

Romulus I: 16-Bit Computer

A Technical Report submitted to the Department of Electrical and Computer Engineering

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Wolfgang Ploch

Spring, 2025

Technical Project Team Members

Wolfgang Ploch

August Bresnaider

Will Rimicci

Austin Chappell

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Todd Delong, Department of Electrical and Computer Engineering

WAWA Final Report: ROMULUS I

1st August Bresnaider

*Charles L. Brown Department of Electrical & Computer Engineering
University of Virginia
Charlottesville, US
august@virginia.edu*

2nd Austin Chappell

*dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID*

3rd Wolfgang Ploch

*dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID*

4th Will Rimicci

*dept. name of organization (of Aff.)
name of organization (of Aff.)
City, Country
email address or ORCID*

I. STATEMENT OF WORK

A. August Bresnaider

Lead software design. Directed instruction set architecture and finite state machine design.

B. Wolfgang Ploch

Lead hardware design. Participated in instruction set architecture design and high level architecture design. Designed circuitry, designed PCB, and wrote hardware description sections for boards 1, 2, 3, 5, and 6. Assisted in circuitry design for board 4. Soldered components on all 6 boards. Lead case and control panel design. Assembled entire device. Troubleshot hardware and implemented hardware solutions during testing. Designed and provided teletype peripheral and wrote driver programs for it. Wrote cost and physical constraint sections of the final report.

C. Will Rimicci

Participated in instruction set architecture design and high level architecture design. Designed circuitry, designed PCB, and wrote hardware description section for board 4. Wrote code to generate control signals FSM flash data. Rigorously tested machine code simulator and assembler. Wrote several test programs to be run on simulator and computer. Troubleshot hardware and helped design hardware solutions during testing. Recorded tests for debugging purposes. Designed circuitry, designed PCB, and soldered general input/output peripheral. Wrote background, societal impact, intellectual property, and full system test sections of the final report. Helped keep team focused on deliverables when due over the course of the semester.

II. TABLE OF CONTENTS

CONTENTS

I	Statement of Work	1
I-A	August Bresnaider	1
I-B	Wolfgang Ploch	1
I-C	Will Rimicci	1

II	Table of Contents	1
III	Background	3
IV	Project Description - Specifications	3
IV-A	Instruction Set Architecture	3
IV-A1	Overview	3
IV-A2	Move instructions (Opcodes 0-3)	4
IV-B	Finite State Machine	4
V	Project Description - Software	4
V-A	Assembler ("romASM")	4
V-B	Compiler ("Hawk")	4
VI	Project Description - Hardware	4
VI-A	Board 1	4
VI-A1	Overview	4
VI-A2	LED Displays	4
VI-A3	Register Design	4
VI-A4	Addressing the Registers	4
VI-A5	Modifications	4
VI-B	Board 2	4
VI-B1	Overview	4
VI-B2	Clock Signals	5
VI-B3	Pause Handling	9
VI-B4	Reset Timing	9
VI-B5	Timing Signals and Counters	9
VI-B6	Interrupt Handling Correction	10
VI-B7	Finite State Machine Flash Chip Lookup Table	10
VI-C	Board 3	13
VI-C1	Overview	13
VI-C2	Memory Address Register (MAR)	14
VI-C3	Address Decoding	14
VI-C4	Mapped IO Port	14
VI-C5	Mapped IO Port Control Signal Logic	14

	VI-C6	Memory SRAM and Supporting Circuitry	14
VI-D	Board 4		15
	VI-D1	Arithmetic and Logic Unit (ALU)	15
	VI-D2	Instruction Register (IR) . .	15
	VI-D3	Compare Logic	18
	VI-D4	7-Segment Hex Displays . .	18
VI-E	Board 5		18
	VI-E1	Overview	18
	VI-E2	Program Counter	18
	VI-E3	Stack Pointer Counter	19
	VI-E4	Program Counter and Stack Pointer Counter LED Displays	21
	VI-E5	Dip-switches for Interrupt Jump Location	21
	VI-E6	Instruction SRAM	21
	VI-E7	Stack SRAM	22
	VI-E8	Interrupt Handling	22
VI-F	Board 6		23
	VI-F1	Overview	23
	VI-F2	Power Regulation Design . .	23
	VI-F3	Over Current and Over Voltage Protection	24
VII	Test Plan		25
VIII	Physical Constraints		26
	VIII-A	Design and Manufacturing Constraints .	26
	VIII-B	Tools Used in the Project	27
	VIII-B1	Visual Studio 2022	27
	VIII-B2	Digital	27
	VIII-B3	Multisim	27
	VIII-B4	Arduino IDE and Arduino Mega	27
	VIII-B5	Notepad	28
	VIII-B6	Physical Tools	28
	VIII-C	Cost Constraints	28
	VIII-D	Producing a Production Version	29
IX	Societal Impact		29
X	External Standards		29
XI	Intellectual Property Issues		29
XII	Timeline		30
XIII	Costs		31
XIV	Final Result		31
XV	Engineering Insight		31
XVI	Future Work		31
	References		31
XVII	Appendix		32

Abstract—Our project revolves around the design and fabrication of a 16-bit CPU and supporting memory using Flash memory, 74XX series logic chips, and passive components. Aimed to be a educational device for students to learn computer architecture, digital logic, machine code, and electronics, the Romulus I is equipped with LEDs on the internal registers, 7-segment displays, debugging tools, and multiple clock speeds. The scope of this project also includes a compiled language and an assembler providing increasing layers of abstraction, allowing for either a top-down or bottom-up approach to teaching about computing systems and organization.

III. BACKGROUND

We chose this project for a number of reasons. First, we were inspired by our experiences in other classes like Computer Systems and Organization and Digital Logic Design. After using the Digital open-source logic simulator [1] and the ToyISA simulator [2], both of which are virtual tools, we felt that we could create a physical, hands-on teaching tool that would help students understand computers in a new way. Second, we wanted to see if we could use what we had learned in those classes through lectures and homeworks to make something new. Finally, we just thought it would be a fun and interesting challenge.

Throughout this project, we used many techniques we learned in our classes. The general structure of the computer, having a control signals FSM, an ALU, a program counter, and memory, was material from Digital Logic Design. In that class, we created a programmable computer with those blocks in Digital [1] (though most of the designs of the blocks themselves were different from the approach we were taught in that class). The design of the instruction set architecture and assembly were from Computer Systems and Organization 1. The knowledge on how to create an assembler and compiler were from Computer Systems and Organization 2. Finally, the knowledge on how to create a PCB layout, order parts, and solder circuits were from the ECE Fundamentals series.

Though this project was based on and inspired by a number of past works, it is a new and distinct product. As mentioned above, we took inspiration from the virtual tools we used in our classes, like Digital [1] and ToyISA [2]. While these are fantastic tools for learning, they are only virtual programs, and do not provide a physical, interactable model from which to learn. Another similar past work is Ben Eater's 8-bit breadboard computer [3]. In his video series, Ben Eater creates a full-functional, programmable 8-bit computer on a set of breadboards using only simple logic gates. It has a clock, registers, ALU, RAM, program counter, and control logic. While this project is similar to our own, it is built on a set of breadboards, meaning the circuits are not as permanent as ours made from soldered PCBs, and it is much less versatile than ours: it is only eight bits and there are only two general purpose registers, compared to our 16 bits and 16 general purpose registers. Our work is different from any projects before it.

IV. PROJECT DESCRIPTION - SPECIFICATIONS

A. Instruction Set Architecture

1) *Overview*: The instruction set architecture (or ISA, for short) acts as a specification that defines the operations the CPU is capable of, as well as their binary representations. By giving an operation a mapping, we are including it in the capabilities of the CPU. Designing an ISA is a balancing act: if an ISA is too simple, common operations may become arduous (i.e., not including A & B requires the programmer to do $(\overline{A}|B)$ any time a bitwise AND is needed), or in the worst case, the CPU may not be Turing complete. On the converse, if the ISA is overly complicated, it may require an overly complicated hardware implementation. To avoid the negative effects of these two poles, our ISA should be:

- Turing complete

What this means and why this is important will be elaborated on further once our ISA is described, but in short, if our ISA is not Turing-complete, there is a strict and provable upper bound on the computational power of our hardware that means our CPU will not be able to compute any given algorithm.

- Logically comprehensive

While the bare minimum number of operations needed to make a Turing-complete ISA is remarkably small (by our count, it's roughly 5 or 6), it's to include more instructions for ease of use purposes, much like the bitwise AND example above, to avoid frustration and code repetition on the programmer's part.

- Easy to code / represent in hexadecimal

As one of the revisions to ToyISA that we aim to implement, the bitcode should be intuitive and require minimal bit-fiddling on the programmer's part. ToyISA had 8-bit instructions, with the most significant 4 bits representing opcode, the next 2 representing rA (parameter 1), and the least significant 2 representing rB (parameter 2) or an optional set of selector bits for unary operations. When programming using the bitcode, it is often a hassle having to compute the lower hexadecimal digit when changing rA or rB, so our ISA should split each 16-bit value into groups of 4 or 8 wherever possible to avoid this.

- Able to be utilized in higher level abstraction

The ISA should be fully functional when programming using bitcode, but there should also be instructions geared towards higher level concepts. For example, while a stack pointer and instructions incrementing and decrementing the stack is not strictly required, it gives way to creating a call stack and allowing for the writing of functions, loops, and recursion: features only truly available in romASM and / or Hawk. The ISA we designed for Romulus I in order to solve these problems to the best of our ability is given by Table I.

As referenced in the table, we settled on using 16 instructions, which allows our opcode to fit within the most-significant 4 bits of the instruction. Bits 8-11 generally act as placeholder for R_A , the first parameter in many operations. From there, bits 4-7 are reserved for R_B , the second parameter for many operations, and lastly, bits 0-3 are usually reserved

for R_Y , the register in which to store the result. Each group of 4 instructions are similar to each other, and so the next 4 sections will discuss the "Move", "Bitwise", "Arithmetic / Logical", and "Control Flow" instruction groups.

2) *Move instructions (Opcodes 0-3)*: The first 4 opcodes deal with moving values into and between registers, as well as interacting with memory.

B. Finite State Machine

Gus

V. PROJECT DESCRIPTION - SOFTWARE

A. Assembler ("romASM")

Gus

B. Compiler ("Hawk")

Gus

VI. PROJECT DESCRIPTION - HARDWARE

A. Board 1

1) *Overview*: Board 1 contains the register file and supporting circuitry. The register file was chosen to have 16 registers because in order to address 16 registers, four bits are needed. Using 4 bits to select the register allows three registers to be chosen and the four-bit opcode to be specified all in one 16-bit instruction. Each register stores 16 bits. Each bit of each register was designed to be indicated with an LED. The SN74HC573 8-bit register was chosen to store the contents of each register. Because each are 16-bit, two 74HC573s are needed per register. Unfortunately, the tri-state buffer built into the 74HC573 was not able to be used as we needed access to the data even when it is not active. The OE# of all the buffers were grounded so the output always reflected the data. The full Schematic of the register file board can be seen in Figure 1. It is referred to as board 1 in other sections.

2) *LED Displays*: The LEDs sourced were tested with various resistor values. A pleasant brightness was found with 820 Ω resistors. One LED draws 4.2 mA in this arrangement. This means the board could draw up to 1.075 amps from just the LEDs if 0xFFFF is stored in all the registers. To save space on the board, instead of using one discrete resistor for each LED, 8-resistor resistor arrays were sourced. These are SIP-9 components that have 8 resistors with one common pin. The anodes of each LED connect to the output of the register and the cathodes are grouped into 8s and connected to the resistor array, 2 resistor arrays per register. The common pin of each resistor array is connected to ground through a 2N3904 with a 1 K Ω resistor on the base. The other ends of the resistors are all connected to a common point. The transistor allows all the LEDs to be turned off with a single signal. This will be useful when running the computer at the highest speed.

3) *Register Design*: The outputs of the registers are connected to a SN74HC541 buffer IC. It is an 8-bit device so two of them are needed per register. The OE# are controlled by the decoder. All outputs of these buffers are connected by bit and then feed into two more 74HC541 so act as the final output enable. The schematic of a single register can be seen below in Figure 2.

4) *Addressing the Registers*: Two 74LS154s were used to make the decoder. One is connected to the 4-bit address and its outputs connect directly to the OE# on the buffer pairs. The outputs of the 74LS154 and OE# inputs on the 74HC541 are both active low. The enable pins of the decoder are connected to ground to constantly enable the decoder. This way, one of the registers is constantly on the internal bus. The final buffer mentioned above connects the selected one to the bus using the OE# inputs on the board. The other decoder is used to write to the registers. The two WE# s connect to its two enable pins. Each output passes through a NAND gate whose other input is wired to MWE# . Pulling MWE# low causes all the registers to be written simultaneously. This is used during an asynchronous reset. Its address pins are also connected to the 4-bit address input. Each chip on the circuit has a 1 nF capacitor across its power pins to decouple the IC. The boards inputs are OE1# , OE2# , WE1# , WE2# , 4-bit address, LED CTRL. It has a 16-bit IO connection for the data bus. A trace width calculator was used to design the widths of the traces on the PCB to ensure they could supply the proper power. A simplified schematic of the control circuitry can be seen below in Figure 3.

5) *Modifications*: The PCB needed two modifications to work properly after its design and order. The first modification was the addition of the second OE# input to the board. The overall low-level architecture was not solidified before the PCB was ordered and it was not yet known that a second OE# would be needed. It was created by adding a daughter board on the PCB with a single 74HC32. The second modification was adding parallel RC networks on each of the four-bit address inputs for addressing the register. Transient issues were observed when multiple registers started to write at the same time due to errant pulses on these lines. The Parallel RC networks suppressed these and stopped the transient voltages. The resistor and capacitor values are 1 M Ω and 1 nF respectively. One of these networks can be seen in Figure 4.

B. Board 2

1) *Overview*: The purpose of the FSM Clock and Reset Board (Board 2) is to generate the clock signals, the logic to switch between them, the reset signal, the control signals for the registers and other items on the other boards, and the timing signals used by the RAM chips. The control signals for the rest of the boards are generated using the outputs of flash memory chips, the SST39SF010A to be exact. The reason that this design approach was taken was to allow for the finite state machine that runs the computer to be customizable. This proved to be a smart choice, as the finite state machine logic stored in these chips was rewritten multiple times to

TABLE I
INSTRUCTION SET ARCHITECTURE MAPPING

Opcode (Op)	Instruction	Registers	Bits (MSB ... LSB)			
0 (0000)	Register to Register	$R_A \rightarrow R_Y$	Op (4)	R_A (4)	X (4)	R_Y (4)
1 (0001)	Immediate to Register	$I \rightarrow R_Y$	Op (4)	X (8)		R_Y (4)
2 (0010)	Register to Memory	$R_A \rightarrow M_{RB}$	Op (4)	R_A (4)	R_B (4)	X (4)
3 (0011)	Memory to Register	$M_{RA} \rightarrow R_Y$	Op (4)	R_A (4)	X (4)	R_Y (4)
4 (0100)	Bitwise OR	$R_A \text{ --- } R_B \rightarrow R_Y$	Op (4)	R_A (4)	R_B (4)	R_Y (4)
5 (0101)	Bitwise NOT	$\sim R_A \rightarrow R_Y$	Op (4)	R_A (4)	X (4)	R_Y (4)
6 (0110)	Bitwise AND	$R_A \&\& R_B \rightarrow R_Y$	Op (4)	R_A (4)	R_B (4)	R_Y (4)
7 (0111)	Bitwise XOR	$R_A \wedge R_B \rightarrow R_Y$	Op (4)	R_A (4)	R_B (4)	R_Y (4)
8 (1000)	Add	$R_A + R_B \rightarrow R_Y$	Op (4)	R_A (4)	R_B (4)	R_Y (4)
9 (1001)	Subtract	$R_A - R_B \rightarrow R_Y$	Op (4)	R_A (4)	R_B (4)	R_Y (4)
A (1010)	Logical NOT	$!R_A \rightarrow R_Y$	Op (4)	R_A (4)	X (4)	R_Y (4)
B (1011)	Logical Shift Right	$R_A >> 1 \rightarrow R_Y$	Op (4)	R_A (4)	X (4)	R_Y (4)
C (1100)	Push	$(R_A \text{ or } PC+2) \rightarrow \text{Top stack}$	Op (4)	R_A (4)	Op2 (1)	X (7)
D (1101)	Pop	$\text{Top stack} \rightarrow (R_Y \text{ or } PC)$	Op (4)	X (4)	Op2 (1)	X (3) R_Y (4)
E (1110)	Unconditional Jump	$R_A \rightarrow PC$	Op (4)	R_A (4)	X (8)	
F (1111)	Conditional Jump	If $(R_B \text{ ?? } 0): R_A \rightarrow PC$	Op (4)	R_A (4)	R_B (4)	Flags (4)

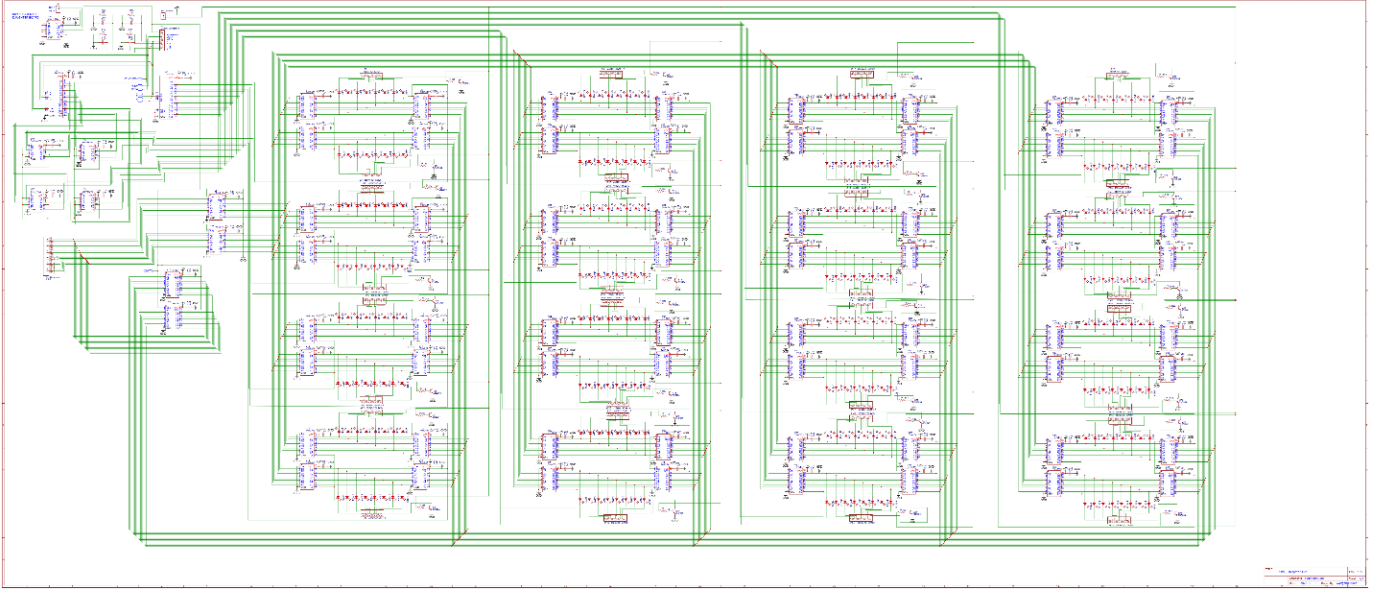


Fig. 1. Full Schematic of Register File Board

compensate for problems discovered during troubleshooting. The full circuit can be seen below in Figure 5. It is important to note that this circuit was made using discrete gates rather than full chips. This was done to make the schematic more intelligible. The schematic used to create the PCB looks different but represents the same circuit. 1 nF bypass capacitors are placed on the power rails of every chip

2) *Clock Signals*: The board creates five different clock signals that can be selected using the rotary switch mounted on the control panel. The speed options are 4 MHz, 400 Hz, 8 Hz, manually increment one full instruction, and manually increment one sub-instruction. The 4 MHz and 400 Hz clock signals were generated using a single SN74S124 oscillator chip. It contains 2 independent oscillators. The chip requires a voltage present at the FC pin, a current through the RNG pin and a capacitance across the two CX pins. Unfortunately, the

datasheet graphs for the voltage, current and capacitance values did not match up completely with the oscillation frequency. Different values were used until the desired frequencies were obtained. The oscillator circuit can be seen below in Figure 6.

The 8 Hz oscillator was created using an NE555 timer chip. It was wired in a generic astable mode with the addition of a 1N4148 diode to bypass the second resistor. The purpose of this diode is to make the capacitor charge through one of the resistors and discard through the other. Setting the values of both resistors to the same value creates an oscillator with a perfect 50% duty cycle. This signal is more pleasing to the eye of the user. With this modification, the equation for the oscillating frequency becomes

$$f_{osc} = \frac{1}{1.34RC}$$

10 KΩ resistors a 10 uF capacitor were chosen which yield

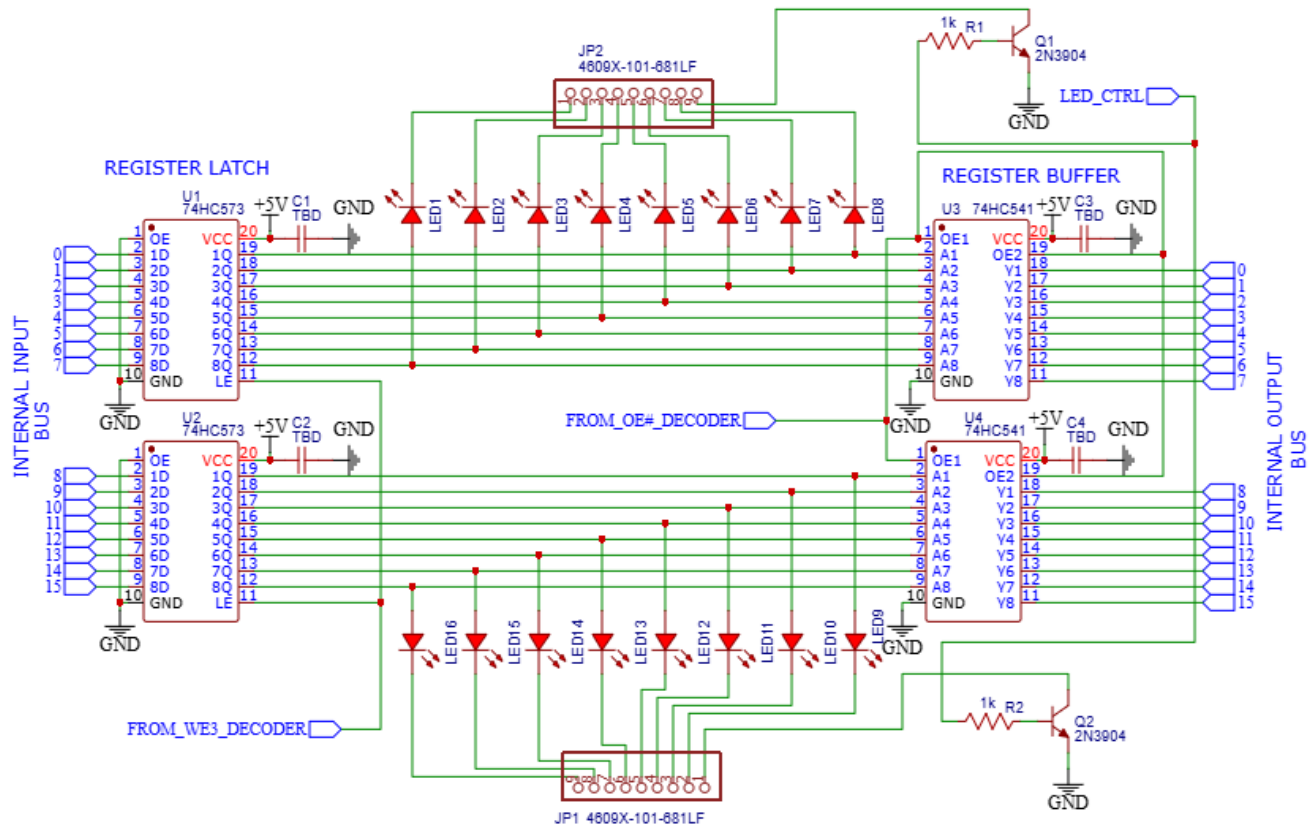


Fig. 2. Simplified Schematic of a Single Register in the Register File

an oscillation frequency of 7.46 Hz. The schematic for this oscillator can be seen below in Figure 7.

The oscillator sections for the two manually increment models were a little more complicated to design as they required their own finite state machine. The sub-FSM and sub-sub-FSM counter needed to be used in this FSM. They will be described in more detail later. The sub-sub-FSM counter is a two-bit counter used to time the operations that happen in a sub instruction. The sub-FSM counter is a four-bit counter that is directly driven by the sub-sub-FSM counter. It is used to keep track of the current sub instruction. In order to increment through a sub-instruction, the FSM needs to start oscillating when a button is pressed and stop oscillating when the sub-sub-counter is 0b00. In order to increment though a full instruction, the FSM needs to start oscillating when the button is pressed and stop when both the sub-counter and sub-sub counter are 0b0000 and 0b00 respectively. Instead of designing two FSMs for each of these options, a single one was designed with an input that is high when either the sub-sub counter is 0b00 and increment one sub instruction is selected on the rotary switch or both the sub-sub counter and the sub counter are 0b00 and 0b0000 respectively. The logic for this part can be seen below in Figure 8. "POS0" is the position on the rotary encoder that corresponds to manually increment one

sub instruction, it is active low.

The finite state machine was then developed using the one zero signal. The finite state machine is a mealy machine because it uses both current and stored values to determine the next output state. It has one output. The output stays low if the button is not pressed. Once the button is pressed, the output will toggle until the zero input is high, signaling that it has either reached the end of the instruction or sub instruction. An artifact of this FSM is that if the button is held down, the computer will increase forever until the button is released, it will then finish the instruction that it is on. The frequency that this oscillation occurs at is 4 Hz. This was chosen so there is enough to see everything that is happening as this is mostly a feature for debugging. Another artifact is that the button needs to be held down for at most a 4th of a second so the FSM can see it and begin its cycle. The last artifact is that the finite state machine will trigger if the sub counters are not zero even if it is not on the manual setting. For instance, if the computer is run at a different speed, paused, and switched to manual, it will automatically start incrementing if the sub counter were not at zero at the time the computer is paused.

To make the finite state machine, a 74HC74 Dual D-Latch was used to store the state and logic was used to realize the transfer table. A 4 Hz oscillator was designed using an NE555

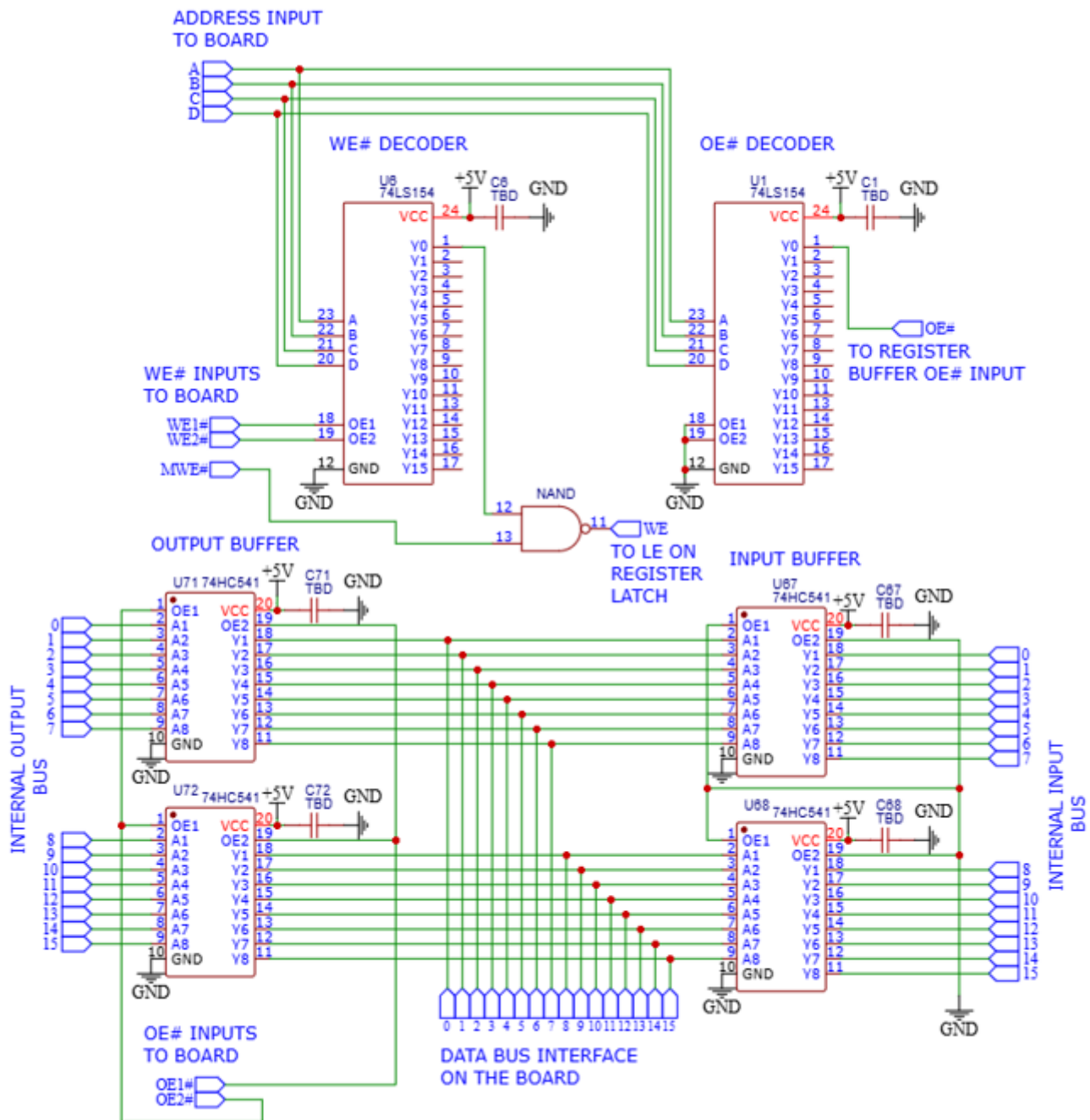


Fig. 3. Simplified Register Control Circuitry Schematic

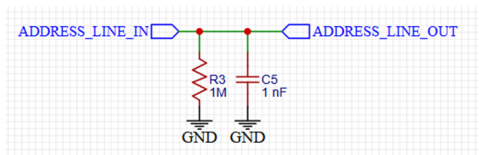
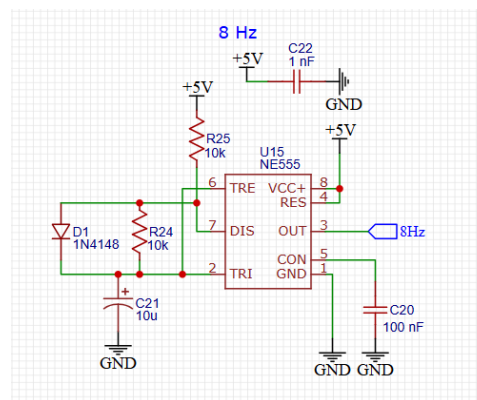
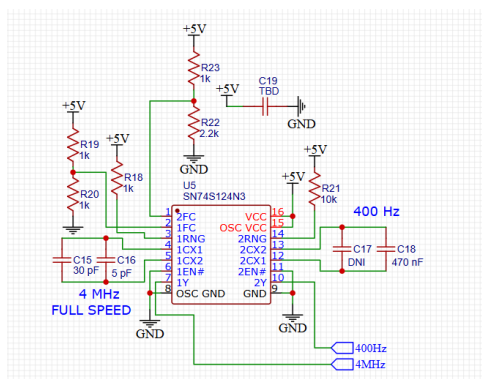
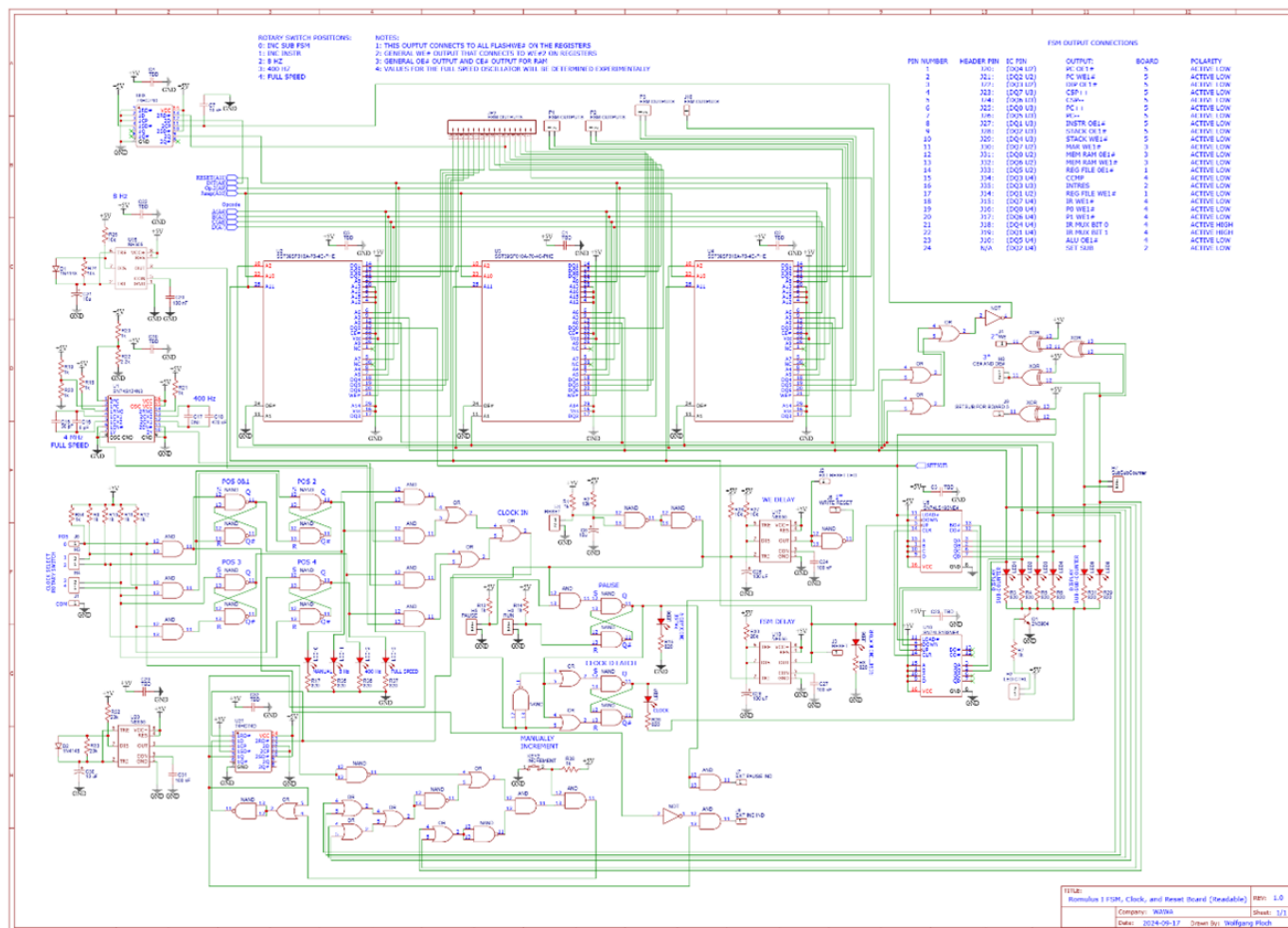


Fig. 4. RC Network Used to Filter Address Lines

and the same equation used to design the 8 Hz oscillator. This circuit can be seen in Figure 9. “POS0” and “POS1” are the two positions on the rotary encoder that correspond to the two manual modes. The logic designed using them flashes the LED indicator on the control panel when clocking through manually.

To switch between all of the clock signals, a 5-position rotary switch is used. The common pin is grounded which each



of the position pins are pulled to 5 volts using a 1 $k\Omega$ resistor. To generate the signals used to select the different clocks, a sophisticated SR latch circuit was designed. Physical switches often introduce “bouncing” in a circuit. The metal contact vibrates at the instant it is closing causing an undesired, short series of pulses upon a transition. To prevent this, the circuit

was designed to latch the positions the instant that they are reached. The basic principle is that the desired position on the rotary switch sets its corresponding latch, this latch is then reset by the positions on either side of it. For instance, the latch for position 2 is set by position 2 and reset by either position

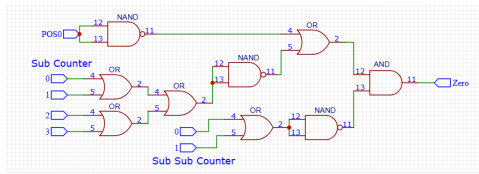


Fig. 8. Zero Determination Logic Schematic

1 or 3. Positions on the edges require one reset (When at position 0, it can only possibly turn to position 1 from there). The positions 0 and 1 both correspond to the manual increment modes. Since both connect the same clock source to the clock input, they are combined and treated as one position as far as this debouncing circuit is concerned. The SR latches used in this circuit and the rest of the board are made from 2 NAND gates in an arrangement shown in Figure 10. The main output (Q) is active high (Q#) is active low and the reset and set inputs are active low as well.

These were used to make the rotary switch selection circuit. The outputs of the position latches then feed into AND gates. The other input of the AND gates are connected to the corresponding clock. These are all ORED together to create a single clock output. This portion of the circuit can be seen below in Figure 11.

3) *Pause Handling:* This “main clock” signal created in the circuit above is connected to the input of a D latch. This D latch is used to pause and un-pause the computer by latching the clock when the pause button is pressed. The pause and un-pause buttons are located on the control panel. There is an indicator for pause on the control panel and the circuit board. To make it easy to keep the computer in its reset state upon startup, reset automatically pauses the computer. Pressing pause sets the pause SR latch while pressing un-pause, resets the latch. These SR latches are again designed using the NAND gate SR latch in Figure 6. The Q of the Pause SR latch is then connected to the LE input of the D latch. The “Conditioned Clock” is the final clock signal that will be used to drive the counter ICs. This portion of the circuit can be seen below in Figure 12.

4) *Reset Timing:* The computer resets itself when it is turned on and when the user presses the reset button on the control panel. To create this reset signal when the computer is powered on, an RC network is used to create an active low signal. The capacitor voltage starts at 0 volts and charges through the resistor until it reaches the high input level of the logic gate. With a resistor value of $10\text{ k}\Omega$ and 10 uF , this reset signal takes 120 milliseconds to reach the high threshold level of the logic gate family used. This pulse is then ANDed with an active low button press to create a combined signal. This active low reset signal “RESET#” then connects to a few latches on this board. It can be seen in Figures 5 and 8. This “RESET#” is used to trigger two NE555 timers in monostable mode. The first one is the “WE DELAY” which creates a one second pulse and the second is the “FSM DELAY” which creates a two second pulse. The reason these two timing

signals are needed comes down to the latches used in all of the registers. They level triggered, not edge triggered. This means that the data that is latched into them needs to be held at the inputs after the latch enable input is deactivated. The data bus is pulled to ground using $4.7\text{ k}\Omega$ resistors on board 4. The FSM DELAY timer is an input to the FSM flashes. When this input is high, the all of the output and write enables of all of the registers are deactivated, meaning that nothing is on the bus. The resistors pull the bus to ground. The output of WE DELAY is inverted, then wired off the board to the master writes on all registers in the computer. The delay for the FSM is longer than the delay used to latch the registers to ensure they all latch 0x0000 properly during a reset. The reset and timing circuitry can be seen in Figure 13.

5) *Timing Signals and Counters:* The sub counter and the sub-sub counter described in previous sections are generated using two 4-bit binary counter ICs, the 74LS193. This IC has a borrow and carry outputs that makes it easy to string them together and create counters higher than 4 bits. The carry output is connected to increment of the next chip and the borrow output is connected to its decrement. This is how the two 74LS193s were connected in the circuit. When the increment or decrement inputs on the chip are being used, the other one needs to be pulled to Vcc. Because this counter is only incrementing, the decrement input is permanently tied to 5 volts. The counter has an active high clear, both are connected to the FSM DELAY. The counter IC also offers 4 inputs for each bit. There is an active low load input as well. When this load input is pulled low, the data on these inputs is latched to the outputs. All of the data inputs on both chips are pulled high to 5 volts. The load input is connected to one of the FSM outputs, called SETSUB. The lowest order 2 bits of this 8 bit counter are the sub-sub counter, the next 4 bits are the sub-counter. The highest order 2 bits are unused. Since not all instructions use the same amount of sub instructions, there is a sub instruction called SETSUB. As a side note, this output is inverted and wired to an output on the board, an active high version is needed by board 5 to deal with hardware interrupts. When it is activated, it loads 0b111111 to the combined counter. This jumps the counter to the end, causing the next clock pulse to take it to 0b000000, and therefore, the next instruction. This is a way of terminating the instruction after all of the sub instructions are done instead of having to wait for the counter to count past the unused sub-instruction slots.

Each of the 6 bits are connected to LED indicators on the board. These indicators (along with the main clock indicator) have their cathodes tied to the collector of a 2N3904 transistor. The base of this transistor is then controlled by the main LED control switch on the control panel. This connects to all of the other cathode transistors controlling all of the other LEDs on the other boards.

Special timing routines need to be used when writing and reading information to the selected SRAM chips, the AS6C1008s. Their wiring is explored in more detail when discussing their circuits, but the timing will be described

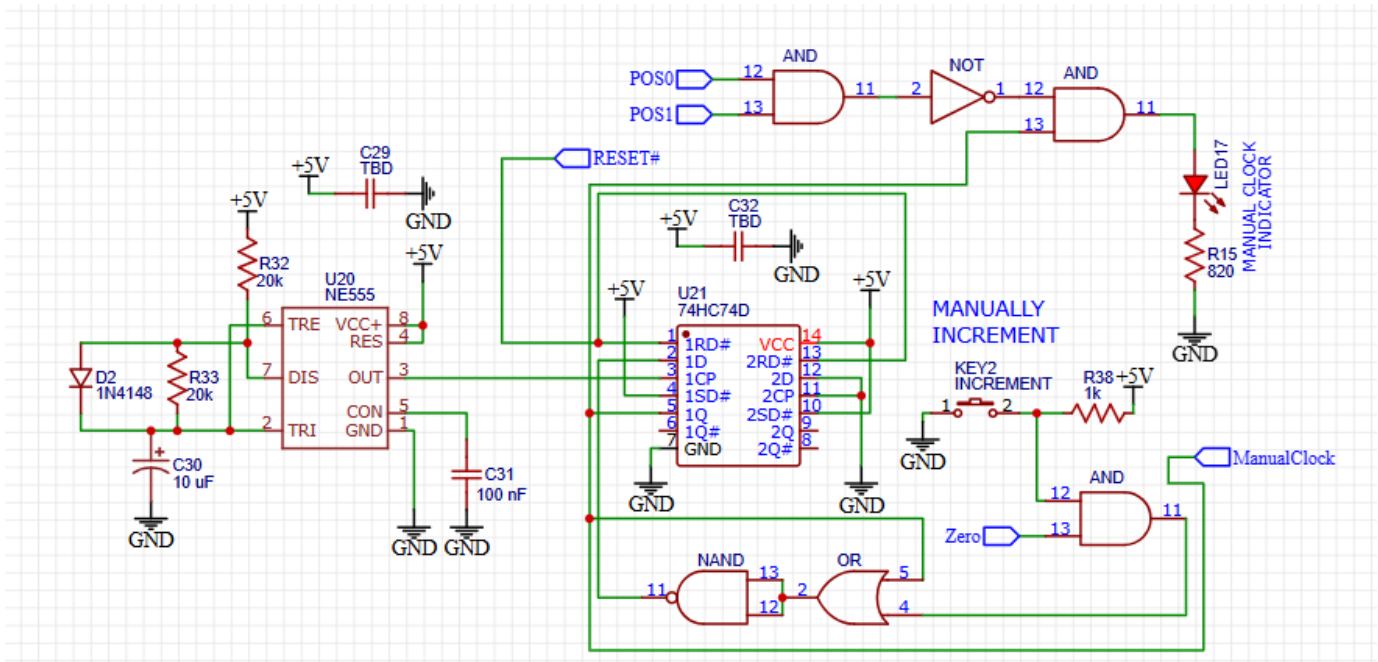


Fig. 9. Manual Increment Finite State Machine Circuit

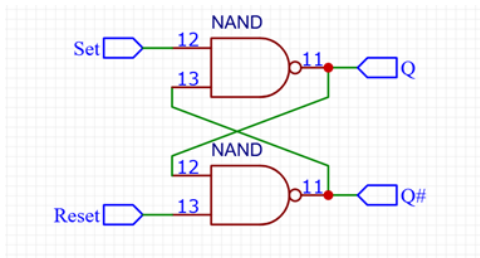


Fig. 10. SR Latch Circuit Using NAND Gates

here. The AS6C1008 does not have separate input and output signals, instead, IO pins that switch. The chip requires the WE# pin be pulled low and a period of time elapse before the CE# input be pulled low and data be present at the pins. Since the IO pins of the chip are connected directly to the data bus, this means the data cannot be present on the bus until a time after the WE# input is pulled low. The period of time is much shorter than the maximum clock frequency of the computer, but it still needs to be accounted for. This timing issue is the reason behind the two-bit sub-sub clock. This creates four different slots that be used to create this timing sequence. The timing sequence can be seen below in Figure 14.

WE is write enable, OE is output enable, and CE is chip enable. This timing sequence works for the write and output enable inputs of the registers as well. The reason that they are labeled with a 2 after the name is due to the way the FSM selected the register of SRAM to use. Every device has two WE# inputs and two OE# inputs. Both inputs must be low for the desired operation to take place. The first ones (OE1# and WE1#) are connected to signals from the FSM flash chips.

This is way the FSM selects the devices needed for the sub-instruction. The second set of inputs (OE2# and WE2#) are controlled by the timing sequence seen in Figure 10. This allows the timing sequence to control the devices selected by the FSM. Each of the SRAM chips has two extra inputs (CE1# and CE2#). CE1# is created on the board by ANDing its OE1# and WE1# . This reduces the number of outputs needed from the FSM flash chips. This allows the chip to be enabled if it is being written to or read from. Logic was used to generate these timing signals from the two bits of the sub-sub counter. The counter circuit and timing signal generating circuit can be seen below in Figure 15.

6) *Interrupt Handling Correction:* The interrupt signal generated on board 5 was initially a direct input to one of the addresses of the flash chips. This caused errors when an interrupt was triggered half way through an instruction. The solution was to latch the state of the interrupt input at the beginning of an instruction and use a control signal from the FSM to reset it at the end of the interrupt path. To accomplish this, a 74HC74 D latch was used. Its data bit is connected to the incoming interrupt signal, its Q pin is connected to the address bit of the flash. The clock pin is connected to logic that triggers it when the sub-counter is 0b0000. The reset pin is connected to INTRES, an active low signal created during the troubleshooting process that is used to control timing associated with the interrupt handling. This circuit can be seen below in Figure 16.

7) *Finite State Machine Flash Chip Lookup Table:* The finite state machines outputs are data locations store across three flash chips. These chips are the SST39SF010As. They have 17 address inputs (A0-A16), and 8 outputs (DQ0-DQ7).

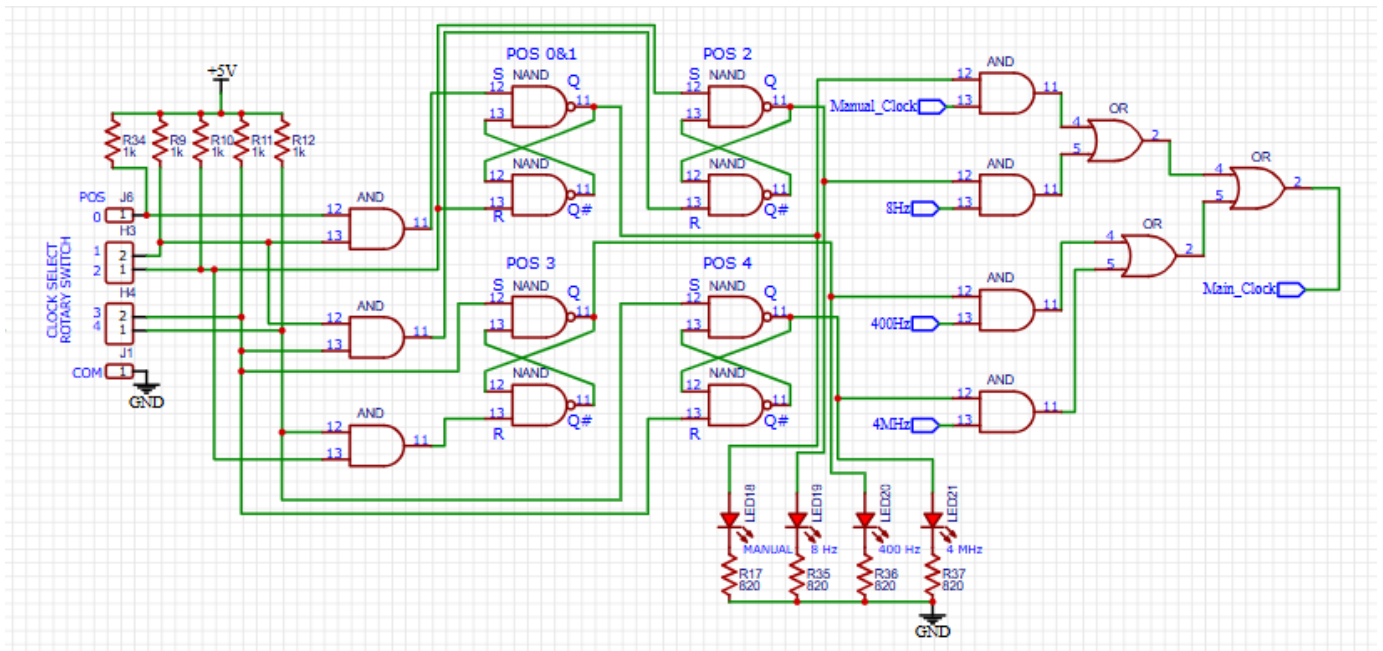


Fig. 11. Rotary Switch De-bounce and Clock Selection Circuitry

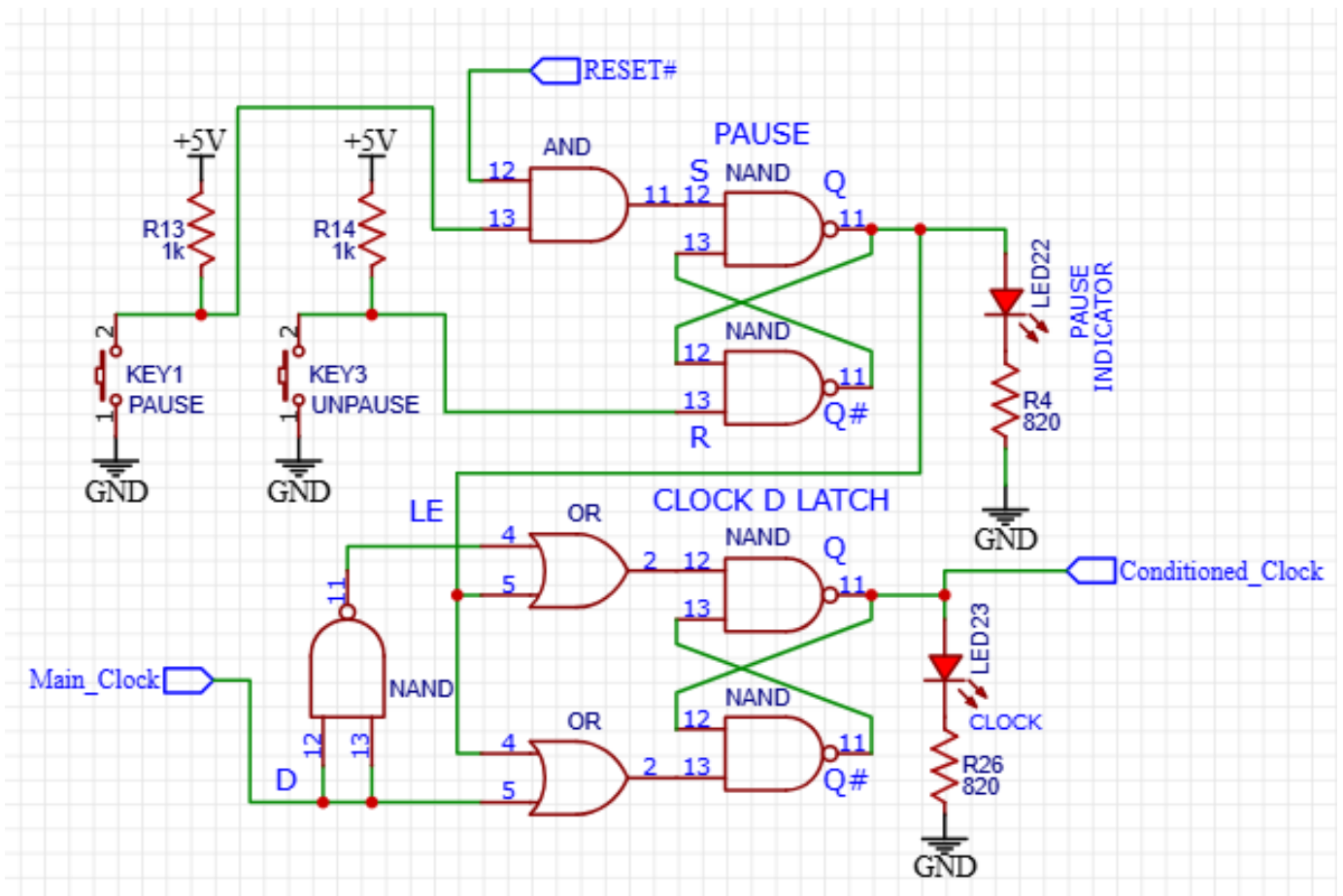


Fig. 12. Clock Pause Handling Circuit

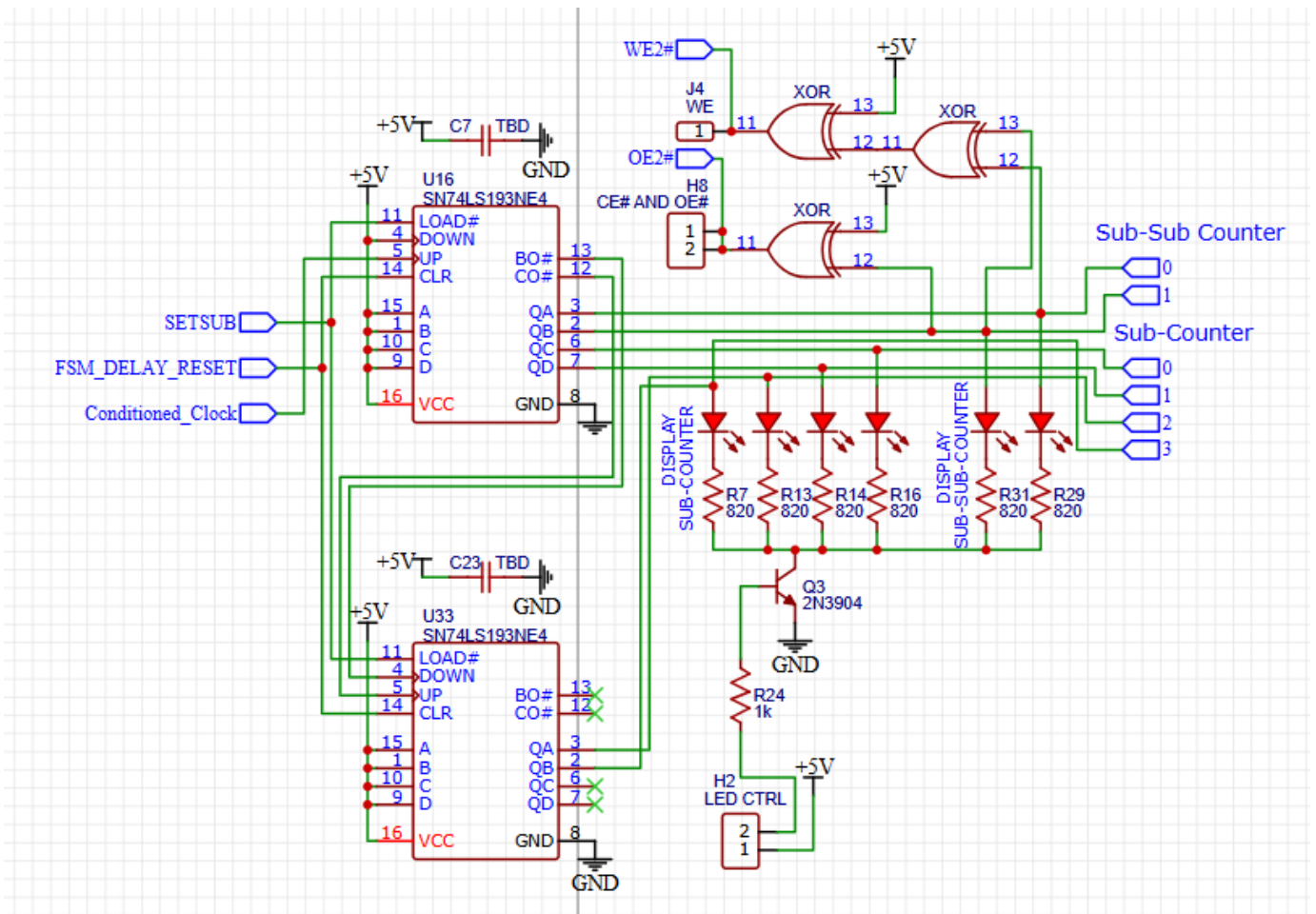


Fig. 15. Sub Counter, Sub-Sub Counter, and Control Signal Timing Circuit

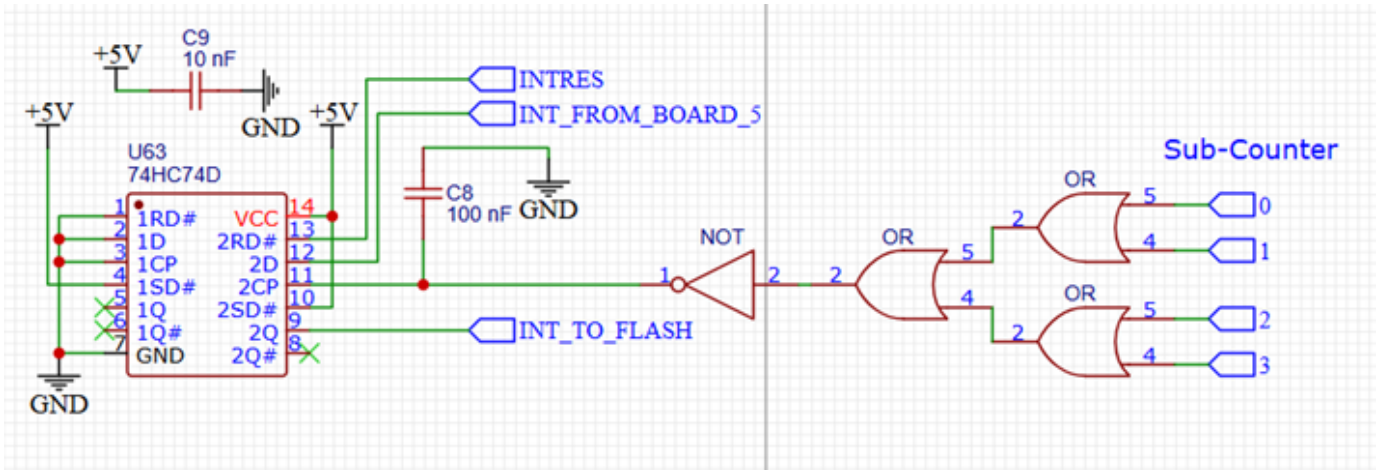


Fig. 16. Interrupt Latching Circuit

C. Board 3

1) *Overview:* The memory and mapped IO board contains the memory SRAM, the mapped IO slots, and the memory address register (MAR). It is referred to as board 3 in other

design sections. It uses the same SRAM chips as board 5 (the AS6C1008s). The IO ports have 16 inputs and 16 outputs each. They are addressed as the last four memory addresses; 0xFFFFC, 0xFFFFD, 0xFFFFE, and 0xFFFFF. The full schematic of this board can be seen in Figure 17.

TABLE III
FSM FLASH CHIP OUTPUTS

Signal	Polarity	Destination Board	Flash IC	Pin
PC OE1#	Active Low	5	U2	DQ4
PC WE1#	Active Low	5	U2	DQ2
DIP OE1#	Active Low	5	U2	DQ3
CSP++	Active Low	5	U3	DQ7
CSP-	Active Low	5	U3	DQ6
PC++	Active Low	5	U3	DQ0
PC-	Active Low	5	U3	DQ5
INSTR OE1#	Active Low	5	U3	DQ1
STACK OE1#	Active Low	5	U3	DQ2
STACK WE1#	Active Low	5	U3	DQ4
MAR WE1#	Active Low	3	U2	DQ7
MEM RAM OE1#	Active Low	3	U2	DQ0
MEM RAM WE1#	Active Low	3	U2	DQ6
REG FILE OE1#	Active Low	1	U2	DQ5
CCMP	Active Low	4	U4	DQ3
INTRES	Active Low	2,4	U3	DQ3
REG File WE1#	Active Low	1	U2	DQ1
IR WE1#	Active Low	4	U4	DQ7
P0 WE1#	Active Low	4	U4	DQ0
P1 WE1#	Active Low	4	U4	DQ6
IR MUX BIT 0	Active High	4	U4	DQ4
IR MUX BIT 1	Active High	4	U4	DQ1
ALU OE1#	Active Low	4	U4	DQ5
SETSUB	Active Low	2	U4	DQ2

2) *Memory Address Register (MAR)*: The MAR stores the memory address used by the RAM to find the correct value. It is identical in operation to the registers on board 1. Its inputs connect directly to the bus and its outputs connect directly to the address inputs of the SRAM chips. Since it is always outputting to the memory RAM, it does not have OE1# and OE2# signals. It's circuitry can be seen in Figure 18. In addition to what is seen in Figure 2, it has a MWE# which writes the register regardless of WE1# or WE2# .

3) *Address Decoding*: The circuit needs to be able to be able to detect if the addresses being written to or read from are in the SRAM or in the memory mapped IO slots (last 4 addresses). This is accomplished using combinational logic. Since the last 4 addresses correspond to the lowest order 14 addresses being 1, (0b111111111111xx), if they are all 1 then the address has to be in the memory mapped IO section. Two signals are created, SELECT RAM and SELECT MAPPED IO. These signals are both active low. The logic used to create these signals can be seen below in Figure 19.

4) *Mapped IO Port*: The four mapped IO ports are identical. They resemble the other registers used in the computer, the difference being they have an IO port instead of LEDs. The output of the two 74HC573s connect directly to the IO port and function as outputs. The inputs of the two 74HC541s are pulled to ground through 4.7 k Ω resistor arrays. This ensures that the pins do not float when a device is not plugged into them, These function as the inputs to the computer. "Writing" to this memory address causes the value to appear on the 16 output pins while "reading" from this address stores the value of the 16 input pins. Because of this, information "stored" in this address cannot be read from it. Each of the 2x20 pin connectors is wired to the input and output bits. Each

connector also has a power and ground pin as well as an interrupt pin. This pin is pulled to ground through a 1 k Ω resistor. The four interrupts from the four IO slots are ORed together and this creates the INT IN signal used by board 5. The schematic of the 0xFFFF IO slot can be seen below in Figure 20, the other three are identical.

5) *Mapped IO Port Control Signal Logic*: Logic is required to control the different write enables and read enables of the different mapped IO slots. Coming into the board, there are five signals which match the convent seen elsewhere in the computer: MEM WE1# , MEM WE2# , MEM OE1# , MEM OE2# , and FSM CE# . When the MAR holds an address value that belongs to one of the IO slots, logic further decodes that and the input signals to determine which mapped IO port is being selected and whether to write to it or read from it. This portion of the circuit can be seen in Figure 21. The MWE# input is used to write all of the IO ports at once during a reset.

6) *Memory SRAM and Supporting Circuitry*: The SRAM chips chosen are the AS6C1008s. They are 128k chips with 17 address pins. Two are used as each only stores one byte (8 bits). Because the computer is only capable of addressing 16 bits, the highest order bit on each chip (A16) is grounded. The IO pins of the chips are connected to the corresponding bits on the data bus. The address bits of the two chips are connected to each other (i.e. chip 1 A0 connected to chip 2 A0 etc.). These pairs are then attached directly to the MAR. The same timing procedure with WE, OE, and CE described in the description of board 2 is used here. On top of the regular timing logic, some needed to be designed to block the signals from the SRAM when the memory mapped IO ports are being used. The SRAM chips and their supporting circuitry can be seen in Figure 22.

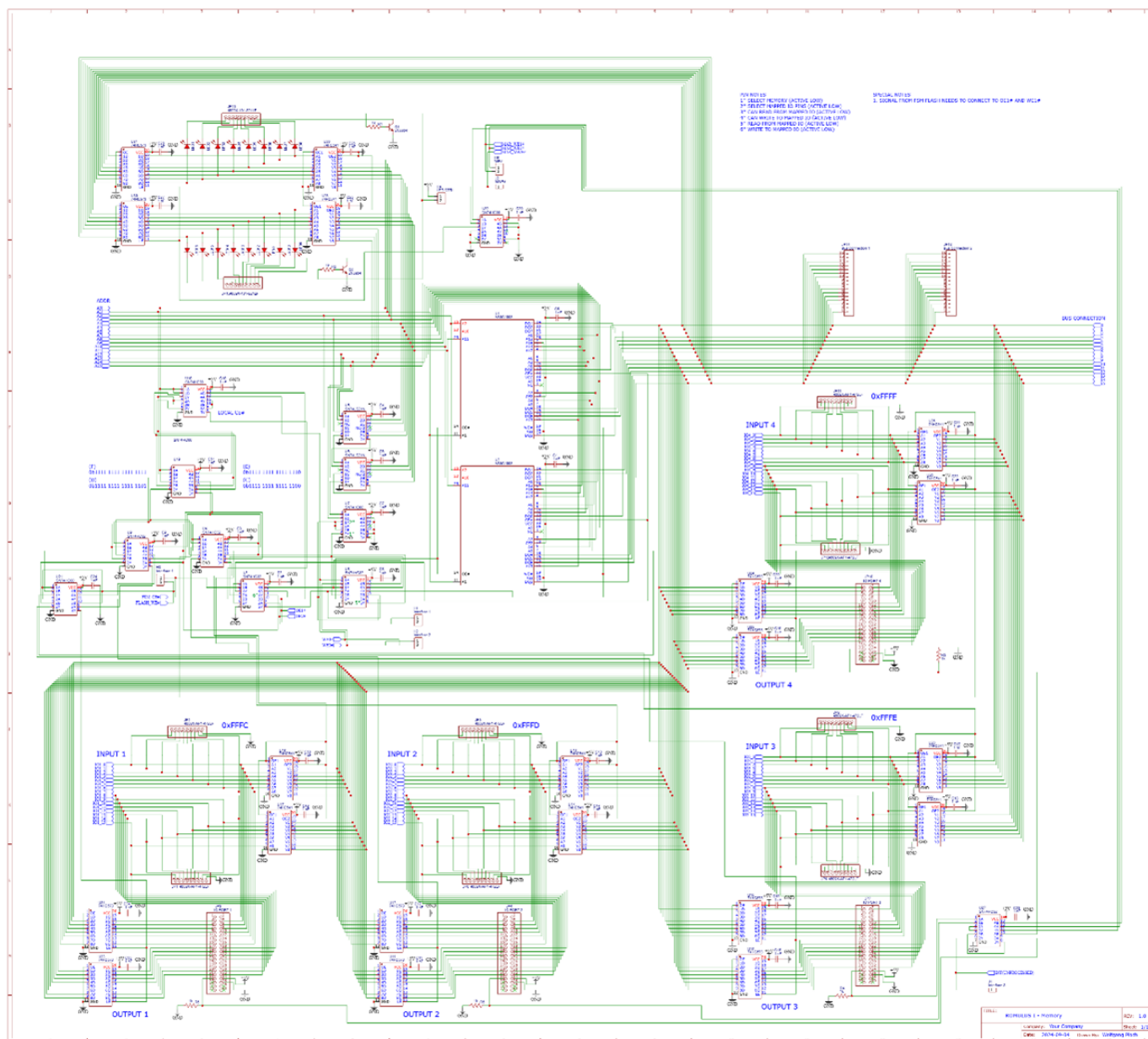


Fig. 17. Full Schematic of the Memory and Mapped IO Board

D. Board 4

1) *Arithmetic and Logic Unit (ALU)*: The ALU is the part of the computer that does all of the arithmetic and logic operations (hence the name “arithmetic and logic unit”). It handles bitwise AND, NOT, OR, and XOR, as well as logical NOT, logical shift right, addition, and subtraction. When one or two parameters are loaded into the “Parameter 0” and “Parameter 1” registers at the bottom of the board and the opcode is sent to the ALU, it immediately calculates the correct answer and, when its output is enabled, places it onto the data bus. The schematic for the ALU in Figure 23.

We decided to make the ALU out of flash chips (SST39SF010A) instead of with logic gates because it takes

up less physical space on the printed circuit board. It works like a lookup table, with the address bits as the inputs and the data bits as the outputs. We had to use three chips because we needed enough address (input) bits for two 16-bit parameters and a 4-bit opcode, and since each SST39SF010A chip only has 17 address bits, we needed three chips to have enough. Table IV shows which pins are used for what values on each chip.

2) *Instruction Register (IR)*: The instruction register stores, in bitcode, the current instruction being executed by the computer. Many things are taken from the instruction: the four most significant bits of the instruction (bits 15-12) are the opcode, bit 7 is the sub-opcode used for push and pop,

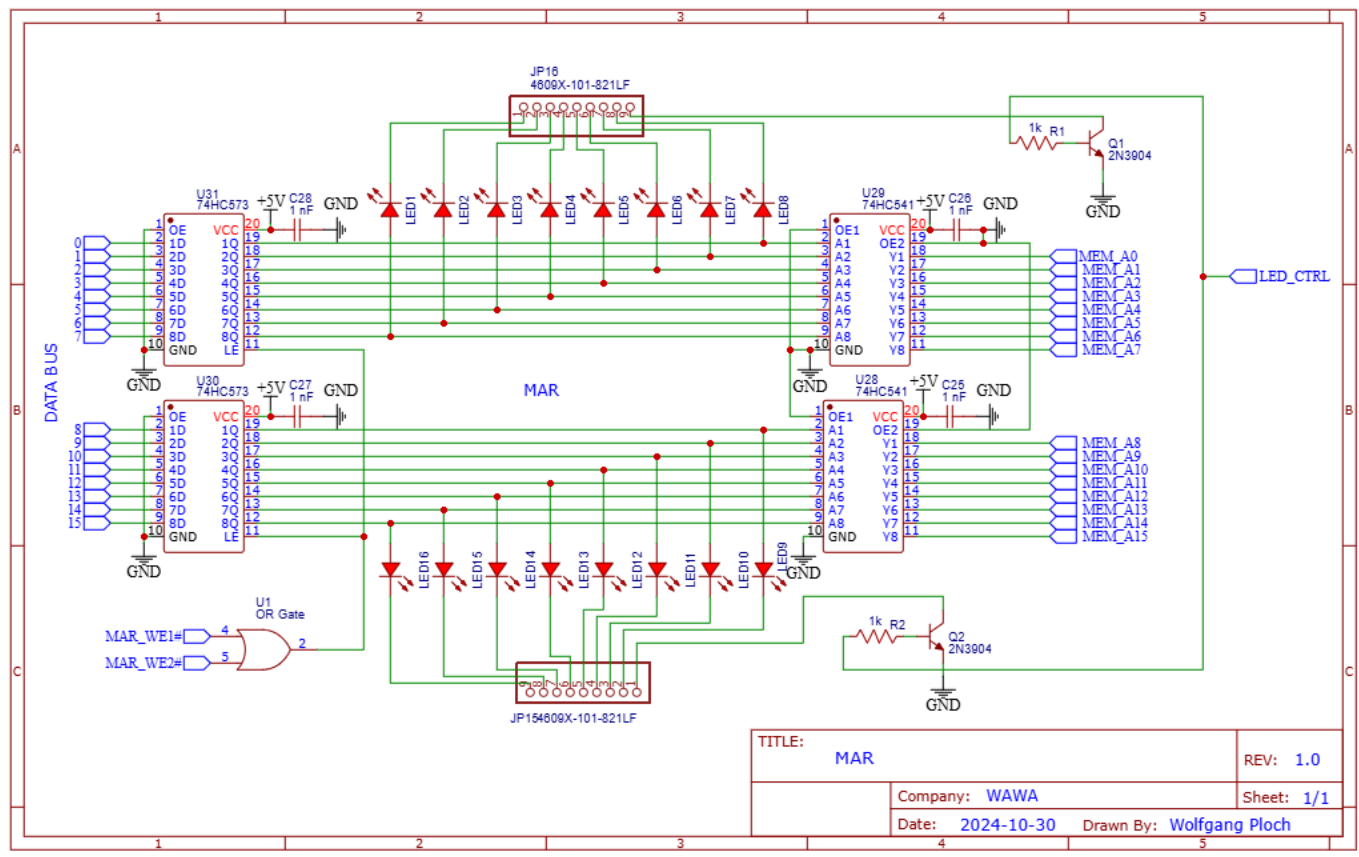


Fig. 18. Memory Address Register (MAR) Circuitry

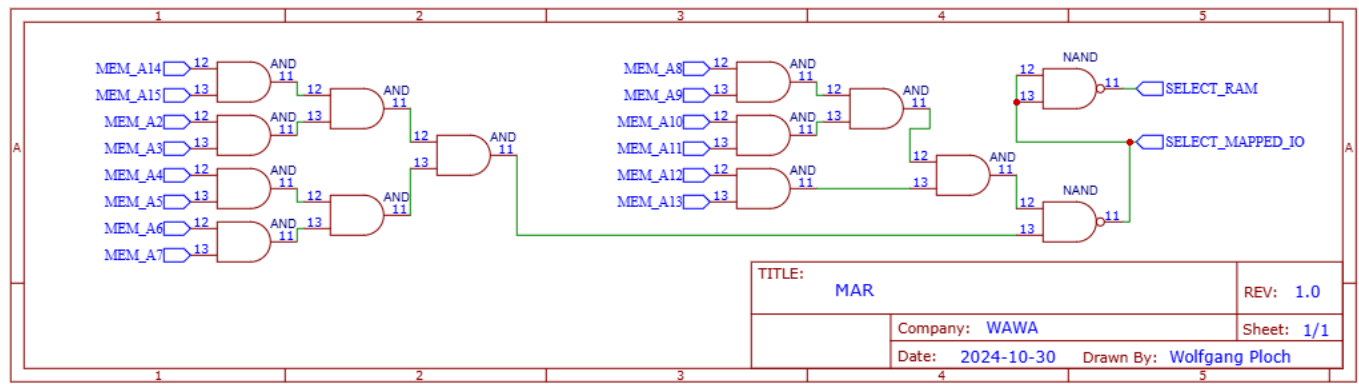


Fig. 19. Logic Used to Create SELECT Outputs

and the four least significant bits (bits 3-0) are used as flags to determine which comparison to make in the jump-compare instruction. Additionally, bits 11-8, 7-4, and 3-0 are used as addresses to the register file to determine which registers to read and write from. Because these signals and values are so important, the IR always outputs its stored value, that is, its output enable is always active. The opcode, sub-opcode, and flags go directly to the control signals board as inputs to the finite state machine. The three groupings of four bits used for

register addressing go through a multiplexer to the register file. The control signals outputs “sel1” and “sel0” are used to determine which portion of the value stored in the IR is used for the register file address: both low sends the first set of four bits, 11-8 (labeled as “Ra” in the instruction set), sel0 high sends the second set, 7-4 (labeled as “Rb”), and sel1 high sends the third set, 3-0 (labeled as “Ry”). Both sel0 and sel1 being high should never occur with our current control signals design, but should this happen erroneously, or should the FSM

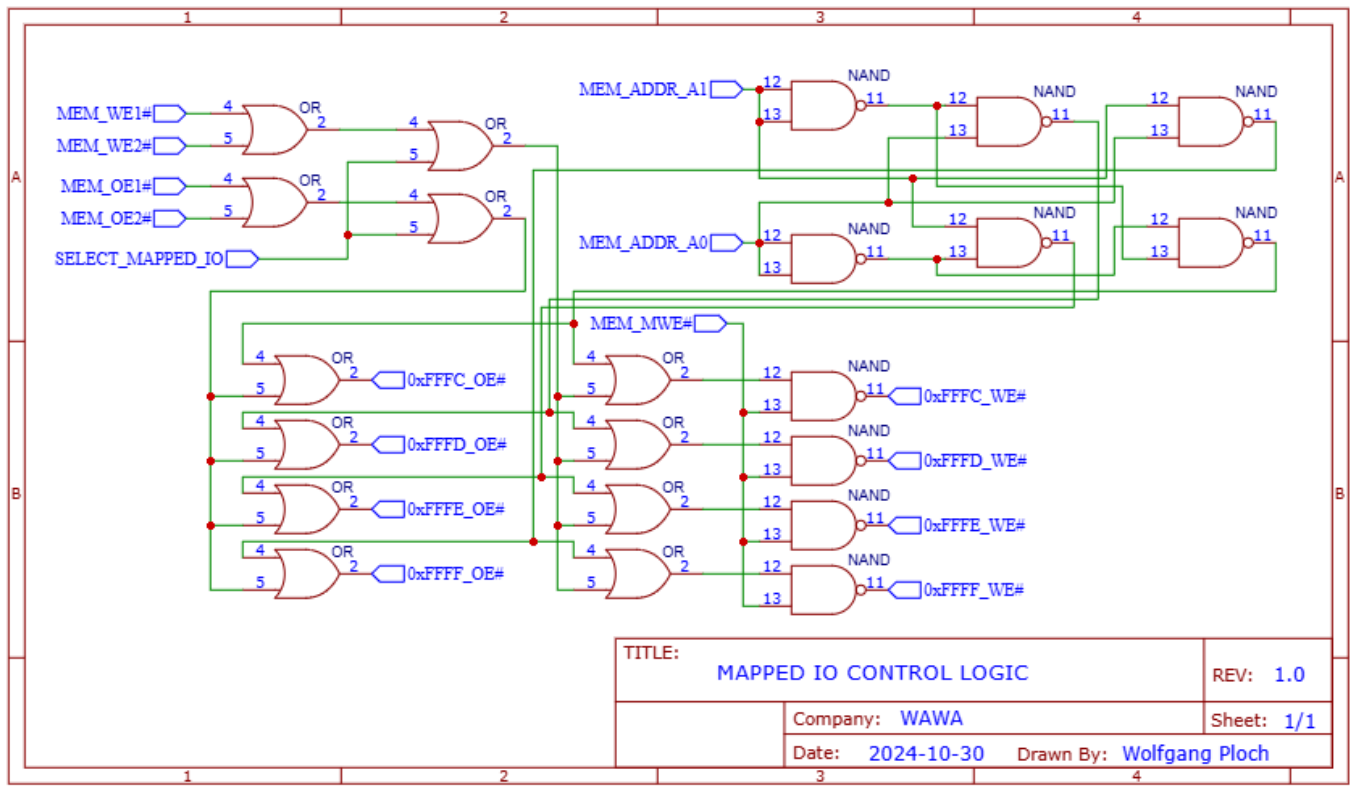


Fig. 21. Mapped IO Port Control Signal Logic

be changed to allow this to occur, the value 0 will be sent as the address. This multiplexer allows the correct register to be read or written to at the proper time.

3) *Compare Logic*: The compare logic uses the value on the bus as well as the flags from the IR to tell the control signals FSM whether or not to jump from a jump-compare instruction. It can do four different comparisons: $\text{bus} == 0$, $\text{bus} != 0$, $\text{bus} \leq 0$, and $\text{bus} < 0$. It computes all four at once, then selects which result (1 for true, 0 for false) to store in the D-latch at the end using the flags. Then the signal from the D-latch goes straight to the FSM. A simplified schematic for the compare logic is in Figure 25.

4) *7-Segment Hex Displays*: The two sets of 7-segment displays on the board show what is in the instruction register and on the data bus. In a similar way to the ALU, they use flash chips (SST39SF020A) as lookup tables, this time to convert the binary value on the data bus or in the IR to the signals corresponding to the correct segments on each display. The circuit also has a high-speed clock, a counter, and a decoder to quickly cycle between the four digits. Only one digit is on at a time, which lets us save on power and flash chips (since we only have to send data to one digit at a time as well, meaning we need fewer output pins), but because the clock cycles between them so quickly, it is impossible for your eyes to see the flickering, and it appears as though all four are on at once. The schematic for the two displays is in figure ??.

E. Board 5

1) *Overview*: This board will be referred to as board 5. It houses the stack and instruction RAM, the program counter (PC), the stack pointer counter (CSP), the dip switches used to set the interrupt handler location, the 40-pin programmer interface, and interrupt handling logic. It receives its control signals from board 2. The entire schematic can be seen in Figure 26.

2) *Program Counter*: The PC is a 16-bit value that needs to be able to be incremented, decremented, and loaded to. It stores the current address of the instruction RAM. The ideal chip to allow for this operation is the 74LS193 counter IC. A more detailed description of its operation can be found in the counter section of the description of board 2. Four of them are connected using the borrow and carry outputs to create a 16-bit counter that can be incremented, decremented and loaded to. The four data inputs on each chip are wired to their respective line on the data bus. The clear inputs of all the 74LS193 are wired to an active high version of the WE DELAY reset signal from board 2. The outputs of all the counters are wired to the inputs of two 74HC541 8-bit buffers. These serve to buffer the outputs of the counter chips so that they do not need to drive the input addresses of the SRAM chips and the LEDs directly. Because the SRAM chips used to store the instructions need the value of PC constantly, these two buffers are constantly enabled. In addition, there are two more 74HC541s whose inputs connect to the buffered PC and

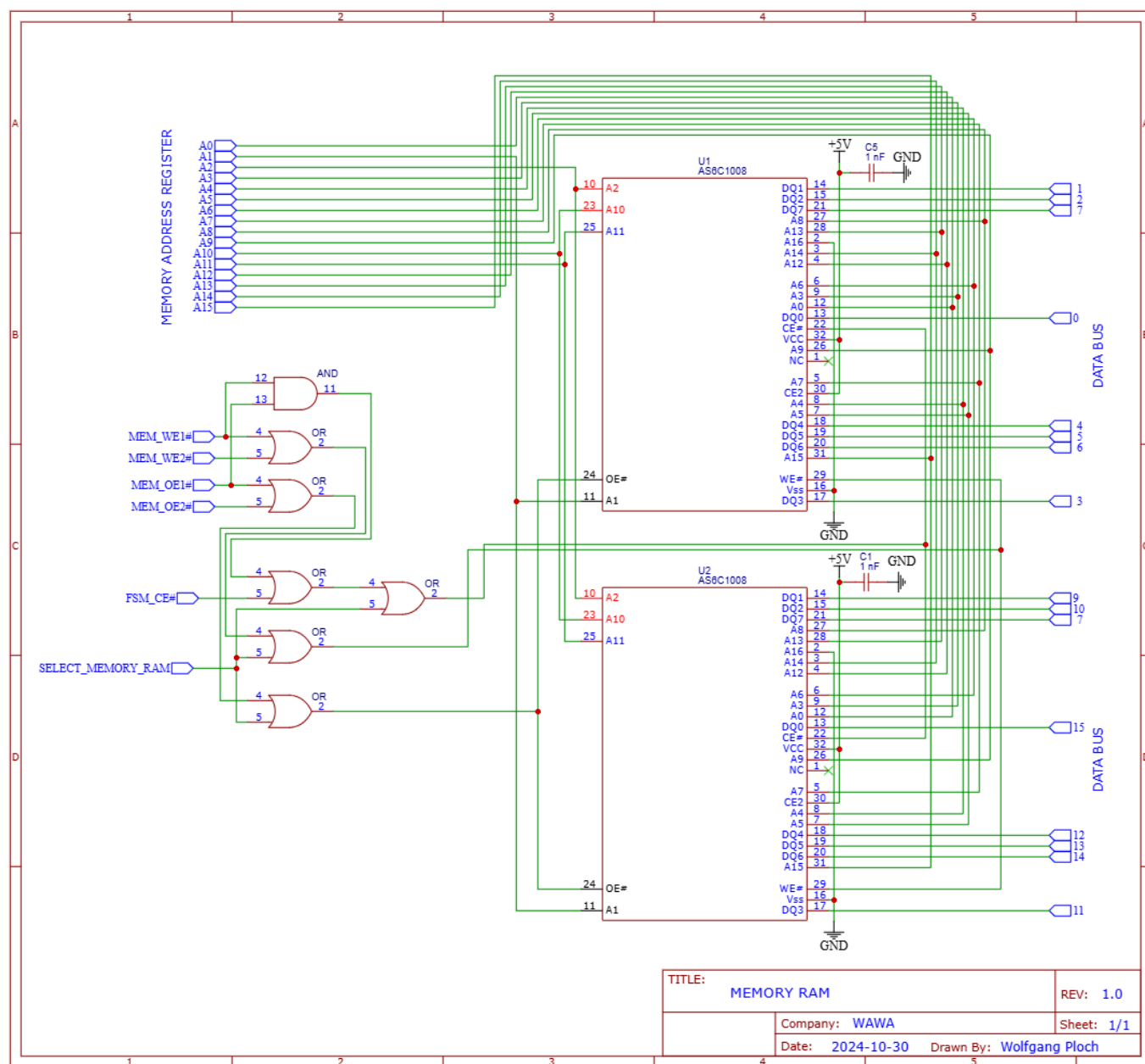


Fig. 22. Memory RAM and Supporting Circuitry

whose outputs connect directly to the data bus. This is to allow the value of PC to be placed on the data bus. When pulling increment low to increment the counter, the decrement input needs to be held high and vice versa. Because of this, the PC++ and PC—control signals are treated as active low WE inputs. The control signals associated with the PC are PC++, PC—, PC OE, and PC WE. These signals conform to the stand outlined in the timing section of the board 2 description. Each of these signals have to inputs (PC OE1# , PC OE2# , etc). The only one that is slightly different is the PC WE2# , it is different from all of the other WE2# as it is only high when the sub-sub counter is 0b10, not 0b01 and 0b10. This is because

the 74LS193s had a weird output state when they were being written to and nothing was on the bus. The PC and supporting circuitry can be seen in Figure 27.

3) *Stack Pointer Counter*: The stack pointer counter is used to store the current address of where the stack is pointing to. Its output connects to the address of the stack SRAM chips and nothing else. There is no need to jump to a specific location inside the stack, so the CSP only needs to be able to decrement and increment. Because of these requirements, it is exactly the same as the PC except it does not have the second set of buffers that connects it to the bus, and it does not have a WE input. The CSP and associated circuitry can be seen in Figure 28.

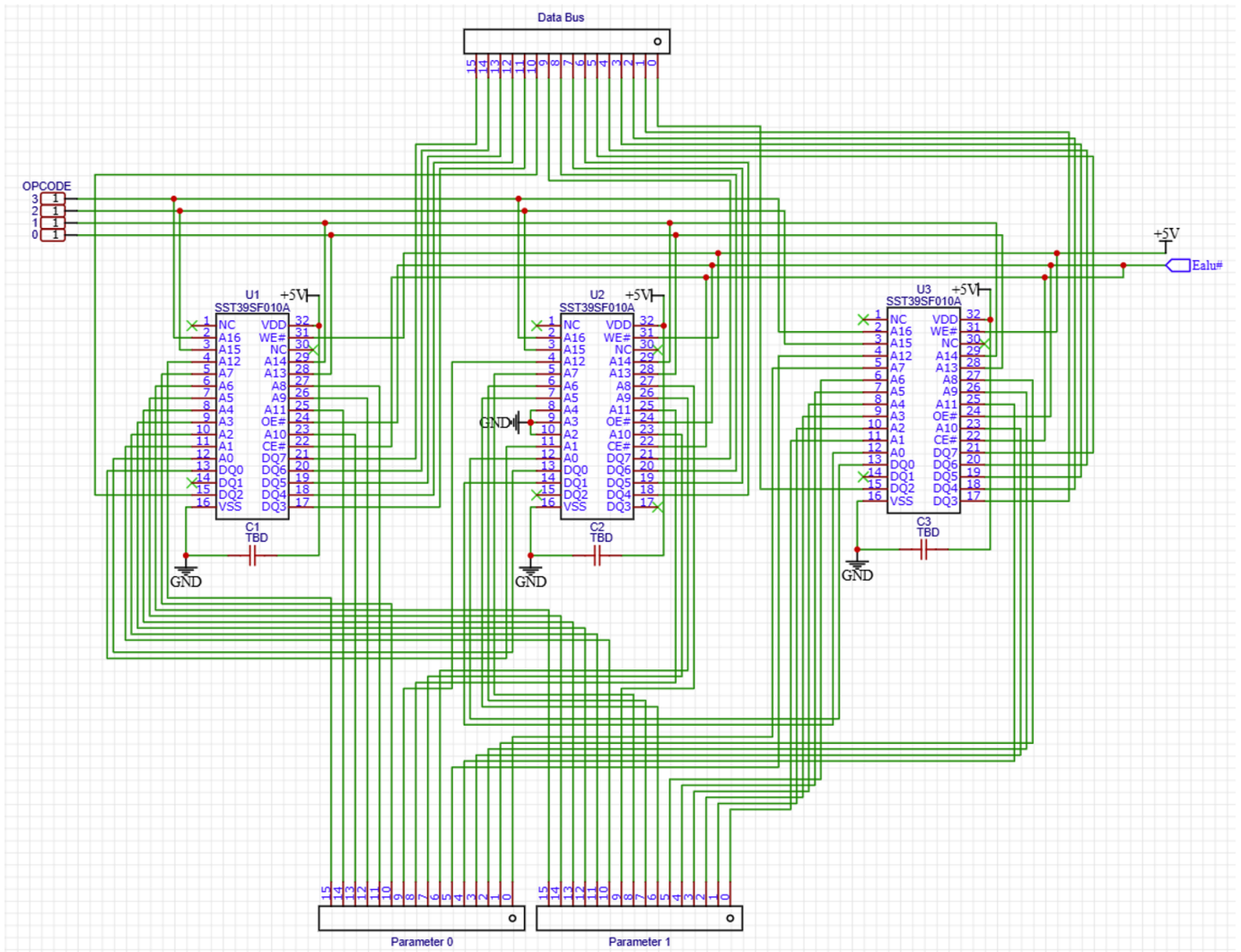


Fig. 23. ALU schematic. A table of connections is provided below, as this schematic is difficult to read.

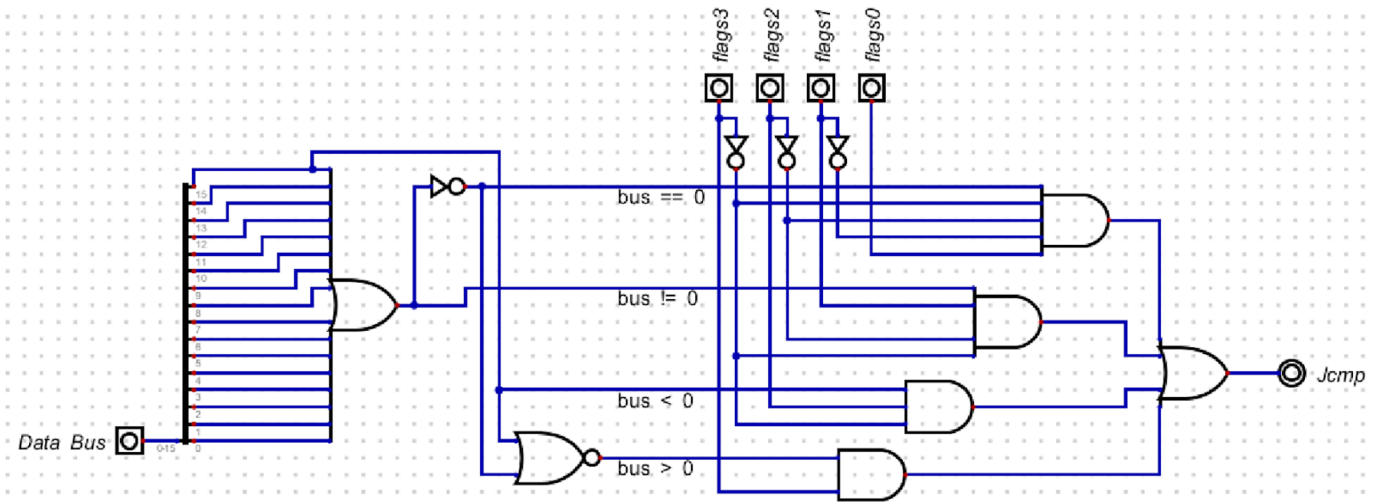


Fig. 24. A simplified schematic for the compare logic. The left side does the comparison, and the right side handles priority: flag3 has priority over flag2, which has priority over 1, which has priority over 0. The D-latch is not pictured.

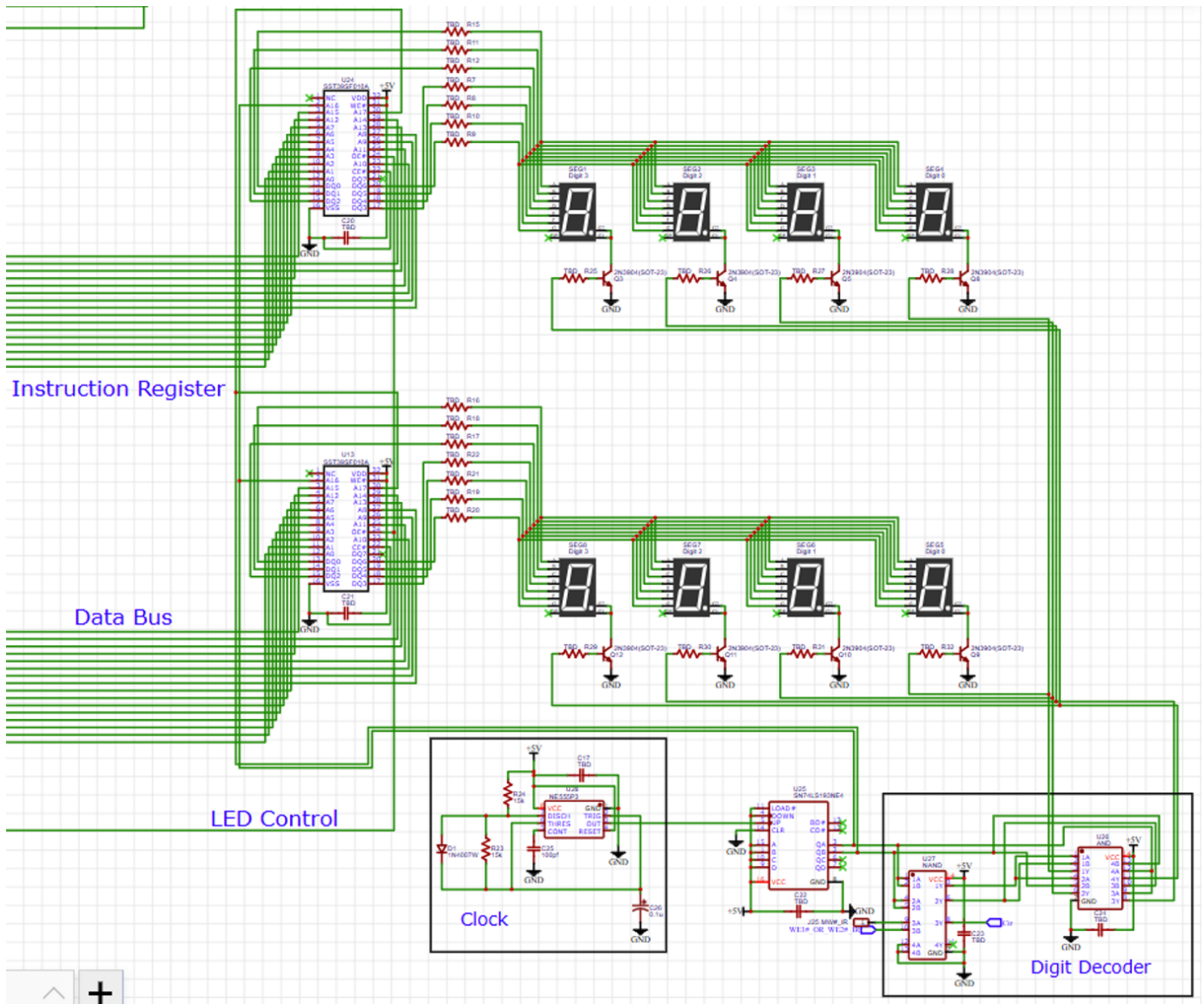


Fig. 25. The schematic for the hex displays. The circuitry at the bottom handles the digit cycling.

4) *Program Counter and Stack Pointer Counter LED Displays:* The 16 bits of the PC and 16 bits of the CSP are displayed by LEDs on the PCB arranged in the same way as the registers on board 1. The LEDs are grouped into 8s, each group connecting to an 8-820 Ω resistor array. The common pin of this array is connected to ground through a 2N3904 NPN transistor to turn on and off the LEDs. All of these are linked to the main LED control switch on the control panel. This portion of the circuit can be seen in Figure 29.

5) *Dip-switches for Interrupt Jump Location:* The computer used the address stored in a set of dip-switches to determine where to jump to when an interrupt occurs. This is accomplished by connecting one side of the dip-switches to 5 volts and pulling the other side of each one down with a 4.7 $k\Omega$ (two 8-resistor arrays were used for 16 dip-switches). The outputs of the switches are buffered by two 74HC541. Their

OE connect to DIP OE1# and DIP OE2# . The outputs of the buffers connect to the bus. This circuit can be seen in Figure 30.

6) *Instruction SRAM:* The SRAM chips used for the instruction and stack RAM in the computer are the AS6C1008s. The timing signals they require are discussed in the section on board 2. The computer technically does not need to have write access to the instruction RAM, the program is written by an external programmer and the computer simply reads it during operation. Write inputs to the instruction were added and the mistake was not realized until the PCB was designed. This was not a fatal error as these inputs were simply pulled high and not used. They still appear in the schematic. Two AS6C1008s had to be used as each only has 8 bits of storage per location. The IO pins on both chips connect directly to the data bus. 16 bits allows for the addressing of 64k memory.

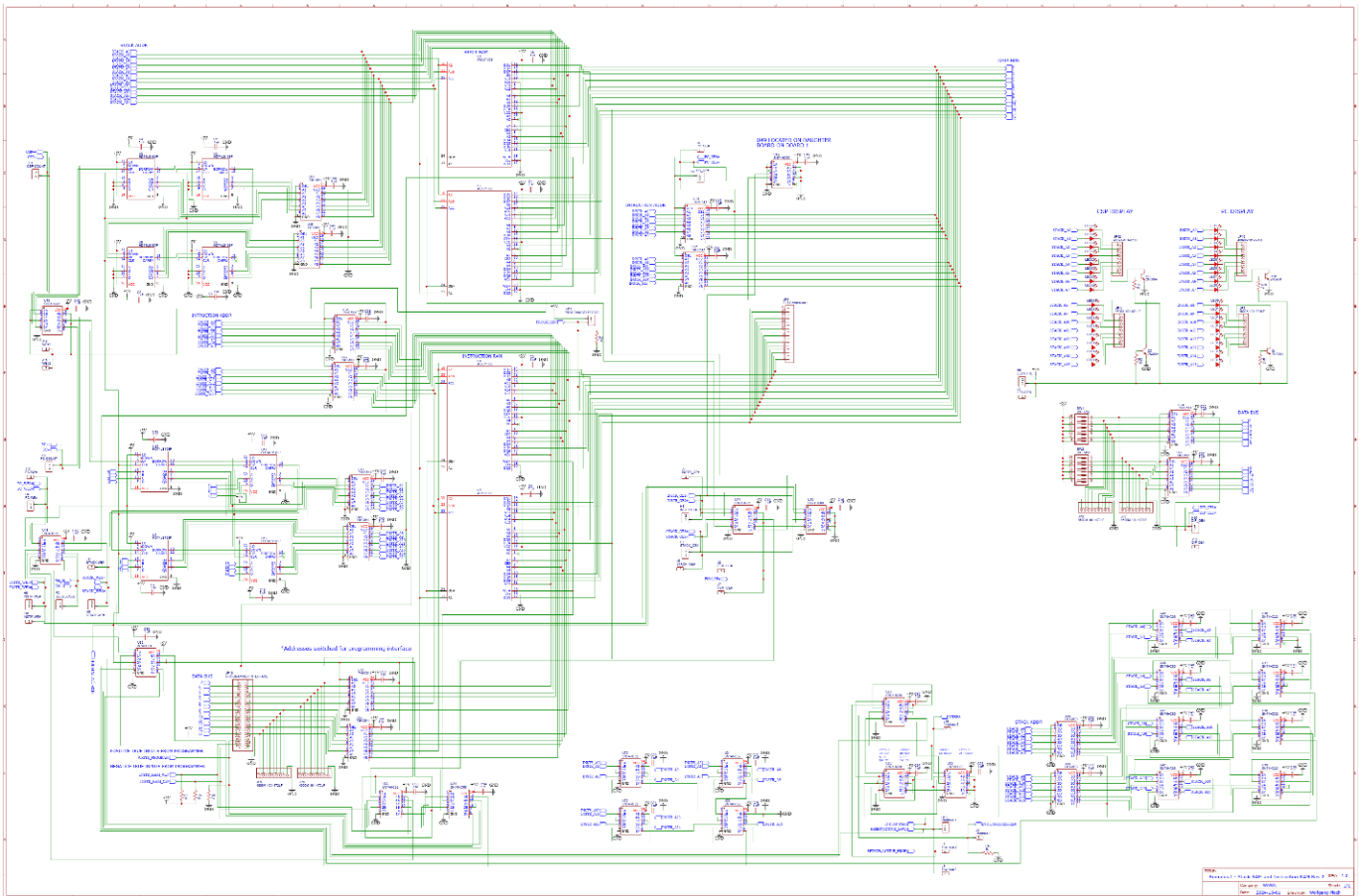


Fig. 26. Full Schematic of the Stack and Instruction RAM Board (Board 5)

Only 32k and 128k versions of the chip were able to be found. The 128k version was selected and MSB of the address for both are connected to a switch on the control panel. This allows the user to store two programs in the computer at once and switch between them with the program switch. The PC does not connect directly to the address pins of the SRAM. It connects through a 1:2, 16-bit multiplexer that switches between the PC and pins of the 2x20 female header on the board. This multiplexer is made using four 74HC541 buffers and some logic gates. There is a bit wired to the connector that changes control of the address pins between the PC and the programmer. The 2x20 female header also has 16 pins connected to the data bus. When the programmer is plugged in and it sets the WRITE PROGRAM bit high, it has control of the address pins of the SRAM as well as its WE and CE inputs. It then sequentially writes instructions by putting the instructions on the bus and incrementing the address bits. There are protections against conflicts where two devices are fighting for control of the bus. The WRITE PROGRAM input will only allow the programmer access if the PC is 0x0000 and the sub-sub-counter is 0b00. This ensures that there is nothing on the bus. The circuitry behind of this can be seen in Figure 31.

7) *Stack SRAM*: The stack SRAM uses two of the AS6C1008 chips in the same arrangement as the instruction SRAM. Its circuitry is less complicated due to the fact that the CSP is the only thing connected to the address pins of the SRAM. The only supporting circuitry the stack SRAM needs are the logic gates to handle the multiple output and write enables and the logic to generate the local chip enable signal. The stack SRAM circuitry can be seen in Figure 32.

8) *Interrupt Handling*: The creation of the initial interrupt signal occurs on board 3. The interrupt inputs from the four IO slots are ORED together, this is the INT IN signal on board 5. The interrupt switch on the control panel disrupts this connection from board 3 to board 5 to enable or disable interrupts. To handle the interrupt correctly, the circuit needs to latch the incoming pulse long enough for board 2 to detect it. It also has to save the current value of CSP and disable any further interrupts until CSP returns to the value saved. Basically, it needs to disable interrupts from occurring until the current interrupt has been dealt with. The program will exit the function is pushed into the stack and it will return to its value before the interrupt was triggered. The SR latch that holds the initial triggering of the interrupt is called the primary interrupt latch. This primary latch will reset when the FSM sees the interrupt. The primary latch triggers a secondary latch

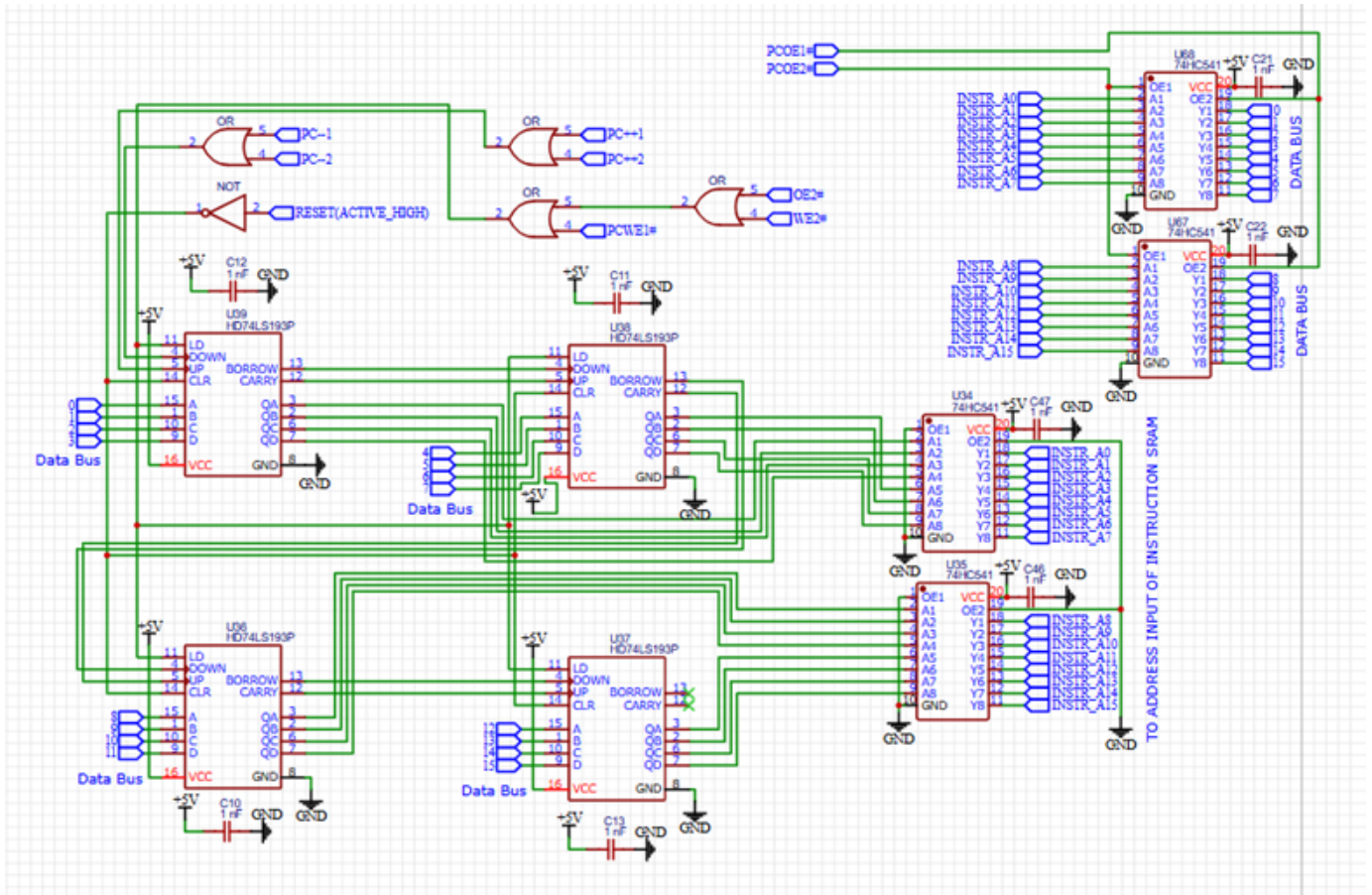


Fig. 27. Program Counter and Associated Circuitry

that stays latched until the computer is done dealing with the interrupt function. This second latch blocks the first latch from triggering again. To determine when the CSP has returned to the starting value, two 74HC573s are used. When there is not interrupt, they pass the CSP from the inputs to the outputs. The secondary latch latches the value of CSP on the inputs when an interrupt is triggered. This stored value is compared to the current value of CSP using logic and when there is a match, this resets the secondary latch. There is a delay in this comparison being able to reset the secondary latch as it would constantly reset itself before the computer jumps to the interrupt location. This is accomplished with logic before the reset input of the SR latch. The primary and secondary SR latches are also reset by the main computer reset signal. The SR latches used in this circuit are composed of NAND gates. The description of board 2 dives into these in more detail. This portion of the circuit can be seen in Figure 33.

The comparison of the stack and the saved stack location is done using XOR and OR gates. The SAME signal is blocked from resetting the secondary latch until the primary latch has been reset and the SETSUB signal is high. Once the SETSUB signal is high and the primary latch has been reset, the computer has already advanced PC to the interrupt function and incremented CSP.

F. Board 6

1) *Overview:* The entire Romulus I computer is powered by 5 volts. The maximum current draw is around 4 amps. A large linear power supply with protection was designed to power the computer. The power supply PCB was designed to be able to handle 8 amps as a precaution. The entire Schematic of the Power Supply Board can be seen in Figure 34. This board will be referred to as board 6 in other sections.

2) *Power Regulation Design:* A 24 VAC transformer with a current capacity of 4 amps was sourced and selected to step down the mains 120 VAC to a level that could be rectified. This transformer is center tapped with two 24 VAC windings. This means that two diodes can be used to rectify instead of a full-bridge rectifier. The diode used were the 6A05 which can handle a current flow of 6 amps. The rectified voltage is then smoothed out with 10000 uF of capacitance. A small 4.7 kΩ bleeder resistor was added to dissipate the capacitors in an absence of load. The rectified, unregulated voltage is referred to as “VCC” in the schematic. A simplified schematic of this section can be seen in Figure 35.

The rectified VCC is then used to create three different supplies, a 5 volt rail used to power the SR latch (SUB 5V+), a 12 volt rail used to power the op amp and comparator (+12V) and the main 5 volt rail used to power the computer (+5V).

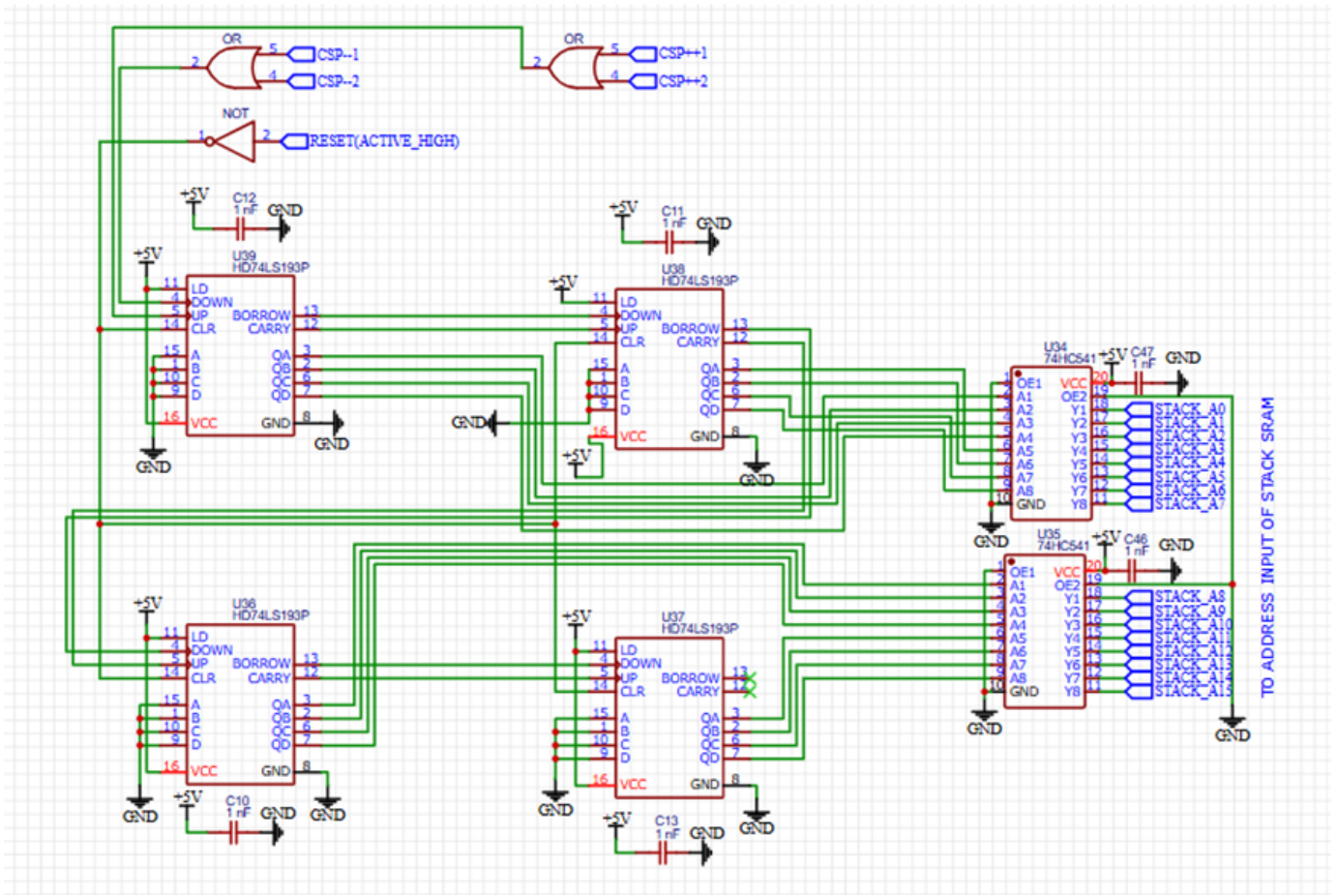


Fig. 28. CSP and Associated Circuitry

To create SUB 5V+, a 7805 linear voltage regulator was used to drop VCC down to 5 volts. This rail is not anticipated to have a large current draw but a small heat sink is attached to the regulator anyway for added protection. A 0.1 μF ceramic capacitor and a 10 μF electrolytic capacitor were connected to the input and output rails respectively. These protect against AC voltage present on the lines. The +12V supply is identical except for the replacement of the 7805 with a 7812. The main +5V supply had to be designed slightly differently because the 7805 can only supply a maximum current of 1 amp. To solve this problem, high-power 5 Ω resistor is placed between VCC and the input pin of the register. A 2N5684 (50 Amp PNP transistor) is then connected as follows: emitter to VCC, base to junction of 5 Ω resistor and voltage regulator and collector to output of voltage regulator. The circuit behaves like this, under low current draw, there is a low voltage drop across the resistor and the regulator provides most of the current. When a large current is drawn, the voltage across the resistor increases, pulling the base voltage of the 2N5684 lower causing it to conduct and supply the needed current to the output. These three supplies can be seen in Figure 36.

3) *Over Current and Over Voltage Protection:* Over voltage and over current protection are implemented to protect the

computer. The over-voltage projection disconnects the +5V rail from the output if the +5V rail exceeds 6.2 volts. To do this, an LM339 compares the +5V rail to a 6.2 reference voltage created with a 6.2 volt Zener diode and a 4.7 $k\Omega$ resistor connection in series on the +12V rail. The comparator pulls the output low if the over-voltage condition is reached. To detect an over-current condition, three 0.1 Ω resistors were connected in parallel and then in series with the +5V rail and the output to the computer. As current is drawn from the supply, a small voltage drop is created across these resistors. This drop is then amplified with a differential amplifier with a gain of 6.8. This signal is then put through an RC low pass filter with a cutoff frequency of 3.39 Hz to remove noise amplified from the supply line. This signal is then compared to a reference voltage set using a 10 $k\Omega$ potentiometer mounted to the board. The output of the LM339 is pulled low if the over-current condition is met. The outputs of the LM339 are open collector meaning that they require a pull up resistor but also that they can be connected together to create an active low OR. This is done with the over-current and over-voltage to produce one active low output that indicates a fault has been reached. This combined output is pulled to +5V with a 4.7 $k\Omega$ resistor. This sets an SR latch built using NAND gates. The latch is reset

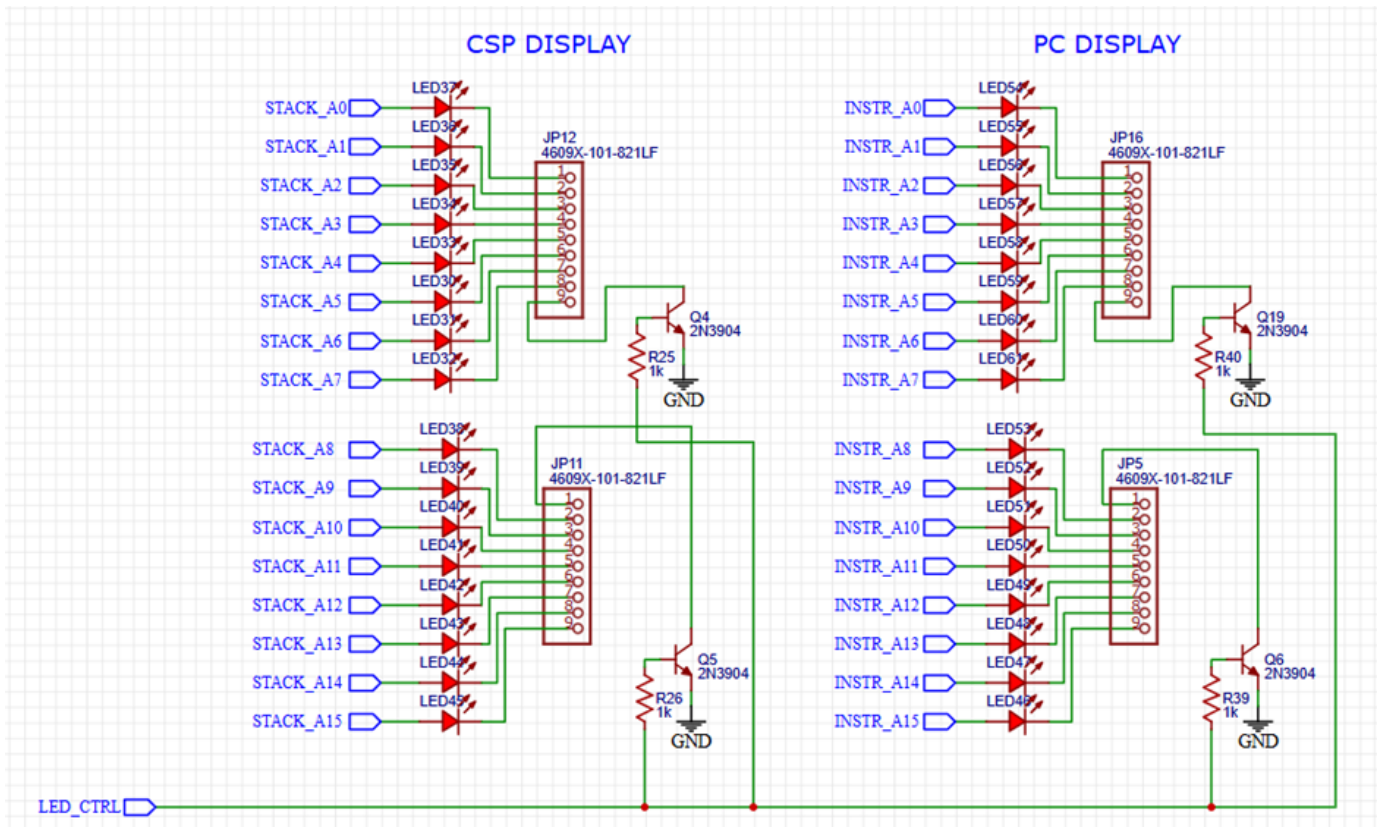


Fig. 29. PC and CSP LED Display Circuitry

by an RC network when the power supply is turned on. An LED turns on when a fault is detected and a relay is turned on with a 2N3904 to connect +5V to the output when no fault is detected. This way, if a fault is detected, the power supply has to be turned off and back on again to reset itself. The schematic of the protection circuitry described can be seen in Figure 37.

VII. TEST PLAN

While our test plan was perhaps not as rigorous as it could have (or should have) been, it worked for our purposes. First, we tested the register file board (board 1) on its own. [\[\[\[Add Big Man testing process here\]\]\]](#)

After this, we decided it would be easiest to just put the computer together and troubleshoot each board as issues came up. The primary reason for this decision was that, if we wanted to test any other board individually, we would have to have used the Arduino Mega to simulate a data bus, something that we felt was unnecessary. We felt safe with this course of action because we were very confident in our designs, as we had checked them all several times between several people, and because each board was fairly independent, so if something went wrong on one, it would be easy to pinpoint where the issue was. These assumptions turned out to be true.

Our overall approach was to write small test programs of only a few instructions to test each opcode individually. We first tested to make sure we could write programs successfully.

We ran into two issues here. The first was that this display for the instruction register had been wired incorrectly, so it was not displaying the value that had been stored. This we fixed in software by changing the data on the flash chips for the 7-segment displays. The second was that bits 4 and 5 of the instruction were always being set to 1. After some more testing, we found that this was an issue with the Arduino Mega we were using to write to instruction memory. We turned off serial communication to the laptop on the Mega (since that uses pins 4 and 5), which fixed the issue.

After those tests were completed, we moved on to testing individual instructions. We first tested register-to-register movement and immediate-to-register movement. We had to fix a small software issue where the PC was not incrementing twice during the load immediate instruction, but it otherwise went smoothly.

We then checked all of the ALU operations: bitwise AND, NOT, OR, and XOR, logical shift right and logical not, add, and subtract. The only issues with these were (1) a faulty latch on one of the parameter registers, and (2) a software fault in the ALU flash chip data, both of which were simple fixes: we replaced the latch and spent some time debugging the incorrect ALU.

At this point, we also tested register-to-memory movement, and memory-to-register movement, both of which worked just fine. During our testing, however, we uncovered a wiring issue

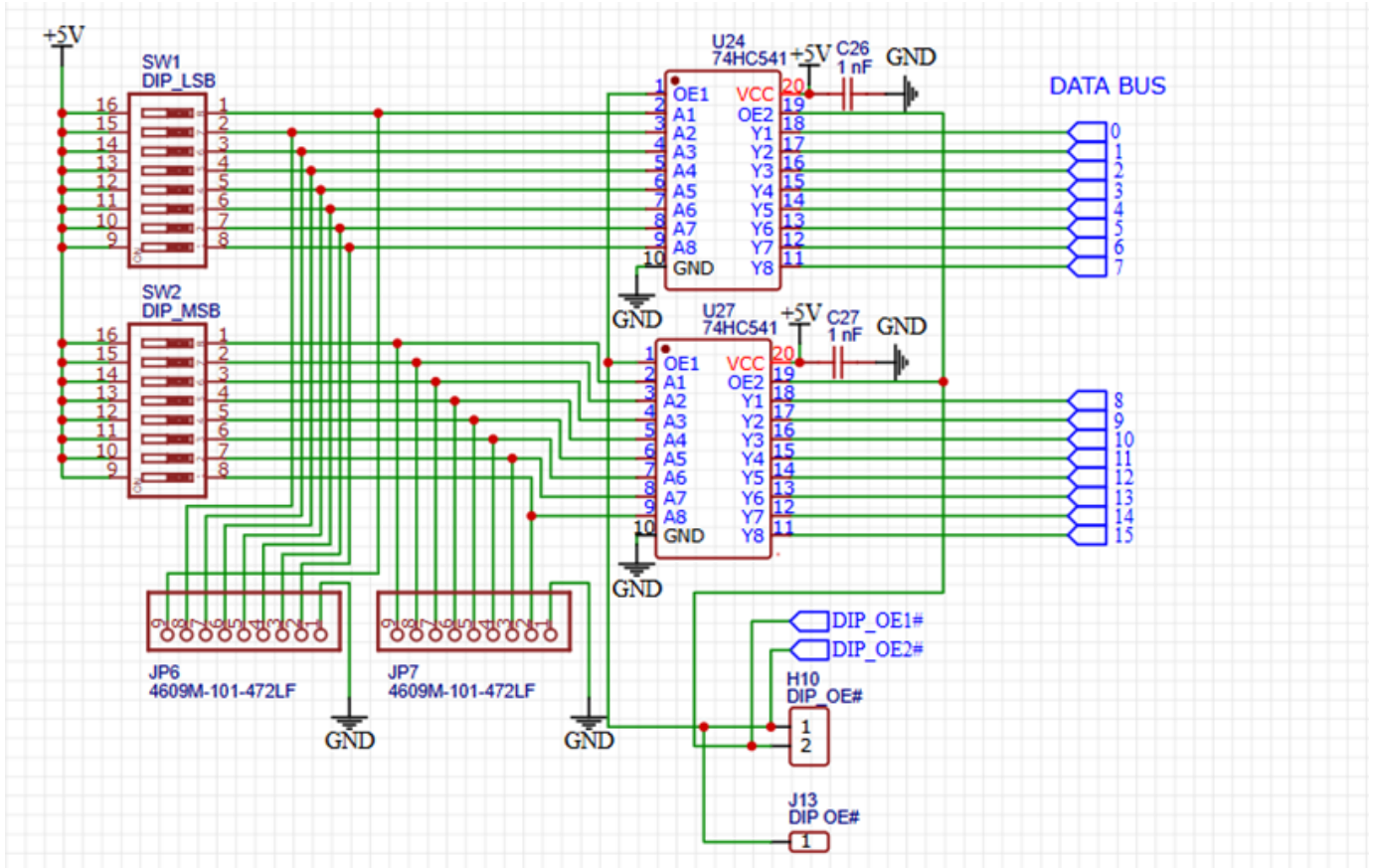


Fig. 30. Dipswitch Circuit Used to Set Interrupt Jump Location

with the program counter (since this was the first time we wrote a program longer than 16 instructions, and the issue was on bits 4-7) that we fixed with a few extra wires.

The last four operations we had to test were push, pop, unconditional jump, and conditional jump. There were once again two issues here. The first problem was that we had forgotten to put a latch on the output of the compare logic, so we added a daughter board with a few chips to create a latch matching the others on the board, with three write-enable signals: one active-low master write NANDed with two ORed active-low write signals, one from the control signals FSM and one from the sub-sub-clock (used for timing signals to prevent race conditions). The second problem was an FSM design issue: on the "push PC" instruction, we should have been pushing PC+2, not the PC+1 that we were pushing. This was easily fixed in software by changing the data on the FSM flash chip.

Finally, we tested the peripheral slots. We created a little makeshift input/output peripheral to test each slot. These tests revealed many small issues, most of which were not recorded because we were iterating so fast. After several hours of debugging, designing, testing, and iterating, we were able to get the peripheral slots working.

Our very last step was to run a comprehensive test program. It tested every instruction and stored a value in the last register

to indicate which instructions had worked and which hadn't. We had run the program throughout the testing process on several occasions, but up until this point, it had not worked properly. After fixing all of the bugs, however, the test program worked perfectly on every clock speed setting.

VIII. PHYSICAL CONSTRAINTS

A. Design and Manufacturing Constraints

Because the project is intended for the educational market, there were not many physical restraints placed on the project. Size was not important as this is not meant to be a pocket sized device. It just needs to be small enough for one person to reasonably carry it which it is. There were also no restraints on power consumption. This device did not need to be low power as it is not portable and does not need a battery. There were no CPU limitations as we designed the CPU ourselves so we could do whatever we needed. For instance, we chose for the stack RAM to be separate from the data RAM. This gives the advantage of having 64k stack locations and the user not being able over-right the stack by accident. This is usually not a commercially available feature in CPUs but we could implement it because we designed the CPU itself.

One limitation was PCB size. We ordered our PCBs from JLCPCB and their maximum size they will manufacture is

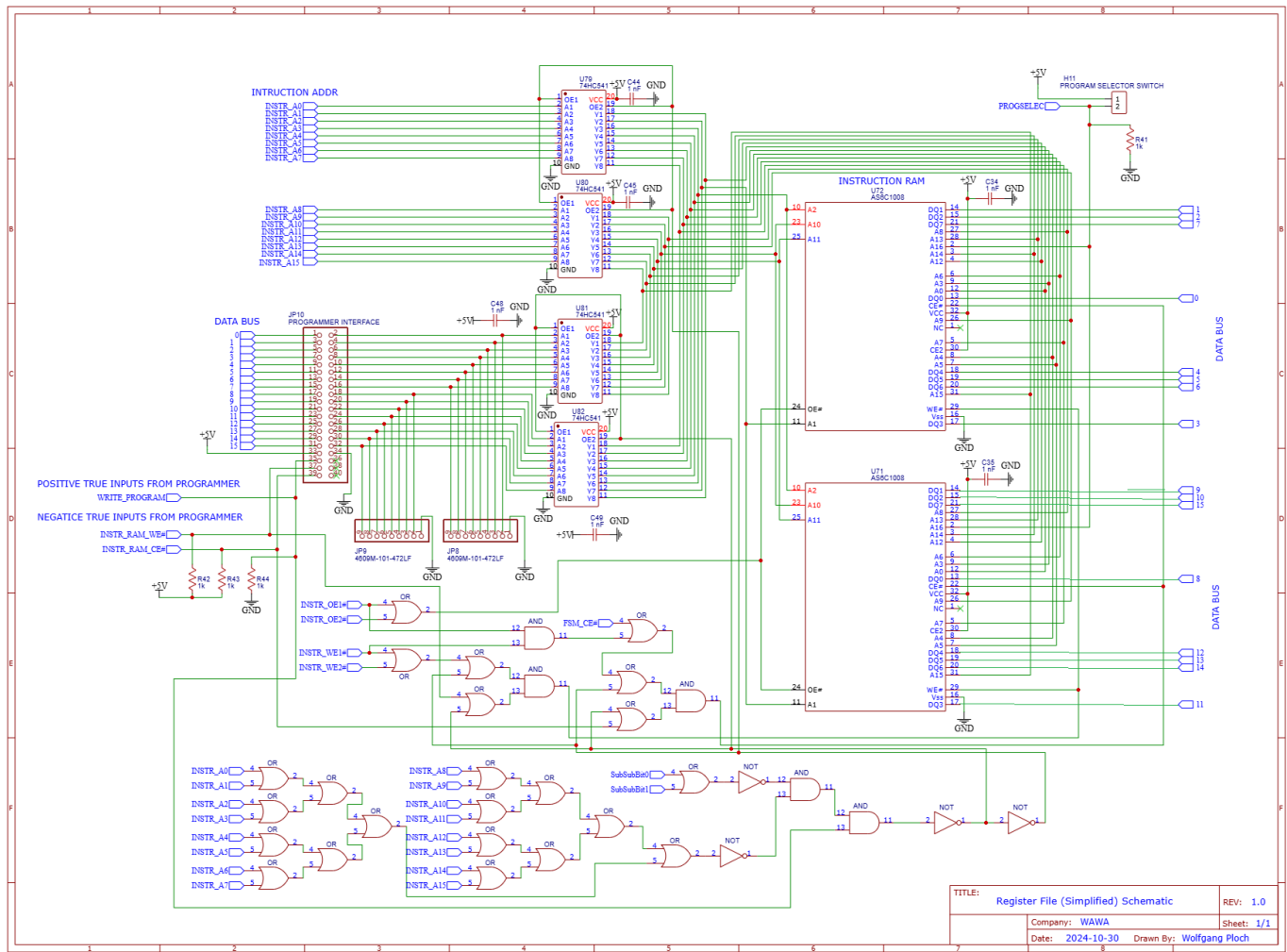


Fig. 31. Instruction SRAM, Programmer Interface, and Associated Circuitry

15x19 inches. Our 6 PCBs exceed that in total area, necessitating the circuit to be split among multiple PCBs

Part availability did constrain the project slightly. There are a lot of 74XX series chips that would have suited our circuits well that have unfortunately been discontinued. These parts are simply obsolete today. We were only able to use the 74LS154 decoder because Wolfgang Ploch had some in his possession, they have been discontinued on all major electronic part websites. Another part that was hard to find was 64k SRAM chips, only 32k and 128k chips could be found. This was used as a positive with the instruction RAM. The extra memory space was leveraged to allow the computer to store two programs simultaneously.

The time frame also constrained the project. While the computer itself functions perfectly as intended, we ran out of time to explore the peripherals we wanted to. For instance, there was no time to design a driver for the 64x64 RGB LED matrix intended to be used. The computer can function well with the Ploch Teletype peripheral, it acts as a terminal.

B. Tools Used in the Project

1) *Visual Studio 2022*: Visual studio 2022 was used to write the C programs associated with the computer. The simulator, assembler, and compiler were written in C using VS. Programs to generate the hex files needed to program the flash chips for the hexadecimal displays, finite state machine, and arithmetic and logic unit were also written in C using VS.

2) *Digital*: This software allows users to drag and drop logic circuit components and create circuits. These circuits can then be tested with specific test cases or in an interactive mode. This software was used to validate parts of the circuits that were complex and in need of verification before manufacture (finite state machines, counter, etc.).

3) *Multisim*: Multisim was used briefly in the design of the power supply to verify the function of the differential amplifier and a filter used in the circuit.

4) *Arduino IDE and Arduino Mega*: The Arduino Mega was used to program the flash chips that contain the encoding for the hexadecimal displays, the ALU and the FSM. This micro-controller was chosen as it had the required amount of

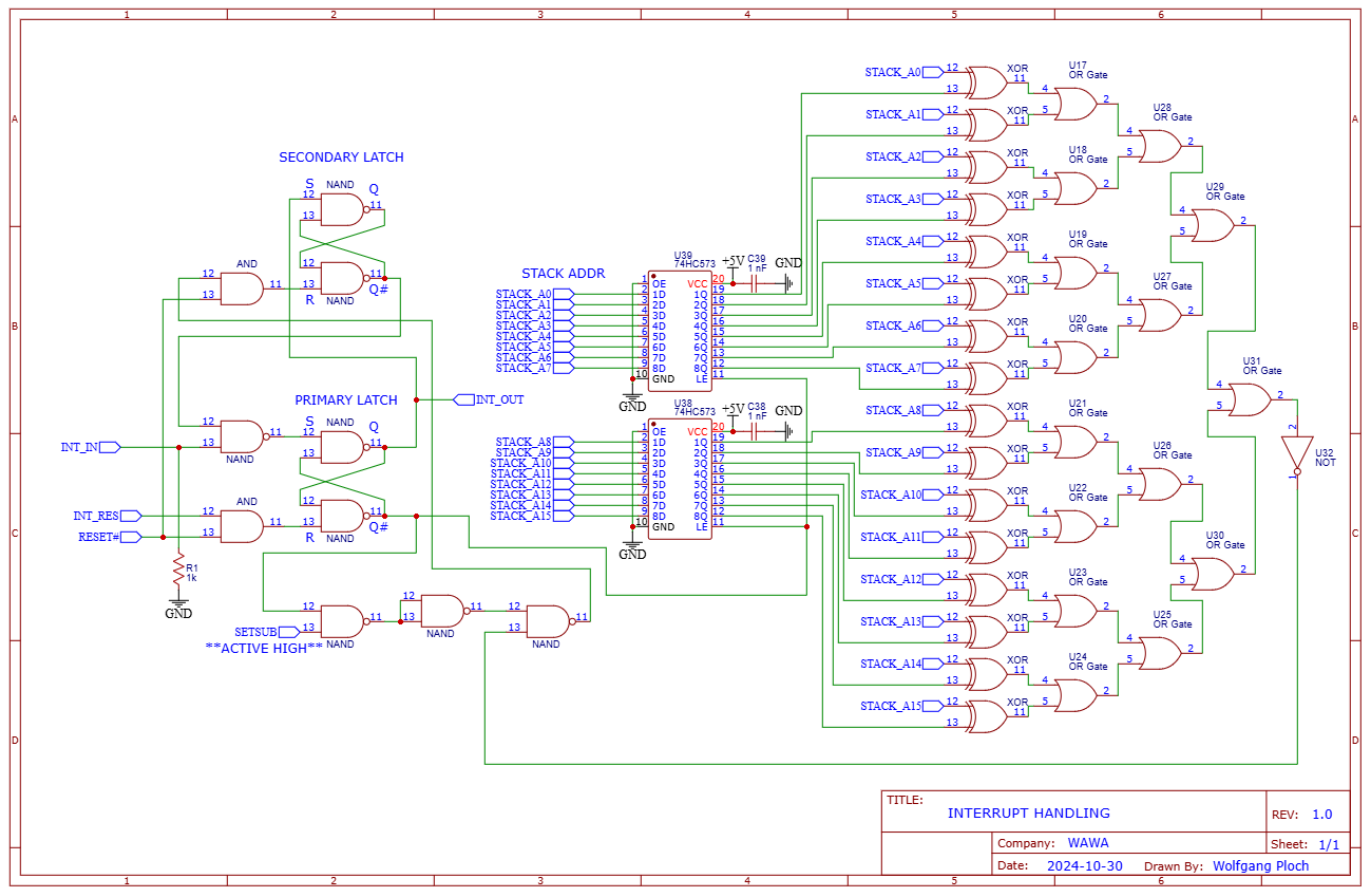


Fig. 33. Interrupt Handling Circuitry

D. Producing a Production Version

To produce a production version, a supplier for the transformer used in the power supply as well as the current meter would need to be found. These parts were found for free to use for the project. The power supply in general would probably be changed to a commercial supply due to the cost of the parts involved and the inefficiency of the current design. A simpler method would need to be established for connecting the boards together. the current method involves a lot of labor and could not be streamlined. The size of the device would also need to be reduced as it is pretty unwieldy in the current state.

IX. SOCIETAL IMPACT

Our project could have impacts on a few different areas of society. The first impact, and the one we mainly intend, is on the academic field. Our hope is that this project will be used to teach college students how computers work in an physical, visual, easy-to-understand way. We hope that this, in turn, will help create better electrical and computer engineers who make more effective designs in their fields of work.

It is also possible that it has a small impact on the environment– if it ever becomes a mass-produced product (which we do not have plans for at the moment, but will not

rule out), the computer does draw a considerable amount of current, which consumes more power and, if enough devices were to be sold, could have an impact on the environment. We find this outcome incredibly unlikely, however, and while we have not done any rigorous calculation, we believe that the environmental impact of a large collection of Romulus Is would be negligible compared to most major sources of pollution.

X. EXTERNAL STANDARDS

External Standards

XI. INTELLECTUAL PROPERTY ISSUES

We believe that Romulus I is patentable. We looked at a few patents potentially similar to our project to get a sense of what inventions were in the space already.

The first patent we looked at was the Kenbak-1 personal computer. It was created as a teaching tool, designed to help students learn how to write simple programs [4]. The patent has only one independent claim describing the specifications of the device, and no dependent claims [5].

While this sounds similar to our own product, ours is significantly different in a number of ways. First, the Kenbak-1 is not at all transparent– it's just a box with some buttons

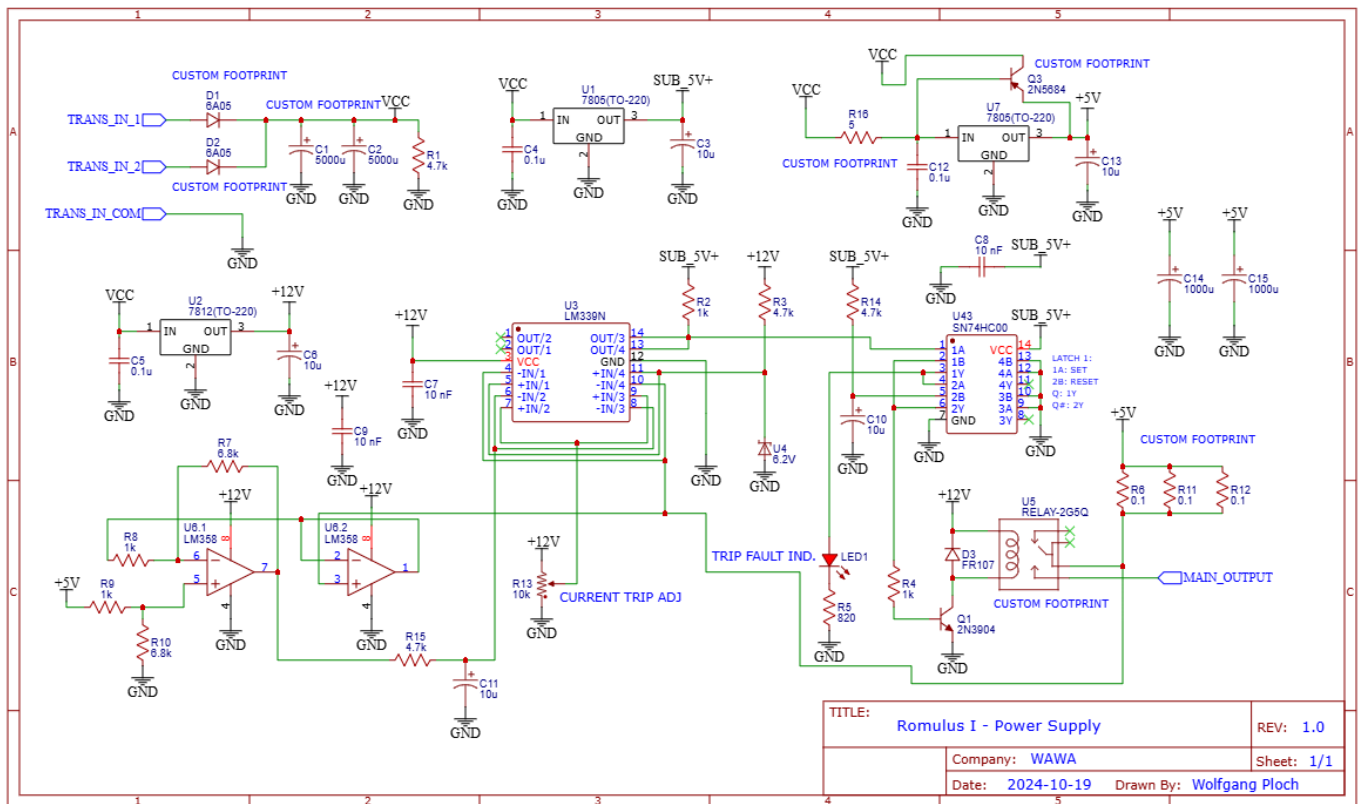


Fig. 34. Full Schematic of Power Supply Board

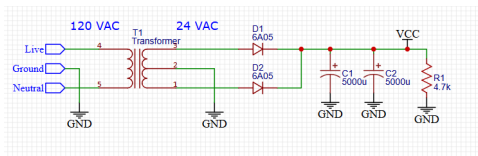


Fig. 35. Transformer and Rectifier Portion of Power Supply Schematic

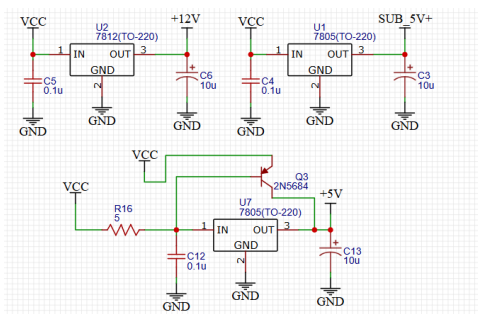


Fig. 36. Voltage Regulator Portion of the Power Supply Schematic

and a few lights. Romulus I has displays and lights for many different parts, making it more transparent. Our device also has multiple clock speeds and the manual step option, neither of which were in the Kenbak-1. It is also generally more powerful, with a higher maximum clock speed and instructions per second, more registers, more storage space, and more

operations [4].

The second patent we looked at was the Apple I computer by Steve Wozniak. This was another, slightly more recent personal computer with support for a screen and keyboard [6]. The patent has two independent claims, both of which focus on the technology used to connect to a video display, and six dependent claims, which discuss individual parts of the independent claims [7].

Our product is once again different for many of the same reasons. Although the Apple 1 is just a breadboard with chips on it, so the user can see the entire circuit laid out, it is not designed for readability and education, like ours is. It also again lacks the adjustable clock speed and manual step option [6].

The last patent we looked at was a digital logic simulator by Yifatch Tzori. There are four independent claims: One that discusses the overall system, one that focusses on the hardware with the logic chips in it, one involving specific algorithm the software uses, and one that describes another, faster-performing system. The 36 dependent claims again discuss smaller systems contained within the full product [8].

This product is also significantly different from ours. It cannot run actual programs, as it is a logic simulator, and it is not specifically designed for education, like ours is.

XII. TIMELINE

Timeline

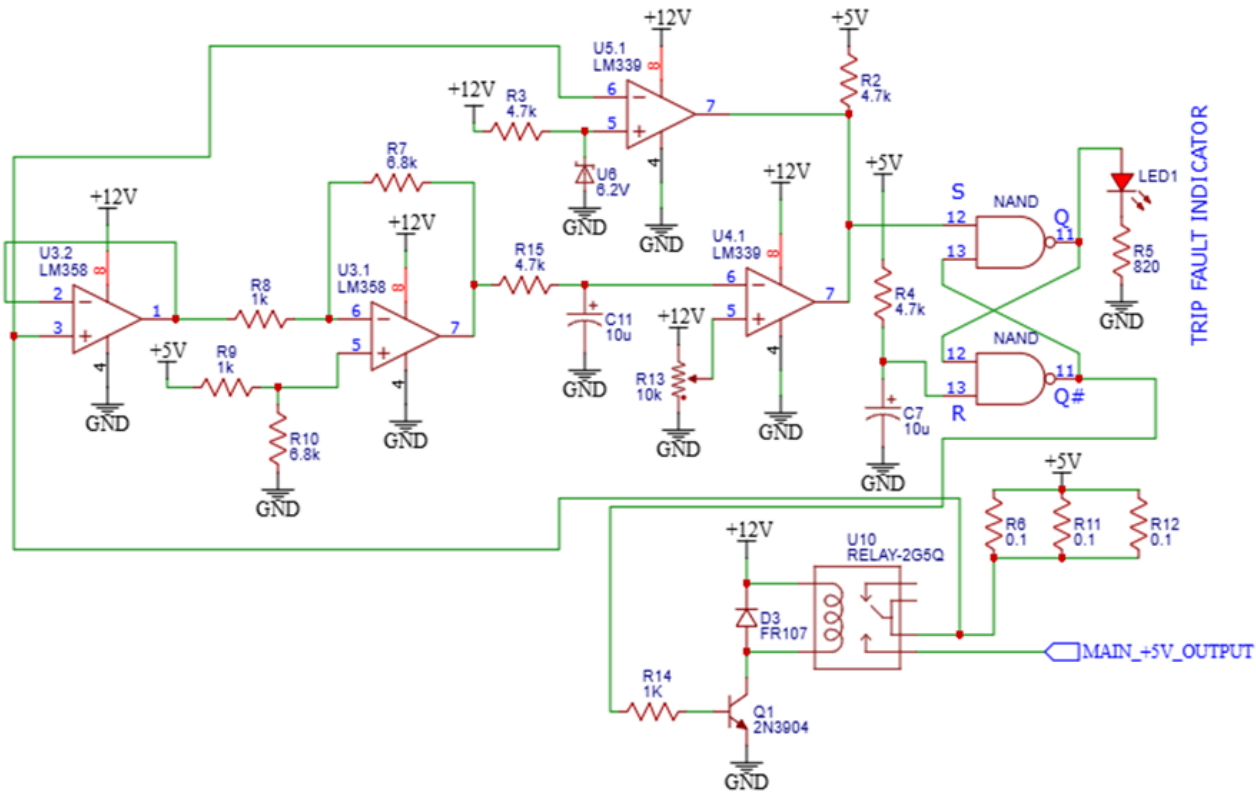


Fig. 37. Protection Circuitry Portion of Power Supply Schematic

XIII. COSTS

This project required the use of many different components. It also utilized parts that are not available anymore and parts that were received from surplus locations for free. A detailed spreadsheet of the part, source, quantity, cost per 1, 10, 25, 50, 100, 1000, 2500, 5000, 10000, and 25000 can be found in the Appendix. Parts that have an asterisk (*) were received for free and the costs are an estimate. Parts that are marked with a double asterisk (**) are slightly different models compatible models than the ones used in the project as some part were obsolete or could not be located. The 6 boards are also included in the spread sheet. The costs in the sheet neglect the shipping costs of the items. The cost to manufacture one device (Including shipping) was roughly \$ 1000 excluding the previously mentioned parts received for free. The estimated costs for manufacturing 100 and 10000 units neglect the price of shipping.

The manufacturing of the boards would not be able to be streamlined using a pick and place machine and reflow oven as all of the components on the boards are through hole devices and those tools only work for SMD components.

XIV. FINAL RESULT

Final Result

XV. ENGINEERING INSIGHT

Engineering Insight

XVI. FUTURE WORK

Future work

REFERENCES

- [1] H. Neemann, "hneemann/Digital," GitHub, Oct. 27, 2024. <https://github.com/hneemann/Digital> (accessed Dec. 03, 2024).
- [2] "Toy ISA Simulator," Github.io, 2024. <https://researcher111.github.io/uva-cs01-F23-DG/homework/files/toy-isa-sim.html> (accessed Dec. 03, 2024).
- [3] B. Eater, "Build an 8-bit computer from scratch." <https://eater.net/8bit/> (accessed Dec. 03, 2024).
- [4] "Kenbak-1 Computer," www.kenbak.com. <https://www.kenbak.com/kenbak-1-home> (accessed Dec. 04, 2024).
- [5] L. Amdahl, G. Amdahl, H. Engel, E. Schneberger, and J. Blankenbaker, "Stored logic computer," Apr. 12, 1966 Accessed: Dec. 04, 2024. [Online]. Available: <https://patents.google.com/patent/US3246303A/en>
- [6] "Apple I Microcomputer," National Museum of American History, Behring Center, 2015. https://americanhistory.si.edu/collections/object/nmah_1692121 (accessed Dec. 04, 2024).
- [7] S. Wozniak, "Microcomputer for use with video display," Jan. 23, 1979 Accessed: Dec. 04, 2024. [Online]. Available: <https://patents.google.com/patent/US4136359A/en>
- [8] Y. Tzori, "Digital logic simulation/emulation system," May 05, 1998 Accessed: Dec. 04, 2024. [Online]. Available: <https://patents.google.com/patent/US5748875A/en>

XVII. APPENDIX

TABLE V
COMPONENT, SOURCE, COST PER 1, COST PER 100, COST PER 10000

Part	Source	Quantity	Cost per 1 (USD)	Cost per 10 (USD)	Cost per 25 (USD)	Cost per 50 (USD)	Cost per 100 (USD)	Cost per 500 (USD)	Cost per 1,000 (USD)	Cost per 2,500 (USD)	Cost per 5,000 (USD)	Cost per 10,000 (USD)	Cost per 25,000 (USD)
Board 1 PCB	JLCPCB	1	27.48	NA	NA	NA	12.55	NA	NA	NA	NA	10.84	NA
Board 2 PCB	JLCPCB	1	8.04	NA	NA	NA	2.31	NA	NA	NA	NA	1.80	NA
Board 3 PCB	JLCPCB	1	8.38	NA	NA	NA	2.38	NA	NA	NA	NA	1.87	NA
Board 4 PCB	JLCPCB	1	11.60	NA	NA	NA	5.15	NA	NA	NA	NA	4.29	NA
Board 5 PCB	JLCPCB	1	10.56	NA	NA	NA	4.13	NA	NA	NA	NA	3.38	NA
Board 1 PCB	JLCPCB	1	2.10	NA	NA	NA	0.79	NA	NA	NA	NA	0.55	NA