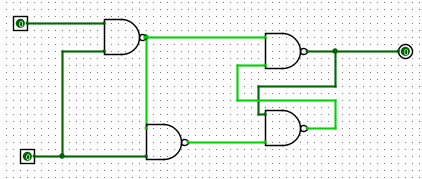
CPU designed following the Ross McGowan "Design a CPU" Course

<https://www.udemy.com/course/design-a-cpu>

<https://www.buthowdoitknow.com/cpu_simulator.html>

***Section 1: RAM, ALU, Registers, and Clock***

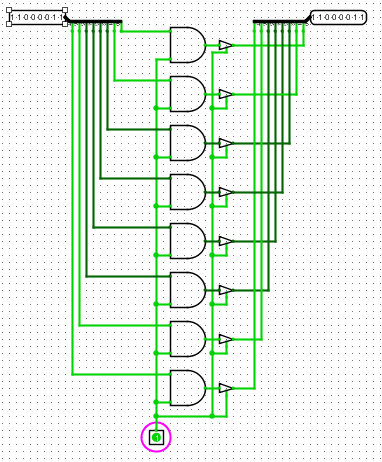
CPU\_MEM\_1



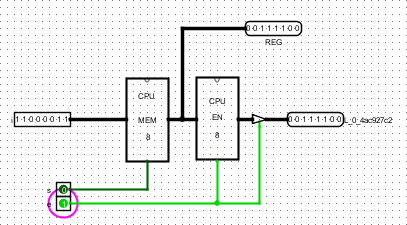
CPU\_MEM\_8



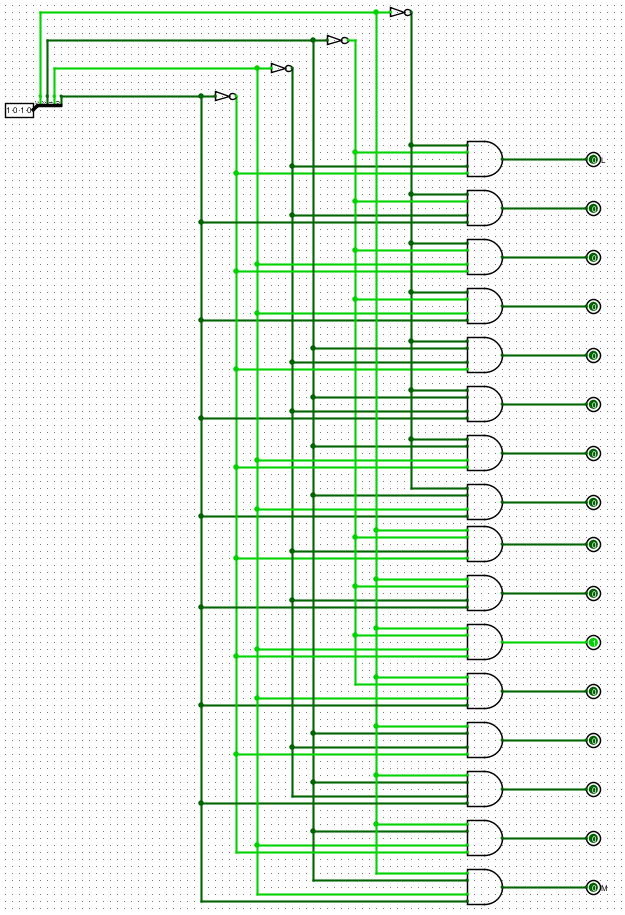
CPU\_ENABLE



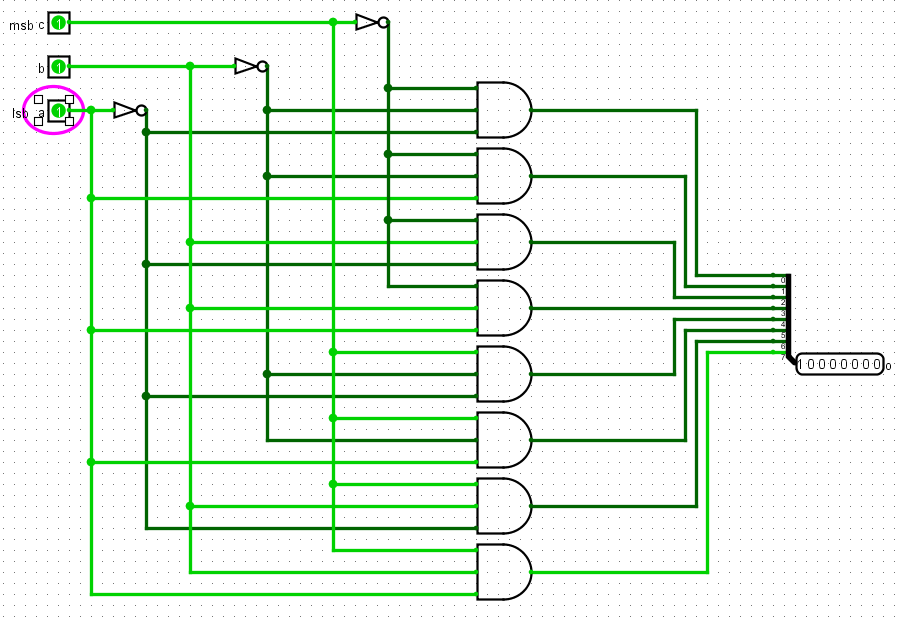
CPU\_REG\_8



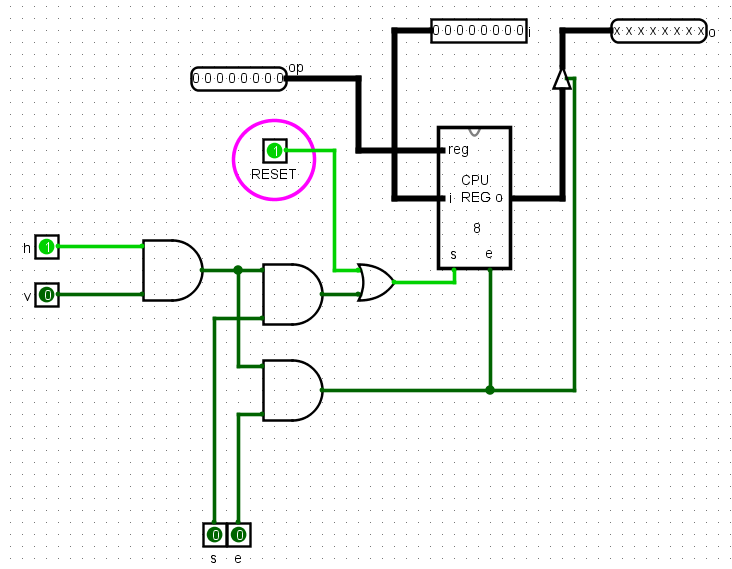
CPU\_DECODER\_4x16



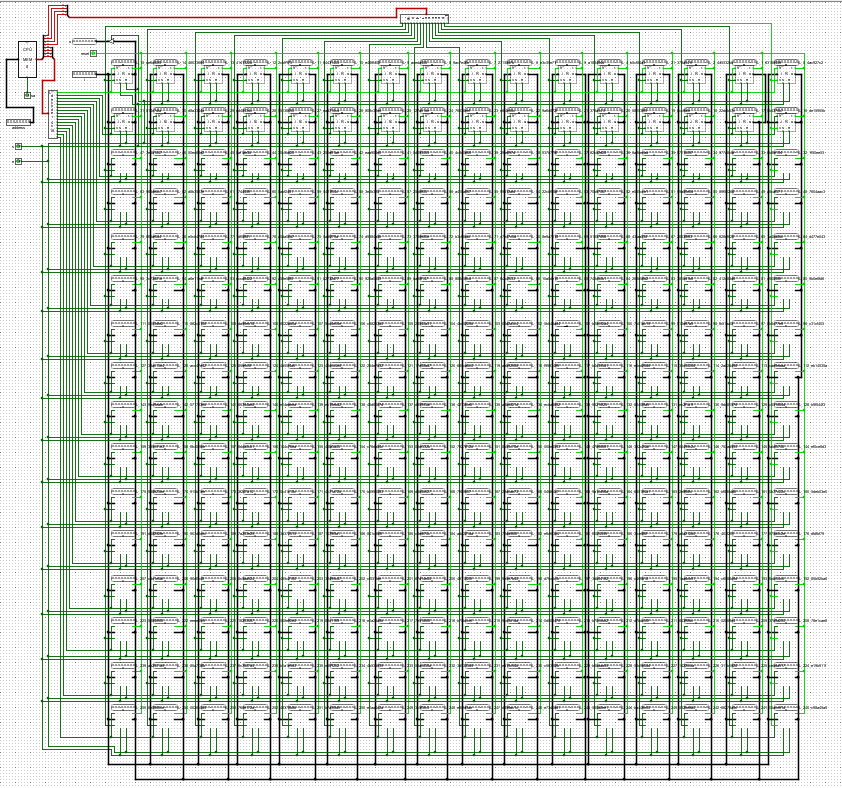
CPU\_DECODER\_3x8



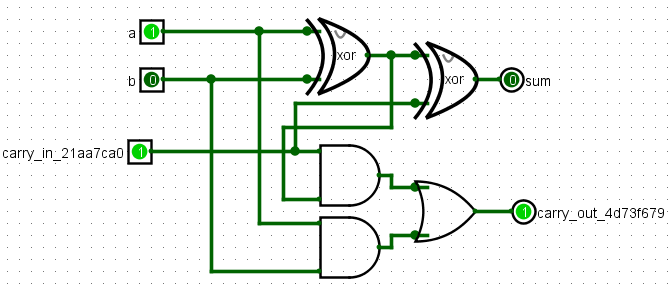
CPU\_REG\_BYTE



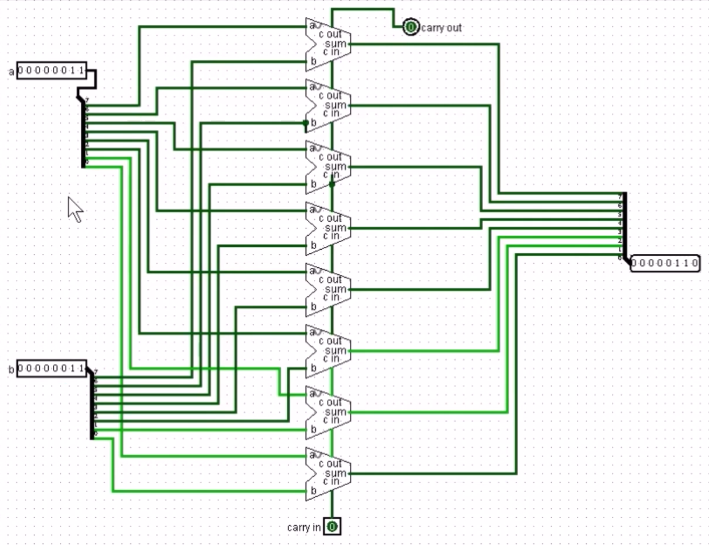
CPU\_RAM\_256



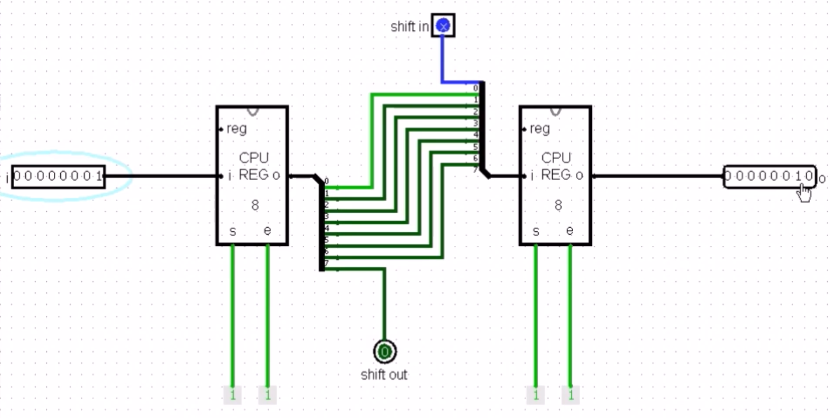
CPU\_ADDER



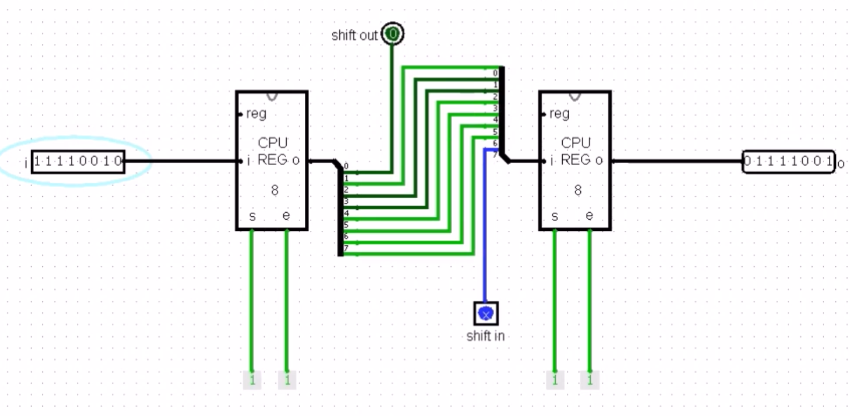
CPU\_ADDER\_8



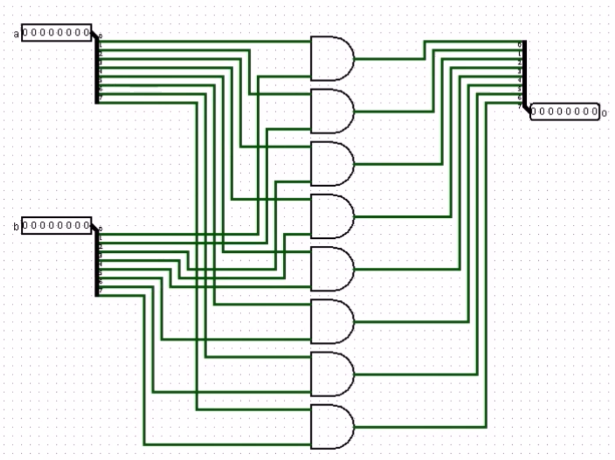
CPU\_SHR



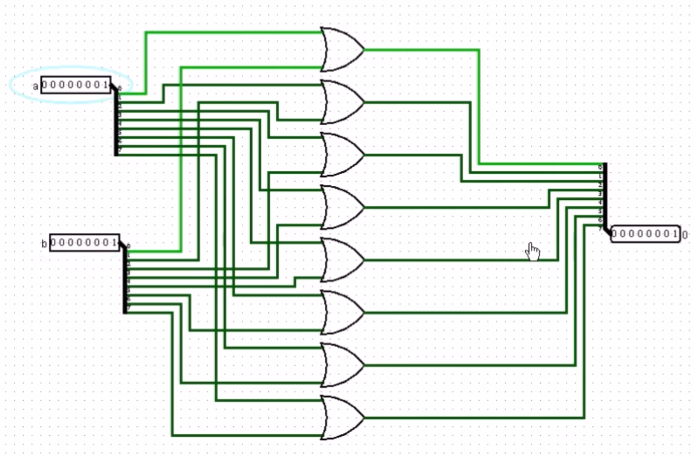
CPU\_SHR



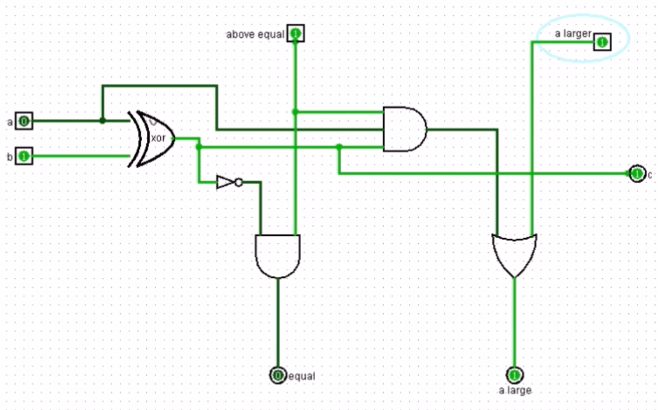
CPU\_AND



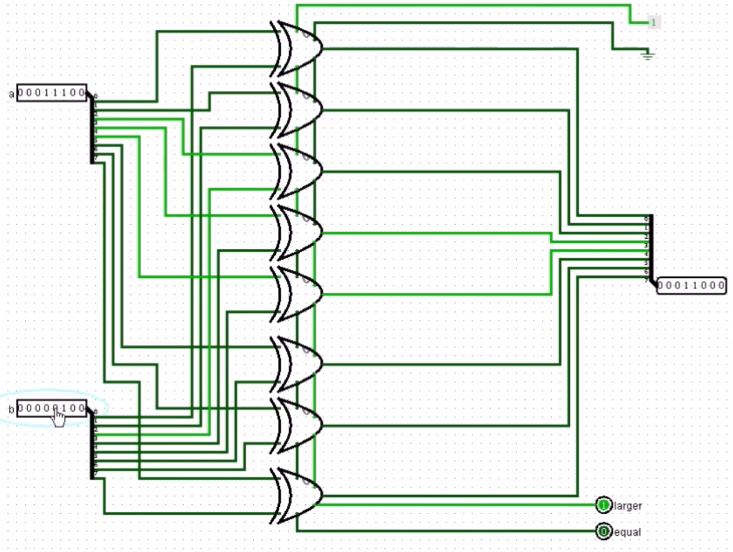
CPU\_OR



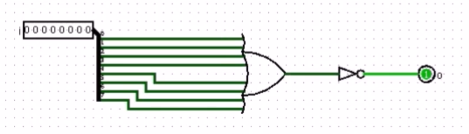
CPU\_XOR\_LE (Larger Equal)



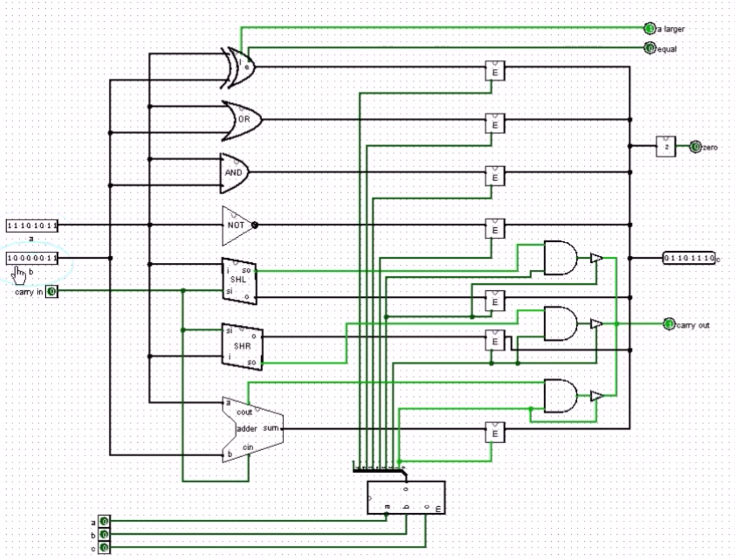
XPU\_XOR\_LE\_8



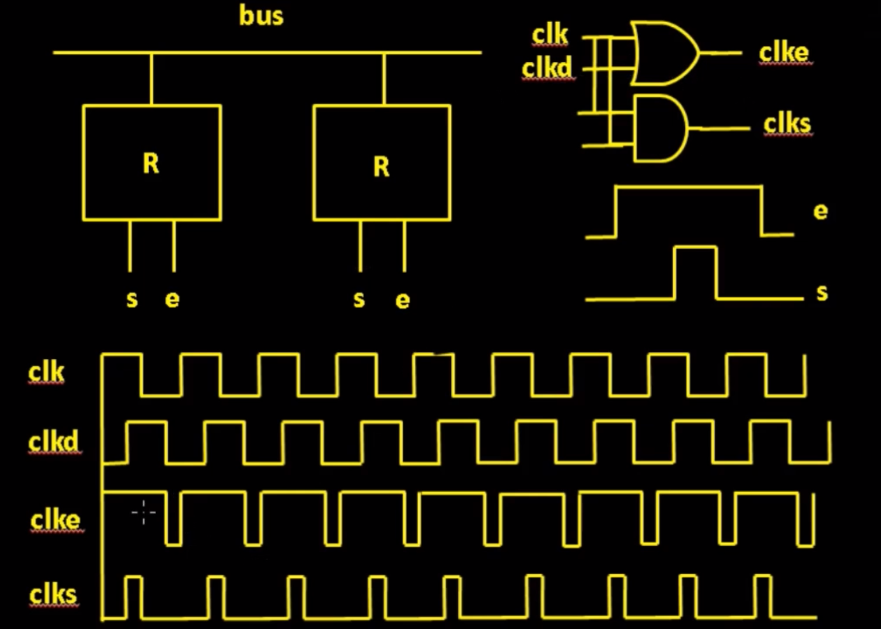
CPU\_Z (Zero)



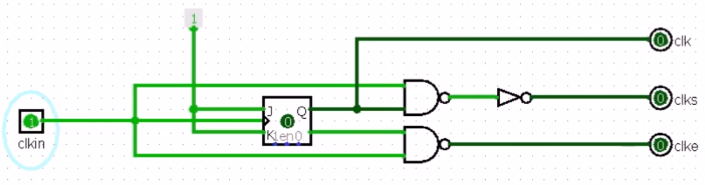
CPU\_ALU



clk, clkd (delayed), clke (enable), clks (set) - designed to hold write enable high, and hold the data on the bus, while memory set completes the store cycle

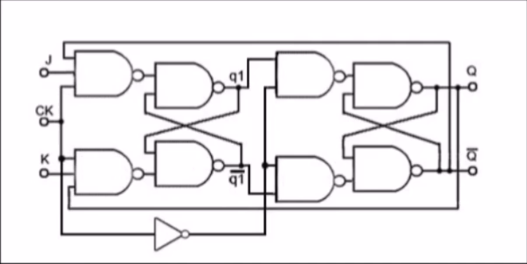
******

CPU\_CLK

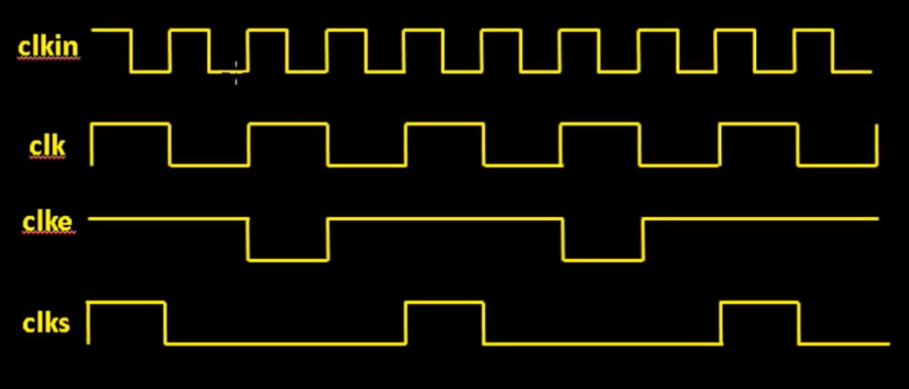


Flip-flop creates a half-cycle clock output instead of a delayed clock output (that is fine for this use case). AND gate used to create clks. AND gate combined with “feed-forward clkin and Qn output” in place of OR gate.

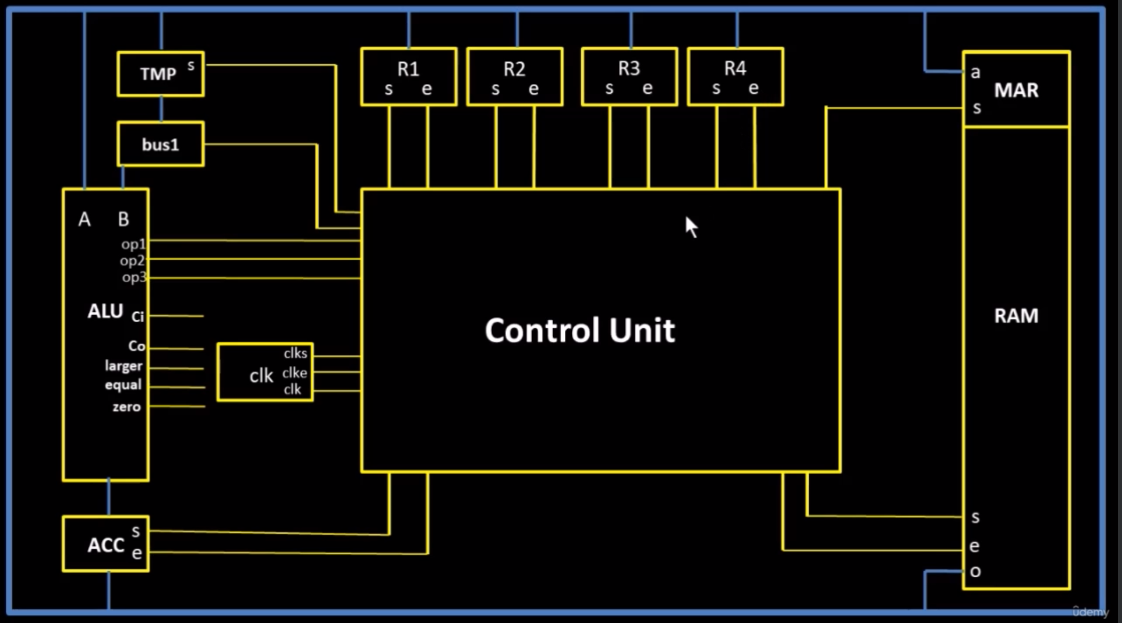
JK Flip-flop Circuit



CPU\_CLK Circuit Timing Diagram



CPU - Simplified Block Diagram



MAR - Memory Address Register

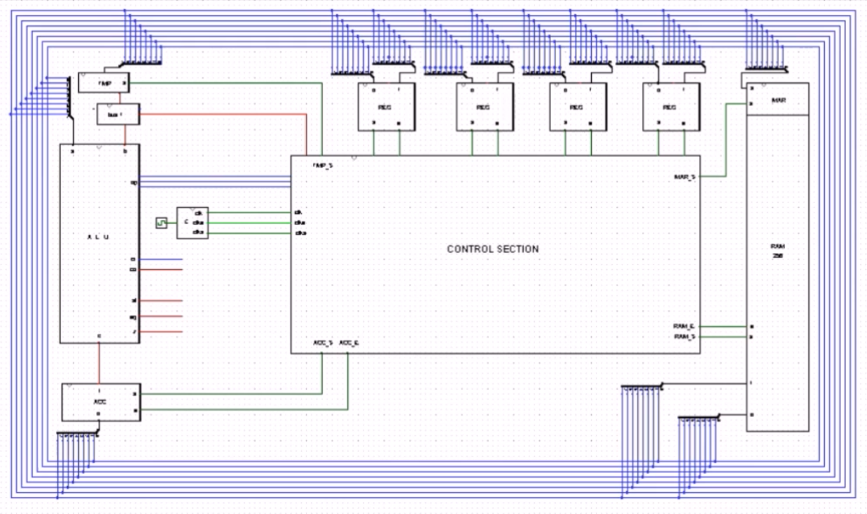
ACC - Accumulator - Just another register like R1-R4, but with a fancy name. Gates ALU output onto the bus.

Bus1 - It’s only function is putting 1 into ALU Input B.

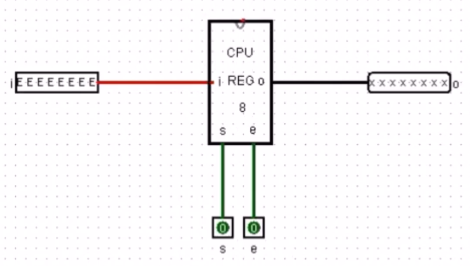
TMP - Temporary Register - A means of storing a value to feed into ALU Input B while the Bus holds the value passed into ALU Input A

Control Unit - Controls all the timing of the Bus gating and register sets/enables

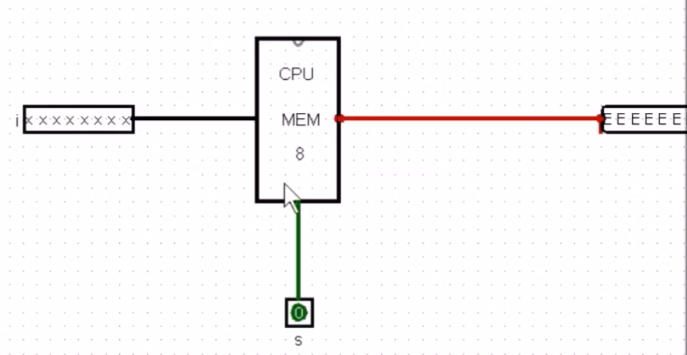
CPU - Simplified Logisim Implementation



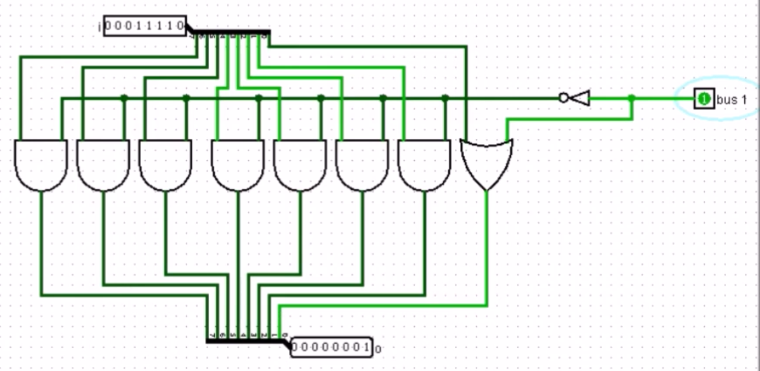
CPU\_ACCUMULATOR



CPU\_TMP - Just a CPU\_MEM\_8 (aka, a CPU\_REG\_8 without an enable)

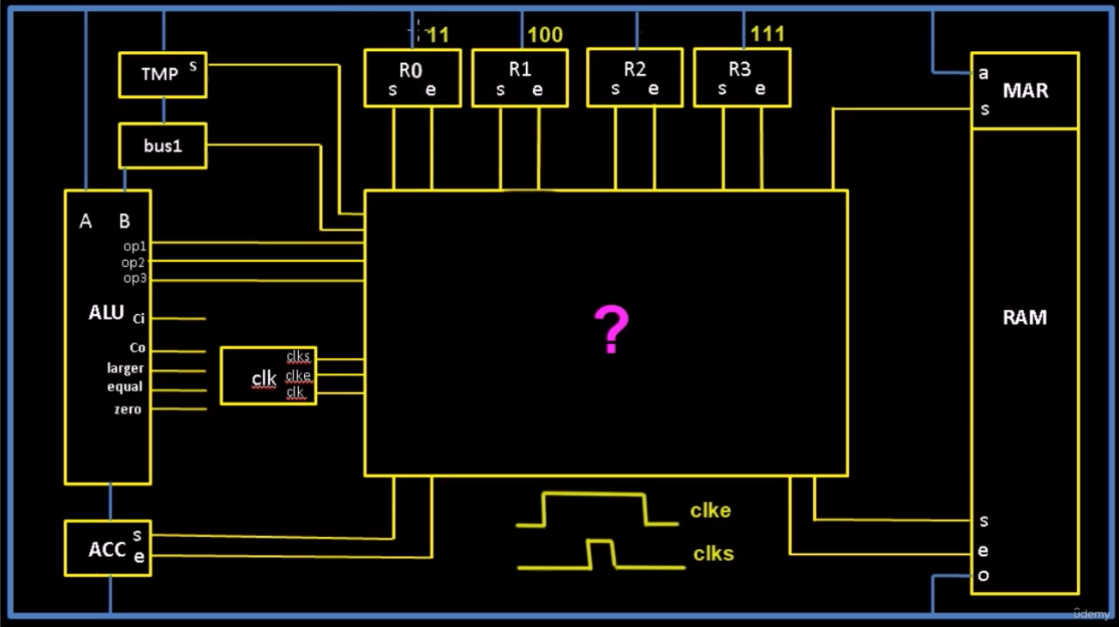


CPU\_BUS\_1



When bus 1 Input is low, the NOT gate ensures the value on the bus is passed directly through to ALU Input B. When bus 1 Input is low, a value of 1 is passed into ALU input B.

***Section 3: Control Section***

******

How would we control a state machine to add 3 plus 4 and save the value to a register?

Assume 3 is loaded into R0, 4 is loaded into R1, and we’re storing the output in R3.

Write Input to TMP (“Write to B”)

1. R1 Enable High to write R1 onto the bus.
2. TMP Set High to begin storing the bus data to TMP.
3. TMP Set Low to end storing the bus data to TMP.
4. R1 Enable Low to remove R1 from the bus.

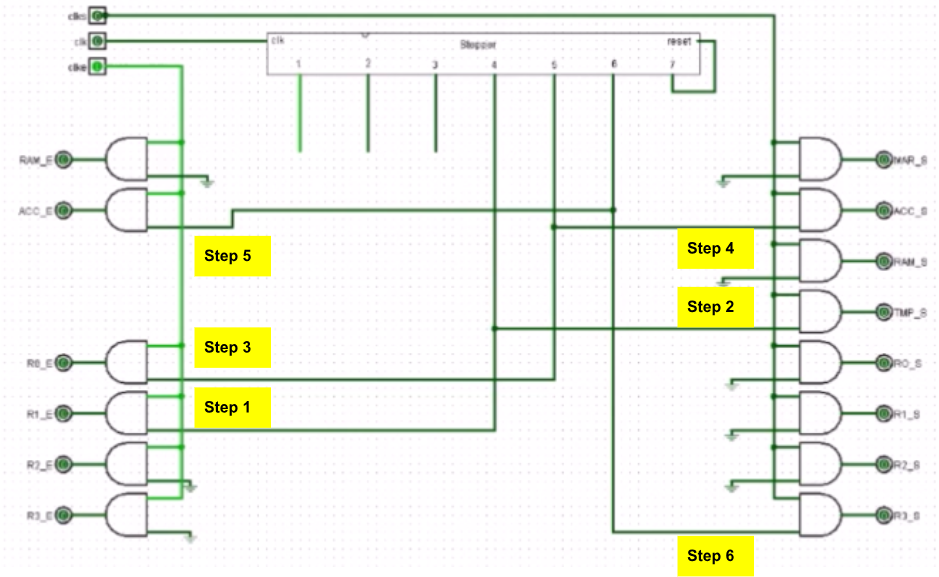
Write Input to A / Arithmetic / Write Output to ACC

1. R0 Enable High to write R0 onto the bus. The bus is always connected to ALU Input A.
2. Write an Op Code to the ALU (000 for Addition)
3. ACC Set High to write the ALU output into ACC.

Write ACC to R3

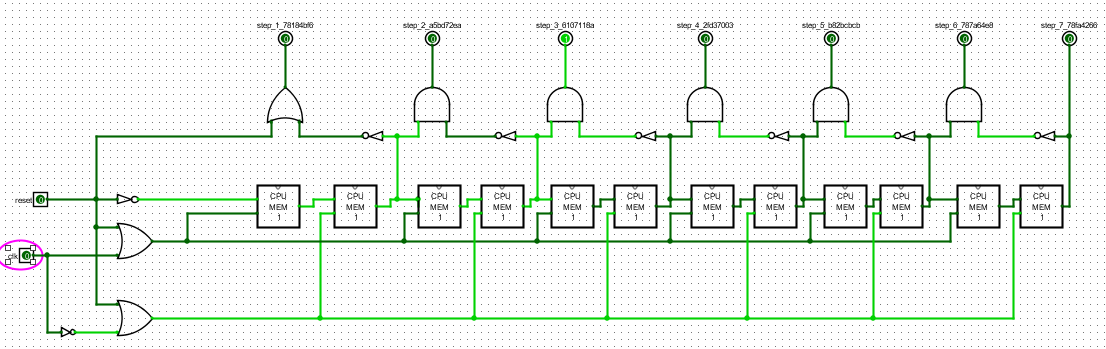
1. ACC Enable High to write ACC onto the bus.
2. R3 Set High to begin storing the bus data to R3.
3. R3 Set Low to end storing the bus data to R3.
4. ACC Enable Low to remove ACC from the bus.

CPU\_CONTROL\_SECTION - Simplified for Adding 4 + 3 and Storing to R3 Example above

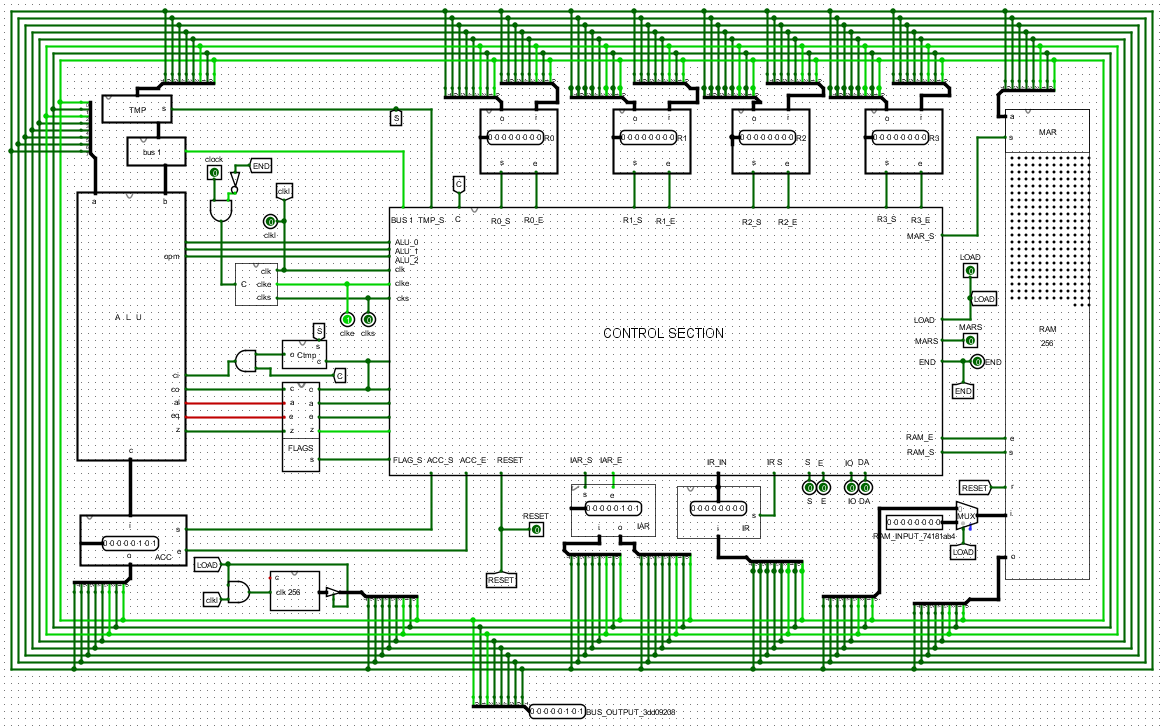


Note Steps 1&2, 3&4, and 5&6 are performed in pairs per the stepper wires 4, 5, and 6. Note that all the Enables require both the Stepper and clke to be high. Note all the Sets require both the Stepper and clks to be high. Note the stepper is driven off of the primary clk.

CPU\_STEPPER



CPU - More Detailed (with IAR and IR)



Default Program - Run the CPU by toggling ‘clock’. No program loaded so CPU loads from IR ‘0’, enables Bus1 and IAR, and since the ALU defaults to ADD, the ACC continuously counts by 1 incrementing the IAR.

Added:

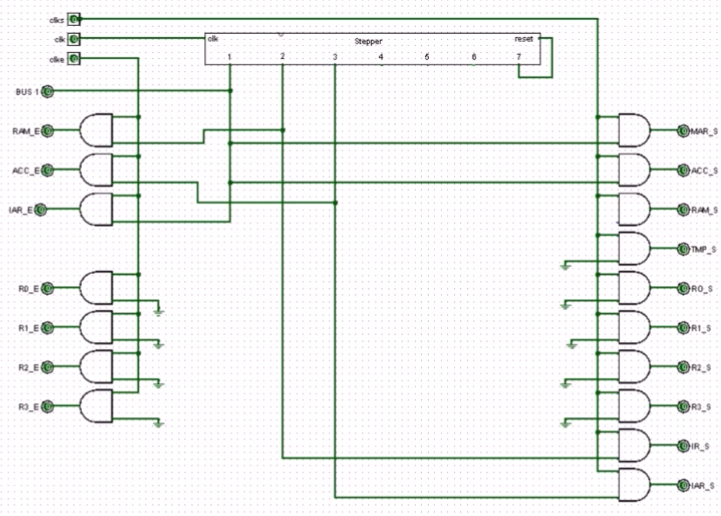
IAR - Instruction Address Register - Where the next Instruction in Stored

IR - Instruction Register - Where the Current Instruction is Stored

FLAGS - Carry, A, E, Zero

RAM\_INPUT - Allows for manual data input into RAM

CPU\_CONTROL\_SECTION - Simplified for Instruction Fetch Cycle



***Section 4: Assembly Language and Debugger***

***Section 5: Load and Run a Program***

***Section 6: Other Interesting Stuff***