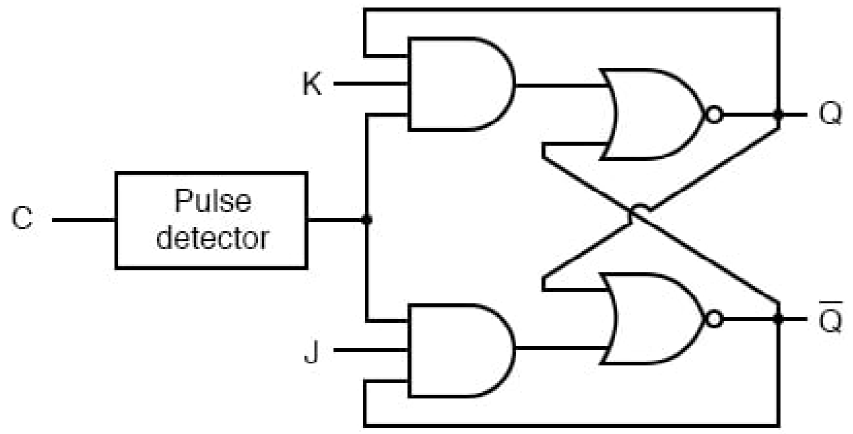
Master slave J-K flip flop:

First, we need to recall the logic of the S-R latch with an Enable Signal and it looked like the shown in the figure, and as the truth table says if the S is high while the R is low then Q will be high and the not Q ($\bar{Q}$) will whatever the inverse of Q but what happen if we connected the R and S pins both to HIGH at the same time! What you may say is that both Q and not Q will be High, and this is not reenable because ($\bar{Q}$) is the inverse of Q, that's way we say in this case the behavior of the output can't be predicted, but before we move to the next point lets clear out that the output signal that appears will be either 1 or 0 for the pin that was triggered first then when the both bins go high then the behavior will be unpredictable so the first signal is a matter of racing between which get to be one first.

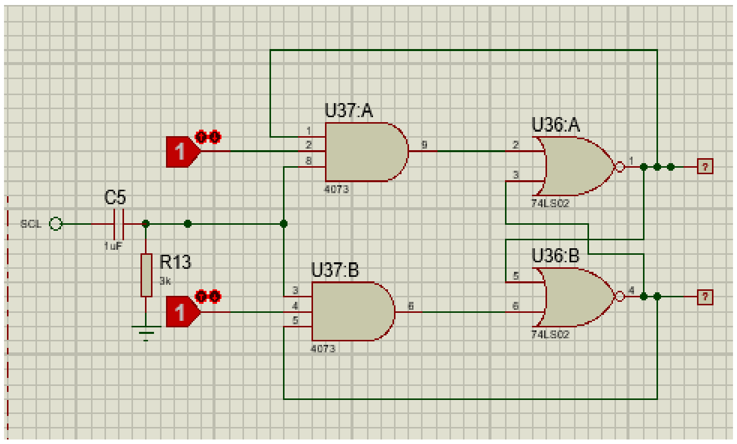


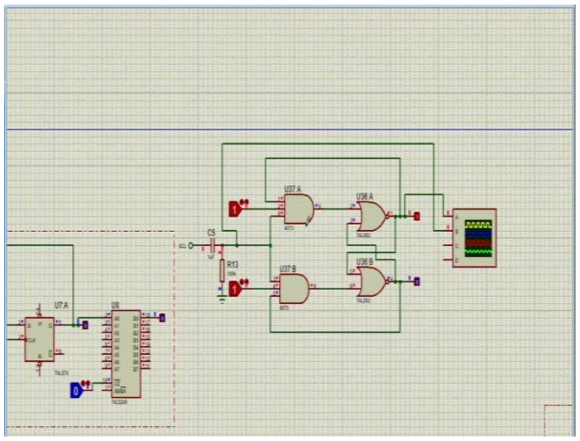
E	S	R	Q	$\bar{Q}$
0	0	0	latch	latch
0	0	1	latch	latch
0	1	0	latch	latch
0	1	1	latch	latch
1	0	0	latch	latch
1	0	1	0	1
1	1	0	1	0
1	1	1	0	0

Will let's make a little change to the circuit and connect the output of Q and ($\bar{Q}$) to a tri-AND gate as follows, then you will note that for the truth table will be the same as for the RS latch, but the only difference is that when both R and S are high the output will toggle between 1 and 0, there for the output of the flip flop in this case can be predictable, and this configuration is called a J-K flip flop, and of course all of this happens only at the rising edge of the clock.

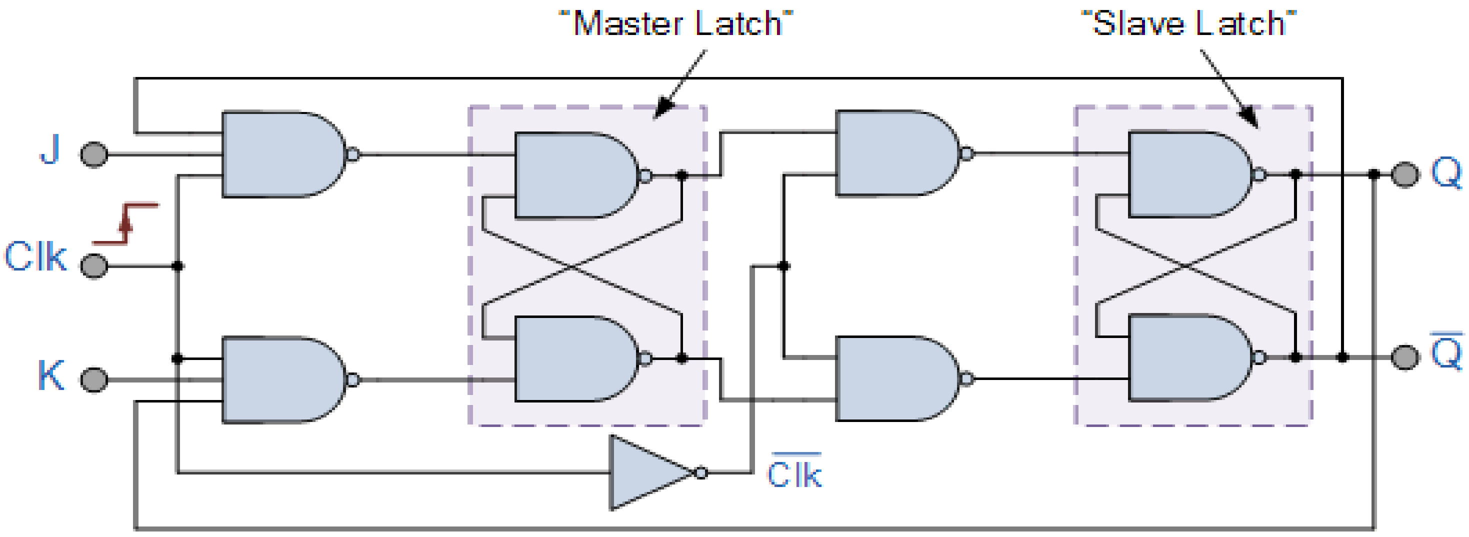


C	J	K	Q	$\bar{Q}$
↑	0	0	latch	latch
↑	0	1	0	1
↑	1	0	1	0
↑	1	1	toggle	toggle
x	0	0	latch	latch
x	0	1	latch	latch
x	1	0	latch	latch
x	1	1	latch	latch





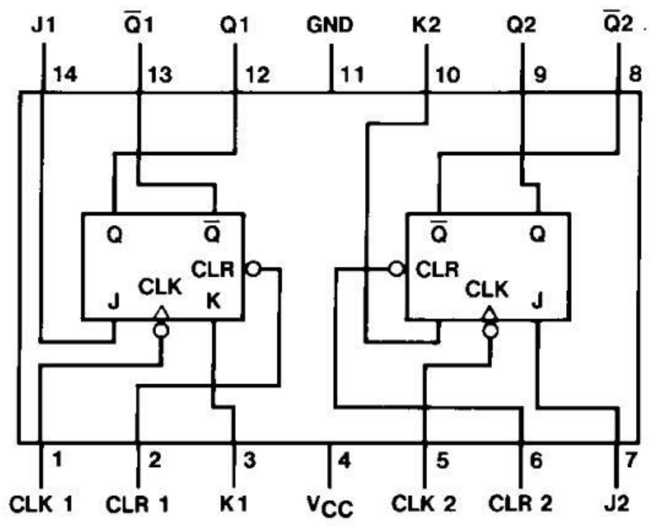
Now if used this configuration and add a RC circuit with a to create a rising edge detecting pulse with a time constant 0.1ms we will note a strange behavior to the output signal as it, this is because the toggling process happens at matter of nano seconds while the pulse stay for approximately 0.1ms there for the circuit toggles more than one time until the pulse time ends and this is what case the output to be unpredictable, a solution might be to lower the pulse time by choosing smaller values of C and R but even thaw this will be in the nano scale and the PCB or the bread board design might add some extra resistance and capacitance and the circuit will be subjected to noise more easily so to address this problem we will use another configuration of the JK flip flop, also known as the master slave JK flip flop, and this logic is also known as a divide by two, because the toggling speed will be half the clock speed.

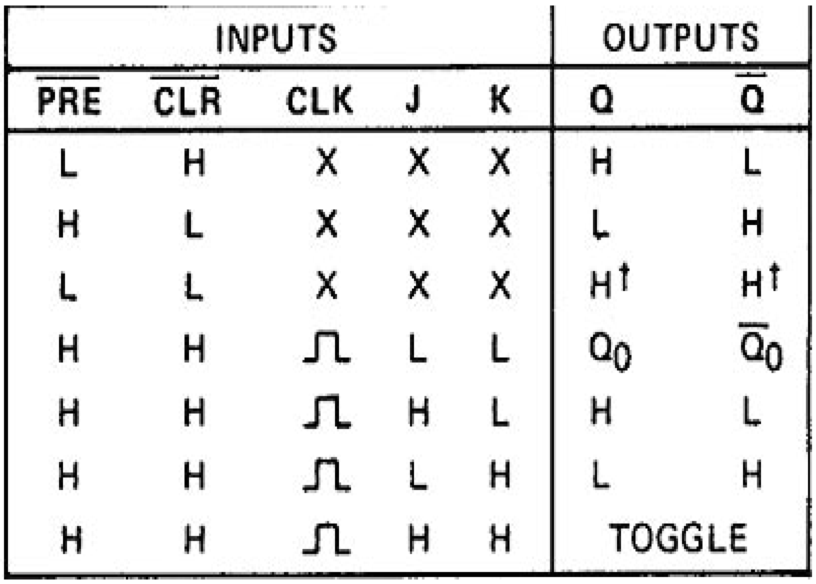


Master slave J-K flip flop

If you looked to the onside of a Master slave J-K flip flop you will note that it's consisted of two RS latch flip flops, but with the enable signal inverted for the second one, and this configuration will actually achieve the stability we want that because at the first part the toggle will happen normally but hence the second flip flop Enable signal is inverted then it will wait till the clock comes low then it will pass the data to the final output and this solves the last problem which was that the toggling happens as long as the Enable is high, and because we need to wait for the enable to go low to output the signal this circuit will take half the time required or half the clock time there for its called divide by two, and here come the question how this will help us to build a counter, and the answer is simple we will pass the output of the first Master slave J-K flip flop to another one and another one and each time we lower the time by 2 so and if you looked at the behavior of the output you will notice that it's the same as a regular counter, I recommend you to see Ben video about this part from here.

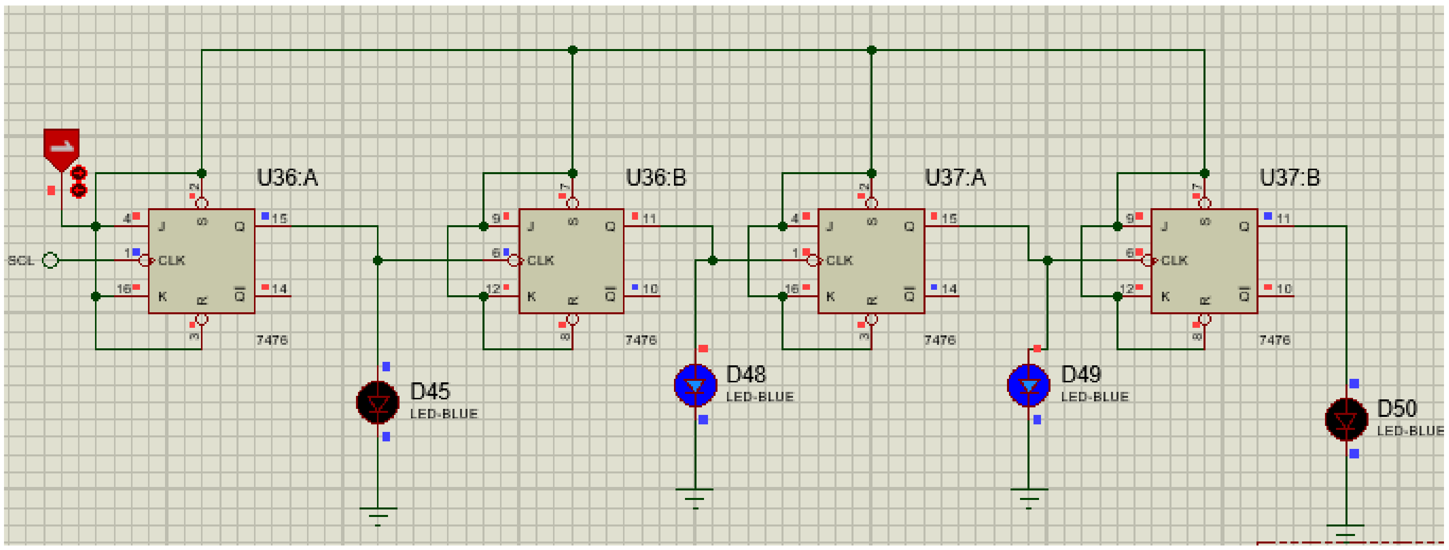
And instead of building the logic using Gates we will use the 74LS76 which comes with two Master slave J-K flip flop inside it. the counter we are building will be 4-bit counter so we will need two 74LS76.





INPUTS					OUTPUTS	
$\overline{PRE}$	$\overline{CLR}$	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H†	H†
H	H	⎍	L	L	Q_0	$\overline{Q}_0$
H	H	⎍	H	L	H	L
H	H	⎍	L	H	L	H
H	H	⎍	H	H	TOGGLE	

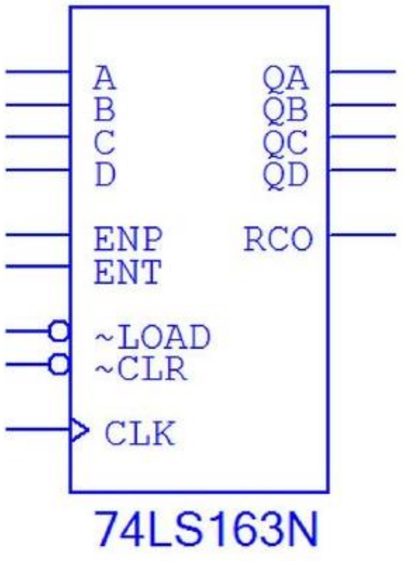
Note that this IC contain 2 Extra pins PRE and CLR and from the truth table we see that P-reset pin sets Q to 0 while Clear pin will set Q to high.



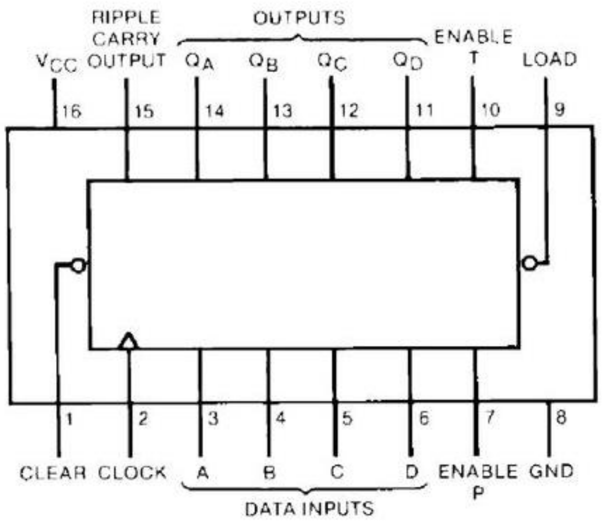
binary counter using master slave JK flip flops.

So lets stop for a moment and see what functionality this adder should have beside being able to count in binary? and it should be like the registers we made earlier, with an enable pin that loads the data to the buss or as we call it a buffer and we already now the IC that can do this job, next we want to add a counting button to display the next number(count enable), an lastly we need a way to load the data into the IC (Jump function) for example we want to start from a given number, and fortunately there is an IC form the 74 series that can do the last task and it actually works as a 4-bit binary counter it also contain a carry pin so if we wanted to increase the counter to 8-bit this IC is 74LS163A

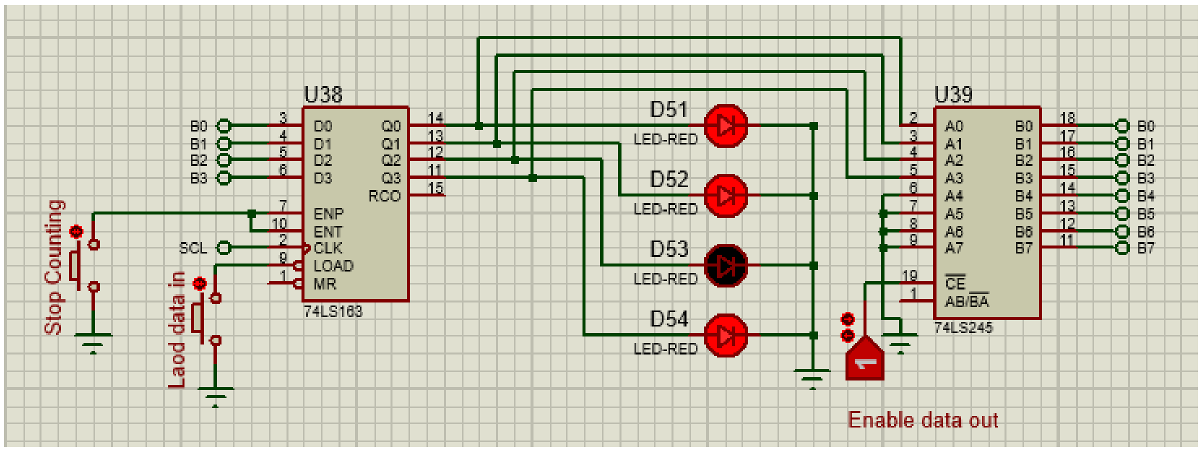
74LS163
Component Diagram
From CDS



74LS163
Connection Diagram
From Datasheet



Notes: the buffer we are using is an 8-bit buffer and we are only using four and hence we will connect the output to the bus we don't want any random values to appear at the last 4-bits so what we can do io to connect the remaining pins to ground to insure an output equal 0, we also should note that the pins A, B, C, D are active high so if LOADED the data in via pin 9 then the input will be ones unless we ground one of them.



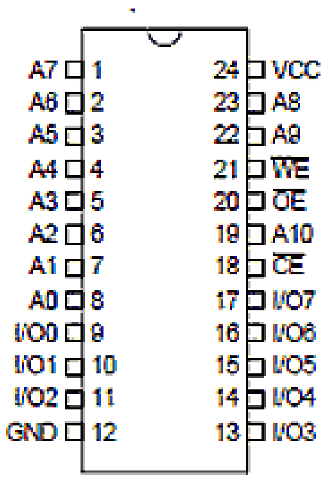
Adding 7-Segment using Gates:

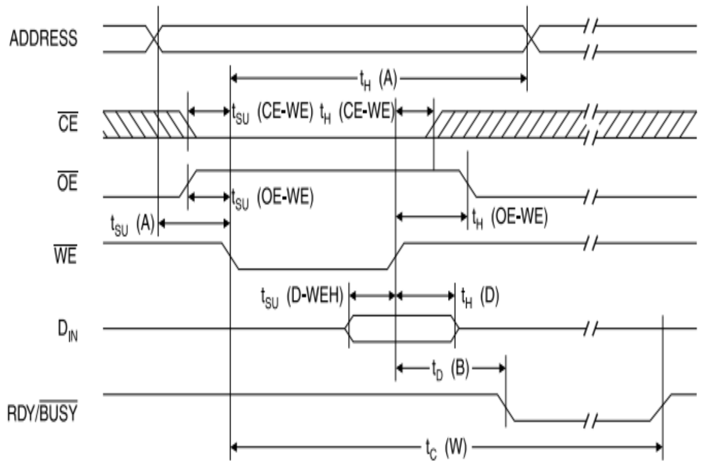
The seven-segment display has an 8 data connection and to light up each segment you need to send a high signal or low depending on whether a common anode or cathode and we can represent up to 16 dismal number on one single display that's by using the hexadecimal numbering system where A ➞ 10, B ➞ 11, C ➞ 12, D ➞ 13, E ➞ 14, F ➞ 15. But the logic would be too hard and costly to do because you need to design the logic for each number for example to display 0 we need all the segments on except the 8th segment and so on, and there is an IC that combines all this logic in one ship and this is the 74LS47/74LS48 but note these ICs can only display the decimal numbers.

EEPROMS How they work and how to program it?

Once we actorly figgerd the binary signal that we should send to the seven-segment display we can use the ram we built before and store the value of each number on it and then use the rebresnting binary number we want to display as the address of the word(row) we want to access and this would work really great but the problem we might face is that when we cut the power from the ram all the stored data will be lost which mean that every time we power up the computer we need to program the ram first before doing any thing, and for a 16 word address this for some people may not be a problem but if we have larger data to store then Rams are obviously not the best options and this should be the right time to introduce EEPROM and it stands for Erasable Programable ROM and a ROM was introduced as a chip that holds some address and they are permanent and can't be erased but with time they evolved it so we both write and read data from it, so to sum up an EEPROM can be treated exactly like a static Ram but the only difference is it won't lose its data after power cut.

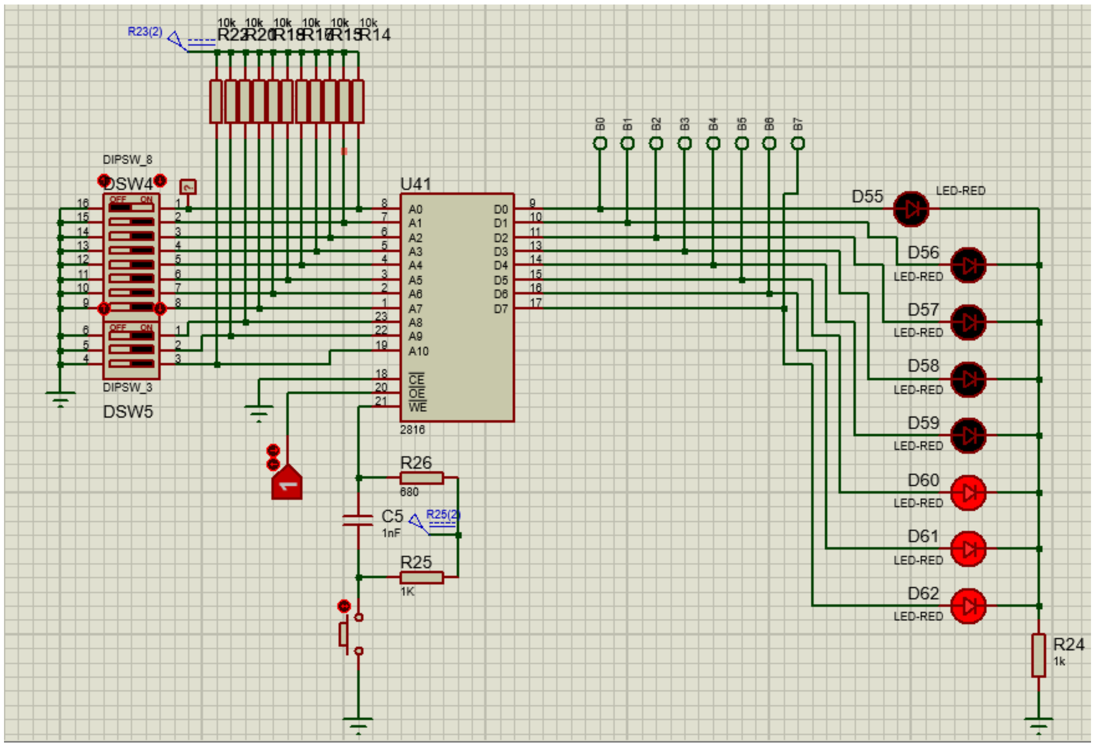
The chip we are going to use is the AT28C16 which has a memory size of 16K bit, (2k address x 8-bits), I recommend to read the datasheet of this chip as it will help us to understand how to write data to it, but further more lets see how to use it.





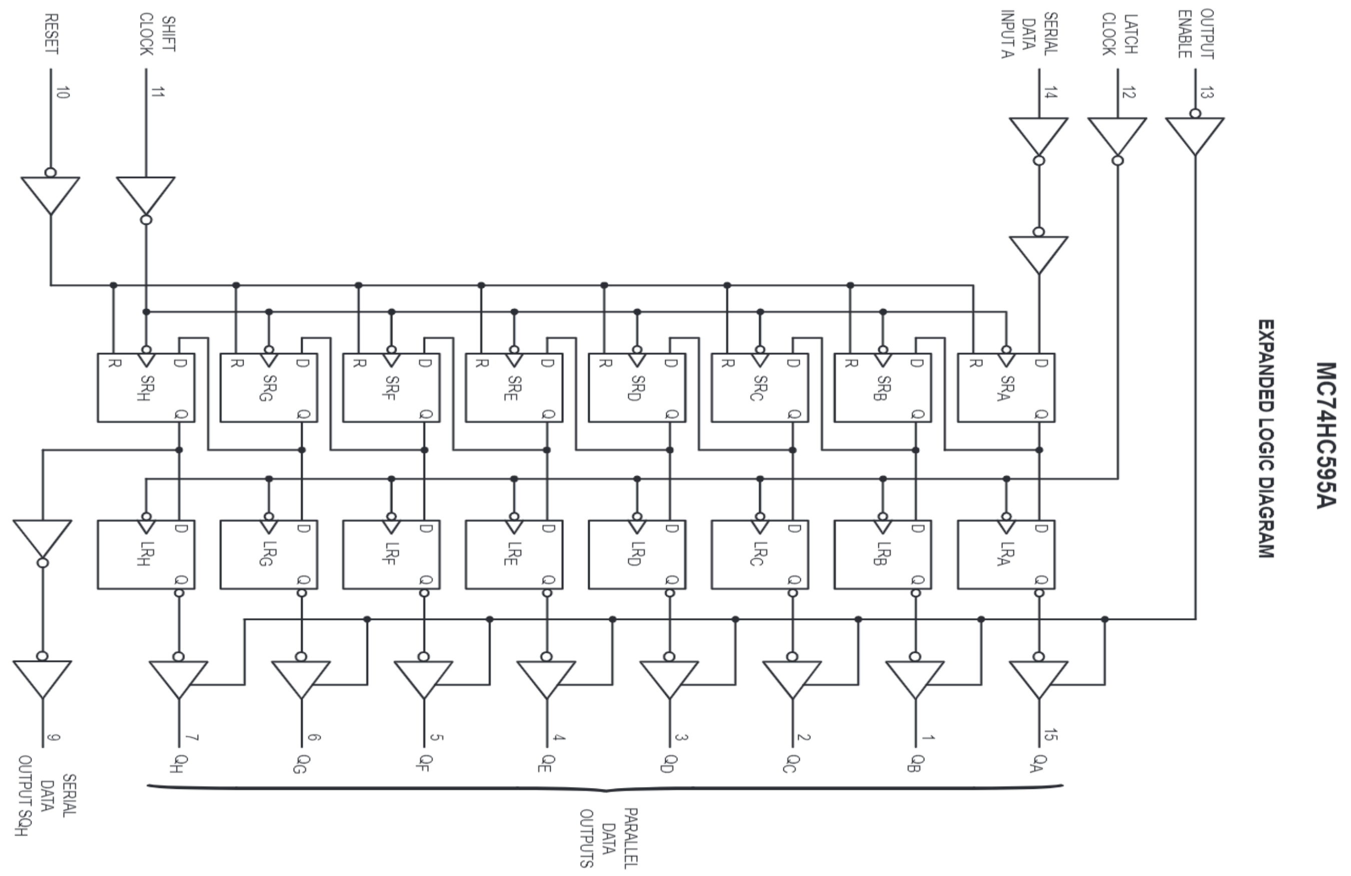
Pin Name	Function
A0 - A10	Addresses
$\overline{CE}$	Chip Enable
$\overline{OE}$	Output Enable
$\overline{WE}$	Write Enable
I/O0 - I/O7	Data Inputs/Outputs
RDY/$\overline{BSY}$	Ready/Busy Output
NC	No Connect

The pins on the chip are as in the table we will use the IO pints to read and write data to the chip, that's by using OE and WE pins there are 10 address pins and we only will use 4 of the them to set the word address to the cell we writing on, the way we use this ship should be the same as we use rams but the only difference is the delay time and you can see from the next figure that the WE signal must be low and between 100 to 1000 nano-sec which can't be done via a button so we need to use an RC edge detecting circuit with a time constant that falls between the write period the values of C and R used are 1nF and 680 *Ω* respectively, and the order of steps should be as follow: first we set the address lines then we put the word we want to write on the IO pins and the last thing is to WE the data.



There are more easier way to program the EEPROM and that's by using a programmer and in this section we will build our arduino programmer, the main function of the programmer is that we will define some words and there addresses in the code and the programmer will write that values to the EEPROM by following the same steps we did before, but if you noted that the number of the pins on the EEPROM is a 20 pin that need to pe programed and if we are using arduino uno or nano we only got 14 digital pin to use, and if we want to pint some messages on the Serial monitor then we cant use the TX and RX pins which leave us with only 12 pins.

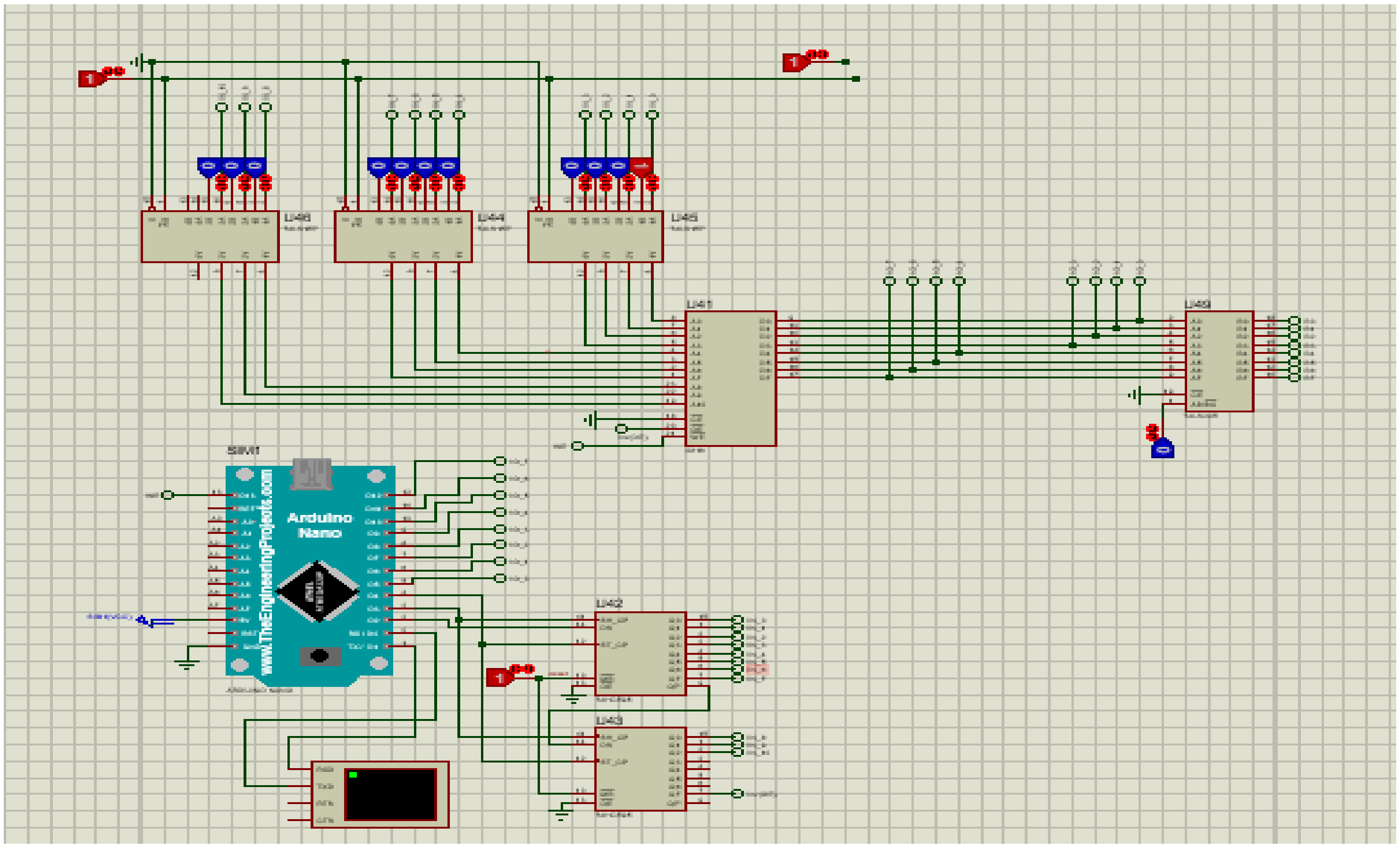
The solution for this problem is simple actually, all we need is to use a shift resistor (Serial-parallel out shift register) and the idea of this type of shift registers is that it take one input every clock pulse and store it inside it using the same concept of D-flip flops, and each time we input a value it shifts the existing value and store the new one, and the chip we are going to use in this project will be 74HC595 and its diagram is as shown



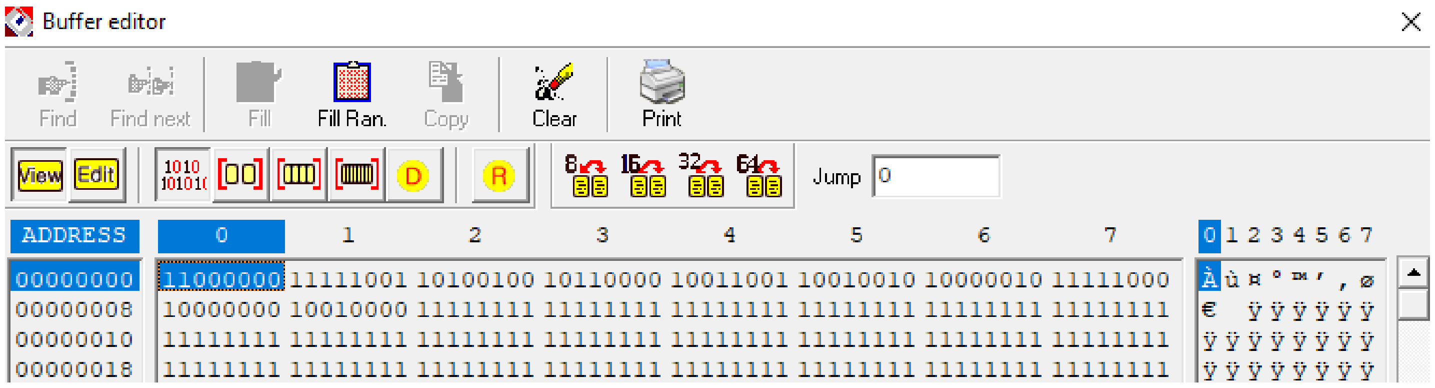
If you noted that the internal is made out of 2 rows of D-flip flops and this adds an advantage to this chip this is because the inputted data will be stored at the first row and won't overwrite the existing one until we LATCH CLOCK(pin 12) to pass the data you will also have an output enable pin(13) and the data in is pin (14), and this IC can store up to

8-bits but of you want to extend it to even more you can use Serial data output pin(19) and connect it to Data input of the next chip, in this project we will need two of these ICs for selecting the Addresses and while the IO pins will be programed using the remaining digital pins on the arduino, the code used in this part can be explained better than ben did in his video which you can find here.

We will add an extra part to isolate the arduino connections from the rest of the circuit because the programmer will be only used once.



Actually, I had some errors using arduino to write the data on the EEPROM in simulation and I’ve tried a lot to solve this problem but I don’t now way it won’t work so, instead I used to a program to get the image that can be uploaded the EEPROM and this program is Max Loader and allow you to select the type of chip you are using then edit each address as you want, in proteus the A28C16 doesn’t support uploading a file to it so I used the A27C256 which support this option the connection should be the same, so if you somehow were able to use arduino so you can skip this step.



```
ADDRESS      0        1        2        3        4        5        6        7       01234567
00000000  11000000 11111001 10100100 10110000 10011001 10010010 10000010 11111000   Àù¤°™',ø
00000008  10000000 10010000 11111111 11111111 11111111 11111111 11111111 11111111   € ÿÿÿÿÿÿ
00000010  11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111   ÿÿÿÿÿÿÿÿ
00000018  11111111 11111111 11111111 11111111 11111111 11111111 11111111 11111111   ÿÿÿÿÿÿÿÿ
```

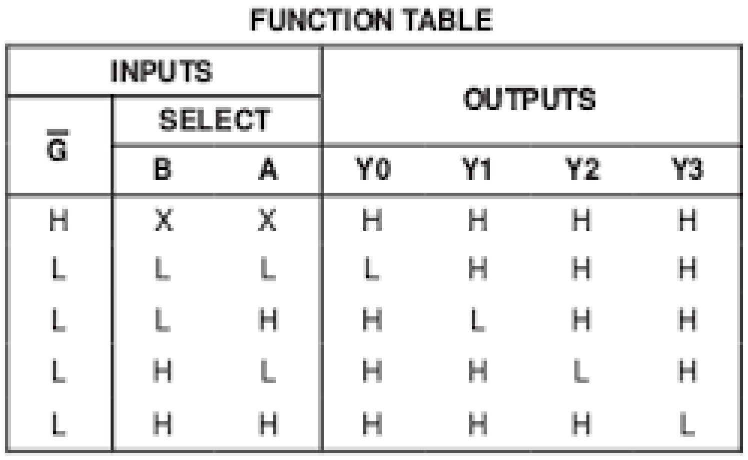
You need to make sure that you save the file as name.bin then you add into proteus.

After we finished how to display a number between 0 and 9 on a seven segment display what we want know is to display an 8 bit number up to 255 and we also want to add a negative sign for the negative numbers.

To achieve this part we will need to first add a 4 seven segment displays (common cathode) and connect them to the same output, but in this configuration it all of this will display the same value, so how could we modify it, lets assume that we want to display 245 on the display so we may start by getting 2 and hence they are 4 displays so the message appearing will be 2222 then we get 4 which appear as 4444 and lastly 5555

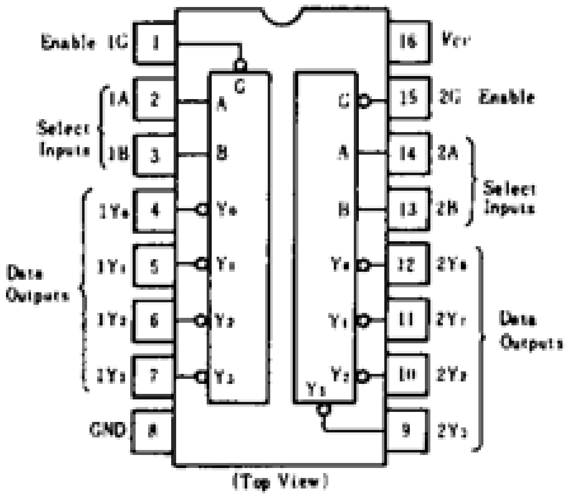
Now if we added an extra logic to switch off all the displays except the first one when we are outputting 5555 the first display will have a 5 in the right position ---5 then we want the same circuit to switch all the displays off except the second display --4- and the same for the third -2—if we switched this movement fast enough we will not notice this movement of the numbers and it will appear as if it was -245.

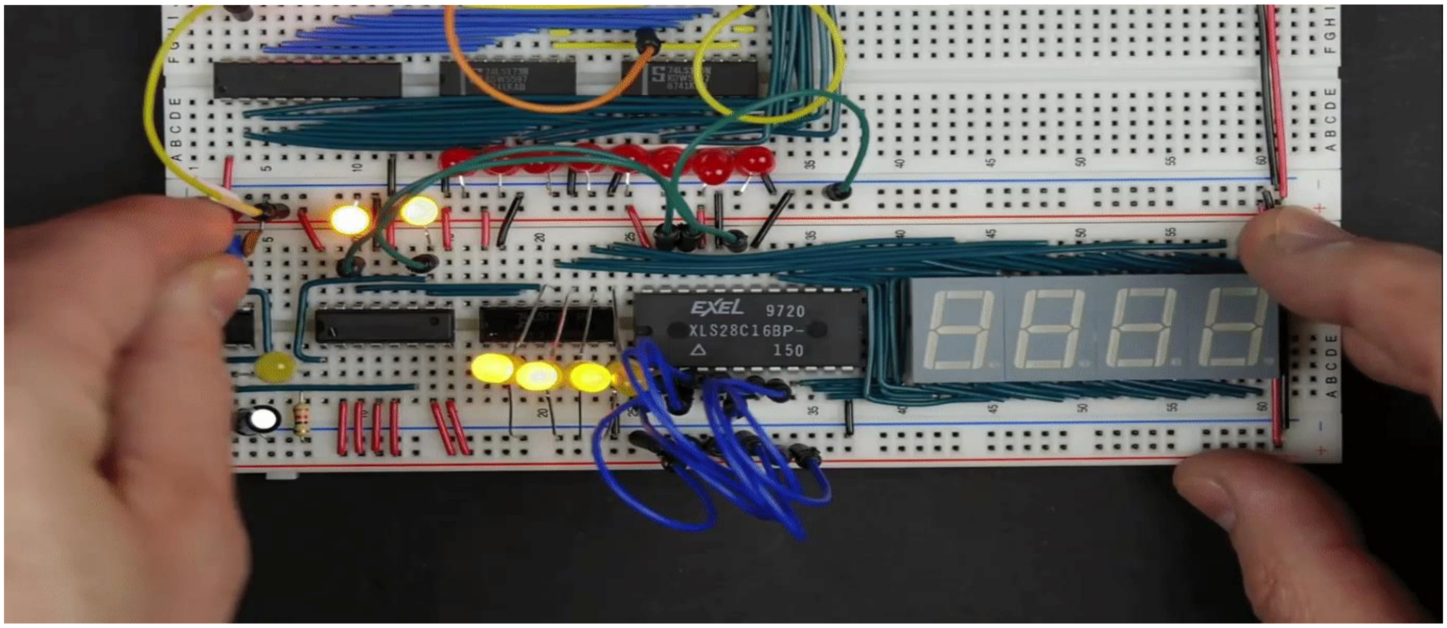
So how this logic works, will first we need to design a 2-bit counter using JK flip flops and we already saw how to do so, this counter will enable us to keep count at which digit we are so the first digit will be 00 the second is 01 and the last one is 10 and will add this signal of zeros and ones to our address so if the output of the counter is 01 this mean that we want the ROM to output the number 4444 at all the displays and if the count is 11 then the number displayed will be 2222 and so on so each and the code for this can be found with ben's get-hup folder, so what about the logic that switches the display on and off, for this we will use the 74LS139 and to understand how this works lets first look at its truth table, and you will notice that it has two inputs A and B and if A and B are in the order 00 then the first output will be low and if AB are 01 then the 2nd output will be low while the rest of the outputs remains high and if AB are 10 the 3th position will be low and you can see hence we are using a common cathode display the output pin that is low will set the display that is connected to it on



FUNCTION TABLE

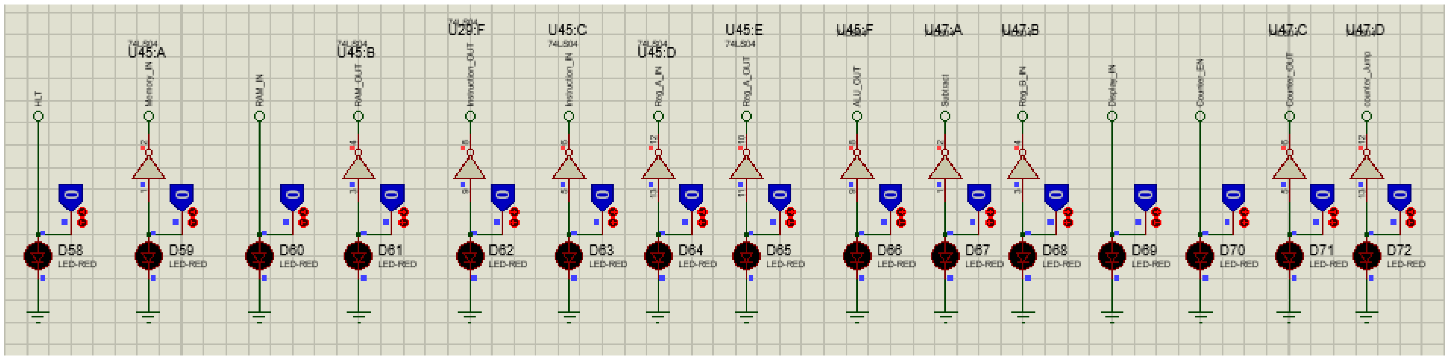
INPUTS			OUTPUTS			
$\overline{G}$	SELECT					
	B	A	Y0	Y1	Y2	Y3
H	X	X	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L





CPU

After we made most of the parts for the computer we now need to group these parts by first connecting them to the bus so they have a way to communicate with each other, then we need to get all out control signals like counters out/in ram out/in... in one place his will make it easier to control the computer when we are building the CPU you should also note that some signals are active low while the others are active high so it will be much convenient if all of them was active low or active high and for this we can use NOT gates to convert the input of a signal the control unit will be as shown:



After we finished this step its time to create out command and by our I mean if you designed this computer that you can set the name of the command, where they should be stored, when to execute literally everything but first there one thing that all these commands will have in common which is there format lets assume that want to make a command called AAA then it should have the next format

Command name	Step done at	Name in binary	Where to point
AAA	0000	0001	1111

So what this format mean? Fist we need to tell the computer when to execute this command at the start at the end, and to do this accurately we will link the step we want the command to be executed at to the computer counter, this mean if we want this command to be the second thing that it dose so we will store it in the EEPROM at the address 0010 (2) so that when the counter is 2 this command will be executed, the second part is the name in binary and this is just a representation of the command so that the computer can now what this command is and again the name can be anything you want the last thing is after all this is done what should the computer do and for this command we want it point us the address 1111 of the memory to see what is in it, but how we will handle all this format will the first two parts will be used in the EEPROM as the address of this step note the step (count) will change with time and the last part of the format (where to point) will go to the Ram, the EEPROM while output the instructions that control each signal, this part Is important and if it looks confusing I strongly recommend you to watch ben videos about how the CPU works as he describes it really good and you can find it here also the examples he used is the same here so you will find the same truth table here:``

Order	Command	Function	Lo cation on memory	Name	Go to
1	LDA	Get data and store it to register A	0000	0001	1110
2	ADD	Add	0001	0010	1111
2	OUT	Display on 7-segment	0010	0011	Not matter

Note you can store the command at any location you want with any name you want.

	Instruction				Step			HLT	MI	RI	RO	IO	II	A I	A 0	0	E0	SU	BO	OI	CE	CO
start	x	x	x	x	0	0	0	0	1	0	0	0	0	0	0	🟥	0	0	0	0	1	0
	x	x	x	x	0	0	1	0	0	0	0	1	0	1	0	🟥	0	0	0	1	0	0
LDA	0	0	0	1	0	1	0	0	0	1	0	0	1	0	0	🟥	0	0	0	0	0	0
	0	0	0	1	0	1	1	0	0	0	0	1	0	0	1	🟥	0	0	0	0	0	0
	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	🟥	0	0	0	0	0	0
ADD	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	🟥	0	0	0	0	0	0
	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	🟥	0	0	1	0	0	0
	0	0	1	0	1	0	0	0	0	0	0	0	0	1	0	🟥	1	0	0	0	0	0
OUT	0	0	1	1	0	1	0	0	0	0	0	0	0	0	1	🟥	0	0	1	0	0	0
	0	0	1	1	0	1	1	0	0	0	0	0	0	0	0	🟥	0	0	0	0	0	0
	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0	🟥	0	0	0	0	0	0

If you noted the number of output instruction is 14 while the max output of the EEPROM is 8 so for this we will use 2 EEPROMS one for the first half and the other for the second half, and again we will use to load the data into the EEPROMS.