Specification of the 8-bit Computer:

Clock: adjustable-speed (from less than 1Hz to a few hundred Hz). Realized up to 350Hz

RAM: Two units of (16 words × 4 bits)

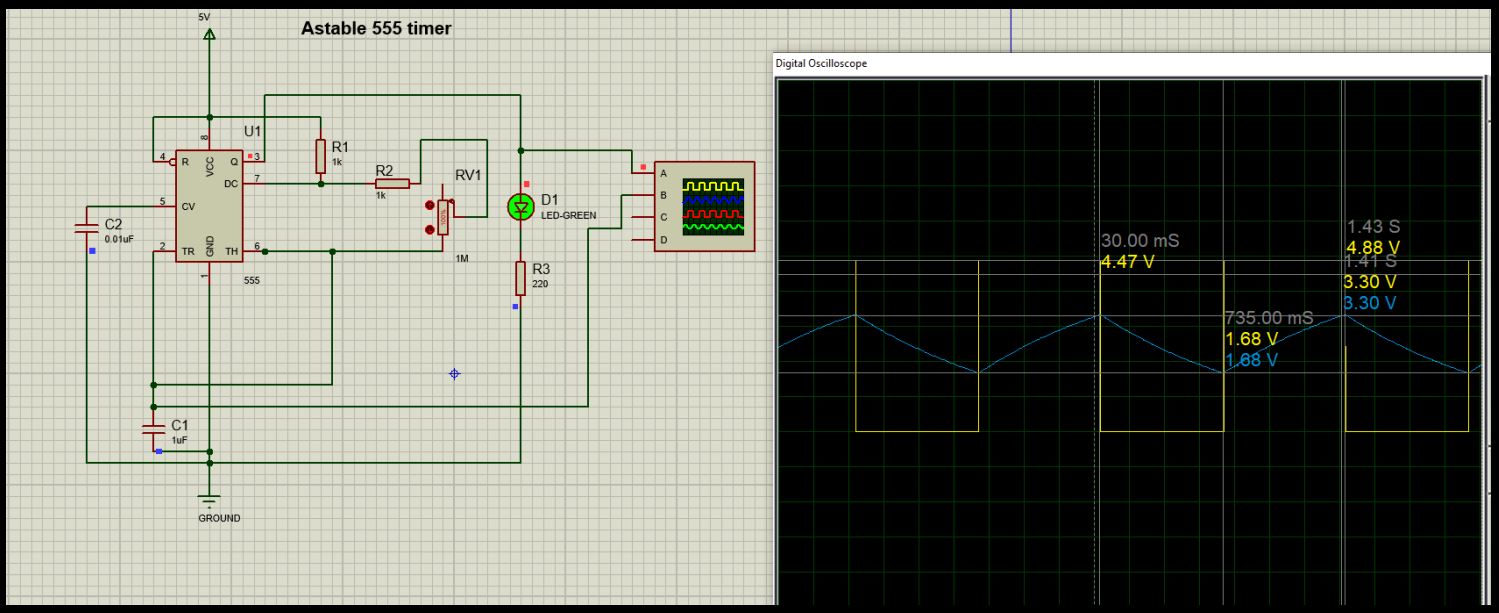
ROM: (2*32KB) for Control Unit

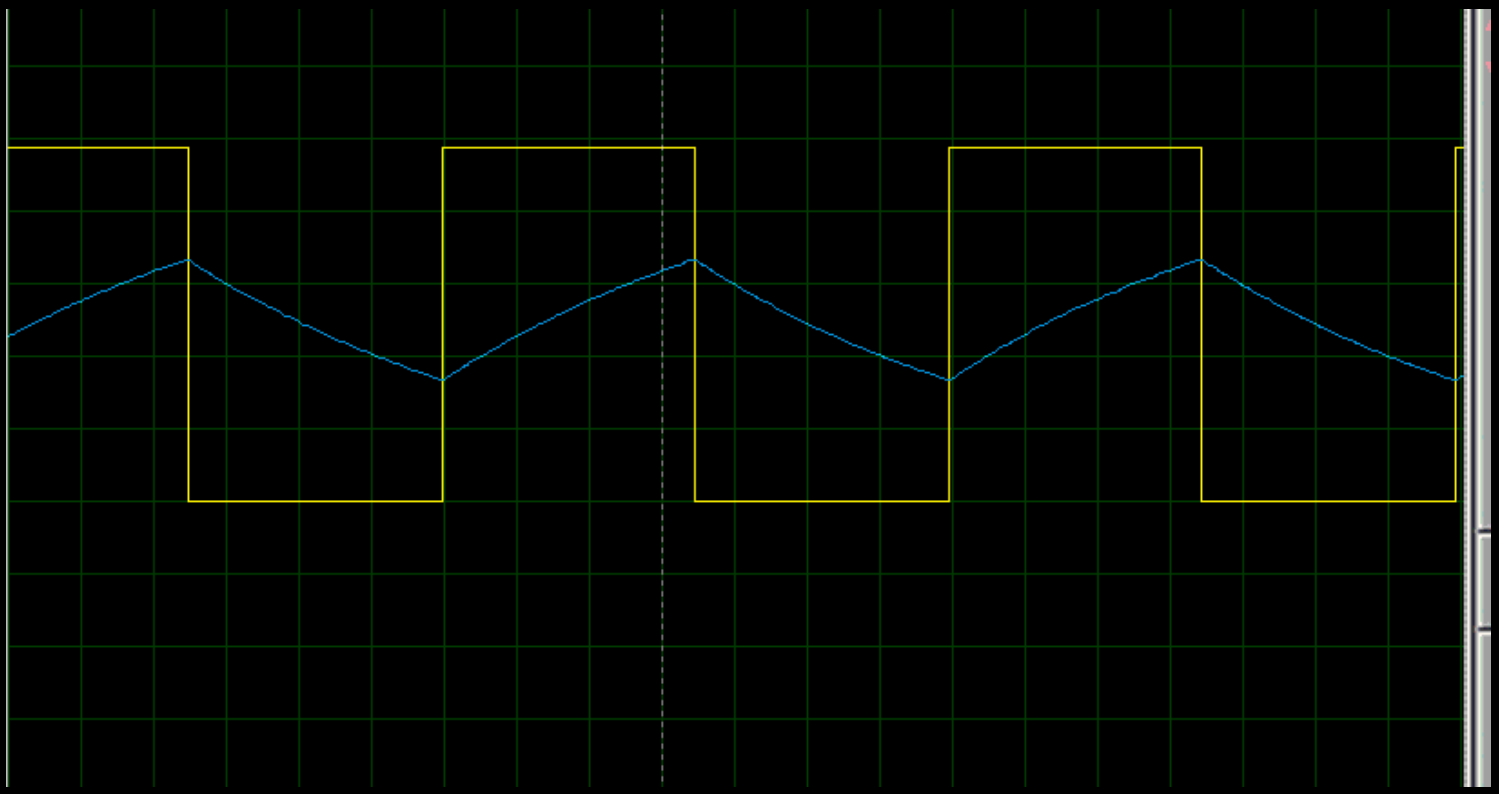
Instructions supported (as of now): LOAD, ADD, OUT, HLT (not Turing Complete)

If we add unconditional jump, conditional jump; it would theoretically make it Turing Complete. Practically, it is still limited by memory, clock constraints.

Units: RAM, Program Counter, Memory Address Register, Instruction Register, Register A, Register B, Output Register, Clock, Arithmetic Logic Unit (Only implemented Arithmetic Instruction as of now), Control Logic Unit.

#1 Astable 555 timer





Electrical Characteristics (continued)

(T_A = 25°C, V_{CC} = 5 V to 15 V, unless otherwise specified)(1)(2)

PARAMETER	TEST CONDITIONS	MIN	TYP	MAX	UNIT
Output Voltage Drop (High)	I_{SOURCE} = 200 mA, V_{CC} = 15 V		12.5		V
	I_{SOURCE} = 100 mA, V_{CC} = 15 V	12.75	13.3		V
	V_{CC} = 5 V	2.75	3.3		V
Rise Time of Output			100		ns
Fall Time of Output			100		ns

The datasheet showing typical output voltage may not be exactly 5V.

T_theoretical=0.693(RA+2*RB)*C = 0.693(10^3 + 2*(10^3+10^6))*10^-6 = 1.38sec
T_observed=1.43s-30ms=1.43sec-0.03sec = 1.4sec

(If we reduce resistor RB to 1kilo ohm, we can realise around 350Hz of clock speed, where T_on=0.693(1000+2*1000)*10^-6=2.07ms and T_off=0.693*1000*10^-6=6.93*10^-4s, f=1/(T_on+T_off)=360Hz

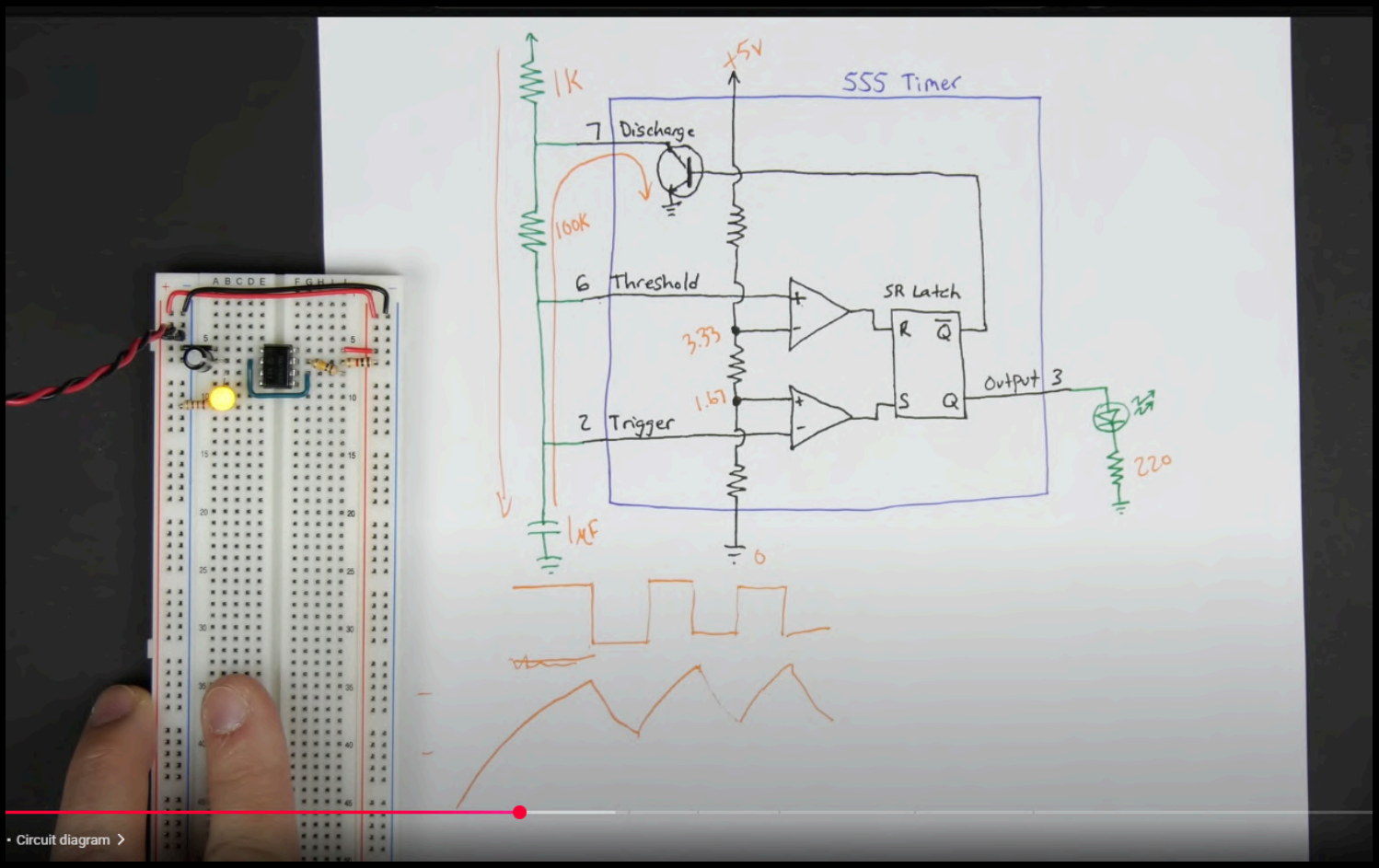
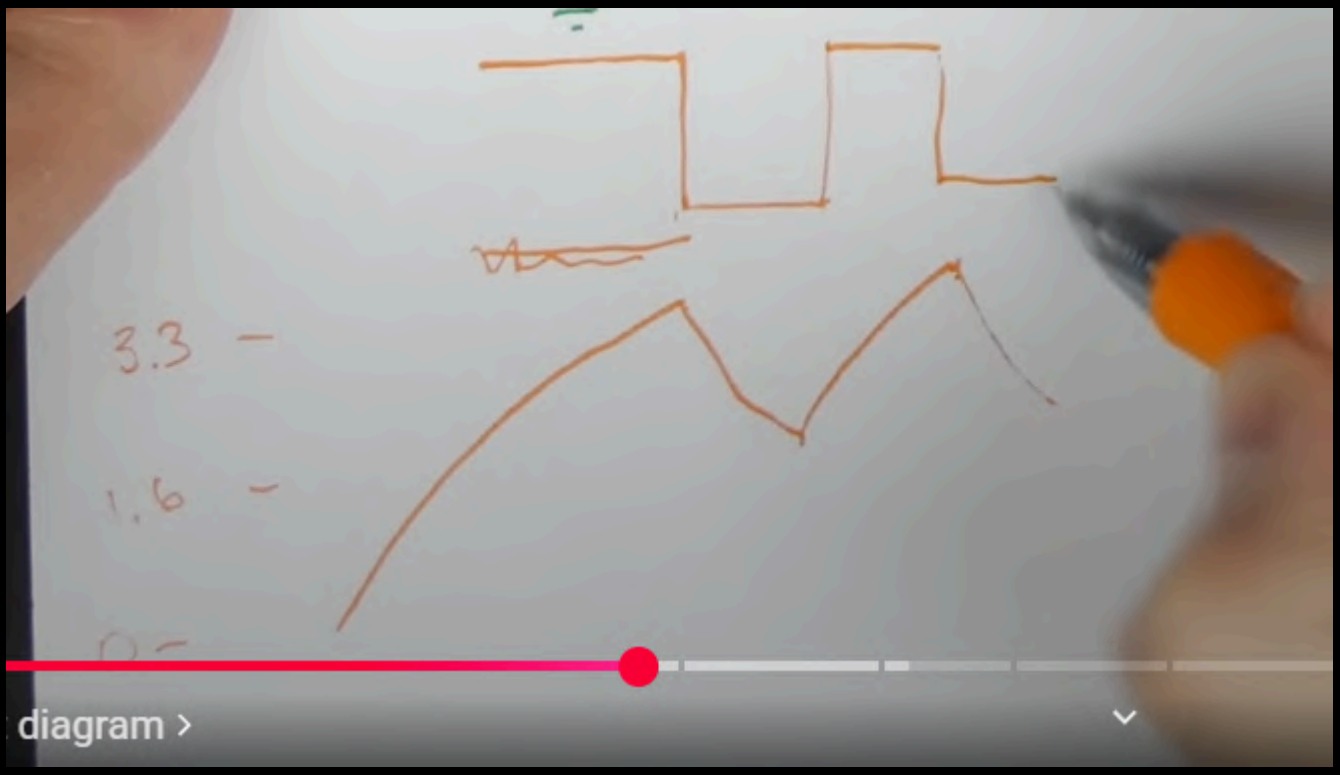
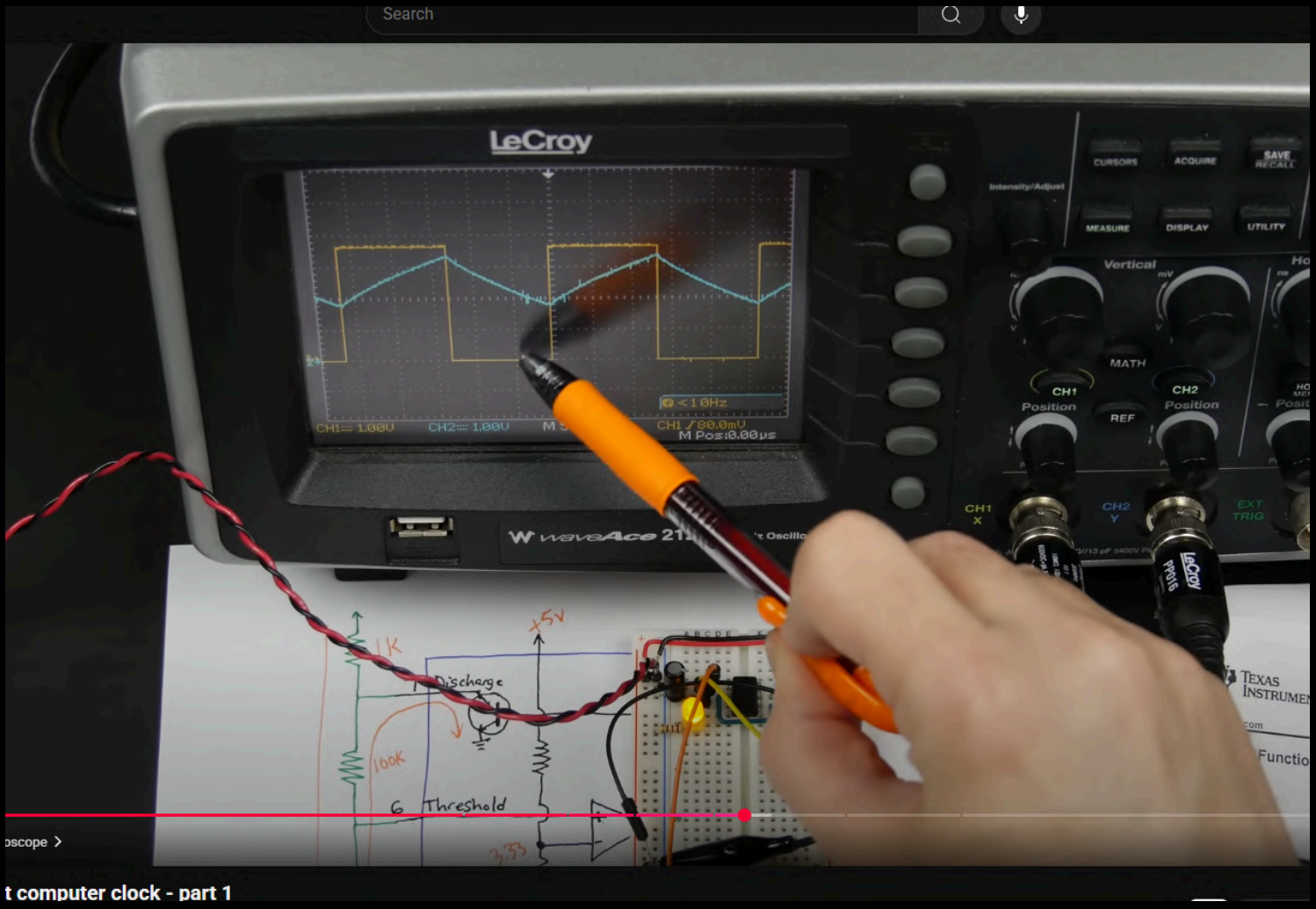


Fig: Clip from Ben Eater's youtube video (from here onwards, it will be assumed by default for any screenshot of Youtube's video)





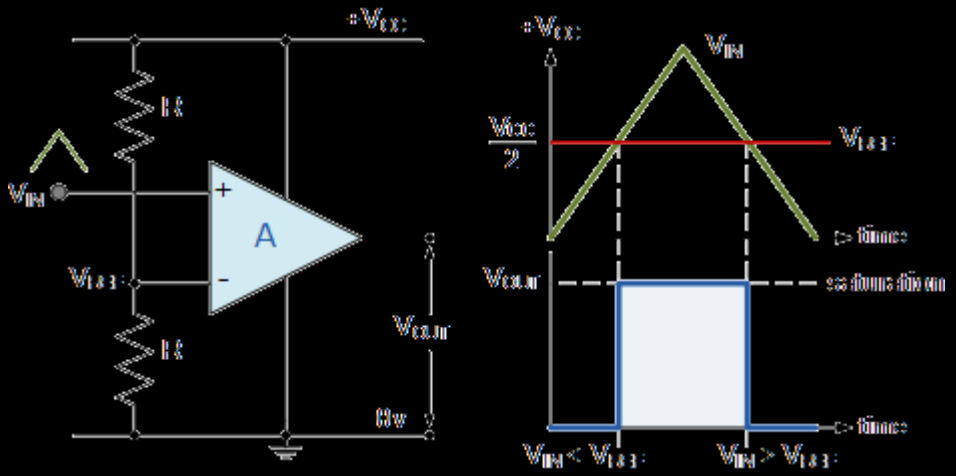
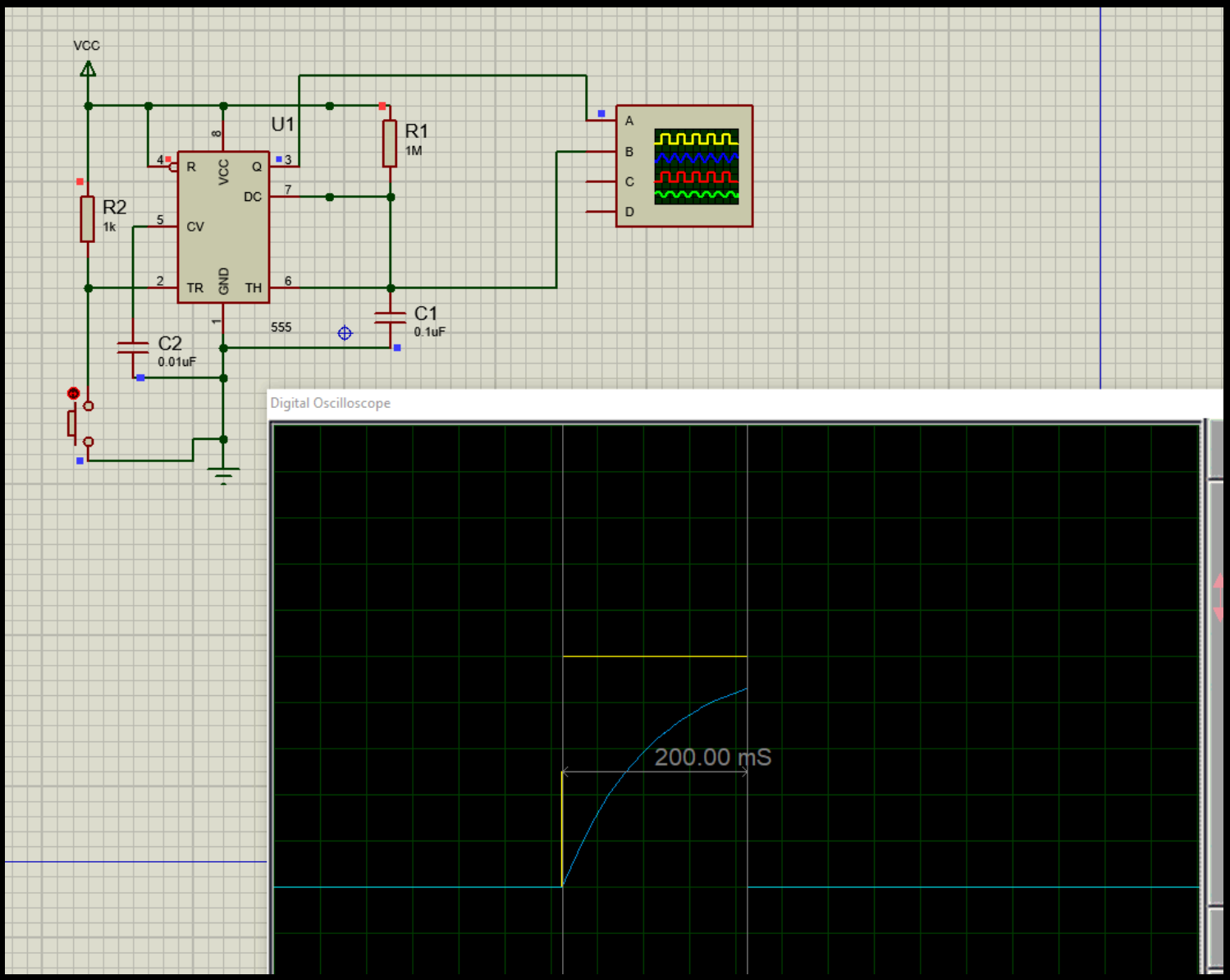


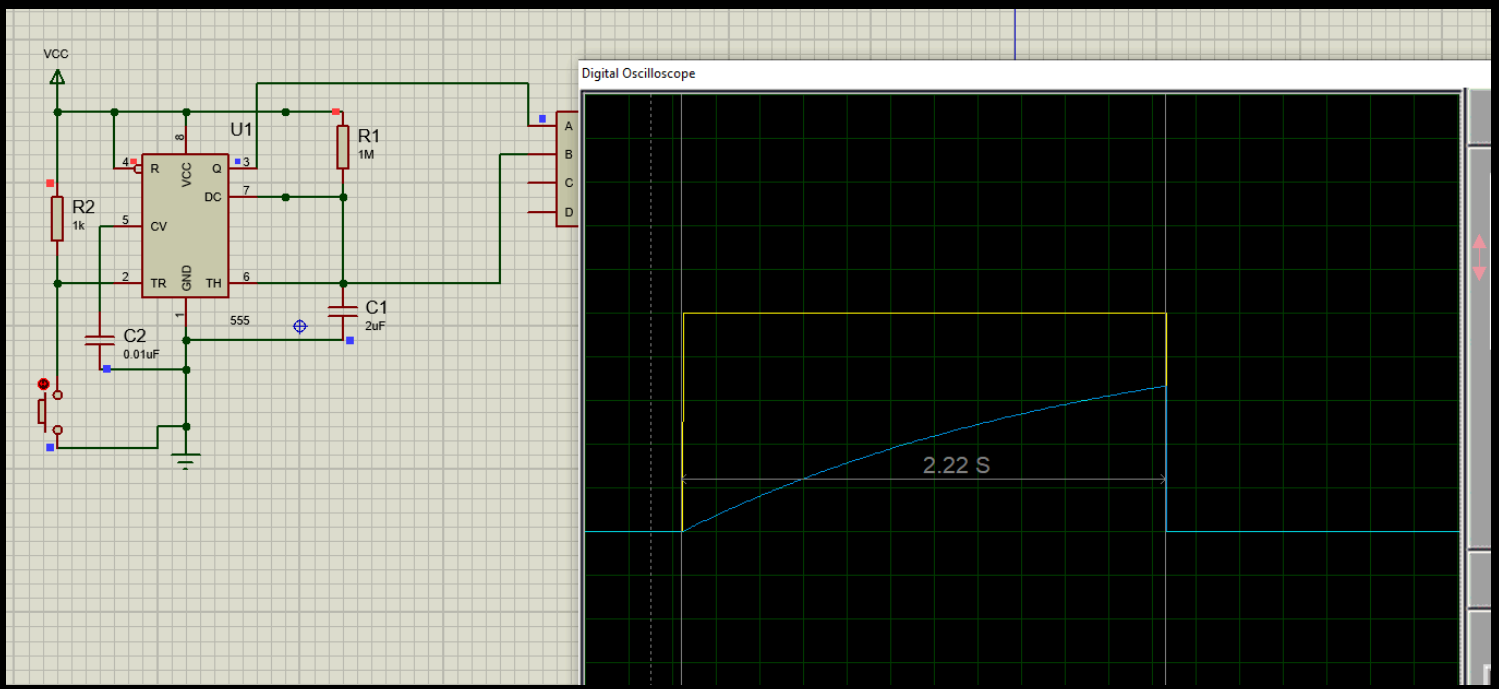
Fig: Working of Comparator

#2 Monostable 555 timer



T_theoretical=1.1*R*C=1.1*10^6*0.1*10^-6=.11sec=110ms

T_observed=200ms (the error is reduced for higher values, but for lower values this is usually large)

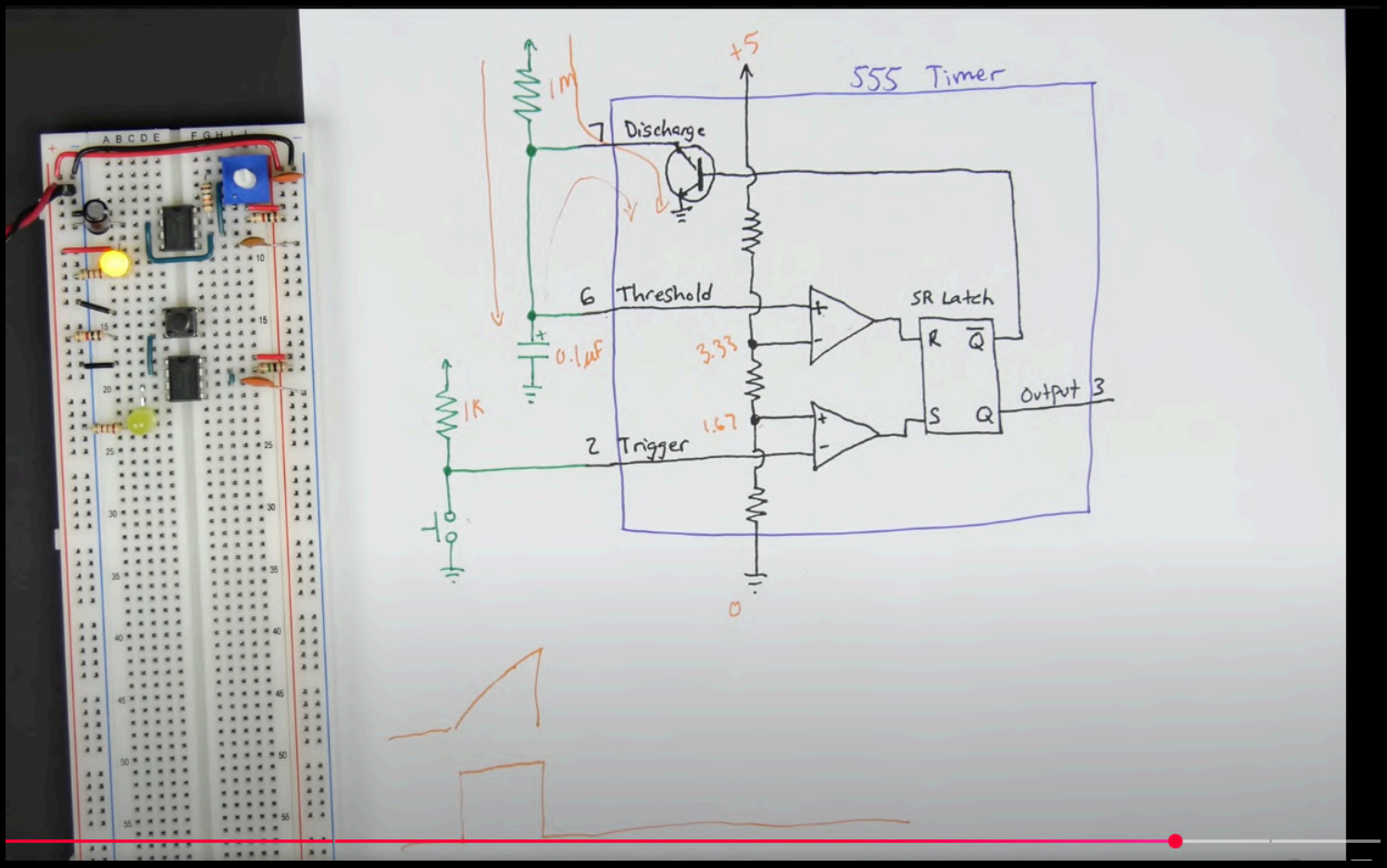


T_theoretical=2.2sec

T_observed=2.22sec

(AC, DC Coupling in an oscilloscope can get messy, so I had to study this https://www.youtube.com/watch?v=VJJ-y9gjjkI)

-Metallic parts in a button can bounce, so even if we pressed once, due to bouncing of plates, the switch inside the button can get closed more than once, hence creating more than one clock pulse.
The solution: Debouncing circuit using 555 timer (ensures one clock pulse, the time period for ON state is determined by the product of resistor and capacitor.)

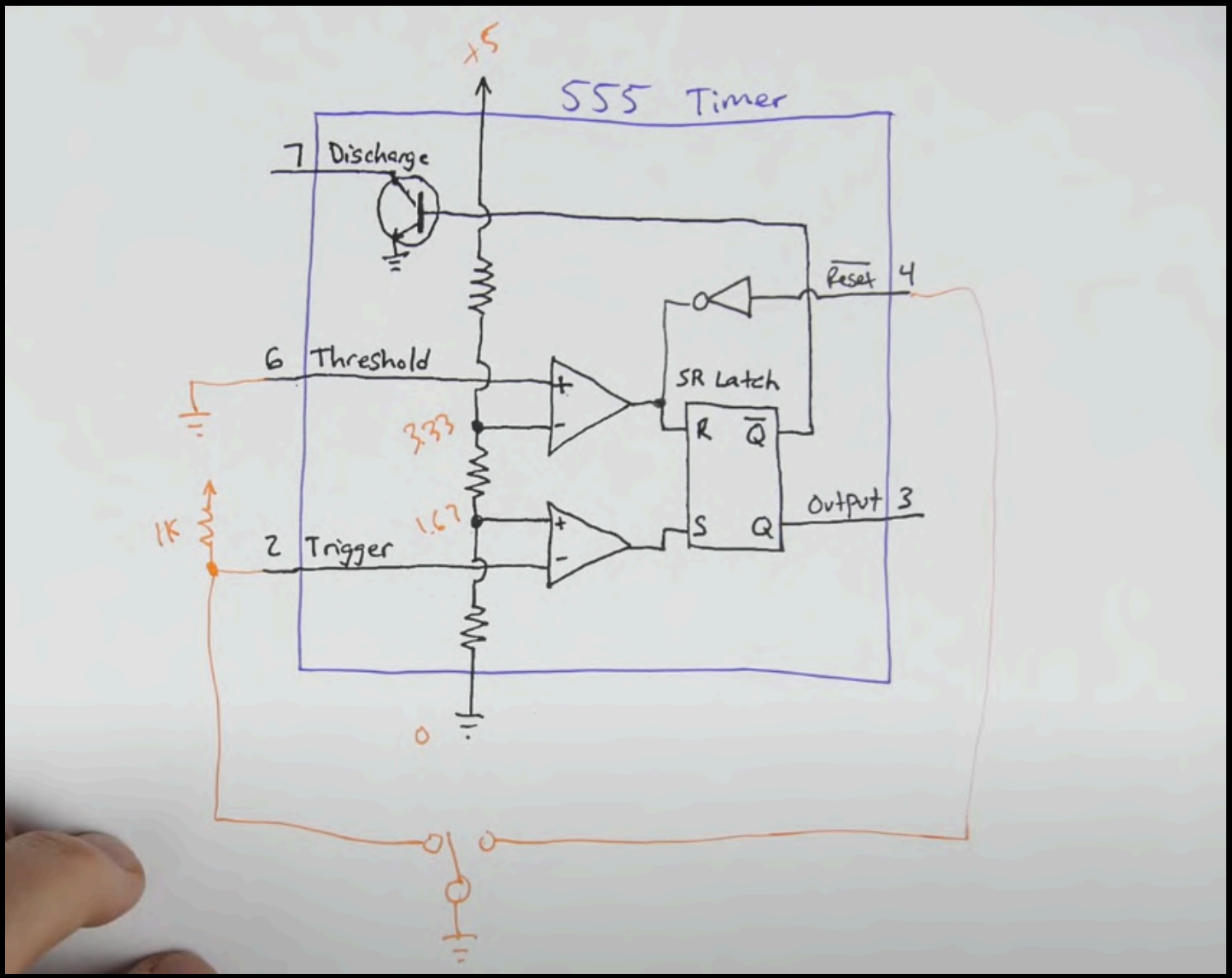


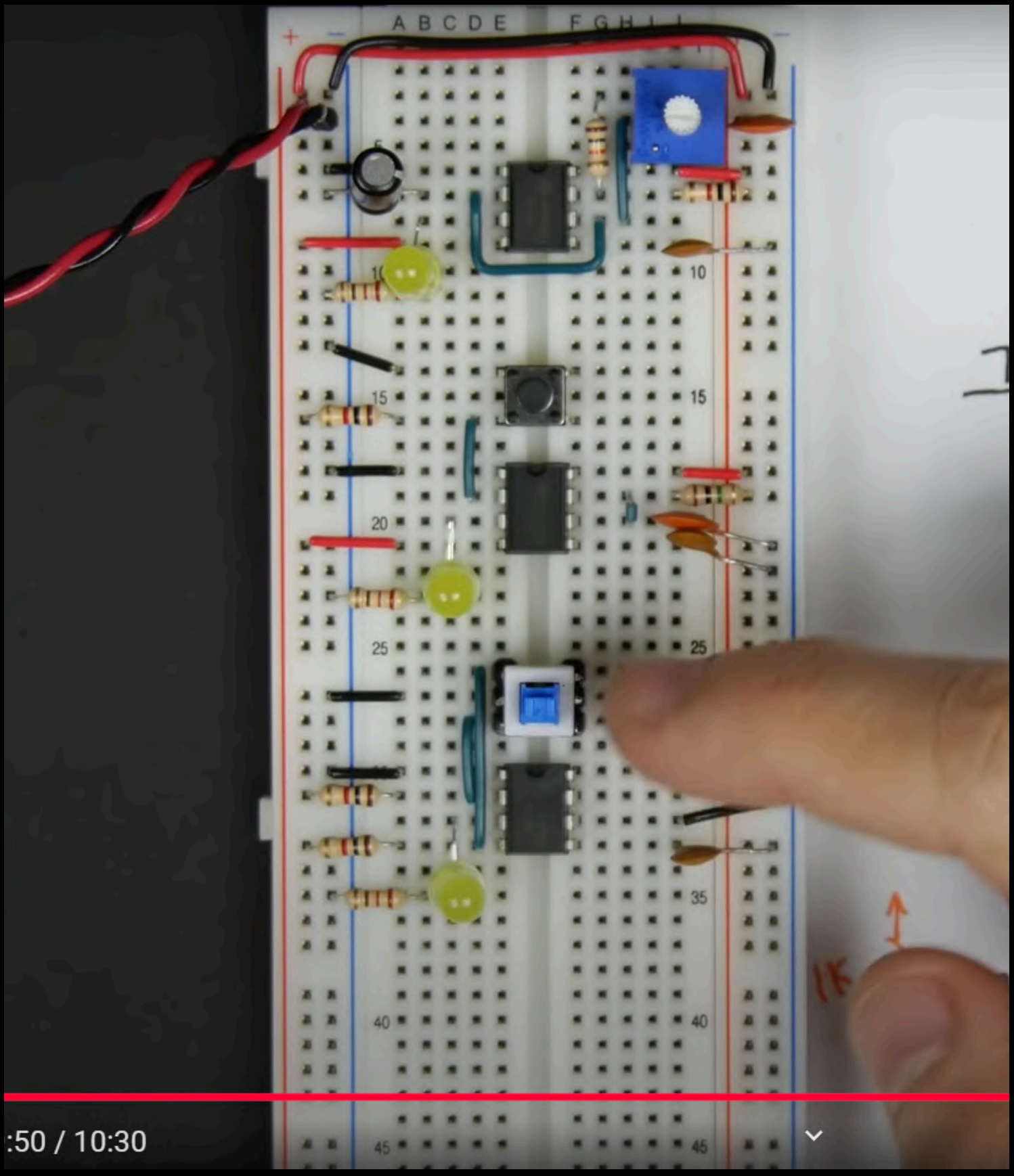
Source: Ben Eater's video

Also, I had to go through documentation of Oscilloscope for Proteus at https://www.labcenter.com/blog/sim-scope-controls/#.

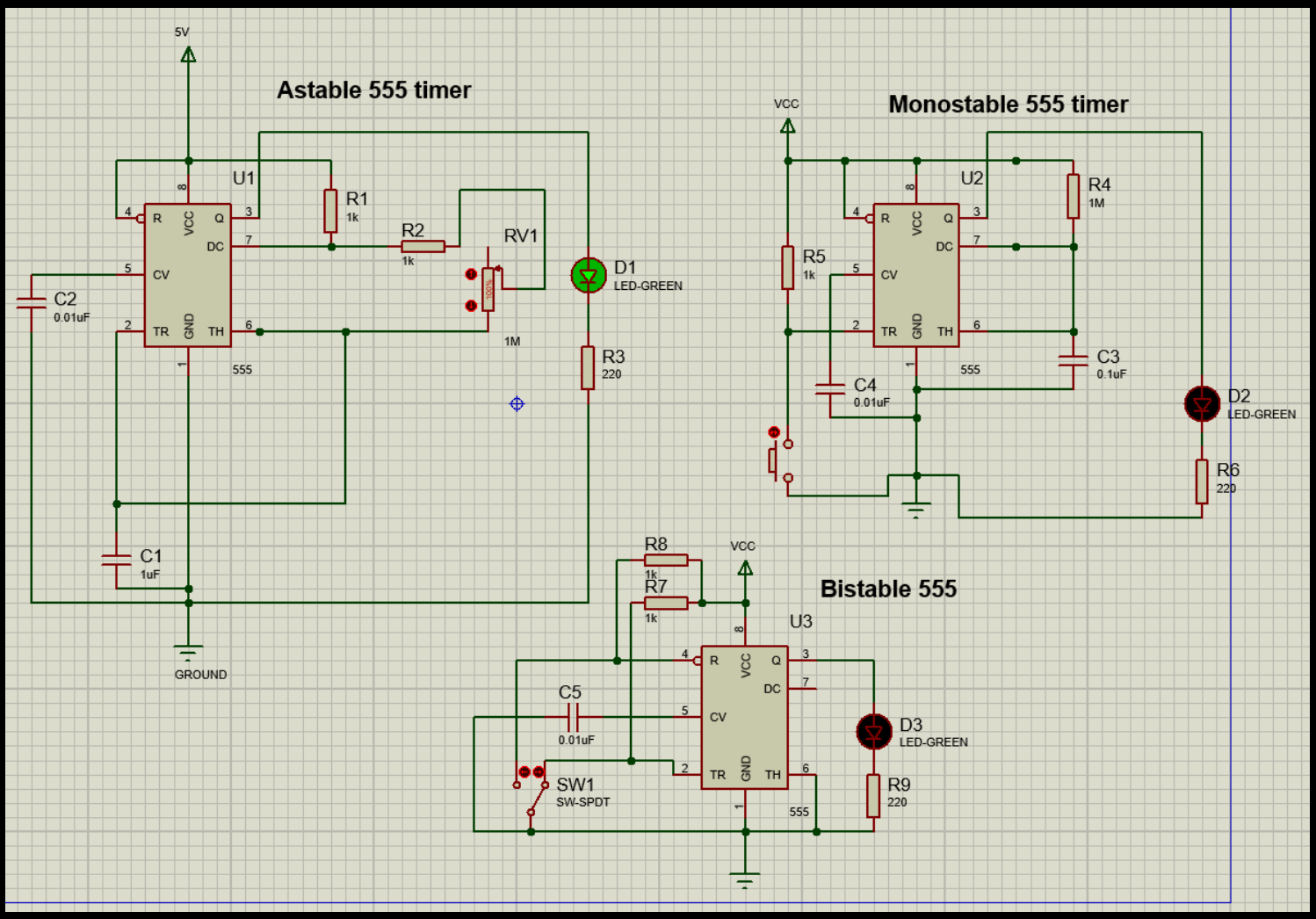
#3 Bistable 555

Switch can have debouncing problems as well. Types of switch: make before break and break before make (most are of this category).

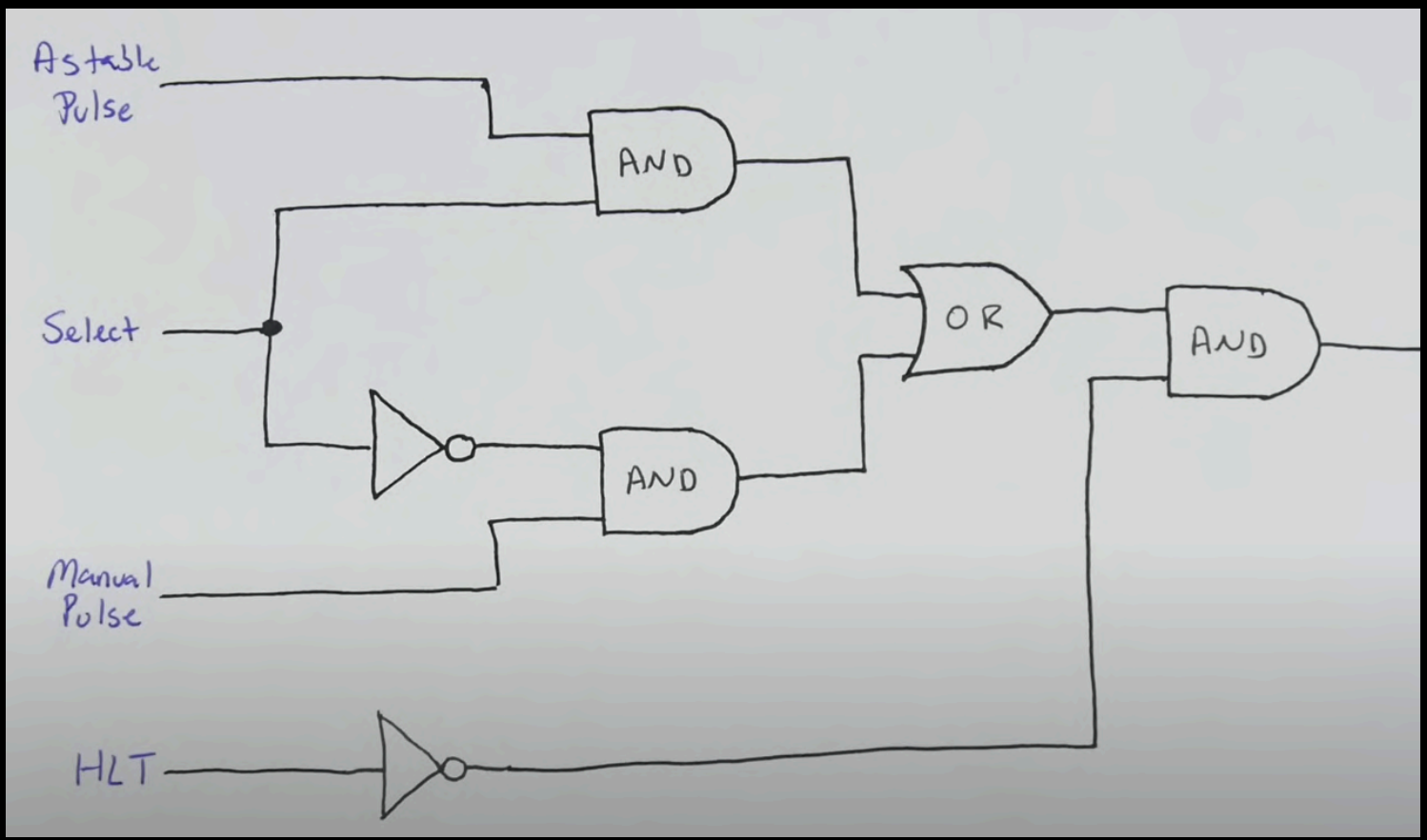

+5
555 Timer
7 Discharge
Reset 4
6 Threshold
SR Latch
3.33
R Q̄
Output 3
1K
1.67
S Q
2 Trigger
0

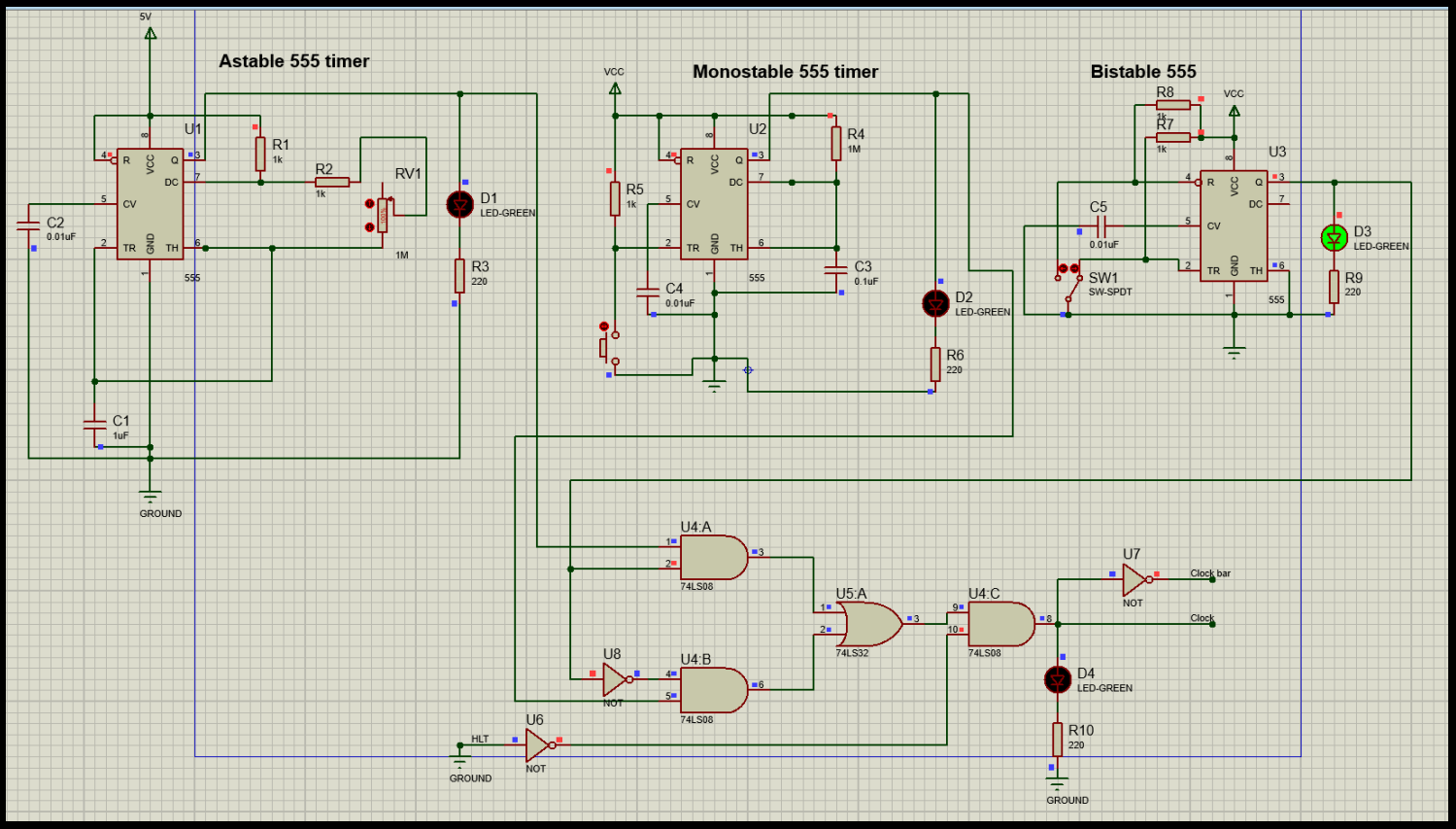


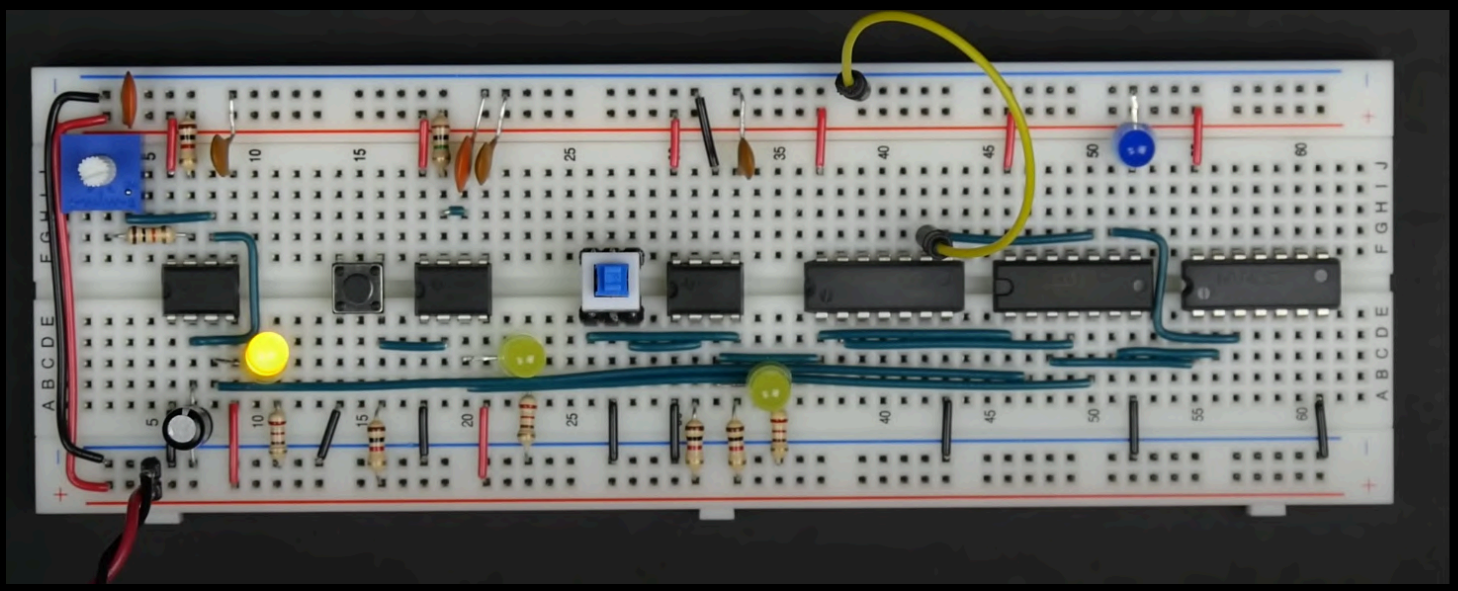
:50 / 10:30



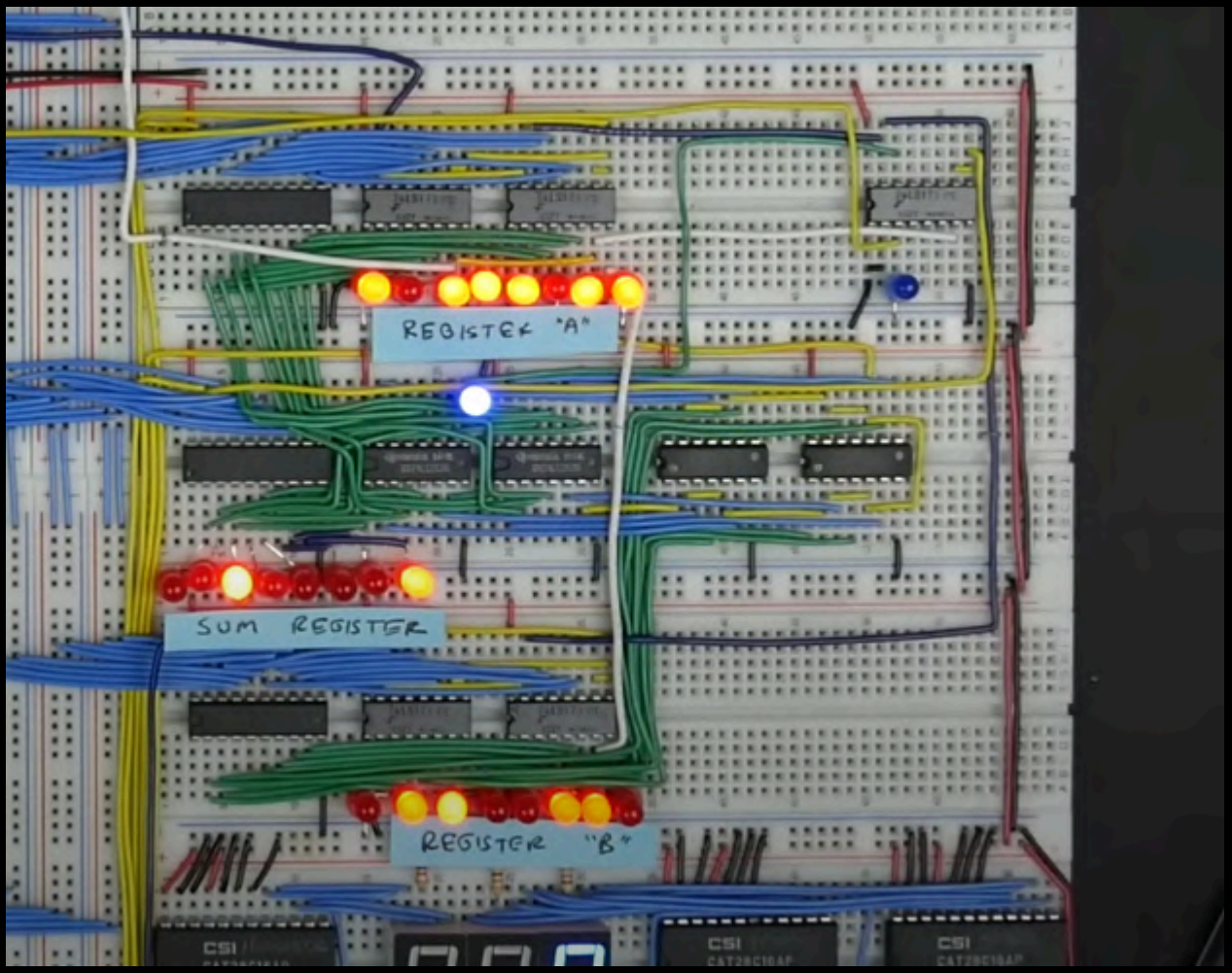
#4 Clock logic



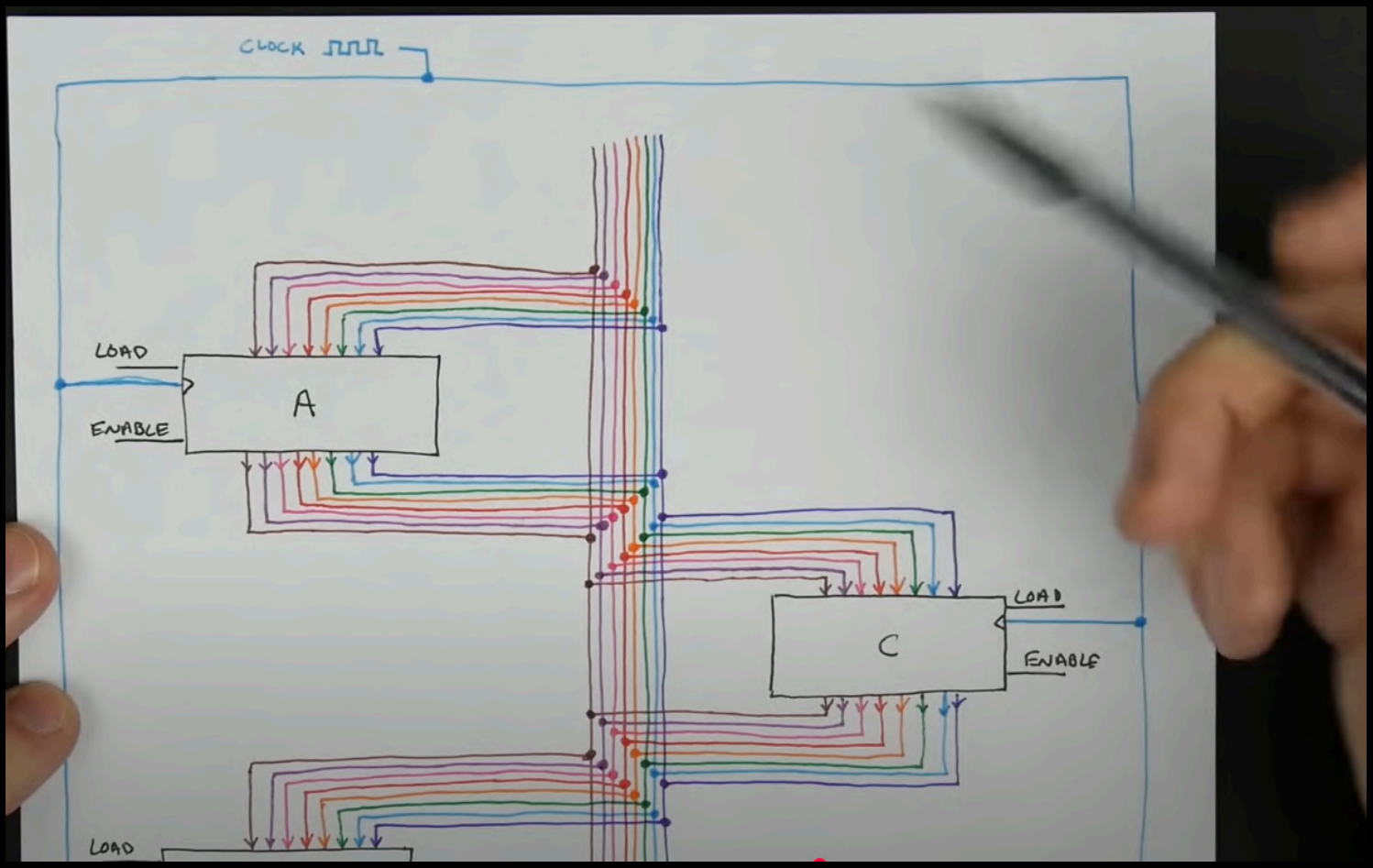

Astable 555 timer
Monostable 555 timer
Bistable 555
5V
VCC
U1
U2
U3
R1
1k
R2
1k
RV1
1M
R3
220
D1
LED-GREEN
C1
1uF
C2
0.01uF
555
GROUND
R4
1M
R5
1k
C3
0.1uF
C4
0.01uF
D2
LED-GREEN
R6
220
R7
1k
R8
1k
C5
0.01uF
SW1
SW-SPDT
D3
LED-GREEN
R9
220
U4:A
U4:B
U4:C
74LS08
U5:A
74LS32
U6
U7
U8
NOT
HLT
Clock bar
Clock
D4
LED-GREEN
R10
220



#5 Registers



74LS245 (https://www.futurlec.com/74LS/74LS245.shtml)
74LS173 (https://www.alldatasheet.com/datasheet-pdf/view/27388/TI/74LS173.html)

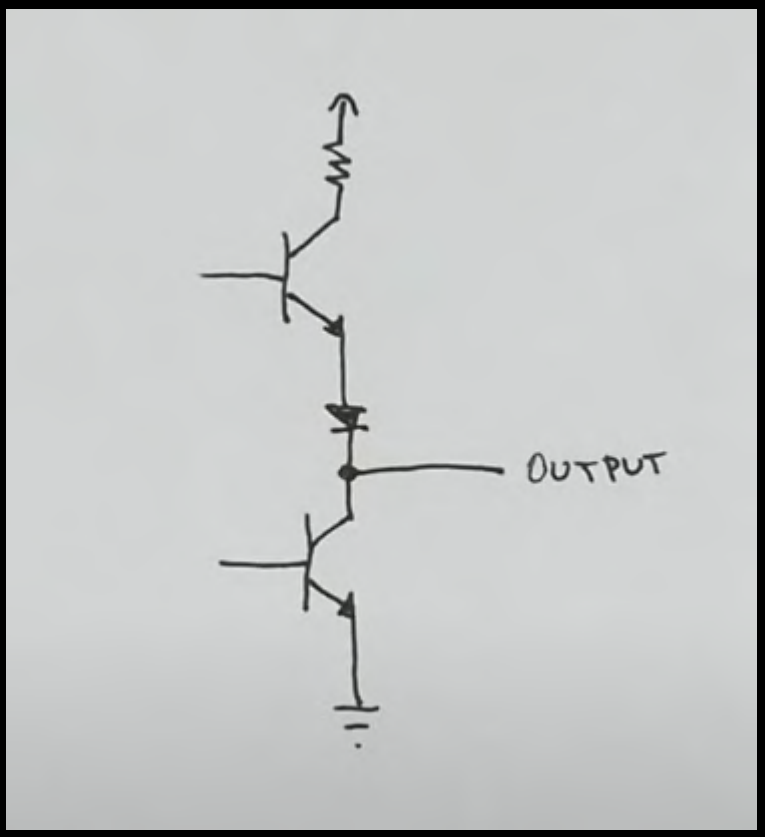


Bus: Collection of wires which act as common connection points for multiple components within the computer. Since, each bit is put on a wire, an 8 bit bus would imply 8 wires.
In short, the bus moves data.

Registers have control lines that tell it when to read from the bus and when to put data on the bus. Enable signal tells a register to put data on the bus, while the load signal tells the register to read from the bus. These two control signals are present for each register.

#6 Tri-state logic: Connecting multiple outputs together
Purpose: To organize outputs from multiple modules on the same bus and prevent any one module interfering with any other.

Three states: 1, 0, and not connected state (not connected state is aka not enabled, high impedance high-Z state, isolated state, disconnected state).

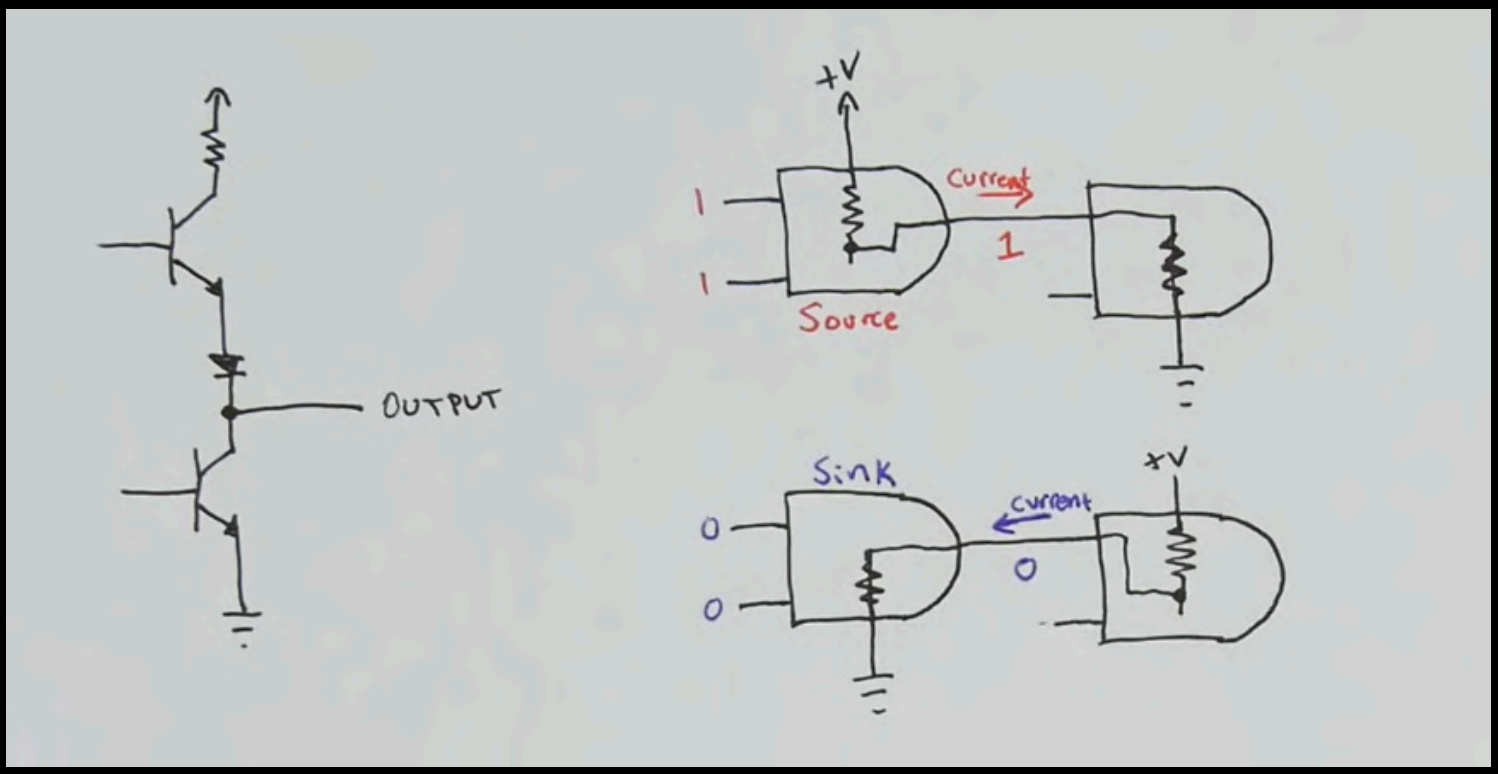


Usual connection at the output of a logic gate after some operation

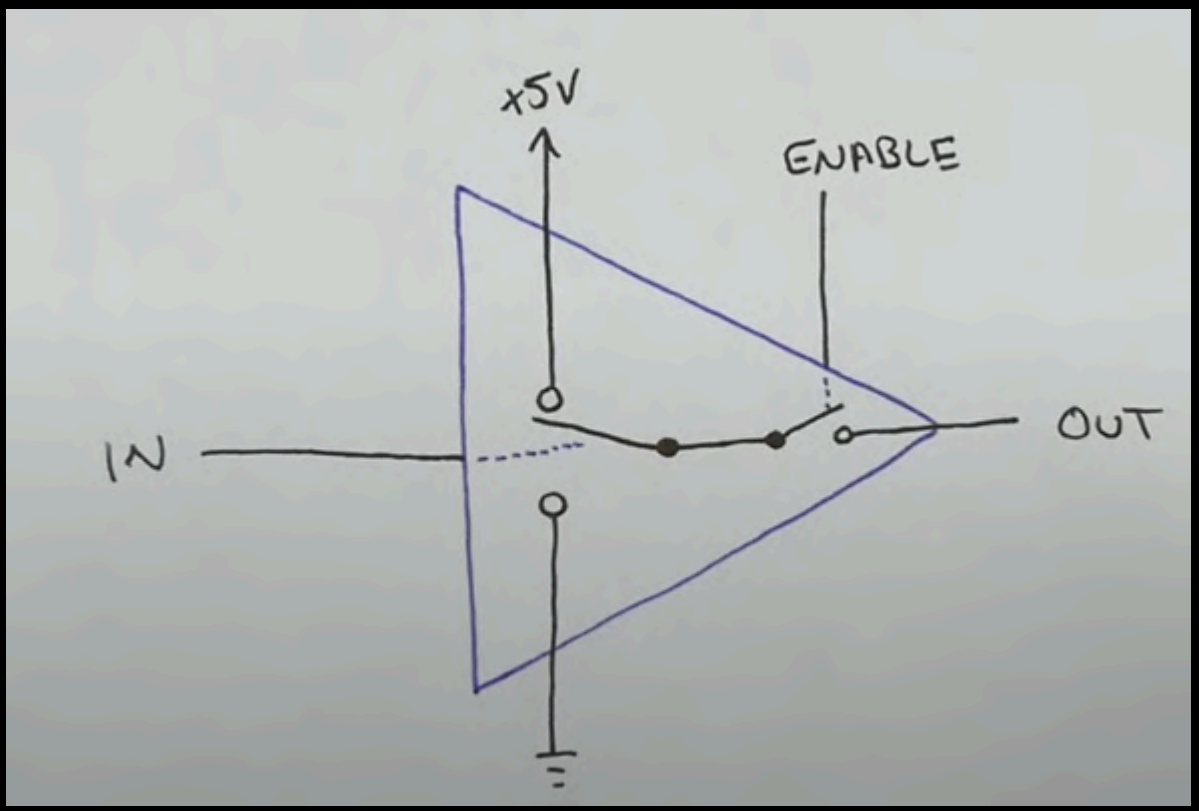
If the upper transistor is ON and the lower transistor is OFF, then the output will source current from the source.

If the upper transistor is OFF and the lower transistor is ON, then the output will sink current to the ground.

The disconnected state (turning both transistors OFF) comes into picture when one bit from a module is sourcing and simultaneously if some bit on some other module connected on the same bus is sinking current then, we get some unwanted connection. Basically, we would want one module isolated with another, which is obtained by using disconnected state. This state neither sources nor sinks current.



Sourcing and sinking current.



The above mentioned goal (of sourcing and sinking current) can be obtained by this circuit (Three state gate 74LS245) as well.

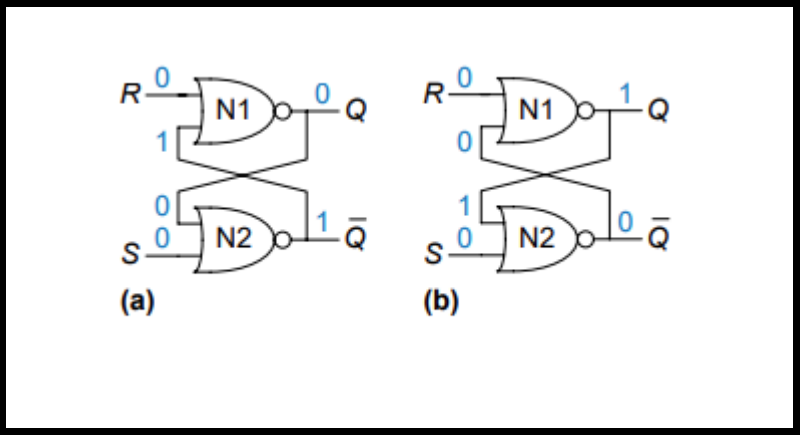
If IN=1, the OUT is connected to 5V. (ENABLE=high)
If IN=0, the OUT is connected to ground. (ENABLE=high)

If ENABLE =low, the OUT is disconnected.

#7 Designing and building a 1 bit register

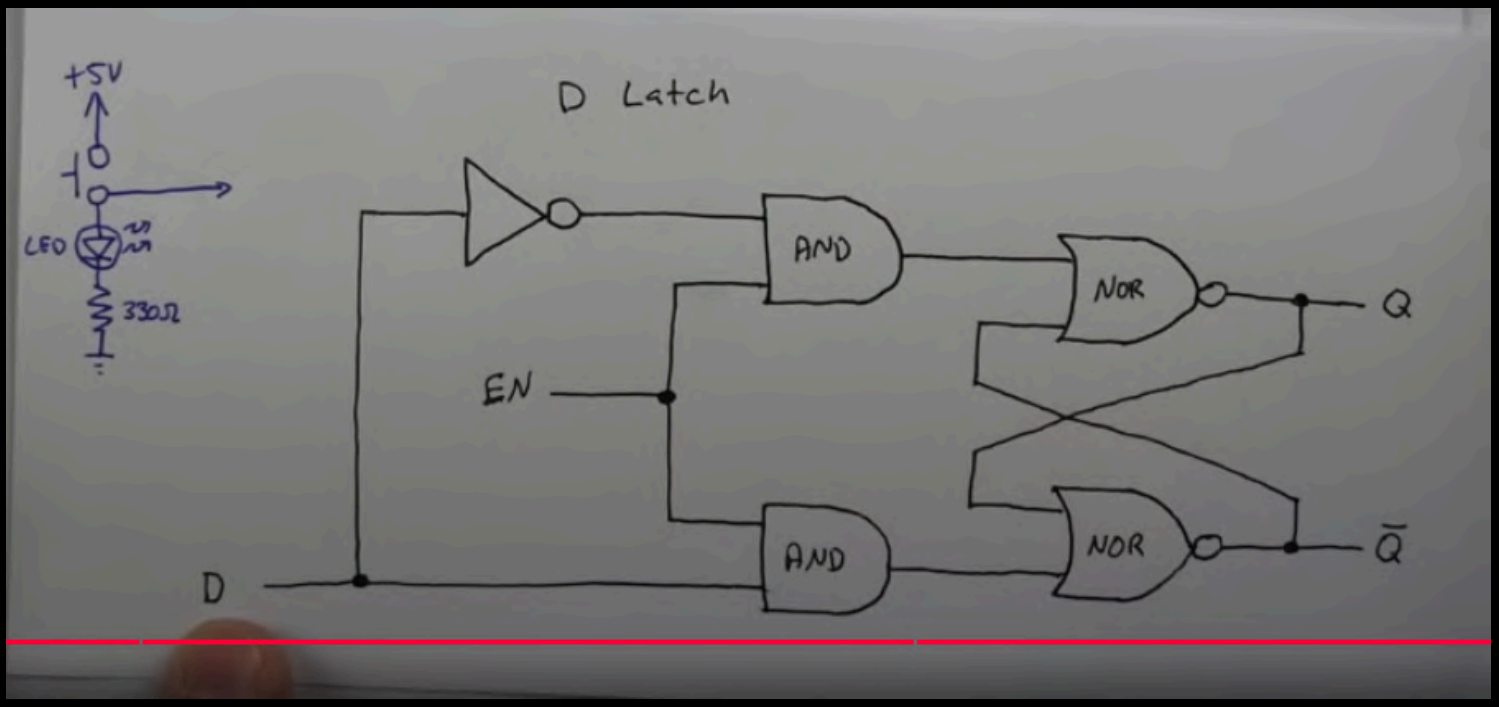
Let us first review Latches and Flip Flops.

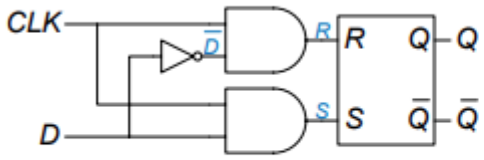
SR-Latch: Stores 1,0. No involvement of clock.



(Book: Harris and Harris: Digital Design and Computer Architecture, 2nd edition)

D-Latch: (Clock (Level Triggered)+SR-Latch). The latch is transparent during the positive level of the clock signal.

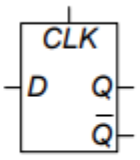




(a)

CLK	D	$\bar{D}$	S	R	Q	$\bar{Q}$
0	X	$\bar{X}$	0	0	Q_{prev}	$\bar{Q}_{prev}$
1	0	1	0	1	0	1
1	1	0	1	0	1	0

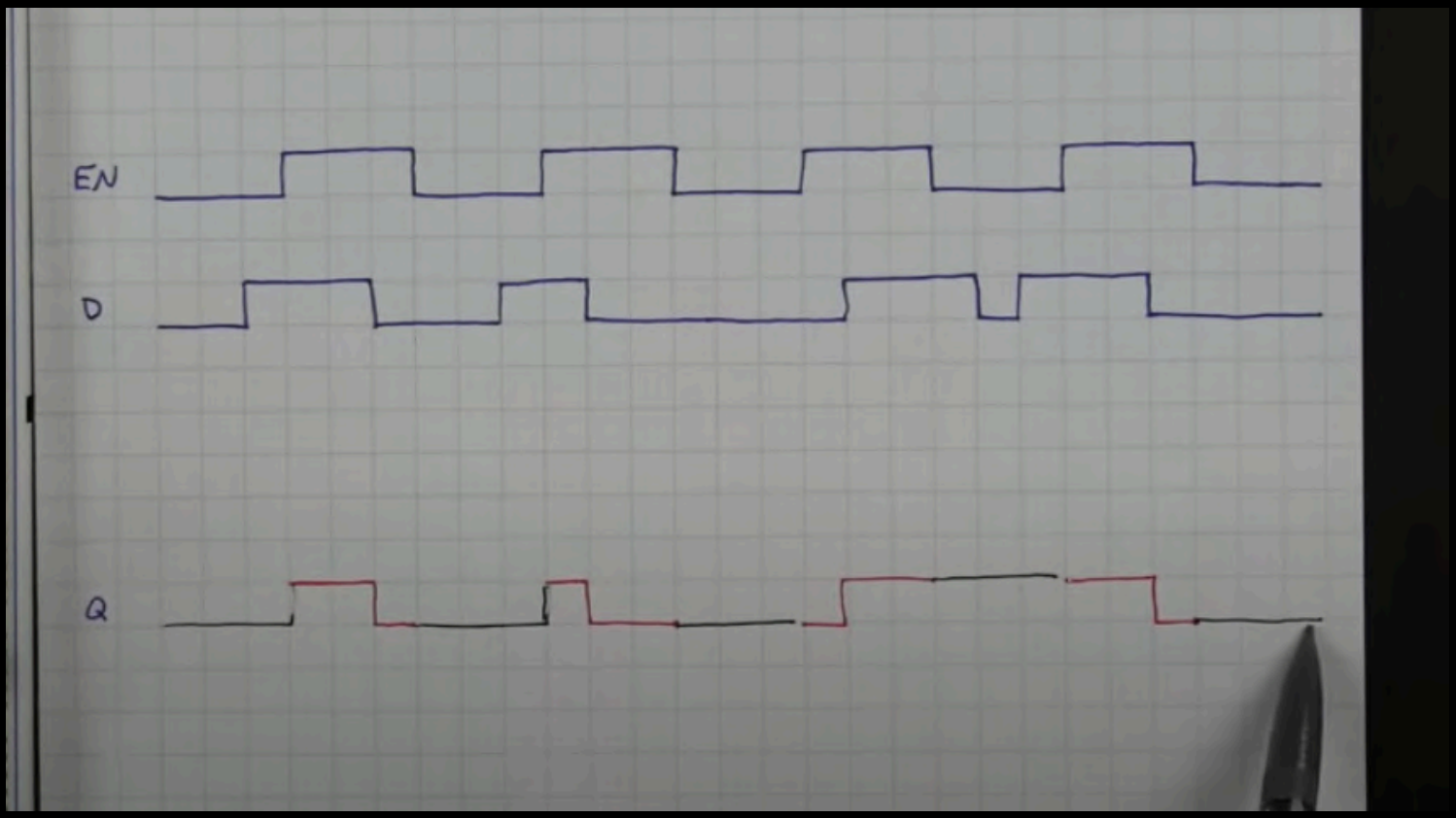
(b)



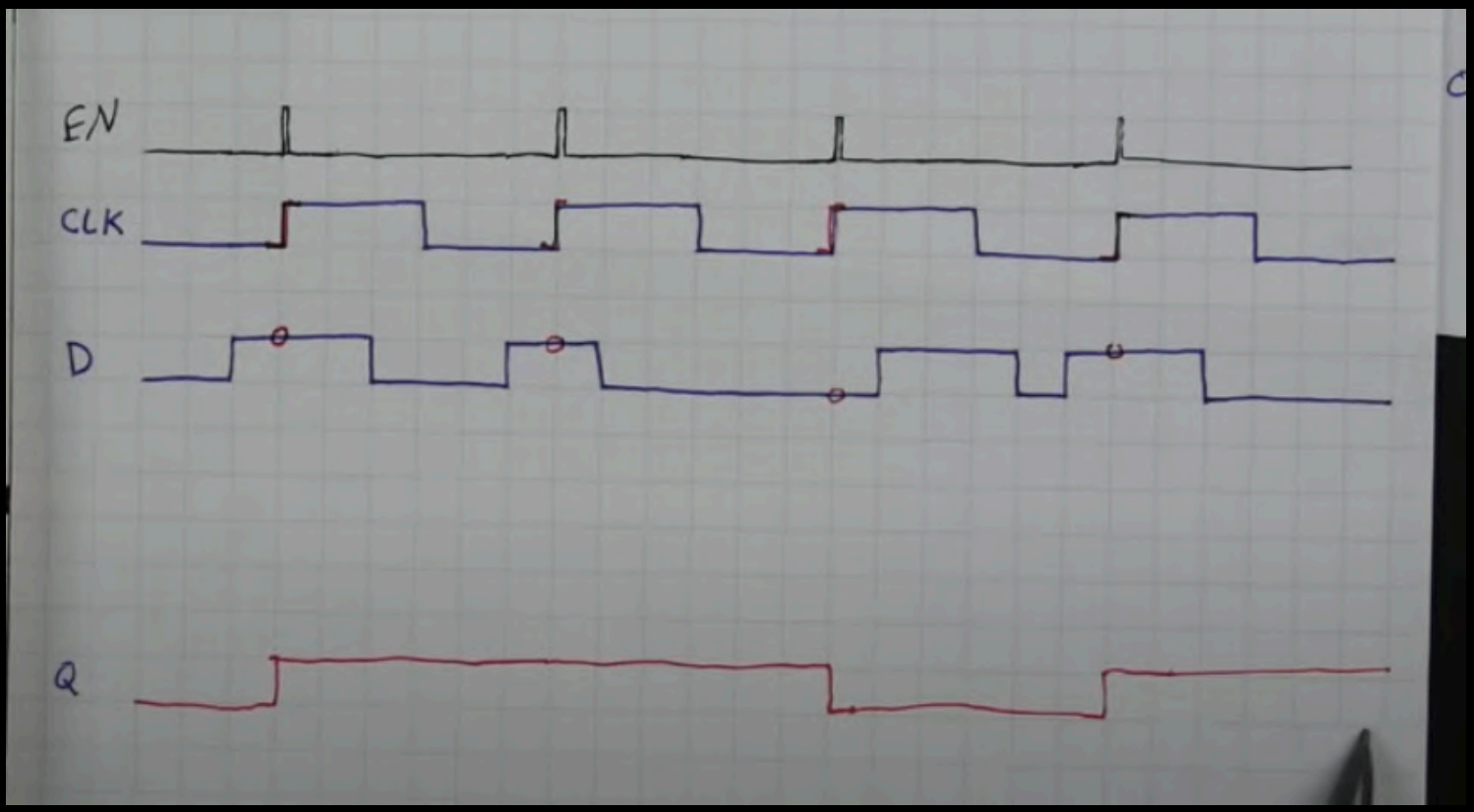
(c)

Figure 3.7 D latch: (a) schematic, (b) truth table, (c) symbol

(here EN = CLK)



D-Flip-Flop: It is a type of D-Latch but it is edge triggered (generally positive edge triggered).



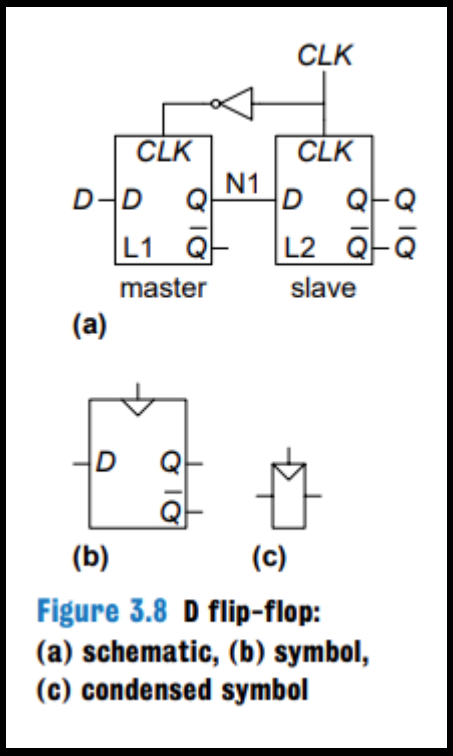
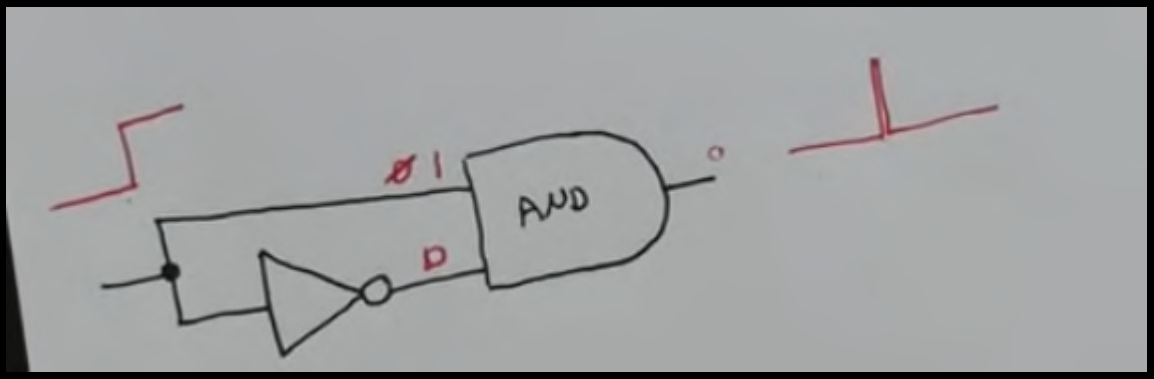


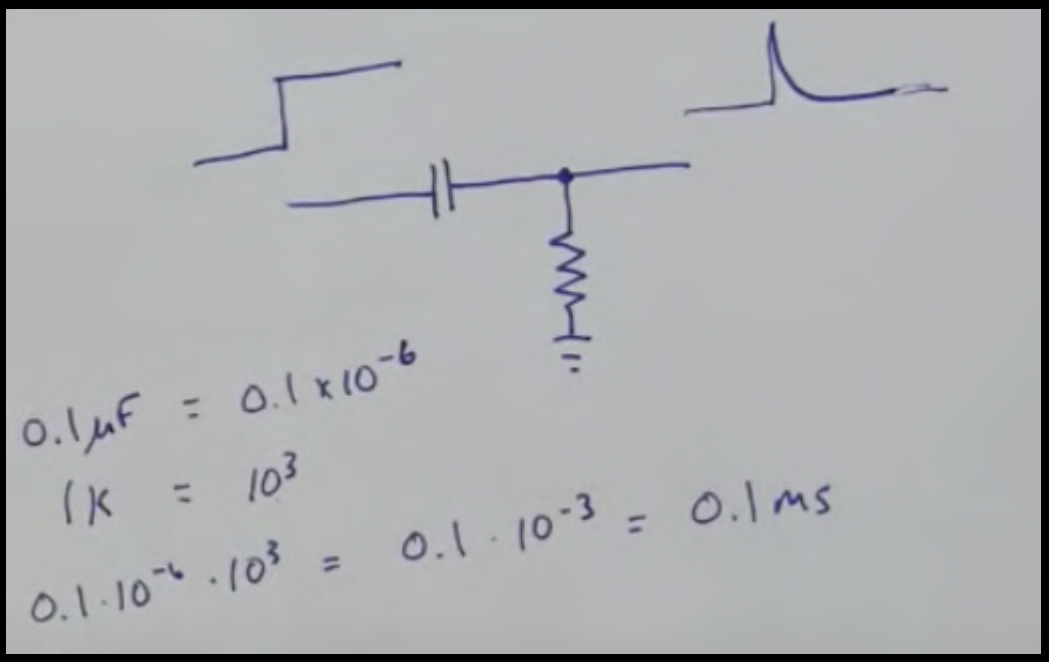
Figure 3.8 **D flip-flop: (a) schematic, (b) symbol, (c) condensed symbol**

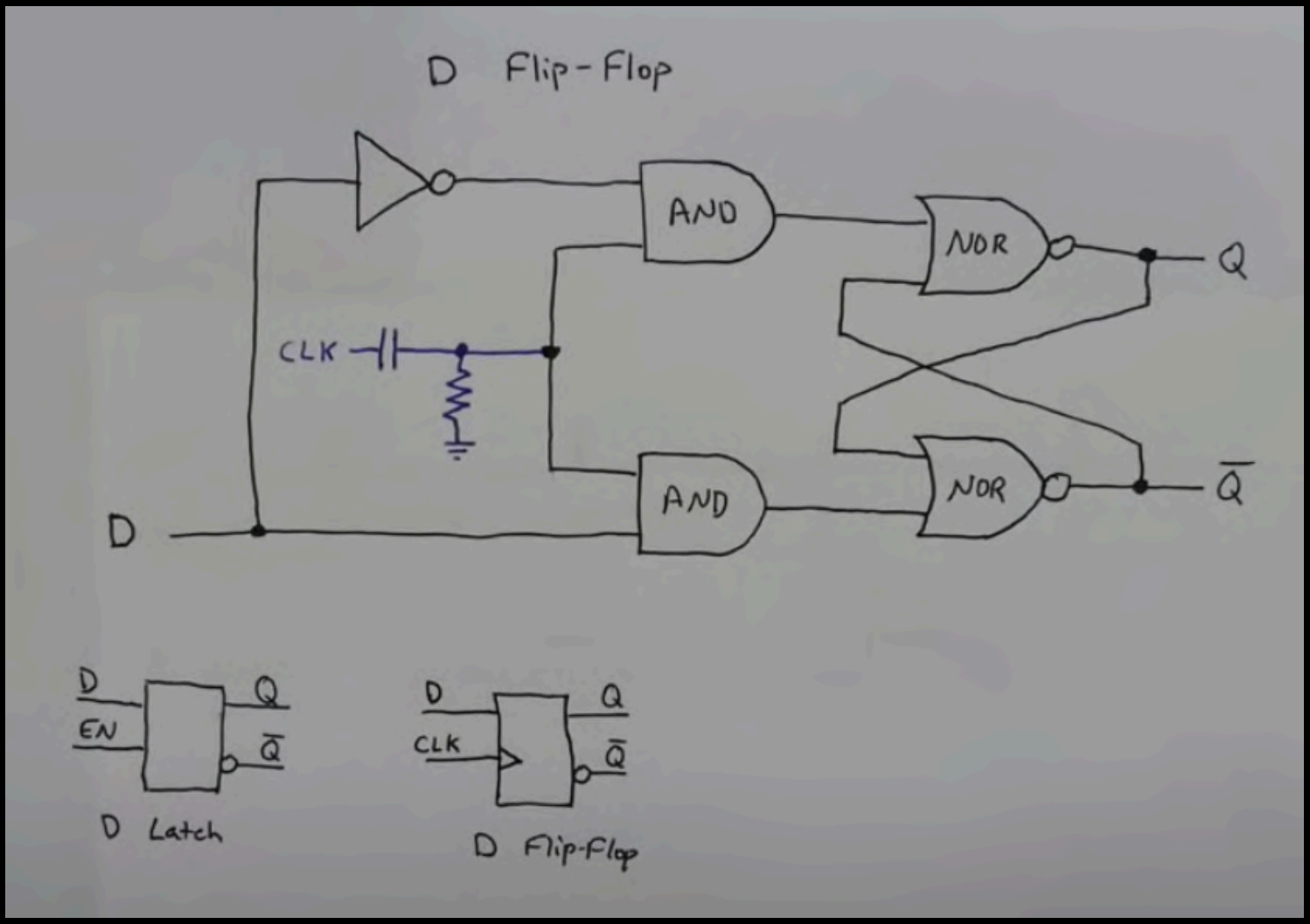
Now, to obtain this edge triggering:

Method 1 (Using propagation delay of NOT gate, we can cascade 3 to 5 NOT gates):



Method 2 (Using Resistor and Capacitor):





Method 3: We can cascade two D-Latches to form one D-Flip-Flop.

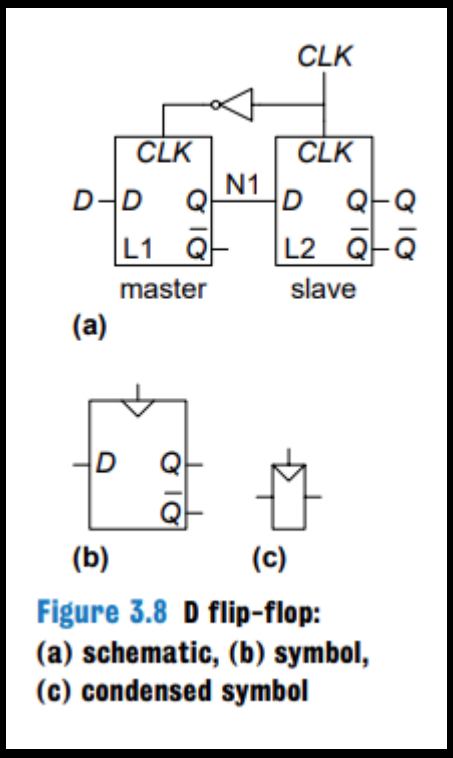
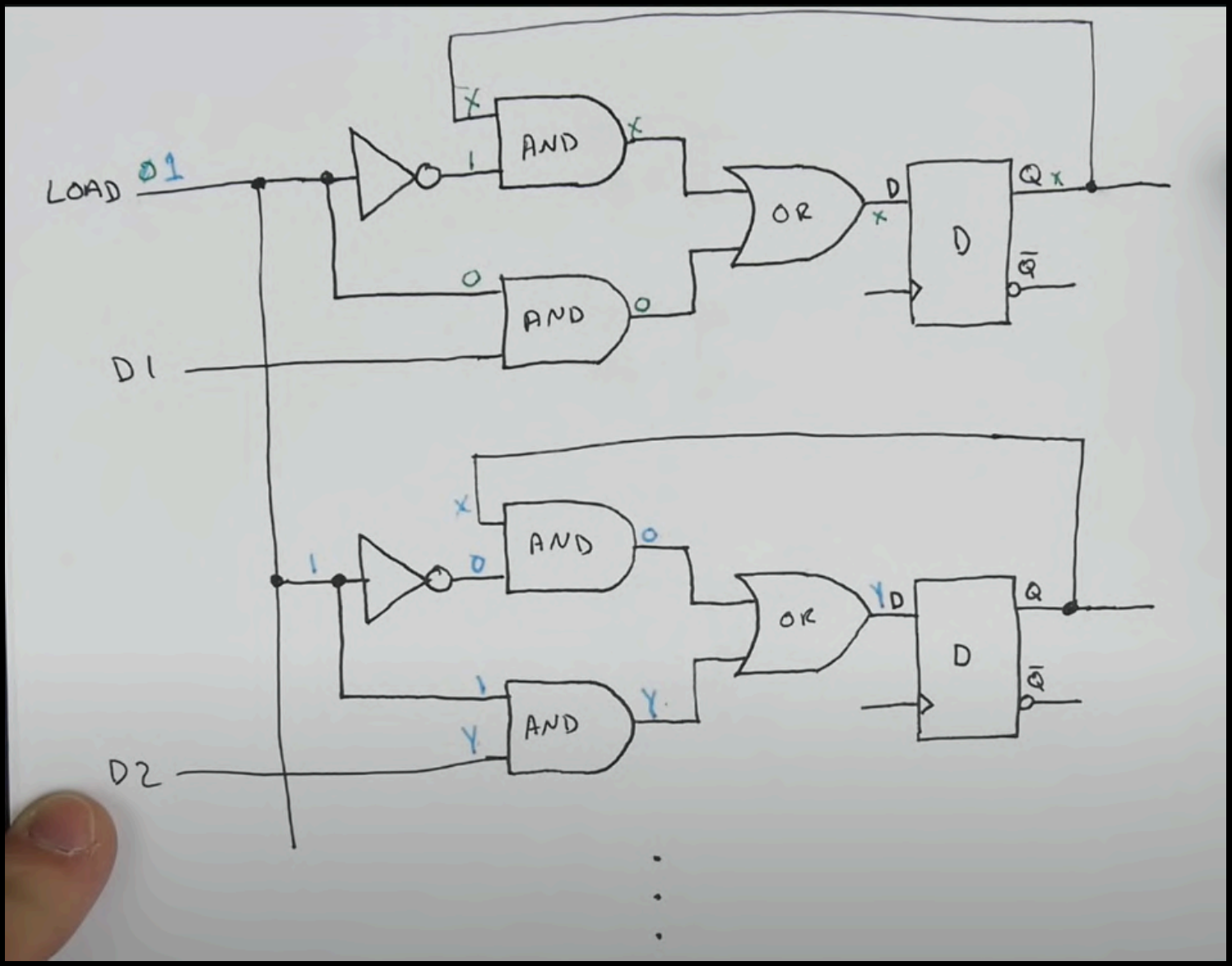
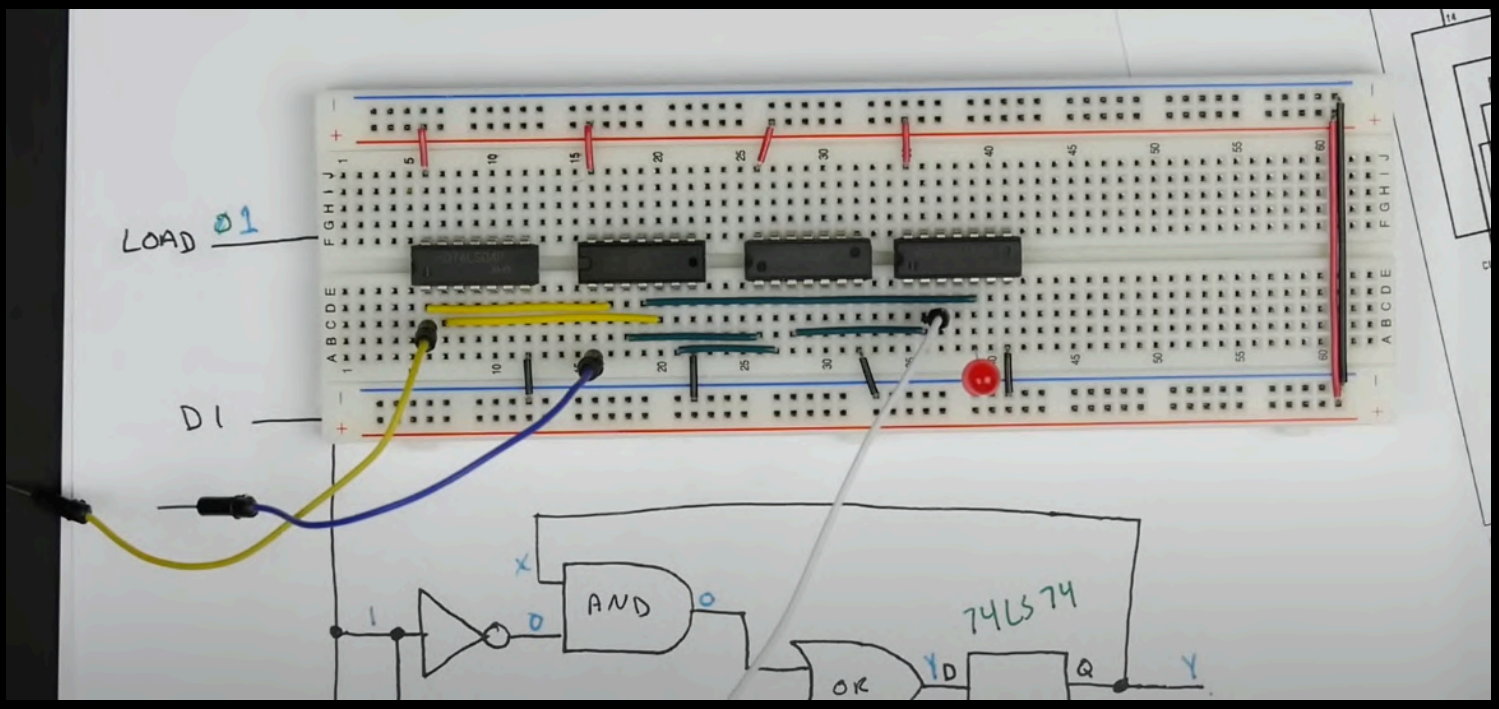


Figure 3.8 **D flip-flop: (a) schematic, (b) symbol, (c) condensed symbol**

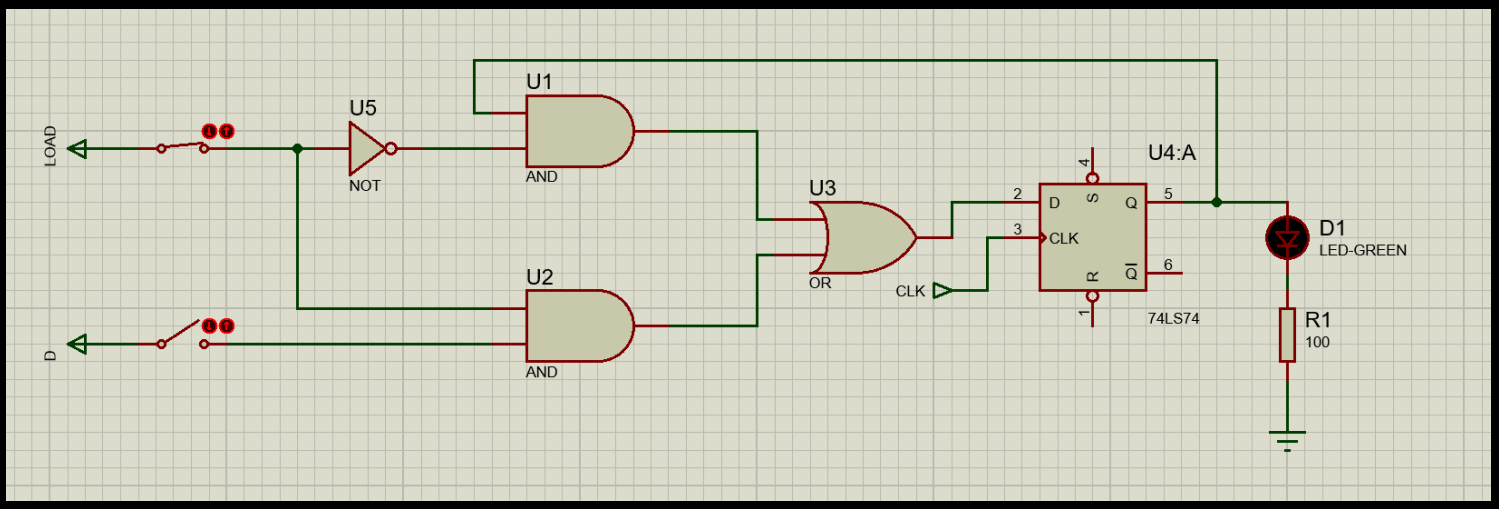
Now, to store bits, we use Flip-Flops. One Flip-Flop stores one bit.

Also, flip-flops, natively, are volatile memory, i.e. they store the bits till the power is supplied. For non-volatile memory, like SSDs, there is usually some mechanism employed to trap in electrons, this is some serious science, really. Like, orchestrating electrons, literally.



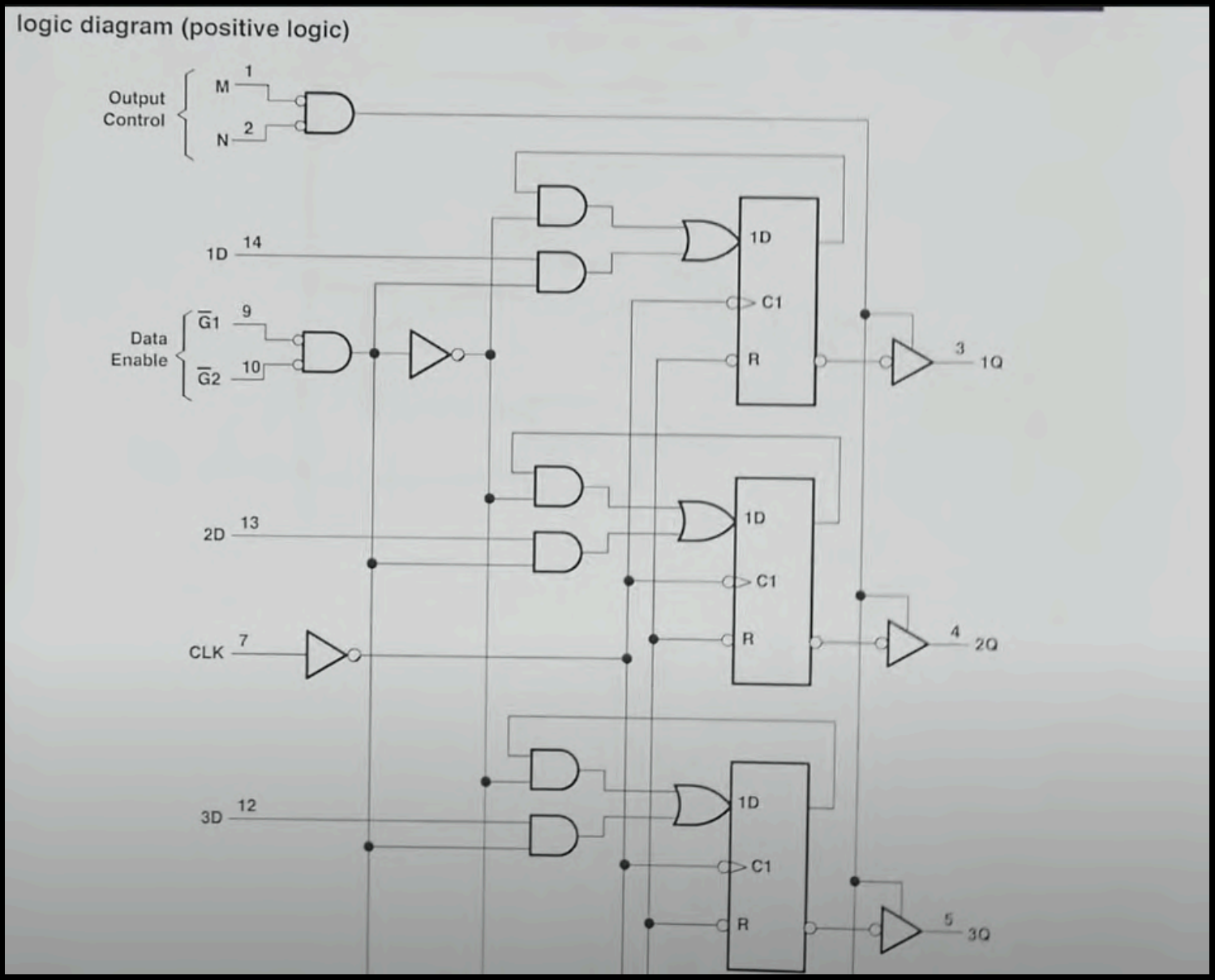


Implementation in Proteus (Clock signal is sourced from clock, Proteus somehow allows wirelessly connect wires (software wizardry, whatever, it saves us some wiring job though)



D Flip-Flop (74LS74A)
INVERTER (74LS04)
AND (74LS08)
OR (74LS32)
TRI-STATE BUFFER (74LS245)

REGISTER (74LS173) [4-BIT D-TYPE REGISTER)

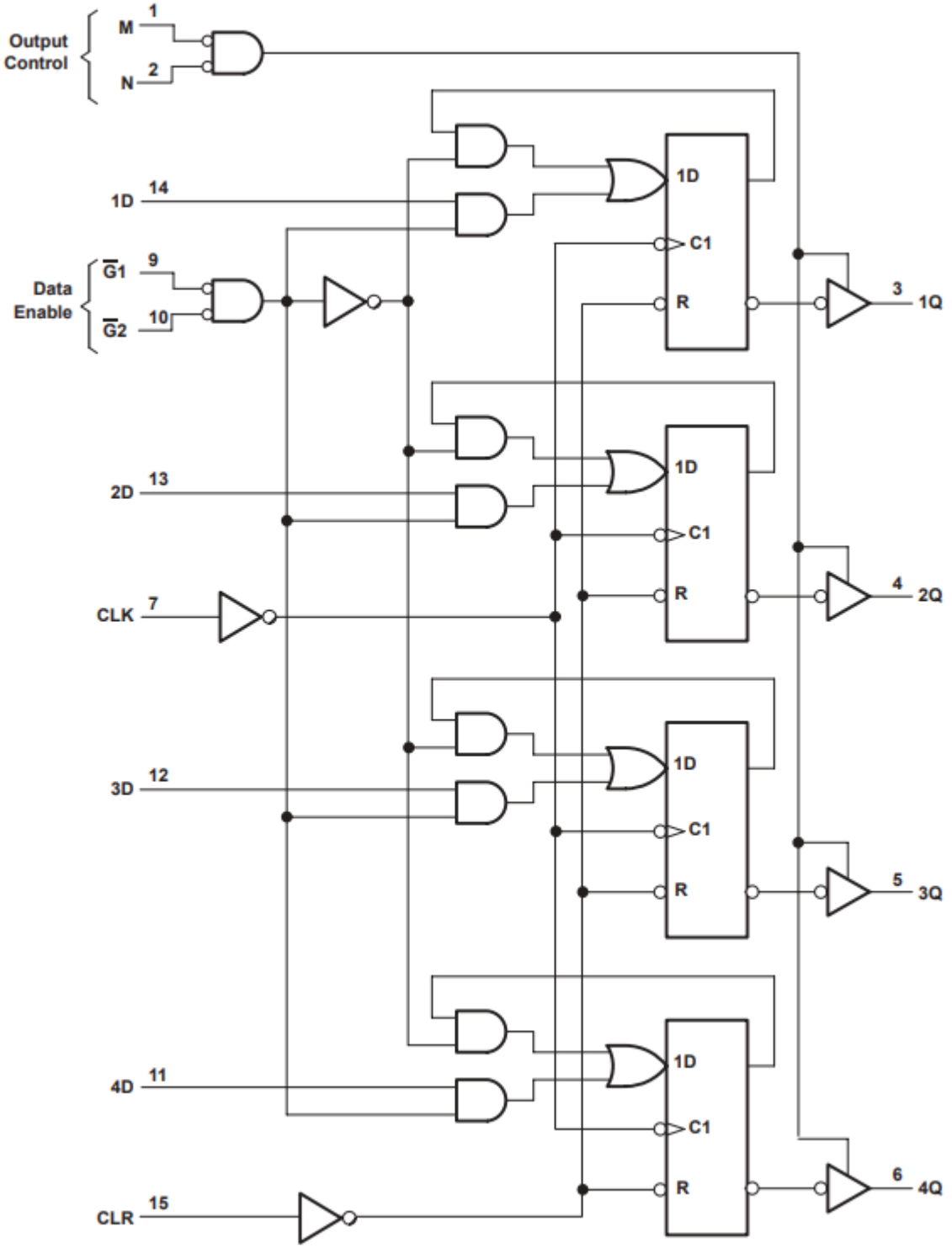


#8 Building an 8-bit register

We will be using 74LS173 to store 4-bits, and in this chip, we will keep output control always ON so that we will be able to see the content of the register. And, we will use 74LS245 TRI-STATE BUFFER to control when to output data onto the bus.

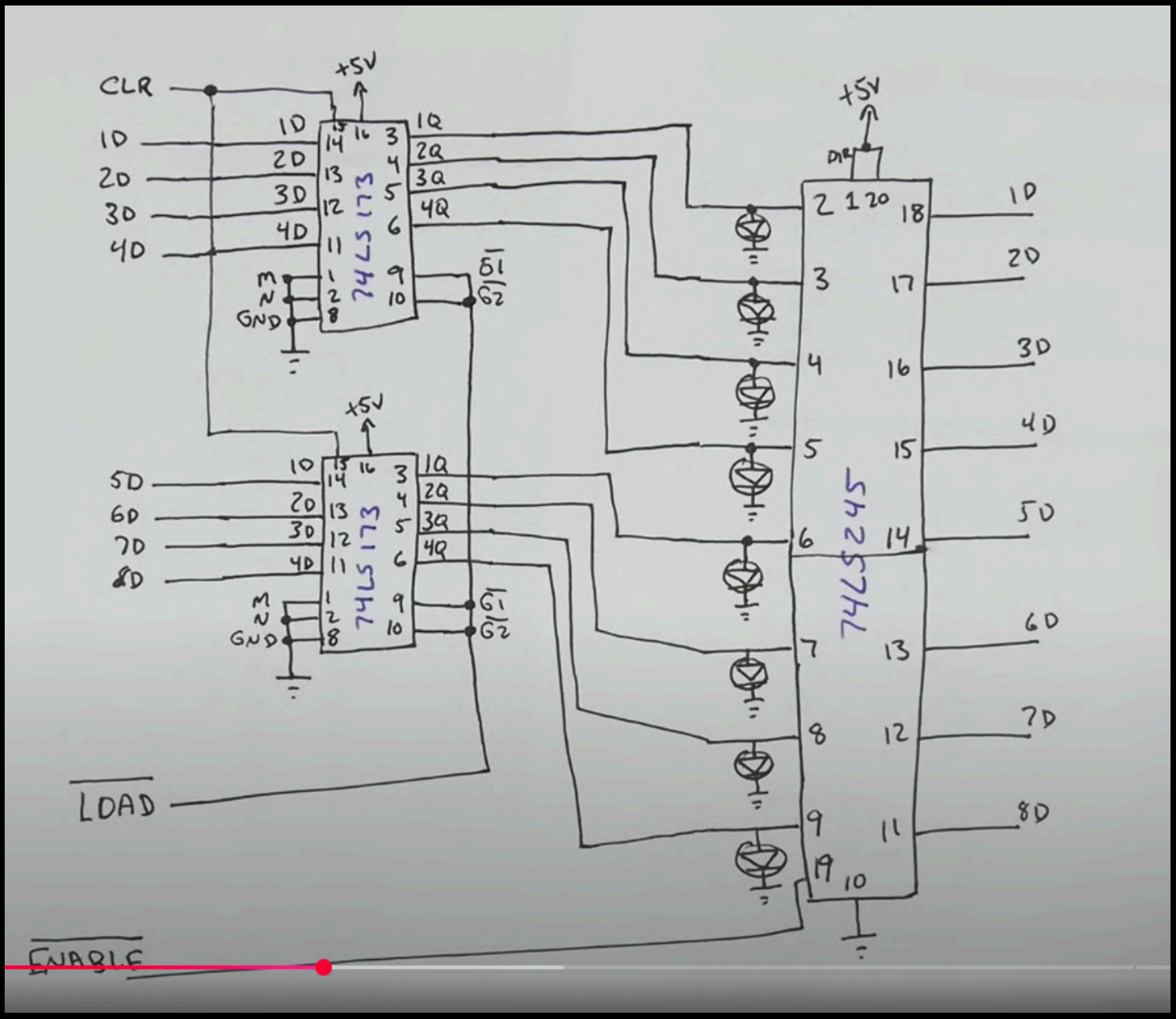
74LS173

logic diagram (positive logic)

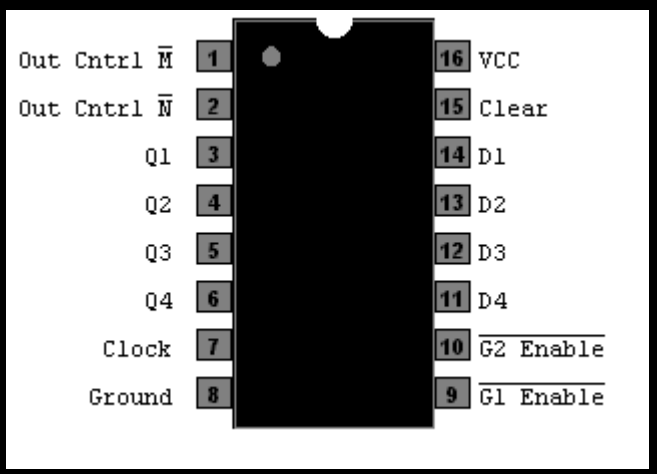


Pin numbers shown are for D, J, N, and W packages.

In 74LS245, DIR is set to +5V so that data movement is from left to right.
Between Load and Enable, only one is used at a time.

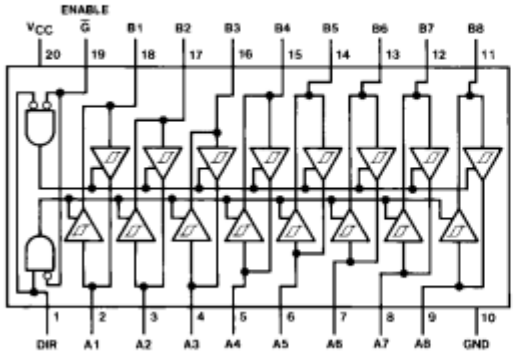


74S173



74LS245

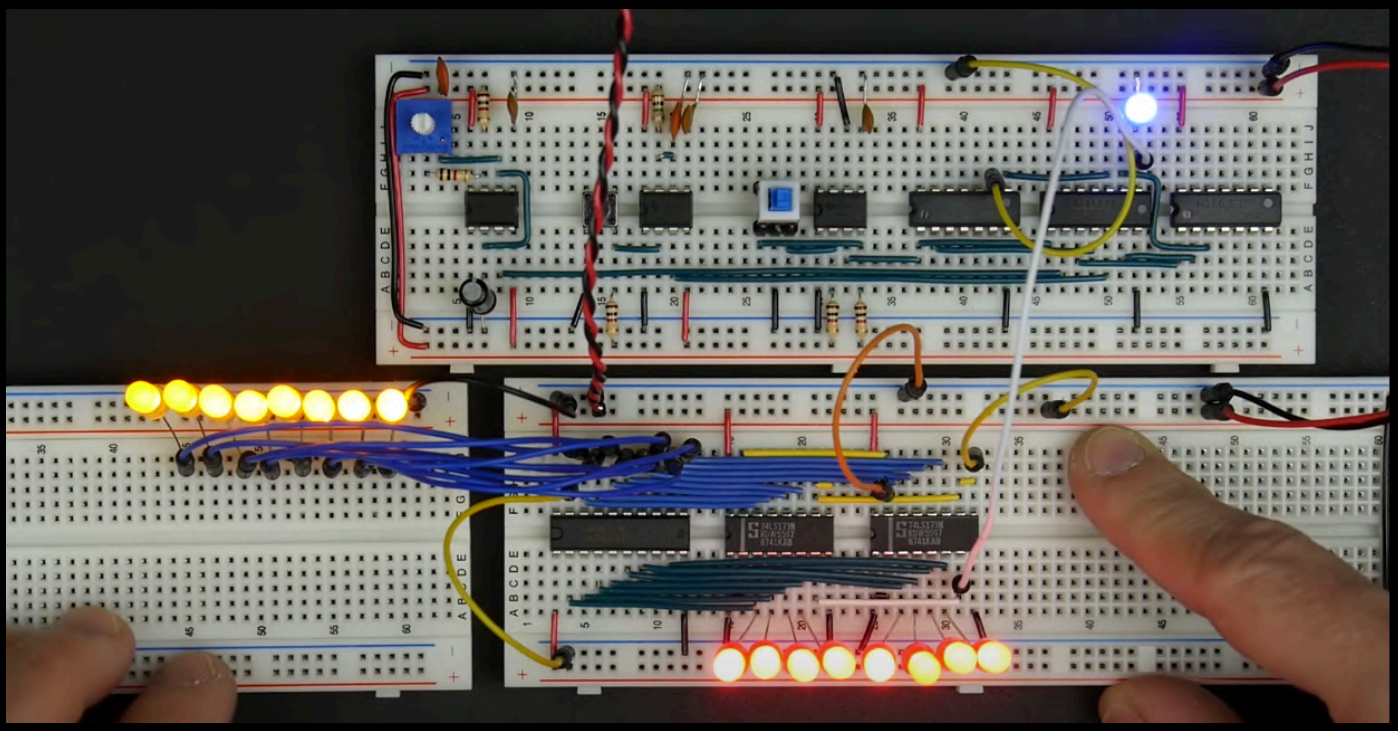
Connection Diagram



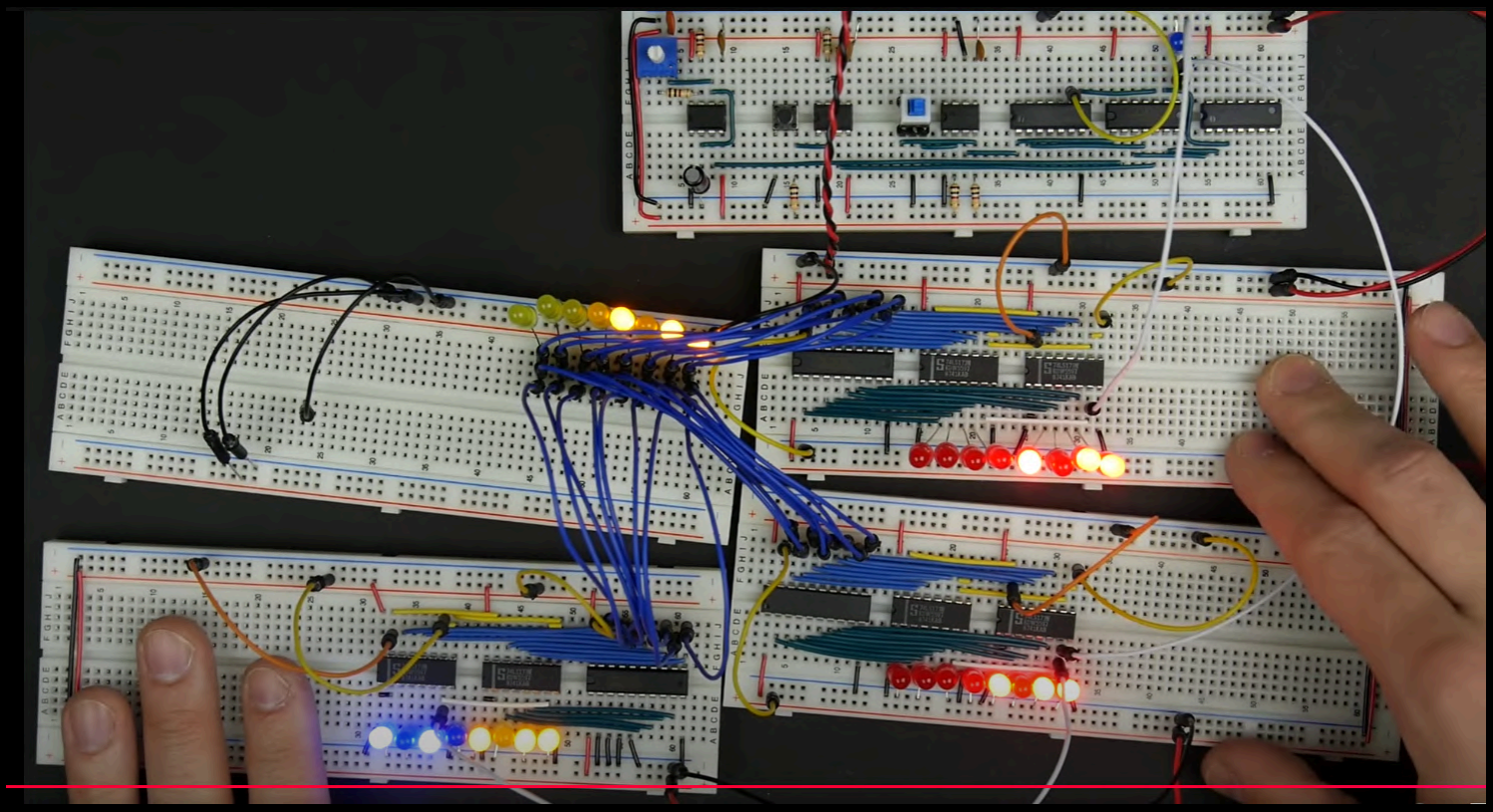
Function Table

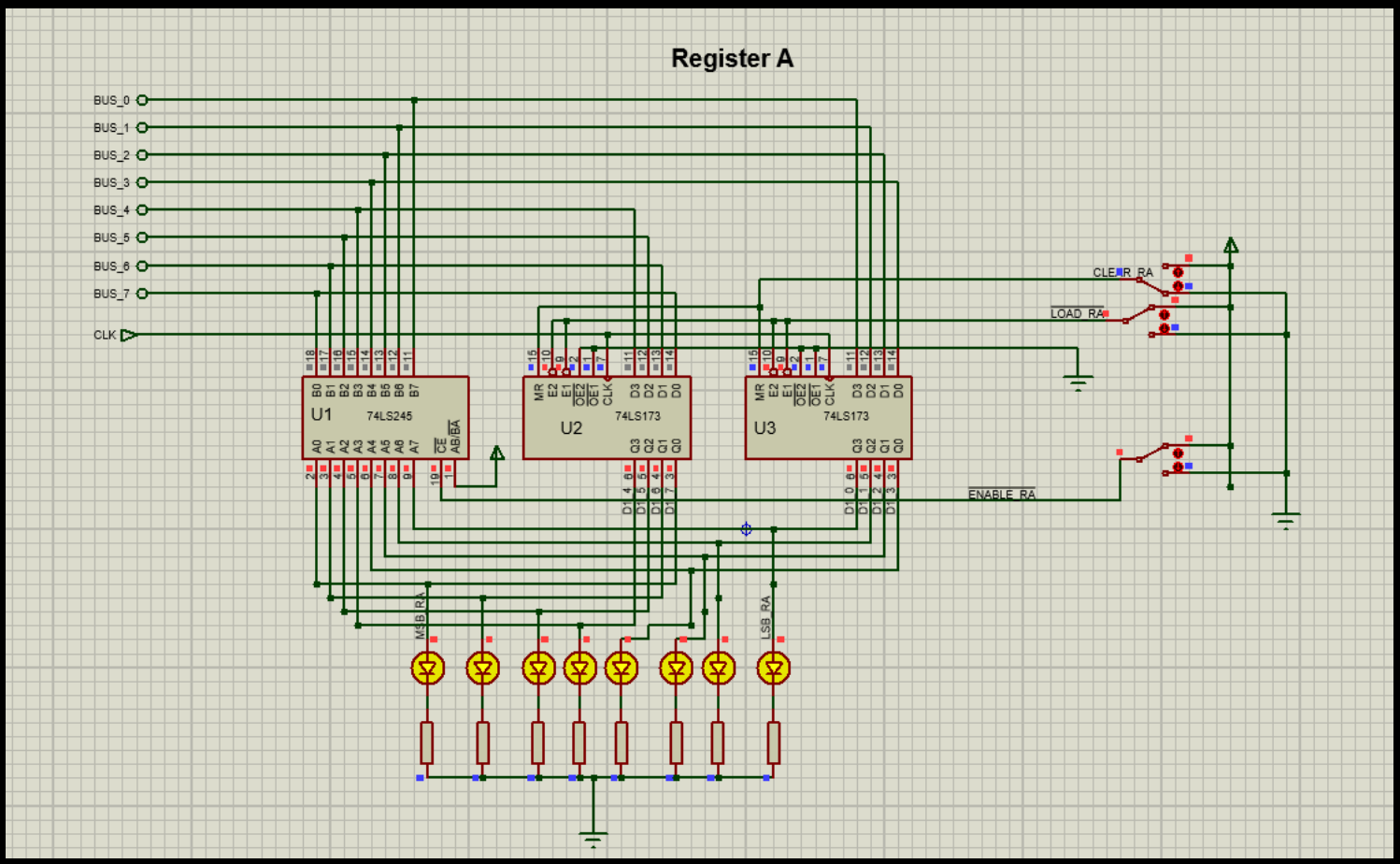
Enable $\overline{G}$	Direction Control DIR	Operation
L	L	B Data to A Bus
L	H	A Data to B Bus
H	X	Isolation

H = HIGH Level
L = LOW Level
X = Irrelevant

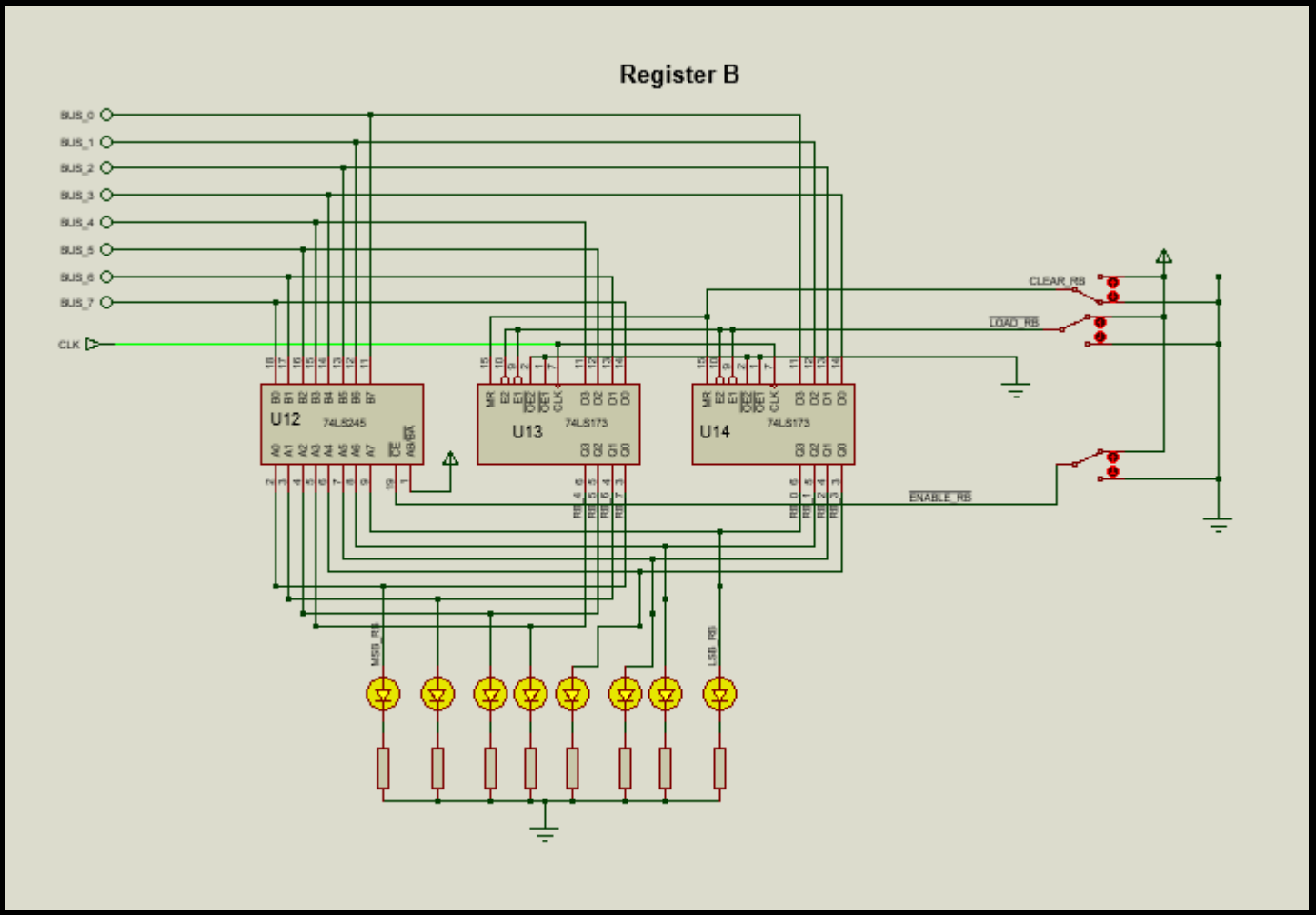


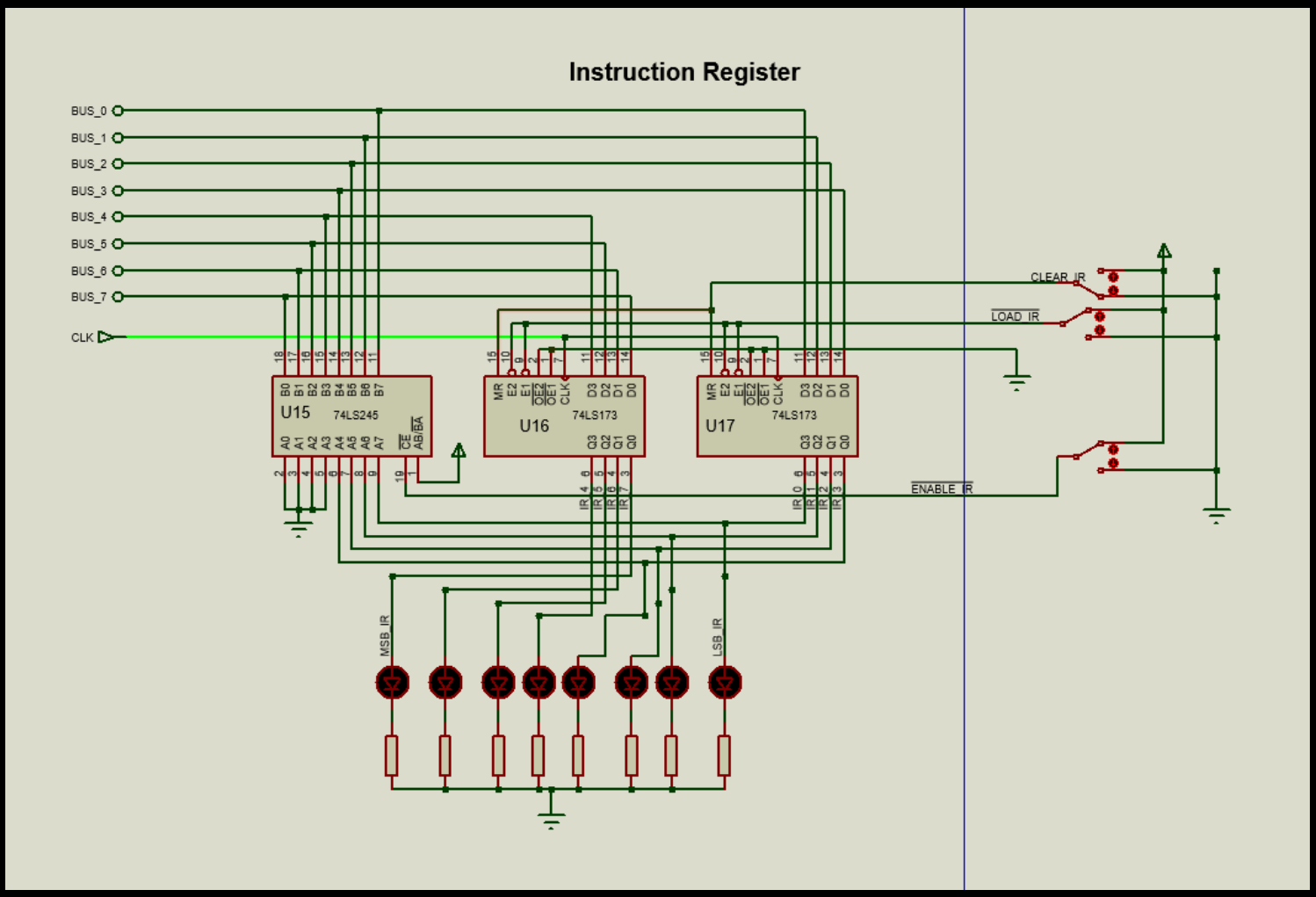
Modules: Register A, Register B, Instruction Register (for this register 4MSBs, the blue ones, are not connected as output to the bus), Clock.




Register A
BUS_0
BUS_1
BUS_2
BUS_3
BUS_4
BUS_5
BUS_6
BUS_7
CLK
U1
74LS245
U2
74LS173
U3
74LS173
CLEAR_RA
LOAD_RA
ENABLE_RA
MSB_RA
LSB_RA

Turns out, wires with the same label are connected internally, so we need to avoid labelling the same in two different wires.





We have three 8-bit Registers; Register A, Register B, and Instruction Register.

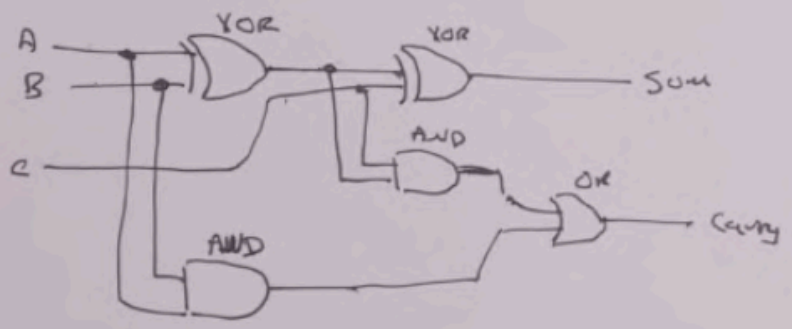
#9 Arithmetic logic unit (ALU)

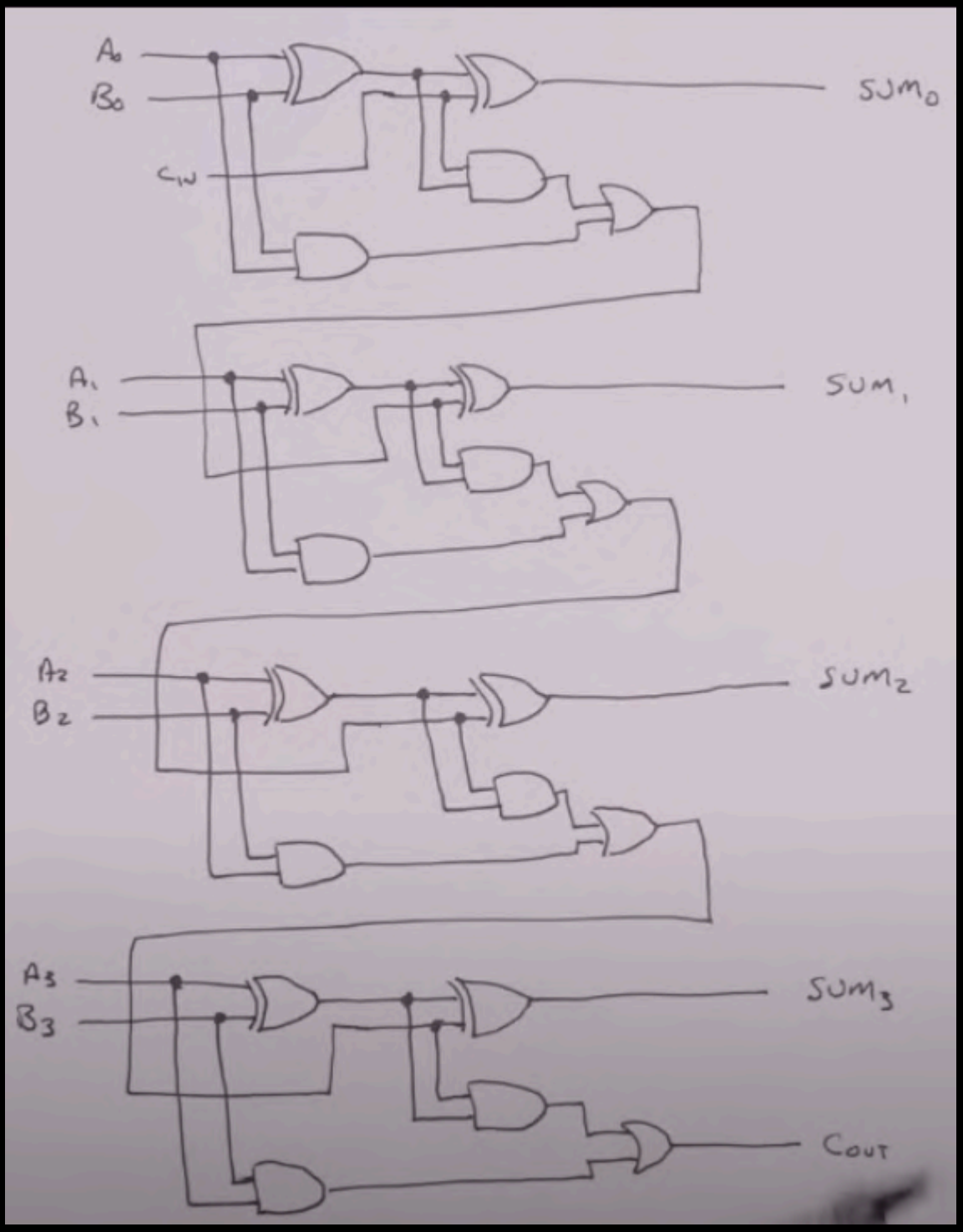
While an ALU is usually capable of performing multiple arithmetic, bitwise and comparison operations, for our relatively simpler CPU, the ALU can just add and subtract. It's connected to Registers A and B, and outputs either A+B or A-B.

$$\begin{array}{r} {\scriptstyle 1} \\ 28 \\ 22 \\ \hline 50 \end{array}$$

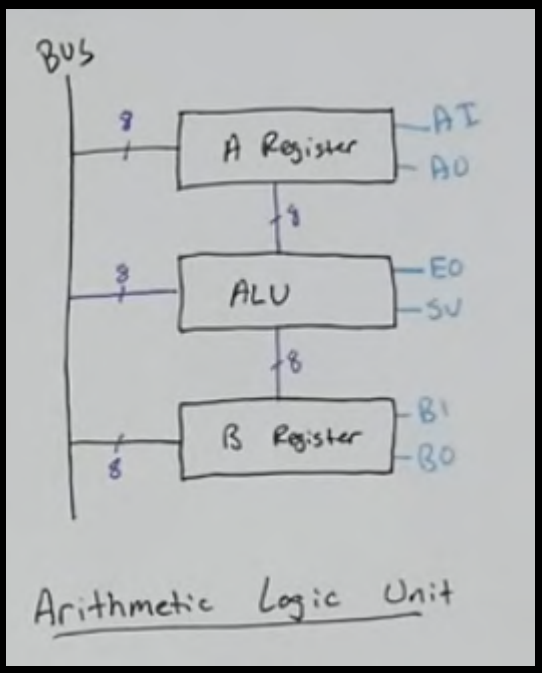
$$\begin{array}{r} {\scriptstyle 1\ 1} \\ 11100 \\ 10110 \\ \hline 110010 \end{array}$$

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



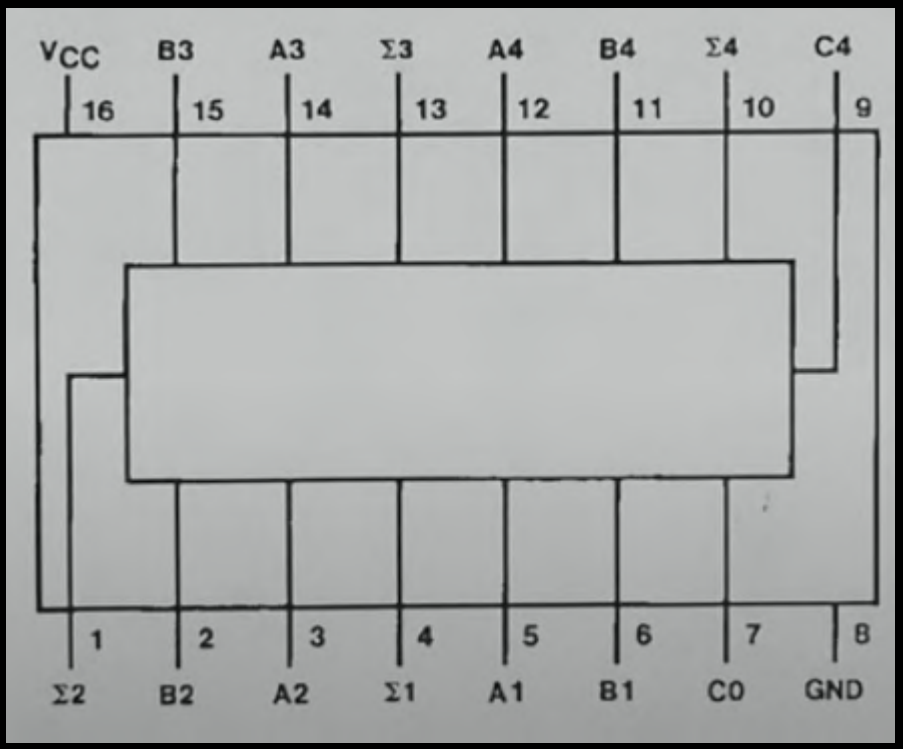


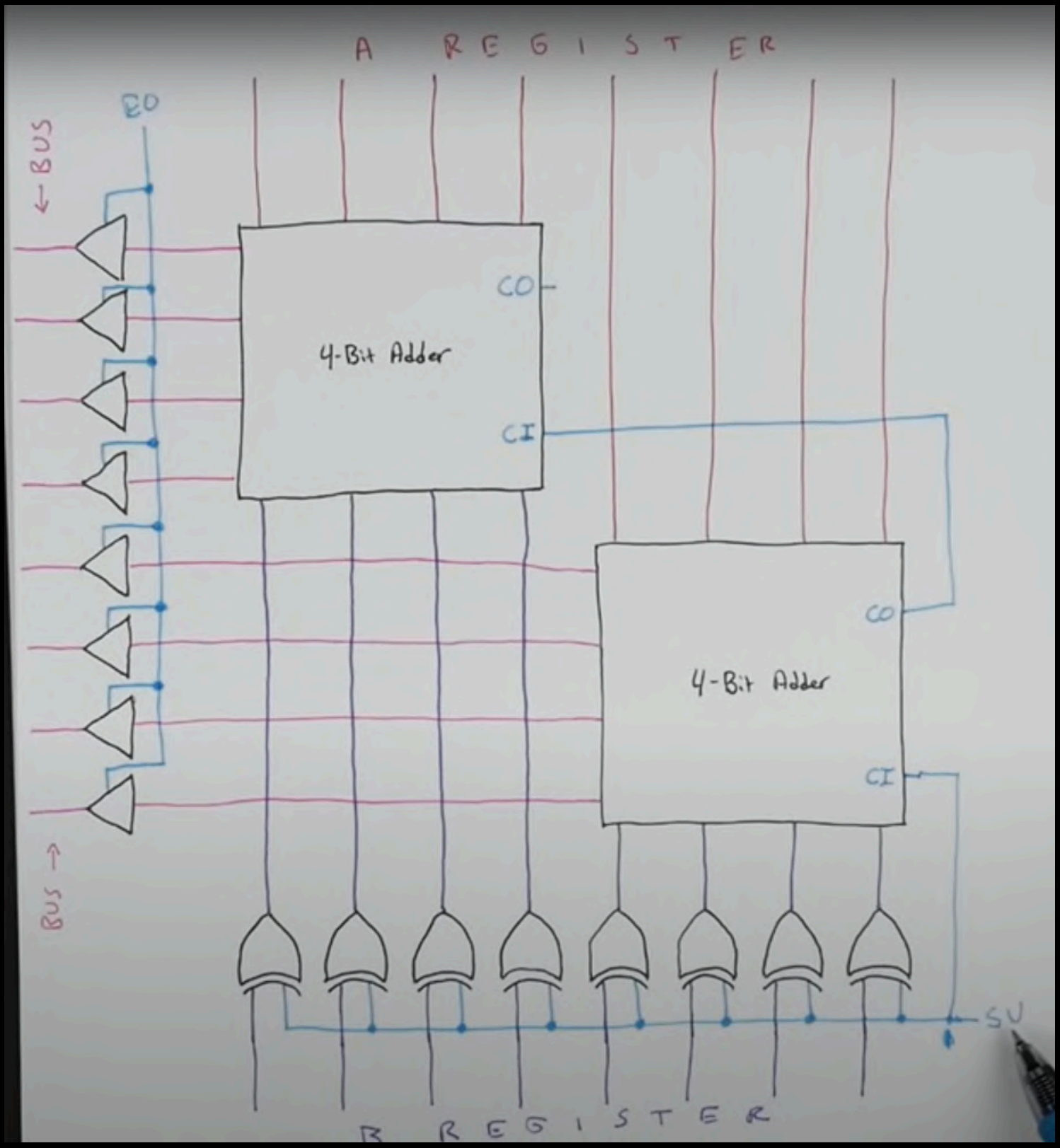
#10 ALU design



When the EO signal is active, the output is forwarded to the BUS. When SU signal is inactive, the ALU performs addition and when SU signal is active, the ALU performs subtraction (2's complement).

4-Bit Binary Adder (74LS283)

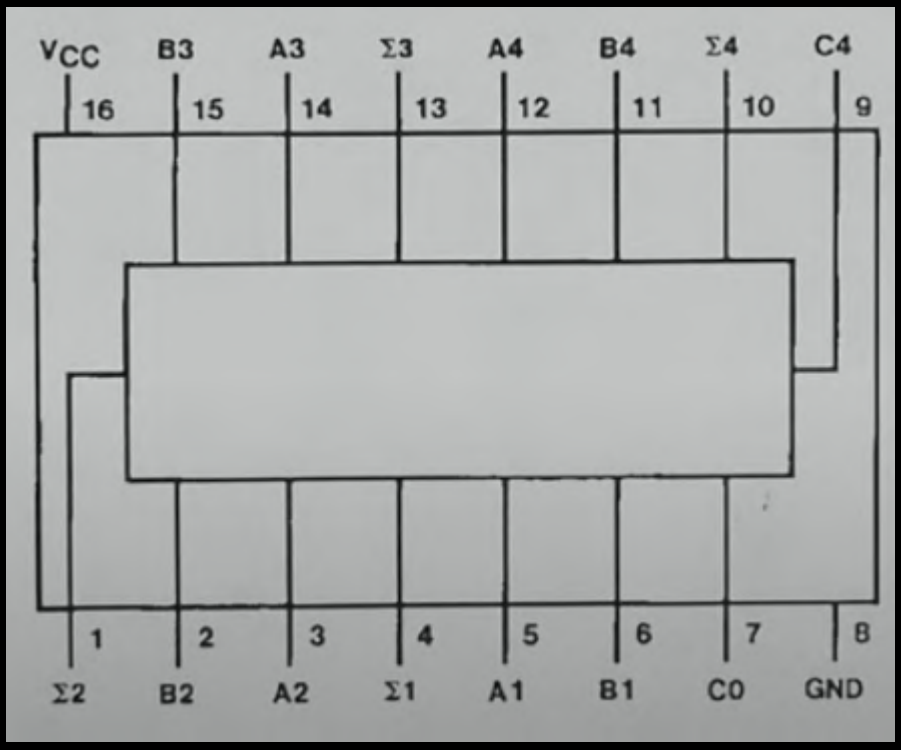




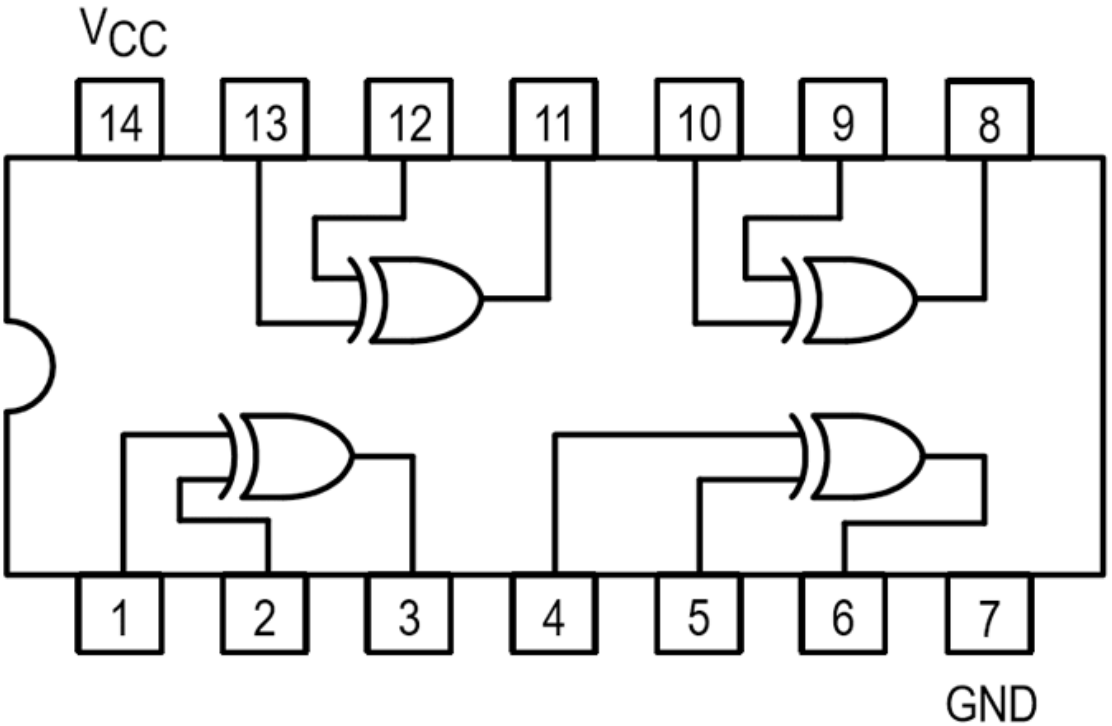
#11 Building the ALU

We'll use two 4-Bit Binary Adders (74LS283), two ICs for X-OR gates (74LS86) and one 74LS245. Btw, we directly used XOR gates available in Proteus.

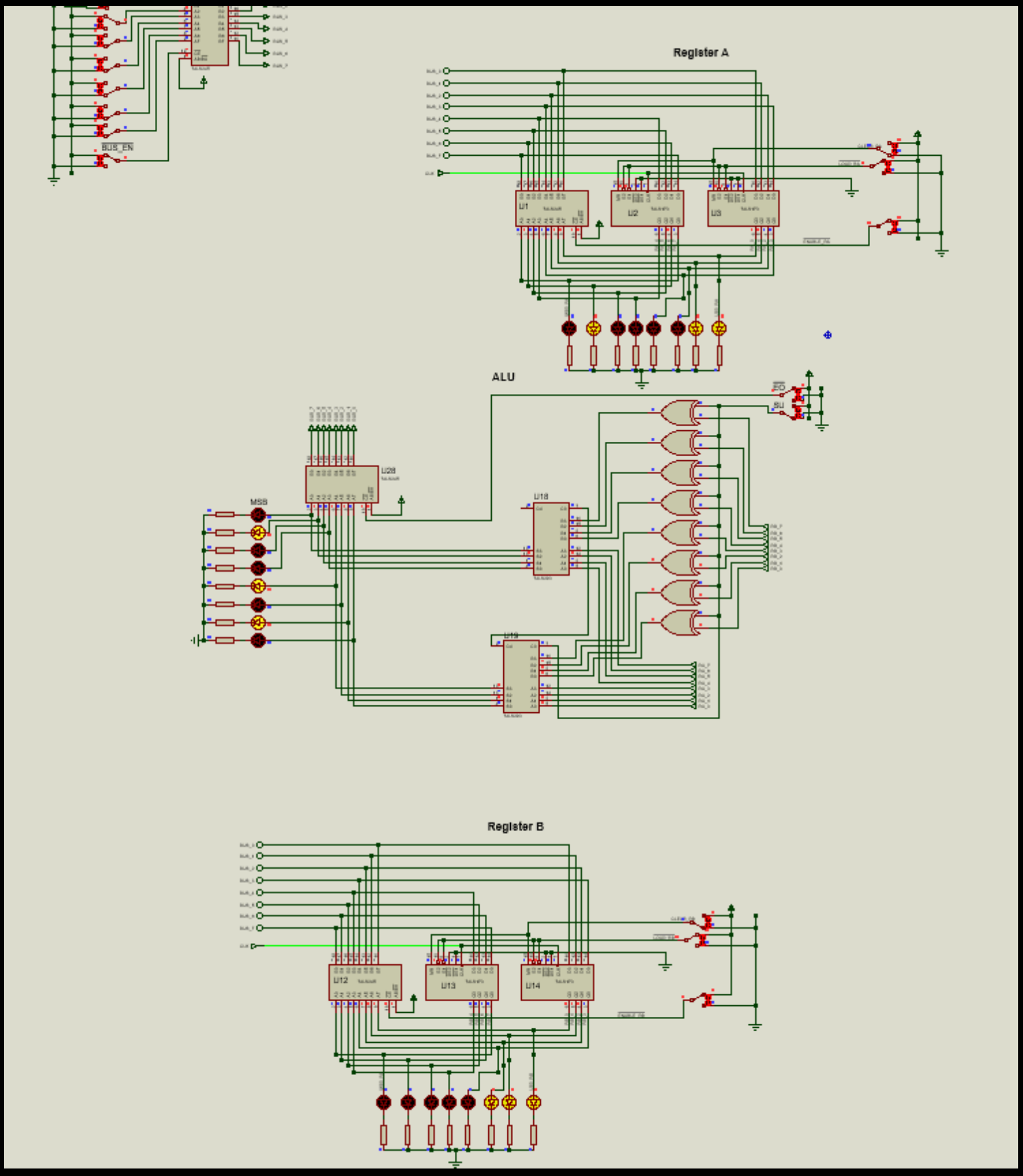
74LS283

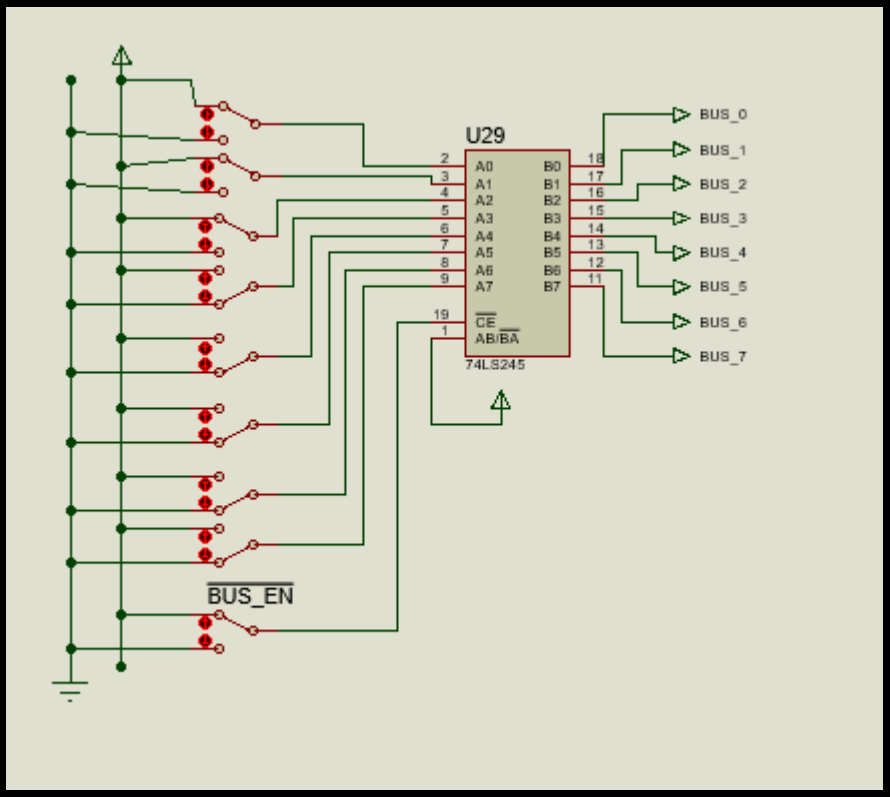


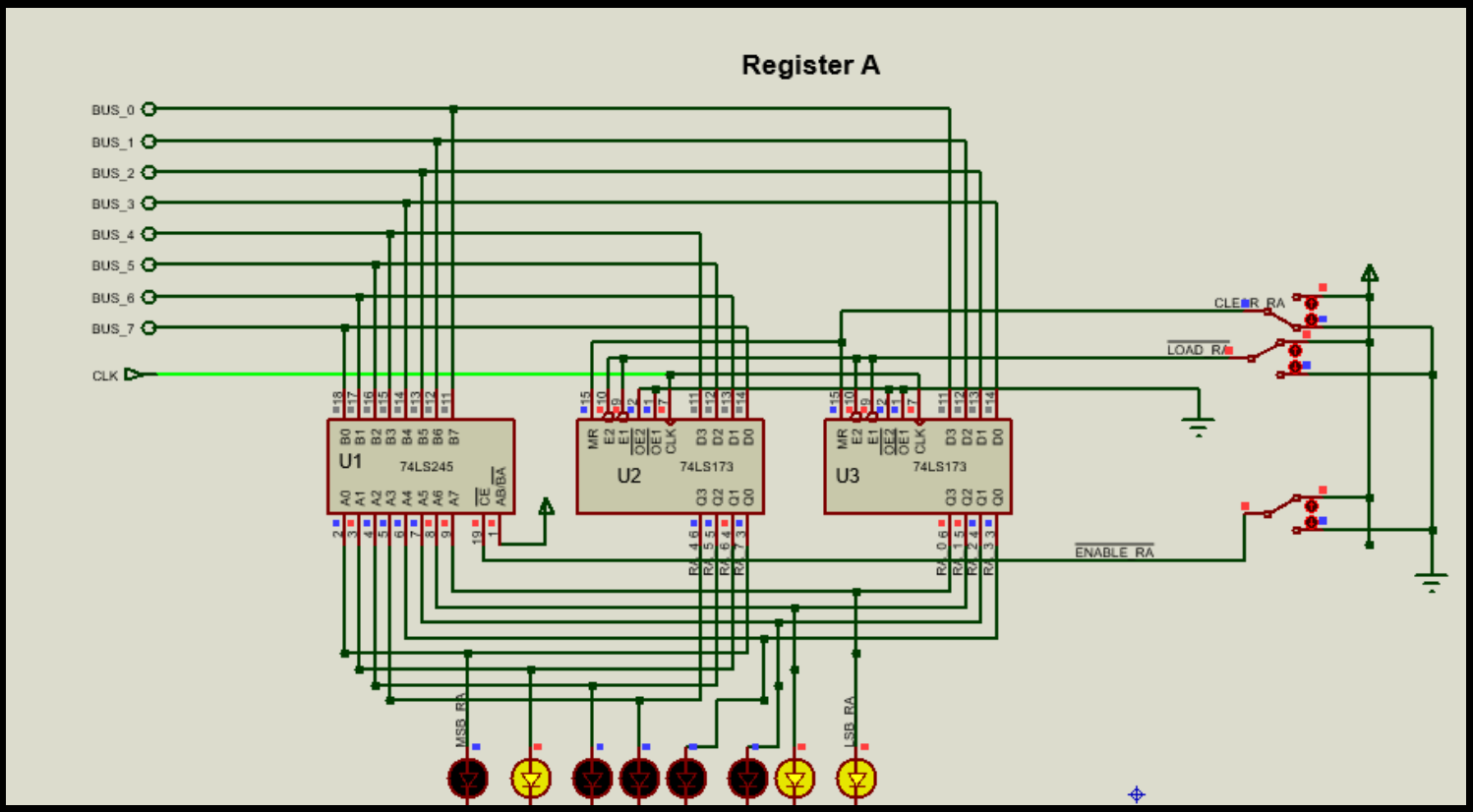
74LS86 Pinout

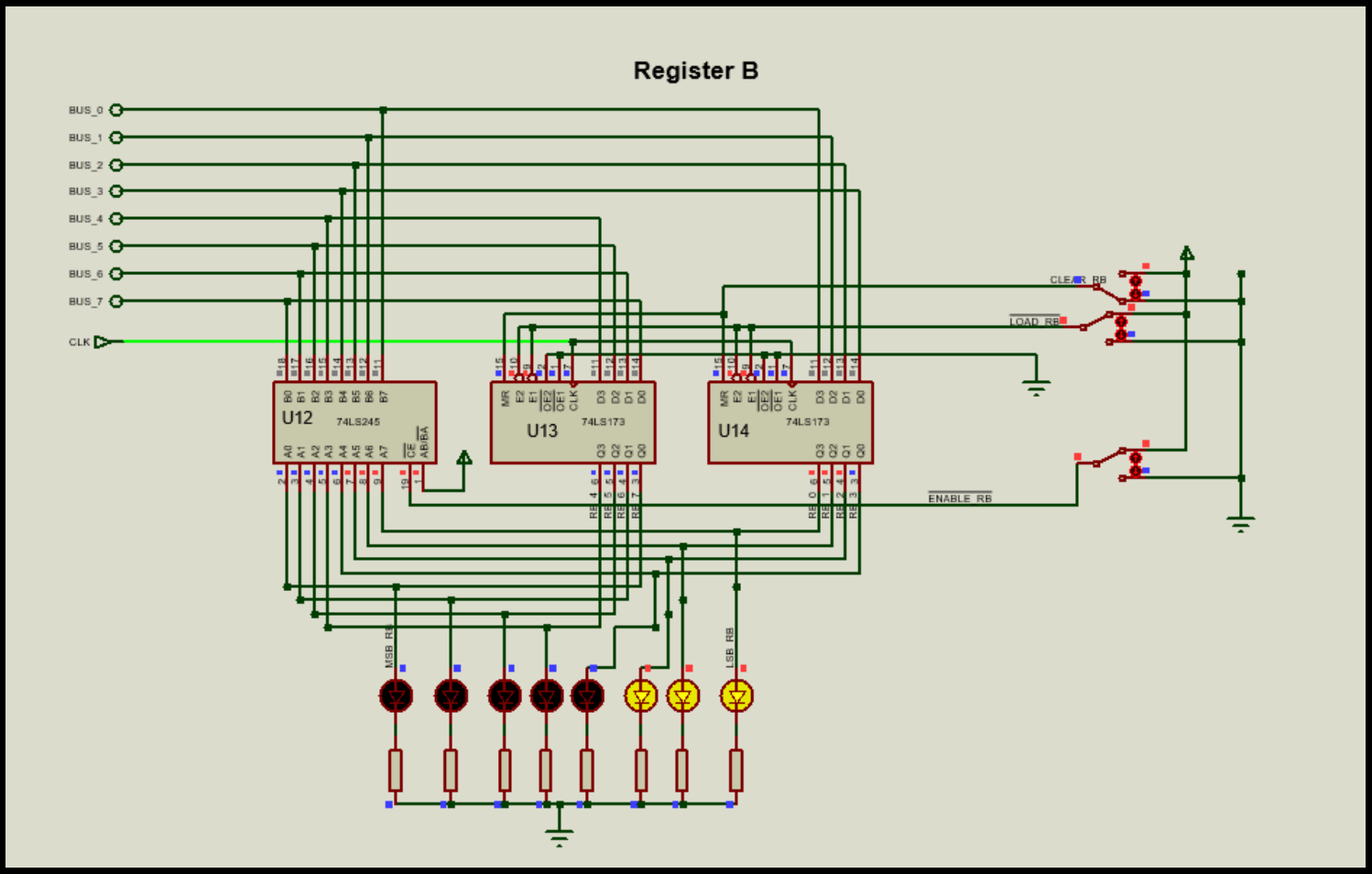


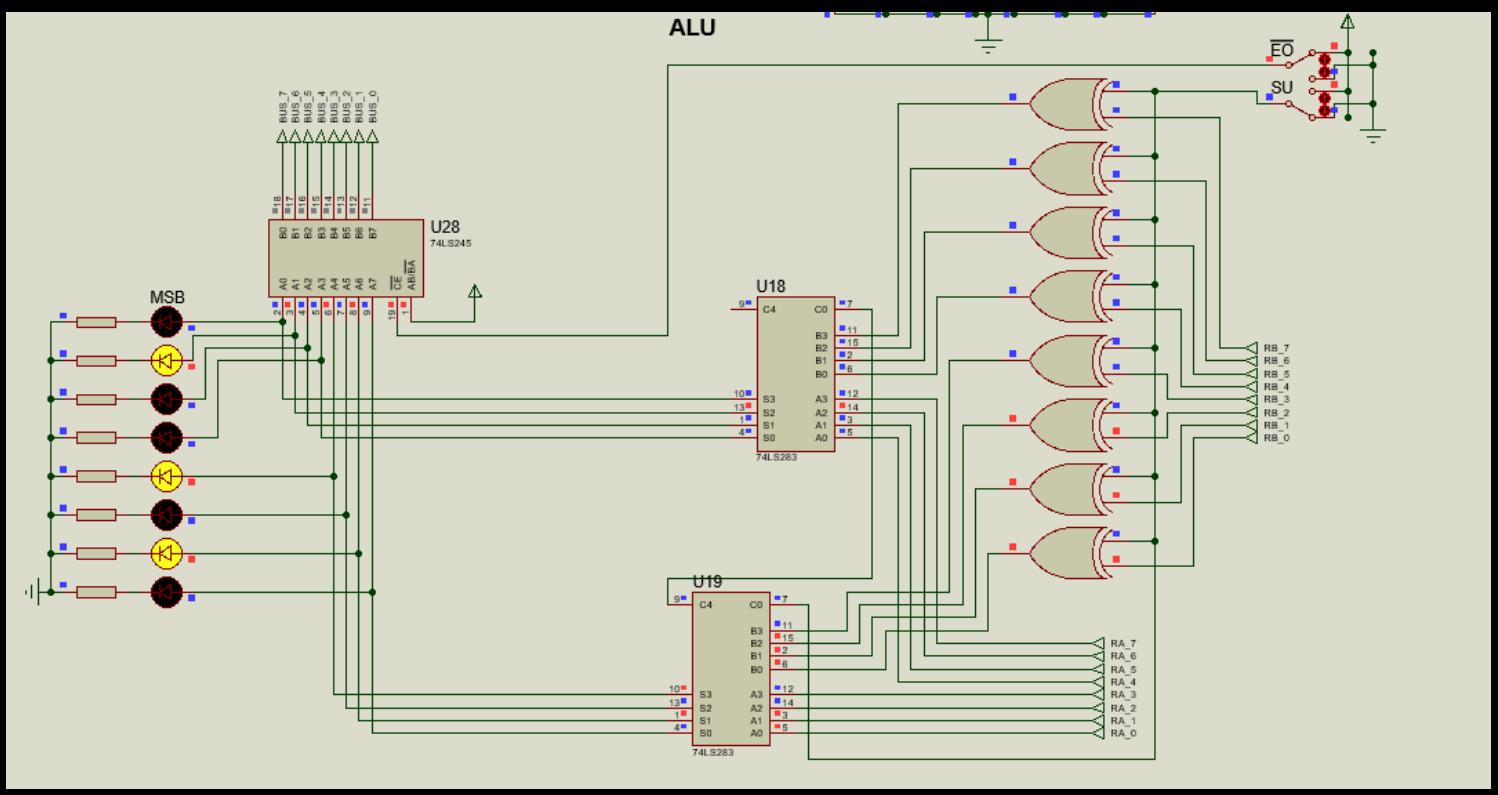
CIRCUITS DIY
SIMPLIFYING ELECTRONICS


BUS_EN
Register A
U1
U2
U3
ALU
MSB
U28
U18
U19
Register B
U12
U13
U14


U29
74LS245
BUS_0
BUS_1
BUS_2
BUS_3
BUS_4
BUS_5
BUS_6
BUS_7
BUS_EN


Register A
BUS_0
BUS_1
BUS_2
BUS_3
BUS_4
BUS_5
BUS_6
BUS_7
CLK
U1
74LS245
U2
74LS173
U3
74LS173
CLEAR_RA
LOAD_RA
ENABLE_RA
MSB_RA
LSB_RA





#11 Random access memory (RAM) module

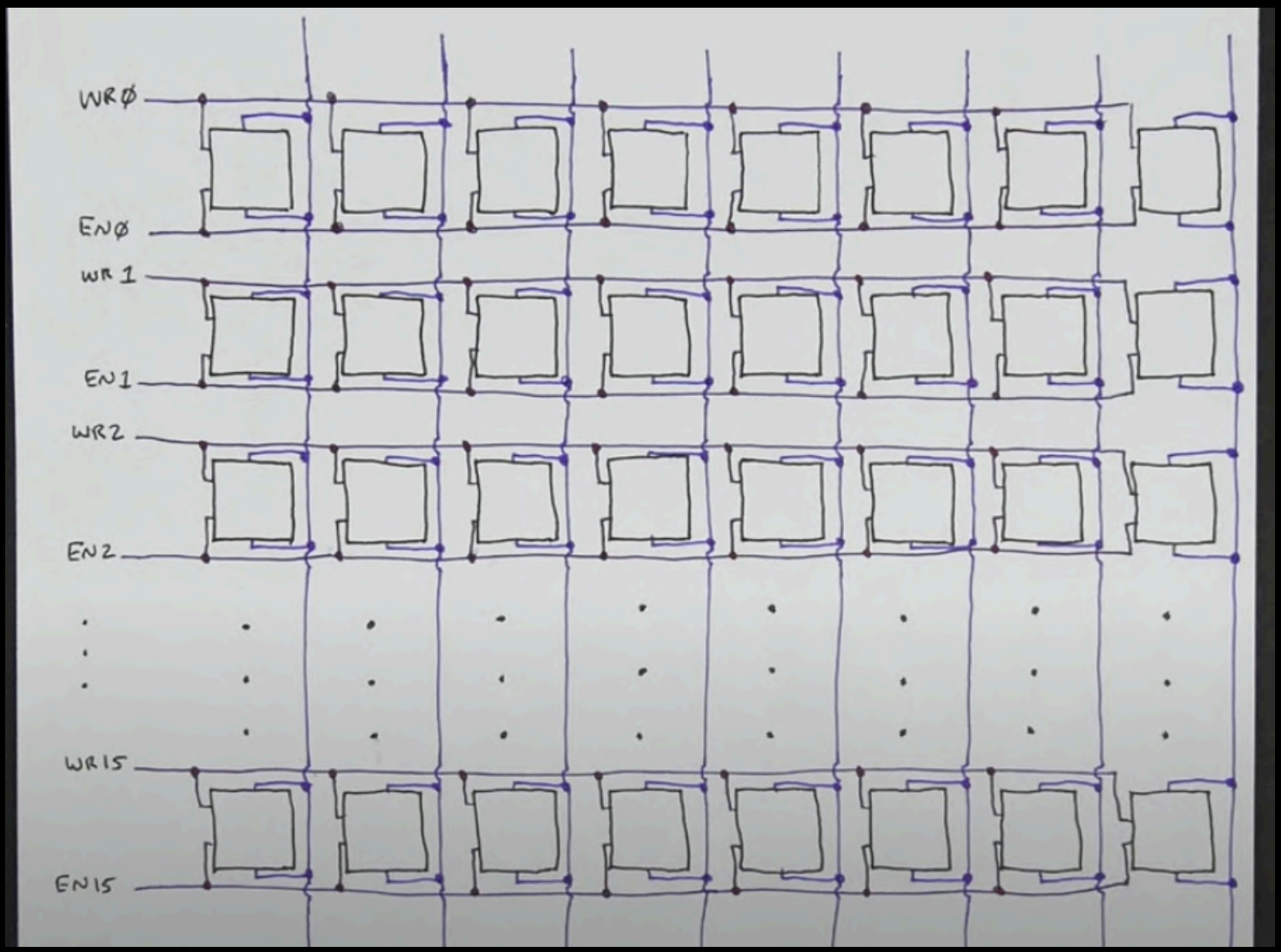
RAM stores the program that is being executed as well as any data the program needs. Our computer uses a 4-bit address, which means 2^4=16 unique addresses, and each address stores 1 Byte of data. So, our computer has 16 Bytes of RAM.

RAM introduction

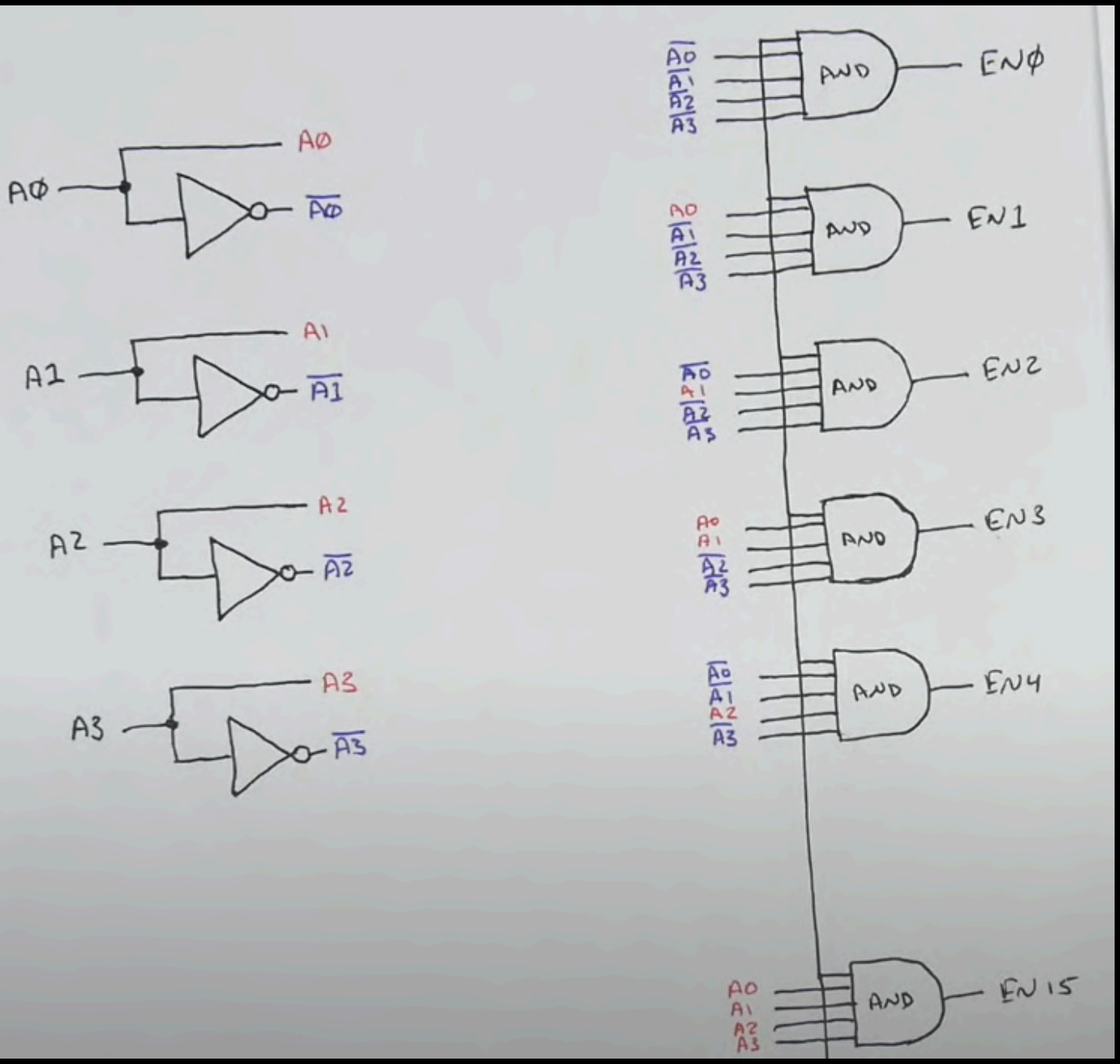
WR signal= writes data into the memory
EN= outputs the data from the memory

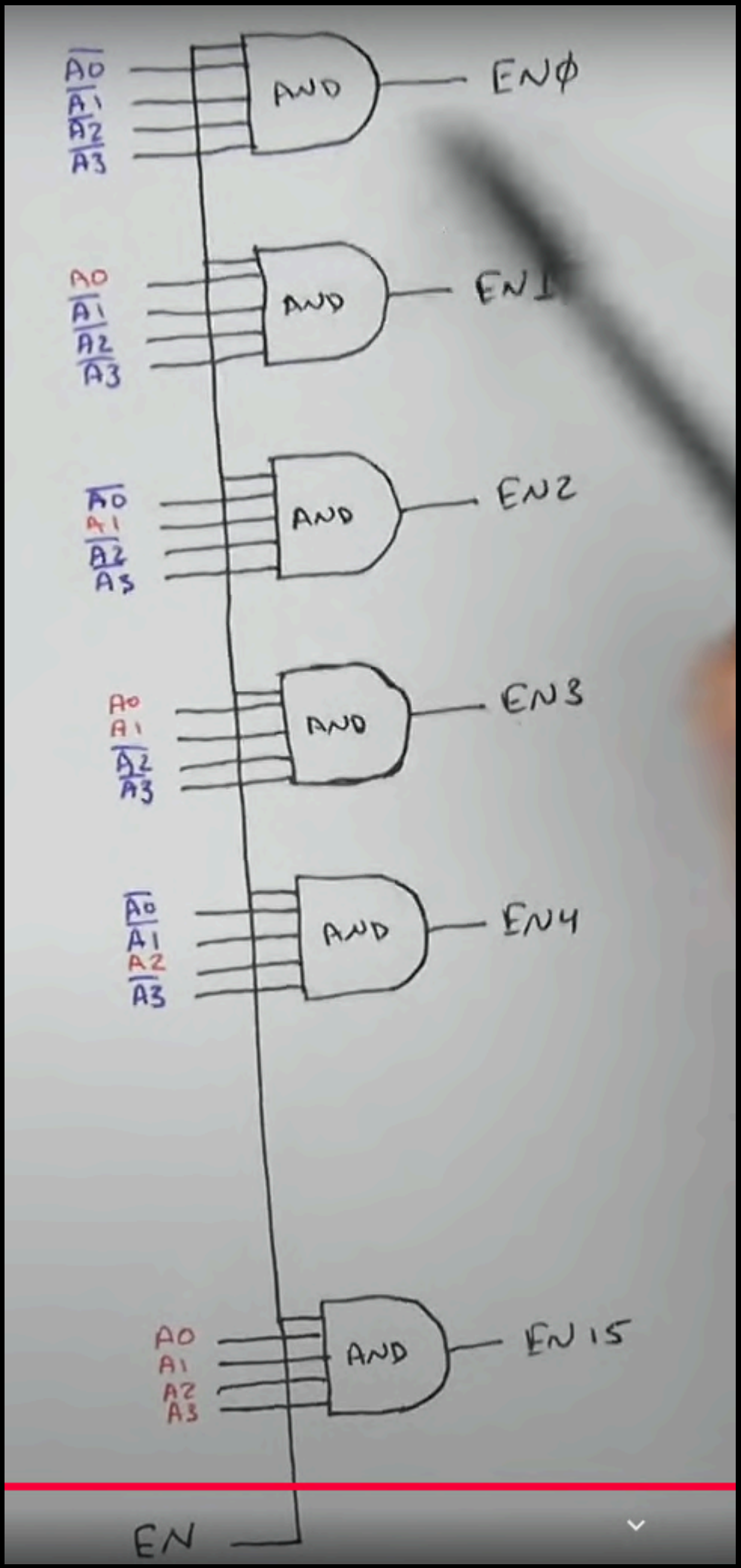
Each memory cell in a DRAM is composed of a transistor and a capacitor, the data in this cell gets periodically written by a circuitry to prevent loss of data from a discharging capacitor. Hence, the word “Dynamic” for the letter “D” in DRAM.

We’ll be using SRAM (Static RAM) for our computer. This uses flip-flops, like we used them for our registers.



Address decode logic


A0
A0
A1
A1
A2
A2
A3
A3
AND
EN0
AND
EN1
AND
EN2
AND
EN3
AND
EN4
AND
EN15


$\overline{A0}$
$\overline{A1}$
$\overline{A2}$
$\overline{A3}$
AND
EN0
A0
$\overline{A1}$
$\overline{A2}$
$\overline{A3}$
AND
EN1
$\overline{A0}$
A1
$\overline{A2}$
A3
AND
EN2
A0
A1
$\overline{A2}$
$\overline{A3}$
AND
EN3
$\overline{A0}$
A1
A2
$\overline{A3}$
AND
EN4
A0
A1
A2
A3
AND
EN15
EN

We'll use two 74LS189 (64-BIT RANDOM ACCESS MEMORY, with 3-state outputs).

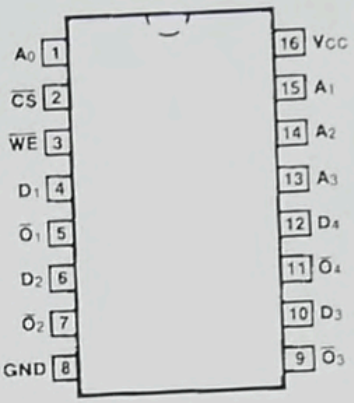
Write enable (WE bar) will put the data into the chip, remember, as evident from the logic diagram for 74LS189, while writing data into the chip, the output becomes OFF.

Yellow color in Chip of proteus means there has been some unintended connection of wires.

54S/74S189 011745
54LS/74LS189 011750

64-BIT RANDOM ACCESS MEMORY

(With 3-State Outputs)

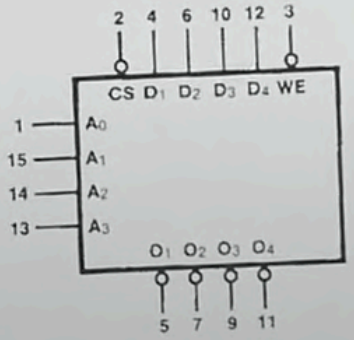


4

DESCRIPTION — The '189 is a high speed 64-bit RAM organized as a 16-word by 4-bit array. Address inputs are buffered to minimize loading and are fully decoded on-chip. The outputs are 3-state and are in the high impedance state whenever the Chip Select ($\overline{CS}$) input is HIGH. The outputs are active only in the Read mode and the output data is the complement of the stored data.

- 3-STATE OUTPUTS FOR DATA BUS APPLICATIONS
- BUFFERED INPUTS MINIMIZE LOADING
- ADDRESS DECODING ON-CHIP
- DIODE CLAMPED INPUTS MINIMIZE RINGING

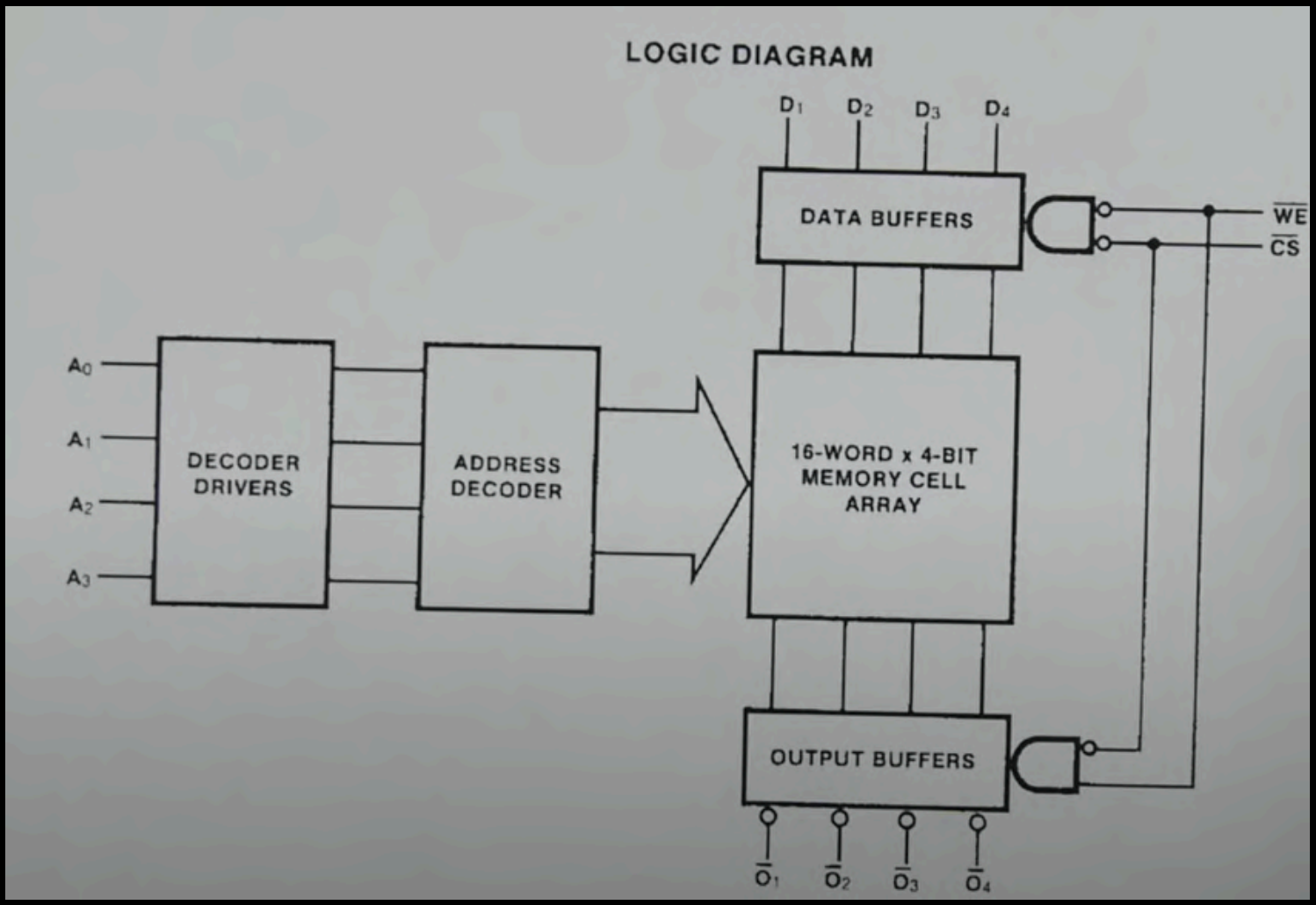
LOGIC SYMBOL



VCC = Pin 16
GND = Pin 8

ORDERING CODE: See Section 9

PKGS	PIN OUT	COMMERCIAL GRADE VCC = +5.0 V ±5%, TA = 0°C to +70°C	MILITARY GRADE VCC = +5.0 V ±10%, TA = -55°C to +125°C	PKG TYPE
Plastic DIP (P)	A	74S189PC, 74LS189PC		9B
			54S189DM, 54LS189DM	6B



Memory Address Register aka MAR (74LS173)

An unconnected TTL (Transistor-Transistor Logic) input is often referred to as a floating input. In TTL logic, floating inputs are generally treated as a HIGH logic level. So, we need to make sure all the inputs are properly connected.

Data Enable stores data from the bus to the chip.
Output control enables the output.

SN54173, SN54LS173A, SN74173, SN74LS173A
4-BIT D-TYPE REGISTERS WITH 3-STATE OUTPUTS

SDLS067A – OCTOBER 1976 – REVISED JUNE 1999

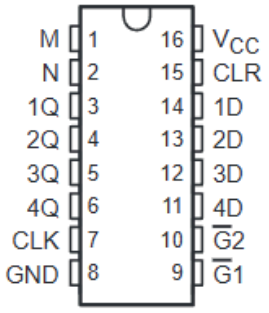
- **3-State Outputs Interface Directly With System Bus**
- **Gated Output-Control LInes for Enabling or Disabling the Outputs**
- **Fully Independent Clock Virtually Eliminates Restrictions for Operating in One of Two Modes:**
  - **Parallel Load**
  - **Do Nothing (Hold)**
- **For Application as Bus Buffer Registers**
- **Package Options Include Plastic Small-Outline (D) Packages, Ceramic Flat (W) Packages, Ceramic Chip Carriers (FK), and Standard Plastic (N) and Ceramic (J) DIPs**

TYPE	TYPICAL PROPAGATION DELAY TIME	MAXIMUM CLOCK FREQUENCY
'173	23 ns	35 MHz
'LS173A	18 ns	50 MHz

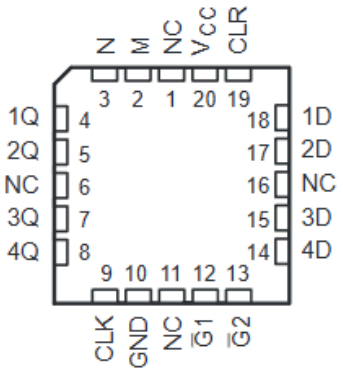
description

The '173 and 'LS173A 4-bit registers include D-type flip-flops featuring totem-pole 3-state outputs capable of driving highly capacitive or relatively low-impedance loads. The high-impedance third state and increased

SN54173, SN54LS173A . . . J OR W PACKAGE
SN74173 . . . N PACKAGE
SN74LS173A . . . D or N PACKAGE
(TOP VIEW)

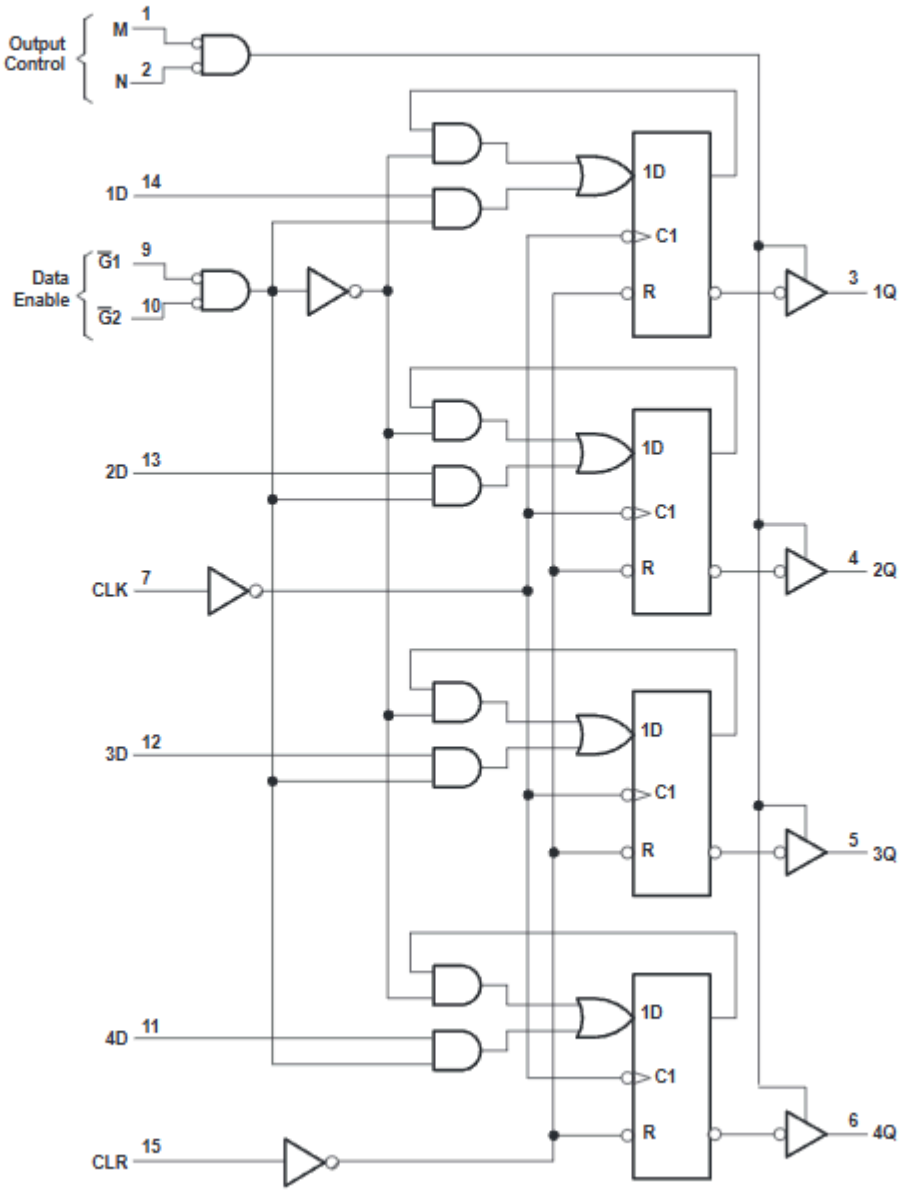


SN54LS173A . . . FK PACKAGE
(TOP VIEW)



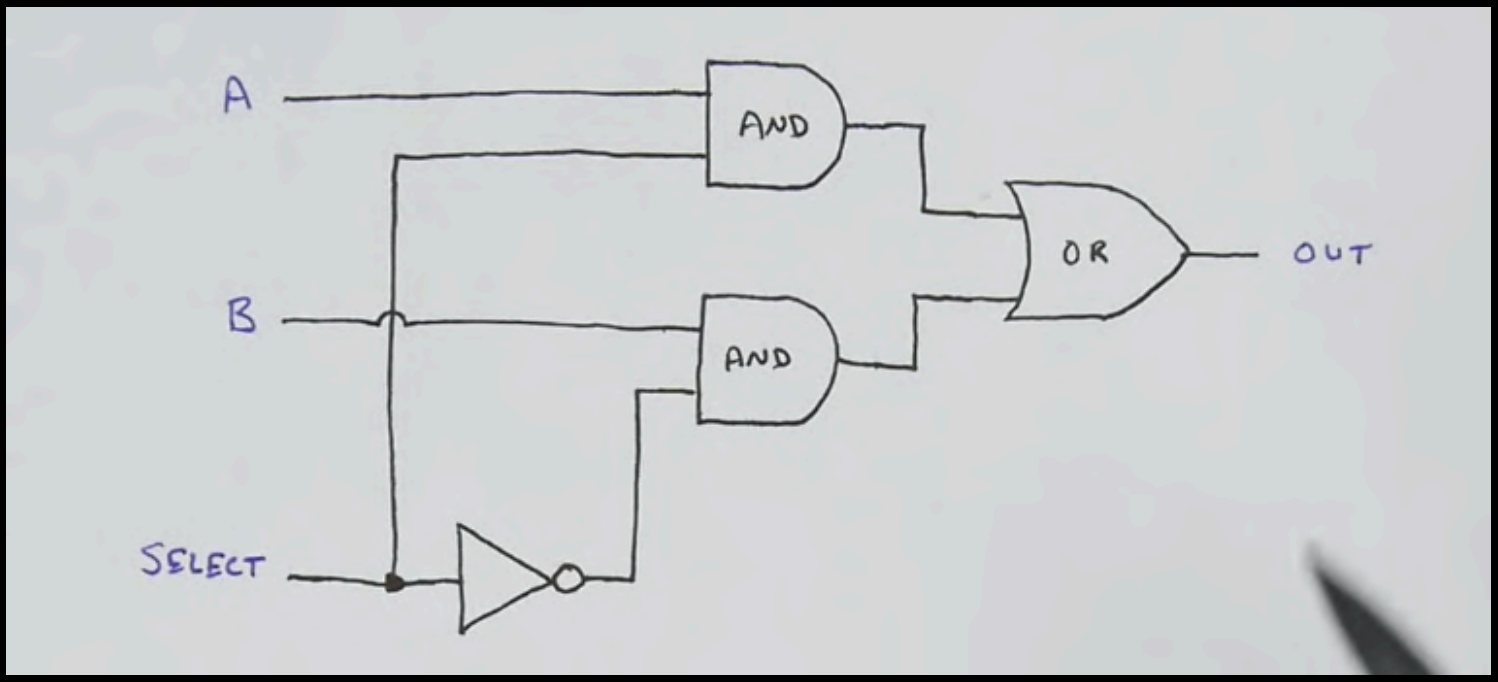
NC – No internal connection

logic diagram (positive logic)

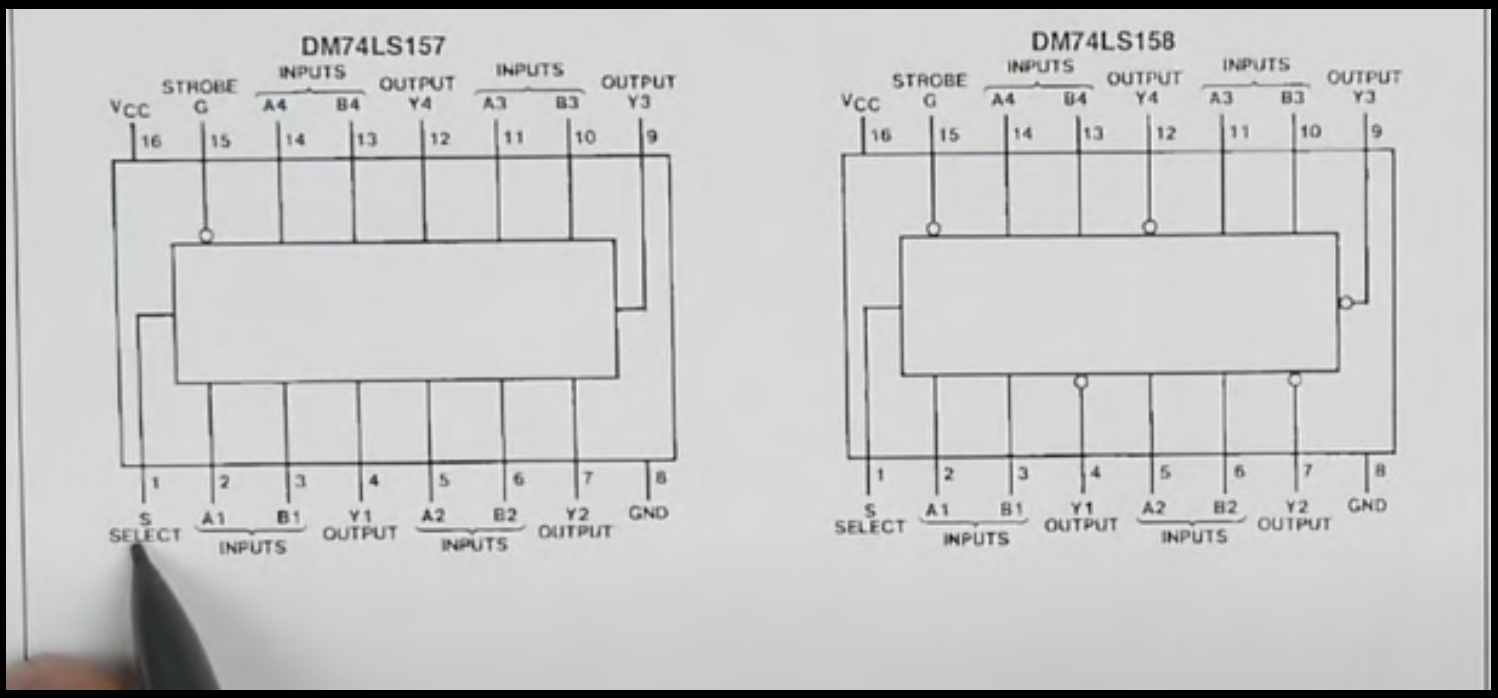


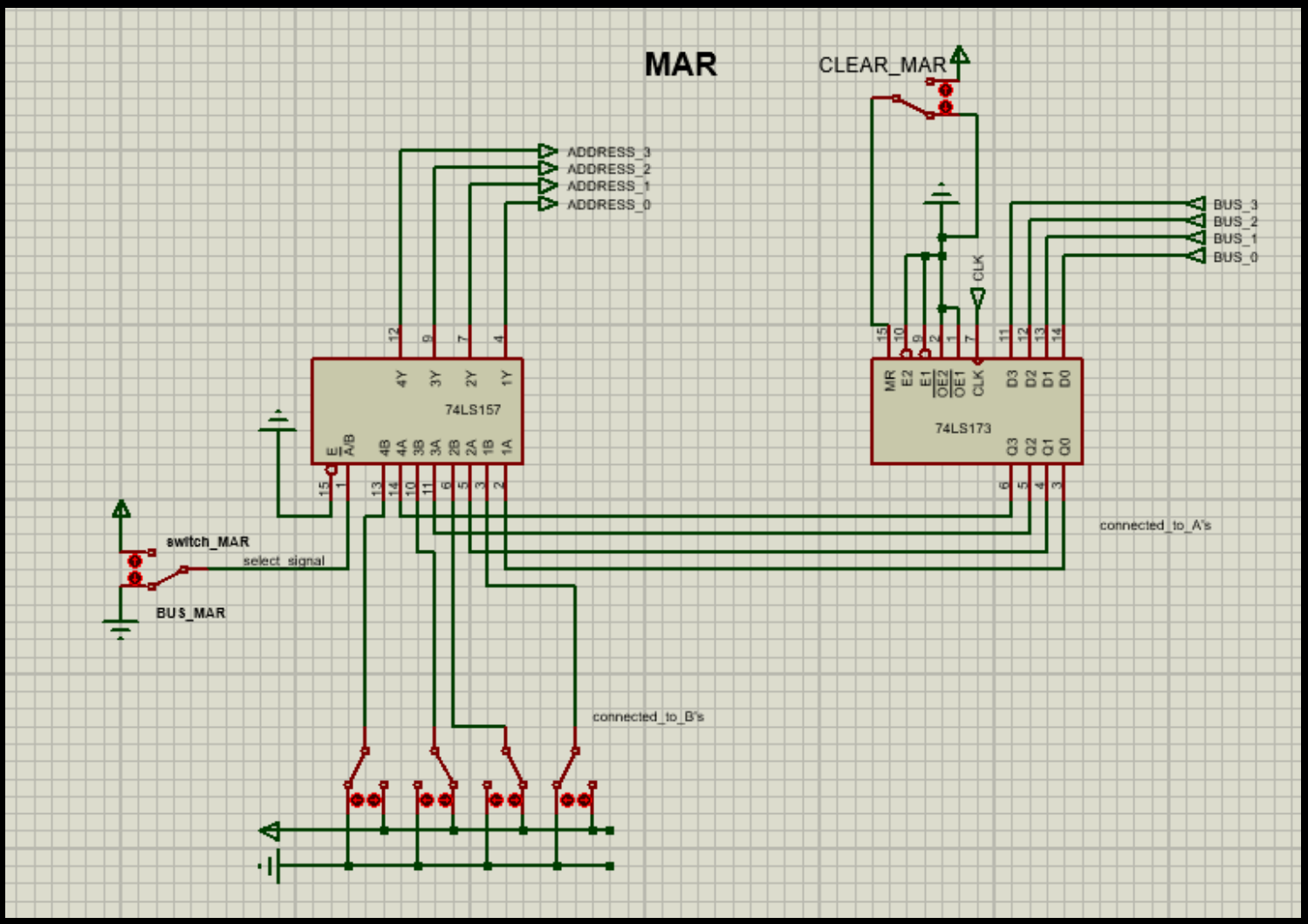
Pin numbers shown are for D, J, N, and W packages.

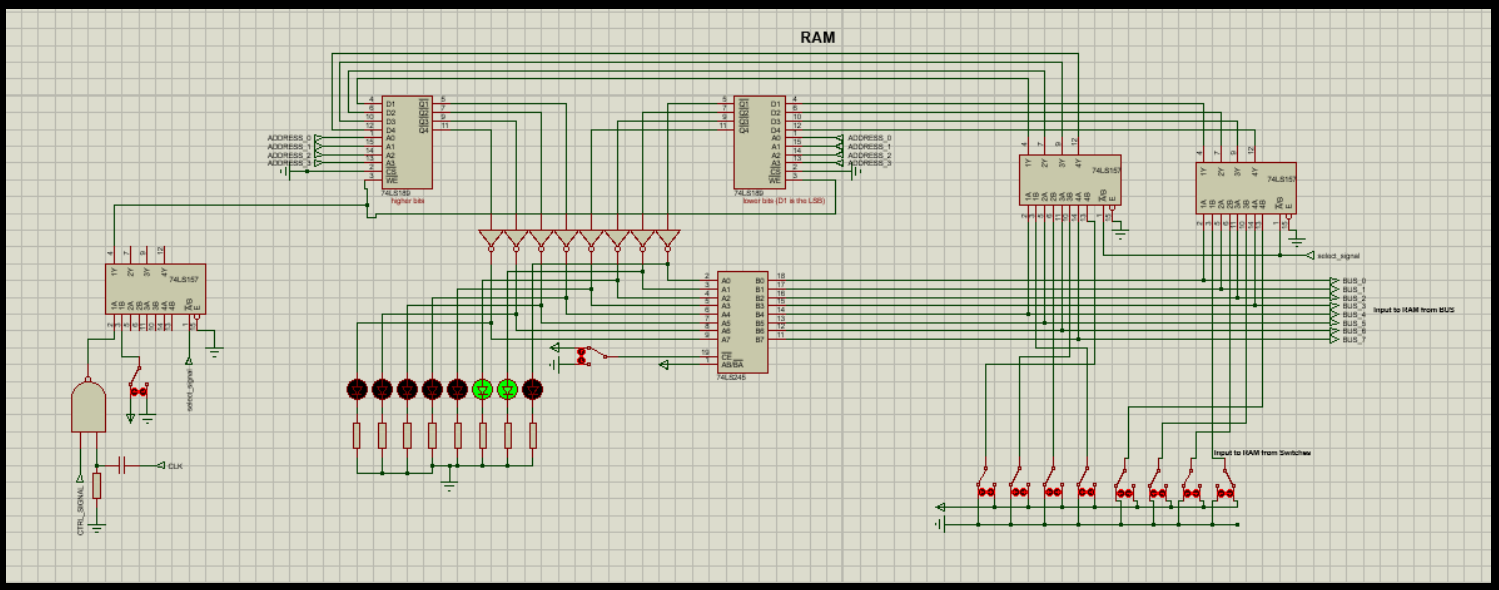
Logic to set address of RAM using DIP switches or MAR for ease of programming (following diagram is for single bit)



Luckily, this task is also done by 74LS157.
This chip will help us toggle between Program mode (RED LED) (input through DIP switches) and Run mode (GREE LED) (input from BUS).







#12 Program Counter

We will use DM7476 Dual Master-Slave J-K Flip-Flops to build Binary Counter.

DM7476

Dual Master-Slave J-K Flip-Flops with Clear, Preset, and Complementary Outputs

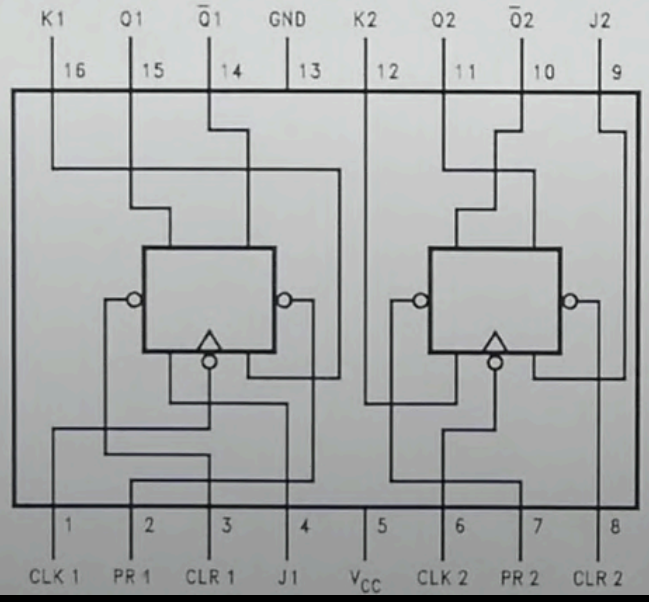
General Description

This device contains two independent positive pulse triggered J-K flip-flops with complementary outputs. The J and K data is processed by the flip-flop after a complete clock pulse. While the clock is LOW the slave is isolated from the master. On the positive transition of the clock, the data from the J and K inputs is transferred to the master. While the clock is HIGH the J and K inputs are disabled. On the negative transition of the clock, the data from the master is transferred to the slave. The logic state of J and K inputs must not be allowed to change while the clock is HIGH. The data is transferred to the outputs on the falling edge of the clock pulse. A LOW logic level on the preset or clear inputs will set or reset the outputs regardless of the logic levels of the other inputs.

Ordering Code:

Order Number	Package Number	Package Description
DM7476N	N16E	16-Lead Plastic Dual-In-Line Package (PDIP), JEDEC MS-001, 0.300" Wide

Connection Diagram



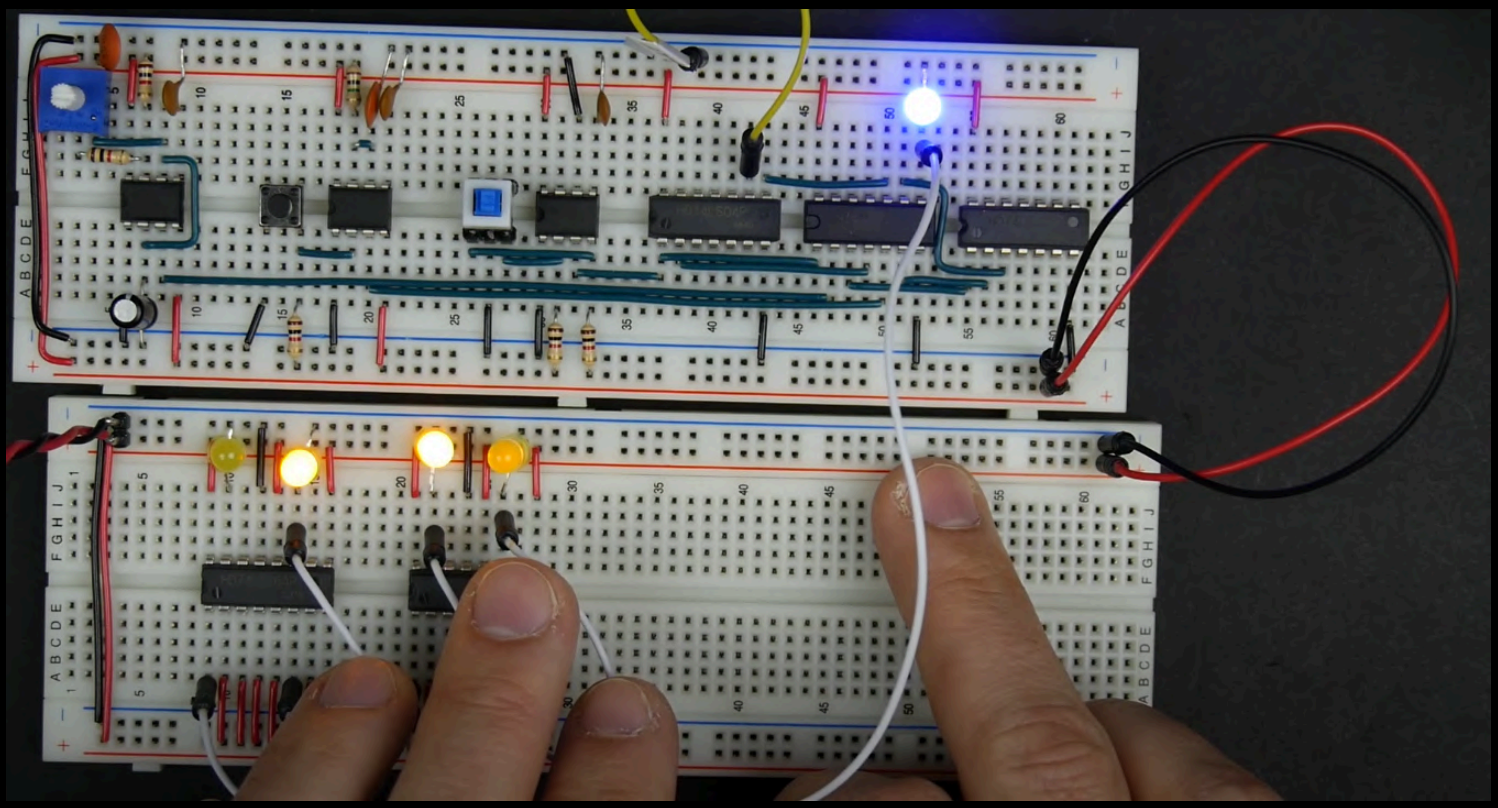
Function Table

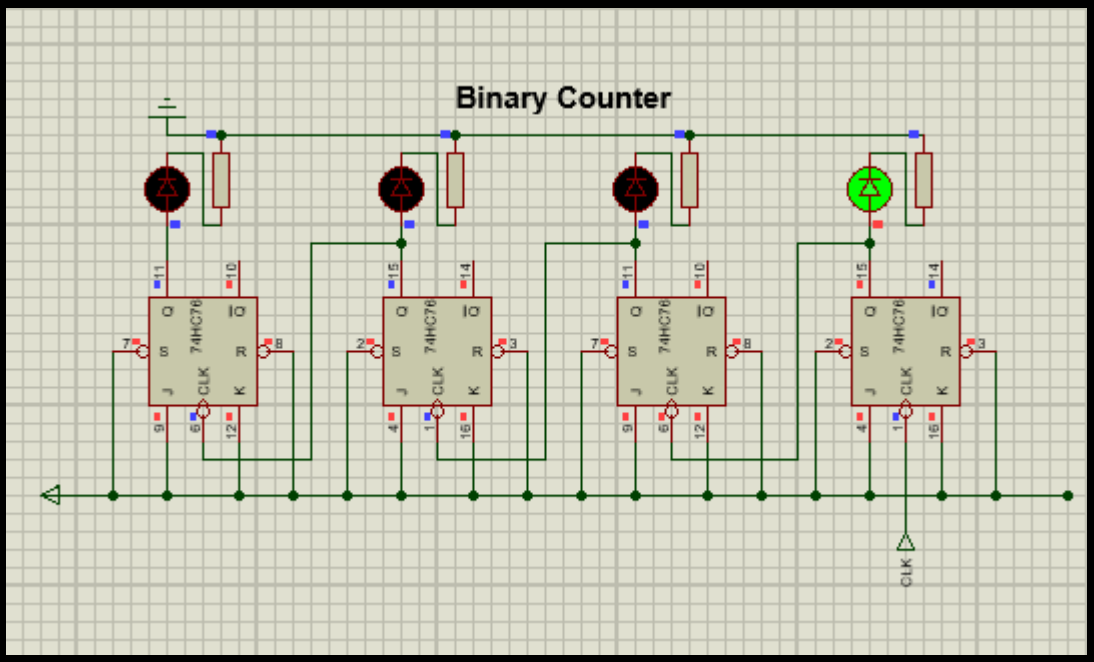
Inputs					Outputs	
PR	CLR	CLK	J	K	Q	$\overline{Q}$
L	H	X	X	X	H	L
H	L	X	X	X	L	H
L	L	X	X	X	H (Note 1)	H (Note 1)
H	H	⎍	L	L	Q_0	$\overline{Q}_0$
H	H	⎍	H	L	H	L
H	H	⎍	L	H	L	H
H	H	⎍	H	H	Toggle	

H = HIGH Logic Level
L = LOW Logic Level
X = Either LOW or HIGH Logic Level
⎍ = Positive pulse data. The J and K inputs must be held constant while the clock is HIGH. Data is transferred to the outputs on the falling edge of the clock pulse.
Q_0 = The output logic level before the indicated input conditions were

Binary counter (0 to 15) using four JK flip-flops

The speed of toggle is exactly half of the input clock pulse. Using this logic, we use the output of the left most JK flip flop as the clock for the next stage JK flip-flop and so on.

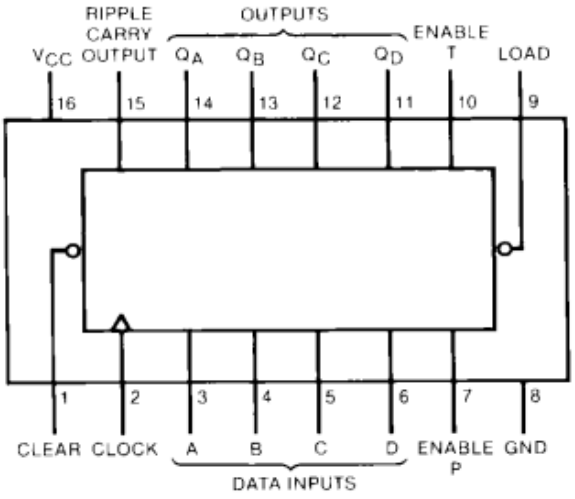




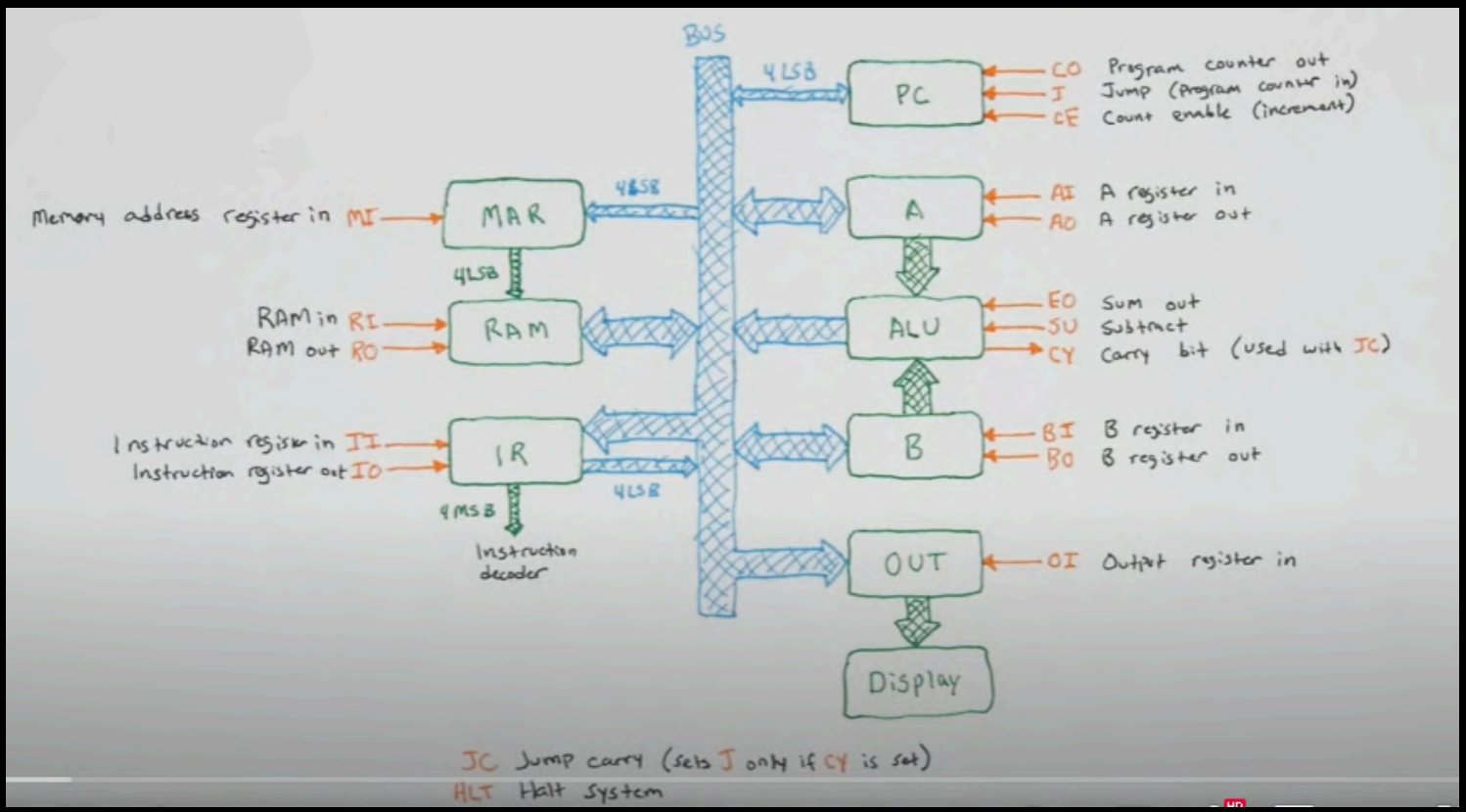
We can achieve the same using 74LS161.

DM74LS161A • DM74LS163A

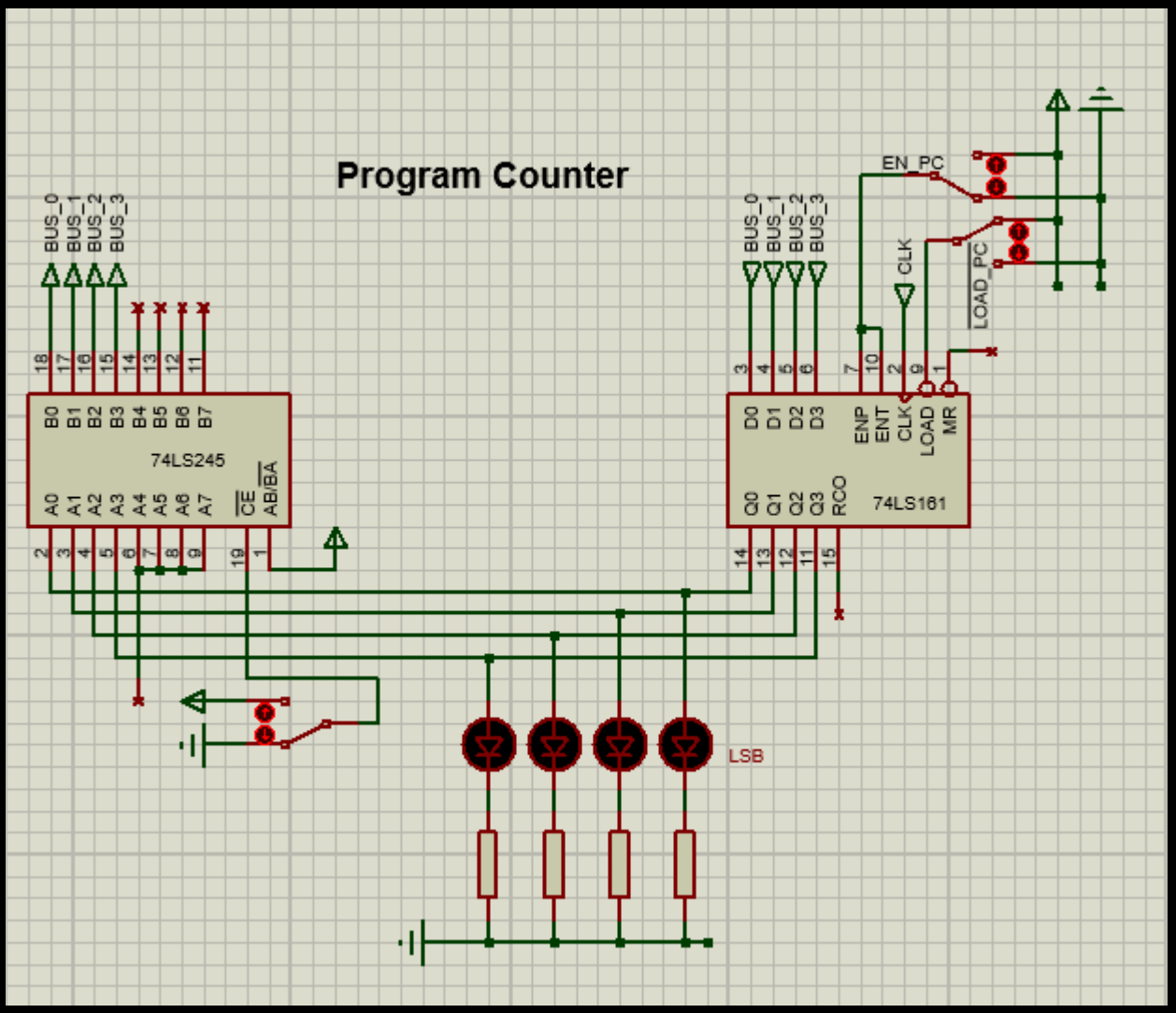
Connection Diagram



Both count-enable inputs (P and T) must be HIGH to count.
Load signal and Data inputs (A,B,C,D) can be used to implement JUMP.



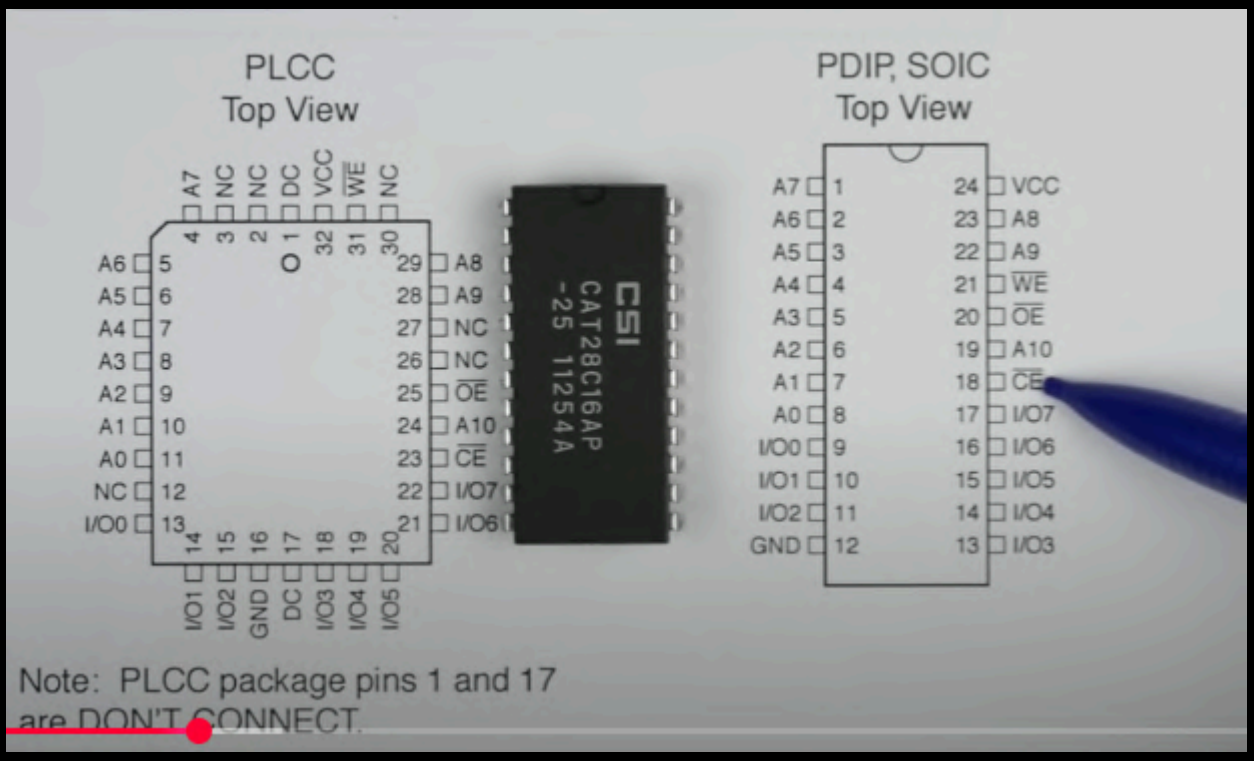
For PC, Program Count Out means enable signal (pin 19) of 74LS245, Count Enable means count-enable inputs (pin 7,10 of 74LS161), and Jump means LOAD signal (pin 9 of 74LS161).



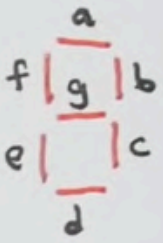
#12 Output Register

To take in 4 binary inputs and display the corresponding HEX value in a seven segment display, we can either just implement the truth table using gates, or since it is a combination logic (output only depends on the inputs), we can implement this using EEPROMs.

We'll use AT28C16.

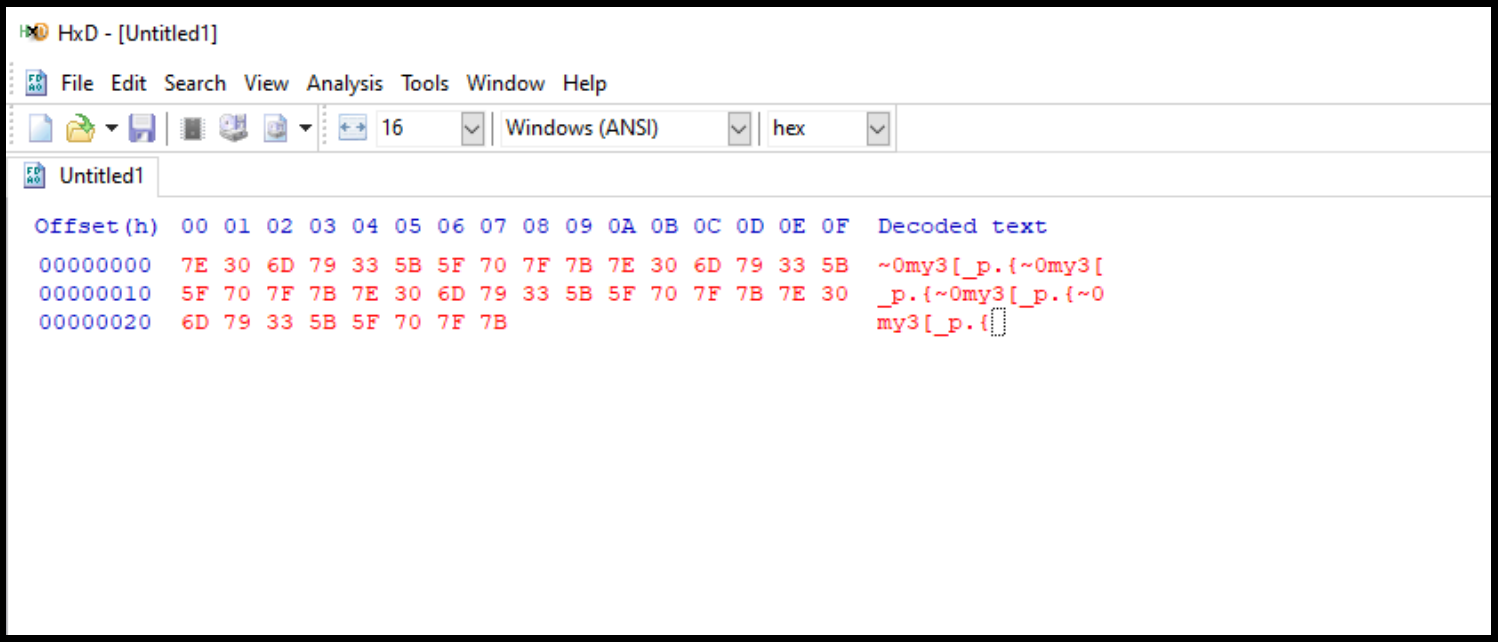


	dp	a	b	c	d	e	f	g	
0	0	1	1	1	1	1	1	0	7E
1	0	0	1	1	0	0	0	0	30
2	0	1	1	0	1	1	0	1	6D
3	0	1	1	1	1	0	0	1	79
4	0	0	1	1	0	0	1	1	33
5	0	1	0	1	1	0	1	1	5B
6	0	1	0	1	1	1	1	1	5F
7	0	1	1	1	0	0	0	0	70
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	1	1	0	1	1	7B



Now, for the Output register we need an Arduino Nano. And, Proteus does not offer this natively.

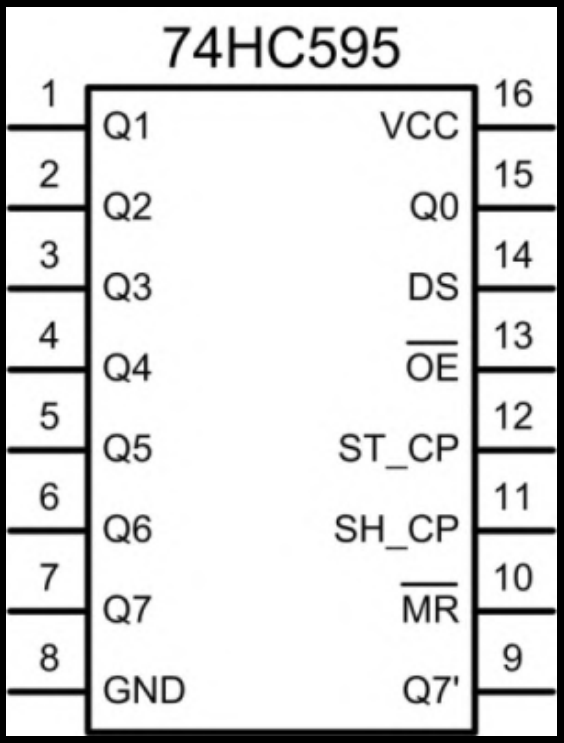
1st method: I created a hex file by manually writing content for each memory location. It looks like this.

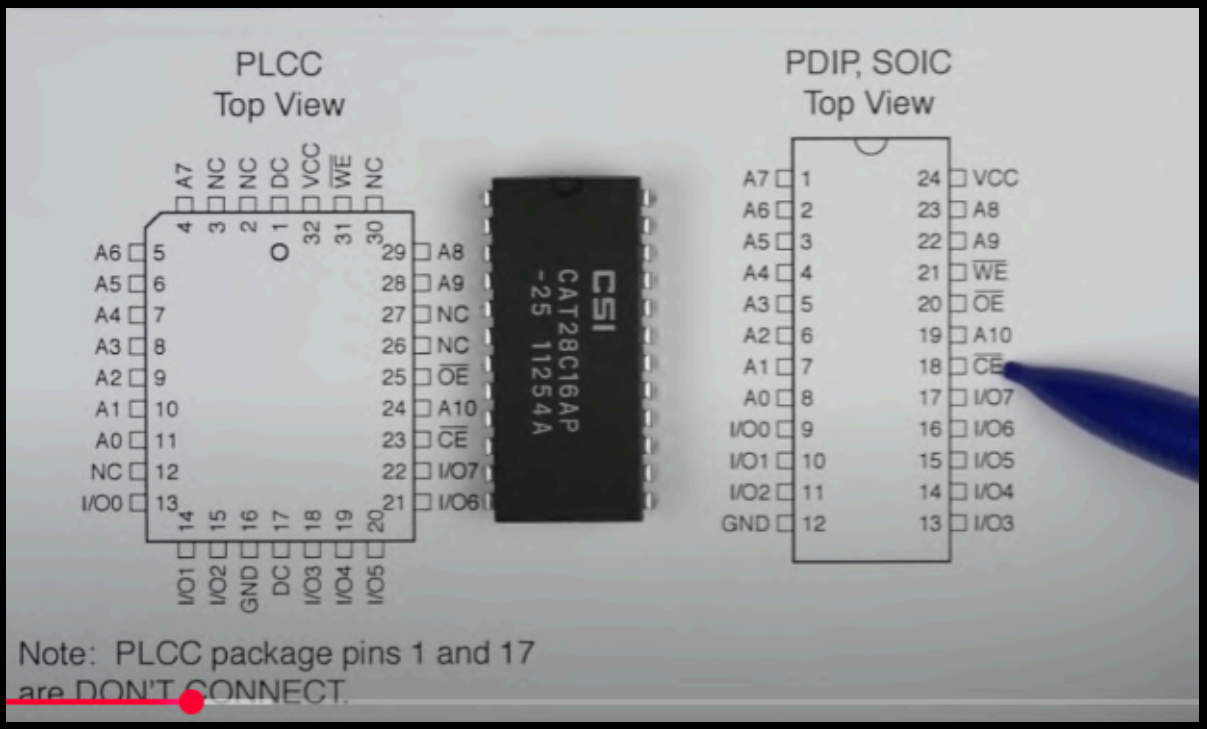


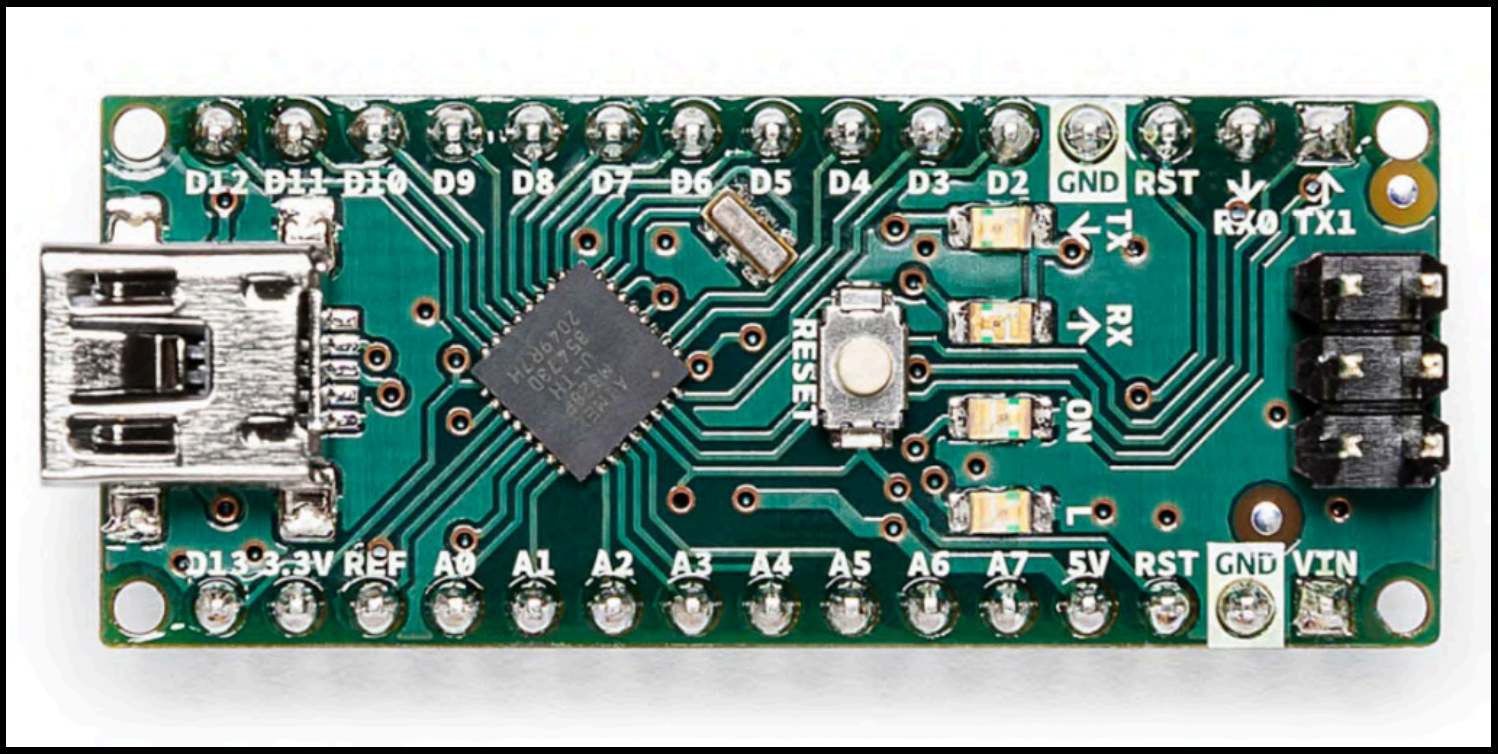
This soon got complex with more addresses to write.

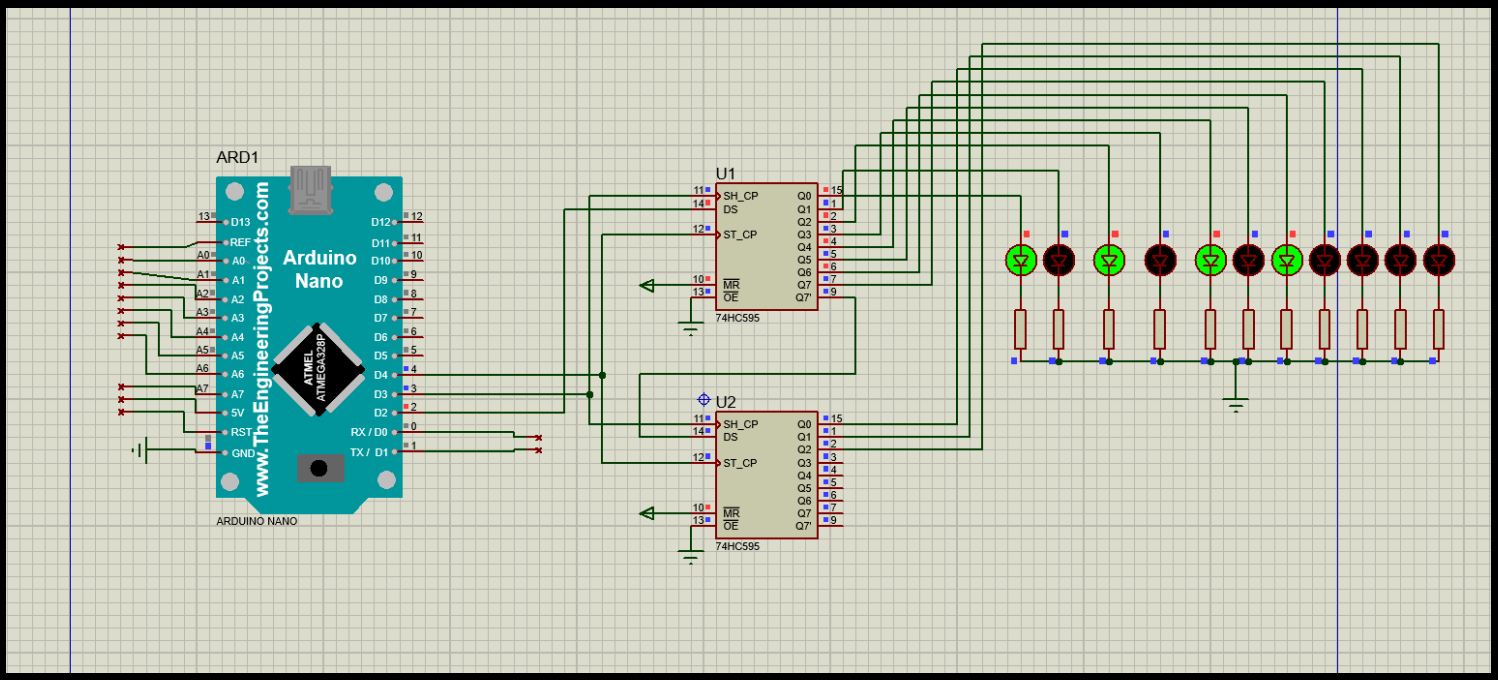
Then, I resorted to the second method.

2nd method: I downloaded the Proteus project for Arduino Nano from some third party website, and it could result in anything, like this project being halted. After playing around with some Nano coding and hex files, I finally got this working.

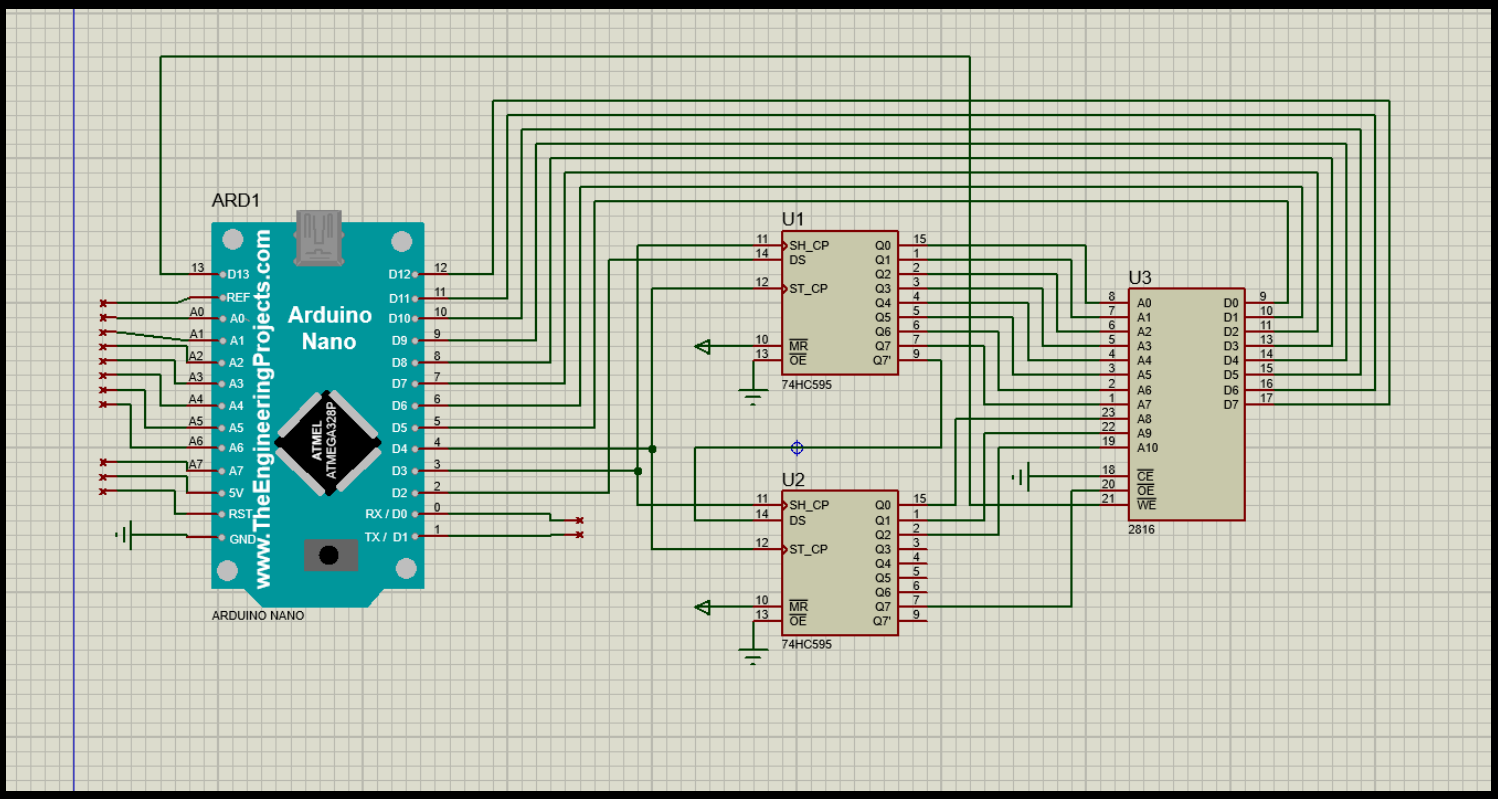

74HC595
1 Q1
2 Q2
3 Q3
4 Q4
5 Q5
6 Q6
7 Q7
8 GND
16 VCC
15 Q0
14 DS
13 OE
12 ST_CP
11 SH_CP
10 MR
9 Q7'

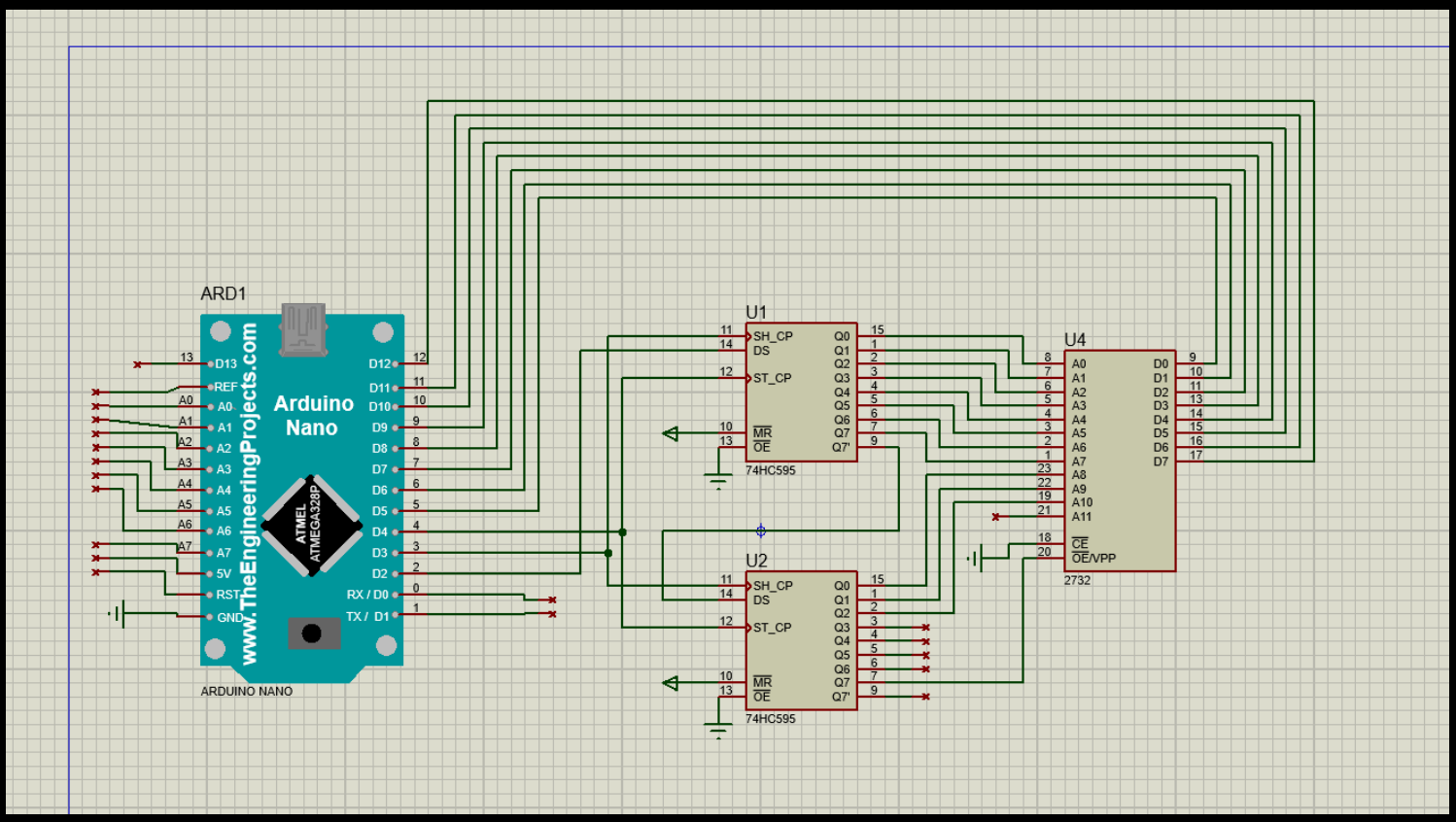

PLCC
Top View
PDIP, SOIC
Top View
CSI
CAT28C16AP
-25 11254A
Note: PLCC package pins 1 and 17
are DON'T CONNECT.

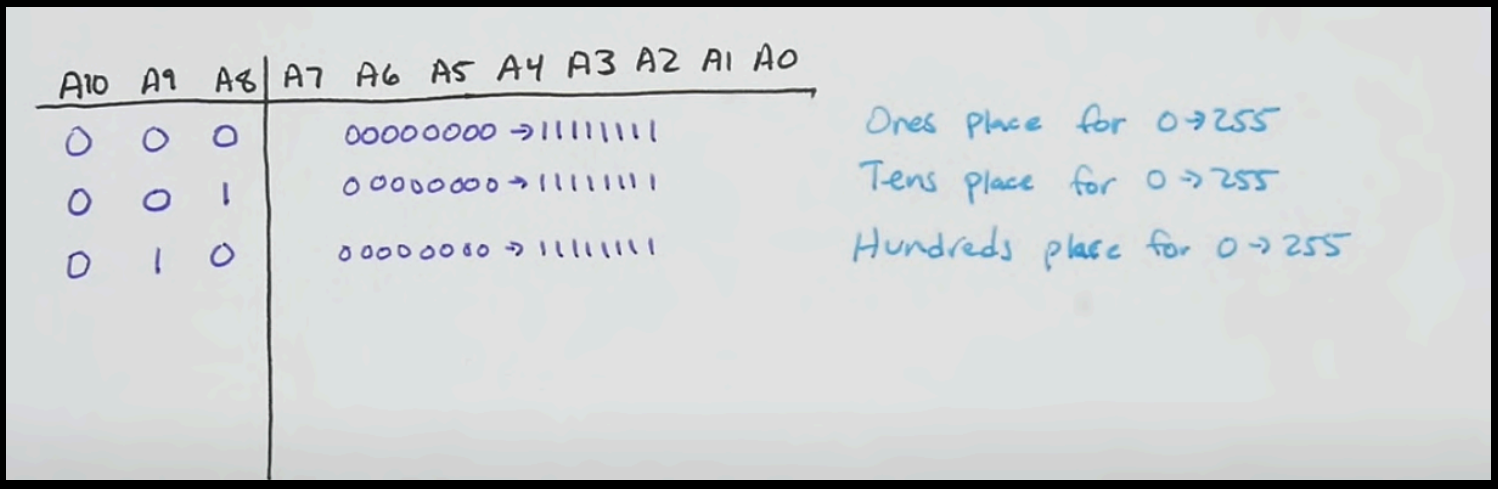


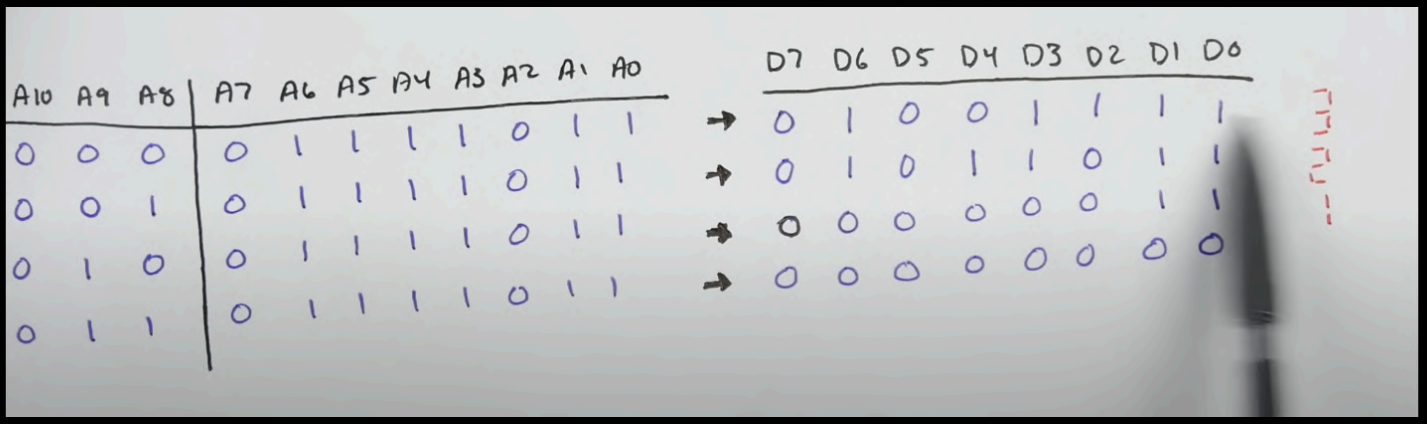


And, a brand new error. Turns out this EEPROM (2816) has no simulation model defined in Proteus, hence it cannot be simulated. At one point, I was like, okay this project was only upto here, cause, firstly, the Arduino Nano was from a third party (reliability issue, no documentation) and now we should opt for a brand new EEPROM. While searching on EEPROMs with simulation models, I found 2732. It is very much alike to 2816, except that it has more memory capacity, does not have WE pin, from the datasheet of 2732. Just read the data sheet of 2732, it has OE and VPP (kind of WE signal) on the same pin, and I kind of have a gut feeling that this will create some trouble in the future. So, let's choose yet another chip 2764, it has dedicated OE and VPP pins, so we can manually control them. God, hope this works. And, it does not


ARD1
Arduino Nano
www.TheEngineeringProjects.com
ATMEL ATMEGA328P
ARDUINO NANO
U1
U2
74HC595
U3
2816


ARD1
Arduino Nano
www.TheEngineeringProjects.com
ATMEL ATMEGA328P
ARDUINO NANO
U1
U2
74HC595
U4
2732

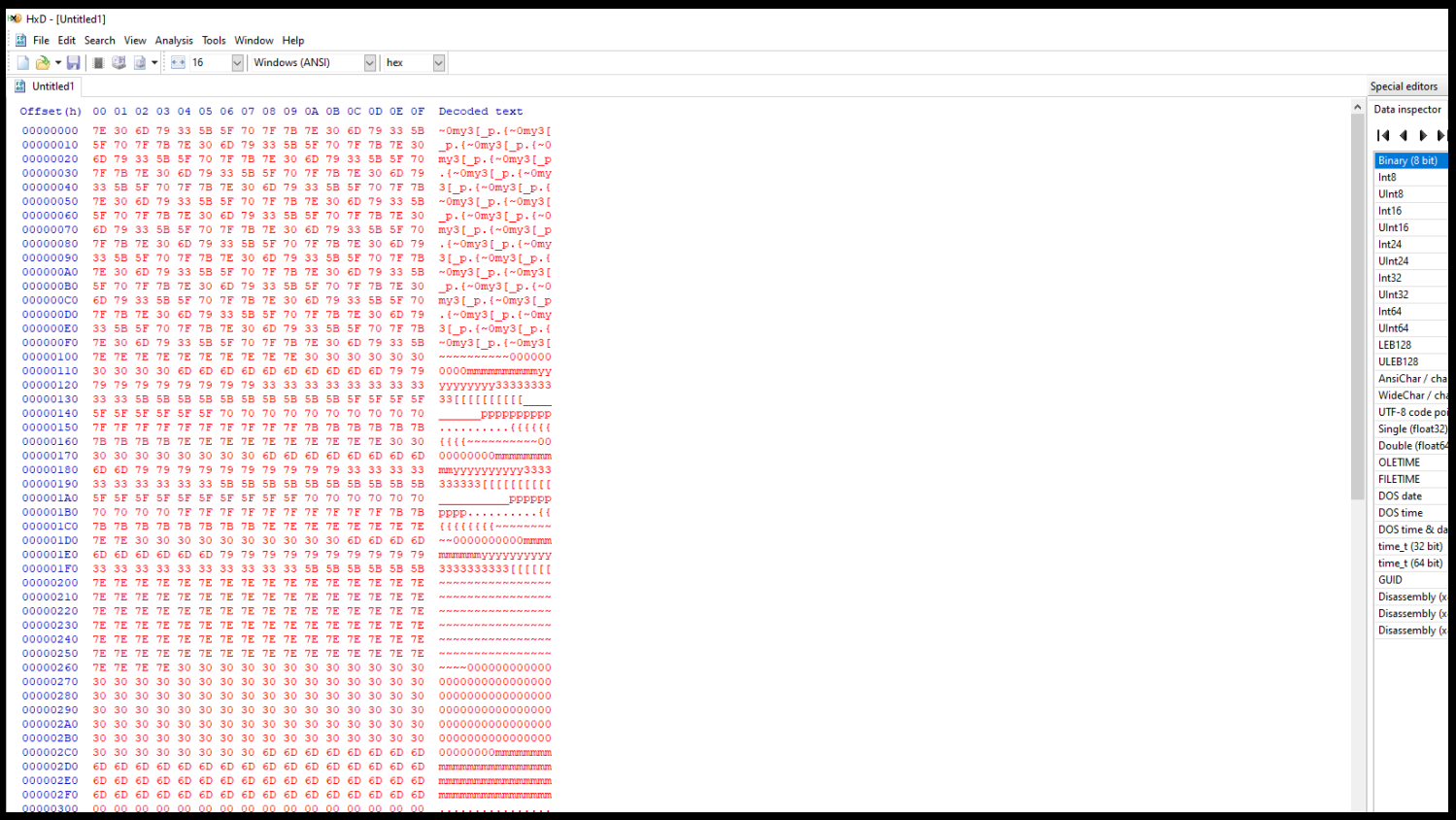




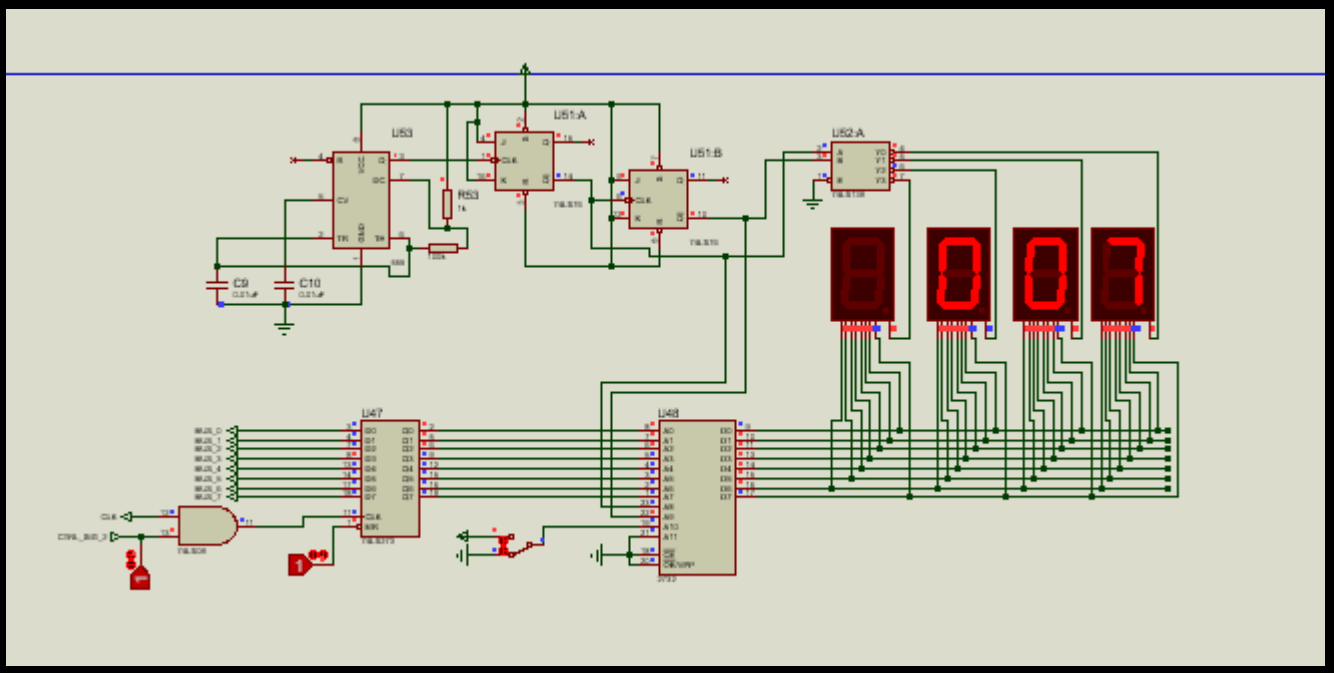
Arduino code:

```
byte digits[]={0x7e, 0x30, 0x6d, 0x79, 0x33, 0x5b, 0x5f, 0x70, 0x7f, 0x7b};

for (int address=0; address<=255; address+=1){
  write(address, digits[address%10]);
}

  for (int address=0; address<=255; address+=1){
  write(address+256, digits[(address/10)%10]);
}

 for (int address=0; address<=255; address+=1){
  write(address+512, digits[(address/100)%10]);
}

 for (int address=0; address<=255; address+=1){
  write(address+768, 0);
}
```

The equivalent HEX values:



Finally, we cranked up the CPU and got to see output. It's slow, but it works!



So, we can see real time output obtained from adding Register A and B, where Register A was reading continuously from the bus where ALU was putting on calculated data continuously, kind of a loop thing.

Future works: Adding Unconditional jumps (loops) and conditional jumps so that this computer becomes a Turing Complete Machine.

#13 Control Logic

I did this work on copy itself, so I've attached the notes here.

Designing the opcode

eg.

LDA	14
ADD	15
OUT	

// means load from memory location 14 to Register A
// means load from memory location 15 add w/ Reg A
// output the content from Reg A to display

this instruction starts from memory location 0000.

####
opcode memory location (if applicable)

eg.

command		opcode	
LDA	⇒	0001	// load to Reg A
ADD	⇒	0010	// Add & store to Reg A
OUT	⇒	1110	// output to display

			RAM address	RAM content	
eg.	LDA	14	0000 :	0001	1110
	ADD	15	0001 :	0010	1111
	OUT		0010 :	1110	XXXX // Don't care
	HLT		0011 :	1111	XXXX
				.	
				.	
				.	
			1110 :	00011100 (28)	// 14 Memory location
			1111 :	00001110 (14)	// 15 Memory location

(PC out → MAR in) (turn them off) → (RAM reads the content from address → RAM out → IR in (instruction is now loaded)) (turn them off) → Increment PC i.e. Enable Counter i.e. PC is made active (turn it off)

Program.

Summary

0000

LDA 14

T0 (1) PC out → MAR IN (turn them off) takes 1 clock cycle

T1 (2) RAM out → IR IN (turn them off)

(3) PC_EN (turn it off)

(4) IR OUT (o/p lower 4 bits only to the bus)

T2 ↓

MAR IN (now we are pointing to mem location 14 of RAM)

(turn them off)

T3 (5) RAM OUT → RA IN (turn them off)

ADD 15

T0 (6) PC OUT → MAR IN (turn them off)

T1 (7) RAM OUT → IR IN (turn them off)

(8) PC_EN (turn it off)

T2 (9) IR OUT → MAR IN (turn them off)

T3 (10) RAM OUT → RB IN (turn them off)

T4 (11) ALU OUT → RA IN (turn them off)

// takes 1 clock cycle.

OUT

T0 (12) PC OUT → MAR IN (turn them off)

T1 (13) RAM OUT → IR IN (turn them off)

(14) PC_EN (turn it off)

T2 (15) RA OUT → ROP IN

output register

HLT

T0 (16) PC_OUT → MAR IN

T1 (17) RAM OUT → IR IN

(18) PC EN

(19) HLT goes high

e! select signal (our notation) = PROG (Ben Eater's notation)

$\overline{LOAD\_PC}$ = $\overline{J}$

Each instruction (eg. ~~PROG~~ LOAD 14) has several steps (eg step ①, ② ... on left page) and each of these steps is called microinstruction.

Control logic will set up the control word appropriately to execute each of the microinstructions.

~~Control logic is updating the content just before the main clock~~ using control word.

Control logic needs to update control word b/w clock cycles (instead of on a ~~cycle~~ clock cycle), hence we use $\overline{CLK}$.

HLT	$\overline{IR\_IN} = \overline{II}$	$\overline{RB\_IN} = \overline{BI}$	$\overline{ALU\_OUT} = \overline{EO}$
$\overline{MAR\_IN}$ = ($\overline{MI}$)	$\overline{IR\_OUT} = \overline{IO}$	ROP_IN = OI	SU = SU
RAM_IN = (RI)	$\overline{RA\_IN} = \overline{AI}$	$\overline{PC\_OUT} = \overline{CO}$	$\overline{LOAD\_PC} = \overline{J}$
$\overline{RAM\_OUT}$ = ($\overline{RO}$)	$\overline{RA\_OUT} = \overline{AO}$	PC_EN = CE	

Two pieces of ~~truth~~ information:

1. Which time step are we in (count $T_0, T_1, \ldots$)? (Found in Decoder's o/p from Control Logic)
2. Which instruction are we executing? (Found in Instruction register)

	INSTRUCTION	STEP	HLT	$\overline{MI}$	RI	$\overline{RO}$	$\overline{IO}$	$\overline{II}$
FETCH	~~XXXX~~	~~000~~						
	XXXX	001						

D1 (D0 is don't care)

	INSTRUCTION	STEP	HLT (MSB)	MI	RI	RO	IO	II	AI	AO (LSB)	SO (MSB)	SU	BI	OI	CE	CO	J	Control word 1	Control word 2
FETCH	XXXX	000	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	40H	~~02H~~ 04H
	XXXX	001	0	0	0	1	0	1	0	0	0	0	0	0	1	0	0	14H	~~04H~~ 08H
CA	0001	010	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	48H	~~00H~~ 00H
LDA 0B	0001	011	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	12H	~~00H~~ 00H
0C	0001	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00H	~~00H~~ 00H
12	0010	010	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	48H	~~00H~~ 00H
ADD 13	0010	011				1							1					10H	20H
14	0010	100							1		1							02H	80H
OUT 72	1110	010								1				1				01H	10H
73	1110	011	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00H	00H
74	1110	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00H	00H
HLT 7A	1111	010	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	80H	00H
		011																	
		100																	

~~For fetch~~ microinstruction

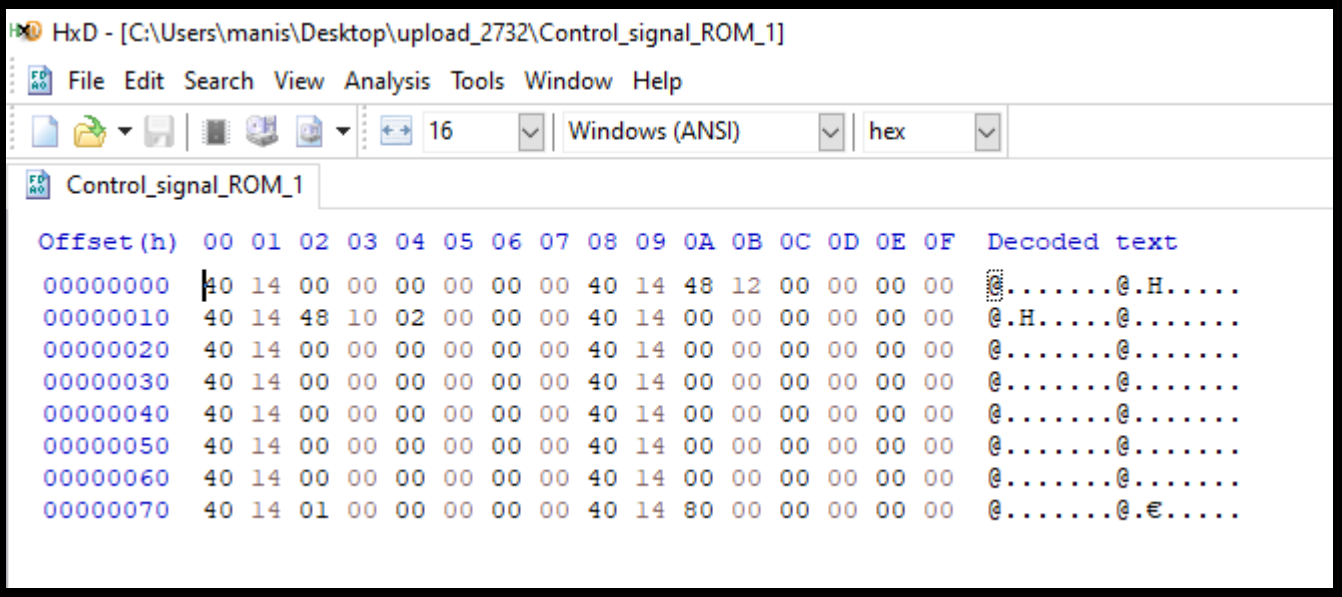
~~0000~~
~~0001~~
~~0002~~
~~0003~~
~~0004~~
~~0005~~

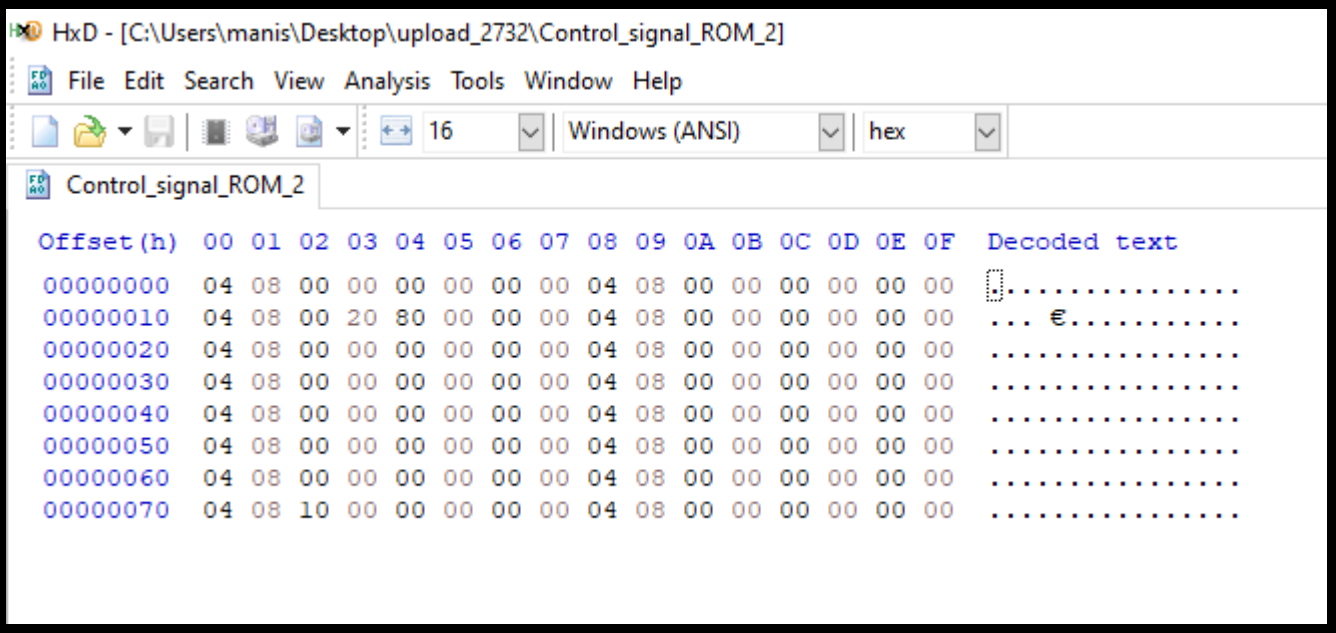
store 40H (ROM1)
store 40H (ROM1)
store 14H (ROM2)
store 14H (ROM1)

Date
Page No.

fetch microinstruction. ⇒ 2732 address location (for ROM)

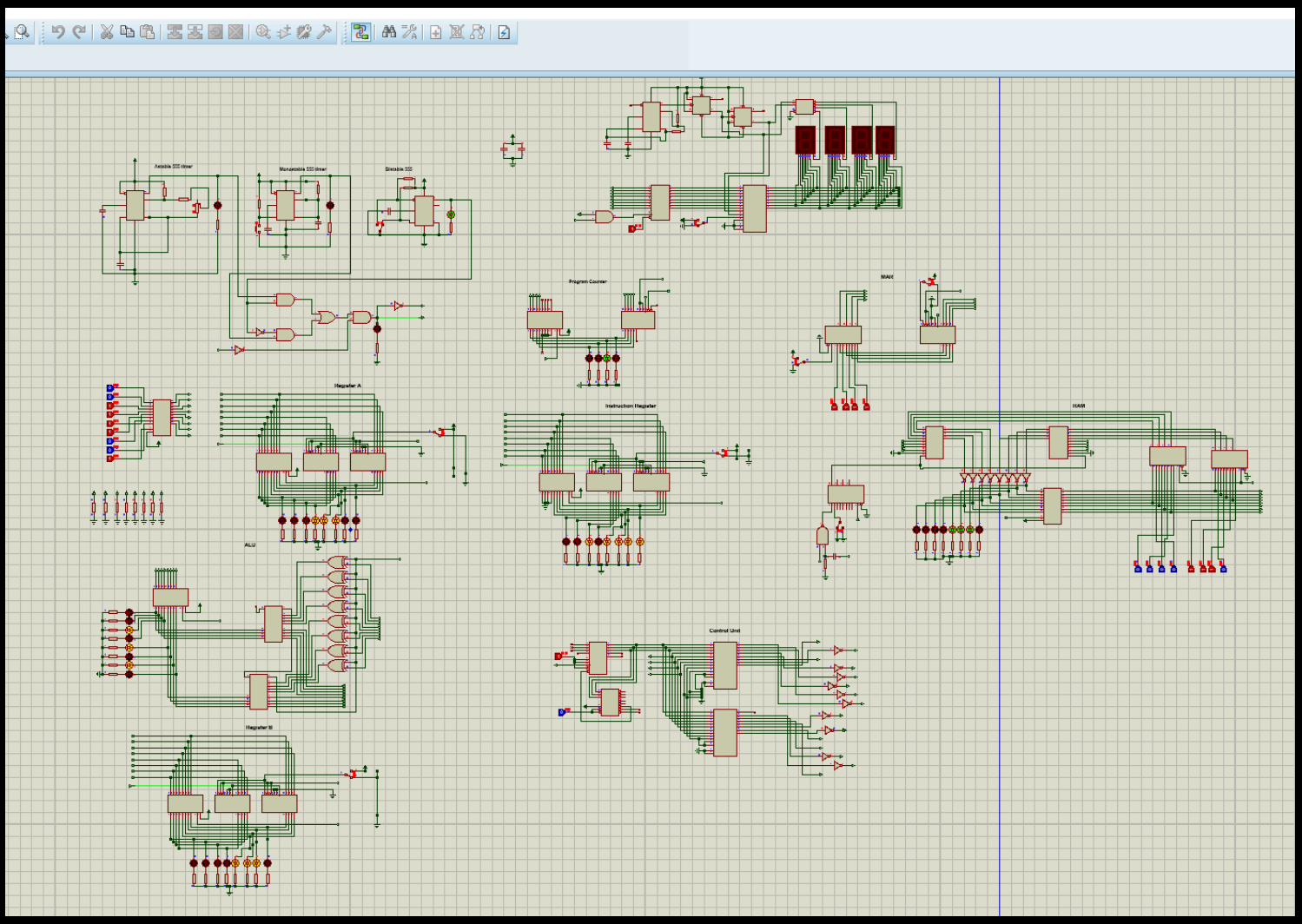
XXXX		XXXX		
0000	000 = 00H	0000	0010 = ~~02H~~	01H
0001	000 = 08H	0001	0010 = ~~0AH~~	09H
0010	000 = 10H	0010	0010 = ~~12H~~	11H
0011	000 = 18H	0011	0010 = ~~1AH~~	19H
0100	000 = 20H	0100	0010 = ~~22H~~	21H
0101	000 = 28H	0101	0010 = ~~2AH~~	29H
0110	000 = 30H	0110	0010 = ~~32H~~	31H
0111	000 = 38H	0111	0010 = ~~3AH~~	39H
1000	000 = 40H	1000	0010 = ~~42H~~	41H
1001	000 = 48H	1001	0010 = ~~4AH~~	49H
1010	000 = 50H	1010	0010 = ~~52H~~	51H
1011	000 = 58H	1011	0010 = ~~5AH~~	59H
1100	000 = 60H	1100	0010 = ~~62H~~	61H
1101	000 = 68H	1101	0010 = ~~6AH~~	69H
1110	000 = 70H	1110	0010 = ~~72H~~	71H
1111	000 = 78H	1111	0010 = ~~7AH~~	79H





#14 Working of Computer

(The above-mentioned program is loaded into the RAM manually.)
Memory location 14 had 28 in it, and memory location 15 had 14 in it. Their sum is 28+14=42, as displayed by the output unit, the seven segment display at top-right corner. Due to high host CPU load (around 65% during the simulation), the simulation runs slow, but it does run and perform as expected.



8-bit-CPU - Proteus 8 Professional - Schematic Capture

File Edit View Tool Design Graph Debug Library Template System Help

Schematic Capture

DEVICES

2SEG-MPX1-CC
74LS08
74LS32
74LS76
74LS138
74LS139
74LS157
74LS161
74LS173
74LS189
74LS245
74LS273
74LS283
555
2732
2816
7476
BATTERY
BUTTON
CAPACITOR
DIPSW_4
LED-GREEN
LED-RED
LED-YELLOW
LOGICSTATE
NAND
NOT
POT-HG
RESISTOR
SW-SPDT
SWITCH
XOR
[74HC76]
[74LS08]
[74LS32]
[74LS76]
[74LS139]
[7476]

12 Messag...

ANIMATING: 00:00:22.238879 (CPU load 65%)

x: -22800.0 y: -11200.0

1:03 PM