Final Report

Senior Design 492

Client/Advisor: Professor Alexander Stoytchev

Team 14 - Hardware Implementation of i281 Processor

Team Members:

Daryl Damman	Team Lead
Logan Lee	Scheduling
Grant Nordling	Parts Manager
Braxton Rokos	Routing Lead
Gavin Tersteeg	Technical & Project Assembly Lead

Website: https://sdmay24-14.sd.ece.iastate.edu/

Table of Contents

Table of Figures

Table of Tables

Table of Acronyms

Acronym	Name
ALU	Arithmetic Logic Unit
BB	Breadboard
BIOS	Basic Input/Output System
BOM	Bill of Materials
CMEM	Code Memory
CPR E	Computer Engineering
CPU	Central Processing Unit
DIP	Dual Inline Package
EE	Electrical Engineering
DMEM	Data Memory
EEPROM	Electrically Erasable Programable Read-Only Memory
EPROM	Erasable Programable Read-Only Memory
ETG	Electronics and Technology Group
FPGA	Field Programmable Gate Array
FR	Functional Requirements
GND	Ground
HC	High-Speed CMOS
HCT	High-Speed CMOS with Transistor-Transistor Logic Voltages
IC	Integrated Circuits
IEEE	Institute of Electrical and Electronics Engineers
LED	Light Emitting Diode
LS	Low-Power Schottky
LSB	Least Significant Bit
MSB	Most Significant Bit
MUX	Multiplexer
PC	Program Counter
PCB	Printed Circuit Board
PWB	Printed Wiring Board
QR	Quantitative Requirements
RAM	Random Access Memory
ROM	Read-Only Memory
SR	Qualitative/Subjective Requirements
SW	Switch
TTL	Transistor-Transistor Logic

Table 1 - Acronym Definition List

1 Introduction

The i281e CPU team is composed of three electrical engineering and two software engineering students (see Table 2). Throughout the last two semesters, we have accumulated a wealth of knowledge and skills from designing and developing a finished i281e CPU prototype. This document marks the design journey and the final details of the project as outlined by the requirements of the Senior Design 492 class.

Primary Degree	Member Name
Electrical Engineering	Logan Lee
	Braxton Rokos
	Grant Nordling
Software Engineering	Gavin Tersteeg
	Daryl Damman

Table 2 - i281e CPU Member List with Majors

1.1 Historical Context

In 2018, Dr. Alexander Stoytchev and Kyung-Tae J. Kim began development on a MIPS-based, single cycle processor that could be taught in CPR E 281 (now CPR E 2810). By Summer 2019, a finished implementation of the processor was developed for the Altera DE2-115 FPGA boards. Curriculum was written for the following Fall semester. During the Fall 2019 semester, the i281 CPU was unveiled to the class to show the culmination of the skills learned throughout the semester.

In Fall 2020, Dr. Stoytchev submitted a project proposal for Senior Design to develop a web simulation of the i281 CPU. This simulator would make interacting with the processor significantly easier, especially if Altera boards were inaccessible or otherwise occupied by other students. A group of six students were formed into sdmay21-38 to develop the web simulator (see Table 3 and Figure 1).

Primary Degree	Member Name
Computer Engineering	Aiman Priester
	Eric Marcanio
	Bryce Snell
	Brady Kolosik
	Jacob Betsworth
Software Engineering	Colby McKinley

Table 3 - Former i281 Web Simulator project members

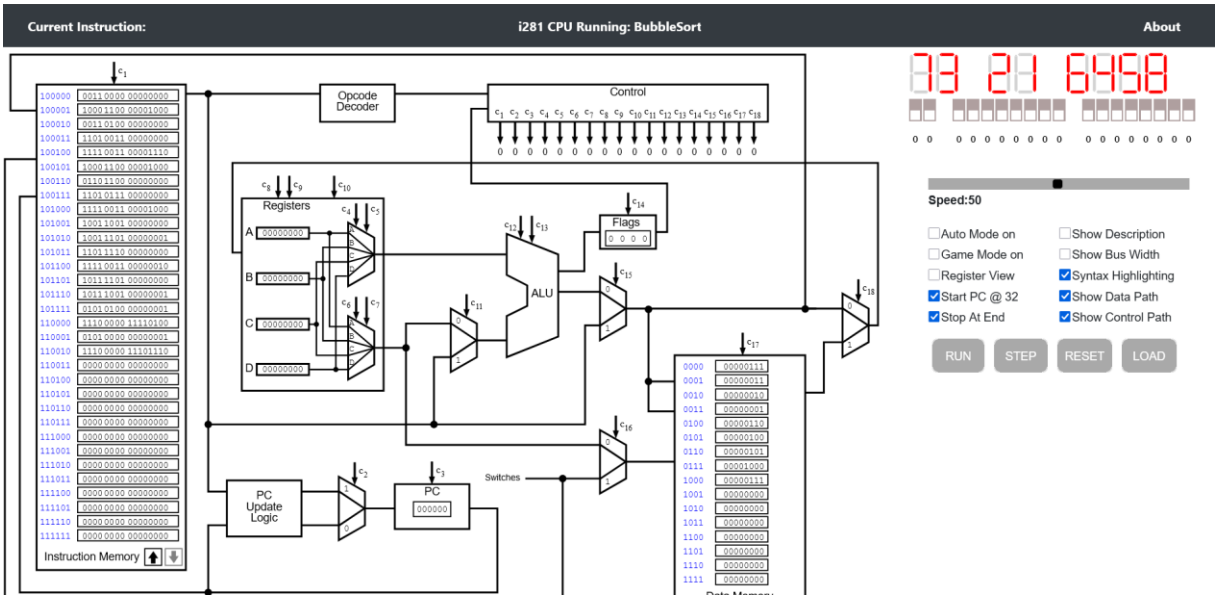


Figure 1 - i281 CPU Web Simulator

The i281 CPU is still taught in CPR E 2810 classes to this day; however, it has a far grander potential waiting to be possessed. Was it possible to simply produce the i281 CPU into a physical device? In Spring 2022, another Senior Design team (sddec22-20) was formed. An effort was made to take the original design of the i281 CPU and produce a physical device. The goal was to make a breadboard version first to ensure logic was correct and sound.

Primary Degree	Member Name
Software Engineering	Saffron Edwards
Electrical Engineering	Joseph De Jong
	Alex Kiefer
	Patrick O'Brien
Computer Engineering	David Vachlon

Table 4 - Former i281 CPU Hardware Implementation members

Unfortunately, time was not an ally with the previous team and while a breadboard implementation was finished by December 2022, it had some fundamental flaws that hindered it from going further.

1.2 Problem

The overarching goal for the project, as declared by the project name, is to provide a physical hardware implementation of the i281 processor without the use of an Altera board, and further, any FPGA-based platform. This implementation will make use of concepts from our classes about electronics circuit design, digital logic, and PCB design.

Our primary vision for this project is to be used as an extended teaching tool for CPR E 2810 to represent the operations of the processor better both physically and visually. As seen with the Altera board, internal registers and calculations weren't visible for students to view and the online simulator is pure software lacking a physical device to tactilely interact upon. Students would benefit greatly from a more detailed examination of the internal workings of the processor.

Alongside being a teaching tool for CPR E 2810, the final objective would be to introduce the final product as the primary processor for junior and senior students in CPR E 4810. This would expand upon the processor design and allow students to fully explore modifying architectures and developing applications/peripherals for processors in a simplified manner.

1.1 Intended Users and Usage

There are three groups that would be considered intended users:

- Dr. Alexander Stoytchev and CPR E 2810/4810 teaching staff
- Sophomore students in CPR E 2810
- Junior and senior students in CPRE 4810

1.1.1 Teaching Staff

The project must be accessible to teaching staff for both CPR E 2810 and 4810 through both design choices and documentation. A comprehensive archive of source files regarding schematics, PCB designs, CAD work, custom software, and documentation for using all aforementioned components must be made available.

1.1.2 CPR E 2810 Students

It is assumed that the earliest students of CPR E 2810 will be sophomores with minimal background to digital logic. As such, the design of the i281e CPU must be accessible. This means the product must be thoroughly labeled and step-by-step guides be written. The students will interact with the product as if it were an Altera board but have far more flexibility in interaction methods.

1.1.3 CPR E 4810 Students

It is assumed these students have already taken both CPR E 2810 and 3810. These students will have a deeper understanding of both digital logic and microprocessor design. While the product must be thoroughly labeled to aid with debugging for these students, the primary focus would be to extend the architecture and peripherals of the current design. Advanced documentation must be accessible for the students to learn finer details about how the expansion bus works, file system design, and to change the instruction set.

These students are also expected to be capable of replacing components and interfacing their own solutions into the design. Breadboard and PCB solutions must be compatible with each other.

1.2 Previous Work and Considerations

As mentioned in the subsection regarding Design Complexity, a homebrew computing kit exists on the market to build 8-bit retro computers through Ben Eater's 8-bit Computer kit[1]. Table 5 provides a non-comprehensive side-by-side comparison between Ben Eater's computer kit and the i281e CPU design as developed by our team.

Ben Eater's 8-bit Computer	i281 CPU
Comprehensive build kit, costs $315	Chips must be purchased and sourced individually. Will cost over $315
A full walkthrough of how the CPU works and performs shown in a video on Ben Eater's channel.	The FPGA and simulator created for the CPRE 281 class with lecture slides on how the CPU works.
Uses 74LS series chips.	Uses 74HCT series chips.
Has hidden fees for equipment not included in the kit. Ex. Oscilloscope, Multimeter, etc.	Equipment is provided by ETG and Iowa State University.

Table 5 - Comparison Between Ben Eater's CPU and the i281 CPU

As noted in the historical context, a previous team attempted to implement the i281 CPU in hardware once before; however, were unable to do so. Our goal is to leverage insights and challenges found from the previous team into our design considerations and development. Our timeline also encompasses the second phase of the project, focusing on designing and producing PCBs using KiCad. While the previous team were unable to design PCBs before the end of the project, our team has been able to successfully develop multiple iterations of PCBs to showcase the computer's processes, serving as an educational tool that is robust and less prone to mechanical failure.

To learn from the previous team's experience, our approach involves categorization, general organization, and careful consideration of the intricacies that the previous team found challenging. To maintain constructive feedback, Dr. Stoytchev provided remarks on what worked and didn't work in the last project, offering valuable insights without denigrating the efforts of the previous team. The prior project faced challenges, resulting in a non-functional breadboard computer due to design inconsistencies and insufficient circuit care.

[1] https://eater.net/8bit/kits (accessed Dec. 03, 2023).

2 Design

2.1 Engineering Standards

The i281e CPU does not adhere to many modern-day standards and practices as the technology used for the project is rooted in the early days of electronic computing. The standards chosen for this project are described by the Institute of Electrical and Electronics Engineers (IEEE) to help us consider design choices and improve development.

- IEEE 162-1963
    - IEEE 162-1963 describes the standard definitions and terms for digital computers. As the project is intended to be educational, using the appropriate terminology for digital computing and related components is paramount for a comprehensive curriculum.[2]
- IEEE 370-2020
    - IEEE 370-2020 describes a standard for predicting electrical characteristics on printed circuit boards and other related interconnects at frequencies up to 50 GHz. This is relevant to our project because we will need to handle signals running at up to 1 MHz on our printed circuit boards for the final product.[3]
- IEEE 2716-2022
    - IEEE 2716-2022 provides a guide for characterizing the effectiveness of printed circuit board level shielding. In our project, we will have a dozen or so PCBs all connected with discrete cables. We will need to take shielding into account, so we don't encounter noise-related problems.[4]
- IEEE 696-1983
    - IEEE 696-1983 describes a computer bus architecture for 8-bit computers running at TTL logic levels. Knowledge of how to avoid signal noise, arbitrate device access, and distribute power to all subsystems will come into use for our own project.[5]

---

[2] "IEEE Standard Definitions of Terms for Electronic Digital Computers," in ANSI/IEEE Std 162-1963 , vol., no., pp.0_1-, 1963, doi: 10.1109/IEEESTD.1963.120147.

[3] "IEEE Standard for Electrical Characterization of Printed Circuit Board and Related Interconnects at Frequencies up to 50 GHz," in IEEE Std 370-2020 , vol., no., pp.1-147, 8 Jan. 2021, doi: 10.1109/IEEESTD.2021.9316329.

[4] "IEEE Guide for the Characterization of the Effectiveness of Printed Circuit Board Level Shielding," in IEEE Std 2716-2022 , vol., no., pp.1-46, 29 May 2023, doi: 10.1109/IEEESTD.2023.10136540.

[5] "IEEE Standard 696 Interface Devices," in ANSI/IEEE Std 696-1983 , vol., no., pp.1-40, 13 June 1983, doi: 10.1109/IEEESTD.1983.81971.

2.2 Requirements

Requirements are split into three categories: functional and non-functional, where qualitative (subjective) and quantitative requirements fall under non-functional. These are denoted as functional requirements (FRs), qualitative requirements (SRs), and quantitative requirements (QRs).

Alongside the design requirements, there were a few design constraints from the original design parameters.

2.2.1 Functional

Req. #	Requirement Description
FR-1	CPU clock must permit stepping through instructions and operating at a specific range of frequencies
FR-2	CPU must allow writing custom programs via interface panel
FR-3	CPU must allow loading example programs from the Boot Hard Disk
FR-4	CPU must be capable of playing a version of the i281 PONG example program
FR-5	Instruction decoding must handle active high and low signals
FR-6	All internal storage and calculations must be visualized through LEDs
FR-7	CPU booting must appear instantaneous to students
FR-8	CPU execution must allow for single-instruction or continuous execution
FR-9	CPU must allow loading example programs from ROM
FR-10	EEPROMs must be used for the control line logic

Table 6 - Functional Requirements

2.2.2 Non-functional

2.2.2.1 Qualitative/Subjective Requirements

Req. #	Requirement Description
SR-1	Data bus cables must be clearly labeled
SR-2	Data bus cables must have the zeroth bit on the right-hand side
SR-3	EEPROMs must be either the same chip or hot-swappable
SR-4	RAM chips must be either the same chip or hot-swappable
SR-5	Visualization for the current address (program counter) must be one color
SR-6	The project must be aesthetically pleasing and attractive
SR-7	CPU must be explainable to CPR E 281 students
SR-8	CPU must be capable of being modular
SR-9	Any implementation (breadboard or PCB) must be interchangeable
SR-10	CPU must be readable from one viewing direction
SR-11	CPU must be fully labeled
SR-12	CPU design must be as close as possible to the original Verilog design

SR-13	Bus entry and exit need to be clearly labeled with direction and connection type
SR-14	LEDs and related visualizers must be color-coded
SR-15	Singular direction (northern indicator) must be standardized and rigorously followed for all CPU components and visualizers
SR-16	A standard cable and LED color code must be used to distinguish

Table 7 - Qualitative Requirements

2.2.2.2 Quantitative Requirements

Req. #	Requirement Description
QR-1	RAM must hold at least 64 words/instructions
QR-2	Program addressing must be at least 6 bits
QR-3	Visualized binary data must be represented in 2's complement by reading the most significant bit (MSB) on the left to the least significant bit (LSB) on the right
QR-4	CPU must achieve 1MHz clock speed on PCB implementation

Table 8 - Quantitative Requirements

2.2.3 Design Constraints

The constraints of the original design have been updated to be more descriptive and representative of limitations of both the breadboard and PCB implementations.

Req. #	Requirement Description
C-1	Both ROM and RAM must use Big Endian
C-2	All breadboards and PCBs must be labeled
C-3	All modules must be constructed using continuously obtainable components
C-4	Modules should be electrically self-contained to show logical separation
C-5	All instructions must be able to execute in a single clock cycle.

Table 9 - Design constraints

2.3 Security Concerns

Security is not a concern for this project. While we have made considerations about security, none were implemented for the sake of project complexity and the lack of requirements. Security features and further considerations will be a topic in the far future for students examining the hardware implementation of i281e processor. Additionally, no security testing was performed on either breadboard or PCB implementations.

As of now, our project's primary focus has been on functionality, performance, and design optimization. Security testing, while crucial, was not within the scope of our current

objectives. However, it remains a critical aspect to address in future iterations or related projects to ensure robustness and resilience against potential vulnerabilities.

2.4 Design Complexity

The i281 processor is more complicated due to the design choices made compared to modern processors. It utilizes a variety of TTL-style logic chips to provide a multi-use execution environment. There are numerous components to the full processor of which most are present on modern processors:

- ROM w/ BIOS
  - This represents where the processor will begin executing instructions, dictating start up procedures and beyond.
- User RAM
  - Programs are loaded into RAM via BIOS startup or program request. These programs will operate the CPU once the ROM has completed its instruction set.
- Register File
  - Handles intermediate volatile memory to store results from the ALU, Data Memory, or instruction immediate values.
- Arithmetic Logic Unit (ALU)
  - Performs the basic arithmetic for instructions via data stored in the Register File.
- Program Counter
  - 8-bit program counter that only uses the lower 7 bits to indicate where in instruction memory the processor is currently executing from.
- Data Memory and Video Card
  - Outputs data from the Data Memory onto eight seven-segment displays. Each segment of each display must be individually togglable to allow more complicated programs.

Beyond the CPU components, there are also complex tasks of handling items related to the components.

- Data Bus
  - Connects component to component in a cleaner and easier to read method.
- Visualizations
  - Individually represent signals and individual bits in registers.

These components do not match or exceed modern solutions. Modern processors are built on silicon chips through lithography, ion doping, electroplating, and etcetera. These processors can and will perform far better in all capacities compared to our project. While we

cannot match or exceed modern solutions, the project trumps complexity of a modern processor by being built on breadboards using integrated components (ICs) and wiring rather than being a design on a computer that will be sent to a manufacturing facility.

2.5 Design Iterations

Several design decisions and milestones were held throughout the product lifespan leading to a primary design and an extended version.

The original i281 design showed theoretical promise but required adjustments for practical implementation. Not all elements from the simulator or FPGA design could be directly translated into ICs. Due to the project's limited timeframe, we streamlined the design for efficiency and ease of testing, focusing on chip efficiency. This involved evaluating the balance between component complexity and functionality, ensuring alignment with project objectives, and staying within scope and timeline constraints.

2.5.1 Design 0 — Original i281 CPU Design

2.5.1.1 Design Visual and Description

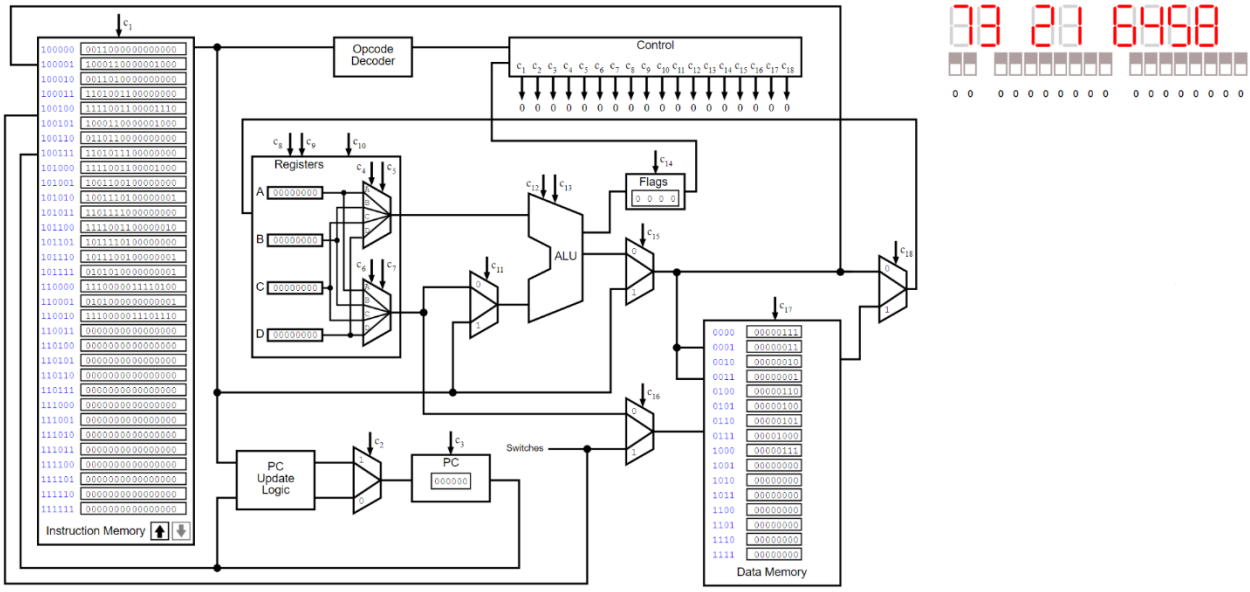


Figure 2 - i281 CPU Simulator Design

The i281 CPU design simulator is depicted above. This design encompasses Design 0 as it was given to us by the client. This includes a BIOS, Code Memory, Program Counter, Opcode Decoder, Control Table, Data Memory, Video Card, Register Files, Arithmetic Logic Unit, Input Switches, and Flag Register Files.

2.5.1.2 BIOS

In the original design, a "loader program" (or BIOS as it is called) was provided to allow users to enter programs into RAM through the interface panel switches (SW15-SW0) depending on switch selection at boot. Alternatively, an example program would be stored in the second half of code memory and the BIOS would jump to the active program segment.

2.5.1.3 Code Memory

Code memory was a 16-bit memory storage solution that contained up-to 128 instructions which were separated in half. The upper half is dedicated to a BIOS and the lower half to the actively used program memory that could be modified while execution was occurring.

2.5.1.4 Register File

The register is responsible for holding data for usage by the ALU. In a modern computer, this is like the code memory component; fast memory for storing the data the computer is actively using. The block of this design can be seen in the two figures below. The register takes many inputs to perform its purpose. The data line, 8 bits, feeds into the register file from a mux, allowing the location to come from the code memory, ALU, or data memory. The register file also has seven dedicated control lines: the write location, write enable, read select one, and read select two.

Since the data line is sent to all four register files, the write enables and location is used to only update the correct register file: A, B, C, or D, when desired. A decoder with enable was used to send the write enable line to the corresponding register file. There are two read-select addresses with two bits each, allowing the register to output two different or the same registers on the two output buses.

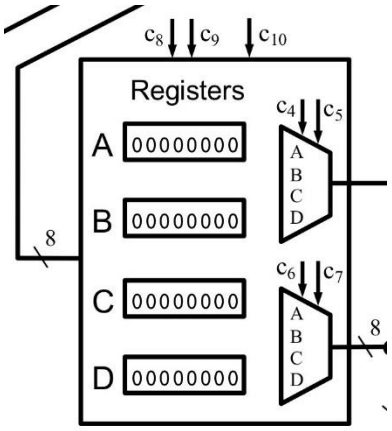


Figure 3 - Register Block from Simulator

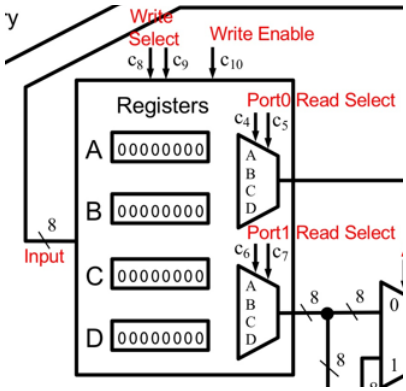


Figure 4 - Register Block from Simulator

Each of the register files is made of the schematic seen below. These use a D flip-flop with a mux to ensure the bit is stored until written to use the write enable. Each register file has the same schematic with different write enable and output bus locations.

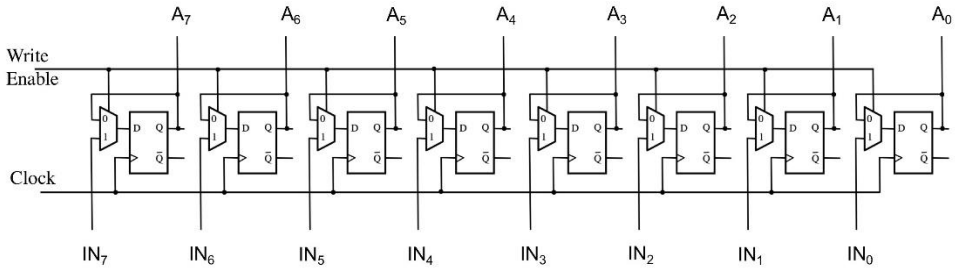


Figure 5 - Register File from Simulator

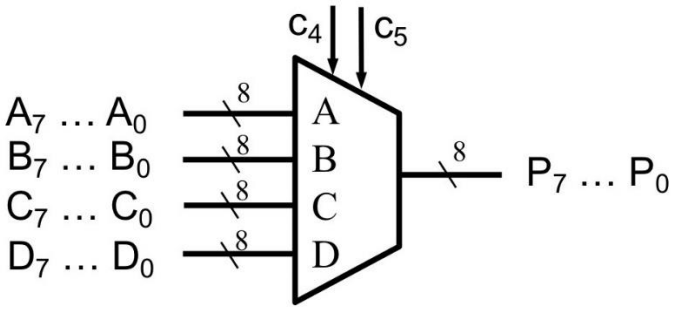


Figure 6 - 8-bit 4-1 Mux

An 8-bit 4-1 mux manages the output of each two-output bus. These are made using 8 4-1 mux with the small selection wires.

2.5.1.5 Arithmetic Logic Unit

For Design 0 of the Arithmetic Logic Unit (ALU), we were given an initial design that was used to create both the simulator and the FPGA design. As we can see from the image below, the ALU contains a few subcircuits: the 8-bit shifter, 8-bit adder, a few multiplexers, and a flag calculator. For our design, we will also be adding the flag registers as part of the ALU design which is just a 4-bit register file. The design employs three control signals. Two of the control signals are labeled

ALU_SELECT0 and ALU_SELECT1 in the picture and they determine the output of the ALU (the second picture shows the different combinations that lead to different opcodes). The third one is to control the flag register. This design will output one 8-bit bus (ALU_RESULT in the picture) and one 4-bit bus (the flag register outputs). The flag register outputs the carry, negative, overflow, and zero flag.

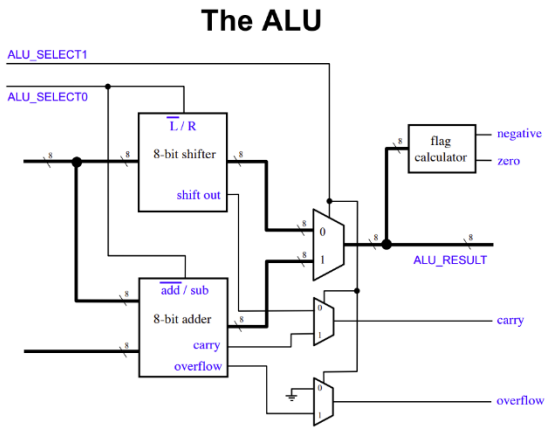


Figure 7 - ALU Subcomponents Diagram

ALU_SELECT1	ALU_SELECT0	Operation
0	0	SHIFTL
0	1	SHIFTR
1	0	ADD
1	1	SUB/CMP

Figure 8 - ALU Arithmetic Mode Table

The 8-bit shifter circuit is capable of shifting a bit either left or right. The ALU_SELECT0 control signal determines which it goes to. If ALU_SELECT0 equals a logic low, then the circuit shifts left and vice versa. The design itself can be seen below. This circuit shows the logic that shifts it both left and right depending on the select signal. This circuit has one extra output called Shift_Out which goes into a multiplexer in the higher up circuit. It works as a carry flag when the ALU_SELECT1 control line is at a logic low.

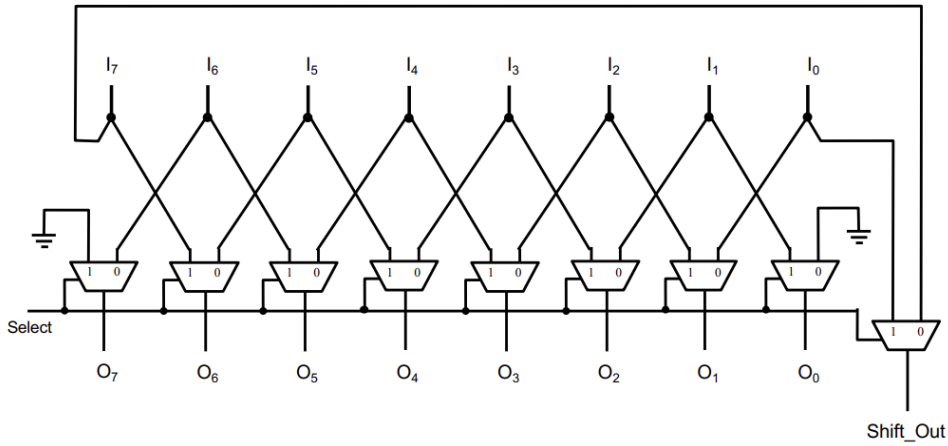


Figure 9 - ALU Shifter Design

The 8-bit adder/subtractor circuit is capable of both adding and subtracting two 8-bit numbers. When the ALU_SELECTO control line is at a logic low, the circuit is in addition mode. When it is at a logic high, then it is in subtraction mode. This circuit is essentially eight full adders tied together in a line. This circuit also outputs a carry bit and a negative bit. All the output bits labeled S_{0-7} can be put into an 8-bit NOR to determine if the value is zero. If it is, it triggers the zero flag. Lastly, the C_7 and C_8 carry bits can be put into an XOR to show if the circuit has overflowed, meaning the result was too large to fit in the number of bits provided.

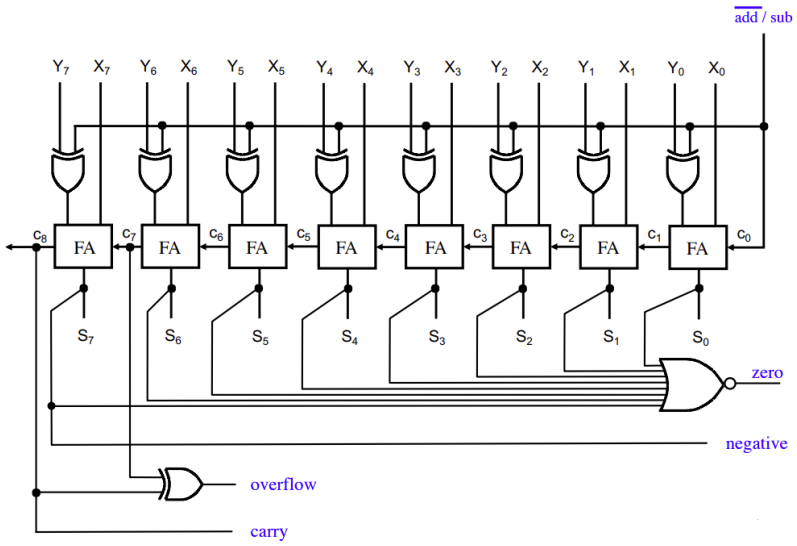


Figure 10 - ALU Addition/Subtraction Design with Flags

2.5.1.6 Program Counter

The program counter design has four main components. The first being a 6-Bit adder with one side tied to 6 bits from the Code Memory and the other side having the first bit

hardwired as one. The C_{in} bit is tied to ground. In the second adder, the left side of it takes the output of the first adder and the right side takes the lowest 6 bits from the Code Memory output. The C_{in} input is also grounded. Next, the first adder's output is multiplexed together with the second adder's output. The first one is set to output of the MUX once the control signal, C_2, is set to zero. The second adder's output moves through the MUX when C_2 is set to a one. The output of the MUX is then thrown into an 8-bit register file where the data is stored. This register is connected to both the clock and C_3. The output of the register file then goes back into the first adder's left side and the cycle begins again.

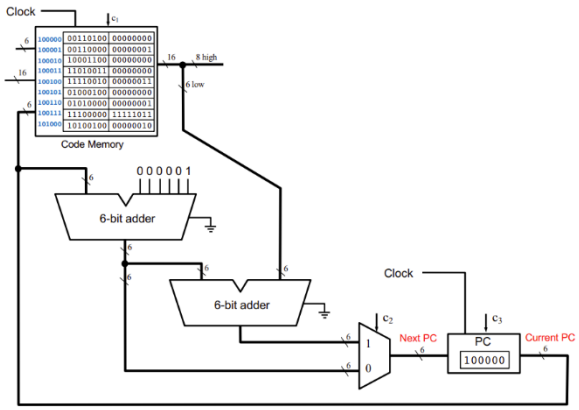


Figure 11 - Program Counter Design

2.5.1.7 Data Memory

The data memory section is a location to store intermediate information not immediately needed in computation. In addition to the 4 bytes of memory that can be stored in the register file, the data memory module offers space to store 128 bytes. The information in data memory must be loaded into a register before it can be used in other operations. The control signals associated with data memory are C_{16}, C_{17}, and C_{18}. C_{16} controls what signal is used as an input to the data memory module. C_{17} enables writing to the data memory module. C_{18} allows the output of the data memory module to be returned to the register file.

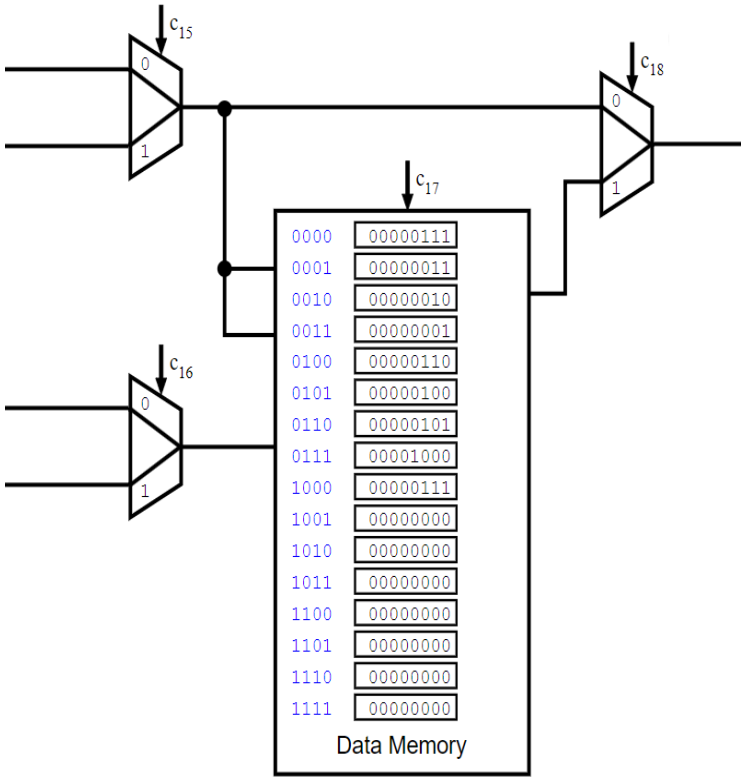


Figure 12 - Data Memory diagram from the simulator

2.5.1.8 Video Card

The video card allows for the results of computations to be displayed to the user. It does this by displaying the lower 8 bytes of data memory on 7-segment displays. The format of the output depends on if the user has the "Game Mode" option selected on the front panel. If so, each bit in a byte will correspond to a single segment of the 7-segment display. Otherwise, the lower 4-bits are converted into hexadecimal and displayed.

To be practically implementable in hardware, the video card does not directly display the contents of data memory. Instead, it acts as a memory mapped I/O device that responds to the first 8 bytes of memory. When a write memory occurs in one of those addresses, the information is stored both in the video card and the data memory module. Due to this design decision, the video card will only update the contents of a cell when a write operation occurs.

2.5.1.9 Control

The control lines throughout the computer are defined by the control logic seen in the picture below. There are two main sections to the initial design of the i281, the decoder and the control logic table. The decoder takes the 16-bit instruction line as an input and outputs logic for which operation is going to be completed by the CPU. The control box sets the corresponding control lines, for an operation, that are distributed to the rest of the CPU.

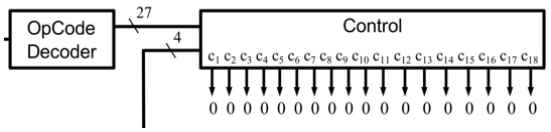


Figure 13 - Control Logic Blocks

The OpCode decoder, shown below, ensures that only one operation is active at a time. It also shows the breakdown of each bit of the instruction line and how it is used in the decoder. The most significant four bits are always used to define the operation, as seen with the 4-to-16 decoder. In o-16 decoder. In addition, bits 9 and 8 are sometimes used to decode the operation as seen in the three decoders to the right.

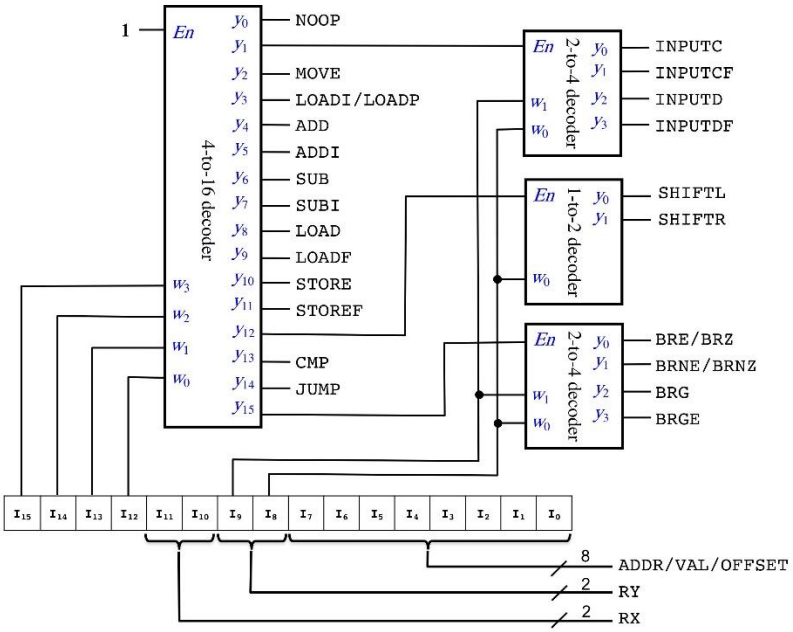


Figure 14 - OpCode Decoder Logic

The 23 operations bits and bits 11, 10, 9, and 8 are passed from the Opcode Decoder to the control box to set each control line. The values of each control line is shown in the figure below for each operation taking place. The way this implantation works in design 0 is using gates to make a Boolean function.

	IMEM_WRITE_ENABLE	PROGRAM_COUNTER_MUX	PROGRAM_COUNTER_WRITE_ENABLE	REGISTERS_PORT0_SELECT1	REGISTERS_PORT0_SELECT0	REGISTERS_PORT1_SELECT1	REGISTERS_PORT1_SELECT0	REGISTERS_WRITE_SELECT1	REGISTERS_WRITE_SELECT0	REGISTERS_WRITE_ENABLE	ALU_SOURCE_MUX	ALU_SELECT1	ALU_SELECT0	FLAGS_WRITE_ENABLE	ALU_RESUT_MUX	DMEM_INPUT_MUX	DMEM_WRITE_ENABLE	REG_WRITEBACK_MUX
NOOP			1															
INPUTC	1		1												1			
INPUTCF	1		1	X1	X0						1	1						
INPUTD			1												1	1	1	
INPUTDF			1	X1	X0						1	1				1	1	
MOVE			1	Y1	Y0			X1	X0	1	1	1						
LOADI/LOADP			1					X1	X0	1					1			
ADD			1	X1	X0	Y1	Y0	X1	X0	1		1		1				
ADDI			1	X1	X0			X1	X0	1	1	1		1				
SUB			1	X1	X0	Y1	Y0	X1	X0	1		1	1	1				
SUBI			1	X1	X0			X1	X0	1	1	1	1	1				
LOAD			1					X1	X0	1					1			1
LOADF			1	Y1	Y0			X1	X0	1	1	1						1
STORE			1			X1	X0								1		1	
STOREF			1	Y1	Y0	X1	X0				1	1					1	
SHIFTL			1	X1	X0			X1	X0	1				1				
SHIFTR			1	X1	X0			X1	X0	1			1	1				
CMP			1	X1	X0	Y1	Y0					1	1	1				
JUMP		1	1															
BRE/BRZ		B1	1															
BRNE/BRNZ		B2	1															
BRG		B3	1															
BRGE		B4	1															

Figure 15 - Control Lines Table

2.5.2 Design 1 — Minimum Viable Processor

Leading into designing a physical hardware implementation, several changes needed to be made to the original design. Additionally, we developed a few standards for the project. These standards will not be discussed in full here and instead can be viewed in Appendix C (Standards).

2.5.2.1 BIOS

The BIOS, which is stored and runs from the ROM, will serve the function of clearing memory and setting up the RAM chips for the user before running the main program. The BIOS will need to run faster than the main code, as the length of BIOS' execution would make waiting for it too long for a normal user. The BIOS then must decide which program to fill the RAM with, determined from the user's input. After filling RAM and Data memory, the CPU will be read for the program to run.

2.5.2.2 Code Memory

To avoid issues with needing to modify or load a program in the same chip as the BIOS, a ROM and RAM chip will be distinguished. In addition to containing ROM and RAM, which holds the BIOS, Code Memory handles all interactions with program loading and execution. Instructions for running the user's program will be stored and executed from the RAM. This

will be filled in with the necessary instructions during boot. Either the RAM will be filled from a storage chip on the device storing sample programs or filled using the switches manually (via BIOS loader).

The size of both ROM and RAM has been extended compared to the FPGA design. Since the code memory will not be implemented as a massive register file, the independent ROM and RAM chips will allow storage far beyond the processor's general capabilities.

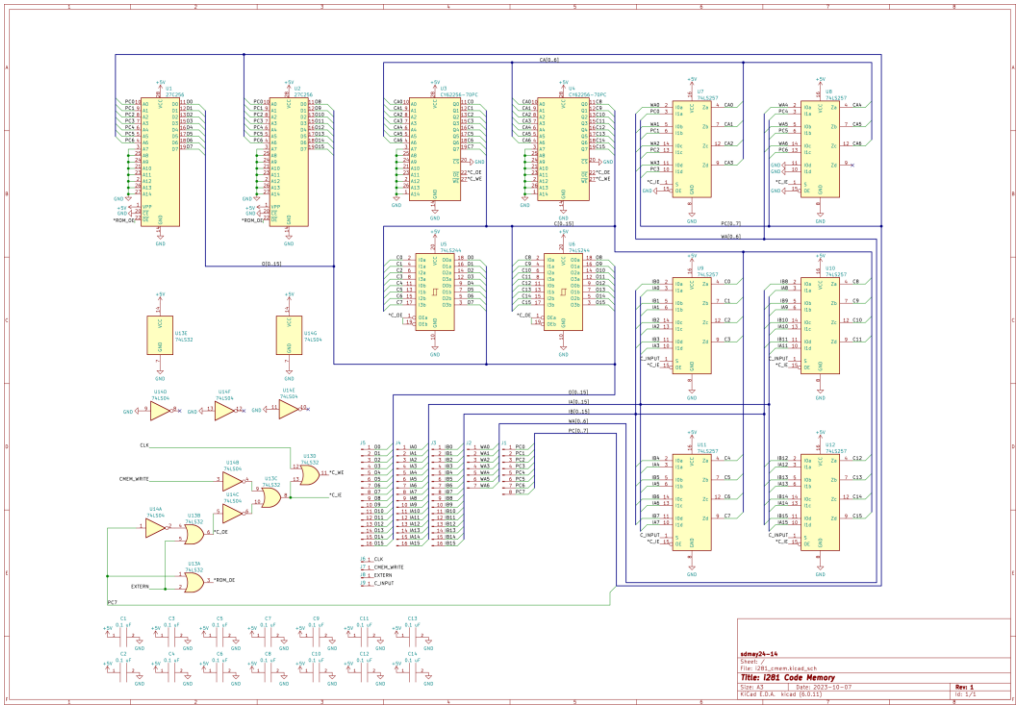


Figure 16 - Code Memory Schematic

2.5.2.3 Register Files

The register file will be implemented using four 8-bit register chips that will be multiplexed between two different sets of 4-1 multiplexer chips. Additional LEDs will be included between registers and output to visualize what is stored and what has been produced as an output.

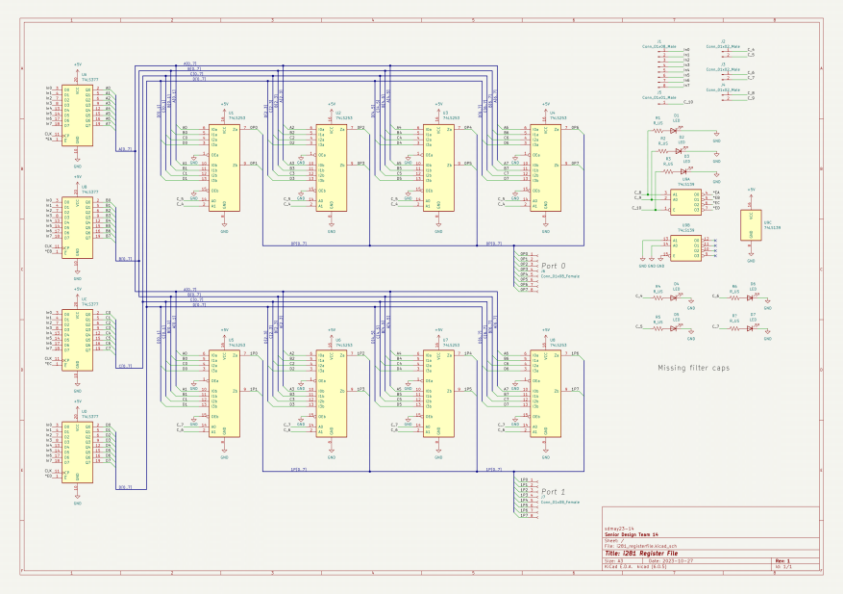


Figure 17 - Register File Schematic

2.5.2.4 Arithmetic Logic Unit

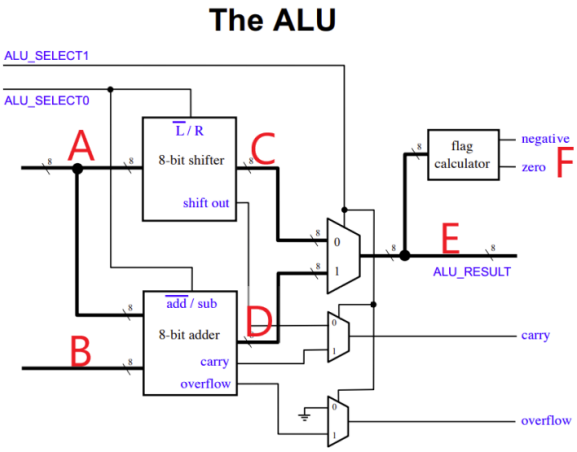


Figure 18 - ALU Ports Labeled on Diagram

Design 1 of the ALU includes the first design of the ALU using 74 series chips and LEDs. The first image shown below is of the 8-bit shifter circuit. It uses three SN74HCT257N chips which are each four 2-input multiplexers. The first two on the left in the picture take the input from Port A. The third multiplexer is where we determine where the shift out bit is determined. The output of this circuit is Port C. We also have three filter capacitors to filter out noise.

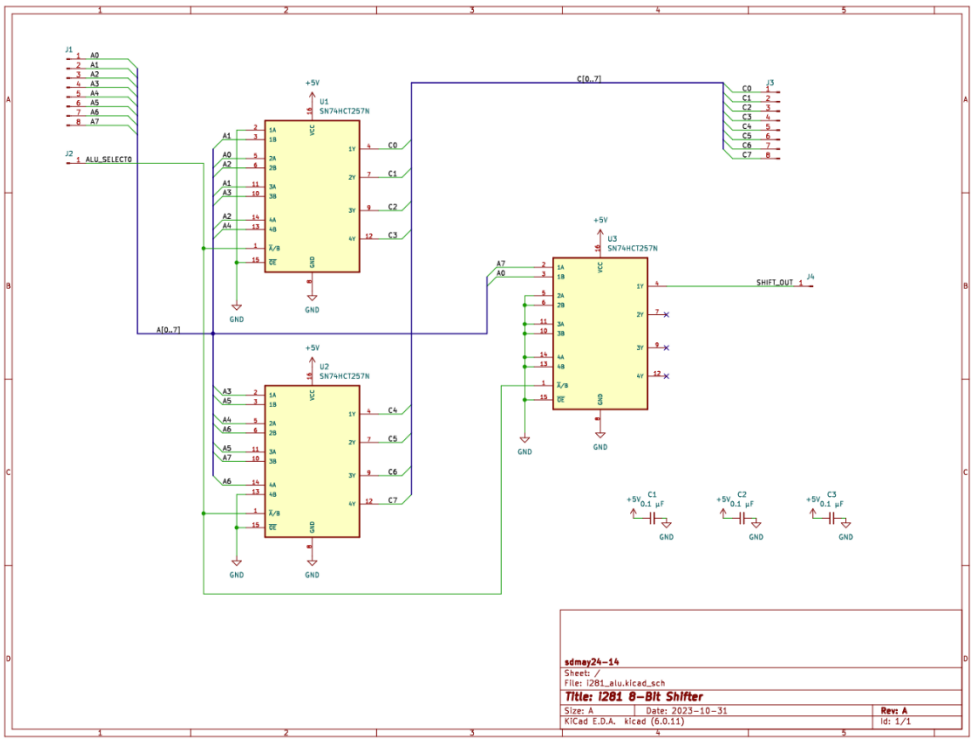


Figure 19 - Initial Schematic Design of ALU 8-Bit Shifter

The Adder/Subtractor circuit design is created using three CD74HCT86E four 2-input XOR chips and two CD74HCT283E 4-bit Full Adder chips. This circuit inputs the ALU_SELECT0 control line along with Port A and B. From this circuit we output Port D, an overflow bit, and a carry bit. The logic is the same as it was in Design 0, just with chips instead of conceptual design. We also have 5 filter capacitors tied to both +5V and GND to filter out noise.

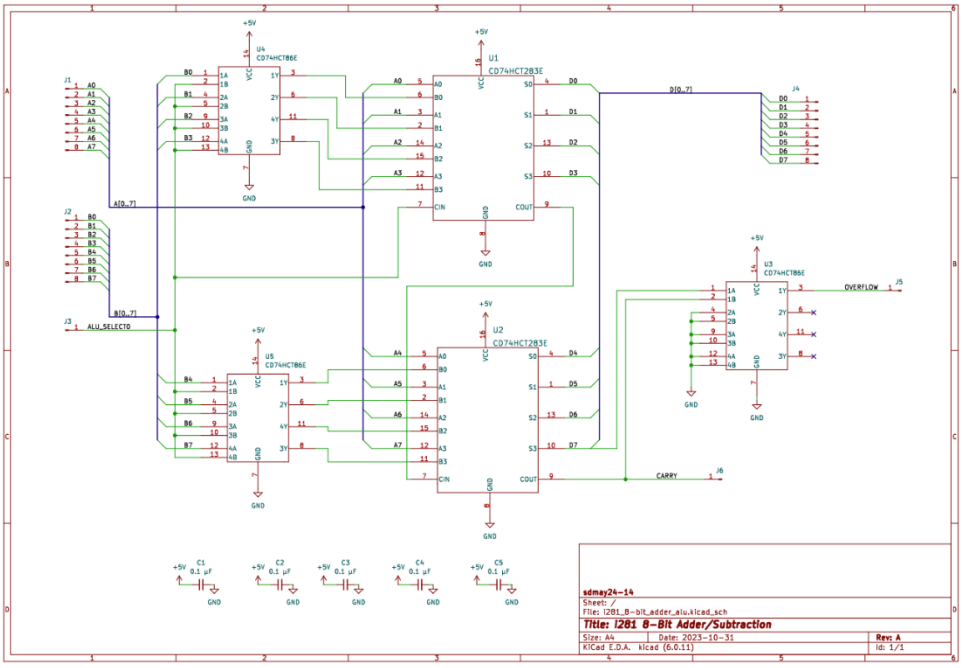


Figure 20 - Initial Schematic Design of 8-Bit Addition/Subtraction Component Schematic

The last schematic includes the 8-bit 2-to-1 multiplexers (created out of two SN74HCT257N chips) that Port C and D feed into and output Port E. Port E is also visualized with eight LEDs accompanied by eight 330Ω resistors. Port E is also inputted into CD4078BE 8-bit NOR chip. The output of the NOR is wired into the CD74HCT377E register file for the flags. We also have another SN74HCT257N 4-bit 2-to-1 multiplexer circuit that takes the shift bit from the shifter, the carry bit from the adder, and the overflow bit from the adder. The output of the multiplexer goes straight into the register file for the flags. Lastly, the last bit in the Port E bus is also tied to the flag register as the negative flag. All the flags are visualized using four LEDs accompanied by four 330Ω resistors. The flag register output is labeled Port F. This design includes five filter capacitors to reduce noise. Notice the order of the flags on Port F as this will be changed in Design 2.

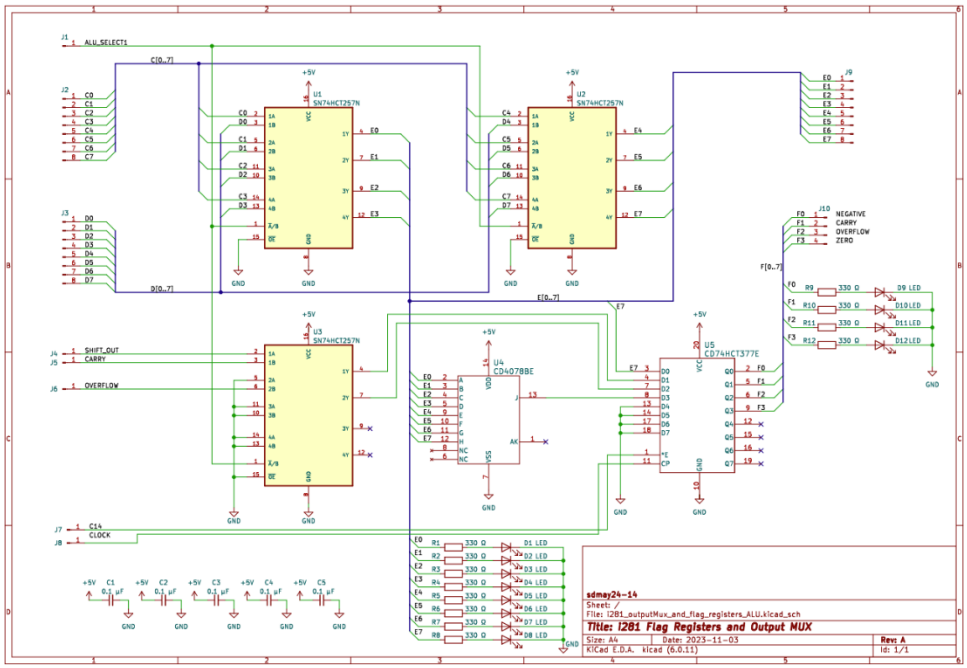


Figure 21 - ALU Flag Registers and Output MUX Schematic

2.5.2.5 Program Counter

The program counter was expanded to 8-bits instead of the original 6-bits as seen in the i281 simulator. A major factor in these changes is the limited physical parts sold; the adder and mux chips are 4 bits each. This allows us to expand to 8-bits without changing our design and no real downside. Another benefit of increasing the bit size of the program counter is we get more space to store and run programs from 2^6, 64 to 2^8, 256 lines. The design greatly benefits from this change without making big sacrifices elsewhere.

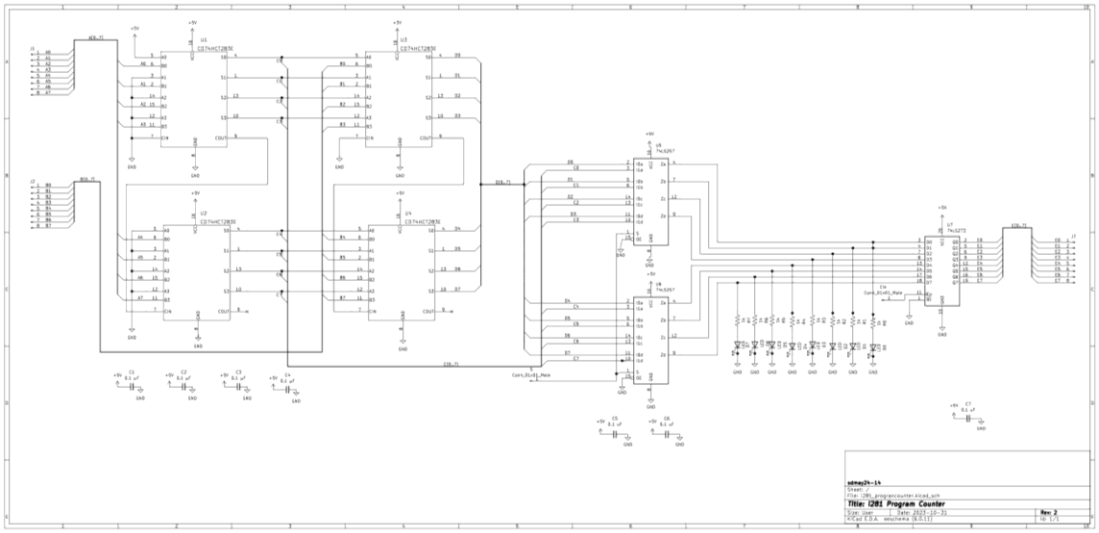


Figure 22 - Program Counter Schematic

2.5.2.6 Control Table

The control logic was changed to use EPROM to convert between the current operation and the control lines. The schematic for the design can be seen below. This simplifies the number of components in the design well, giving the same functionality. The EPROM is programmable, allowing the ability to update the control logic later.

Since the CPU displays each control line's state at two locations, where it is generated and used, we are using a buffer chip. Each chip should only drive one LED, so this buffer is required to ensure proper voltage levels in the control lines.

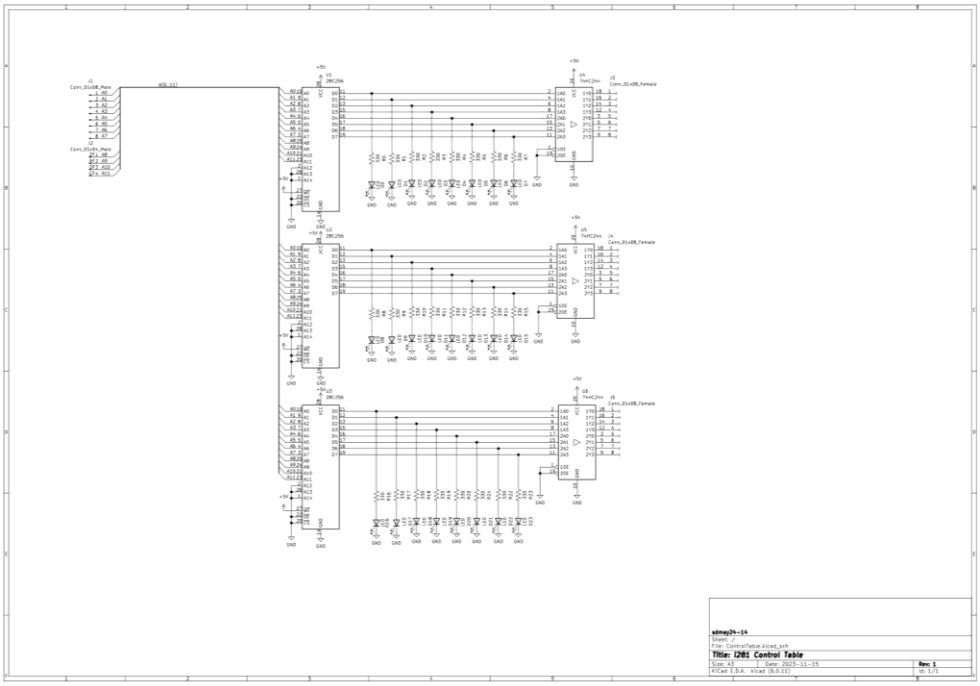


Figure 23 - Control table schematic

2.5.2.7 Video Card

The Video Card was designed to look the same as the simulator and have two different functions: normal and game mode. There are not too many differences between the two functions, but game mode enables an extra bit and configures the EEPROM slightly differently. We used one 8-bit register file for each seven-segment display so the clock could moderate the output and the displays wouldn't change too quickly. We also added two 100Ω connected in parallel with the common cathode of the display and ground. This allowed us to achieve a total resistance of 50Ω. We had issues with resistance values higher than this as with every segment of the display that turns on, the brightness dims. The value of 50Ω allowed us to not burn out the displays and get the most brightness on them. This issue was corrected for the PCB design. This design also includes a 27C256 EEPROM that takes input from the data memory and converts that data into letters, numbers, and symbols and outputs the data to the register files. The Video Card also uses a 3-to-8 Decoder to activate the enable pin of each of the eight registers. This function allows each register to accept the data from the EEPROM at the proper time. Lastly, there is a 8-bit NOR chip that enables and disables the decoder depending on input from the DMEM. The picture below shows the schematic for the design. The picture below shows the physical implementation of it on the breadboards. This design was attached directly to the DMEM, so they acted as a single island in the breadboard design.

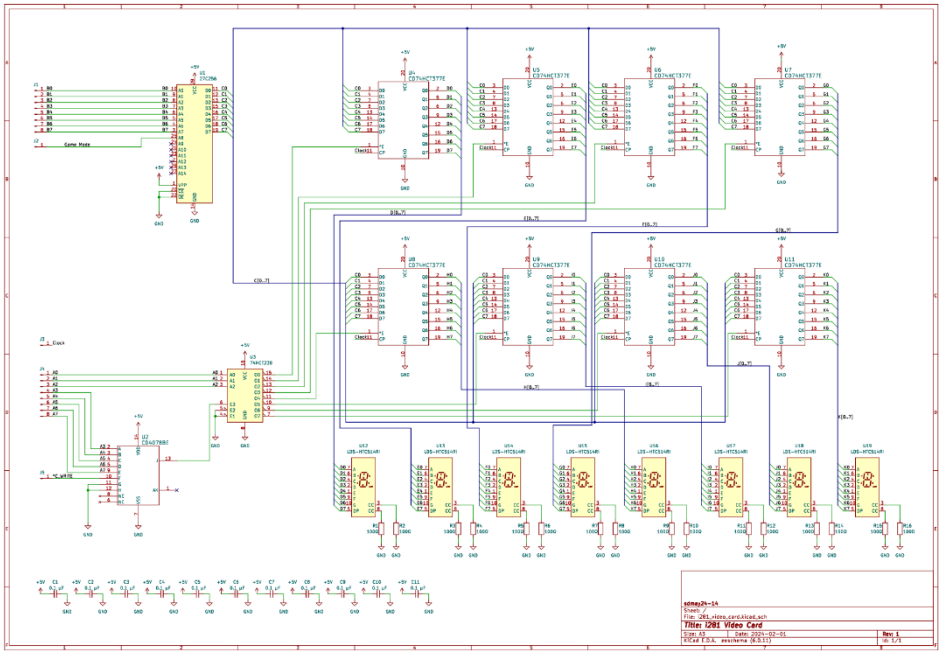


Figure 24 - Video Card Schematic

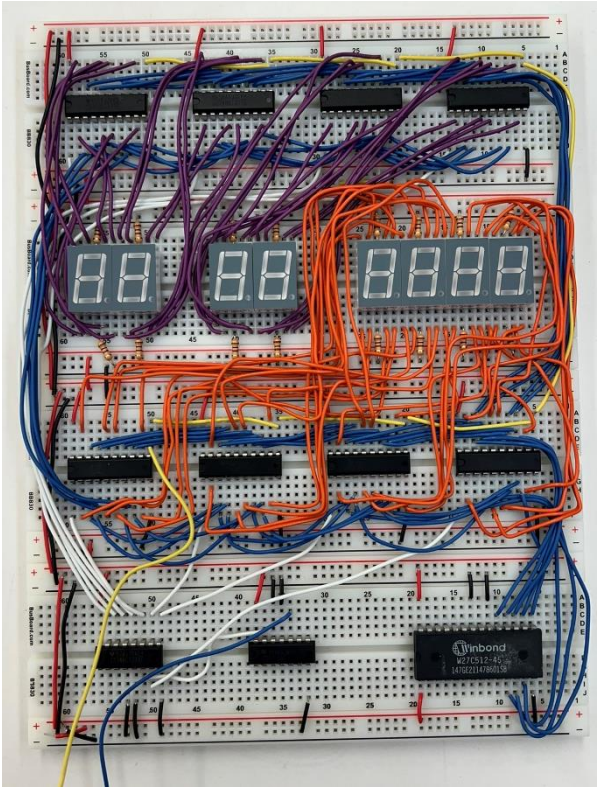


Figure 25 - Video Card Breadboard Implementation

2.5.3 Design 2 — Improved Breadboard

Certain components were iterated upon in this second design.

2.5.3.1 Code Memory

A visualization panel was added in extension to the component to showcase the current instruction and program address. This module is known as the "Debug Module" and attaches directly to the Code Memory module. In addition to displaying the current instruction, the module can interface with the front panel to provide debugging facilities to the user. It accomplishes this by overriding the CMEM instruction output with hardcoded instructions. This hardware was used to implement the "Examine" and "Deposit" front panel functions.

2.5.3.2 Arithmetic Logic Unit

The image below is the Design 2 iteration of the ALU schematic. It is the combination of all three of the schematics seen in Design 1. There are two major changes to the schematic. The first one is the order of the output flags. The second is we made an error in calculating the overflow bit, so we added an extra 4-bit Adder chip (CD74HCT283E). In addition to the two changes, the ALU has been built on breadboards and works as expected. Another smaller change to the schematic was that the control signals are now names the same as they will be called from the control table.

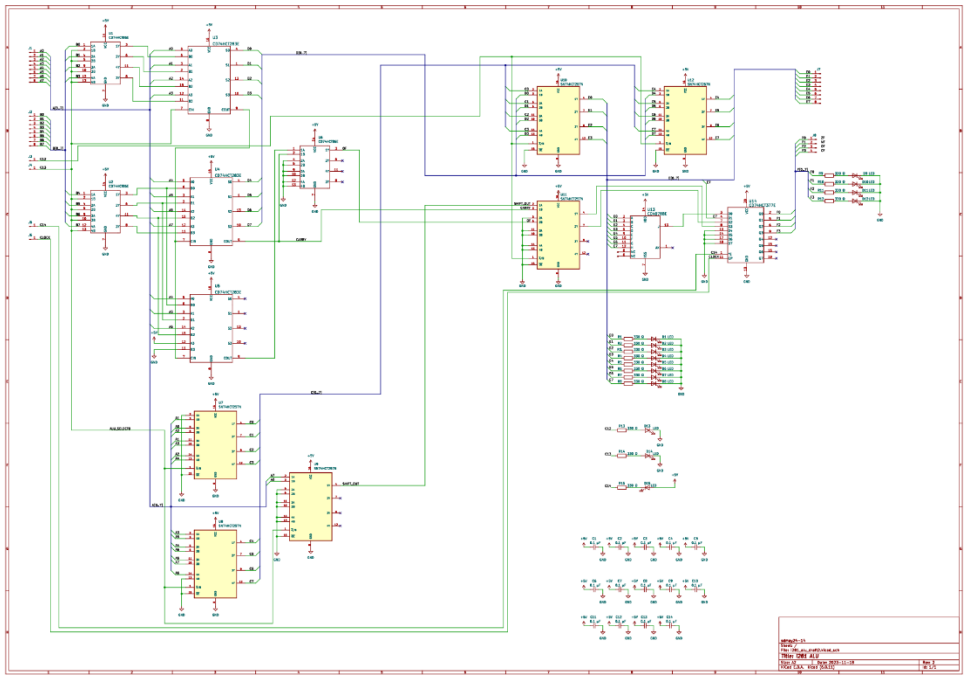


Figure 26 - ALU Subcomponents Final Schematic

The output flags were rearranged to match the same layout as the simulator version of the i281 CPU current supports. Bit 0 is now designated as the zero flag. Bit 1 is now designated as the negative flag. Bit 2 did not change and is still the overflow flag. Bit 3 is now designated as the carry flag.

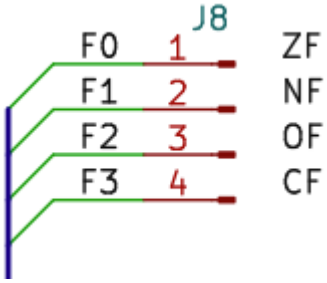


Figure 27 - Output Flags on Schematic

In Design 1, the mistake we made for the overflow bit was we assumed that S_7 was the bit that would go into the XOR to create the overflow bit. This was incorrect as C_7 is not S_7. C_7 is also a bit internal to CD74HCT283E. There are multiple ways to imitate that bit. We had the whole design already built on the breadboards and there wasn't much space to implement a large multi-chip solution. Instead, we realized we could use another adder chip, except this one would take the same inputs as the second adder chip from Design 1. The only difference is that we want to preserve the C_7 so that it will output as the new C_8 or C_{out} on the chip. In order to do this, we just needed to use one of X_7 or Y_7 as a logic high and the other as a logic

low. This allows that carry bit to flow through. We are unsure if it is the most power efficient method of implementation, but for the functionality, it works.

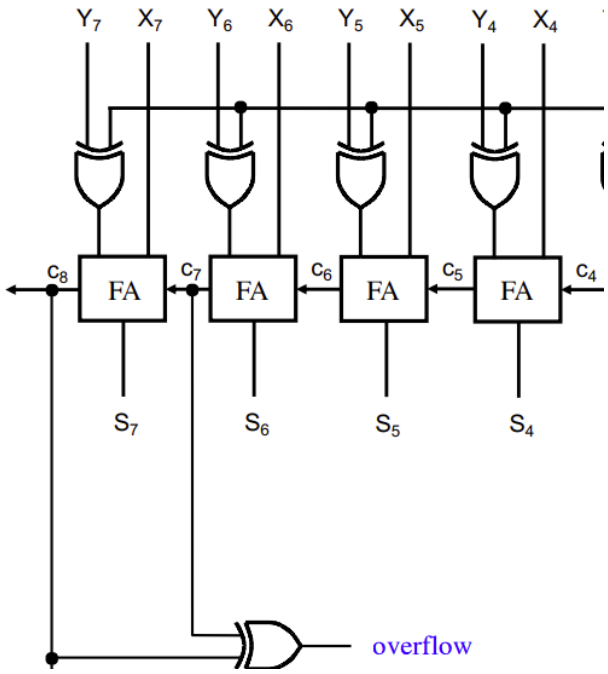


Figure 28 - Adder Circuit Zoomed in on Last Few Bits

The image below is the breadboard version of the design. Visually, we can see that there are a lot of wires going all over the place. This design has so many components and it was very difficult to layout. From the image, we can see Port A, B, E, and F. We can also see the flags, control lines, and ALU output. The LEDs have the smallest bit on the right and greatest bit on the left.

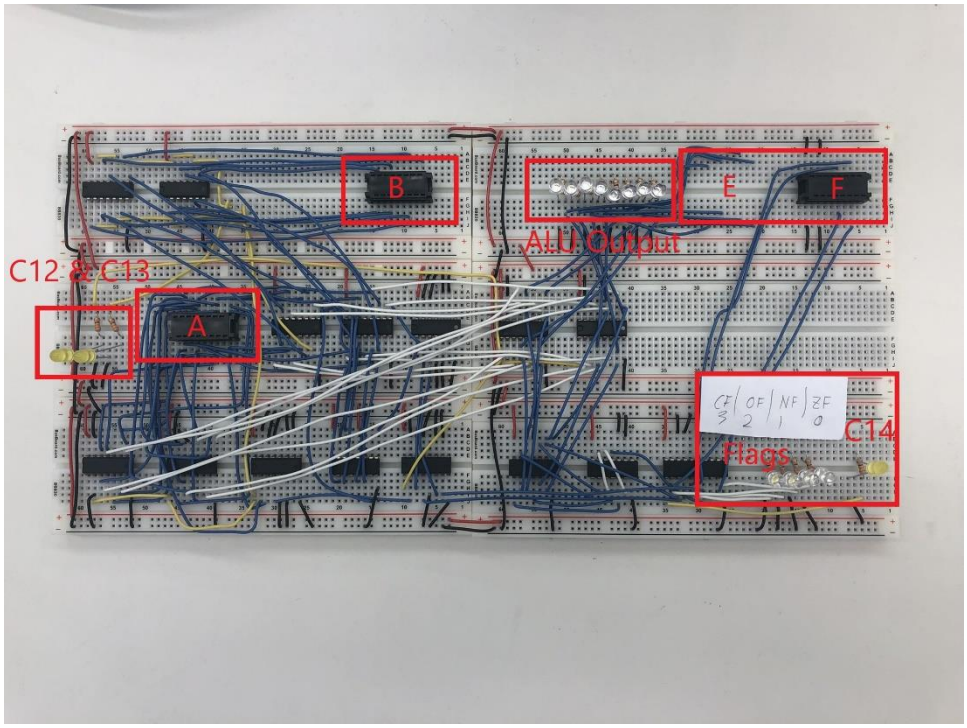


Figure 29 - Implementation of the ALU on Breadboards

2.5.3.3 Writeback Module

An additional module was added to efficiently write to RAM with the programs inputted from data memory. This slot in between the bus that carries the switches to the code memory and adds additional function with the help of a control signal C3.

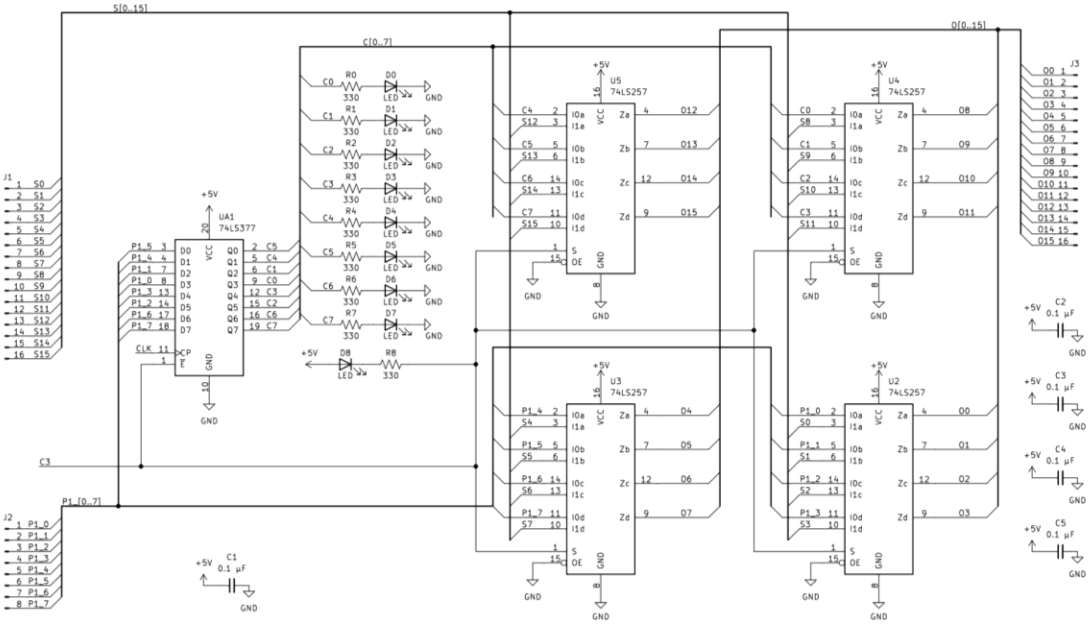


Figure 30 - Writeback Module Schematic

The control signal used, C3, is active low in this usage. When C3 is logic high the writeback module is in the normal operations and outputs the switches to the code memory. When C3 is active or logic low the input from port one of the registers is saved in a register, saving the high eight bits of the instruction and gets ready to output the entire instruction line loaded from data memory to the code memory. Since the input bus from the register file is only eight bits, the instruction line has to be split into two parts. The high eight bits are saved in a register, and the lower eight bits are passed from the input bus.

This allows for compact flash modules in the data memory to be utilized to save programs to be loaded into the RAM chips. The write back module still allows for the switches to be used in their original functionally as well. An user can input programs manually and us instructions codes that expect user input.

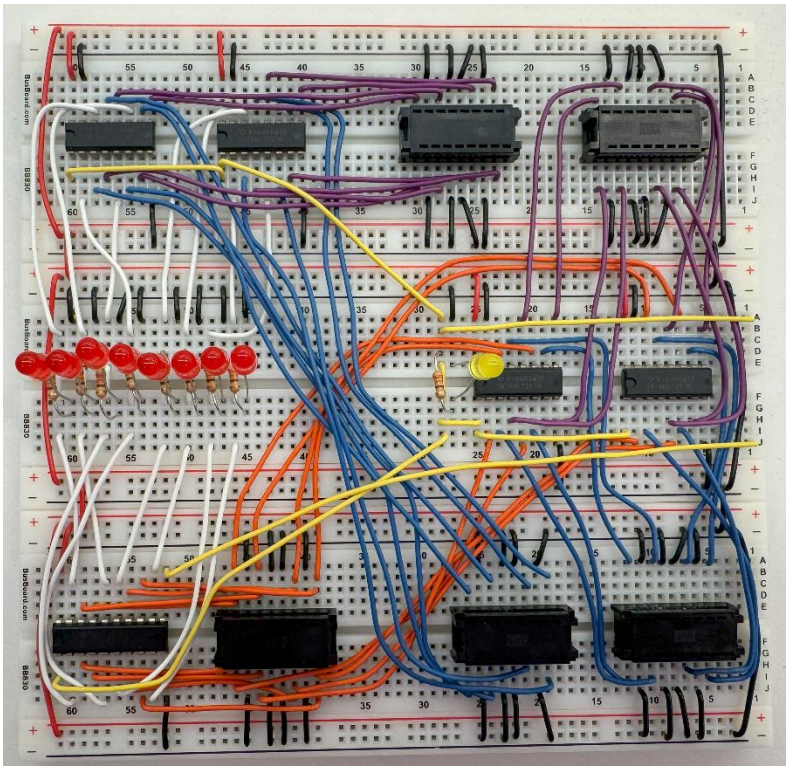


Figure 31 – Writeback Module Breadboard Implementation

2.5.4 Design 3 — PCB Revision A

2.5.4.1 Mainboard

On the breadboard prototype, the individual modules were interconnected using a large amount of ribbon cables. Based on the application requirements, and personal experiences using building the prototype, we decided that using ribbon cables was not practical for the PCB machine. After several design meetings, we decided to condense all the ribbon cable

interconnects onto a single PCB. Each module connects on top of the mainboard PCB using 2.54 mm pin headers, and buses are routed on the PCB itself.

To comply with our original design requirements, we added breakout connectors on the bottom middle of the mainboard PCB. This allowed for existing PCB modules to be removed and be replaced with breadboard versions. With this feature, students would be able to make new versions of the PCBs on breadboards, and then connect them to the rest of the system using these breakout connections.

2.5.4.2 Code Memory

To prepare Code Memory for the PCB implementation, the existing Code Memory module from Design 1 was combined with the Writeback module from Design 2. This was done to simplify the amount of board needed to produce. These modules are also heavily connected in functionality. Overall, this reduced the number of integrated circuits needed to create the processor by two.

It was decided that ZIF (Zero Insertion Force) sockets would be used for the BIOS ROM chips. This was done to allow students to change the programming of the BIOS easier without risking damage or mechanical wear to the code memory IC sockets.

Minor changes were made to the schematic after combining the two modules. The power bus was included in the schematic, as well as giving each input and output bus a pin header to connect to the main board. The schematic is shown in Figure 32 and the PCB model is shown in Figure 33.

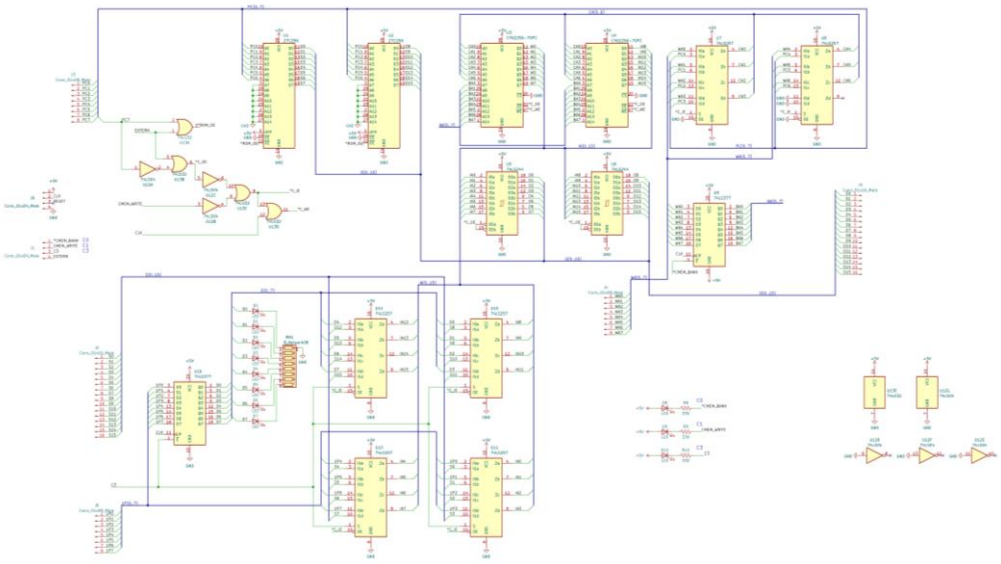


Figure 32 - Code Memory PCB Schematic (Rev A)

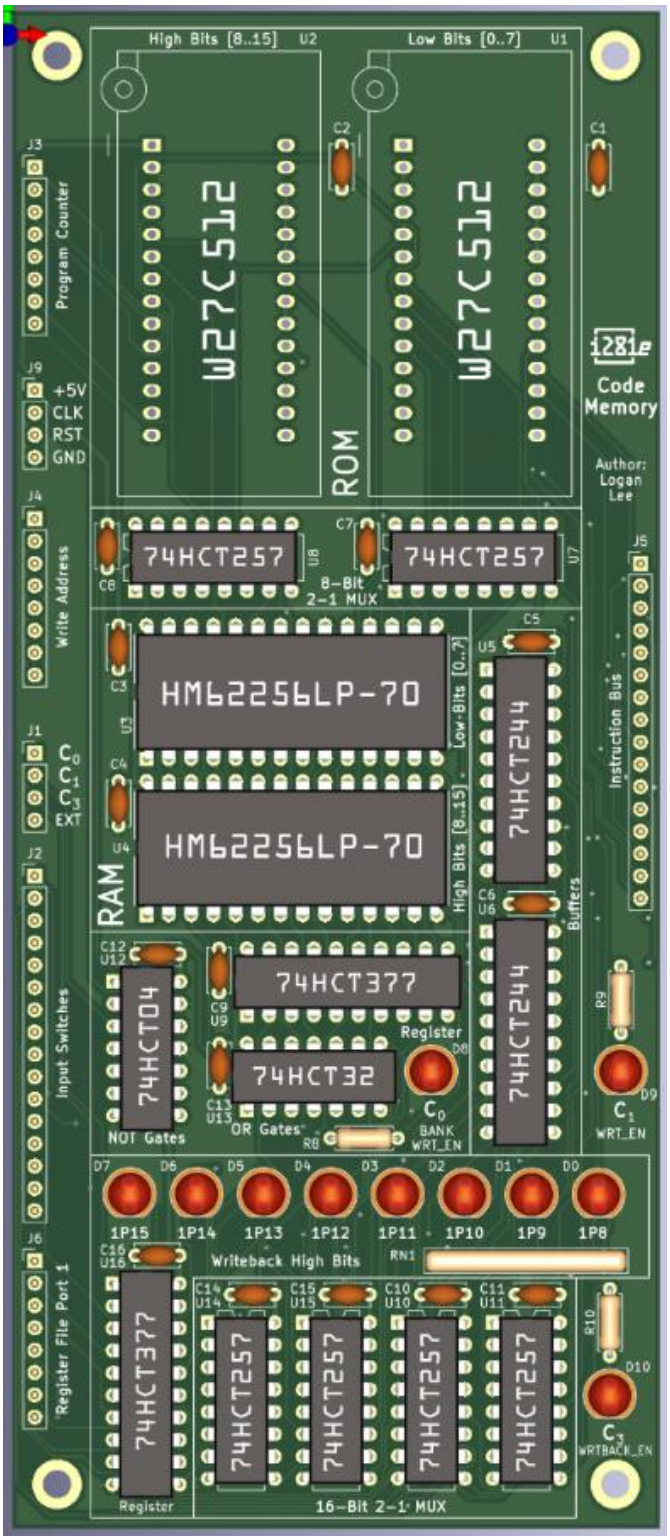


Figure 33 - Code Memory PCB Model (Rev A)

2.5.4.3 Register File

The circuitry of the register file was not changed significantly from the prototype module to the PCB implementation. Unlike other modules, the register file was not combined with any other module and remained as implemented in the original design. Due to space constraints, the physical layout of the register file was changed from a horizontal layout on the breadboard machine to a vertical layout. This allowed the register file to have a smaller footprint while maintaining all the existing visualization LEDs.

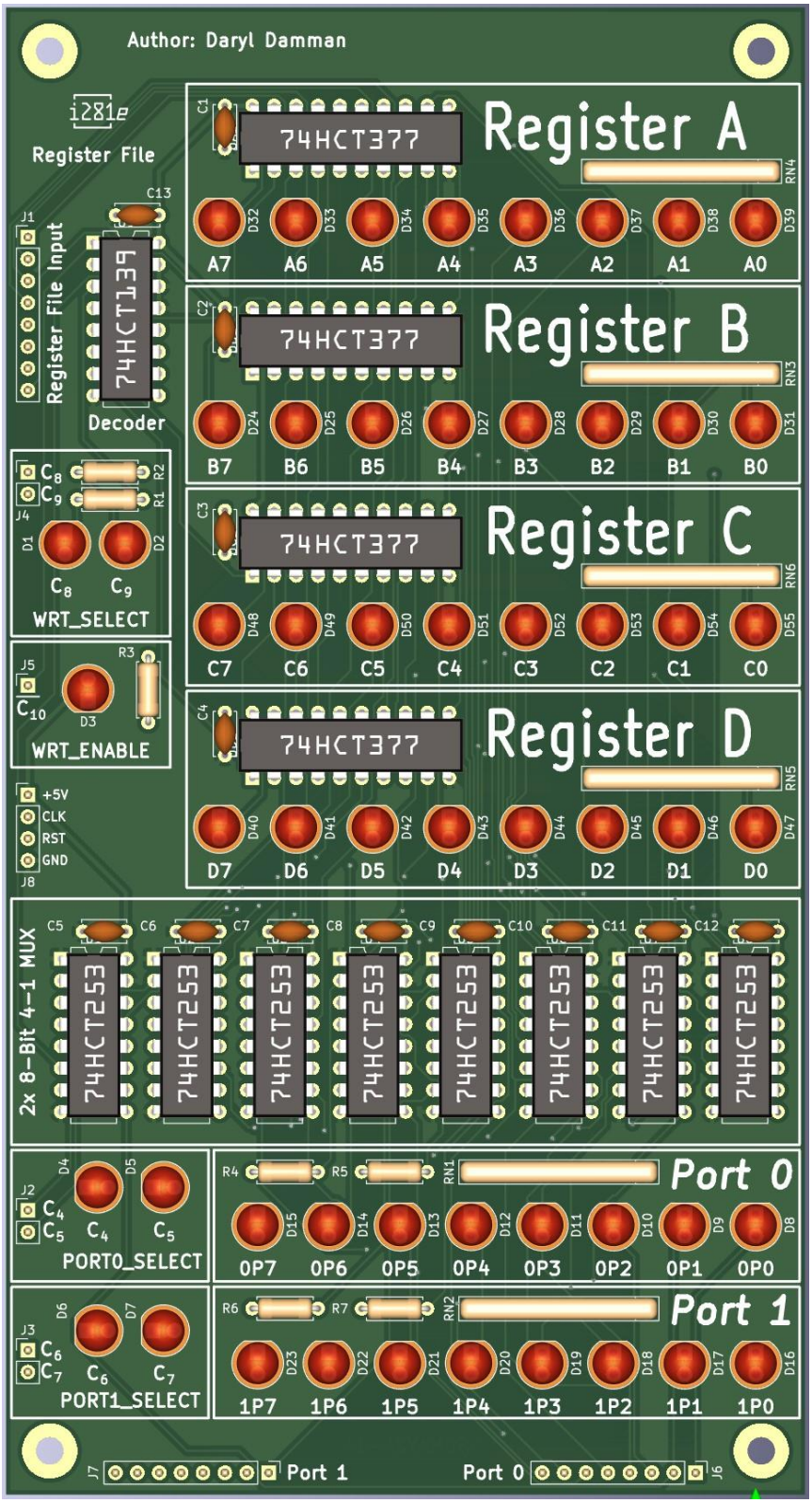


Figure 34 - Register File PCB (Rev A)

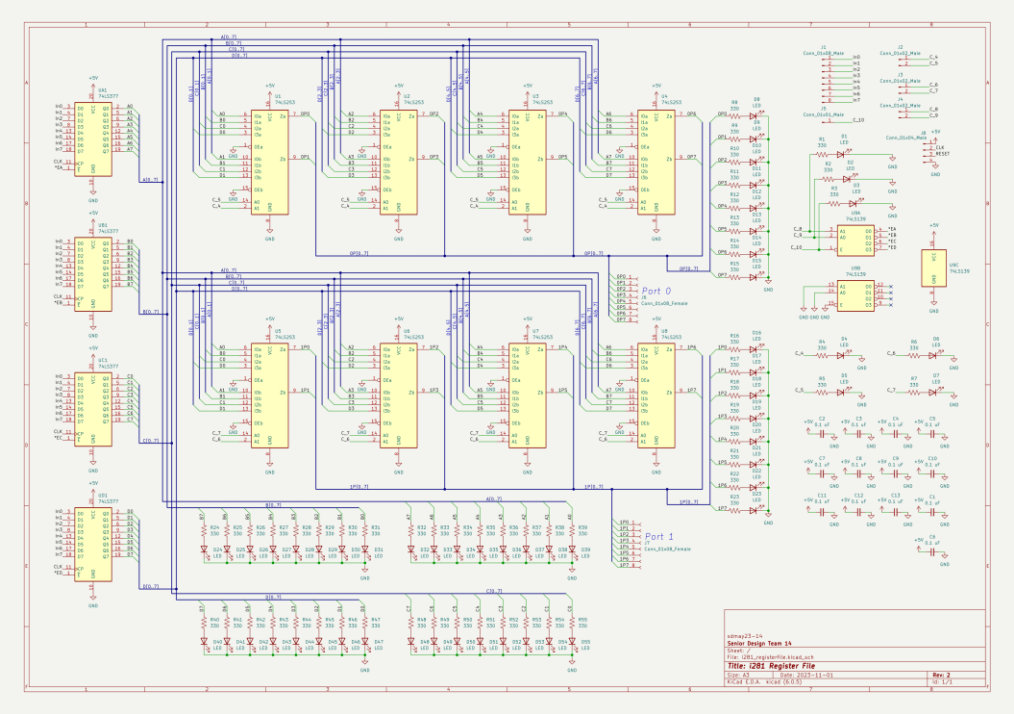


Figure 35 - Register File PCB Schematic (Rev A)

2.5.4.4 Arithmetic Logic Unit

In this iteration of the ALU, we split it up into two different boards. Both were to be used in the same slot on the mainboard. For the first version had the legacy functions as the previous iterations where it contained four arithmetic functions: shift left, shift right, addition, and subtraction. The second version of the design deemed ALU NOR removed the shift left function and added a NOR function. Figure 36 shows this updated version. This change required two additional chips (Figure 37). Figure 38 shows the schematic of the original ALU, but slightly edited for the PCB implementation. Those chances were we moved added a 4-bit input pin and consolidated the control line into one 3-bit input pins.

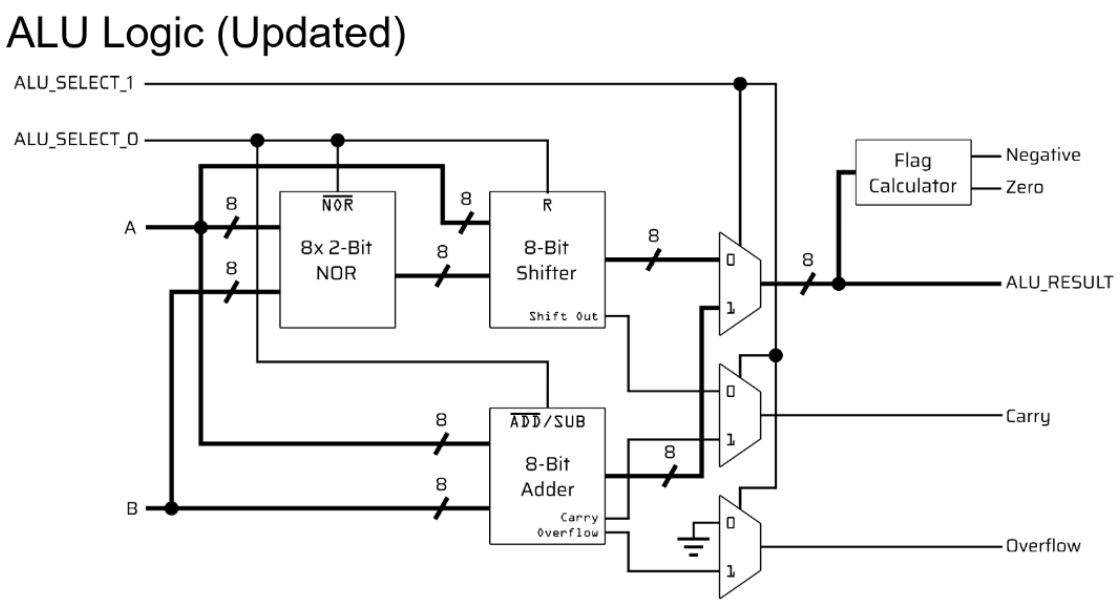


Figure 36 - ALU NOR Logic

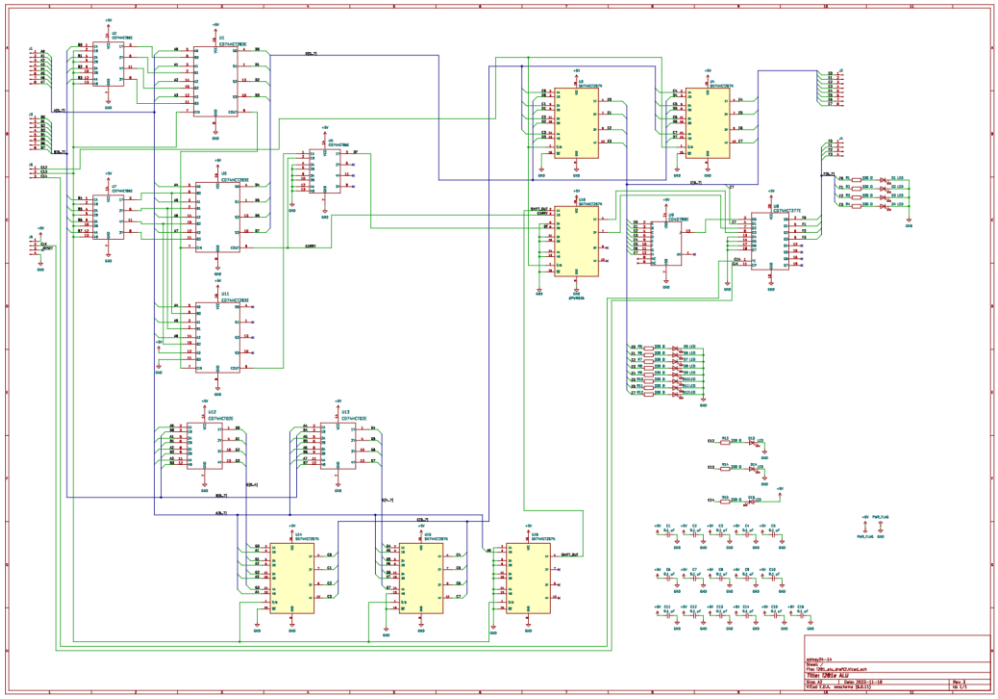


Figure 37 - ALU NOR PCB Schematic

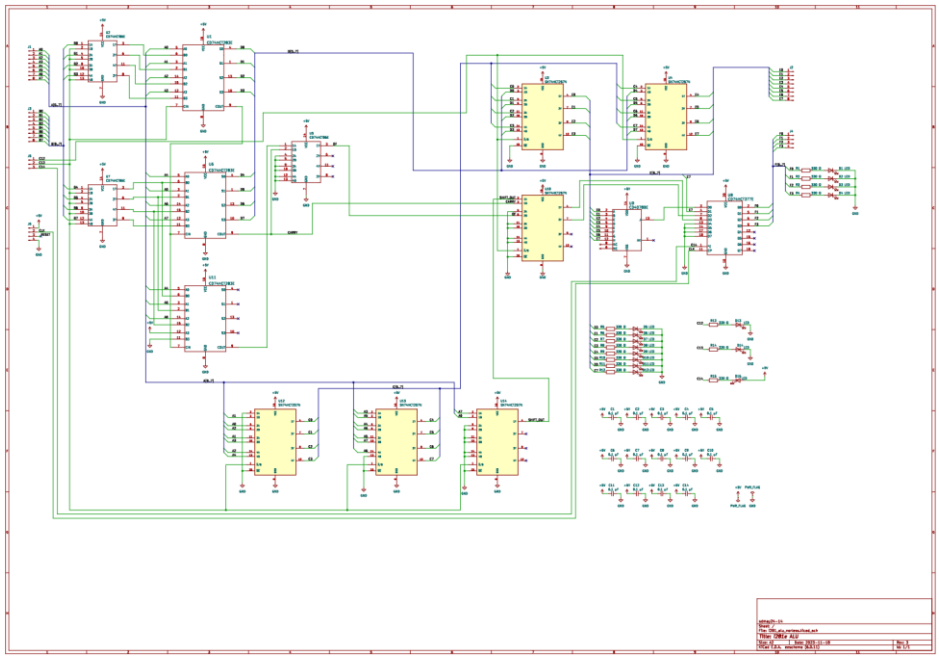


Figure 38 - ALU PCB Schematic

From the schematic, we were able to create the PCB layouts. In order to make both the ALU and ALU NOR versions easy to identify, they have the exact same layout except for the two

NOR Chips U12 & U13 and their accompanying capacitors. On the ALU version, it just has that section empty. Figure 39 shows the ALU NOR PCB and Figure 40 shows the ALU Version.

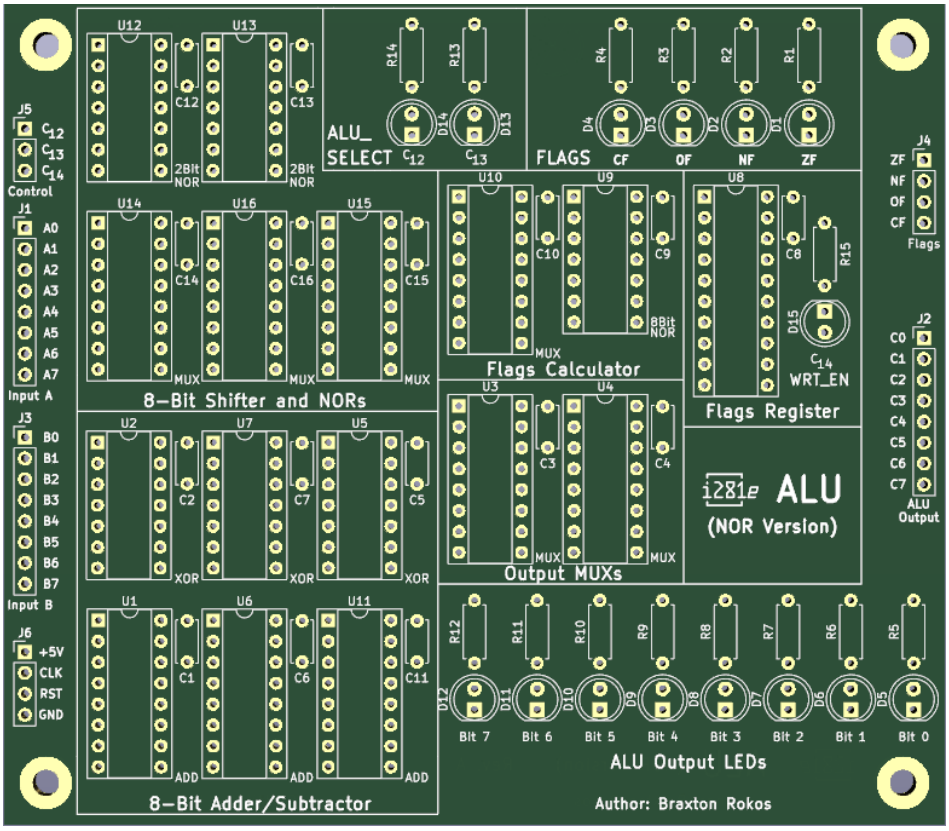


Figure 39 - ALU NOR PCB

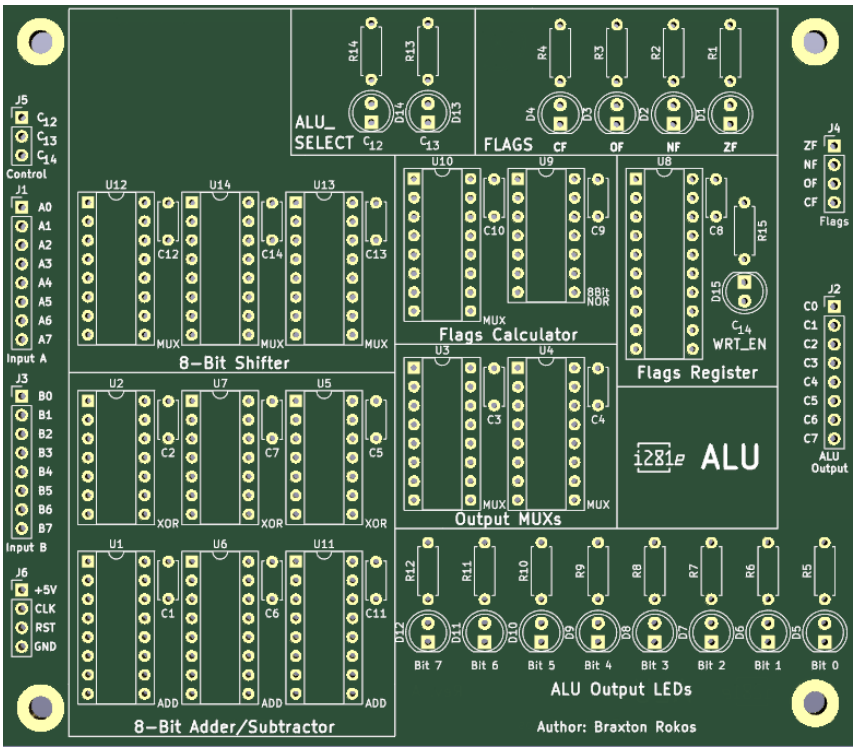


Figure 40 - ALU PCB

2.5.4.5 Program Counter

When adapting the Program counter from breadboard to PCB, not much changed with the schematic. The power bus was included and the resistors for LEDs were replaced with a resistor pack to reduce footprint. After adding layouts to each part and completing in routing of the traces the schematic from Figure 41 was implemented into the PCB shown in Figure 42.

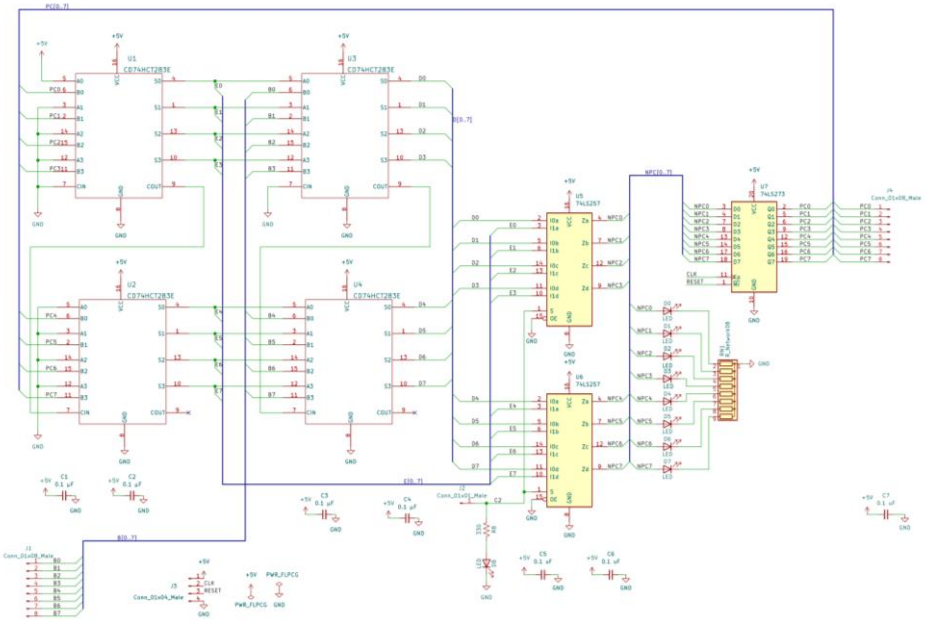


Figure 41 - Program Counter PCB Schematic (Rev A)

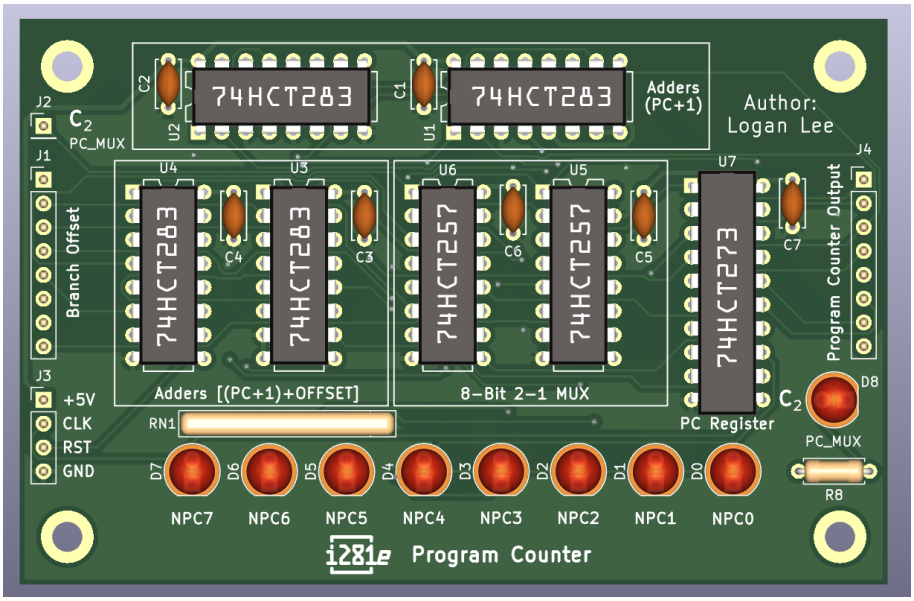


Figure 42 - Program Counter PCB Implementation (Rev A)

2.5.4.6 Control Table

The circuitry behind the control table remained essentially unchanged between the improved breadboard prototype and the Rev. A PCB implementation. The only major circuitry change involved switching out the individual resistors for resistor networks to save space. Like the code memory module, it was decided that ZIF socketed would be used for each ROM chip, as that is a part that will see frequent insertion and removal.

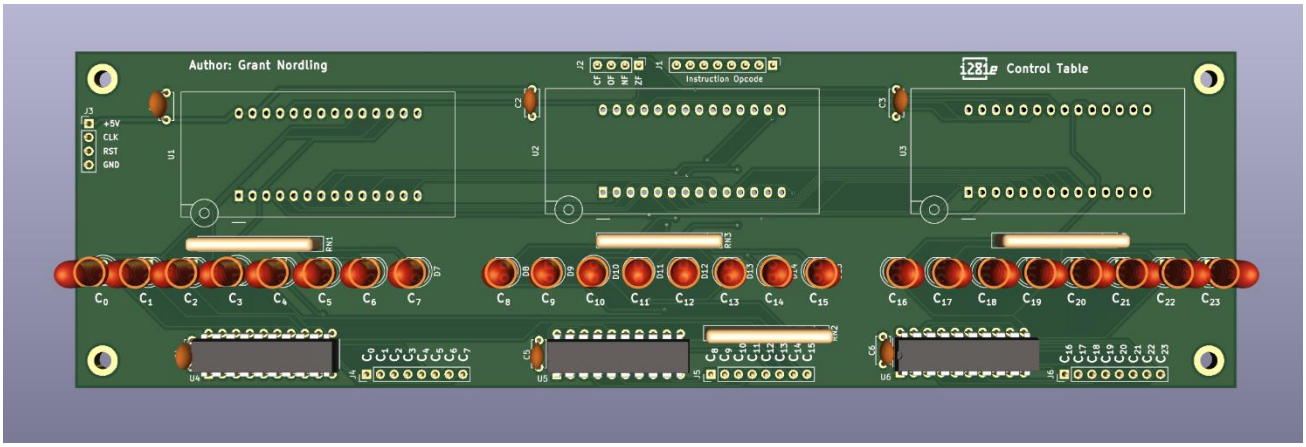


Figure 43 - Control Table PCB (Rev A)

For the first revision of the control table, we decided against additional inputs for the address pins used on the control ROMs. Adding headers for them would have required extra pull-up circuitry on the control table, as well as requiring extra connections and wiring on the mainboard. Omitting this potential feature reduced the number of points of failure on the first revision.

2.5.4.7 Video Card & Data Memory

On the PCB version of the data memory module, it was decided that the video card and data memory sections of the original breadboard prototype would be combined into a single board. This was done because the two modules share several internal signals that are not used anywhere else in the processor. The circuitry for the video card could be piggybacked off the I/O signals of the data memory module, therefore reducing the number of total connections to the mainboard.

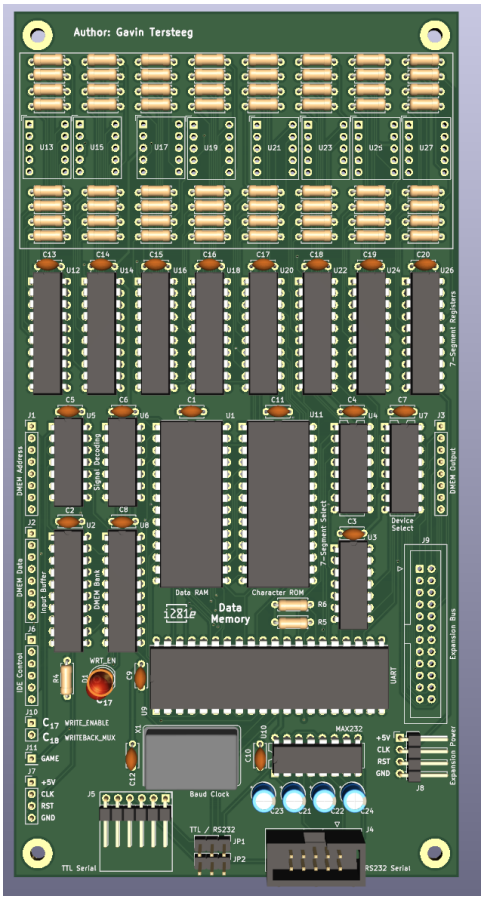


Figure 44 - Data Memory + Video Card PCB (Rev A)

Like the enhanced data memory module from design 2, several essential I/O devices are included on the PCB itself. These devices are the UART and data memory bank register. Circuitry for the compact flash interface also exists on the data memory module, but the actual connectors for the device were moved to the main board PCB due to space constraints.

2.5.5 Design 4 — PCB Revision B

In this revision of the i281e CPU, we made changes to the PCB designs based on our physical testing. Some of the errors were simple mistakes and easily correctable. These mistakes also led to new versions of the PCBs being manufactured for the final product.

2.5.5.1 Mainboard

Since the mainboard was rushed at the end of the order time, we found errors in this board that needed to be corrected in revision B. Most of these changes were tested on the Rev A boards before implemented on the new PCBs.

The power circuit had two errors the input barrel jack connector was moved to avoid any high power solder joints being too close. The power circuit itself was cleaned up to ensure better stability in our output current. This part of the circuit is shown in Figure 45 – Power Circuits Errors Rev A.Figure 45.

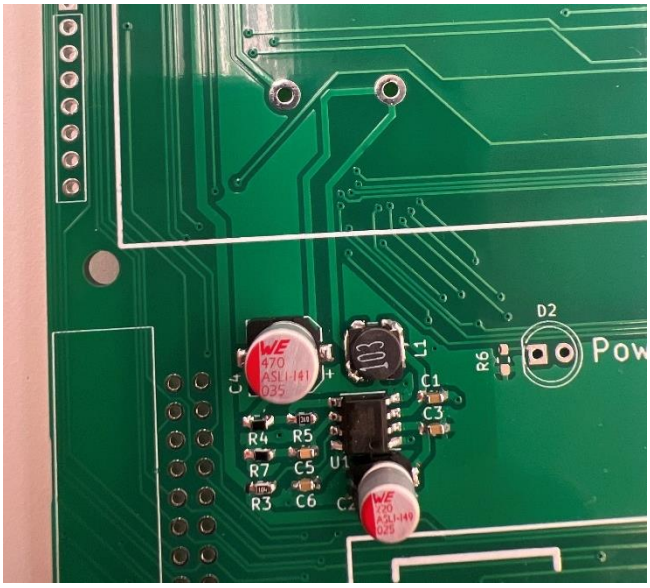


Figure 45 - Power Circuits Errors Rev A.

The reset line was not properly attached to the input reset signal due to names being different, this was correct in the schematic. Since the game mode signal was not properly used in Data Memory this also had to be router to the new heard. Two of the output buses on the main board were flipped label wise.

Additional chips were added under Data Memory allow with resistor and capacitor to improve the flash card's stability when in use. These additional parts can be seen in the bottom right of Figure.

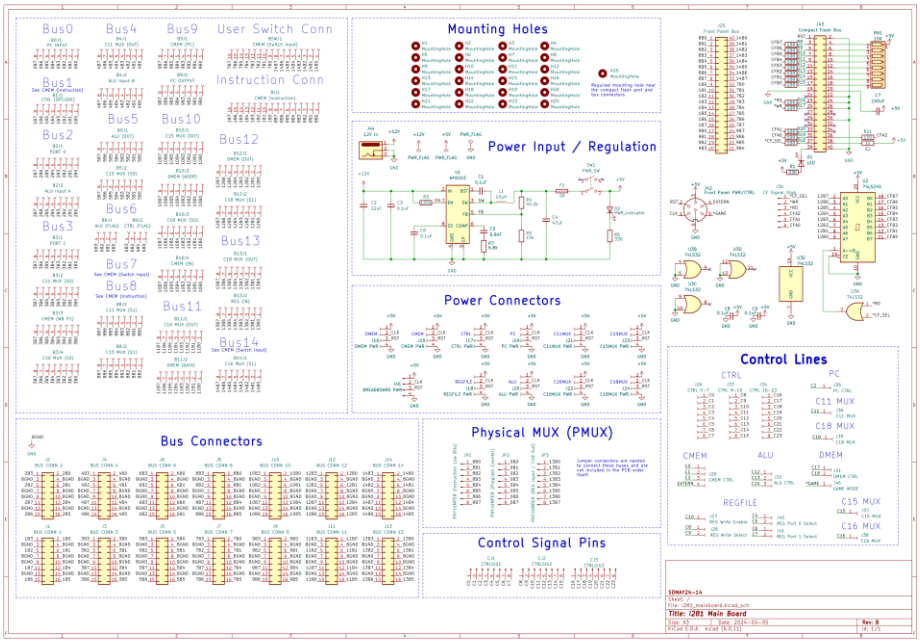


Figure 46 - Main Board PCB Schematic (Rev B)

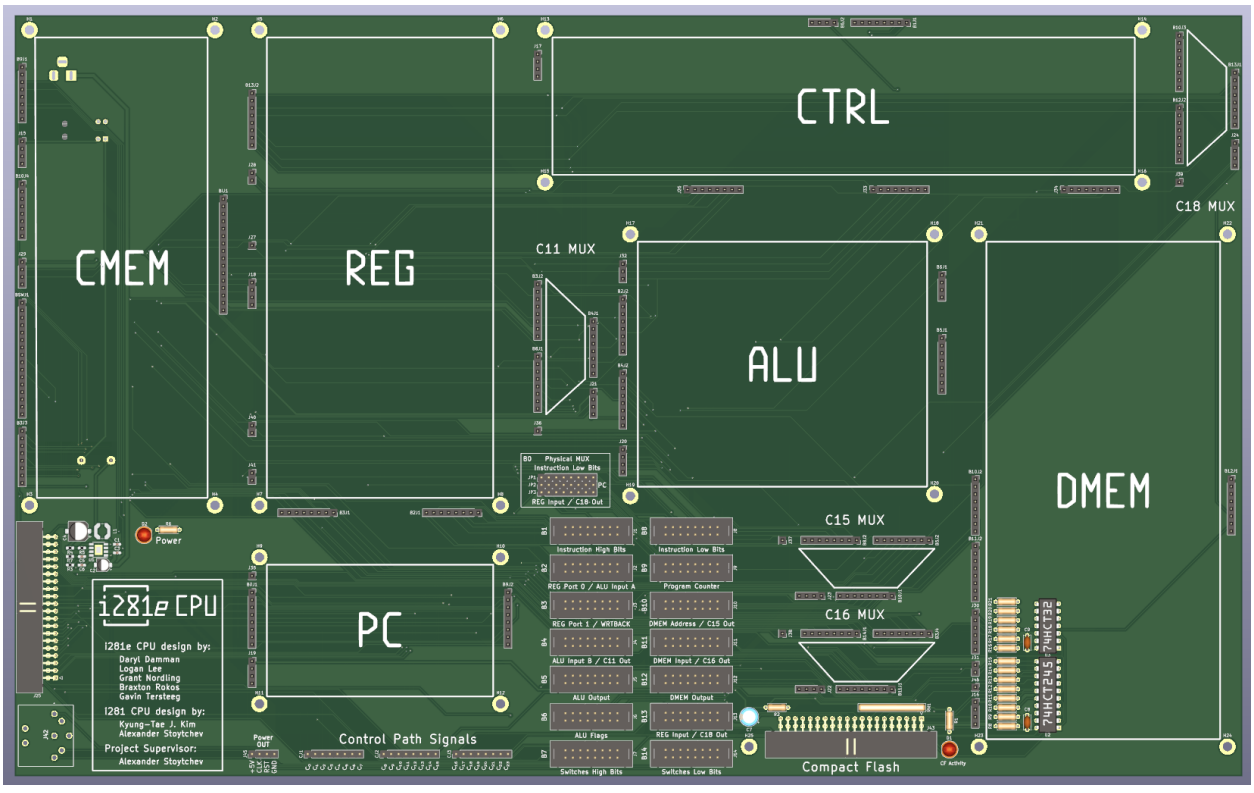


Figure 47 - Main Board PCB Implementation (Rev B)

2.5.5.2 Code Memory

After Revision A was produced into a physical PCB, we noticed there were two problems with the schematic from Rev A. Chip U9, a register, had 5 volts hooked up to the ground connection instead of the ground, Figure 48. Also, C1 was misidentified in the schematic as

an active low signal, causing the LED to be flipped as expected, Figure 49. These changes were fixed on Rev A boards to test functionally and changed in the Rev B version. The updated schematic is shown in Figure 50, and the PCB did not change from Rev A on the outside.

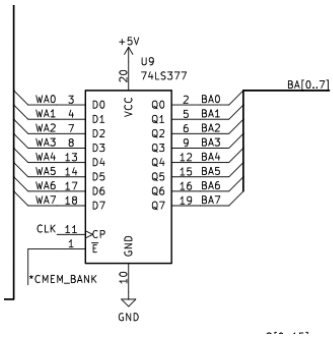


Figure 48 - Code Memory PCB Change 1

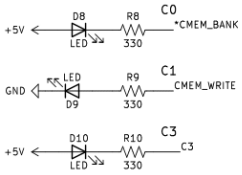


Figure 49 - Code Memory PCB Change 2

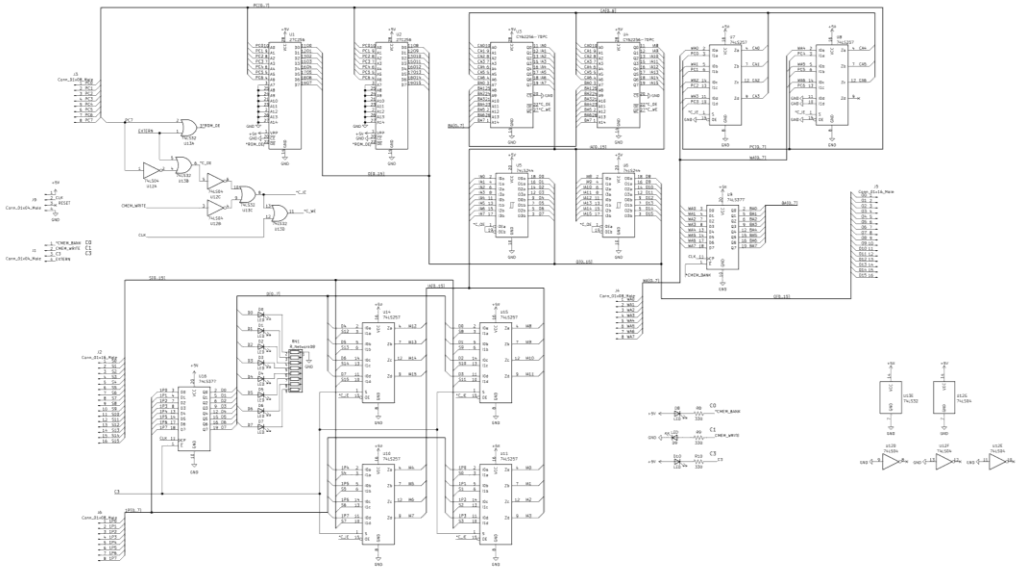


Figure 50 - Code Memory PCB Schematic (Rev B)

2.5.5.3 Register Files

A number of LEDs had errors in them that affected the cosmetic of Rev A but not the functionality of the Register File. This error needed to be fixed, including the control signal C10 LED was set up to be active low when it needed to be active high. The two control signals, C8 and C9, were flipped, representing the write select address. One of the output buses, Port 1, had incorrect LED labeling on the lower 6 bits. These were all corrected in the PCB, as shown in Figure 52.

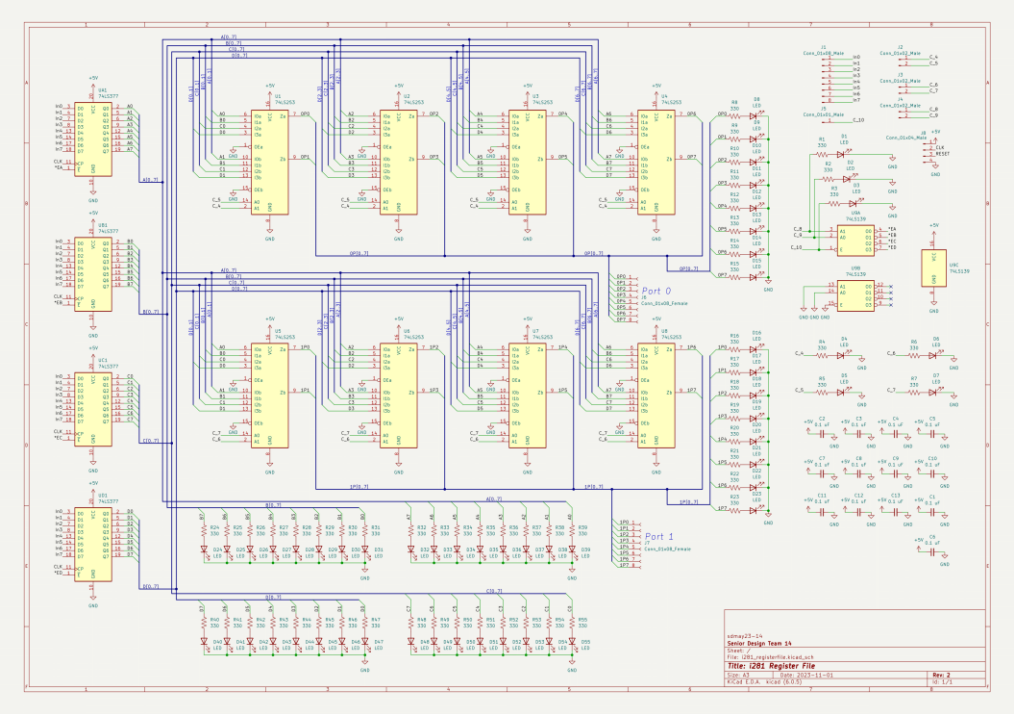


Figure 51 - Register File PCB Schematic (Rev B)

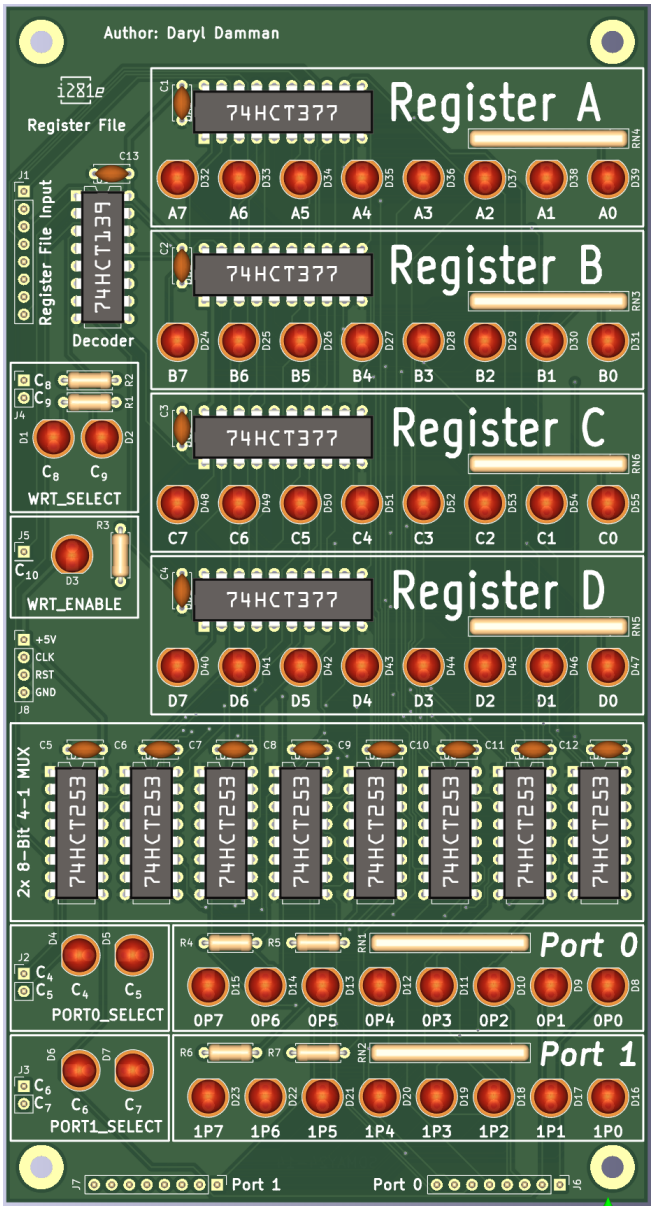


Figure 52 - Register File PCB Model (Rev B)

2.5.5.4 Arithmetic Logic Unit

There were two changes that needed to be made to both versions of the ALU. LEDs and signals for C12 and C13 were switched. The symbol for the XOR chip was inaccurate and needed to be fixed. This caused the board to need to be rerouted.

2.5.5.5 Program Counter

After testing the PCB for the program counter, the PC mux, seen in Figure 53, had the flipped inputs. This made it so that the control signal would output the opposite next program counter value as expected. This was fixed temporarily in Rev A for testing by adding a chip to the back of the board to invert the control signal for the mux chips.

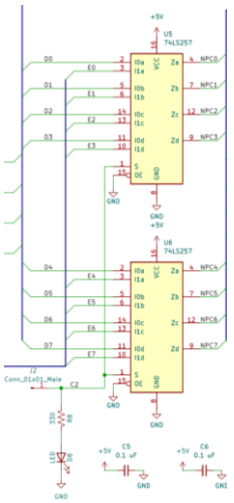


Figure 53 - Program Counter Error from Rev A

The Program counter was fixed for a Rev B by swapping the inputs to U5 and U6. This new revision's schematic is shown in Figure 54, the PCB itself only had internal wiring changes.

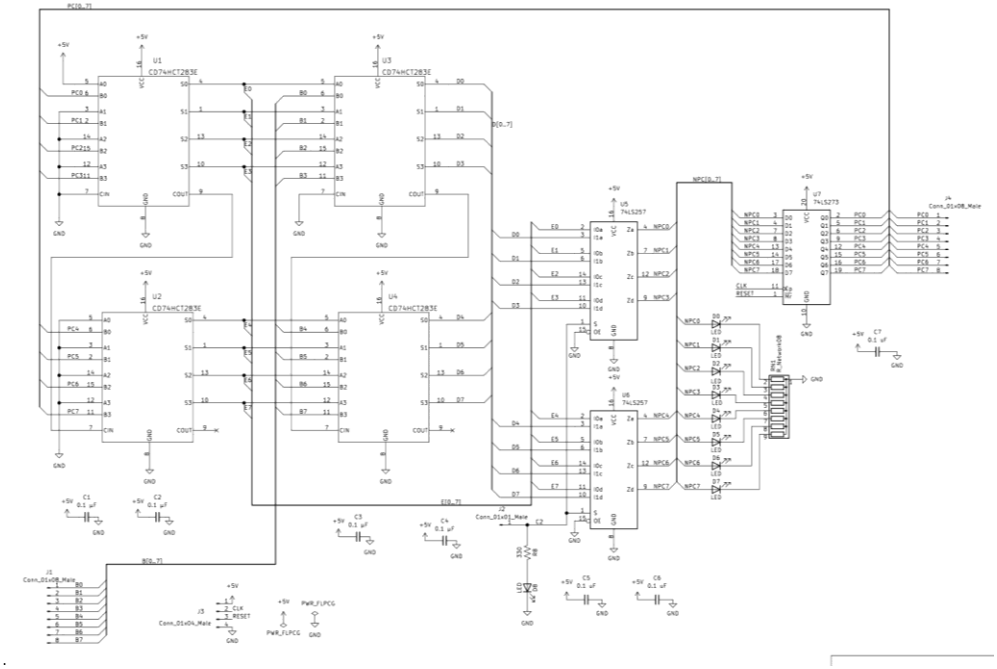


Figure 54 - Program Counter PCB Schematic (Rev B)

2.5.5.6 Video Card & Data Memory

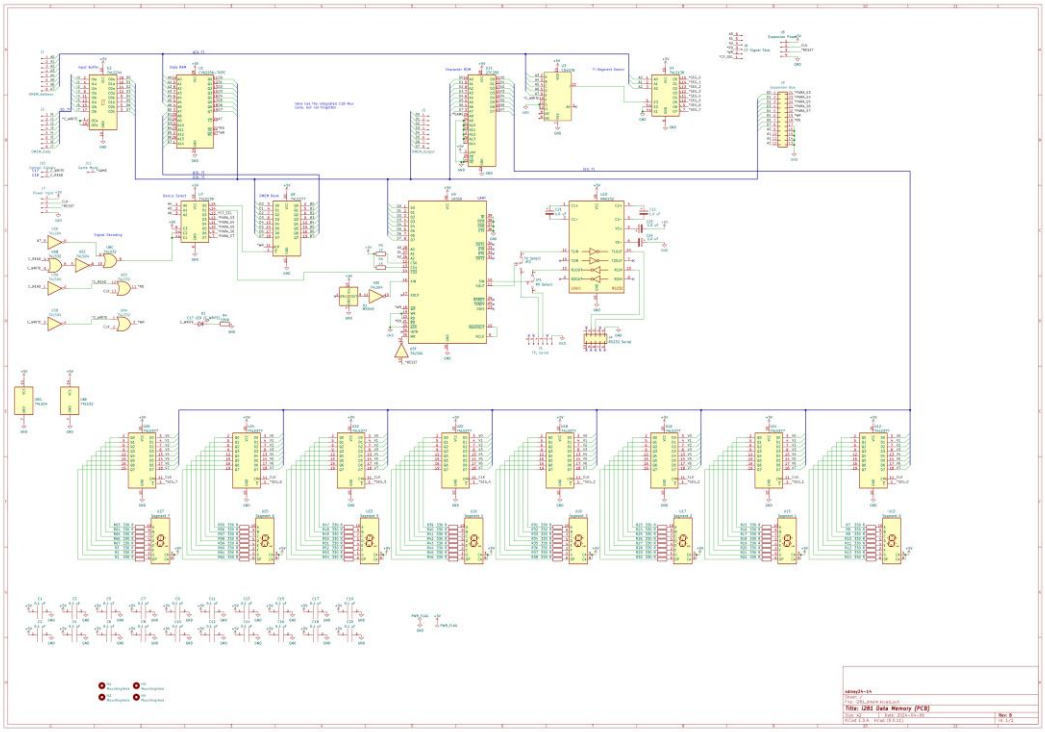


Figure 55 - Video Card & Data Memory PCB Schematic (Rev B)

On the first revision of the data memory module, it was found that the circuitry for "Game Mode" was not present. To rectify this, an additional pin was added on revision B which allowed the Video ROM to be correctly connected to the mainboard signal.

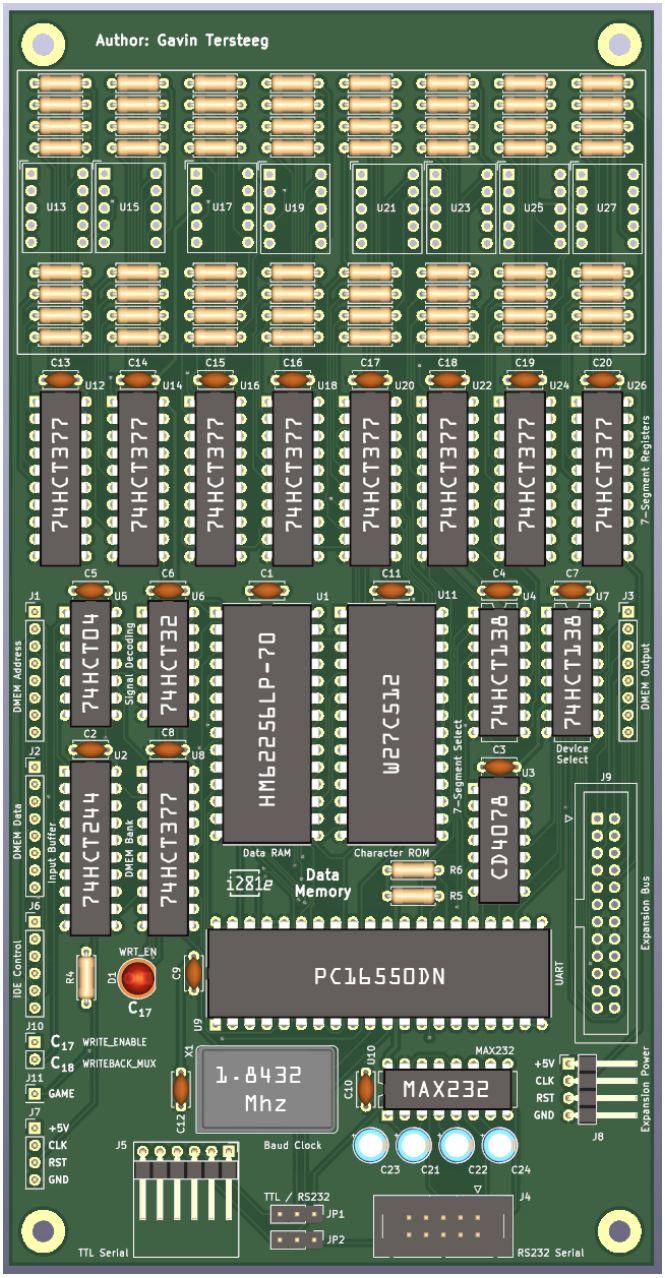


Figure 56 - Video Card & Data Memory PCB Implementation (Rev B)

2.5.6 Design 5 — PCB Revision C

The final design of the project was not completed in time to have a prototype ordered. Unfortunately, most of these mistakes/errors found could have been resolved with some more time and effort spent toward examining Revision B.

2.5.6.1 Mainboard

The inclusion of a capacitor on Pin 20 of the compact flash IDC connector (see Figure 57) was intended to be a bulk filter capacitor; however, the design used is fundamentally not a bulk

filter capacitor. Not only does the capacitor need to be polarized, but it must also be between +5V and ground alongside additional smaller capacitors to dampen switching frequency and PARD noise.

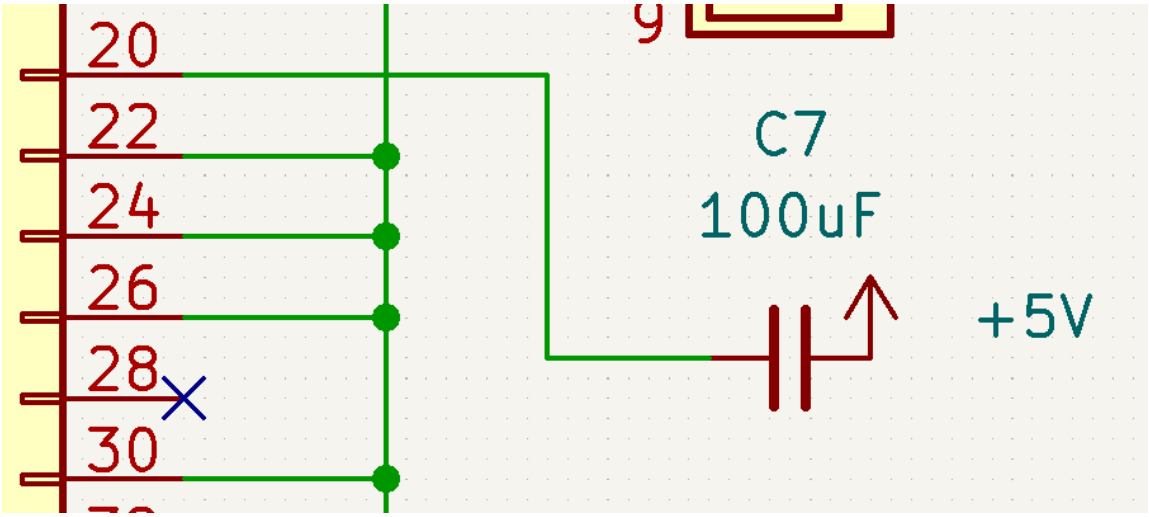


Figure 57 - Inline capacitor for Mainboard

Revision C should include a proper electrical design for bulk filter. Current Revision B boards can be repurposed by simply bridging the connection using a jumper wire.

2.5.6.2 Arithmetic Logic Unit

Something was unfortunately looked past in the ALU during a revision check. Since there are two ALU designs, it was assumed that both designs are modified in tandem; however, a simple mistake was made during the modifications from Revision A to Revision B for the ALU. In Figure 58, Pins 8 and 11 are not connected to the adjacent IC. These wires were incorrectly swapped in Revision A but were forgotten to be reconnected for Revision B in the ALU NOR version. For the ALU design, the wires are appropriately connected.

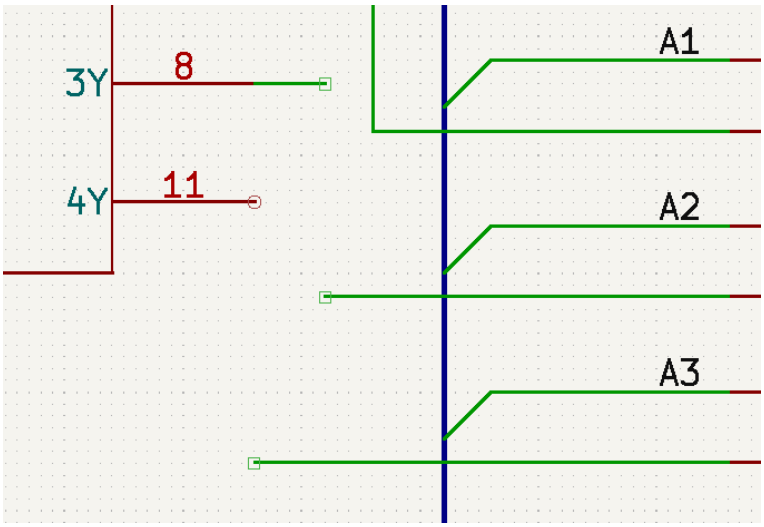


Figure 58 - Missing wire connections in ALU NOR schematic

3 Implementation Details

3.1 Initial Implementation Plan

Before building any implementation, the simulator's section had to be mapped to modules able to work on the breadboard. This is the most important part of the planning stages, to ensure our designs would work before putting everything together. The goals and focus for the first semester were to work on the breadboard prototype and get the design working on a more modular version before the PCB. The PCB was going to be a goal in the second semester after we knew we had a working prototype and design. Although we did not know at the beginning how this was going to come together, we understand that a module-focused design would be the best as a learning tool, and this was initially the goal.

3.2 Design

To start implementing the design of the i281 CPU in both a breadboard and PCB implementation, we needed to start laying out the individual modules functionally and how they interact with each other and the user. The beginning of this work was available in the past interactions, the Verilog, and the Simulator implementation. This core was used heavily in the Program Counter, ALU, Register File, and MUXs. Having clear diagrams for functionality and logic already explained allowed for the logic to be mapped to physical chips and schematics to be created. These were some of the first breadboards to be built due to the lower initial design required.

Other modules required for the i281 CPU implementation need to have the purpose maintained but method functionality reworked to suit a physical implementation. The control table was the simplest of these reworked modules; the individual Boolean logic was swapped with the look-up tables type chip. This allowed for the control table to serve its function of converting the instruction line to control signals around the machine and allowing easy updates to the logic used.

Two modules needed heavy rework to allow for all the required functionality to be present. Code Memory and Data Memory were discussed in depth for a long time as solutions were looked for to allow functionality and simply implement and explain to students learning about the machine. Since Code Memory was deemed a vital part of any testing occurring in the first semester, this was prioritized over Data Memory.

This process allowed for Code Memory to be implanted with debugging functionally added on to ensure the learning tool is the best it can be. The existing BIOS chip seen in simulator i281 was split into a set BIOS stored in ROM and user programs stored in RAM. The

advantage of this system was that ROM was not easily writeable, but RAM chips were, allowing for different programs to be uploaded and run with little delay.

The minimum viable processor was built in the first semester of our project. This allowed testing to be done on most of the breadboard modules and ensured the functionality of our design so far. Continuing the work into the second semester was first finishing the breadboard implementation. With the plan for Data Memory finalized, the module was built. Another module was added to the design to help Code Memory get the user programs from Data Memory, the Writeback module.

While the final breadboard implementation is being thoroughly tested, the focus of part of the team moved to starting PCB schematics. These were quite like the existing breadboard versions simplifying the design process. The design process for overall PCB functionality and appearances was discussed until the main board supported the modules and connected all buses through this.

The first iteration of the PCB modules was created in KiCad and fabricated. With our previous work with the breadboard implementation, we were confident in the design of the i281e CPU and moved forward with soldering all the various logic chips and LEDs onto the boards for testing. After putting them all together, we found various errors in our schematic and PCBs, causing reworks to be needed. These were small errors with the PCB that were all corrected on the Rev A boards to ensure the board works in theory.

We started to create a second revision of the PCB modules, Rev B, that fixed our errors, allowing for printing future boards. These were ordered to be built and tested again.

3.3 Functionality

The i281e CPU is capable of a wide range of functionality since we have built a computer. The BIOS we have developed allows the user to control the i281e CPU using a terminal and DOS/281 commands. A user can hook up a dedicated terminal using a serial or a laptop using USB. This allows freedom in the user options. By using a terminal and command, a user can load programs into RAM from a compact flash card that is integrated with the data memory or upload their own programs to the compact flash card. This allows the i281e CPU to be programmed for a wide range of functions. We have existing programs to show off sorting algorithms, do basic math, play pong, and display graphics in the email window.

The i281e CPU was also made with an expansion bus, adding the ability to add even more functionally to the computer. We have already added printing ability. Even more expansions can be attached here, like a sound card.

The i281e CPU itself is already quite capable as a learning tool. When designing, we were hoping to get clock speeds of up to 1 MHz; in testing, we were able to achieve 2.5 MHz and remain stable. It currently runs at 2 MHz the fastest normal operation speed. While running, we measured the power of the PCB to take 800 mA at 5 V.

The size of storage in the i281e CPU has also been increased compared to the existing i281 CPUs. We have 128 words for the BIOS, 32 kilowords for Code Memory's RAM, and 32 kilobytes of data memory. The compact flash memory also adds an additional 32 Megabytes of storage, like a hard disk.

The PCB implementation also kept the benefit of the breadboard implementation since they are both modular. By taking a module out like a mux, a user can attach an external mux, and the entire i281e CPU works like nothing is different. This makes it so that the breadboard and PCB implementations are capable of each other.

3.4 Finances

During our first semester on this project, we started with a budget of $1000. However, as the semester progressed, it became evident that this budget was insufficient, with total expenditures reaching $731.44. Recognizing the expanding scope of our project, Thankfully, the ETG recognized the project's significance and increased our budget accordingly. Nevertheless, we maintained meticulous financial tracking and documentation throughout this period.

Our documentation efforts extended to creating an automated Bill of Materials (BOM) system, streamlining the process of cataloging parts and sourcing information. This system, derived directly from our schematics, not only facilitated internal organization but also served as a valuable guide for potential project replication.

The breadboard implementation incurred a total cost of approximately $850, encompassing expenses related to breadboards, wiring, LEDs, and supplementary components like the wooden mounting board.

However, the PCB implementation mainly involved costs associated with soldering components onto ordered boards. While precise tracking of these costs was relaxed following the budget adjustment, a rough estimate places the cost of one machine at around $400.

Our progress has been met with amazement from both our client and the ETG. While the breadboard implementation served as a valuable steppingstone, it was not an effective long-

term solution. Instead, it provided crucial insights and experience, laying the foundation for the creation of the PCB implementation.

Accomplishing this transition was imperative not only for our project's success but also to provide future students in 2810 labs with a clearer understanding of the requirements and challenges they might encounter.

In summary, our project's financial journey reflects a dynamic adaptation to evolving requirements and resource constraints. The transition from breadboard to PCB implementation not only optimizes costs but also enhances project robustness and reliability, aligning with our goals of efficiency and sustainability.

4 Broader Context

As noted in the project requirements, the i281 processor aims to be an educational tool for both CPR E 281 and future classes related to processor architecture. Our target audience is students in the ECpE department of Iowa State University and individuals interested in computer architecture.

The ECpE department is affected economically by our project. Not only are they the current funding source for the senior design project but they will also be responsible for purchasing and maintaining the i281 processor's components when deployed to future labs.

It will be noted that there is no societal impact or need for this project. While it is beneficial for students to be educated about grand-picture computing, the hardware-based i281 processor does not provide a significant societal impact to the educational program as seen by the team. This impact may become larger after the development and deployment of the hardware-based i281e processor.

4.1 Identification of New Effects

As mentioned in section 4.4 of our design document, the primary effect that our project is educational. The ultimate purpose of the i281 CPU is to be used as a teaching tool to better educate ECpE students on digital logic and computing basics. As we continue to work through the design and implementation process, we have been able to identify new effects that our project may have on students.

Originally, the primary goal of our project was to teach CPU architectural basics. Through the experience of our own implementation work, we have found that it can be useful for teaching other related subjects. As the i281 CPU is built out of discrete logic chips, the electrical characteristics of these parts must be considered alongside the purely functional. A student working with the i281 platform would get to learn about how digital logic and basic electrical engineering intersect to create useful products.

4.2 Evidence Demonstration of Positive Effects

Through a classroom setting, the i281e CPU can be used to teach electrical design, computer architecture, and assembly software design in a more refined detail than shown in any pre-existing class at Iowa State. The processor itself has been demonstrated to show a large range of capabilities to ETG and our advisor/client.

4.3 Justification of Negative Effectives

The i281e CPU project can be costly to manufacture and implement. Throughout our design process, we have ensured that the overall design and implementation are robust and can

withstand a reasonable time. Measures have been taken to mitigate the cost of PCBs, components, and assembly. Lastly, all implementations are or will be heavily documented to ensure future reuse and ease of repairability.

5 Testing

5.1 Process

What's the overall process of testing?

5.1.1 Breadboard Testing

5.1.1.1 Unit Testing

The i281 project involves building several smaller sub-components that will eventually need to interface and interoperate with one another. Instead of building all modules and then testing them together as one system, we settled on a strategy of individually verifying the functionality of each sub-component before attempting to connect them. This unit-testing strategy is done in two stages.

First, the unit is tested electrically. There are many common mistakes that can be made when wiring a solderless breadboard. Before power is applied to the module, the resistance between the 5 volt and ground rails is checked to ensure that there are no short circuits. If that test is successful, then the power pins on each integrated circuit will be checked to confirm that they are on the correct power rail. By doing this, we can be confident that no damage will come to the components when power is applied. Finally, once power is applied to the breadboard the electrical characteristics are checked. If the voltage is found to be sagging, too much current is being drawn, or if chips are getting hot, power will be removed and the design to be reviewed for errors.

After electrical testing is complete, the sub-component is tested logically. To assist us in this task, we built testing boards which allow us to run manual test cases for the modules. These "testing rigs" consist of banks of switches and LEDs, so inputs and be manually set and outputs and be visualized (See the image below). Using the switches, we can test how the circuits react to changes in the switches. By doing this, we can test most, if not all, scenarios that the circuit will go through.

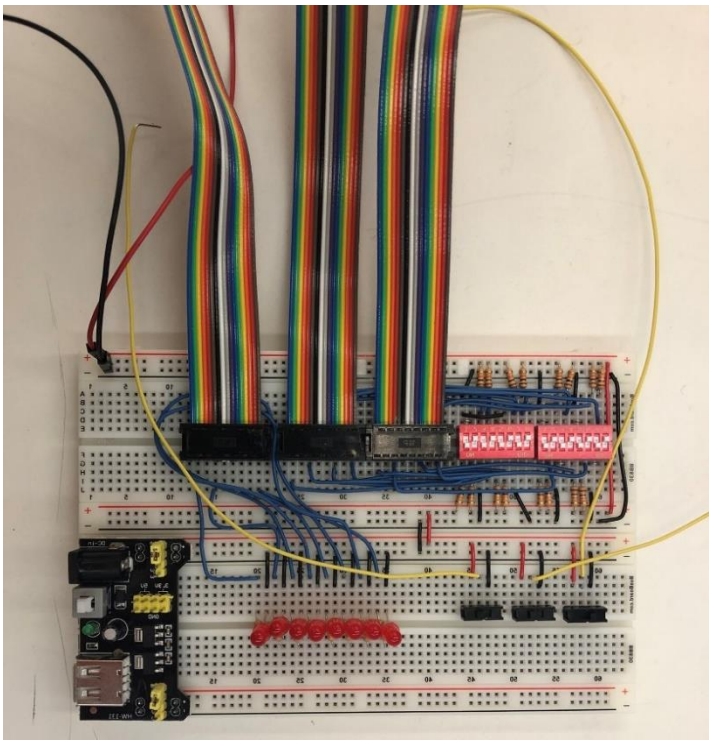


Figure 59 - Testing Rig

Assuming all electrical and logical tests pass, the sub-component will be marked "OK" and set aside for integration testing with other modules. By performing this process, we will have some degree of confidence about the functionality of each module before attempting to make them work together in a larger system.

5.1.1.2 Interface Testing

In the design of the original i281 processor, there are a few points where the user can interact with the state of the machine:

- The "switch register", which manually be set by changing a bank of 16 switches.
- The execution control section, where programs can be stopped, started, and stepped through depending on the desires of the user.
- The "game mode" switch, which changes how the 7-segment displays format information.

These interfaces are adequate for casual users running example programs for educational purposes. However, we realized earlier on that we would need slightly more sophisticated interface facilities for system and integration level testing. For that reason, we decided to combine almost all user inputs, execution control, and debugging options into one module. This module, known as the "user panel", possesses all existing interface options plus a few new features meant to assist in system debugging.

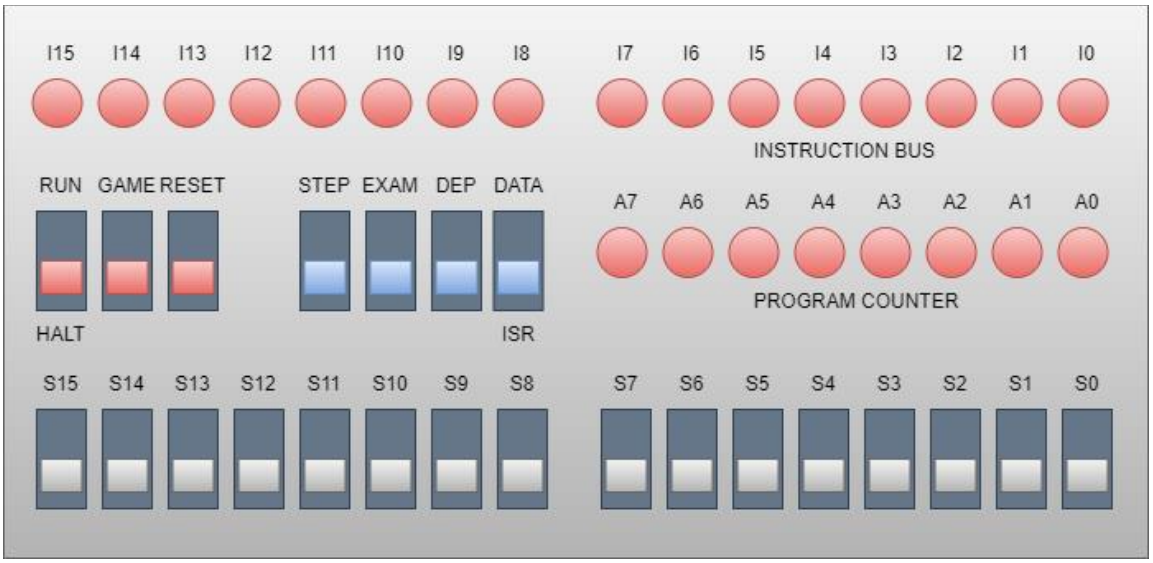


Figure 60 - User Panel Design

Component Type	Description
Switch Register	This is a set of 16 switches at the bottom of the user panel. These switches are further broken down into two banks of 8 switches. They exist as the main way that user data can be inputted to the processor. The switch register performs different operations depending on the instruction executing or debugging operation selected.
Run / Halt Switch	This is the first switch in the control group. It allows the user to toggle between automatic and manual program execution. In the "Run" state, the program counter will be automatically incremented depending on the configured speed of the system clock. When the switch is moved to the "Halt" state, execution will indefinably pause. In this state, the user is free to use any of the facilities in the debug group. Moving the switch back into the "Run" state will resume program execution.

Game Mode Switch	This is the second switch in the control group. It controls how the first 8 bytes of data memory are visualized on the 7-segmenet displays of the video card. When the switch is down, the contents of memory will be displayed in hexadecimal format. When the switch is up, the bits of each byte will be mapped directly to segment on the display.
Reset Switch	This is the third and last switch in the control group. It is used to reset the processor state back to the boot state. At this state, the processor can be booted, or optionally debugging operations can be executed. When the processor is first powered on, the reset switch must be used to put the processor into a defined state.
Single Step Switch	This is the first switch in the debug group. When the processor is in a “Halt” state, strobing this switch will send a single clock cycle to the processor. This can be used to manually step through a program for debugging and educational purposes.
Examine Switch	This is the second switch in the debug group. It is also the first switch that is unique to the hardware implementation of the i281 design. When the switch is strobed, the contents of the lower 8 bits of the switch register are added to the program counter, and then incremented. This value becomes the new program counter. During this operation, the state of the other processor components is not affected. Since the program counter and the location in code memory that it points to is always visualized, this allows for the

	contents of code memory to be manually checked without executing the instructions stored.
Deposit Switch	This is the third switch in the debug group. Like the examine switch, it is unique to the hardware implementation of the i281 design. This switch is designed to be used in conjunction with the examine switch. When this switch is strobed, the contents of the switch register are placed in the memory location pointed to by the program counter. The program counter is then incremented by one. The purpose of this switch is to allow for the manual programming of memory without the assistance of the BIOS or Boot Hard Disk.
Code / Data Switch	This is the fourth and final switch in the debug group. It controls if the deposit switch enters data into code memory or data memory.

Table 10 - Switch Types and Applications

In addition to the front panel, the clock speed can be controlled via the rotary encoder found on the clock module. This can be used to dramatically slow down processor execution.

The main way that the user panel can perform these debugging operations is by "mocking" instructions on the instruction bug. The "Examine" and "Deposit" switches work almost the same as the "Single Step" switch, except for one key different. When these switches are depressed, the code memory module will de-assert the bug, and allow the debugging board to assert a single instruction instead. This instruction can be a JUMP, INPUTC, or INPUTD depending on the desired operation. When the single step circuity first, this instruction will be executed instead of an instruction from code memory. This allows for the debugging features to be added to the user panel without incurring much hardware complexity costs.

Technically, the Boot Hard Disk (BHD) can be swapped out to provide different user/example programs; however, this is not considered a traditional interface. To test the boot procedure, we will be swapping programs on the BHD to confirm: instructions load into RAM, critical control lines are solid, and operation after boot is as expected.

5.1.1.3 Integration Testing

What are the critical integration paths in your design? Justification for criticality may come from your requirements. How will they be tested? Tools?

Due to the complexity of the i281 design, we figured that attempting to build the entire processor and then debug it would be too difficult of a job. To streamline the integration process, we decided to build and test a "minimum viable processor" before attempting to test the entire system. This minimum viable processor, or MVP, consists of the bare minimum required to test processor activity. The MVP consists of:

- The User Panel
- The Code Memory Module
- The Clock Circuit Module
- The Program Counter Module
- The Instruction Decoder Module
- The Register Module
- The ALU Module
- Various MUX Modules (Interconnection).

Importantly, the MVP explicitly excludes:

- The Data Memory Module
- The Video Card Module
- The Boot Hard Drive Module

By implementing a minimal processor, the critical path of the processor can be tested and verified before more complex components are added.

After the minimum viable processor has been constructed and verified, the rest of the system can be put together. This involves constructing and integrating the data memory, video card, and boot hard drive modules. These are all complex pieces of hardware, so it is important that the rest of the processor is known to be functional before debugging of those modules begins.

5.1.1.4 System Testing

Each "island" of the i281 CPU needs to be tested independently to ensure the functionality of each section. These will be tested for base functionality to prove they were properly built and can interface with other sections of the CPU. Testing is completed using an existing testing board capable of inputting two 8-bit numbers and outputting an 8-bit number; since two of these boards exist, we can test up to four inputs and two outputs simultaneously. We also use an Arduino microcontroller for testing purposes that creates a clock signal. This can

output a clock signal at any frequency required and a manual toggle clock; for testing, this often kept a low frequency so that we can watch individual steps of the component to ensure proper functionality.

When putting the different "islands" together, we will need to test the interconnects of each component to ensure the sections are working as expected. Along with individual testing when putting "islands" together, we will also need to test the overall operations of the CPU. After connecting different components, these integration tests will be completed frequently, whenever possible.

5.1.1.5 Regression Testing

We are ensuring that new additions do not break the old functionality by ensuring compatibility between the new and old components before connecting and running them. After they are connected, we can test the functionality of it via the 7-segment displays and various other LEDs around the board. This is driven by requirements as one of the project's goals is to have a class taught about and around the CPU. The students will then build and test their designs with our CPU.

5.1.1.6 Acceptance Testing

The first form of acceptance test we must do is check the functionality of the individual system modules. Each module has a set of requirements defined by our client. This outlines what features the module should have, what parts of the module must be visualized, and what implementation strategy should be used. We will check with our client during the development and debugging process so ensure that each module meets their requirements.

After system integration is complete, acceptance testing is done by ensuring that the i281 CPU can fulfill all the requirements originally set out by our client. The main aspect of this is that the system must be able to execute all existing i281 example programs with little to no modification. If all example programs can be successfully executed, it is safe to say that the processor is in an acceptable state.

5.1.2 PCB Testing

5.2 Results

Presuming the system/design/implementation has been tested, what were the results for testing? Did the implementation perform adequately? What was the coverage?

5.2.1 Breadboard Results

We are testing the sections of the CPU as they are built to ensure they are functional. So far, we have built and tested the 8-bit 2-1 mux, code memory, program counter, arithmetic logic unit, register files, switch board, and control table circuits.

The 8-bit 2-1 mux were some of the first components we built for the i281 CPU. Since we had no existing testing hardware, we had to make a new testing board capable of proving input and output paths to the component. This testing board is used throughout the other components to check functionality.

The 8-bit 2-1 mux was modeled after the picture below, taken from i281 class notes explaining how the i281 CPU worked. We ensured each input bit mapped to the correct output bit when that signal was active.

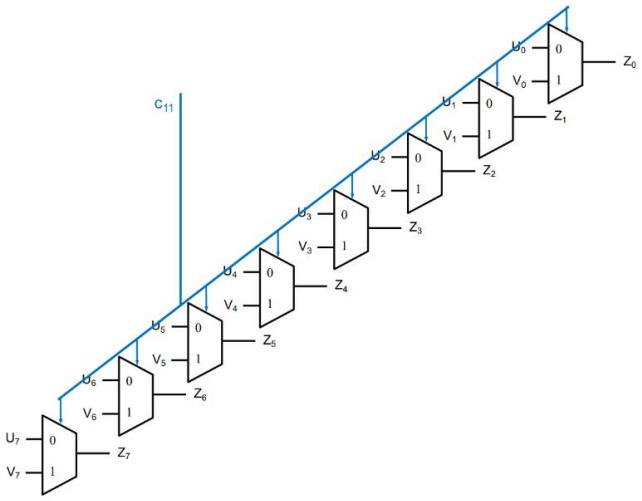


Figure 61 - 8-Bit 2-1 Multiplexer Design

When testing the program counter, we initially tried to use a switch as the clock pulse but found this unusable without a denouncer. We used an Arduino as the clock for this sensitive component to ensure our testing conditions would match the final usage. This will be used as the clock in future testing as well. The picture above demonstrates the expected functionality of the component. We started by only testing the program counter when c2 = 0, increasing the stored number by one each time. This would help us narrow down problems in the circuit before adding additional signal paths to the data path. After getting the main wires of the output stage into the register correct, we added an offset to the program counter ensuring that all functions worked as expected.

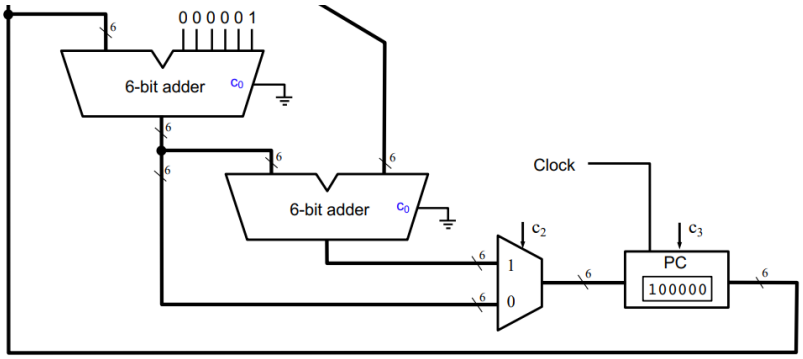


Figure 62 - Program Counter Design

Testing the code memory section was done in the same manner that all the other modules were tested. The only difference is that two testing modules had to be used to accommodate the number of inputs and outputs for the module. All features of the code memory section were then manually checked out. This includes reading from RAM and ROM, writing to RAM, and different bus arbitration states depending on the input address, program counter, and control lines.

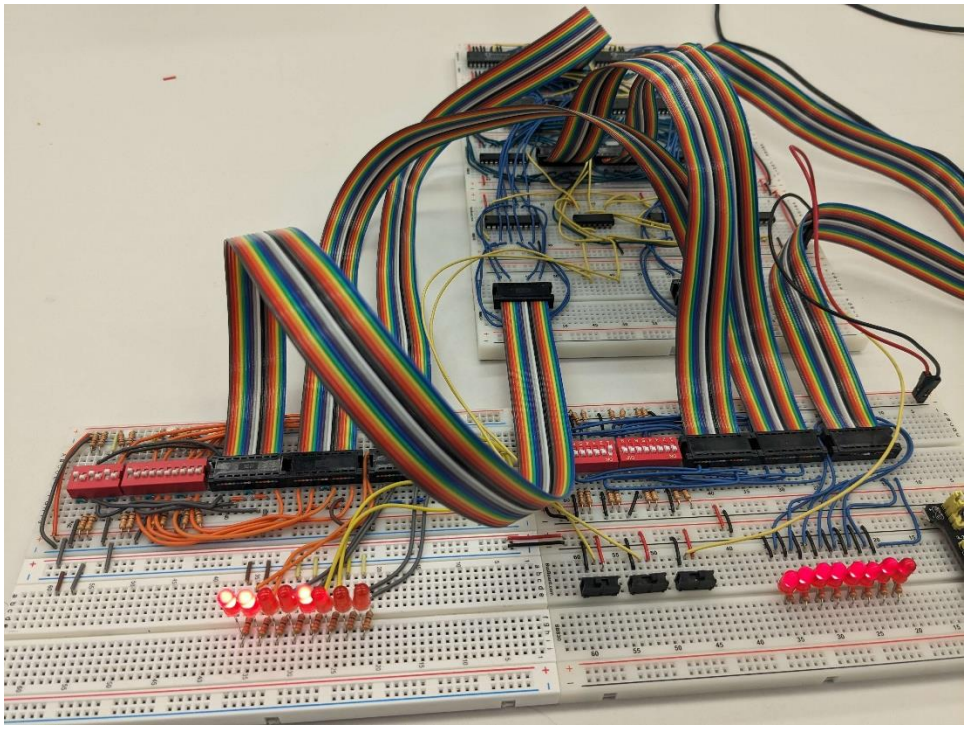


Figure 63 - Testing Rig Connected to the RAM and ROM

We tested the ALU in a very similar way. We used one of the testing rigs to hook up to the ALU since they were designed to have two 8-bit input buses and one 8-bit output bus. The ALU had three control signals, so we connected each of those two by a switch. We also used the Arduino Nano clock board to simulate the clock signal in the flag register chip. By changing the values of C_{12} and C_{13}, we were able to switch the mode the ALU was in. We

exhaustively tested each mode with multiple different potential inputs. To see if the design was working, we looked at the output LEDs. Based on our inputs, we predicted the outputs and checked if they matched the LEDs. For the addition and subtraction arithmetic, we used inputs from Professor Stoytchev's CPR E 281 class slides to verify our results. In addition to the outputs, we used the slides to check if the Zero, Negative, Overflow, and Carry flags were triggered. When we initially tested the design, we found a variety of errors such as an incorrect orientation of the output LEDs, incorrect overflow errors, and issues with the shifter circuit. These issues were then debugged, fixed, and some led to Design iteration two. After the issues were addressed, the design was retested and passed.

5.2.2 PCB Results

Testing for the PCB model was done in several ways. Since the PCB schematics were based on the schematics used to build the prototype unit, the fundamental circuitry behind each unit has already been tested. That way, certain points of failure on the PCB implementation can be ruled out when debugging.

In terms of physical testing, the PCB modules were first evaluated the same way as the breadboard modules. After each module was assembled, continuity was tested between the power and ground plains to ensure that there were no shorts. The +5V and GND pins of each chip were also tested to catch any wiring mistakes. Finally, power was applied to these boards individually to check power consumption and look for overheating chips as a sign of faulty wiring.

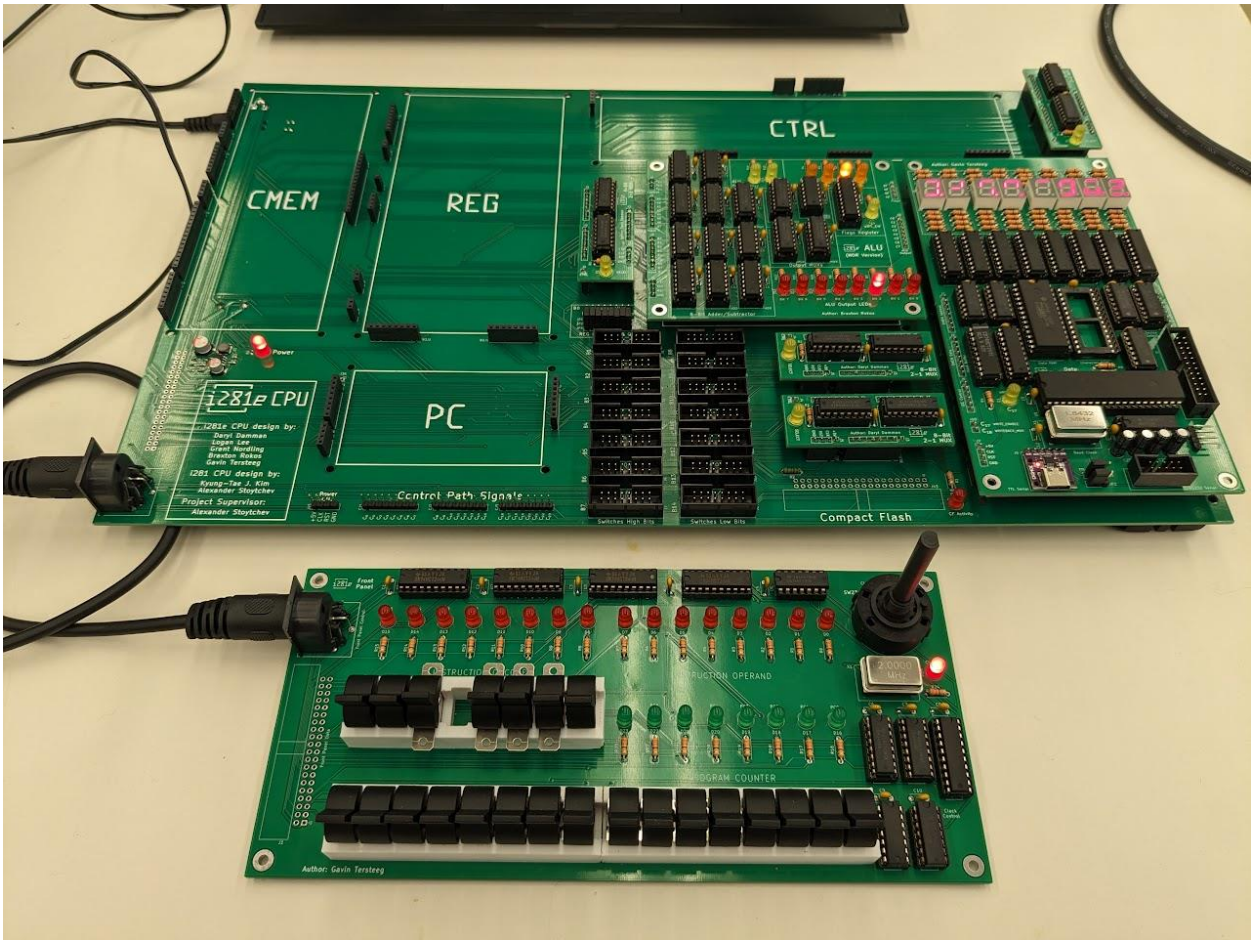


Figure 64 - Partially constructed PCB machine to test electrical characteristics

Once a module has been confirmed to be electrically safe to work with other hardware without risk of damage, the functional components of the module are tested next. Using the 24 pin IDC connectors found in the bottom-middle section of the mainboard, different buses could be connected. By connecting the input and output buses of some modules to the switch inputs and front panel output, several modules were able to be tested. The program counter was also able to be independently tested by supplying it with a clock signal from the front panel.

Using a multimeter, we also tested individual bus connections between different modules to ensure they went where they were on the breadboard prototype. We found several connections on the revision A machine that did not match their expected destinations on the design document. These issues were fixed in revision B.

Most of the testing and debugging on the PCB machine occurred when the full system was assembled. By doing this, we could attempt to execute simple programs and then compare the CPU state to the breadboard prototype and the emulator. Any instruction that failed to

result in the correct state was traced back to its corresponding circuitry and monitored using an oscilloscope. Using the single step feature on the front panel, instructions could be fully evaluated on a cycle-by-cycle basis.

With most basic instructions confirmed to be working, we were able to use the "Monitor" program to continue our testing and evaluation of the PCB machine. The monitor allows for memory and program execution to be manually manipulated over the serial UART connection.

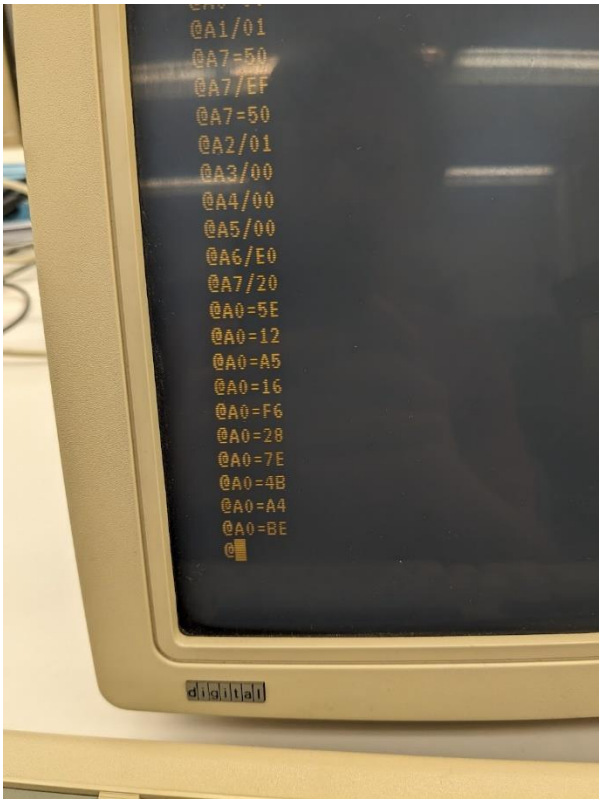


Figure 65 - Usage of the monitor program to examine portions of memory

Using the monitor program, it was significantly easier to enter information into instruction and data memory. Information input into data memory could also be examined to ensure that the memory circuitry was storing and retrieving information correctly. In addition to RAM, hardware devices hooked up to the data memory module could also be manually interacted with to confirm basic functionality.

With the monitor program working, this opened the option for test programs to be uploaded to the processor over the UART and executed. Many of these test programs were existing i281 programs such as bubble sort or the banner program. We also wrote custom unit test programs that exercised the ALU and made sure that all flags and outputs were being set as expected.

```
36          ; ADDITION TEST #1
37          LOADI   A,9
38          LOADI   B,10
39          ADD     A,B
40          BRZ     @
41          BRC     @
42          BRO     @
43          BRN     @
44          SUBI    A,19
45          BRNZ    @
46
47          ; ADDITION TEST #2
48          LOADI   A,0X7F
49          LOADI   B,1
50          ADD     A,B
51          BRZ     @
52          BRC     @
53          BRNO    @
54          BRNN    @
55          SUBI    A,0X80
56          BRNZ    @
```

Figure 66 - Snippet from ALU unit test program

6 Conclusion

6.1 Progress

What's the progress of the project? Is it fully completed? Is there more work to be done?

Our constraints for the project in our goals were our academic and external work. There were technical hurdles with translating an FPGA design into a hardware implementation; however, these were minimal and handled in a timely manner. Physically implementing the prototype on breadboard took the most time out of the project.

6.1.1 Time Spent on Project

For the second semester of this project, Table is the recorded time of project work within an estimate 10% deviation. Hours were recorded per quarter hour on a periodic basis in tandem with our "biweekly reports." The total cumulative hours for the semester across all persons are 773.75 hours.

Name	*Period 01*	*Period 02 (est.)*	*Period 03*	*Period 04*	Period 05	Cumulative Hours
Daryl Damman	N/A	45	14.5	65	80.5	205
Logan Lee	N/A	20	15	41	34	130
Grant Nordling	N/A	7	7	15	14	43
Braxton Rokos	N/A	27	12	37.5	41.25	144.75
Gavin Tersteeg	N/A	30	25.25	80.75	85	251

Table 11 – Hours toward i281e project for second semester

Unfortunately, time was not recorded for the first semester of the project. As such, we can only assume that the amount of time spent on the previous semester is roughly 85% of the total cumulative hours of this semester, which is approximately 657.75 hours. This would mean that the total cumulative hours are somewhere in the range of 1400-1600 hours.

6.2 Project Value

The final version of the i281e CPU we have put together is quite efficient for its usage in the classroom and as a learning tool. Throughout our project's design and implementation, our client, ETG, and other visitors have been impressed with what we have managed to do.

When starting this project, we were told to make a physical version of the i281 CPU like the existing version in CPR E 2810 slides, a simulator, and Verilog implementation. We took their existing work and made an equivalent physical i281 CPU and went above the old capability,

making our PCB into the i281e CPU. The base functionality was not comprised in our implementation only improved upon.

The i281e CPU is helpful for students taking CPR E 2810 and can be used further in follow-up classes to teach more about the computer system. CPR E 2810 finally has a physical version that students can use and learn more from.

By building a fully modular design, it also allows even more usage than just explaining inside of a classroom or lab. Dr. Stoytchev has discussed with us his goals of having an additional class that is a follow-up in the sequence to CPR E 2810, like CPR E 4810, that discusses the i281 CPU in more depth and even works on building some modules in the lab. This shows another advantage of making a breadboard implementation before the PCB implementation, as we already have a working version of these for a base of these hypothetical labs.

Overall, this project, the i281e CPU, often gives many educational opportunities for the student body to take future computer engineering classes.

6.3 Future Considerations

Although we have tried to accomplish everything in our senior design goals, there is always more work on this project that can be down. For our process to be utilized effectively in a learning environment, the information we have produced about the i281e CPU needs to be properly organized and managed. We have done work to help this process throughout our time, but it is never complete. A user manual is being written to help future users understand how to use and manage our design.

The PCBs have been designed, and many have been produced for testing, but not all were able to be produced, mainly Rev C. There is also software development that can be worked on, allowing for more expansion devices to be interfaced with the i281e CPU.

7 Appendix A

Operation Manual

This appendix provides all necessary information required to fully use, maintain, and apply the i281e CPU. Below, each category is split into their respective sections.

7.1 Module Layout

7.1.1 Layout of the i281e CPU

From Figure 67, we can see the proper alignment and orientation of the modules. Use this Figure as a reference to the installation of the modules. When installing the modules onto the Mainboard, ensure that the i281e CPU logo is in the bottom left corner. There are 10 modules on the Mainboard, and 4 of them are the same MUX Module used multiple times in different orientations. As a rule of thumb, for all modules excluding the MUX Modules, all the module titles silkscreens should be angled the same way as the text on the Mainboard.

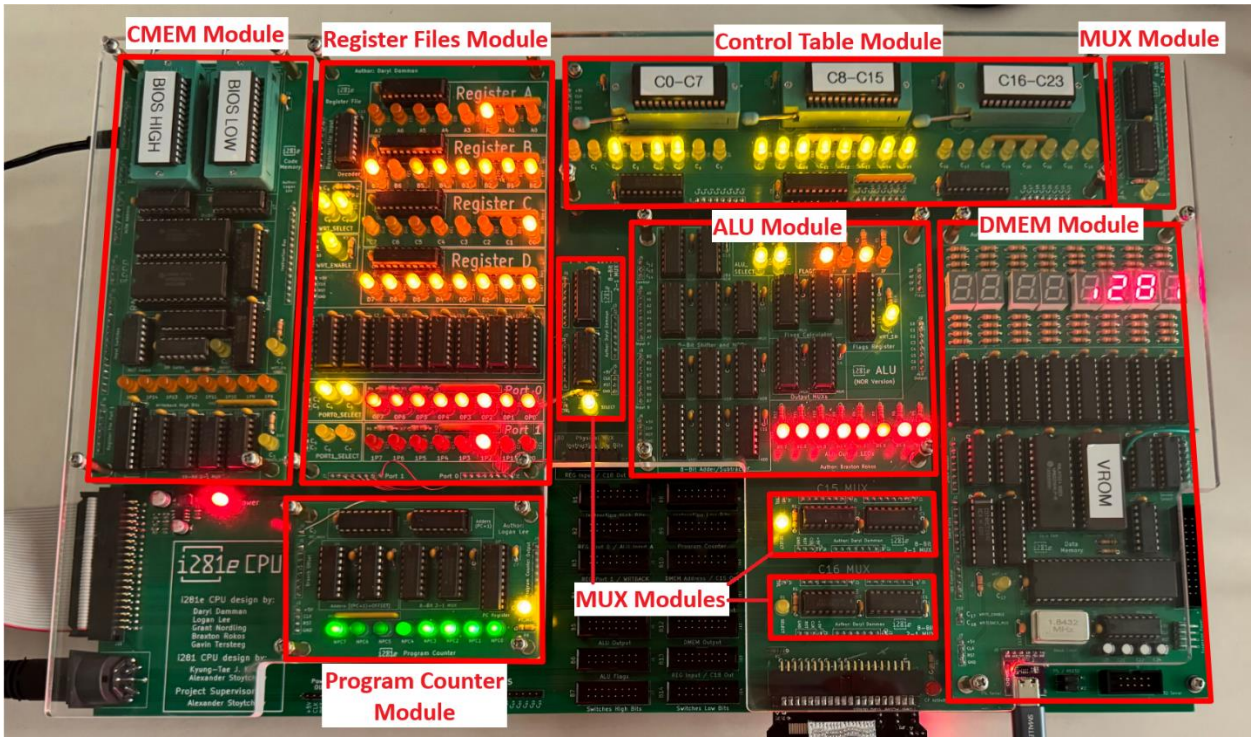


Figure 67 - Modules Labeled on i281e CPU

Additionally, if we look at Figure 67, we can see the layout of the Mainboard. Each of the modules has a location designated by the silkscreen. The four holes of the larger modules match up with the four holes on the backboards of the desired location. This is where the mounting hardware will connect the modules securely to the backboard in addition to the Polycarbonate top and bottom covers, as seen in Figure 68.

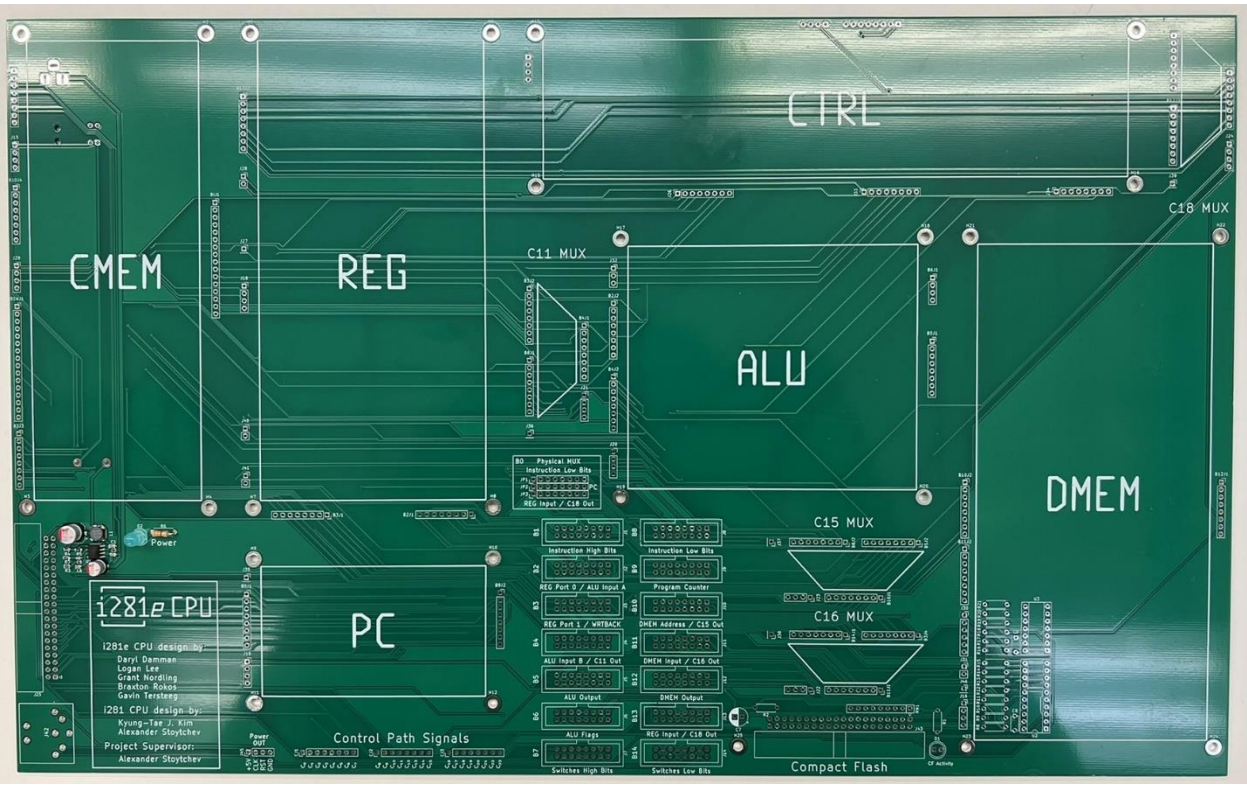


Figure 68 - Mainboard PCB without Most Components

7.1.2 Mounting Methods

To safely attach the module to the Mainboard, ensure the mounting hardware is installed in the holes on the Mainboard. The mounting hardware is done in a "sandwich" design.

- Layer 1
  - 1/4" acrylic/polycarbonate panel with screw holes appropriately aligned.
  - There shall be five >10mm rubber feet in cross formation underneath the acrylic/polycarbonate panel.
- Layer 2
  - Mainboard PCB
- Layer 3
  - Respective modules (REG, PC, DMEM, etc.) will be seated on the pin headers.
- Layer 4
  - 1/4" acrylic/polycarbonate panel using top panel A or B (see hardware repository).

In between each layer, there will be respective standoffs and spacers to ensure the hardware remains in a soundly rigid fashion. See Figure 69 for an example of this sandwich design put together. The hardware repository contains information regarding the exact part information. The following steps should be taken for each accessible mounting location.

- Layer 1
    - Insert screws with one washer inserted bottom-up with male-to-female standoff fastened with the male end facing upward.
- Layer 2
    - Set the Mainboard PCB on-top of the upright standoffs and fasten two washers and a spacer.
- Layer 3
    - Set each respective module in the correct orientation and fasten male-to-female standoffs with the male end facing downward.
- Layer 4
    - Set top panel A or B and fasten screws with one washer into female receptacle of standoff.

Some PCB modules cannot be attached to the top panel and will need to have a screw with washer inserted at Layer 3. Once the mounting hardware is installed, one can line up the corners of the module with the 4 mounts. Take note of the orientation of the module aligning it. Once you are ready to attach the module, simply press down on the module evenly. If done correctly, all the pin headers should have gone into their corresponding receptors on the Mainboard.

Figure 69 - Mounting Hardware

To safely detach a module from the Mainboard, ensure that the acrylic top sheet is no longer attached to the boards and the top 15mm standoff is removed. With an even distribution of strength, pull up on the module. The module should pull up without too significant of force. If you are having a hard time detaching the module, double check nothing is restricting you. If there are no restrictions, hold the module by its input and output header edges and wiggle it back and forth until it starts to give way. Be gentle to not bend pins or crack a solder joint.

7.2 Power / Power Safety

One of the most important things to know about the i281e CPU is how to turn it on properly. Questions you may be thinking to yourself might be "Do I need to wear an anti-static wrist strap?" or "How do I know if the CPU is on?" This section will give you all the answers to those questions and ensure that the CPU is not damaged in the process of turning it on or using the device.

IMPORTANT:

When using the i281e CPU, one MUST wear an anti-static wrist strap or alternative ESD protection equipment/methods.

An anti-static wrist strap will prevent electrostatic discharge (ESD) from damaging the CPU. ESD can damage the IC chips or alter their programming. To prevent damage, always ensure that you are grounded prior to touching the equipment, this includes plugging in the anti-static wrist strap into the designated areas. Most lab tables and desks have ports in them that correspond to the pin at the end of your anti-static wrist strap. If your anti-static wrist strap has a clip at the end, connect it to something made of metal that is not painted. It is preferred that you ground yourself to a table or alternative location, but there are always options. When anti-static wrist straps are not provided, this grounding could be as simple as touching an unpainted metal object prior to touching the equipment. An anti-static wrist strap overall is a simple method to ensure you do not harm the electronics.

7.2.1 How do I plug the i281e CPU in?

First, you need to find the power cable and find an outlet to plug it into. The port to plug in the power cable is on the top left of the Mainboard. This port is on the underside of the Mainboard. With one hand, hold the Mainboard and with the other, insert the plug into the port. Ensure that it is a snug fit.

Second, press the button right next to the port on the underside of the Mainboard. Once you press the button, there should be a light blue LED (Rev B) that lights up near the bottom right of the mainboard. Once this LED is lit, the CPU is on. Power will be provided to the board and the startup sequence for the i281e CPU must be achieved.

If the LED is not powered on when pressing the button, the plug may not be fully in. Ensure that the plug is firmly pressed into the slot.

7.3 Front Panel Usage

The switches on the front panel are the main tool to interact with the i281e CPU and influence its operation. A diagram with the switch layout is shown in Figure 70.

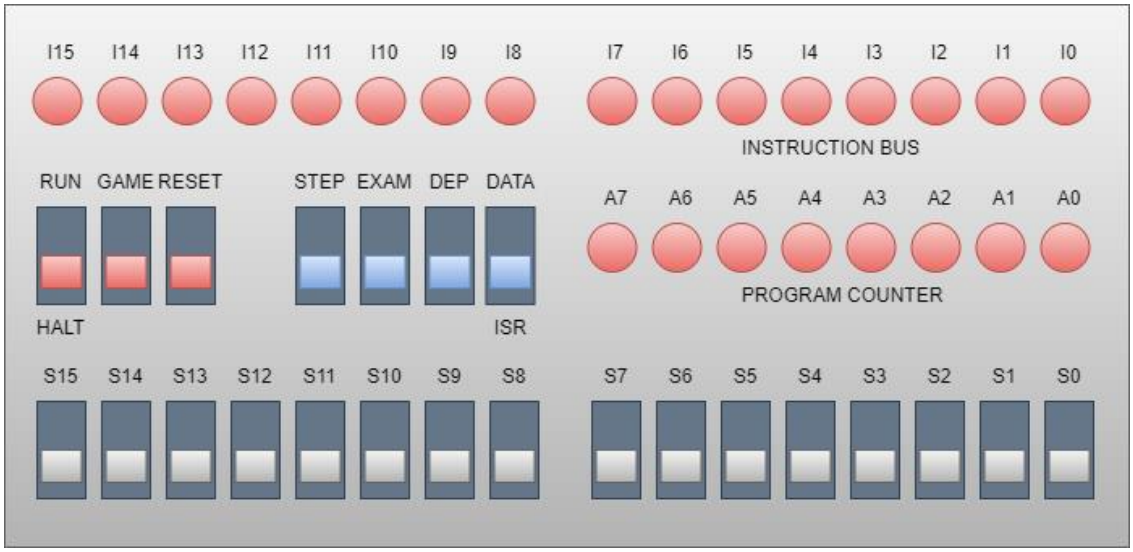


Figure 70 - User Panel Design

The red switches are used in the normal operation of the computer. The blue switches are specialty switches used for debugging, setup, and learning. The white switches are the normal input switches used in other designs.

The current location of the program counter is shown with green LEDs in the middle right. The top red LEDs show the current instruction line output by the code memory.

The PCB front panel design is shown in Figure 71, with each switch labeled.

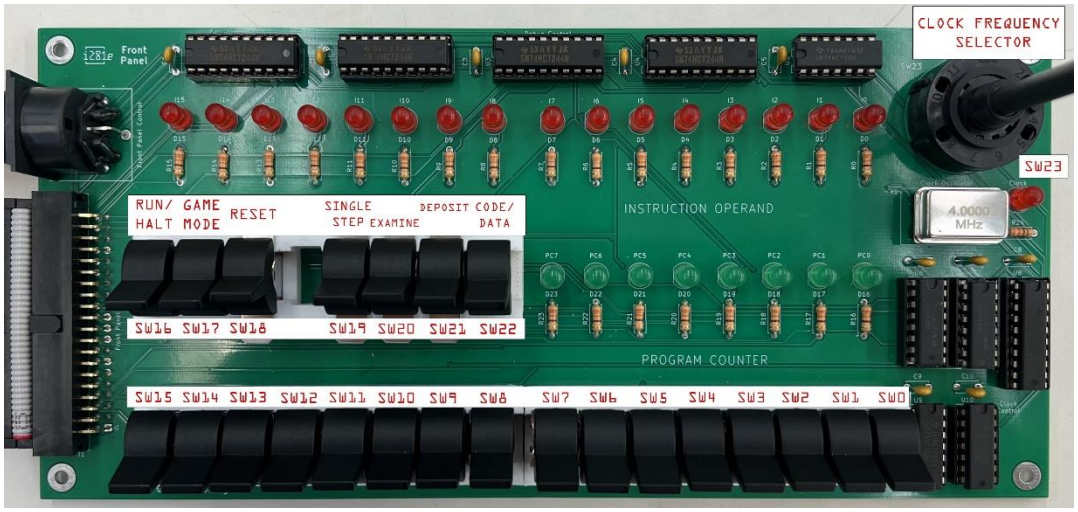


*Figure 71 – **Front** Panel PCB*

Switches 0-15 are the input switches to the i281e CPU. These are used to input data into the computer manually. They are utilized when manually writing programs to RAM, when the program is expected to input like in pong, and when using the debug switches to change the location of the program counter (this will be talked about later).

Switch 16 controls the clock; kind of like the clock's enable switch. This works by allowing the clock frequency from the clock oscillator to propagate. Although other processes like the step switch pulse the clock, this stops the i281e from running. When turning the i281e CPU on, the computer should be in the halted state.

Switch 17 controls the game mode signal. This is a pseudo control signal that affects the operation of the video card. Normally, the video card reads the data memory and converts the data from binary to the decimal seven-segment equivalent; switch 17 is low. However, when switch 17 is active, the video card reads each bit in the data memory as an individual LED of the display. This is utilized in some programs, i281 PONG, and banners.

Switch 18 is the reset line for the i281e CPU. The reset signal is sent to the rest of the i281e CPU when activated. The reset signal clears the program counter, the data bus, the compact flash register, and reset the UART.

Switch 19 performs a single step in the i281e CPU by pulsing the clock single once. The i281e should be in halt mode before using the single step switch to ensure stability. Single step is a helpful functionally to have and use when understanding how the process is working, allowing all the LEDs to be seen before an instruction is done.

Switch 20 is used for examine functionally. This allows the location of the program counter to be changed. When examine is active, the instruction bus is overridden with jump instruction; the eight lower switches are passed to the offset bus in the program counter. This allows for the location of the next location of the program counter to be set to any value using the input switches. The program counter's LEDs can be used to quickly tell where the program counter will go after releasing the switch. The i281e CPU needs to be in halt mode for stability.

Switch 21 is deposit and switch 22 is used as the code/data memory selection for deposit. These switches are used to load programs into the RAM chips and load numbers into data memory. When the switch is active, the instruction bus is over with the INPUTC/INPUTD instruction, and the switches are sent to the RAM chips and data memory input mux. After the switch is released back to the lower position, the instruction is executed and loaded into the corresponding location. The program counter is also increased by one in preparation for the next input. Switch 22 is in the lower position when writing to code memory and in the higher position when writing to data memory. These write locations are also located at

different points in memory; RAM addresses start at 0x80, and data memory starts at 0x00. Examine will allow the i281e to move through these addresses for writing to and afterward when starting the program loaded into RAM.

The clock frequency selector is a 12-point rotary switch that allows one to choose from the 12 selected frequencies. The different frequencies available with a 4 MHz oscillator are shown in Table 12. The process shown be halt state before switch frequency to ensure stability.

Position	Frequency
12	2 MHz
11	1 MHz
10	250 kHz
9	62.5 kHz
8	31.25 kHz
7	7.81 kHz
6	1.95 kHz
5	244 Hz
4	61.04 Hz
3	7.63 Hz
2	1.91 Hz
1	954 mHz

Table 12 - Clock Frequencies

7.4 I/O Ports

For the i281e CPU, there are several different types of ports. Some are used to connect the Front Panel and others are used to output data to other devices. From Figure 72, we can see five different highlighted locations. The Front Panel Bus (40Pin) and the Front Control Bus (6DIN) are the two cables that connect the CPU to the Front Panel. See Figure 71 in the Front Panel Usage section to see how those two cables connect to the Front Panel on the left side. The Compact Flash Bus (40Pin) allows for a compact flash input adapter to connect to the board. From here we can load data into the system. The Serial I/O ports allow for RS-232 cables to connect the system. Additionally, not highlighted in Figure 72, you can see a USB-C cable sticking into the CPU. The USB-C is another method for inputting programs into the system. There is an Expansion Bus (24Pin) and power output bus which allows for input and output to newly created cards in the future. Additionally, there are also breakout pins, which the next section covers.

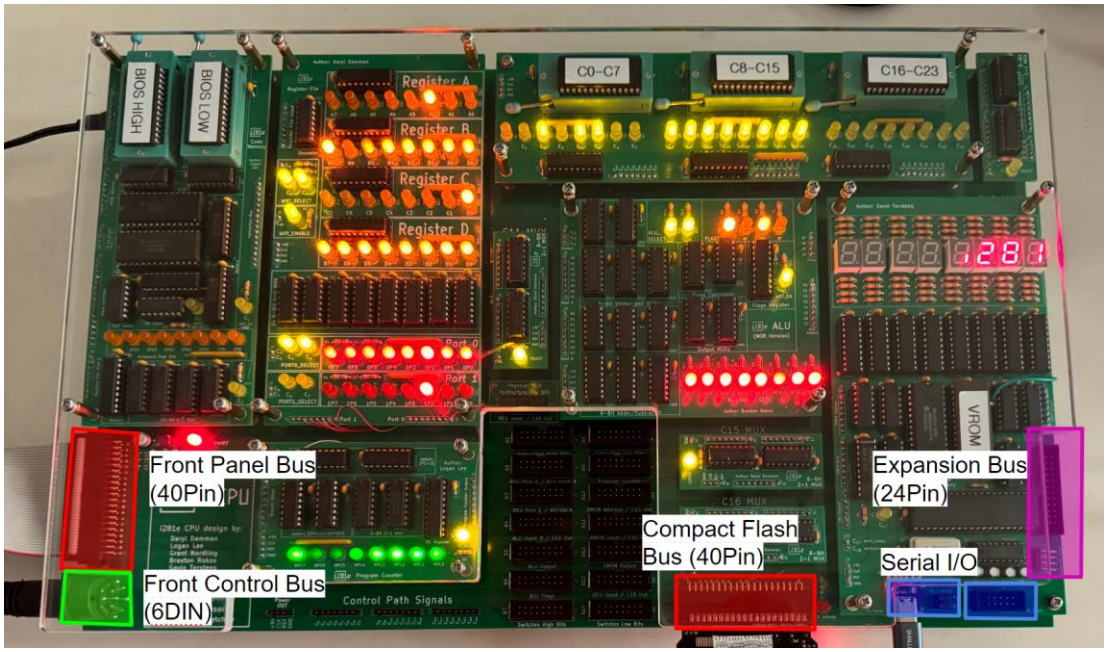


Figure 72 - IO Ports of the i281e CPU

7.5 Breakout Pins

On the Mainboard in the bottom middle, there are 14 breakout female headers that can be used to connect breadboards to the i281e CPU. A specialized cable must be created that connects to those 16-bit female breakout header to the breadboard using another 16-bit male header. Figure 73 and Figure 74, below shows how the breakout headers are labeled and laid out on the i281e CPU. Each of the headers are labeled with a Reference Designator "BX", X being a number 0-14.

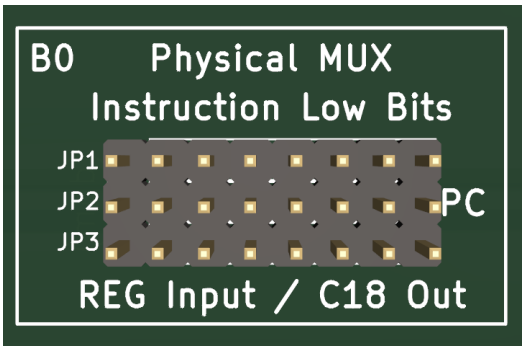


Figure 73 - B0 Physical MUX Pinout

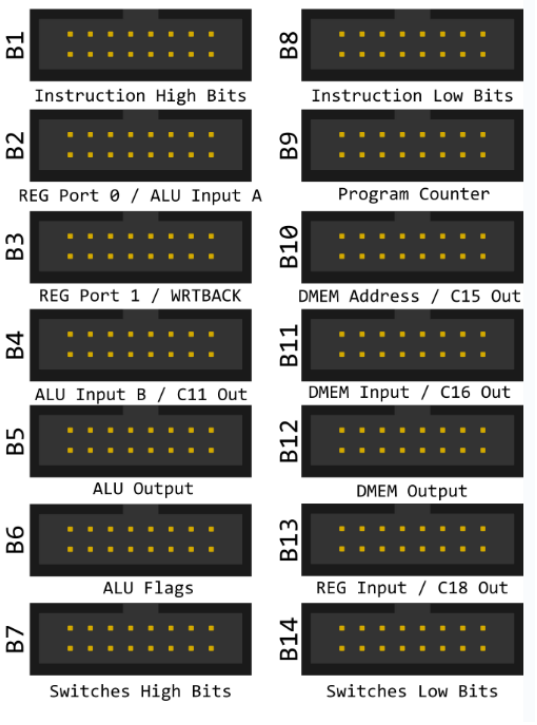


Figure 74 - B1-B14 Breakout Pins

Figure 75 shows the pin layout for both the headers on the ribbon cable. The ribbon cable header that has the 16P, attaches to the breadboards. The ribbon cable header that has the notch in the middle on the longest side for the PCB headers. Pin 0 is always on the side with the red wire.

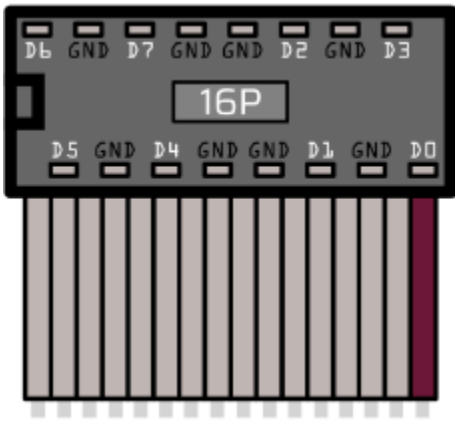


Figure 75 - Pin Layout for both i281e CPU and Bread Board Headers

Table 13 - Breakout Pins Explained below shows the Reference Designator, the labeled name on the CPU, and what its data path really means. Table 13 - Breakout Pins Explained below shows those paths more visually. Here you can follow the data path from and to components, so one could appropriately attach breadboards to the signals they desire. One

could effectively remove one of the modules and use these headers to create a bread board version of the module and test it with the full system.

Reference Designator	Labeled Name	Meaning
B0	REG Input / C18 Out	Data Path from Physical MUX to Program Counter
B1	Instruction High Bits	Data Path from CMEM Instruction to Control Table
B2	REG Port 0 / ALU Input A	Data path from REG Port 0 to ALU Input A
B3	REG Port 1 / WRTBACK	Data path from REG Port 1 to WRTBACK module and C16 MUX Input 0
B4	ALU Input B / C11 Out	Data path from C11 MUX Output to ALU Input B
B5	ALU Output	Data path from the ALU Output to C15 MUX Input 0
B6	ALU Flags	Data path from ALU Flags to Control Table
B7	Switches High Bits	Dath path for the Switches High Bits to the Writeback Module
B8	Instruction Low Bits	Data path from CMEM Instruction Low Bits to Physical MUX, C11 MUX Input 1, and C15 MUX Input 1
B9	Program Counter	Data path from Program Counter to CMEM
B10	DMEM Address / C15 Out	Data path from the C15 MUX Output to the DMEM Address, C18 MUX Input 0, and CMEM
B11	DMEM Input / C16 Out	Data path from C16 MUX Output to DMEM Input
B12	DMEM Output	Data path from DMEM Output to C18 MUX Input 1
B13	REG Input / C18 Out	Data path from C18 MUX Output to Physical MUX and Register File
B14	Switches Low Bits	Data path from Switches Low Bits to Writeback Module and to C16 MUX Input 1

Table 13 - Breakout Pins Explained

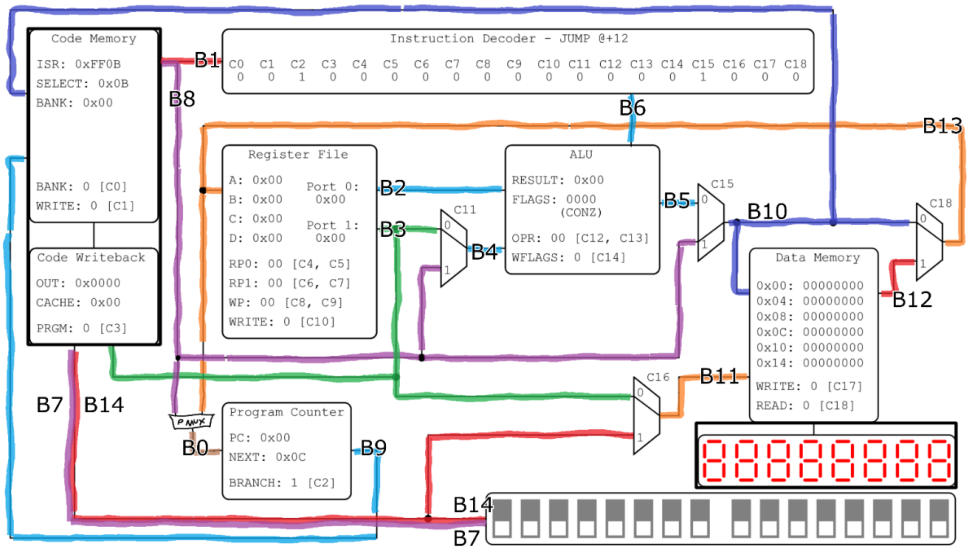


Figure 76 - Breakout Pins Data Paths Visualized

The i281e CPU also has pins that can be tapped to input the +5V rail, clock, reset, ground, and control signals to the bread boards. If the module that you are removing from the system requires a clock signal and reset signal, you can expect to tap all four signals on the left of Figure 77. If the module also requires control signals 12, 13, and 14, you could use a wire to bring them from the CPU to the bread boards.

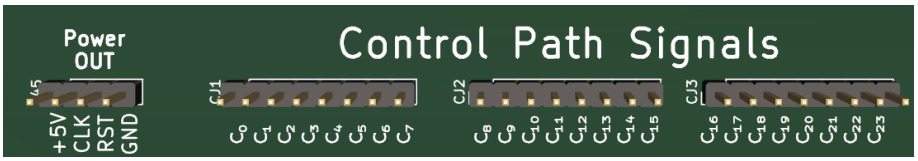


Figure 77 - Power and Control Line Output Headers

Figure 78 shows one of these tests. Here, we removed the C11 MUX, and replaced it with a breadboard MUX. We used the +5V and GND line from the "Power OUT" headers. Since it was the C11 MUX, we used the C11 control signal. As for the 16-bit headers, we used B3, B4, and B8 which are the inputs and outputs of the C11 MUX.

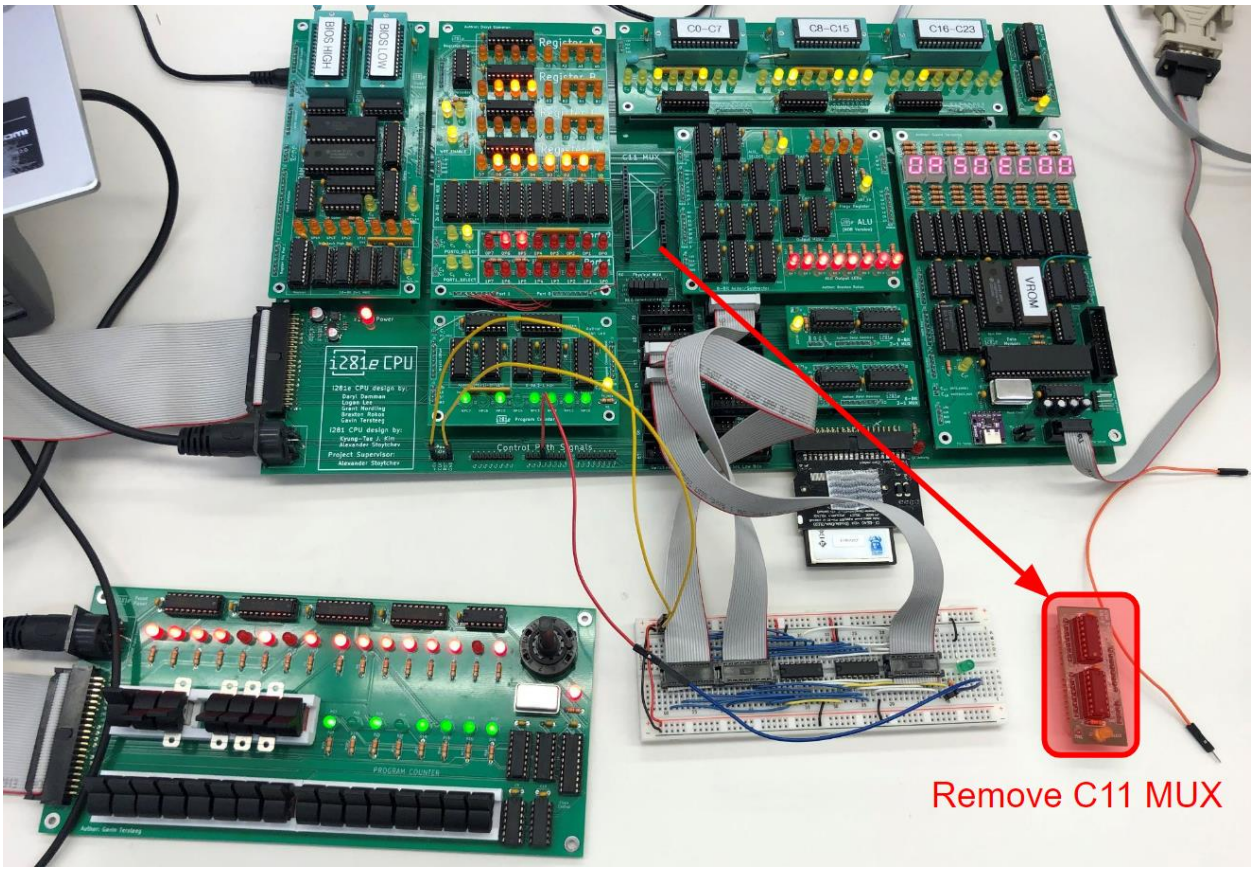


Figure 78 - Bread Board to PCB Implementation

7.6 Running Programs

A terminal must be connected to the computer to load programs into RAM for running them on the i281e CPU. "Tera Term" is a good option for communicating with the i281e CPU. Plugging in a USB cable to the USB-C port located on data memory and a computer with allow the user to see the DOS boot screen (DOS is explored more in later sections). To ensure the DOS functionality works, the process should run at maximum speed. After selecting the fasted clock frequency, toggle the reset switch and the i281e CPU will load the DOS menu.

The compact flash has different blocks for programs ready to run, from zero to nine. Use "CD" followed by a number to navigate to the corresponding block; for example, "CD 4" with allow the terminal to see into block four. Use "DIR" to list the programs stored in the corresponding block. This will print out a list of programs that can be called; by navigating with these commands, a program can be found. To run the program, type the block it is located in, semicolon, followed by the name of the program. For example, "4:BANNER" will run the banner.sv program from the fourth block of the compact flash.

A halt command can also be called by using an ampersand symbol before "$4:BANNER". This is helpful if the program needs a slower clock speed to be visible than the boot process.

After calling the program with a halt command, the program loads into RAM like normal, but it will not start executing the RAM instructions after loading them. Instead, the i281e CPU will be in a loop in ROM. This loop can be exited after setting the clock to a lower frequency, by halting the i281e CPU and then using the examine switch. Simply toggle the switch with no switches active and the i281e CPU starts executing the program selected.

7.7 Maintenace

7.7.1 Finding Bad Parts

When the machine is not working as expected, it might not be the program currently running. There is a chance that a component overheated or something short circuited. If you smell something like plastic burning or see a plume of smoke, it is probable that a component burnt out. It is difficult to find the component when you are not sure where the smoke came from. Look for physical defects on all components such as burn marks, melted plastic, or misshaped components. Hover your hand over the board and feel for residual heat coming from components. If you see or feel anything from those components, chances are that those are the ones that got damaged. If you have access to a thermal camera, you can use it to see which components are getting hot when the i281e machine is on. This data can be useful to prevent the destruction of components.

Additional testing can be done via a multimeter or oscilloscope on components such as resistors, capacitors, transistors, and diodes. You could run a continuity test on them to see if they broke down into an open-circuit meaning that no current can flow through the components. You can also test to see if the parts are working as expected and have the properties that their datasheets say they should have.

One could also do some isolation testing. Remove modules from the board and individually test components and output pins. If a module does not act as it should, do more detailed testing on the individual components of that module. Once you find the issue parts, you need to switch them out for a new chip of the exact same part number, package size, and pin layout.

7.7.2 Switching Out Parts

Throughout the life of the i281e machine, parts will become old, damaged, or simply breakdown. It is an occurrence that is simply bound to happen, so you need to know how to fix it. When a component isn't working properly and it needs to be replaced, there are two methods that can be used: prying out the part or unsoldering components.

Prying out the part includes using a flathead screwdriver or some alternative object to get underneath a chip that sits in a dip package. From this, you can get leverage on the chip and

pry it up. This method is used primarily for the 74HCT logic chips that are in the dip packages. We can see in Figure 79 on the left there is a logic chip in the dip package and on the left, there is the part with the chip removed.

Figure 79 - Dip Package with and without Chips

Unsoldering components requires either a soldering wick or heat gun. To use the soldering wick, press the wick on top of the solder you wish to remove. Press your soldering iron into the wick at the same location as that solder. As the wick heats up, it will start to draw the solder into itself. If the solder is stubborn, add additional solder to the problem area and repeat the process of using the wick. Sometimes adding additional solder helps the wick absorb the solder.

If you have access to a heat gun, you can use that to unsolder several pins at once. This method is better suited for components with multiple pins or leads close to each other. The solder used on the PCB boards melts at around 700°F, so ensure that you set the heat gun to the right temperature. To remove components, aim the heat gun at the solder until it starts to shine like a mirror. At this point, the solder is in a liquid form. Use pliers to pull the component out of the other side. Repeat as necessary. Be careful not to aim the heat gun at yourself. Do not touch the component with your fingers as the component may be hot.

Once you have the component removed, you can install the new one. For parts that are inserted into the dip packages, firmly press the chip into the package. Ensure that the semicircle at the center of the dip package aligns with the semicircle on the chip. If the chip is not going in easily, use the pin adjuster or some pliers to make 90° angles on all the pins pointing downwards. Then retry pressing the chip into the package.

For parts such as a resistor or LED, simply slot the part into the board. Then flip the board over onto the opposite side and find the location where the part is located. With your soldering iron, apply heat to the solder pad. The solder pad is a ring of silver colored material at the base of the through-hole connection. Once the solder pad is heated, press your solder into the pad and pin it until it melts. If you cannot get the solder to melt, put a dot of solder onto the tip of your soldering iron and try again. Once the solder is melted, it should attach to

both the pad and the pin. If done correctly, the solder should take up the whole pad and not be rounded like a ball (See Solder Joints 1 and 3-7 on Figure 80). If the pad can still be seen, reflow the solder and heat the pad until it covers the whole pad. If the solder is not visibly attached to the pad and pin all the way around the pin, add more solder until it covers the whole pad. Solder Joint 2 on Figure 80 is lacking some solder. The joint should still function correctly but would be more reliable if there was more. If the solder joint looks like a ball, you have put too much solder on the joint (See Solder Joint 8 in Figure 80). Consider applying a solder wick as discussed above to remove If the solder joint looks darkened, you applied too much heat and the flux in the solder is burned out. This type should still work but look for the issues mentioned previously.

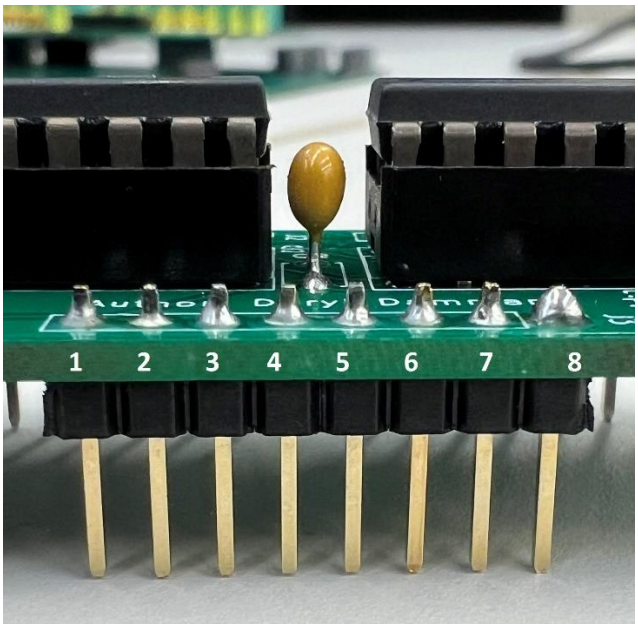


Figure 80 - Different Types of Solder Joints

7.8 Expansion Capabilities

Since the i281e CPU works as a computer, there is a world of opportunities to expand the functionality of the design. Keeping this in mind, the i281e CPU hardware team thought ahead and left a 24-bit expansion bus and power bus on the bottom right side of the DMEM Module. This location allows for newly designed modules to be placed on the right side of the machine and connected. The signals that are sent out are the DMEM output bus, read & write lines, control lines C17 & C18, lower bits of the DMEM address line, clock, and

demultiplexer select line using the DMEM address line. Figure 81 below shows the expansion bus and power outputs located on the bottom right side of the DMEM Module.

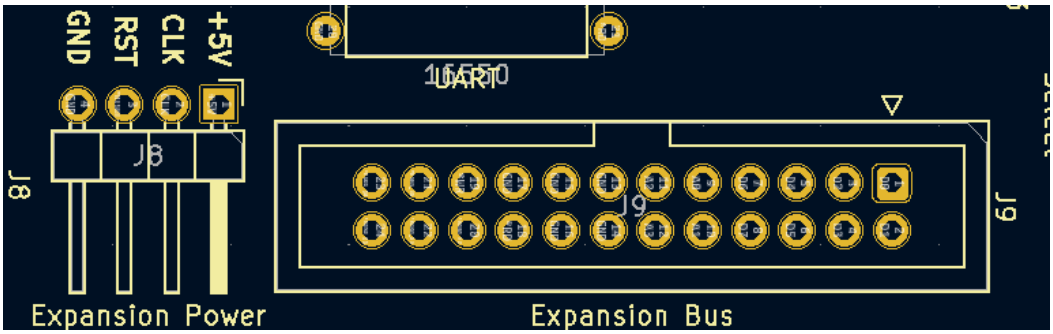


Figure 81 - Expansion Bus and Expansion Power

During the design and development of the i281e CPU, some of the group members created some passion projects to expand the project beyond the original requirements. One of these projects was the Mega I/O Expansion PCB card. This card allows the use of a printer, speakers, PS2 keyboard, and PS2 mouse. Figure 82 shows the card. This card barely scratches the surface of the expandability of the i281e CPU.

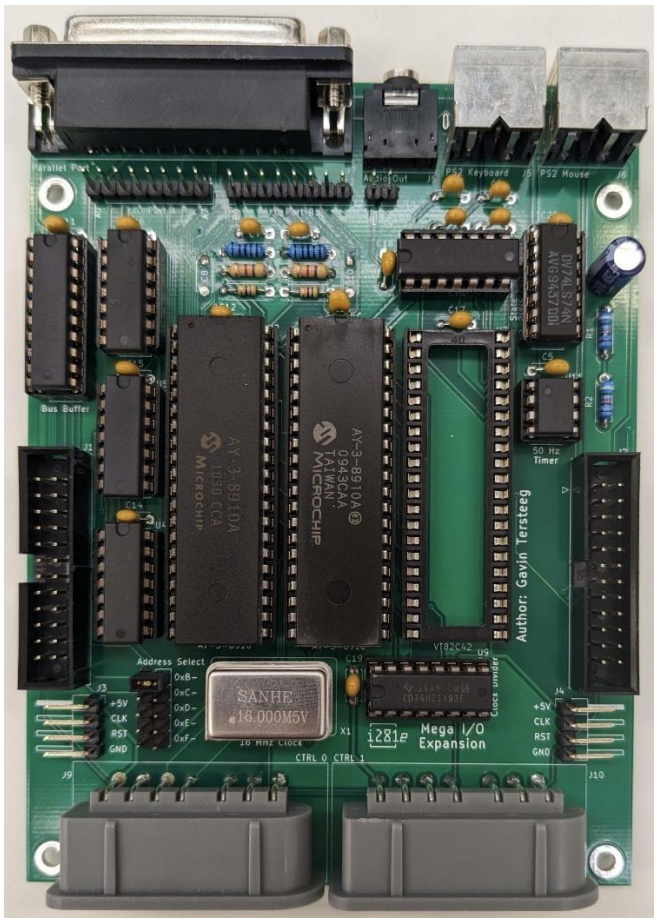


Figure 82 - Mega I/O Expansion Module

8 Appendix B

Intermediary Design Iterations

- Versions considered before client's specifications have changed
- Versions considered before learning more about the project
- Versions that resulted in failure to achieve specifications, etc.
- Describe why they were scrapped/revised

8.1 Design Decisions

8.1.1 TTL Chip Selection

As this project aims to implement the i281 processor using discrete logic, 74-series chips are utilized to handle the computational burden. There are three chip choices for TTL-style logic for the 74-series: LS, HC, and HCT.

The previous project utilized 74LS chips. These are valid chips to be utilized; however, these are harder to come across and less performant compared to the following options. Alternatively, there are 74HC chips. These have a high-power requirement and do not match the pin-out of 74LS chips.

Our team decided on using 74HCT chips. These are cheaper, easier to obtain, and match the pin-out of 74LS chips. If we need to prototype a CPU component while waiting for our 74HCT chips, there are spare 74LS chips courtesy of the previous project. This will allow us to produce a breadboard component sooner and we can swap out for the 74HCT chip when testing.

8.1.2 Design Layout

For the breadboard design, it should be known that an 8-bit 2-1 multiplexer circuit takes up an entire breadboard (63 columns, 8 rows). This is the smallest CPU component that will be built. Since there are approximately 8+ multiplexers, we already consume 8+ breadboards. For larger CPU components, at least 3 breadboards may be needed up to 8. To compensate, 50 breadboards were ordered for this project.

Given the context, the i281 processor on breadboard is a colossal project that will cover an entire table. As seen with the previous project, they attempted to compact the processor by constraining it to a single breadboard surface with wires going everywhere. This is not sustainable for maintenance and debugging.

Rectifying this problem, all breadboard CPU components will be built into "islands." These islands must be connected through a ribbon cable which will act as a bus data line (up to 8 bits). This resolves our issue of maintaining a large processor by manually separating the components and requiring loose coupling. Unfortunately, we will need to use more breadboards and more table space. The trade-off has been considered acceptable regardless.

8.1.3 Read/Write of RAM

The original i281 design was built using Verilog for FPGA devices. As such, certain constraints could be ignored One of these major constraints was reading and writing to

RAM in a single clock cycle. Nearly all RAM on the market is considered "single-port" RAM. The data lines for reading and writing are on the same pins of the IC. Given this restriction, reading and writing to RAM must be performed in two separate clock cycles which conflicts with the requirements of the project. This only occurs if RAM passes an instruction that would alter the contents of RAM in the same cycle.

Mitigating this dilemma started with separating what is permitted during BOOT and RUN. The primary decision was to remove the ability to modify RAM while in RUN mode. While self-modifying code is permitted in certain scenarios on other processors, said processors may take more time to operate single instructions thus avoiding single cycle read/write. For our design, BOOT will run from ROM exclusively and write user programs (and any additional instructions) to RAM. Finally, RAM will operate as a "read-only" while we are currently in RUN mode.

8.2 Boot Sequence Discussion

When a processor is powered on, it doesn't go from a powered-off state to running instantaneously. A process exists to begin execution of user operations as soon as the device is ready. Unfortunately, simulations of the i281 processor ignore this circumstance and immediately begin executing instructions without hesitation.

Rectifying this problem spun about the idea of banking/partitioning example programs into the ROM and allowing for quick handling of example programs and provide a way for users to manually enter programs by hand if so desired. This solution was not perfect for the client as the RAM chip was heavily underused.

The current solution is to include a "hard disk" (separate ROM, hereby User ROM) per the request of the client that will contain any premade programs that can be loaded into RAM at boot. The boot sequence will start with reading the first five (SW4-SW0) switches to determine what program to load in from User ROM into RAM. If a program needs to load another section into RAM, an instruction will jump back to ROM and load the next sequence.

8.2.1 Dismissal Reasons

While this in-theory would work, the overarching design is now different in the final design starting from breadboard implementation's second design iteration. The "hard disk" of the final design is now the compact flash memory. This is intertwined with DMEM data paths and new opcodes to be fully accessible and controlled by software, not hardware.

9 Appendix C

Other Considerations

Any miscellany you deem important, what you learned, anything funny, anecdotes from your project experience.

9.1 Design Definitions

In addition to all the terms and acronyms listed throughout the document, these definitions or additional acronyms are intended for this appendix specifically and do not have any meaning in the grander document.

9.1.1 Breadboards

A "breadboard island" is a CPU component that is completed on breadboard and is isolated from other CPU components. The only way for logic/data to leave these boards is from bus data lines.

Alternatively, the name "module" can be used to describe the same element; however, this is typically used for the PCB implementation.

9.1.2 Scoping

There are two types of "scope": global and local.

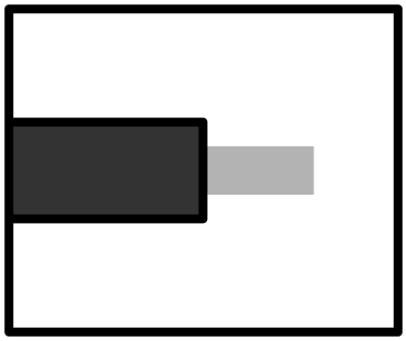
Global scope, or globally scoped, mandates the requirements across all breadboards or PCBs of a particular component, wire, etc. For example, if a wire is globally scoped, it should be the only wire color used for a particular case in all scenarios regardless of breadboard or PCB.

Local scope, or locally scoped, does not mandate adhesion to a particular requirement but gives strong preference on to how it should be used in the project. For example, if a wire is locally scoped, it should be left to the discretion of a board upon which usage it should fall under. Once more, it is strongly recommended that the suggestions given be used unless otherwise needed.

9.2 Breadboard Wiring Standards

9.2.1 Wire Color Scheme

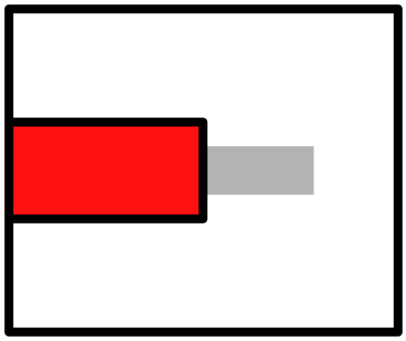
Through the usage of a 6-color wire spool set and two separate colors, the following wire colors must generally represent the appropriate usage on a breadboard.



Black

Purpose: ground (GND)
Scope: global

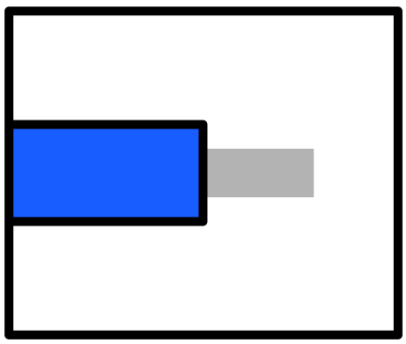
Standard color choice. Black is specifically reserved for ground only. This should be used for jumpers to the ground line in a breadboard or across breadboards.



Red

Purpose: +5V power line
Scope: global

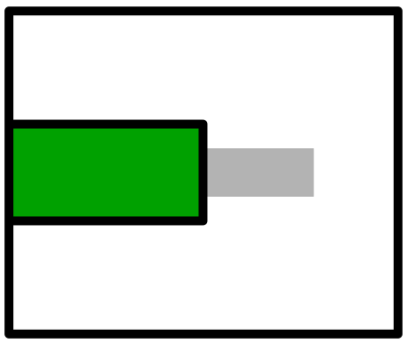
Standard color choice. Red is specifically reserved for power (+5V) only. This should be used for jumpers to the power line in a breadboard or across breadboards.



Blue

Purpose: data line, primary
Scope: global

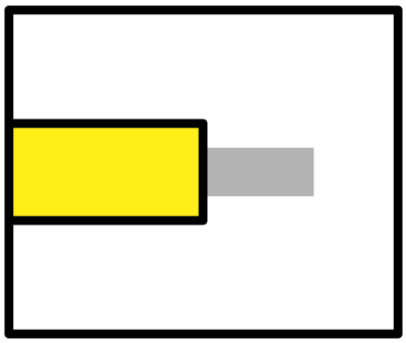
In all other cases where information being transferred across a component isn't an address or control line, the line is considered data.



Green

Purpose: address line
Scope: global

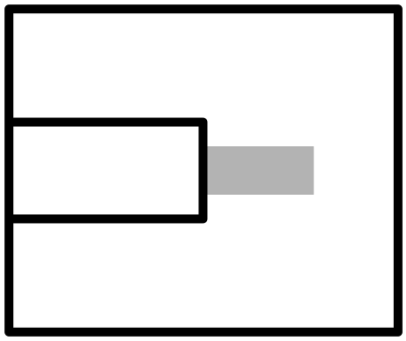
Processor addresses will be illustrated as green wires when known. When uncertain, use appropriate alternate color (data line(s)).



Yellow

Purpose: control line, clock pulse
Scope: global

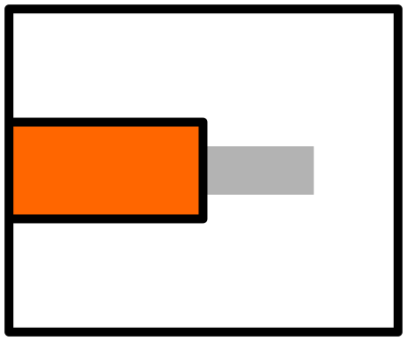
Control line jumpers are indicated using yellow. A separate cable may be used to carry more than one line but must be indicated as such.



White

Purpose: data line, secondary
Scope: local

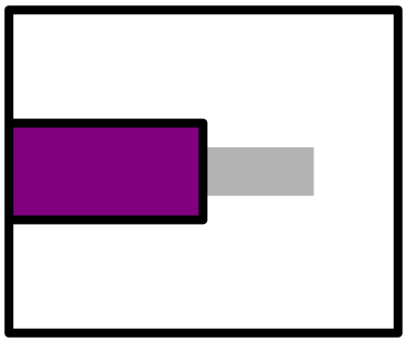
In cases where a significant number of blue wires would be used for data, white wire may be used to help alleviate eye strain.



Orange

Purpose: data line, ternary
Scope: local

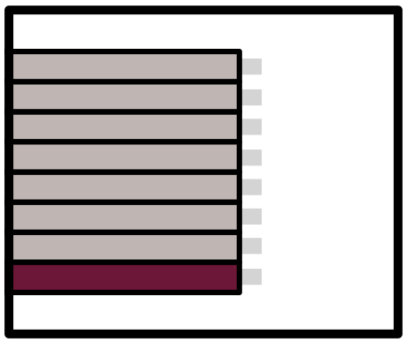
Typically used as a data line in the Register File, the wire may be used for the clock line in debugging to differentiate between control lines.



Purple

Purpose: data line, ternary
Scope: local

Typically used as a data line in the Register File, the wire may be used for control lines.



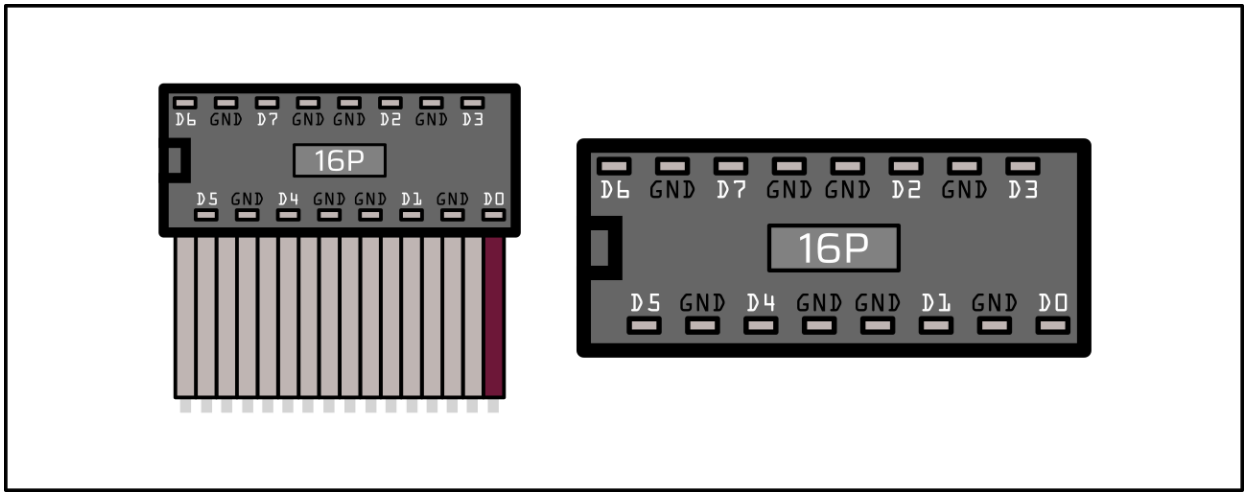
Gray Ribbon Cable

Purpose: bus data line

Scope: global

Data transfer between breadboard islands. Measure between islands and cut with reasonable slack.

9.2.2 Connector for Bus Data Lines



Pin Description

D0-D7	Data lines 0-7
GND	Ground

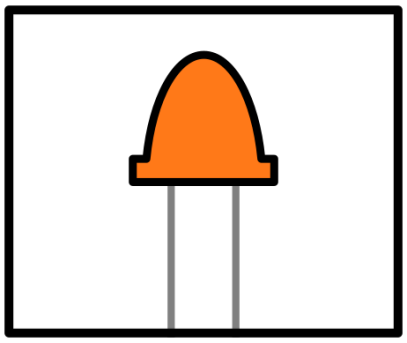
The primary indicator for the connector is that the zeroth bit line (non-gray wire) must be on the right side of the connector when facing the i281 processor. Alternatively, when viewing the breadboard implementation, the non-gray side must point/face toward the right side of the machine.

If uncertain what direction the breadboard implementation should face, ensure the positive line is north (from perspective) and all numbers are readable. From there, east is the direction the cables should face when plugged into breadboards.

9.3 Visualization Standards

An initial 6 colors were purchased for the i281 CPU. There are two sets of standards enacted for the project: breadboard and PCB. Ultimately, the PCB standard is considered "de facto" and the breadboard standard remains a legacy component to better understand the original implementation. Future implementations, either breadboard or PCB, should exclusively use the PCB standard.

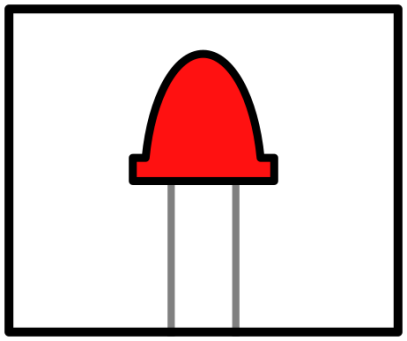
9.3.1 Breadboard Visualization



Orange

Purpose: register storage
Scope: global

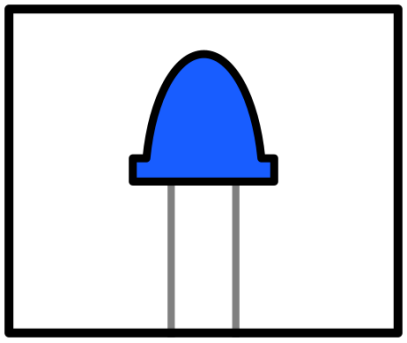
Meant to exclusively be used for representing what is currently stored in the individual registers of the Register File.



Red

Purpose: instruction representation
Scope: local

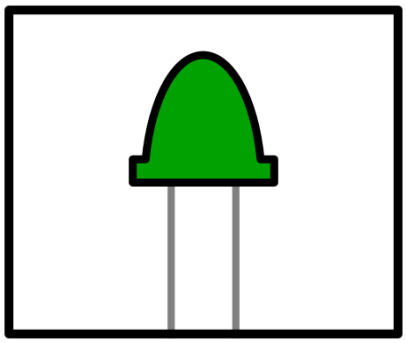
Can be used across several modules to represent a range of bits from the instruction output.



Blue

Purpose: open
Scope: local

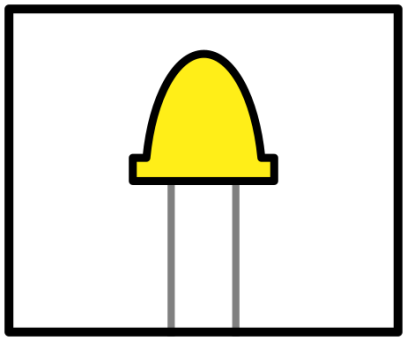
No reservation for this color. Primarily used as the output of the ALU.



Green

Purpose: program address
Scope: global

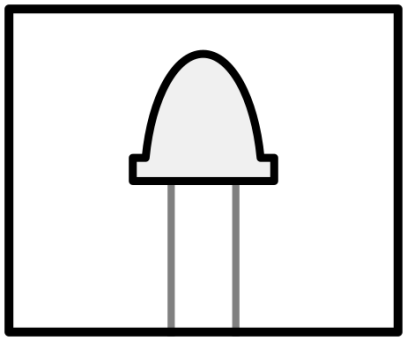
Green is the representation of the program address across all CPU components.



Yellow

Purpose: control line indicator
Scope: global

A yellow LED is meant to indicate the assertion of a control line at both the source (instruction decoder) and destination (multiplexer, ALU, etc.)

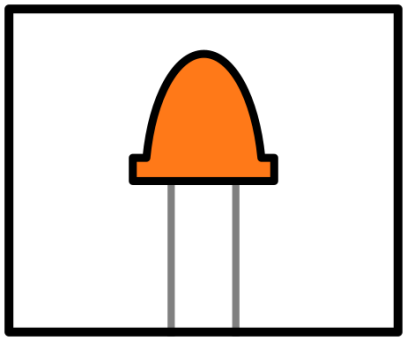


White

Purpose: flags register
Scope: global

Meant to visualize a given flag value for the ALU.

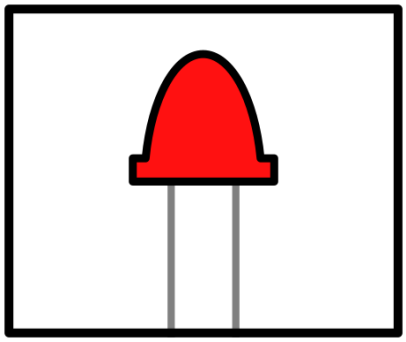
1.2.2 PCB Visualization



Orange

Purpose: latching signals
Scope: global

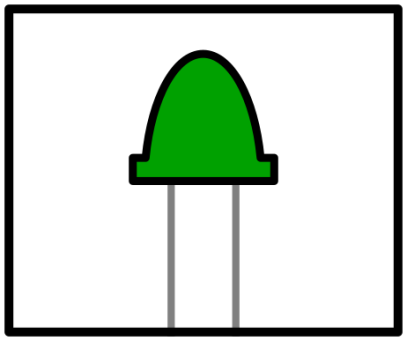
Represents a 4-bit or 8-bit value stored using a latching register (usually octal latch) chip. Heavily represented for the Register File.



Red

Purpose: transient signals
Scope: global

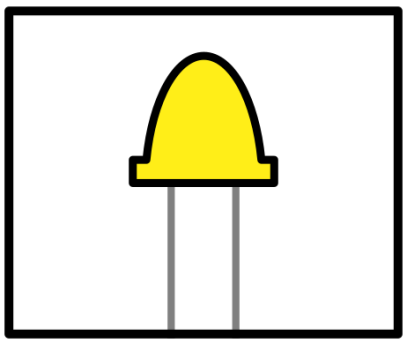
Represents signals that are not meant to remain upon the following instruction. This color may also be used as an indicator light, primarily power or CF activity.



Green

Purpose: program address
Scope: global

Exclusively used for the program counter and front panel. Represents the current or next program address in relation to ROM/RAM address.



Yellow

Purpose: control path signal
Scope: global

Represents a control path signal in the respective module or on the control table module. A lit LED confirms the assertion of a control path signal.

10 Appendix D

Software and Hardware Resources

10.1 Software Resources

The technical lead, Gavin Tersteeg, has compiled a large collection of homebrew software (including DOS/281) on his personal GitHub repository: https://github.com/tergav17/i281-dev.

10.1.1 Third-Party Software

While the project is fully capable of standing on its own, several pieces of software were used in the designing process of the project alongside day-to-day debugging and development.

Software	Usage	Link
GitLab	Version Control	https://about.gitlab.com/
Google Drive	Documentation	https://drive.google.com/
Inkscape	Branding/Diagrams	https://inkscape.org/
LibreCAD	CAD	https://librecad.org/
KiCad	Schematic/PCB Development	https://www.kicad.org/
Microsoft Office365	Documentation/Budgeting	https://www.office.com/
Python	Programming Language	https://www.python.org/
Tera Term	Serial Terminal for i281e	https://teratermproject.github.io/index-en.html
Visual Studio Code	Script Programming	https://code.visualstudio.com/
Xgpro	EEPROM Programming	http://www.xgecu.com/EN/download.html

10.2 Hardware Resources

Additional hardware resources regarding the mounting hardware and DXF files are located on Daryl Damman's personal GitHub repository: https://github.com/brandtdamman/i281-hardware. The repository houses all the necessary DXF files for water/laser cutting polycarbonate/acrylic panels for both breadboard and PCB implementations.