# i281 CPU Hardware Implementation

## Final Report

**Team Number:** sddec22-20
**Client:** Dr. Alexander Stoytchev
**Advisor(s):** Dr. Alexander Stoytchev
**Team Members/Roles:**
**Alex Kiefer**/Group Lead and Lead Hardware
**Joseph De Jong**/Hardware Testing, Communications, and Reporting
**David Vachlon**/Purchasing Lead, Software Development, and Hardware Testing Lead
**Saffron Edwards**/Lead Software, Software Testing, and Website Maintenance
**Patrick O'Brien**/Documentation, Progress Reports, and Meeting Notes

**Team Email:** sddec22-20@iastate.edu
**Team Website:** https://sddec22-20.sd.ece.iastate.edu/

**Revised:** 12/07/2022

# Table of Contents

# Introduction

The College of Electrical and Computer Engineering at Iowa State offers programs that help its students learn about and work with different types of hardware and software. However, Iowa State and many other universities around the country lack material that can effectively build a connection between hardware and software with undergraduate students. Professor Alexander Stoytchev seeked to bridge that gap by developing an easily understandable hardware implementation of a simple computer processor known as the i281 CPU. This senior design project aimed to prove that the design worked by using breadboards to implement each subcomponent of the computer and resolve any issues that were discovered.

# Project Context

When this project was proposed, two projects had already implemented different variations of the i281 CPU. Professor Stoytchev had originally developed an FPGA variation of this CPU with his graduated TA, Kyung-Tae Kim. They introduced this iteration of the CPU in the Fall of 2019 for Professor Stoytchev's CprE 281 class to examine at the end of the semester. The idea behind this was to link all the information taught throughout the class into a real-world applicable example. While this provided a chance for students to examine the capabilities of the CPU when loaded onto an FPGA, it was not visual enough to provide a good example of how programs were executing on the underlying circuitry.

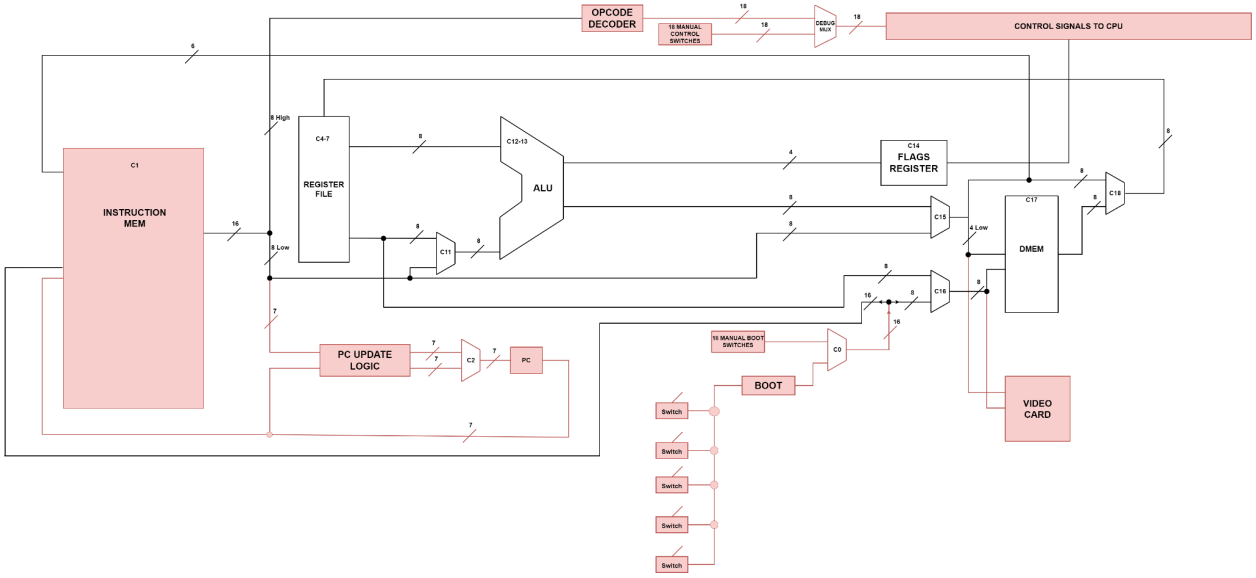
This led to the senior design project of fall 2020/spring 2021, where the i281 CPU was implemented as a simulator attached to Professor Stoytchev's CprE 281 website. This simulator proved to be a good tool for visualizing the CPU. However, it was not a tangible representation of the CPU that students could examine in real life, which led to the proposal for our senior design project. The idea, this time, would be to have both a visual and tangible representation of the CPU for students to examine, ideally bridging the gap between software and hardware in the computer engineering curriculum.

# Team Members and Contributions

| Team Member | Role in Project | Major Contribution |
|---|---|---|
| Alex Kiefer | Hardware Lead | Datapath Implementation |
| Joseph De Jong | Administrative Lead | Documentation |
| Saffron Edwards | Software Lead | Website |
| Patrick O'Brien | Hardware Design | Module Designs |
| David Vachlon | Interface Design | EEPROM Software |

# Project Plan

The initial plan for this senior design project was to directly implement the i281 CPU, as shown in the FPGA design, using discrete THT chips. To do this, we planned to implement the CPU in a modular fashion. This was because by developing each module independently, we could test each smaller module more easily, and ideally that should lead to us being able to simply connect all modules. As our requirements changed, though, we ended up having to implement more than just the initially planned i281 CPU. We had to add a user interface as well as a display system for users to see output, and hardware to boot up this system. This increased complexity caused our initial project plan to have to expand to incorporate this newly expanded scope. We will outline this modular implementation in the following sections, as well as our design challenges.



# Intended Users

Dr. Stoytchev and his students would ideally benefit from the eventual successful implementation of this project. This is because this will be used to enhance the learning experience of future undergrads in the ECpE program here at Iowa State.

Additionally, the eventual success of this project will hopefully cause more students to feel they have mastered the material, which should help them in future courses. It is because of this that Professor Stoytchev pursues better implementations of his CPU to better serve his students education.

This project, when fully functional, will be used in the lab setting of CprE 281 to teach students how the different aspects of digital logic that they have learned all semester coalesce into a single, functional computing unit. They will interact with it by running programs through it, examining the output, and identifying the different components comprising this computing system.

# Functional Requirements

- Able to run all instructions in the instruction set architecture (ISA). This includes general register instructions, immediate operations, and branches and jumps.
- DIP and SPDT switches must be used as user inputs to the CPU.
- Seven segment displays and LEDs must be used to visualize information propagating through the CPU.
- The design must include a variable speed clock for normal operations and a manual clock to step through operations.
- The CPU must be reprogrammable.

# Non-Functional Requirements

- The final project must be easy to visualize, allowing users to see where data flows in the CPU.
- Must function similarly to the previous versions of the i281 CPU, including the software simulation and FPGA implementation.
- The CPU must be easy to reproduce, allowing for mid-level manufacturing.

# Project Milestones

| Milestones | Stretch Goals |
|---|---|
| Design and build all subcomponents (ALU, Register File, Memory, Displays, Program Counter, Opcode Decoder) | Create PCB modules |
| Connect every subcomponent to create a datapath | Create PCB design based on the breadboard CPU |
| Test the datapath for basic functionality and continuity | Review and order PCB prototypes |
| Test datapath for functionality with PONG and Bubble Sort | Solder PCBs, combine and test for system bugs |

# Technical Standards

IEEE Standards:

1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation: We determined that we needed this standard to assist us in testing the hardware that we built as well as Professor Stoytchev's ISA (instruction set architecture) on this CPU. Building hardware via breadboards is a complex and difficult task, so this requirement was important in producing a working implementation.

162-1963 - IEEE Standard Definitions of Terms for Electronic Digital Computers: We needed this standard to ensure that the PCB we build for students uses the correct names for its components. Because this will be used in a learning environment, it must be properly labeled so the students can more easily understand it.

# Bill of Materials

We compiled a list of all the materials needed to complete the breadboard implementation of the i281 CPU. We found an estimated cost for the entire project was greater than $1200. This was greater than our initial cost estimates, but a fair amount of the high cost was due to the worldwide chip shortage. Specialized chips that were necessary to complete the project while following our client's specifications would normally cost around $4 per chip, but the chip shortage caused those chip costs to jump into the $20-30 price range. Additional costs for the project included the used breadboards, power supplies, and shipping costs.

**Cost Estimate of Breadboard Implementation:**

| i281 CPU | Breadboard Implementation Costs | | |
|---|---|---|---|
| | Cost Per Module | Quantity | Total Cost |
| ALU | $46.79 | 1 | $46.79 |
| Program Counter | $41.31 | 1 | $41.31 |
| Register File | $57.75 | 1 | $57.75 |
| Bus Mux | $38.93 | 9 | $350.37 |
| Video Card | $41.82 | 1 | $41.82 |
| Instruction Mem | $95.83 | 1 | $95.83 |
| Boot | $95.83 | 1 | $95.83 |
| Imem | $95.83 | 1 | $95.83 |
| Controls | $95.83 | 1 | $95.83 |
| Shipping Costs | NA | NA | $138.20 |
| Wire Costs | $25.00 | 5 | $125.00 |
| Power Supply | $50.00 | 1 | $50.00 |
| | | | **$1,234.56** |

# Individual Module Implementations

## *ALU Module*

The ALU module is a component in the i281 CPU which performs all of the necessary computations needed in arithmetic instructions. It can perform 4 different operations, including addition, subtraction, shift left by 1, and shift right by 1.

### Parts List:

| i281 CPU | | ALU | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantity | Total Cost |
| 1 | Capacitor | $0.95 | C1 | 1 uF | CP_Radial_D5.0mm_P2.00mm | 1 | $ 0.95 |
| 2 | THT 1/4W Resistors | $0.05 | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, | 220 Ohm | R_Axial_DIN0204_L3.6mm_D1.6mm_P | 19 | $ 0.95 |
| 3 | LED | $0.15 | D1, D2, D3, D4, D5, D6, D7, D8, D9, | LED | LED_D3.0mm | 49 | $ 7.35 |
| 4 | 74LS157 Quad 2-1 | $0.93 | U5, U6, U7, U12, U14, U15 | 74LS157 | DIP-16_W7.62mm | 6 | $ 5.58 |
| 5 | Quad 2 Input XOR | $0.74 | U1, U2 | 74LS86 | DIP-14_W7.62mm | 2 | $ 1.48 |
| 6 | 4 Bit Binary Adder | $2.20 | U3, U4 | 74LS283 | DIP-16_W7.62mm | 2 | $ 4.40 |
| 7 | Quad 2 Input OR | $0.92 | U8, U9 | 74LS32 | DIP-14_W7.62mm | 2 | $ 1.84 |
| 8 | Binary Inverter | $1.09 | U10 | 74LS04 | DIP-14_W7.62mm | 1 | $ 1.09 |
| 9 | 4 Bit D Flip Flop | $1.96 | U11 | 74LS173 | DIP-16_W7.62mm | 1 | $ 1.96 |
| 10 | Tri 3 Input NAND | $0.97 | U13 | 74LS10 | DIP-14_W7.62mm | 1 | $ 0.97 |
| 11 | SPDT Switch | $0.76 | SW1, SW2 | SW_SPDT | SWITCH_SLIDE_3hole | 2 | $ 1.52 |
| 12 | 10 Pin IDC Connector | $3.08 | J2, J5 | 2x5 Female | IDC-Header_2x05_P2.54mm_Vertical | 2 | $ 6.16 |
| 13 | 4 Pin Molex Mega-Fit | $4.06 | J6, J6 | 2x2 Female | Molex_Mega-Fit_76829-0104_2x02_P5 | 2 | $ 8.12 |
| 14 | 2.54mm Header Pins | $0.67 | J1 | 1x2 Male | PinHeader_1x02_P2.54mm_Vertical | 1 | $ 0.67 |
| 15 | 2.54mm Header Pins | $0.67 | J3 | 1x4 Male | PinSocket_1x04_P2.54mm_Vertical | 1 | $ 0.67 |
| 16 | 10 Pin IDC Connector | $3.08 | J4 | 2x5 Female | IDC-Header_2x05_P2.54mm_Vertical | 1 | $ 3.08 |

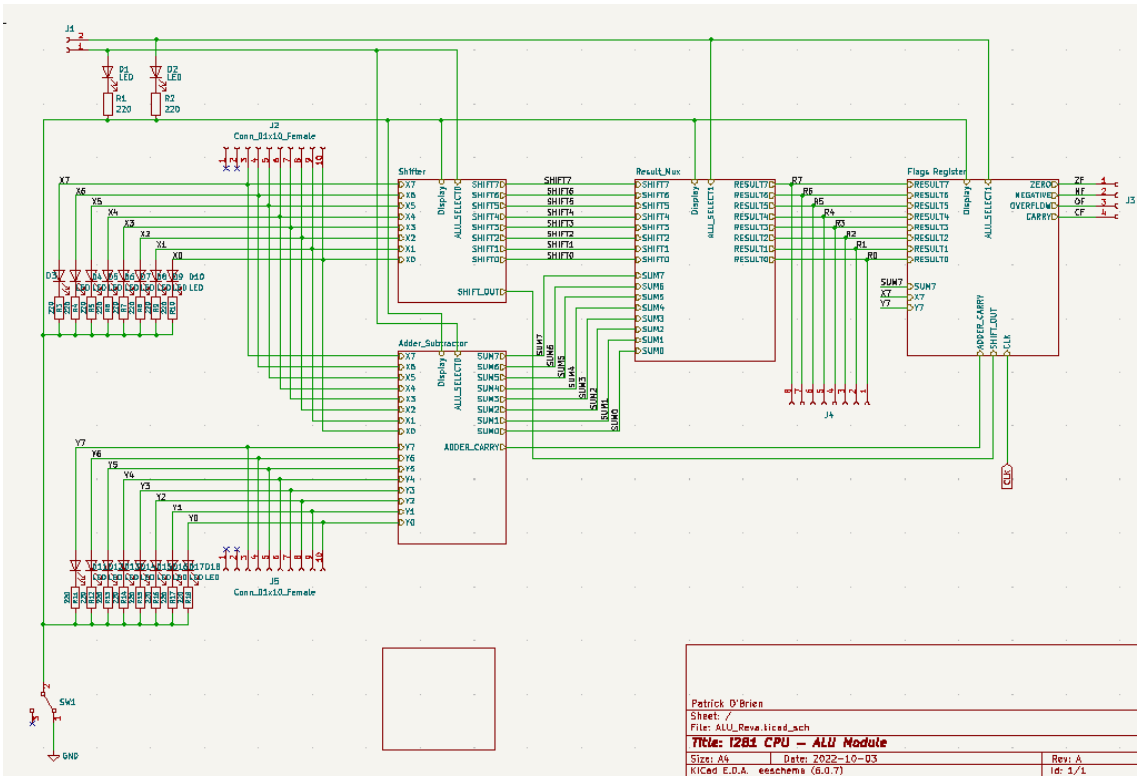
### Schematic:

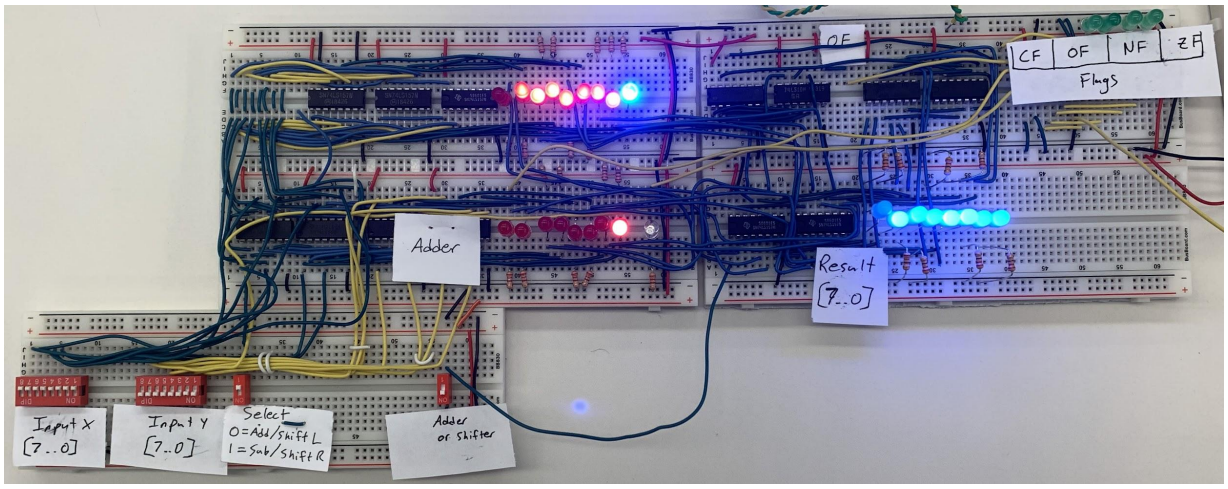
*Inputs*:

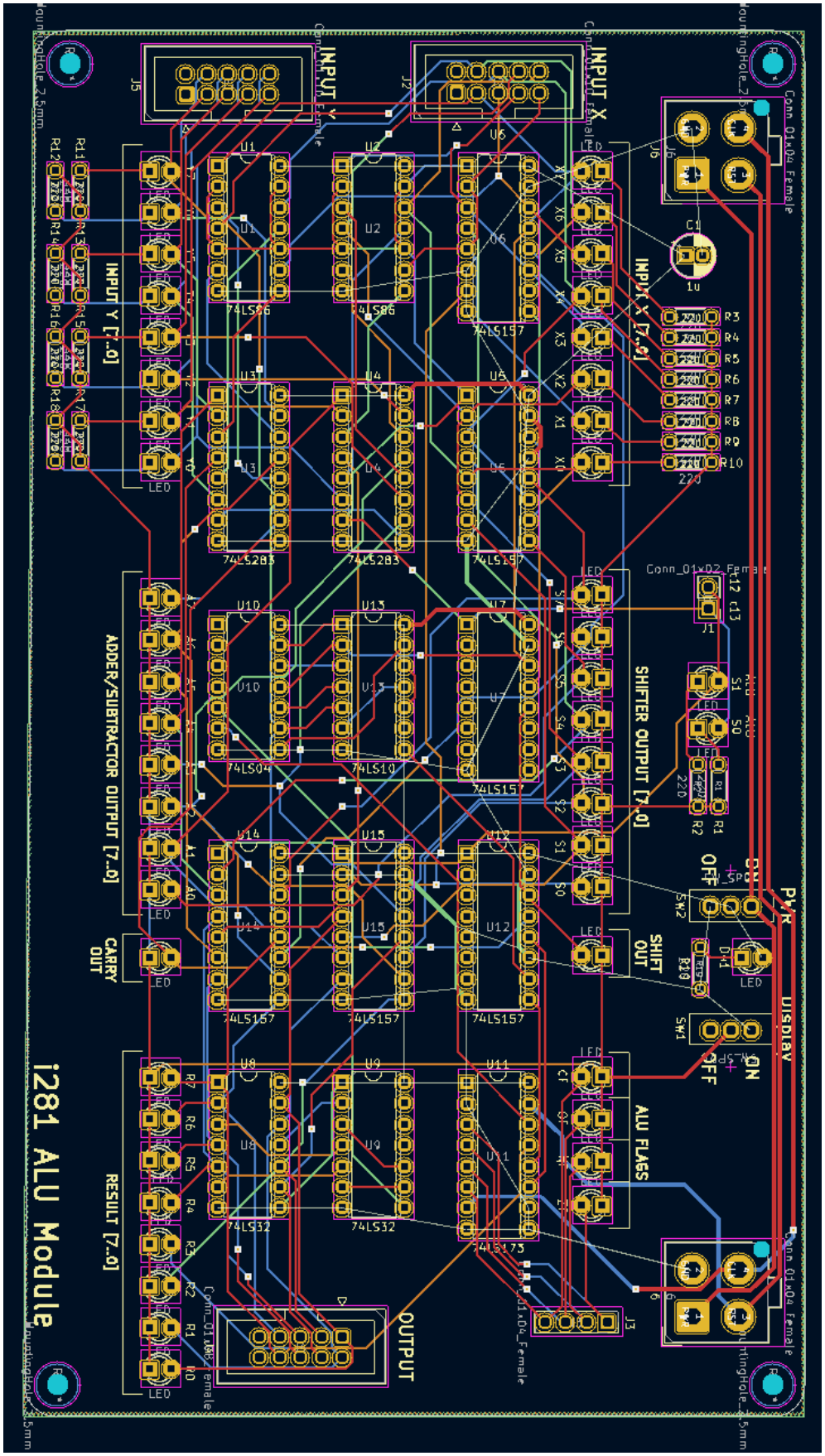
- X [7:0]
- Y [7:0]
- ALU Select 1
- ALU Select 0

*Outputs*:

- Overflow Flag
- Carry Flag
- Negative Flag
- Zero Flag
- Result [7:0]


Shifter
Adder_Subtractor
Result_Mux
Flags Register
Conn_01x10_Female
Patrick D'Brien
Sheet: /
File: ALU_Revs.kicad_sch
Title: I281 CPU – ALU Module
Size: A4
Date: 2022-10-03
KiCad E.D.A. eeschema (6.0.7)
Rev: A
Id: 1/1


OF
CF OF NF ZF
Flags
Adder
Result
[7..0]
Input X
[7..0]
Input Y
[7..0]
Select
0=Add/ShiftL
1=Sub/ShiftR
Adder
or Shifter


i281 ALU Module
INPUT Y
INPUT X
INPUT Y [7..0]
INPUT X [7..0]
ADDER/SUBTRACTOR OUTPUT [7..0]
SHIFTER OUTPUT [7..0]
RESULT [7..0]
CARRY OUT
SHIFT OUT
ALU FLAGS
DISPLAY
OUTPUT
PWR
OFF
ON
74LS86
74LS283
74LS157
74LS04
74LS10
74LS32
74LS173
LED
220
1u

## *Program Counter*

The program counter is a clock-based parallel read/write register. Data is stored as a 7-bit value and is updated every clock cycle. By default, the PC increments by one every clock. However, during a jump or branch instruction, the PC value can be incremented to the desired value.

## Parts List:

| i281 CPU | | Program Counter | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantit | Total Cost |
| 1 | Resistor | $0.05 | R1, R2, R3, R4, R5, R6, R7, | 220 | R_Axial_DIN0207_L6.3mm_D | 33 | $1.65 |
| 2 | LED | $0.15 | D1, D2, D3, D4, D5, D6, D7, | LED | LED_D3.0mm | 47 | $7.05 |
| 3 | 4 Bit Adder | $2.20 | U1, U2, U3, U4 | 74LS283 | DIP-16_W7.62mm | 4 | $8.80 |
| 4 | 74LS157 Quad 2-1 | $0.93 | U5, U6 | 74LS157 | DIP-16_W7.62mm | 2 | $1.86 |
| 5 | 4 Bit D Flip Flop | $1.96 | U7, U8 | 74LS173 | DIP-16_W7.62mm | 2 | $3.92 |
| 6 | 10 Pin IDC Connector | $3.08 | J2, J3, J5 | 2x5 Female | IDC-Header_2x05_P2.54mm_ | 3 | $9.24 |
| 7 | 4 Pin Molex Mega-Fit | $4.06 | J1, J6 | 2x2 Female | Molex_Mega-Fit_76829-0004 | 2 | $8.12 |
| 8 | 2.54mm Header Pins | $0.67 | J4 | 1x1 Male | PinHeader_1x01_P2.54mm_ | 1 | $0.67 |

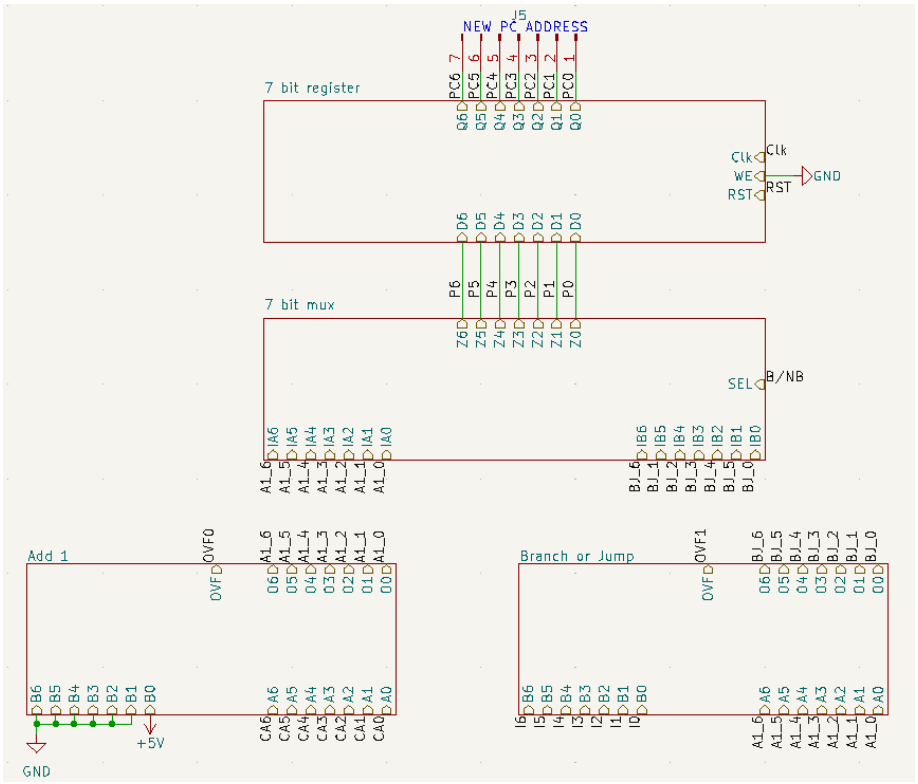
## Schematic:

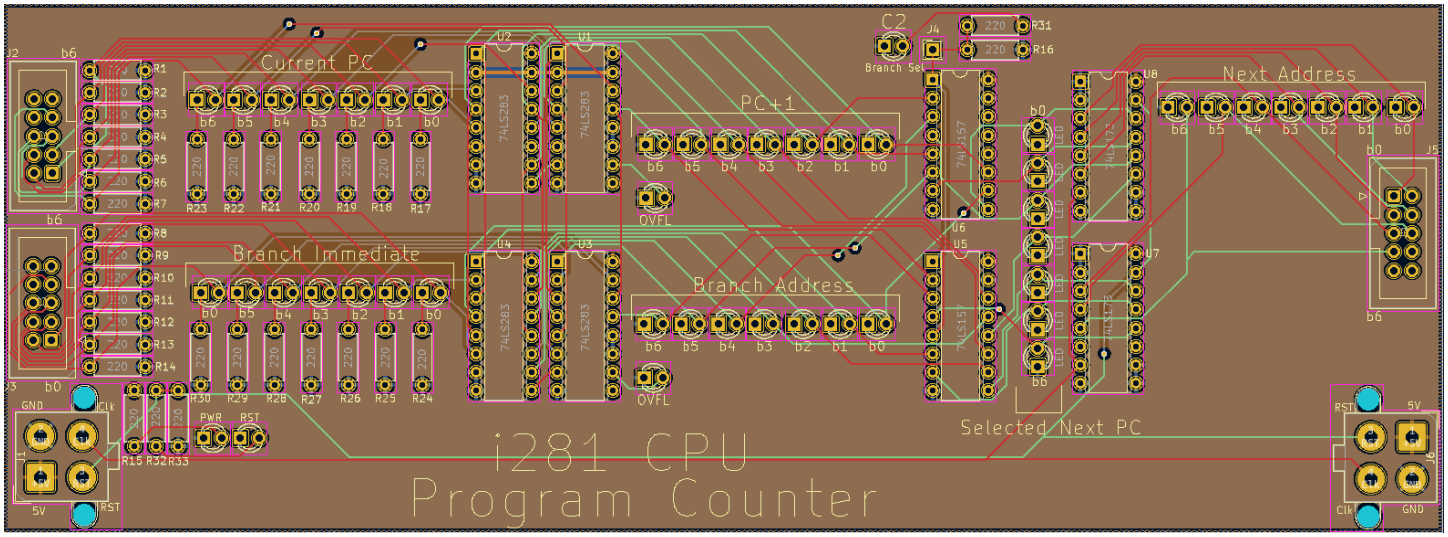
*Inputs*:

- Previous PC [6:0]
- IMEM [6:0]
- C2

*Outputs*:

- PC [6:0]


NEW PC ADDRESS
7 bit register
Clk
WE
RST
GND
7 bit mux
SEL
B/NB
Add 1
Branch or Jump
+5V
GND


Current PC
PC+1
Next Address
Branch Immediate
Branch Address
Selected Next PC
i281 CPU
Program Counter

# *Register File*

The Register File module takes in 8 bits of data (D) and stores them in one of 4 registers on every high edge of the clock pulse (synchronous write). Control bits (C8 -C9) decide which of the four registers to store the data. The data is then chosen by the 4-1 read port muxes controlled by (C4-C7) asynchronously. Other CPU modules can use the output data.

## Parts List:

| i281 CPU | | Register File BOM | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantit | Total Cost |
| 1 | Resistor | $0.05 | R1, R2, R3, R4, R5, R7, R8, R9, R10, R11, | 220 Ohm | R_Axial_DIN0207_L6.3mm_D2.5mm | 33 | $1.65 |
| 2 | LED | $0.15 | D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, | | LED_D3.0mm | 65 | $9.75 |
| 3 | 4 Bit D Flip Flop | $1.96 | U0, U1, U2, U3, U4, U5, U6, U7 | 74LS173 | DIP-16_W7.62mm | 8 | $15.68 |
| 4 | 2 Bit 4-1 Multiplexer | $0.99 | U9, U10, U11, U12, U13, U14, U15, U16 | 74LS153 | DIP-16_W7.62mm | 8 | $7.92 |
| 5 | 2-4 Decoder | $0.34 | U8 | 74LS139 | DIP-16_W7.62mm | 1 | $0.34 |
| 6 | 10 Pin IDC Connector Socket | $3.07 | J0, J5, J6, J7 | 2x5 Female | IDC-Header_2x05_P2.54mm_Vertic | 4 | $12.28 |
| 7 | 2.54mm Header Pins | $0.67 | J2, J3 | 1x2 Male | PinHeader_1x02_P2.54mm_Vertical | 2 | $1.34 |
| 8 | 4 Pin Molex Mega-Fit Socket | $4.06 | J4, J8 | 2x2 Female | Molex_Mega-Fit_76829-0004_2x02 | 2 | $8.12 |
| 9 | 2.54mm Header Pins | $0.67 | J1 | 1x3 Male | PinHeader_1x03_P2.54mm_Vertical | 1 | $0.67 |

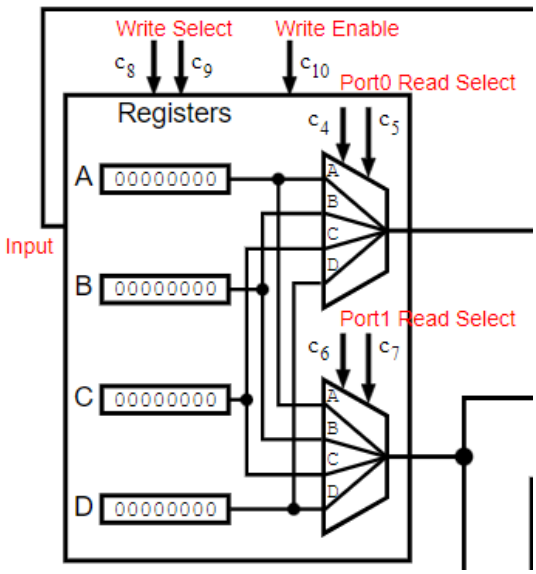
## Schematic:

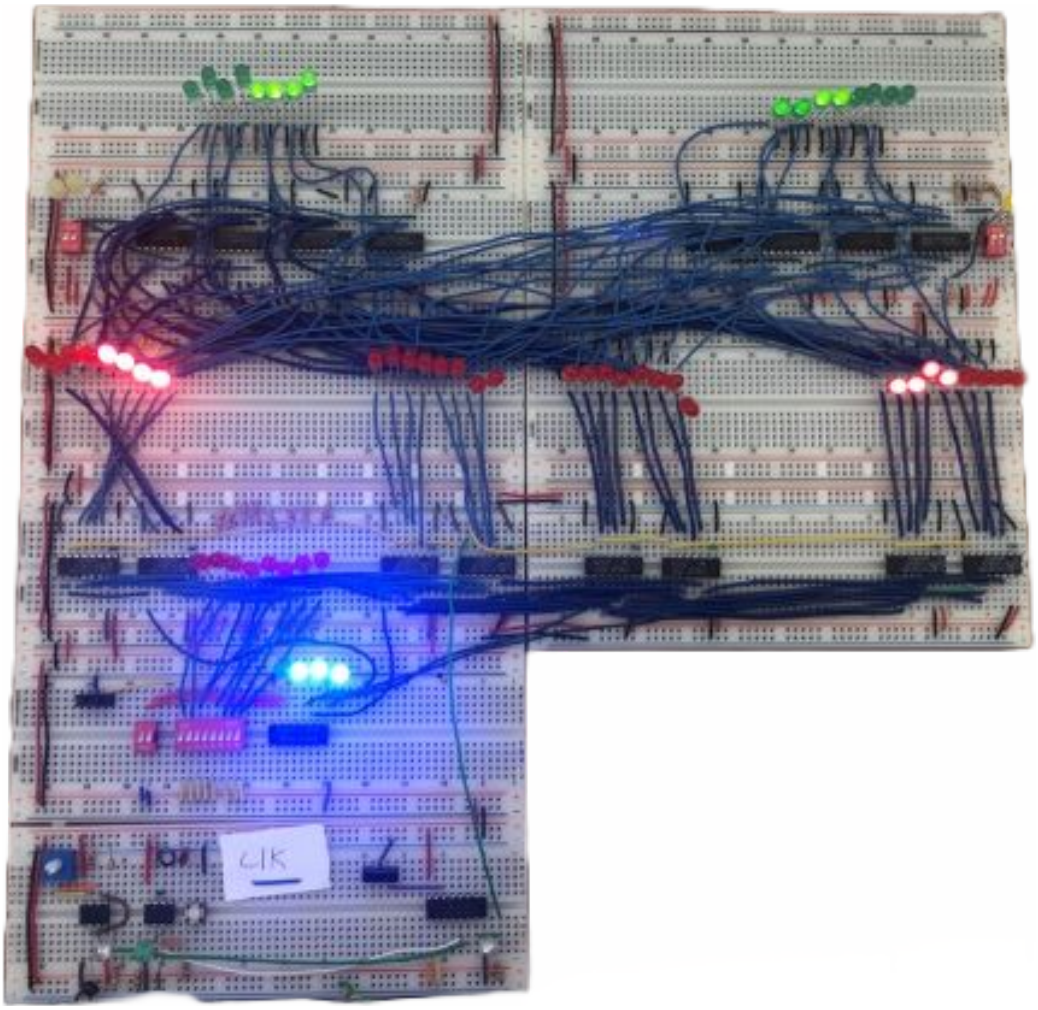
*Inputs:*

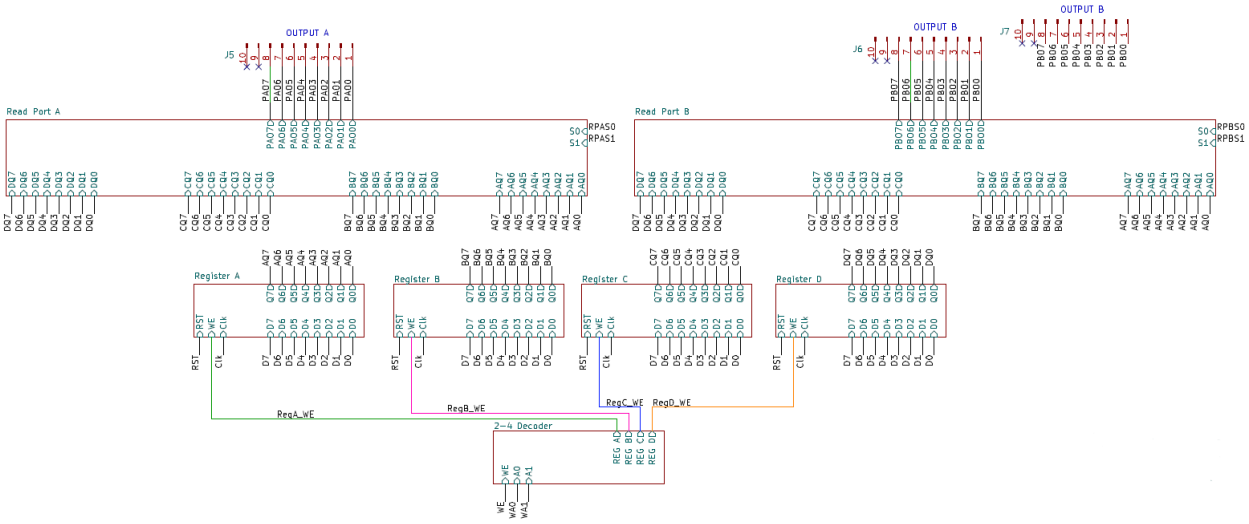
- 8 Input data bits (D)
- 2 Write Address bits (C8-C9)
- Write Enable (C10)
- 8 bit read port A (C4-C5)
- 8 bit read port B (C6-C7)

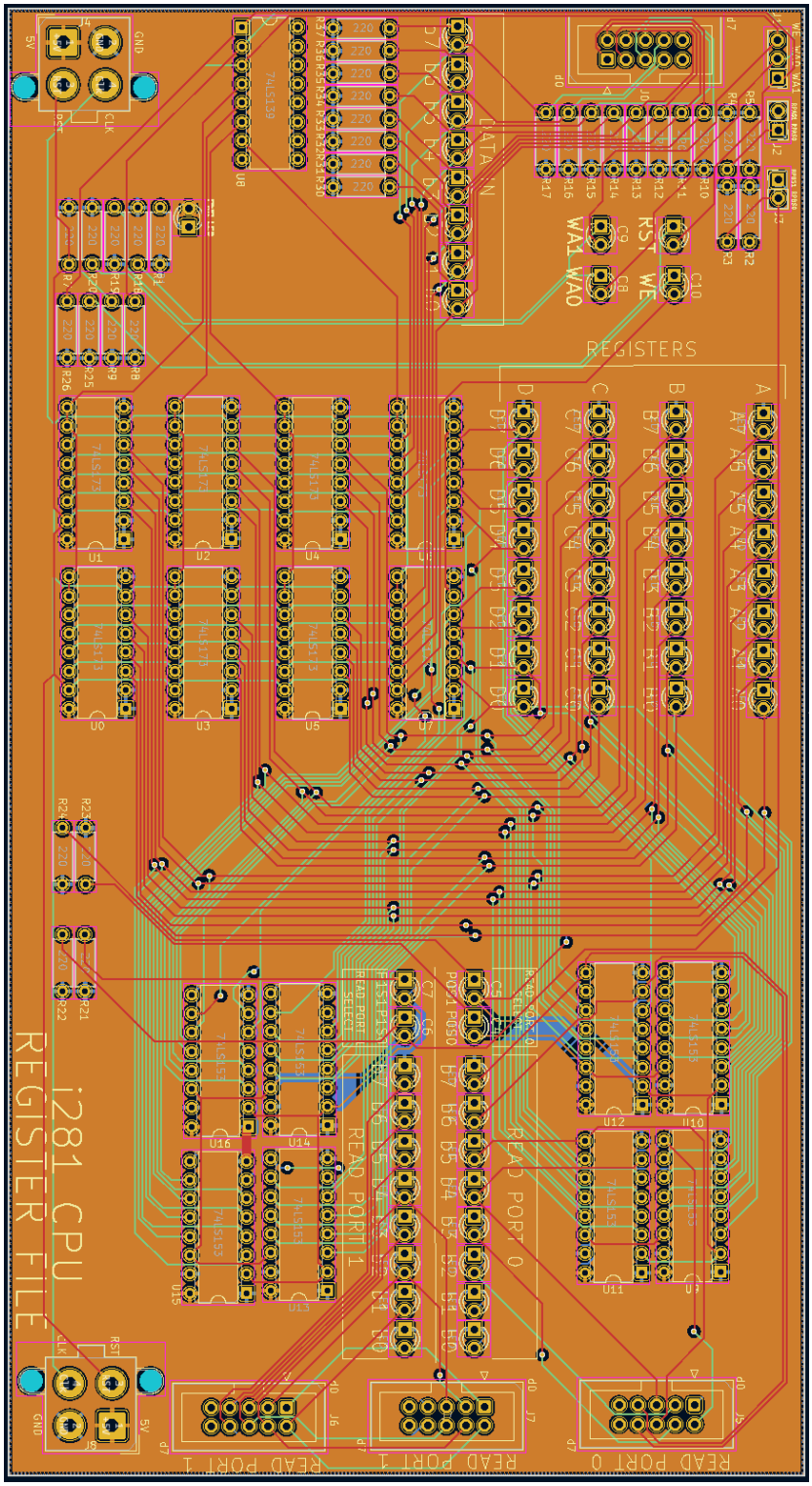
*Outputs:*

- 8-bit data from the two read ports (R1, R2)




CLK


OUTPUT A
OUTPUT B
OUTPUT B
Read Port A
Read Port B
Register A
Register B
Register C
Register D
RegA_WE
RegB_WE
RegC_WE
RegD_WE
2-4 Decoder


i281 CPU
REGISTER FILE
REGISTERS
READ PORT 0
READ PORT 1
READ PORT 1
READ PORT 0
DATA IN
WA1 WA0
RST WE
CLK
RST
GND
5V

# *Control Logic*

## Parts List:

| i281 CPU | | Control Logic | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantity | Total Cost |
| 1 | Resistor | $0.05 | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19 | 220 | R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal | 19 | $0.95 |
| 2 | LED | $0.15 | D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16, D17, D18, D19 | LED | LED_D3.0mm | 19 | $2.85 |
| 3 | 74LS157 Quad 2-1 Multiplexer | $0.93 | U2, U3, U5, U6, U7 | 74LS157 | DIP-16_W7.62mm | 5 | $4.65 |
| 4 | EEPROM | $29.00 | U1, U4 | 28C256 | DIP-28_W15.24mm_LongPads | 2 | $58.00 |
| 5 | EG1218 | $0.76 | SW1, SW2, SW3, SW4, SW5, SW6, SW7, SW8, SW9, SW10, SW11, SW12, SW13, SW14, SW15, SW16, SW17, SW18, SW19 | SW_SPDT | SWITCH_SLIDE_3hole | 19 | $14.44 |
| 6 | 10 Pin IDC Connector Socket | $3.08 | J1, J2 | Conn_01x08_Female | IDC-Header_2x05_P2.54mm_Vertical | 2 | $6.16 |
| 7 | 4 Pin Molex Mega-Fit Socket | $4.06 | J3, J5 | Conn_01x04_Female | Molex_Mega-Fit_76829-0004_2x02_P5.70mm_Vertical | 2 | $8.12 |
| 8 | 2.54mm Header Pins | $0.67 | J4 | Conn_01x02_Female | PinHeader_1x02_P2.54mm_Vertical | 1 | $0.67 |
| 9 | 2.54mm Header Pins | $0.67 | J6 | Conn_01x18_Female | PinHeader_1x18_P2.54mm_Vertical | 1 | $0.67 |

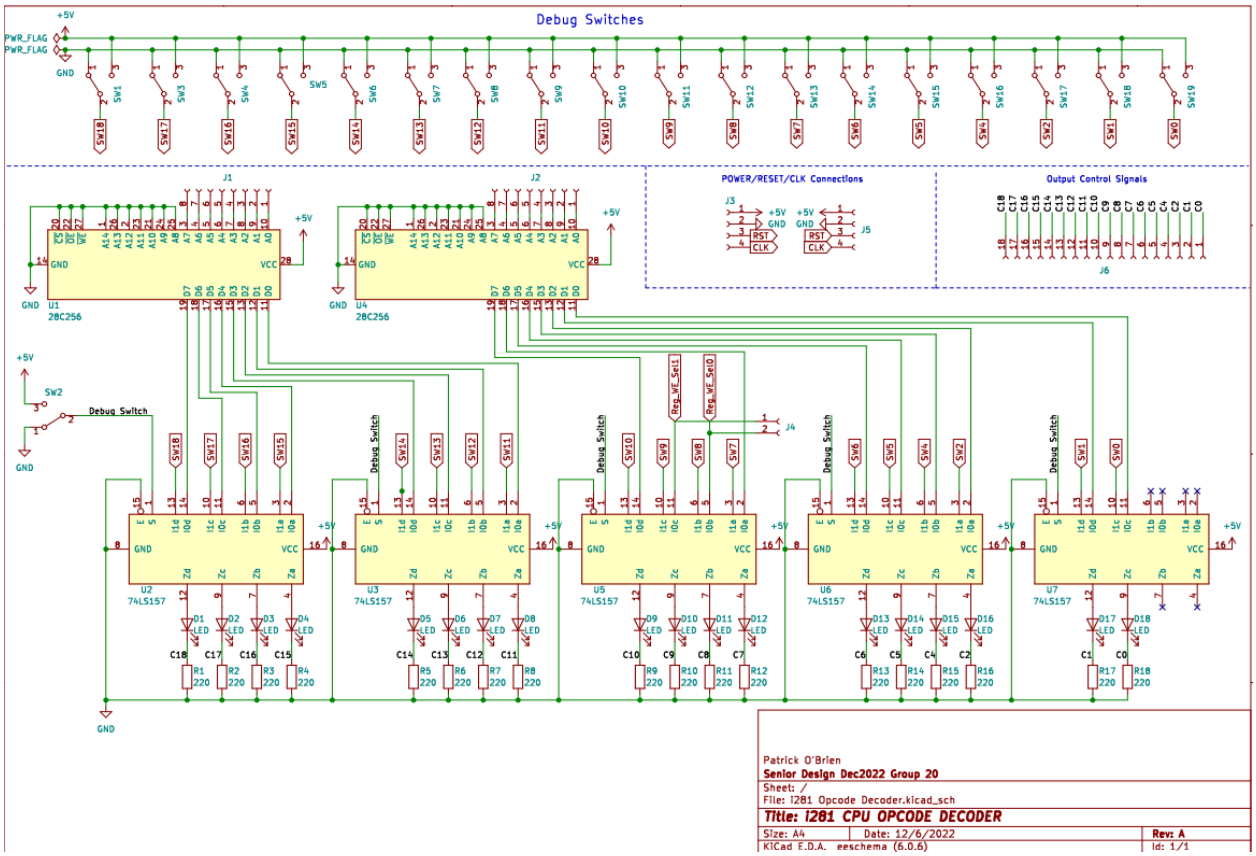
## Schematic:

*Inputs*:

- Switches
- IMEM Addresses

*Outputs*:

- CPU Control Signals



## *2 to 1 Bus Multiplexer*

The multiplexer is used as a data select. The i281 CPU uses 8-bit and 16-bit multiplexers. The multiplexers have two buses for data input and a single select bit. The select bit is selected via the manual control switches or the opcode decoder.

## Parts List:

| i281 CPU | | Bus Mux | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantity | Total Cost |
| 1 | Resistor | $0.05 | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24 | 220 | R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal | 24 | $ 1.20 |
| 2 | LED | $0.15 | D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16, D17, D18, D19, D20, D21, D22, D23, D24 | LED | LED_D3.0mm | 24 | $ 3.60 |
| 3 | 74LS157 Quad 2-1 Multiplexer | $0.93 | U1, U2 | 74LS157 | DIP-16_W7.62mm_LongPads | 2 | $ 1.86 |
| 4 | 10 Pin IDC Connector Socket | $2.00 | J1, J2, J3, J4, J5, J6, J7, J8, J9, J10, J11, J12 | Conn_01x08_Female | IDC-Header_2x05_P2.54mm_Vertical | 12 | $ 24.00 |
| 5 | 4 Pin Molex Mega-Fit Socket | $4.06 | J14, J15 | Conn_01x04_Female | Molex_Mega-Fit_76829-0004_2x02_P5.70mm_Vertical | 2 | $ 8.12 |
| 6 | 2.54mm Header Pins | $0.15 | J13 | Conn_01x01_Female | PinHeader_1x01_P2.54mm_Vertical | 1 | $ 0.15 |

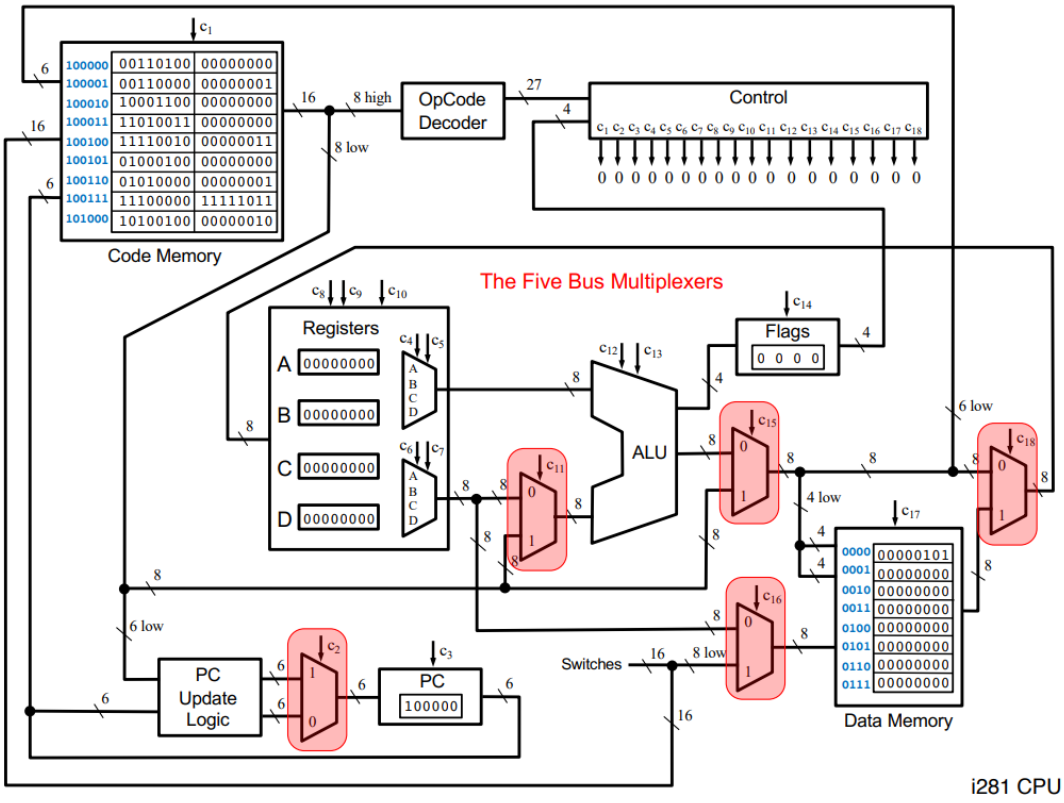
## Schematic:

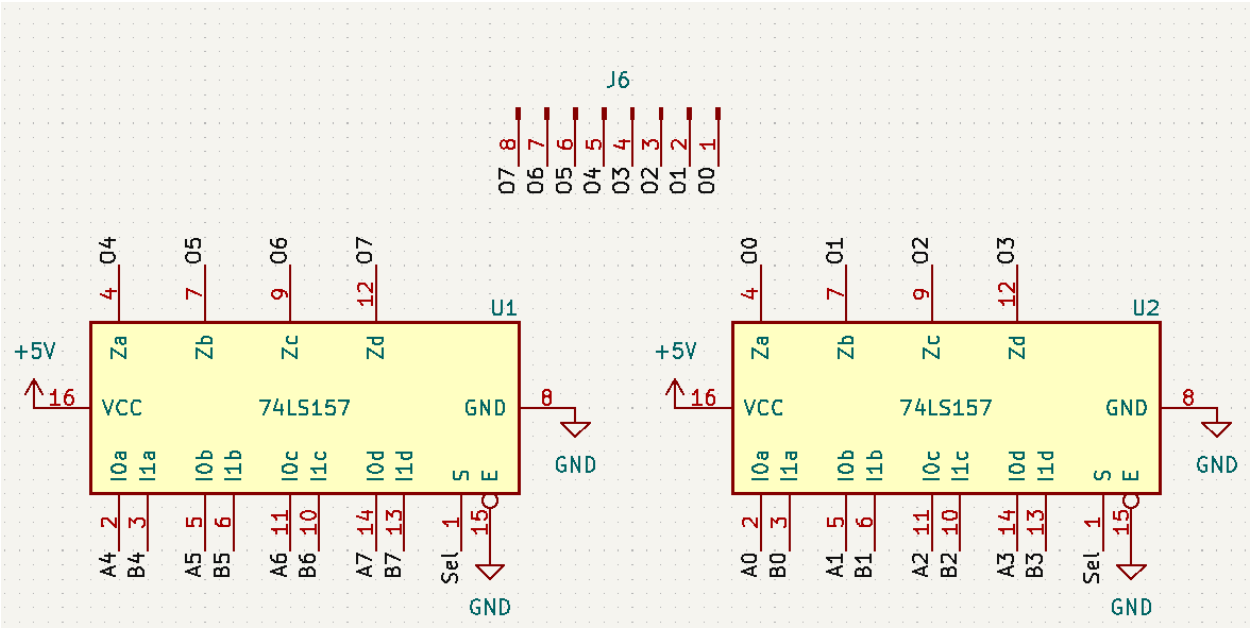
*Inputs:*

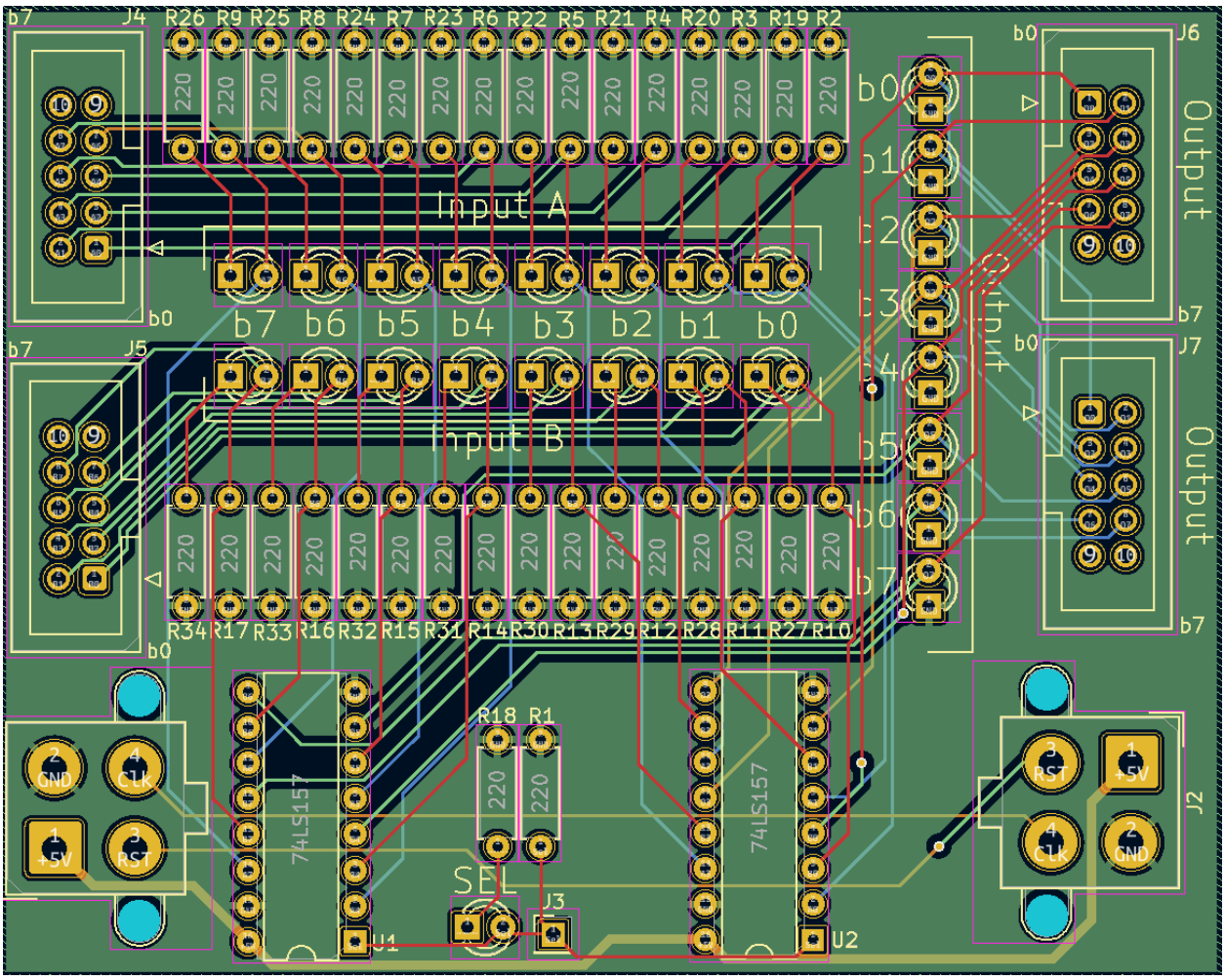
- Input A [N:0]
- Input B [N:0]
- Select

*Outputs:*

- Data [N:0]




J6
O7 O6 O5 O4 O3 O2 O1 O0
U1
U2
74LS157
+5V
VCC
GND
Za Zb Zc Zd
I0a I1a I0b I1b I0c I1c I0d I1d
S E
A4 B4 A5 B5 A6 B6 A7 B7 Sel
A0 B0 A1 B1 A2 B2 A3 B3 Sel


J4
J5
J6
J7
J2
J3
Input A
Input B
Output
b7 b6 b5 b4 b3 b2 b1 b0
R26 R9 R25 R8 R24 R7 R23 R6 R22 R5 R21 R4 R20 R3 R19 R2
R34 R17 R33 R16 R32 R15 R31 R14 R30 R13 R29 R12 R28 R11 R27 R10
R18 R1
220
74LS157
U1
U2
SEL
GND
Clk
RST
+5V

## *Data Memory*

The data memory is used to store data during runtime. The four read/write address bits are received from mux C15. The 8 data bits are received from mux C16. The eight output bits are sent to the graphics processor and C18. The DMEM and graphics processor are combined as a single unit when creating the schematics and breadboards. The graphics processor displays data stored in the eight least significant addresses of the data memory. Data and addresses are shared between the DMEM and graphics processor, allowing both components to update simultaneously.

## Parts List:

| i281 CPU | | Video Card | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantity | Total Cost |
| 1 | THT 1/4W Resistors | $0.05 | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16 | R | R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal | 16 | $ 0.80 |
| 2 | 7 Seg Display | $1.14 | Disp0, Disp1, Disp2, Disp3, Disp4, Disp5, Disp6, Disp7 | D168K | D1X8K | 8 | $ 9.12 |
| 3 | Register | $1.59 | Register0, Register1, Register2, Register3, Register4, Register6, Register7, Regsiter5 | 74LS377 | DIP-20_W7.62mm | 8 | $ 12.72 |
| 4 | SRAM | $7.00 | SRAM1 | CY62256-70PC | DIP-28_W15.24mm | 1 | $ 7.00 |
| 5 | EEPROM | $12.18 | EEPROM | AT28C18 | | 1 | $ 12.18 |

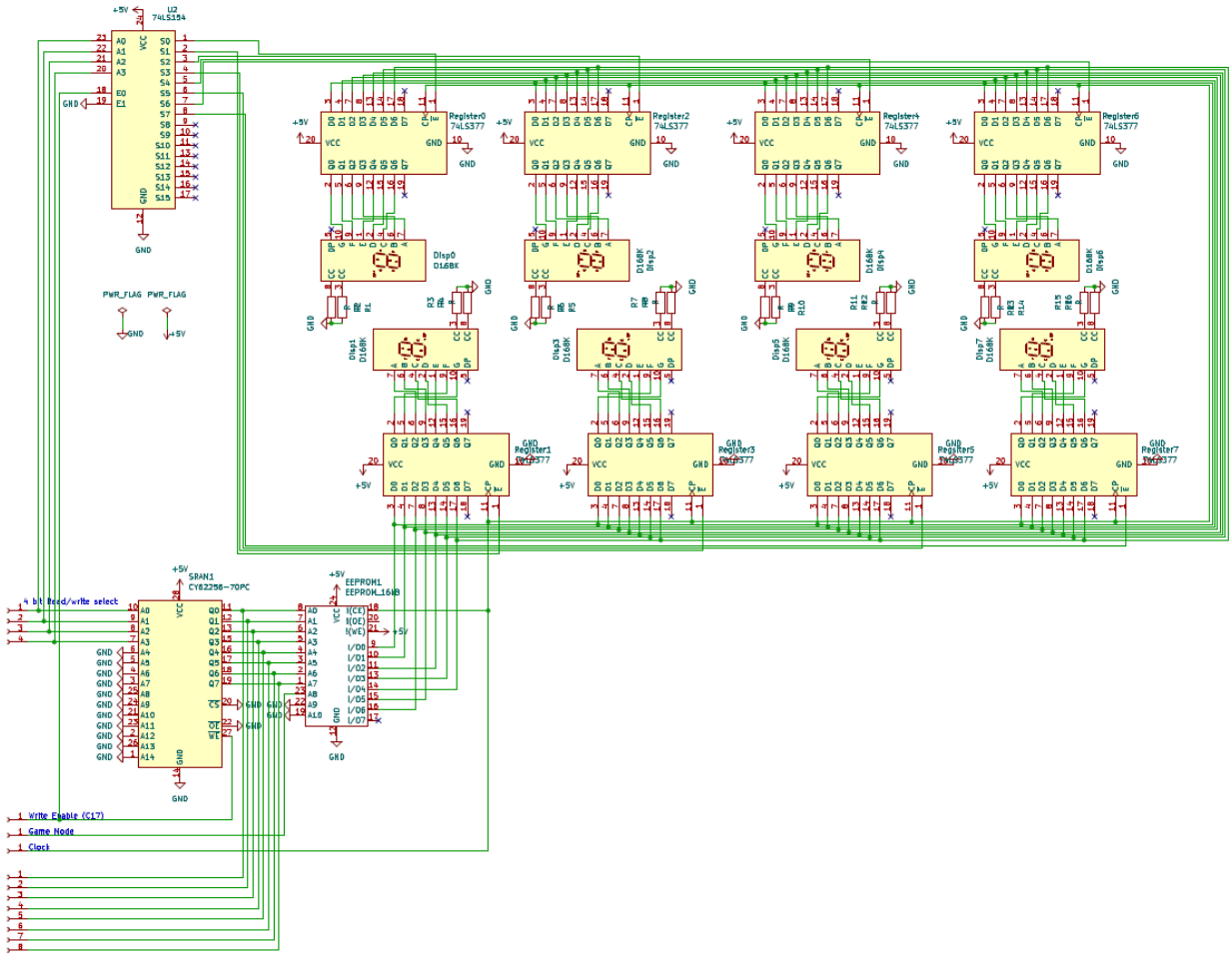
## Schematic:

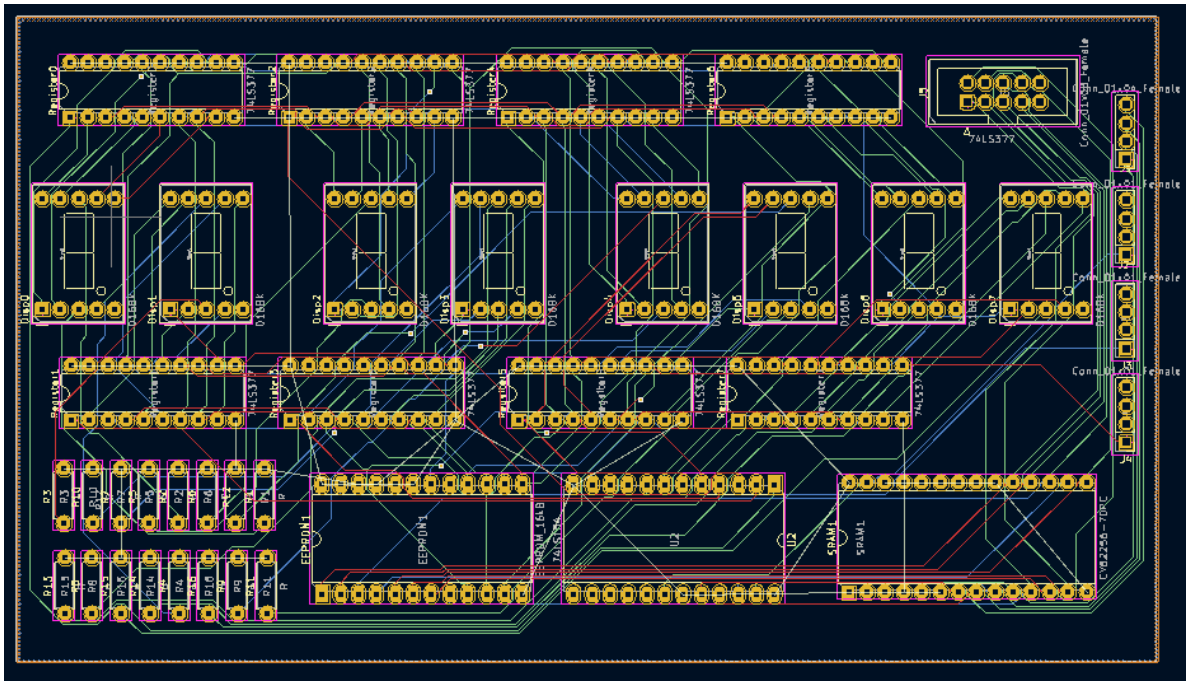
*Inputs:*

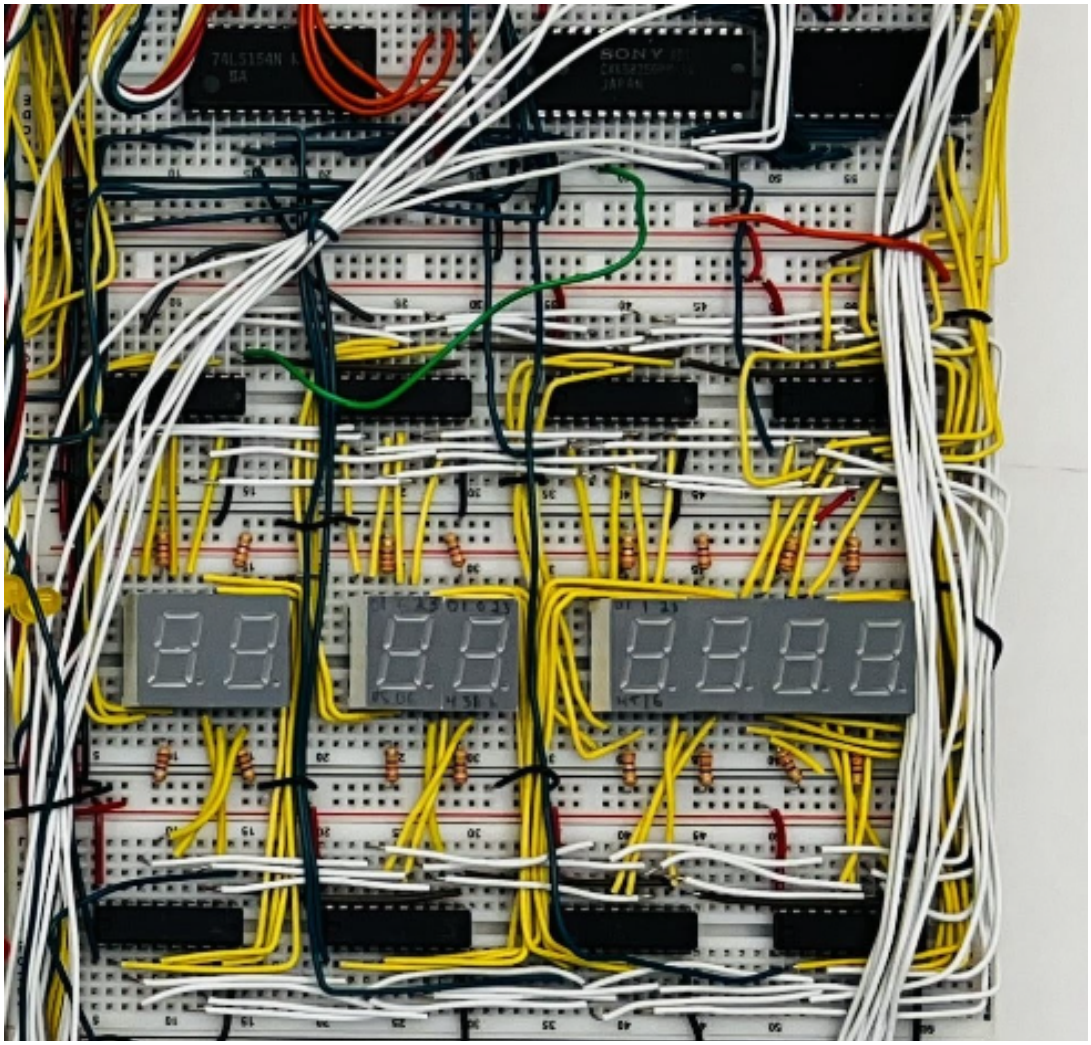
- Read/Write Select [3:0]
- Data Input [7:0]

*Outputs:*

- Data Output [7:0]






SONY

## *Instruction Memory*

The Instruction Memory is a 128-address x 16-bit memory. Addresses 0 to 63 contain the BIOS information stored on EEPROM chips. The BIOS data is used to prepare the CPU for execution. The BIOS calls the BOOT and waits for execution to complete. After the boot process has been completed, the Instruction Memory enters addresses 64 to 127. The data in these addresses are stored on the SRAM chips. These addresses are used during runtime to execute a program on the CPU. The SRAM and EEPROM output are sent to a set of muxes. The mux outputs EEPROM data when the PC MSB is 0 and SRAM data when 1.

## Parts List:

| i281 CPU | | IMEM | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantity | Total Cost |
| 1 | Resistor | $0.05 | R1, R2, R3, R4, R5, R6, R7, R8, | 220 | R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_ | 16 | $0.80 |
| 2 | LED | $0.15 | D1, D2, D3, D4, D5, D6, D7, D8 | LED | LED_D3.0mm | 16 | $2.40 |
| 3 | Octal Tranciever | $1.09 | U1, U2 | 74LS245 | DIP-20_W7.62mm | 2 | $2.18 |
| 4 | 74LS157 Quad 2-1 Multiplexer | $0.93 | [8:11]1, [8:11]2, [8:11]3, [8:11 | 74LS157 | DIP-16_W7.62mm | 4 | $3.72 |
| 5 | EEPROM | $29.00 | BIOS_High_bits1, BIOS_Low_b | 28C256 | DIP-28_W15.24mm | 2 | $58.00 |
| 6 | SRAM | $7.00 | IMEM_high_bits1, IMEM_low_ | CXK58256PM-10 | DIP-28_W15.24mm | 2 | $14.00 |
| 7 | 2.54mm Header Pins | $0.15 | PC_Address1 | Conn_01x07_Female | PinHeader_1x07_P2.54mm_Vertical | 1 | $0.15 |
| 8 | 2.54mm Header Pins | $0.15 | J1, J4 | Conn_01x01_Female | PinHeader_1x01_P2.54mm_Vertical | 2 | $0.30 |
| 9 | 10 Pin IDC Connector Socket | $3.08 | J2, J3 | Conn_01x08_Female | IDC-Header_2x05_P2.54mm_Vertical | 2 | $6.16 |
| 10 | 4 Pin Molex Mega-Fit Socket | $4.06 | | 2x2 Female | Molex_Mega-Fit_76829-0004_2x02_P5.70mm_Vertical | 2 | $8.12 |

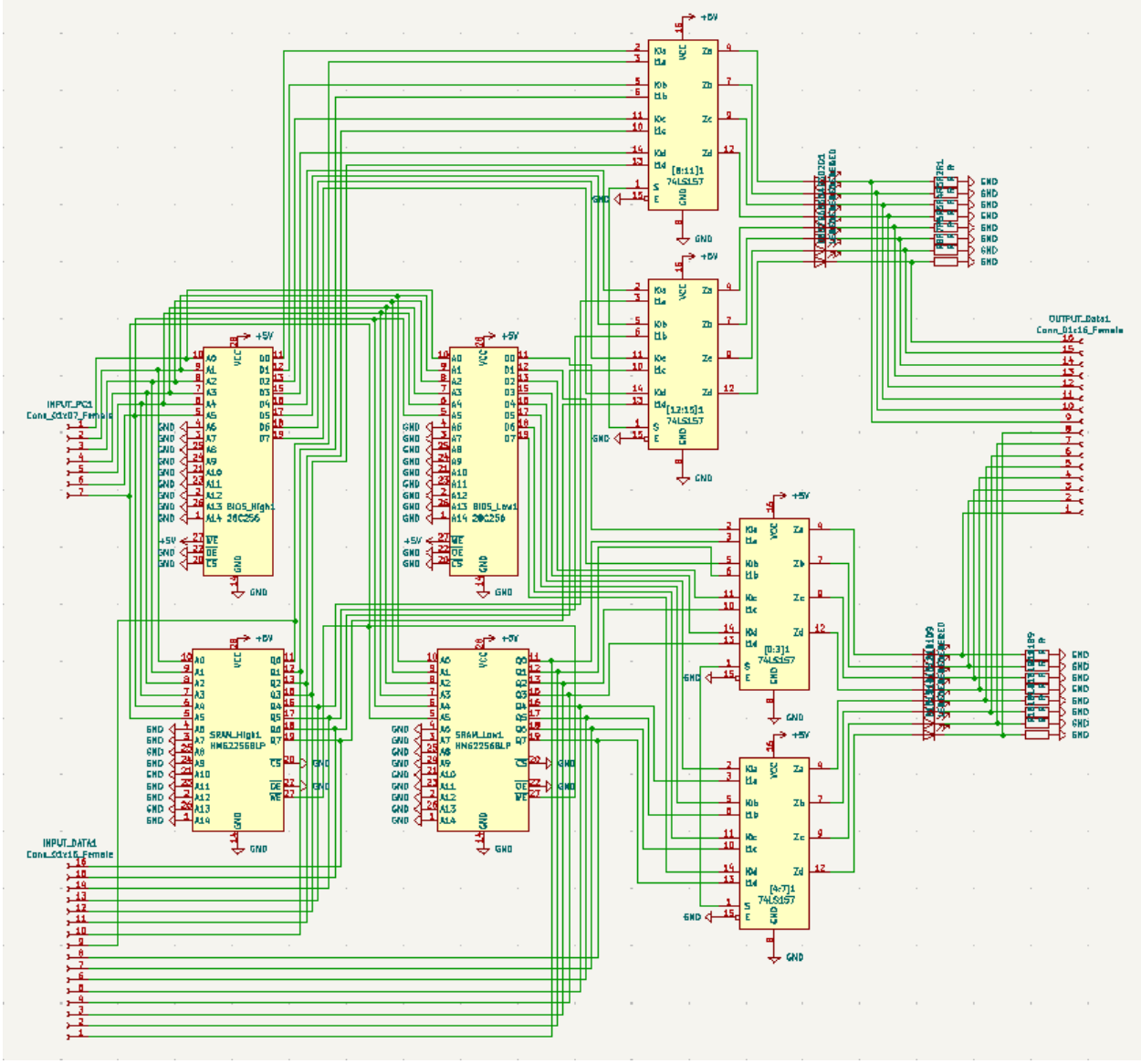
## Schematic:

*Inputs*:

- PC [6:0]
- Boot Data [15:0]
- Write

*Outputs*:

- Data [15:0]


INPUT_PC1
Conn_01x07_Female
BIOS_High1
28C256
BIOS_Low1
28C256
SRAM_High1
HM62256BLP
SRAM_Low1
HM62256BLP
INPUT_DATA1
Conn_01x16_Female
OUTPUT_Data1
Conn_01x16_Female
74LS157
+5V
GND

## *Boot Sequence*

The Boot module is used to initialize data in the static memory on the CPU. During startup, the BIOS calls boot requesting data to fill the register file, DMEM, and the lower 64 addresses of the IMEM. Data can be programmed onto the Boot EEPROMs or 16 switches that can manually add data.

## Parts List:

| i281 CPU | | BOOT | | | | | |
|---|---|---|---|---|---|---|---|
| Item | Description | Cost Per Unit | PCB References | Value | Footprint | Quantity | Total Cost |
| 1 | Resistor | $0.05 | R1, R2, R3, R4, R5, R6, R7, R8, R9, R10, R11, R12, R13, R14, R15, R16, R17, R18, R19, R20, R21, R22, R23, R24, R25, R26, R27 | 220 | R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal | 27 | $1.35 |
| 2 | LED | $0.15 | D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16, D17, D18, D19, D20, D21, D22, D23, D24, D25, D26, D27 | LED | LED_D3.0mm | 27 | $4.05 |
| 3 | EG1218 | $0.76 | addr[7]1, addr[8]1, addr[9]1, addr[10]1, addr[11]1, SW[00]1, SW[01]1, SW[02]1, SW[03]1, SW[04]1, SW[05]1, SW[06]1, SW[07]1, SW[08]1, SW[09]1, SW[10]1, SW[11]1, SW[12]1, SW[13]1, SW[14]1, SW[15]1 | SW_SPDT | SWITCH_SLIDE_3hole | 21 | $15.96 |
| 4 | 74LS157 Quad 2-1 Multiplexer | $0.93 | [0:3]1, [4:7]1, [8:11]1, [12:15]1 | 74LS157 | DIP-16_W7.62mm | 4 | $3.72 |
| 5 | EEPROM | $29.00 | HardDiskHigh1, HardDiskLow1 | 28C256 | DIP-28_W15.24mm | 2 | $58.00 |
| 6 | 2.54mm Header Pins | $0.15 | INPUT_C0 | Conn_01x01_Female | PinHeader_1x01_P2.54mm_Vertical | 1 | $0.15 |
| 7 | 2.54mm Header Pins | $0.15 | INPUT_PC1 | Conn_01x07_Female | PinHeader_1x07_P2.54mm_Vertical | 1 | $0.15 |
| 8 | 10 Pin IDC Connector Socket | $3.08 | J1, J2 | Conn_01x08_Female | IDC-Header_2x05_P2.54mm_Vertical | 2 | $6.16 |
| 9 | 4 Pin Molex Mega-Fit Socket | $4.06 | J4, J8 | 2x2 Female | Molex_Mega-Fit_76829-0004_2x02_P5.70mm_Vertical | 2 | $8.12 |

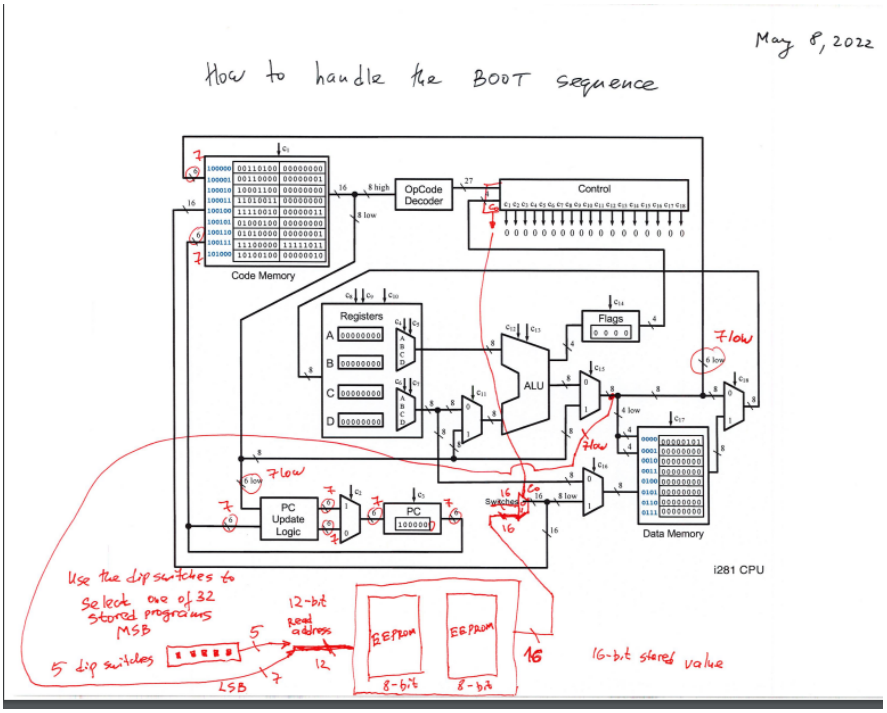
## Schematic:

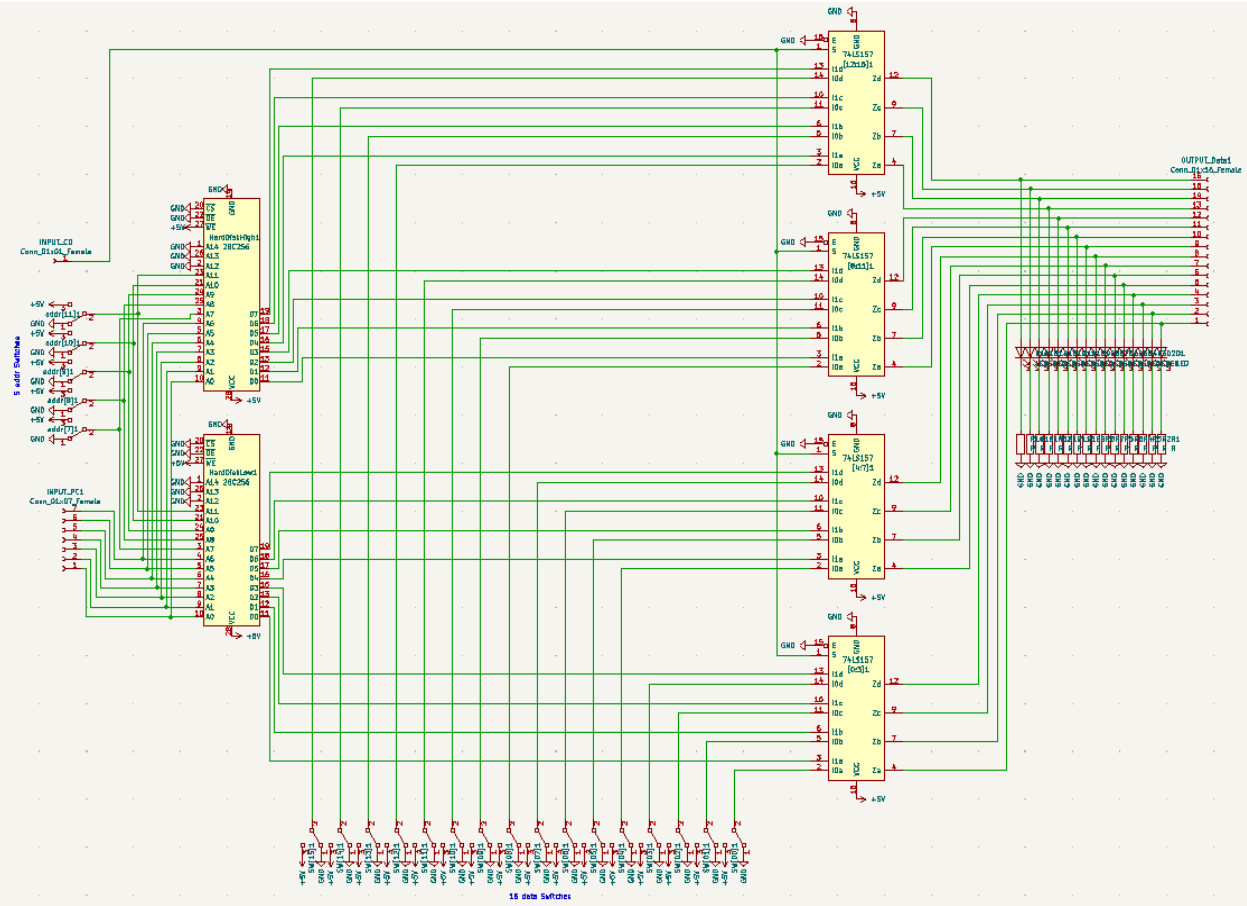
*Inputs*:

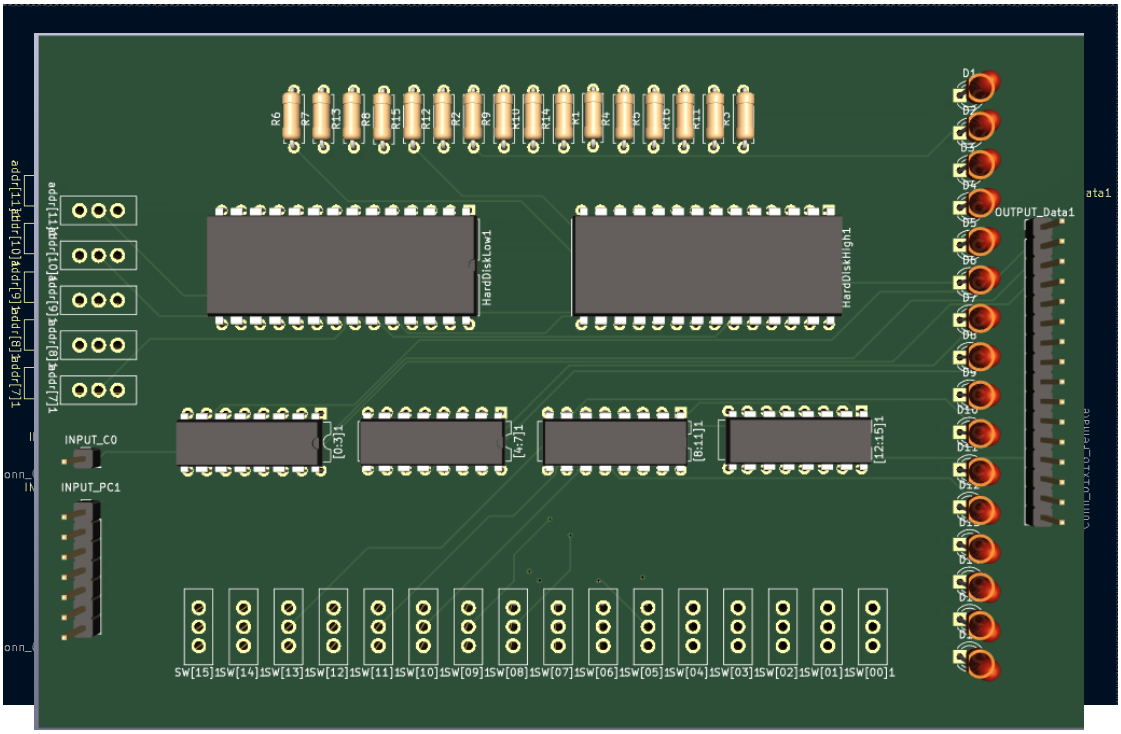
- PC [6:0]
- C0 [1]

*Outputs*:

- Data [15:0]






HardDiskHigh1
HardDiskLow1
OUTPUT_Data1
INPUT_C0
INPUT_PC1
SW[15]1SW[14]1SW[13]1SW[12]1SW[11]1SW[10]1SW[09]1SW[08]1SW[07]1SW[06]1SW[05]1SW[04]1SW[03]1SW[02]1SW[01]1SW[00]1

# EEPROM Programming

To test the final implementation of this CPU, we had to program the EEPROMs with test values. To do this, a java program was developed to convert programs written in hexadecimal values–that directly correlated to the assembly language Professor Stoytchev wrote–to a version of Intel Hex. This was done to program our EEPROMs since we bought an XGecu programmer, which required a special version of intel hex to load our programs onto the programmer's associated software. However, this programmer didn't work for every one of our EEPROMs, so we additionally had to develop an Arduino programmer attached to a breadboard to program our remaining EEPROMs. To use this programmer, we no longer needed the java files to convert to intel hex, but we did need to divide our hex files into segments that could be individually loaded onto the Arduino. Ultimately, we could load various test programs onto the EEPROMs, allowing us to do limited testing of our CPU.

# Combined Implementation

After creating and testing each module individually, we combined them to create a single datapath. Modules can be connected in three ways. First, modules can be connected through a bus. Whenever a module receives data from a single location, we can use a bus to transport data. However, if a module receives data from multiple locations, the second method is to use a mux to select between the data received. The most commonly used size mux is an 8-bit 2-to-1 mux. However, 4-to-1 muxes and 16-bit 2-to-1 muxes are used as needed. Finally, the third method to connect modules is to use a tri-state buffer. A tri-state buffer is used when data needs to be sent only one directionally. This is rarely used but is needed in locations where chips have shared I/O pins, such as the CXK58256 SRAM, AT28C256 EEPROM, and the AT28C16 EEPROM.

When we combined all required modules, a common power, ground, clock, and reset was implemented to be shared throughout all modules. The breadboard implementation handles the common power and ground through the side rails on the breadboards. The reset and clock are shared through a loop that sends data to each module.

# Datasheets

74LS04: Hex Inverter
74LS10: 3 Input Nand
74LS32: Quad 2 Input Or
74LS86: Quad 2 Input Xor
74LS139: 2-4 Decoder
74LS153: 4-1 Multiplexer
74LS154: 4-16 Decoder
74LS157: 2-1 Multiplexer
74LS173: 4 Bit Register
74LS245: Octal Transceiver
74LS283: 4 Bit Adder
74LS377: 8 Bit Register
AT28C16: 11 Address EEPROM
AT28C256: 15 Address EEPROM
CXK58256: SRAM
LM555: 555 Timer

# Project Evolution

During the first semester of senior design, our team's goal was to implement the i281 CPU on breadboards and create a working PCB implementation. We planned to finish a working breadboard implementation by the end of the first semester. This would give us the second semester to create PCB designs, purchase PCBs, and solder the required chips. However, our team was unable to follow this schedule.

At the end of the first semester, we had only completed a handful of the breadboard implementations and were not ready to move on to PCB design. We met with our client to rewrite our milestones to set the breadboard implementation as the final project goal. Our team worked diligently during the second semester to stay on track with our deadlines. After the CPU was fully implemented, we found many bugs. We changed our design to reduce errors and redesigned the CPU to a hardware-friendlier version. The various design changes are noted in the individual modules section of the report.

# Testing Process

There were multiple subsystems of the CPU, including the data memory, register file, ALU, instruction memory, and control logic. Each CPU subsystem was tested individually using dip switches to emulate data from the register file/instruction memory. We tested these by setting the switches to a specific value and observing the LED indicators on the unit's output (tools include oscilloscopes, multimeters, and LED indicators). This was the most concrete way to test our project because we already knew what outputs to expect, so it was a matter of verifying that our breadboard builds matched our expectations.

Our interface was switches, LEDs, 7-segment displays, and the text interface used to write the assembly. First, the LEDs and switches were tested by programming our instruction memory with basic instructions to ensure the correct LED values. Next, the 7-segment displays were tested to ensure the proper hexadecimal values were outputted with the correct input.

All CPU subsystems (ALU, Register File, Instruction memory, data memory, and control logic) were integrated after they were completed. They were slowly integrated to make a basic data path that could add instructions. After the first datapath was completed, more capabilities were added to support more and more of our instructions. The testing initially consisted of creating a basic datapath and checking the LED indicators to ensure correct data flow. When errors occurred, we used multimeters and oscilloscopes to debug problems.

As each component was critical to the success of the project as a whole, we tried to ensure that our testing was incredibly thorough when attempting to find any issues. While our initial tests of the modules looked positive, combining the modules together proved to be somewhat of an issue. Wiring all of the bus routes, as well as the connections between our interface switches, quickly got messy and complicated. While our previous testing methods were utilized to attempt to find issues as they arose, such as shorts from missed placed wires, we were never able to come up with a testing method that could easily catch those issues. Additionally, while our testing process could verify the functionality of individual modules, wires slipping out of place during transport of the units between our locker and our workstation quickly created an escalating problem where finding those issues began to take more and more of our time. While we were ultimately unsuccessful in solving all of these issues, we did gain valuable insight into test engineering and how the complexity of testing scales as a project does.

# Testing Results

The testing results informed us that our project had more bugs than we could realistically handle in the time we left. While we used these results to fix as many issues with our design as possible, ultimately, our results informed us of the simple fact that not all issues with this project could be solved.

# Engineering Constraints

There are many constraints for our project, however, we will list the most significant ones here. The first and our main constraint was that we had to match the software implementation. This meant that all the modules and connections on the software simulation had to be the same on the hardware implementation. Another major constraint was that the processors needed to be visually understandable to students, which, created an issue as LEDs have significant power requirements when using a lot of them. Another significant constraint was that the CPU needed to be able to run the entire Stoytchev ISA (Instruction Set Architecture), which was a massive logistical undertaking when we were already dealing with wiring issues. A final constraint for our team was simply the supply chain issues that plagued the tech industry during and following the pandemic. This resulted in us not always being able to get the exact parts we wanted for our implementation. While those were alleviated during our second semester, it was too late to change to many of the components we had initially wanted since most of our modules had been built by that point.

# Conclusion

The overall objective of our project was to create a breadboard implementation of the i281 CPU that closely follows the original i281 FPGA design. However, due to the nature of the hardware devices, we were required to modify the original design. These design changes required a strong understanding of the i281 CPU and the effects of the change.

Throughout the two semesters of this project, we got the wonderful opportunity to learn and grow as engineers. Our team's largest learning outcome is the importance of documentation. We found that bugs always occur, and minimal documentation makes solving these bugs near impossible. However, as the semester progressed, so did our documentation skills. We ended the semester with a well-documented project and the ability to fix bugs quickly.

Our team worked continuously to reach our defined objectives. However, in the end, we could not reach our final goal and run the i281 CPU using PONG and Bubble Sort. We believe that the knowledge and skills we learned throughout this class outweigh this missed objective, and we hope to see this passed on to another team for further completion and improvement.

# Appendix I – “Operation Manual”

In order to set up our senior design project:

1. One must first set up our breadboard implementation carefully on a stable surface.
2. Then, one must attach the connector to our power supply in order to connect to the board


3. After attaching the connector, one must connect both the red wire to a positive (+) row and the black wire to a negative (-) row on any open area of the breadboard. This will provide enough power for the CPU to function fully.
4. Next, the CPU will be powered on, and the user can utilize our controls to activate the test programs on our CPU.

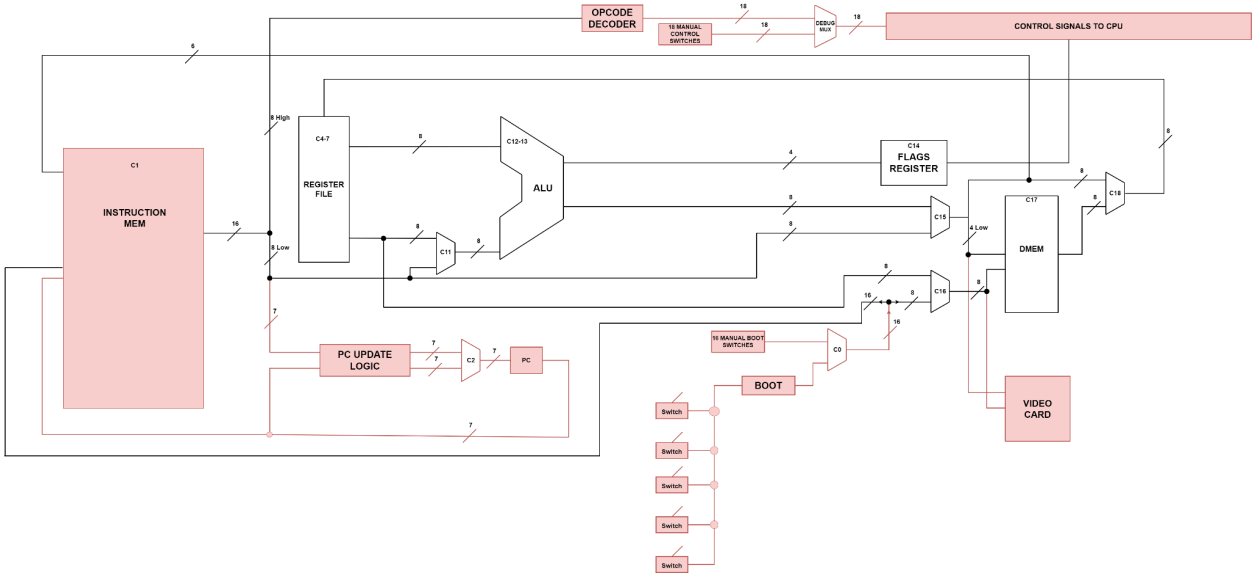
Note: One needs to be extremely careful not to knock out any wires on our project when picking it up and transferring it to the designated test station. Should any wires be knocked out of place, the CPU will have to be tested extensively to determine where a wire is loose. This will additionally consist of then consulting the manual of the chip that the wire is supposed to be connected to determine where the wire needs to be placed.

In order to test our senior design project:

1. One must turn on the CPU and determine which program will be run for testing.
    a. Should a new program wish to be tested, the user will have to study Professor Stoychevs assembly language to write the desired program. Reference materials are provided in the form of the .java files that were written to allow writing to the

EEPROMs in this project. This will allow a user to reference already written programs that are meant to be able to run on this CPU.

2. Next, one must study the program execution LEDs and make their way around the board following the path outlined in our design diagram.




   a. If the clock setting is too high, it will be impossible for the user to trace the program execution, so the user will have to locate the clock module and use the potentiometer on that module to adjust to a traceable speed.

3. Due to issues with the video card module, the user will not be able to determine if the CPU is outputting the proper values. However, by utilizing the numerous LEDs that show what each module of our CPU is doing, the user can manually determine if the expected values are appearing at different modules, during execution, to determine the functionality of the CPU.

Note: Should any issues be encountered in testing the execution of the program, this means a wire has likely come loose. In that case, one will, at the very least, need multimeters to test and identify where the wire is loose in order to fix the problem.

In order to demo this CPU:

1. One must follow similar steps as with the testing of the CPU.
2. However, one must also provide an accurate account of what the expected values are for the output of the program that they are running.
3. This can be achieved by referencing Professor Stoytchevs assembly language explanation slides on his website, available at this link.
4. Then, one must manually trace out the expended hex and binary values expected at each step of the program.
5. Finally, one can demonstrate through observation of the onboard LEDs that the CPU is executing the program according to expectation.