i281e CPU

Implement the i281 CPU in Hardware

Team Members:

Daryl Damman, Logan Lee, Grant Nordling, Braxton Rokos, Gavin Tersteeg

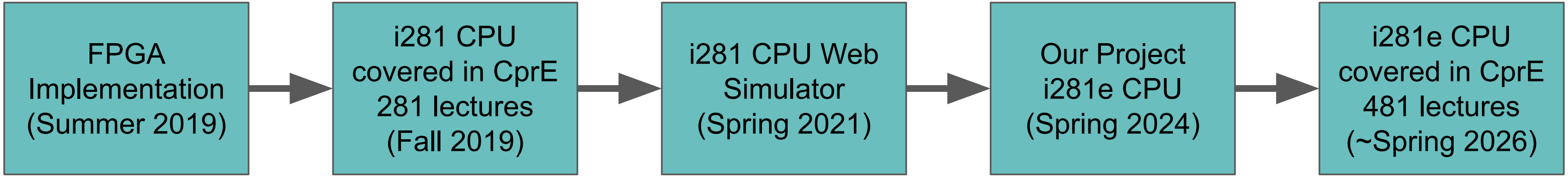
Client/Advisor: Professor Alexander Stoytchev

Project Vision

The i281 CPU was designed to support the curriculum in CprE 281: Digital Logic. Currently, this design is implemented to run on an FPGA and a web simulator. To fully realize the potential of this teaching tool, students also need a physical implementation to interact with.

Our implementation of the i281 CPU must be created in dedicated hardware and remain similar to the FPGA and simulator designs. Two implementations must be designed on breadboards and printed circuit boards, respectively.

Historical Timeline



FPGA Implementation (Summer 2019)

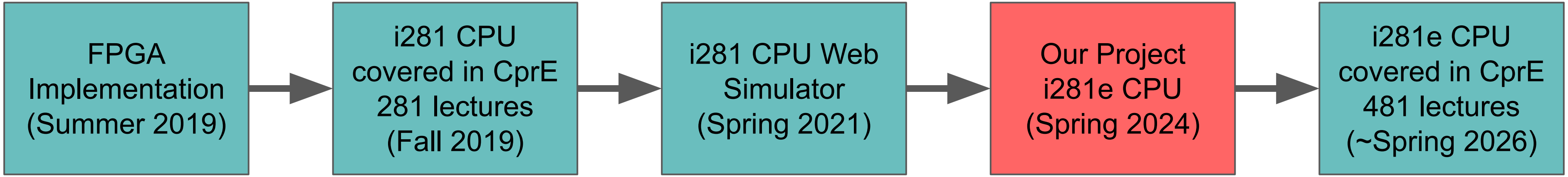
i281 CPU covered in CprE 281 lectures (Fall 2019)

i281 CPU Web Simulator (Spring 2021)

Our Project i281e CPU (Spring 2024)

i281e CPU covered in CprE 481 lectures (~Spring 2026)

Historical Timeline



FPGA Implementation (Summer 2019)

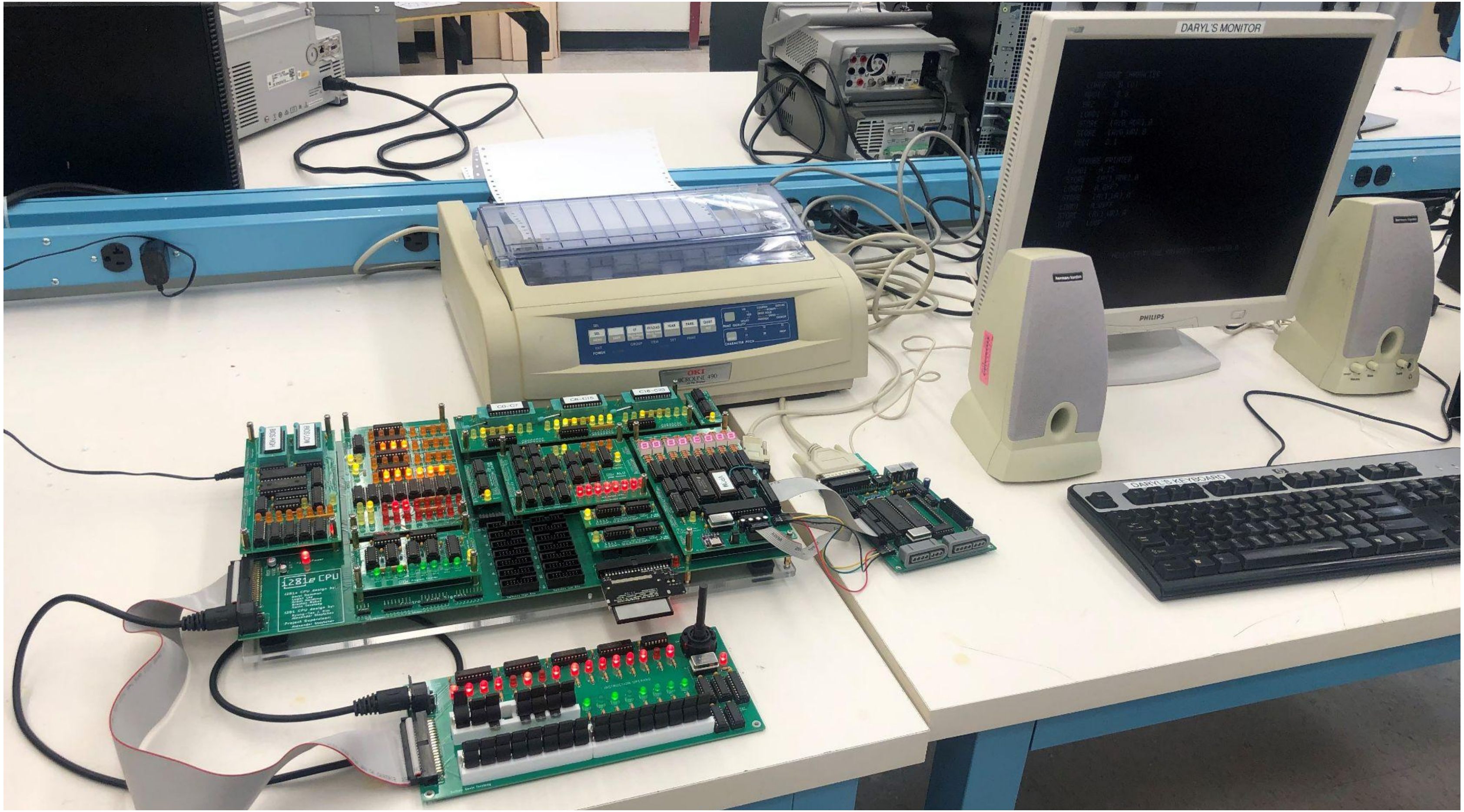
i281 CPU covered in CprE 281 lectures (Fall 2019)

i281 CPU Web Simulator (Spring 2021)

Our Project i281e CPU (Spring 2024)

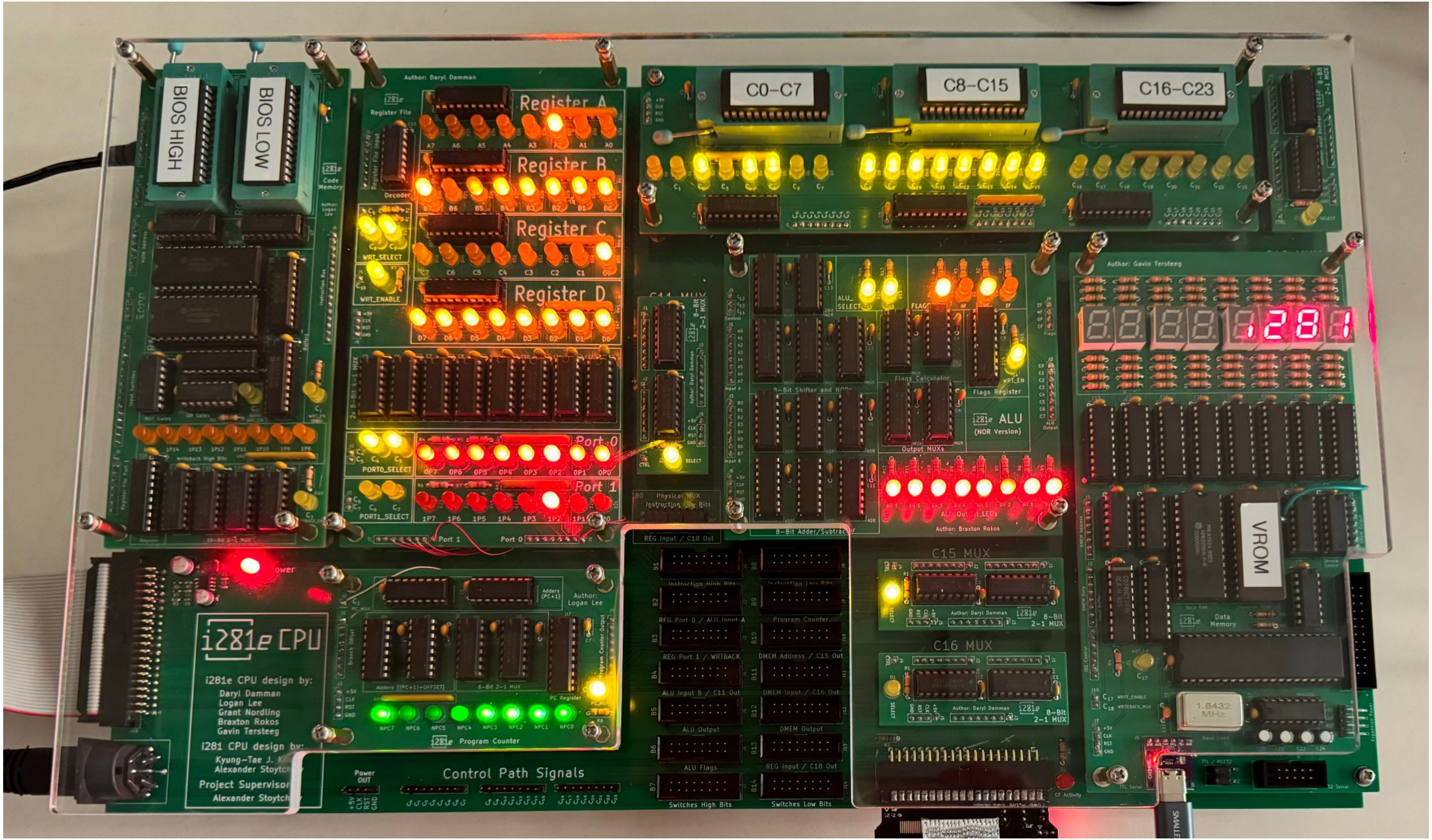
i281e CPU covered in CprE 481 lectures (~Spring 2026)

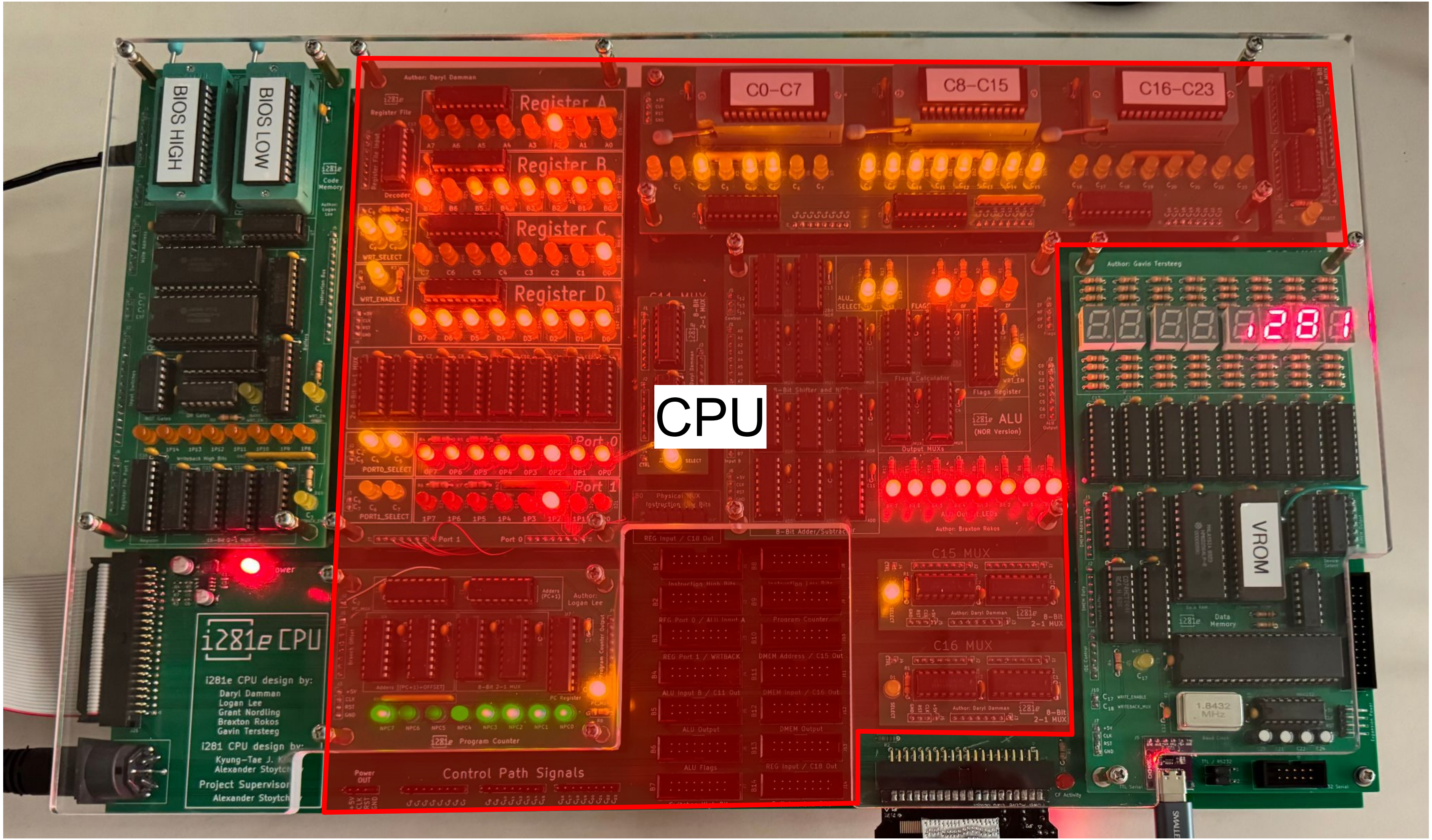
Fully Functional Prototype

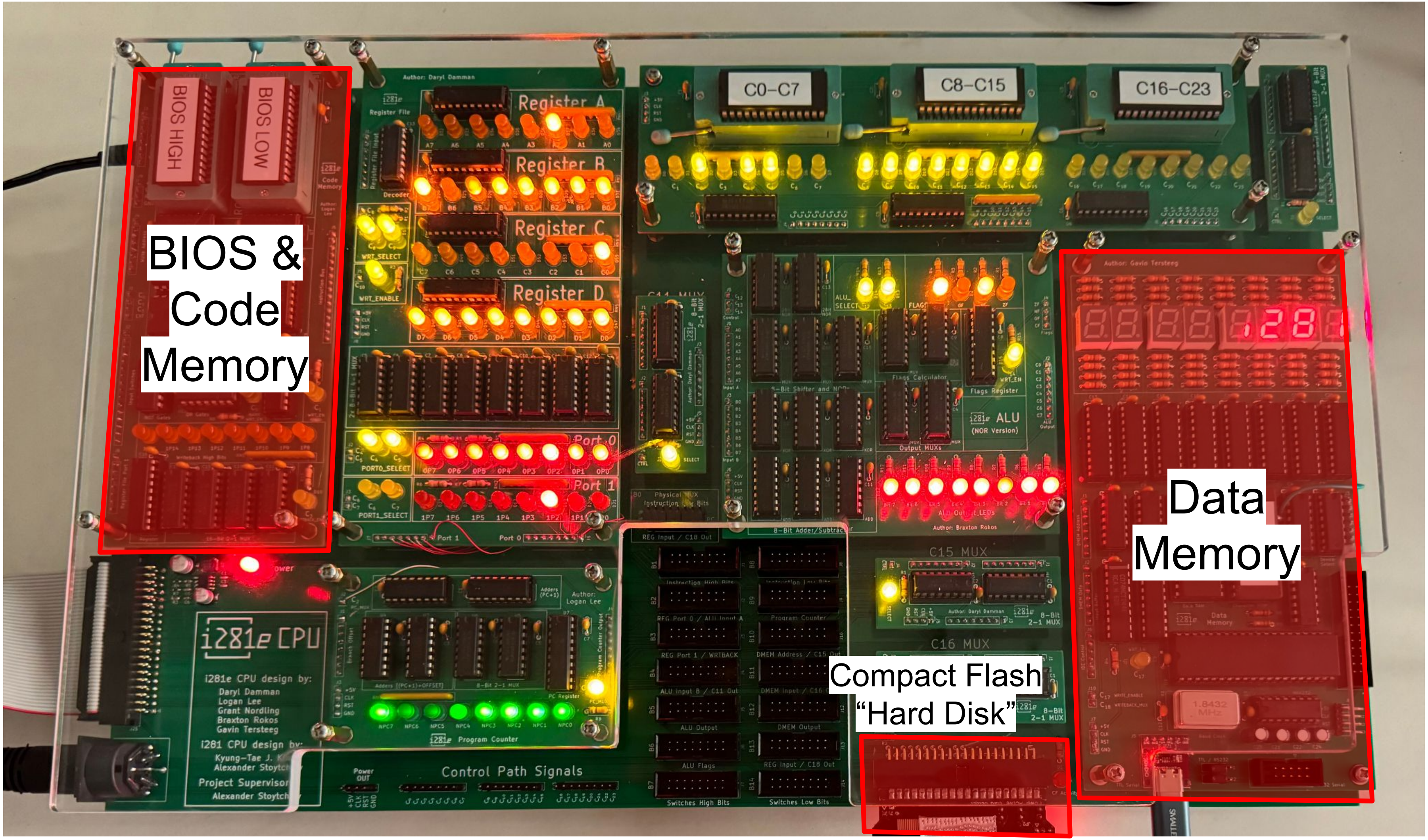


i281e CPU Specifications

- Clock Speed: 1Hz up to 2MHz (up to 2.5MHz overclock)
  - Processor fails around 2.75MHz
- Power Requirements: 0.8A @ 5VDC (Input 5-12VDC)
  - Fuse for overcurrent protection @ 2A
- Memory: 32 KW (64 KB) of Code RAM, 32 KB of Data RAM
  - Also includes 128 words of Code ROM for booting the system
- Compact Flash: extended memory for long-term storage
  - Acts as the “hard disk” and stores the OS and File System for DOS/281


BIOS HIGH
BIOS LOW
Code Memory
Author: Daryl Damman
Register File
Register A
Register B
Register C
Register D
Decoder
WRT_SELECT
WRT_ENABLE
Port 0
Port 1
PORT0_SELECT
PORT1_SELECT
C0–C7
C8–C15
C16–C23
C11 MUX
8-Bit 2-1 MUX
Physical MUX Instruction Bits
ALU_SELECT
FLAGS
8-Bit Shifter and NOR
Flags Calculator
Flags Register
i281e ALU (NOR Version)
Output MUXs
ALU Output LEDs
Author: Braxton Rokos
8-Bit Adder/Subtractor
Author: Gavin Tersteeg
1281
VROM
Data Memory
1.8432 MHz
C15 MUX
C16 MUX
Power
i281e CPU
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281 CPU design by:
Kyung-Tae J. K
Alexander Stoytch
Project Supervisor
Alexander Stoytch
Author: Logan Lee
Adders (PC+1)
Adders [(PC+1)+OFFSET]
8-Bit 2-1 MUX
PC Register
i281e Program Counter
Control Path Signals
Power OUT
Instruction High Bits
Instruction Low Bits
REG Port 0 / ALU Input A
Program Counter
REG Port 1 / WRTBACK
DMEM Address / C15 Out
ALU Input B / C11 Out
DMEM Input / C16 Out
ALU Output
DMEM Output
ALU Flags
REG Input / C18 Out
Switches High Bits
Switches Low Bits
CF Activity


CPU
BIOS HIGH
BIOS LOW
Code Memory
Author: Daryl Damman
Register File
Register A
Register B
Register C
Register D
C0–C7
C8–C15
C16–C23
Port 0
Port 1
PORT0_SELECT
PORT1_SELECT
WRT_SELECT
WRT_ENABLE
Decoder
Author: Gavin Tersteeg
i281e ALU
(NOR Version)
Flags Register
Flags Calculator
8-Bit Adder/Subtractor
ALU Output LEDs
Author: Braxton Rokos
C15 MUX
C16 MUX
8-Bit 2-1 MUX
Program Counter
Author: Logan Lee
Control Path Signals
Power
VROM
Data Memory
1.8432 MHz
i281e CPU
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281 CPU design by:
Kyung-Tae J. K
Alexander Stoytch
Project Supervisor
Alexander Stoytch


BIOS &
Code
Memory
Data
Memory
Compact Flash
"Hard Disk"
BIOS HIGH
BIOS LOW
C0–C7
C8–C15
C16–C23
Register A
Register B
Register C
Register D
Port 0
Port 1
i281e CPU
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281 CPU design by:
Kyung-Tae J. K
Alexander Stoytch
Project Supervisor
Alexander Stoytch
Control Path Signals
C15 MUX
C16 MUX
ALU (NOR Version)
Program Counter

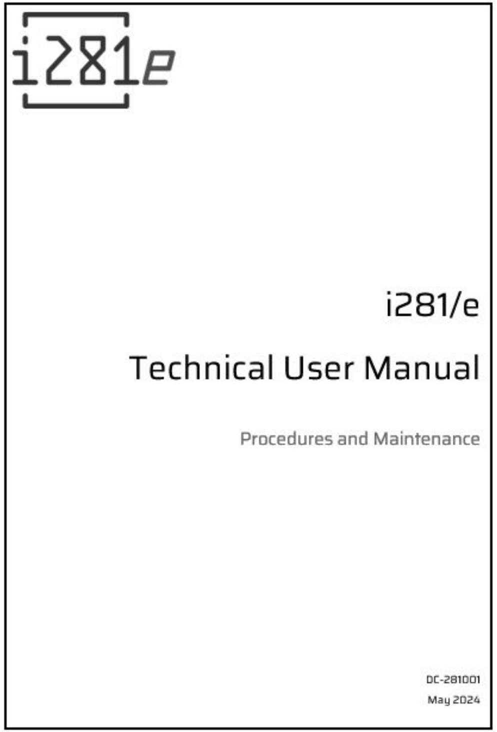
Requirements

Braxton Rokos
Electrical Engineering

Project Requirements

- The CPU must be explainable to sophomore students in CprE 281.
- The CPU data path and control path must be visualized with LEDs.
- Adjustable CPU clock to enable stepping through individual instructions and operating at different clock frequencies.
- The CPU must be capable of playing the original i281 PONG video game and other example programs from the FPGA and the Web Simulator.
- The CPU design must be as close as possible to the original design.
- It must look cool.

Explainable Content



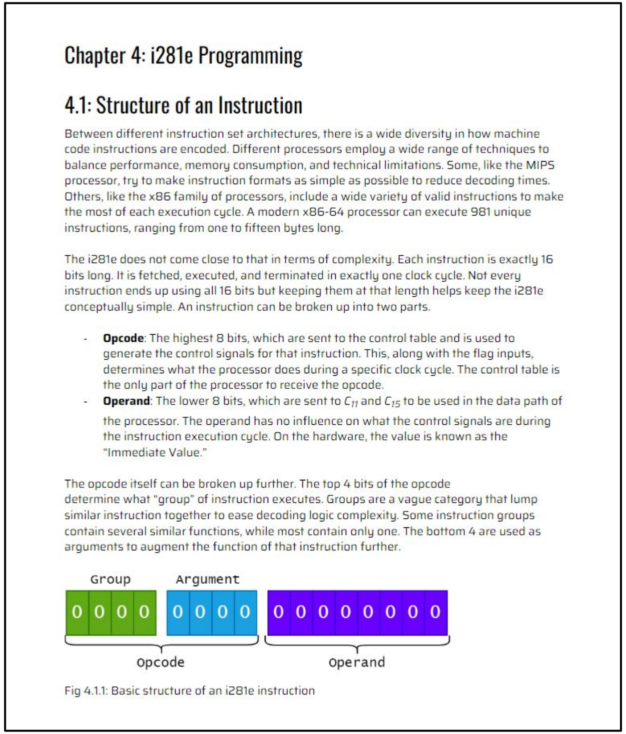
i281/e

Technical User Manual

Procedures and Maintenance

DC-281001

May 2024



Chapter 4: i281e Programming

4.1: Structure of an Instruction

Between different instruction set architectures, there is a wide diversity in how machine code instructions are encoded. Different processors employ a wide range of techniques to balance performance, memory consumption, and technical limitations. Some, like the MIPS processor, try to make instruction formats as simple as possible to reduce decoding times. Others, like the x86 family of processors, include a wide variety of valid instructions to make the most of each execution cycle. A modern x86-64 processor can execute 981 unique instructions, ranging from one to fifteen bytes long.

The i281e does not come close to that in terms of complexity. Each instruction is exactly 16 bits long. It is fetched, executed, and terminated in exactly one clock cycle. Not every instruction ends up using all 16 bits but keeping them at that length helps keep the i281e conceptually simple. An instruction can be broken up into two parts.

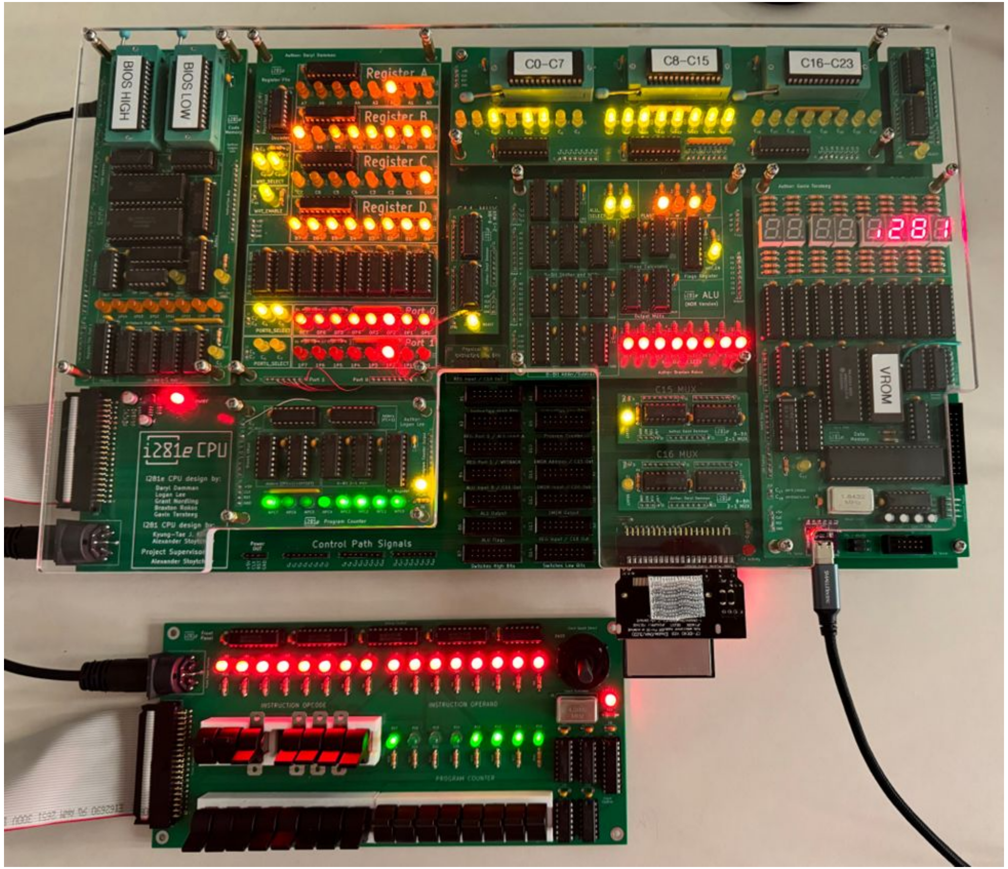
- **Opcode**: The highest 8 bits, which are sent to the control table and is used to generate the control signals for that instruction. This, along with the flag inputs, determines what the processor does during a specific clock cycle. The control table is the only part of the processor to receive the opcode.
- **Operand**: The lower 8 bits, which are sent to C_{11} and C_{15} to be used in the data path of the processor. The operand has no influence on what the control signals are during the instruction execution cycle. On the hardware, the value is known as the "Immediate Value."

The opcode itself can be broken up further. The top 4 bits of the opcode determine what "group" of instruction executes. Groups are a vague category that lump similar instruction together to ease decoding logic complexity. Some instruction groups contain several similar functions, while most contain only one. The bottom 4 are used as arguments to augment the function of that instruction further.

Fig 4.1.1: Basic structure of an i281e instruction

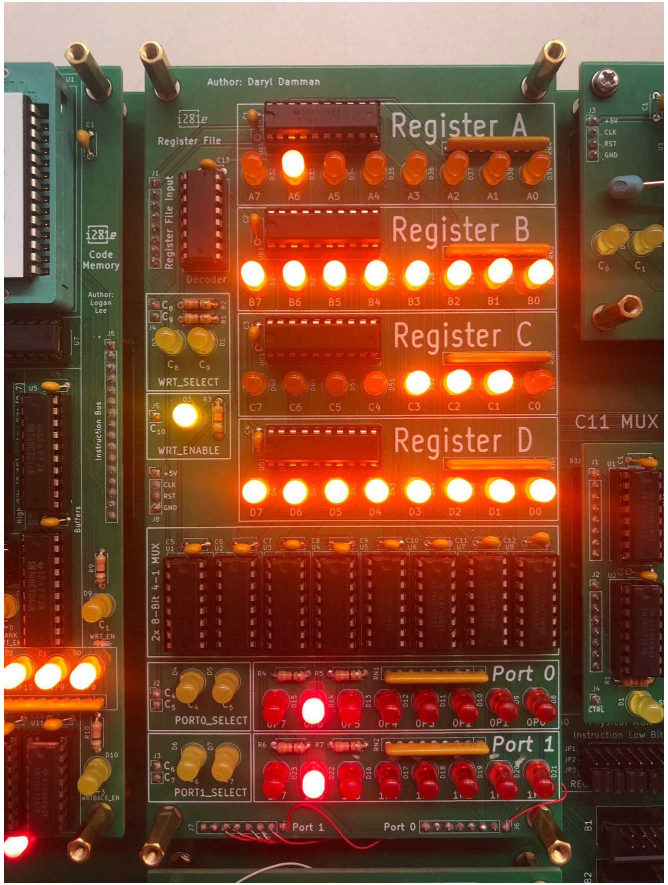
The CPU must be explainable to sophomore students in CprE 281.

Visualization



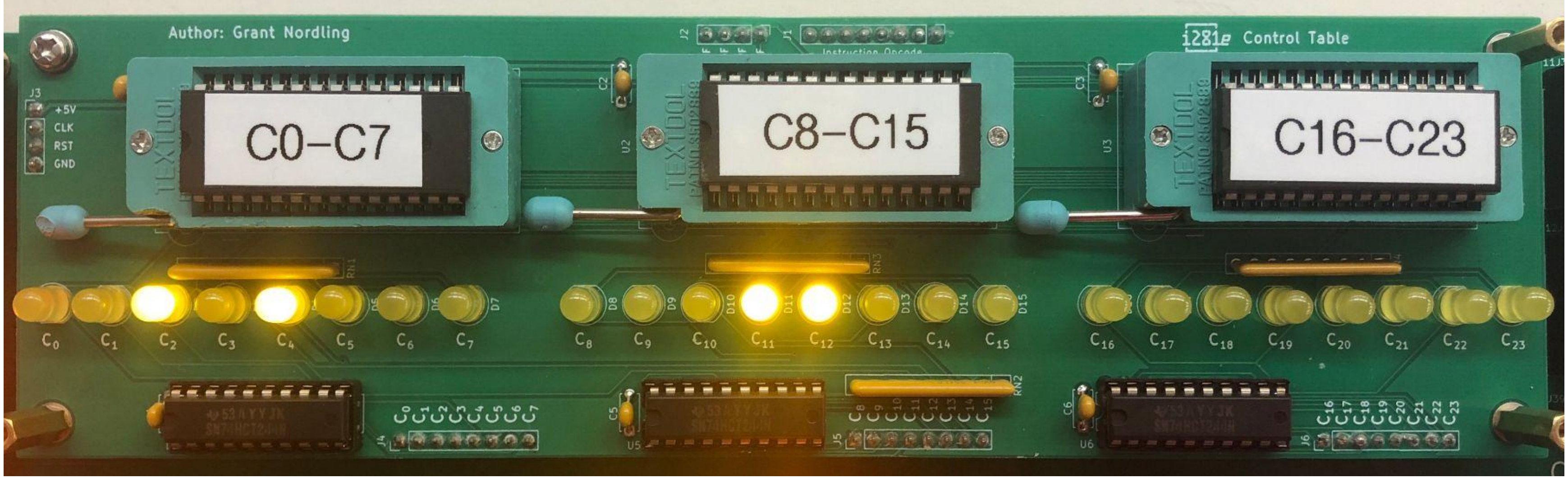
The CPU data path and control path must be visualized with LEDs.

Visualization



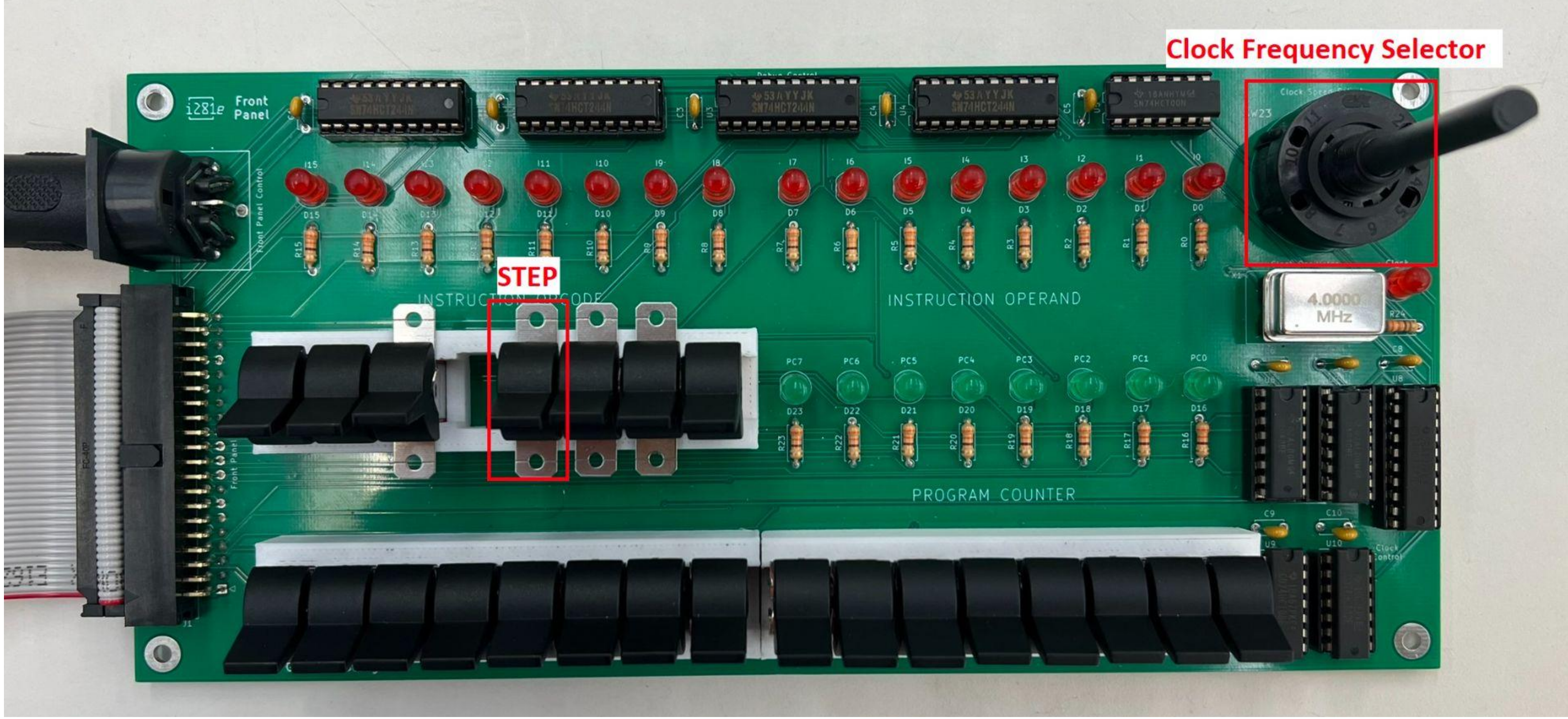
The CPU data path and control path must be visualized with LEDs.

Visualization



The CPU data path and control path must be visualized with LEDs.

Adjustable Clock



Clock Frequency Selector

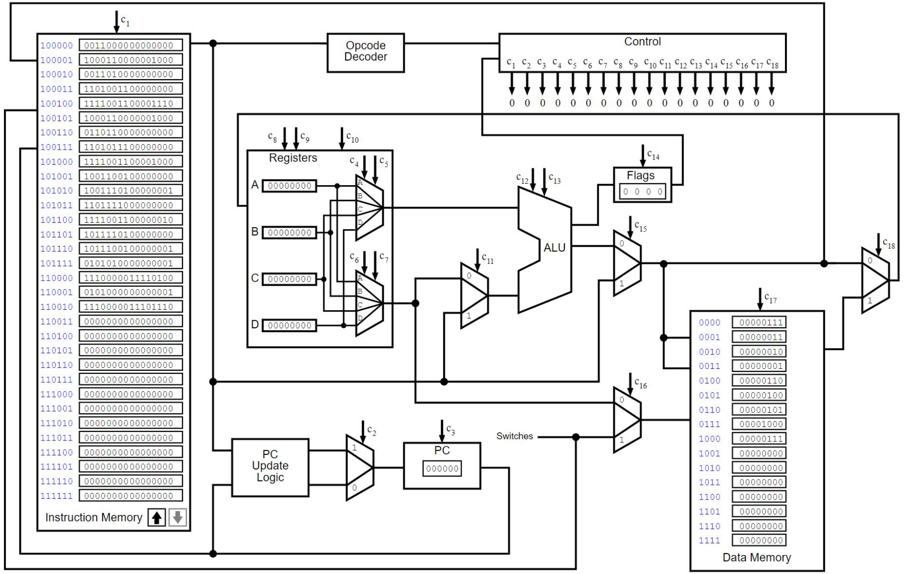
STEP

Adjustable CPU clock to enable stepping through individual instructions and operating at different clock frequencies.

Example Programs

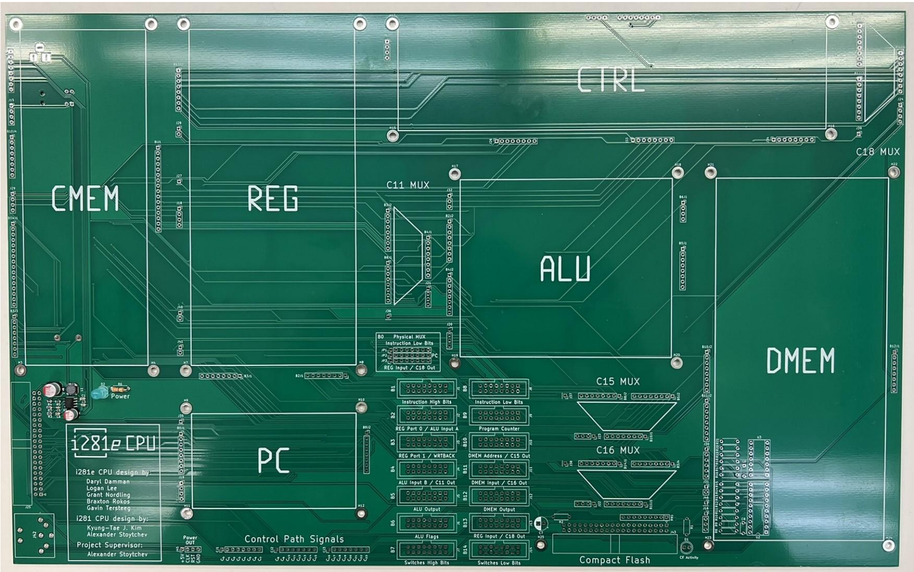
The CPU must be capable of playing the original i281 PONG video game and other example programs from FPGA and Web Simulator.

Design Similarity



100000	0011000000000000
100001	1000110000001000
100010	0011010000000000
100011	1101001100000000
100100	1111001100001110
100101	1000110000001000
100110	0110110000000000
100111	1101011100000000
101000	1111001100001000
101001	1001100100000000
101010	1001110100000001
101011	1101111000000000
101100	1111001100000010
101101	1011110100000000
101110	1011100100000001
101111	0101010000000001
110000	1110000011110100
110001	0101000000000001
110010	1110000011101110
110011	0000000000000000
110100	0000000000000000
110101	0000000000000000
110110	0000000000000000
110111	0000000000000000
111000	0000000000000000
111001	0000000000000000
111010	0000000000000000
111011	0000000000000000
111100	0000000000000000
111101	0000000000000000
111110	0000000000000000
111111	0000000000000000

0000	00000111
0001	00000011
0010	00000010
0011	00000001
0100	00000110
0101	00000100
0110	00000101
0111	00001000
1000	00000111
1001	00000000
1010	00000000
1011	00000000
1100	00000000
1101	00000000
1110	00000000
1111	00000000



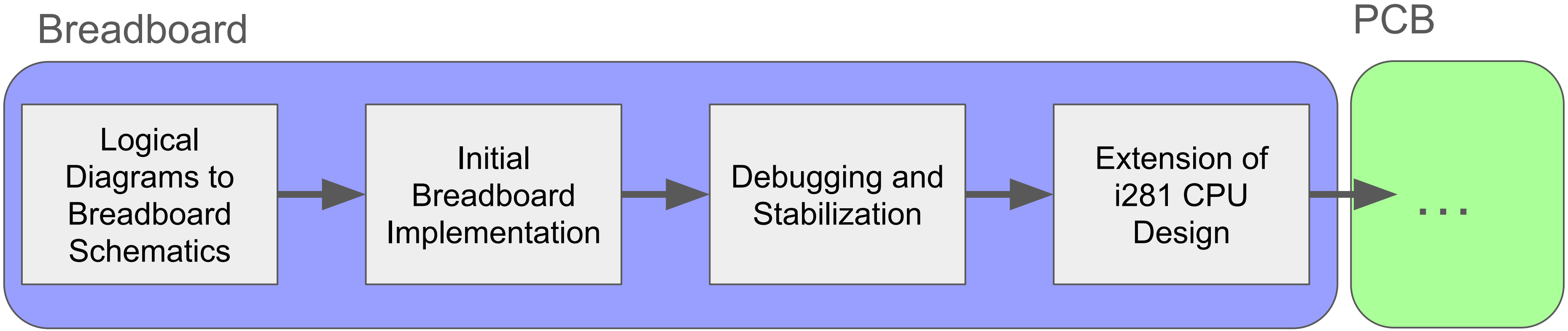
The CPU design must be as close as possible to the original design.

It must look *cool*

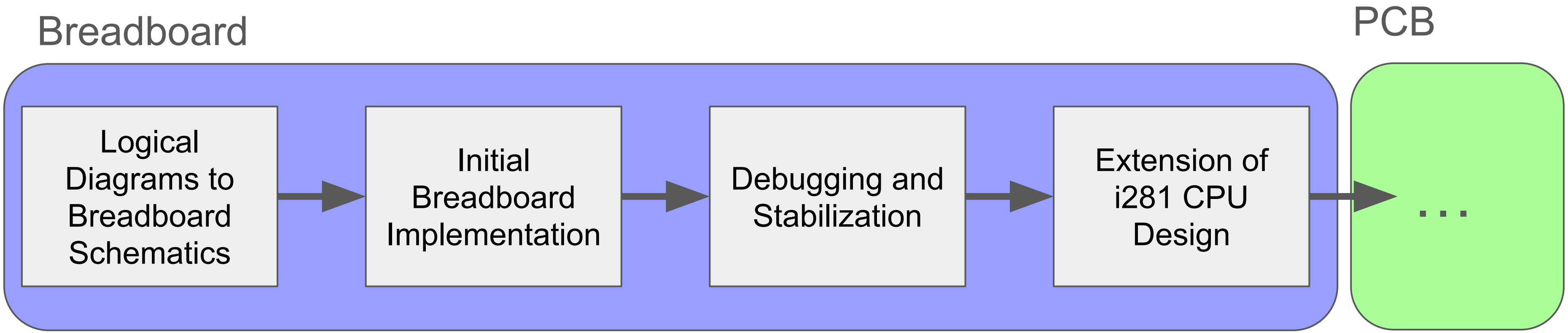
Design Process

Daryl Damman
Software Engineering

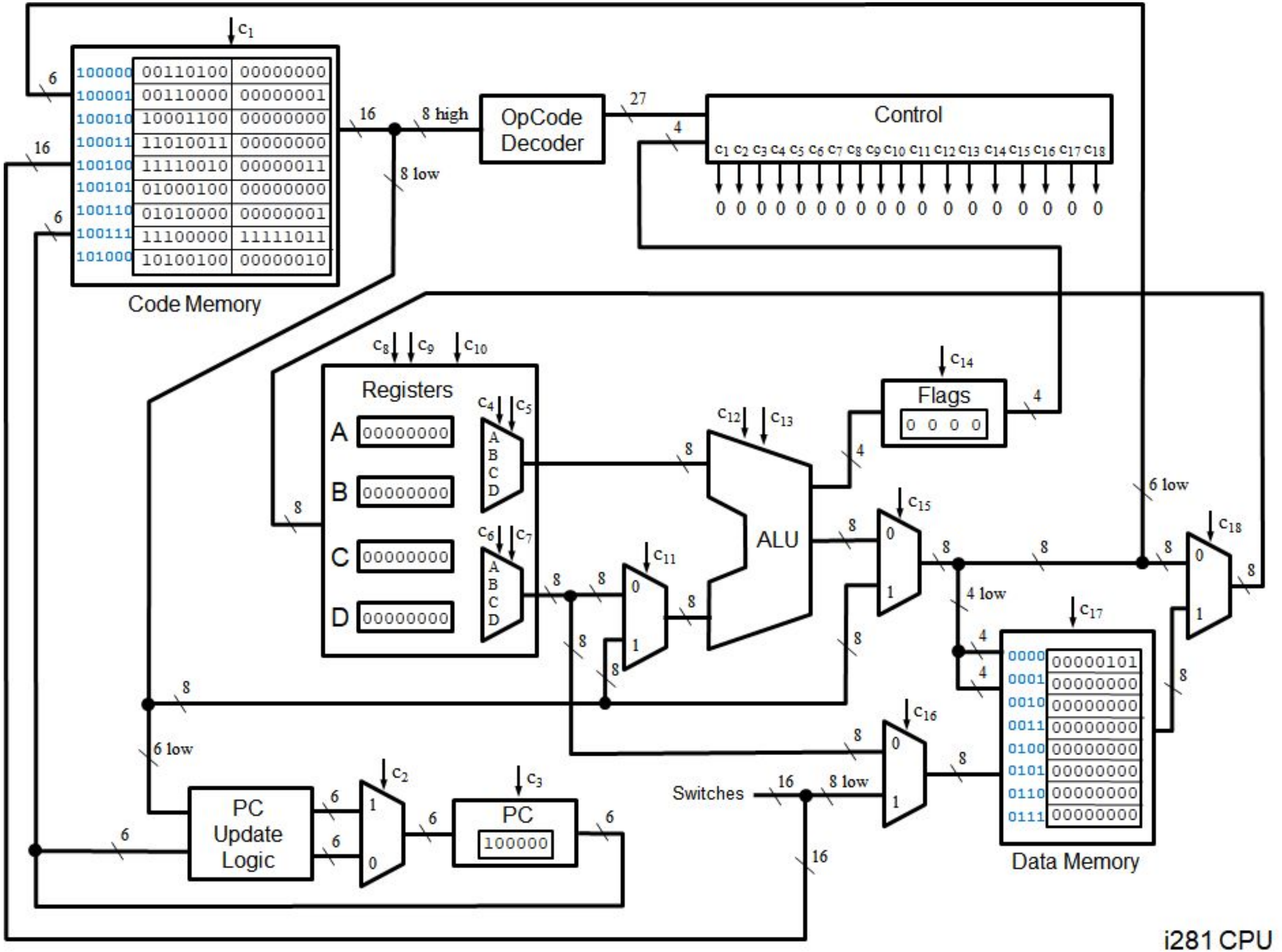
Design Iterations


Breadboard
PCB
Logical Diagrams to Breadboard Schematics
Initial Breadboard Implementation
Debugging and Stabilization
Extension of i281 CPU Design
...

Design Iterations


Breadboard
PCB
Logical Diagrams to Breadboard Schematics
Initial Breadboard Implementation
Debugging and Stabilization
Extension of i281 CPU Design
. . .

Let’s look at the Arithmetic Logic Unit (ALU) as an example.



Code Memory

100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

Registers

A	00000000
B	00000000
C	00000000
D	00000000

Flags: 0 0 0 0

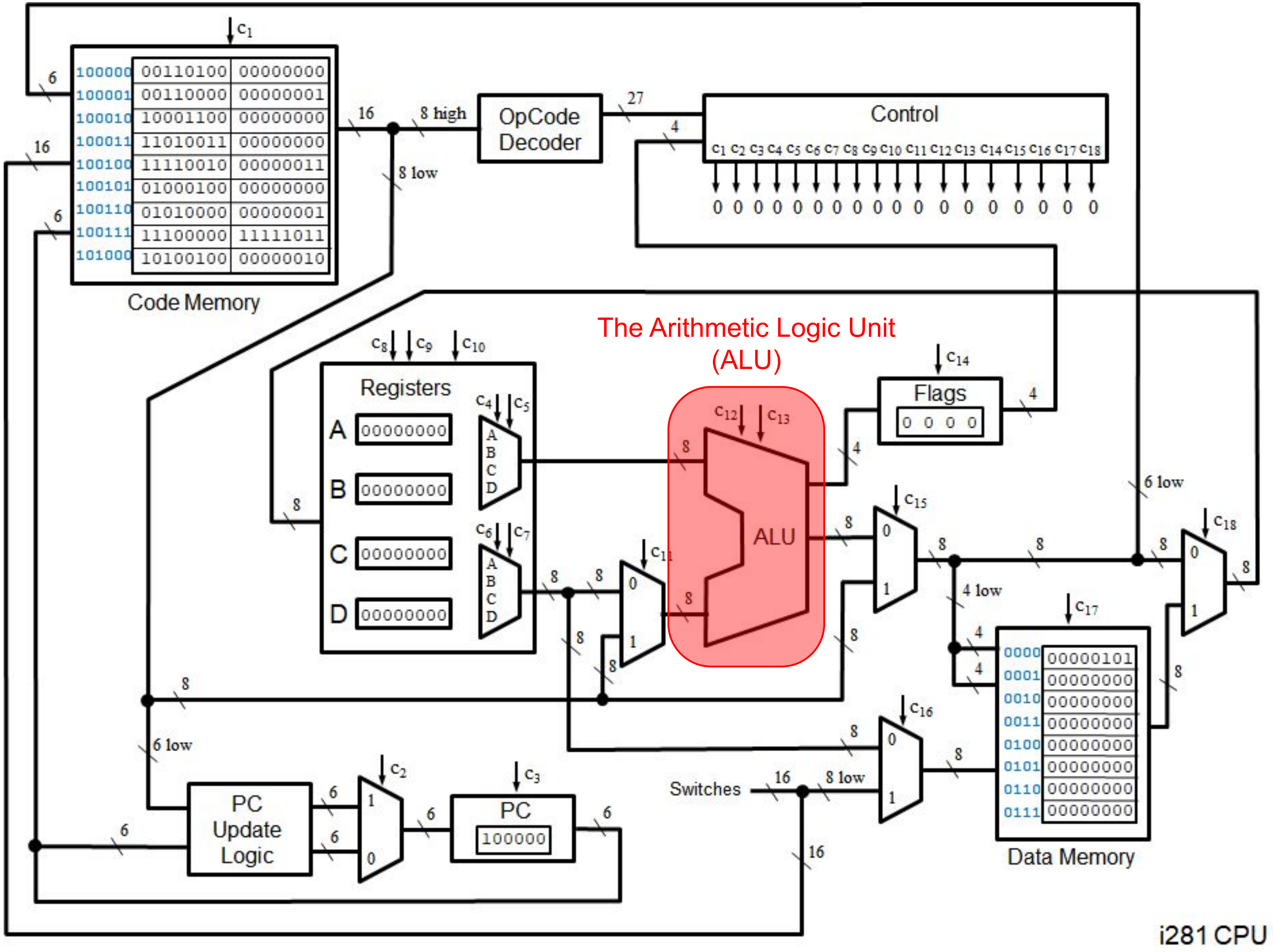
PC: 100000

Data Memory

0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

i281 CPU

The Arithmetic Logic Unit (ALU)



Code Memory

100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

Registers

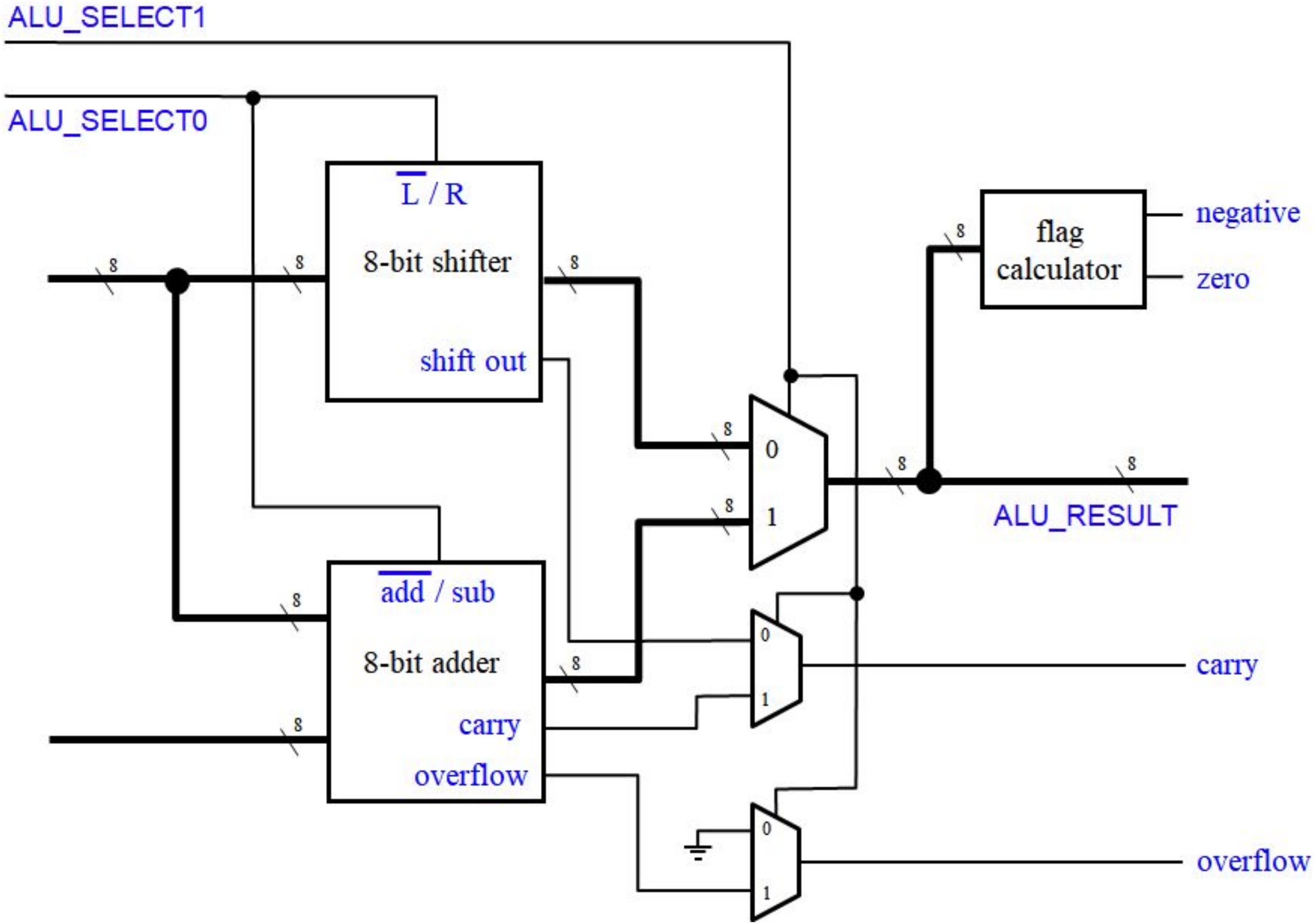
A	00000000
B	00000000
C	00000000
D	00000000

Data Memory

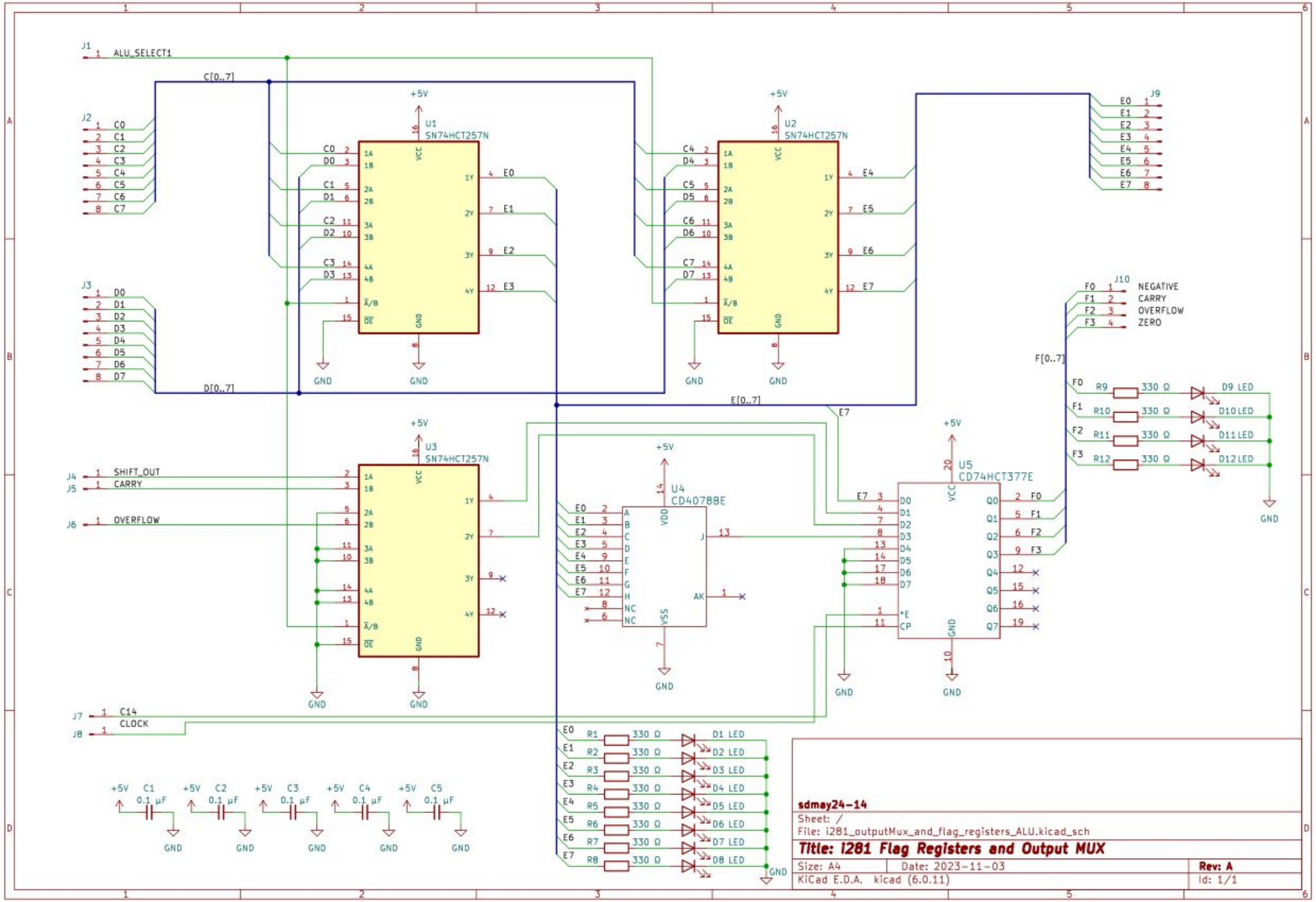
0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

i281 CPU

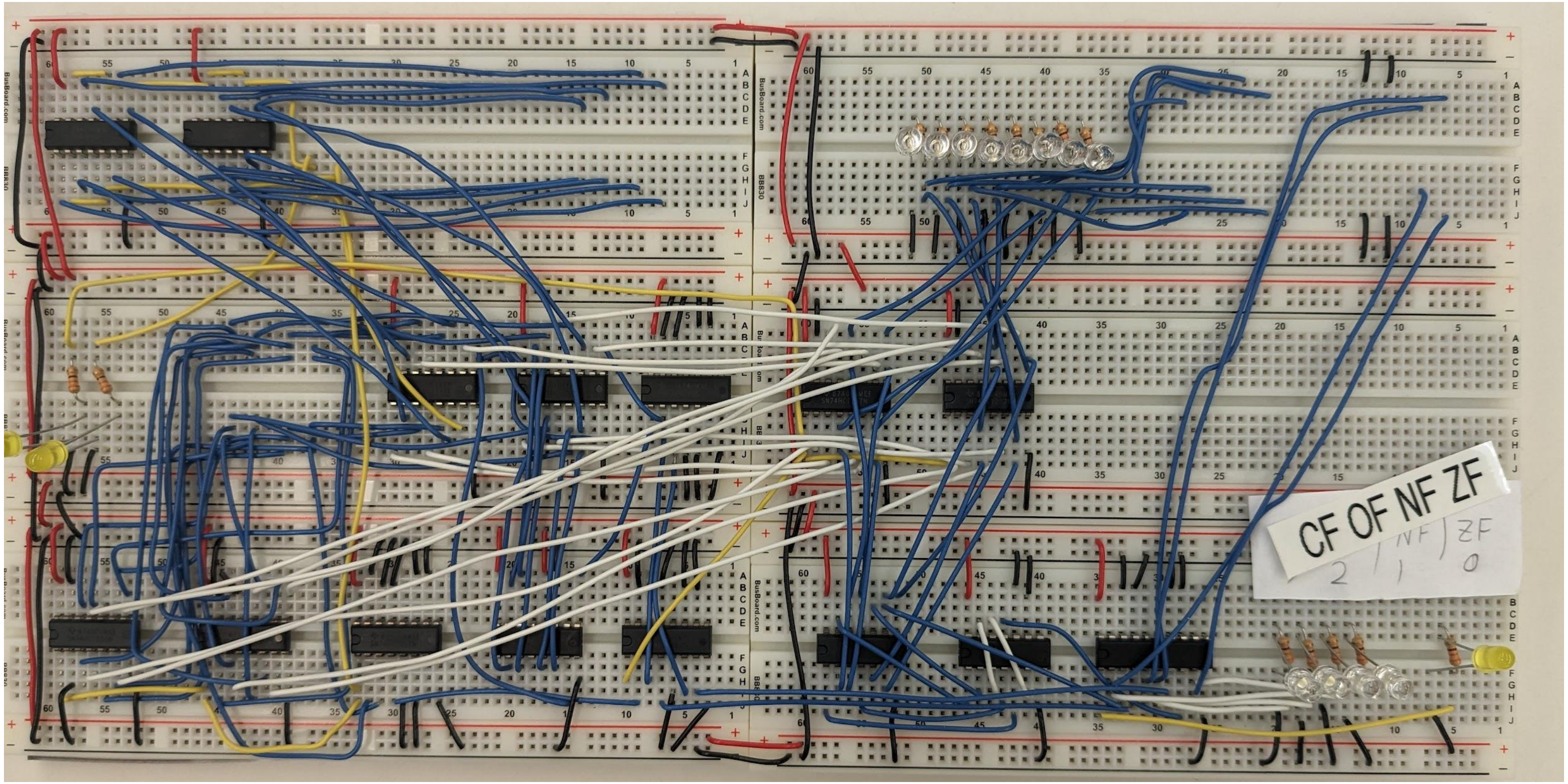
ALU Logic


ALU_SELECT1
ALU_SELECT0
$\overline{L}$ / R
8-bit shifter
shift out
$\overline{\text{add}}$ / sub
8-bit adder
carry
overflow
8
0
1
flag
calculator
negative
zero
ALU_RESULT
carry
overflow

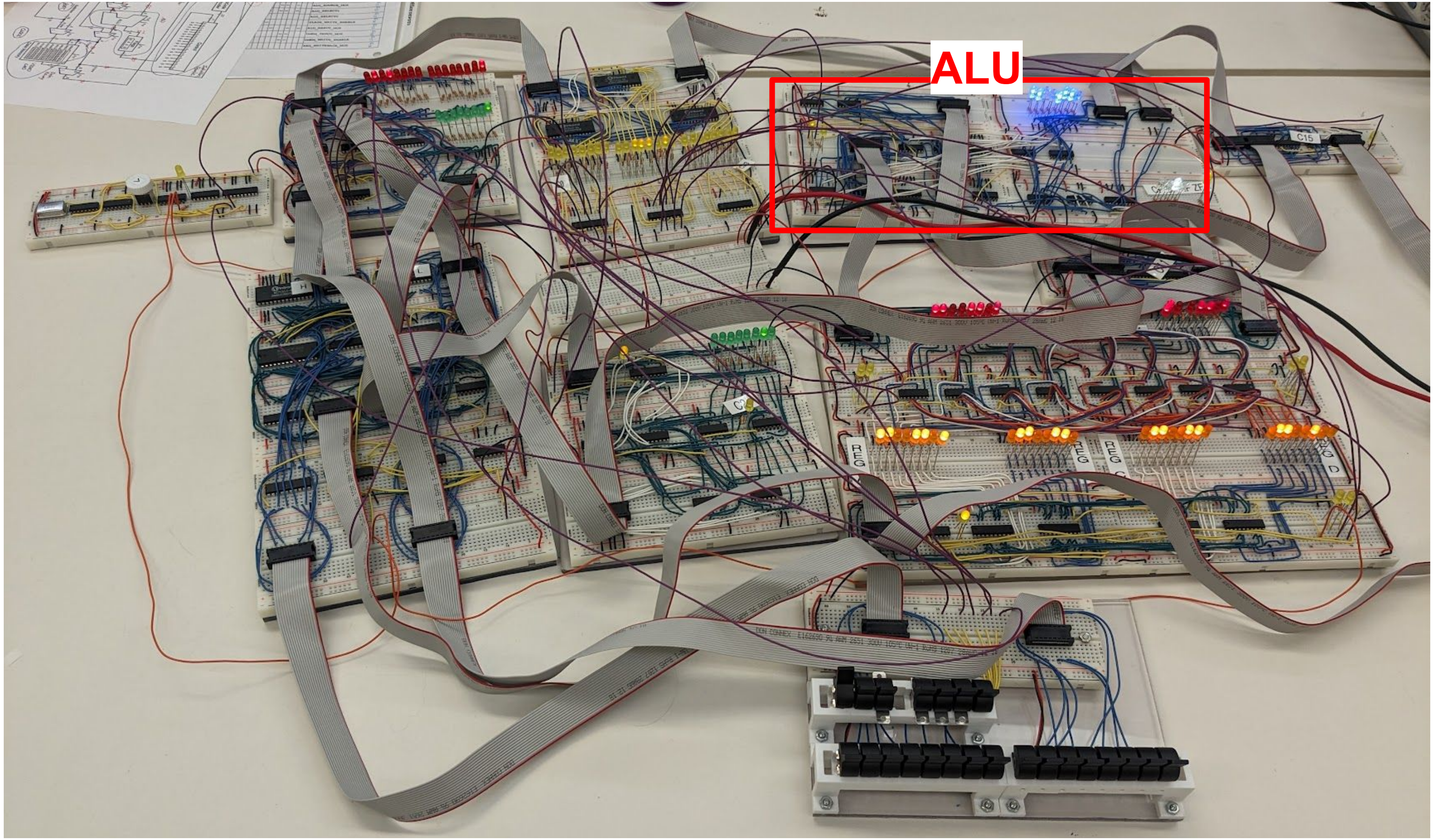
ALU Schematics



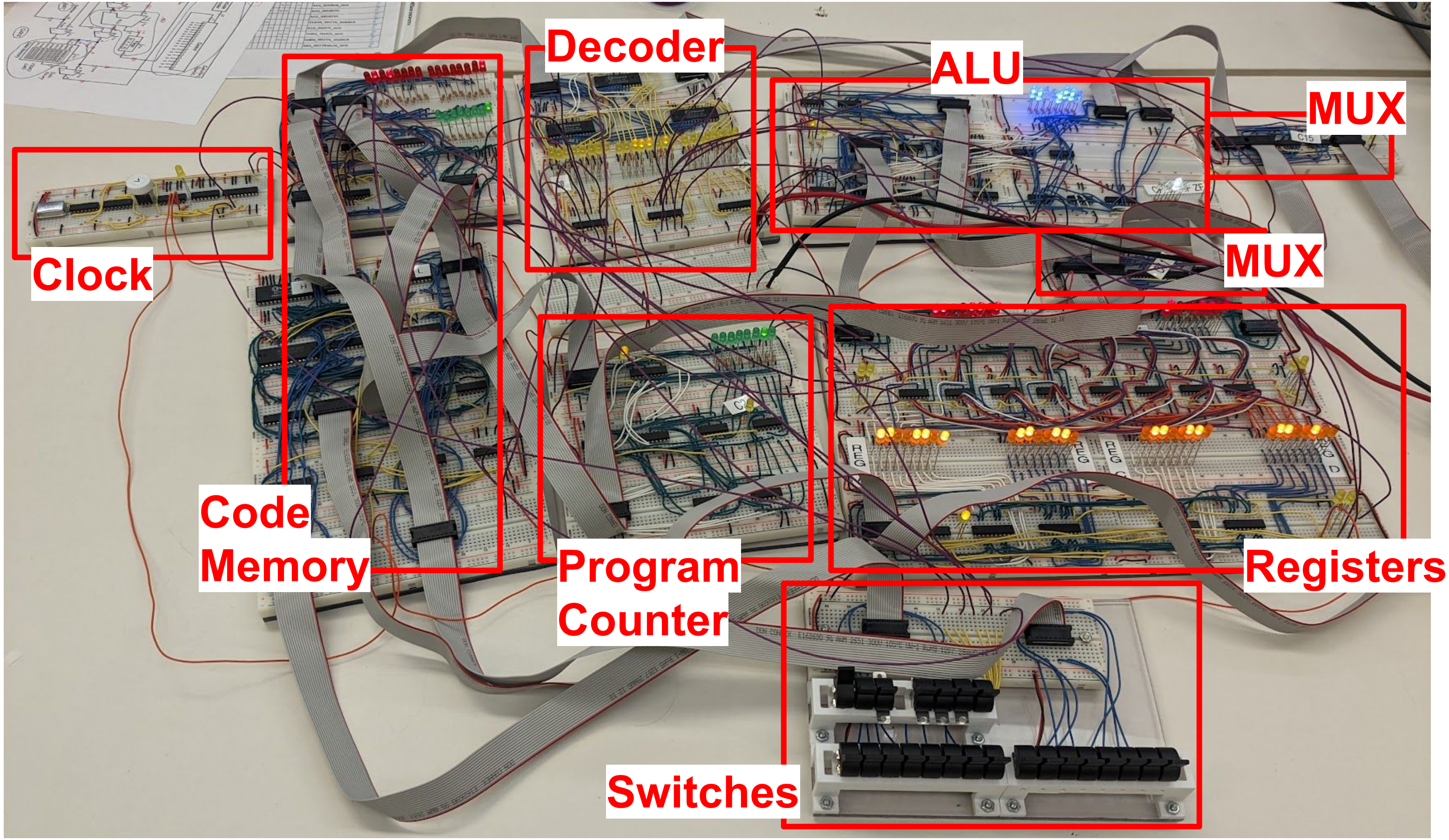
ALU Implementation



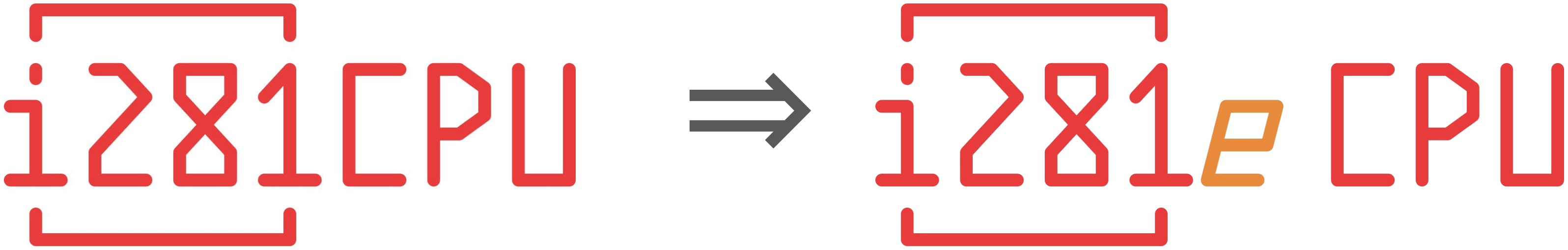
Where’s the ALU?



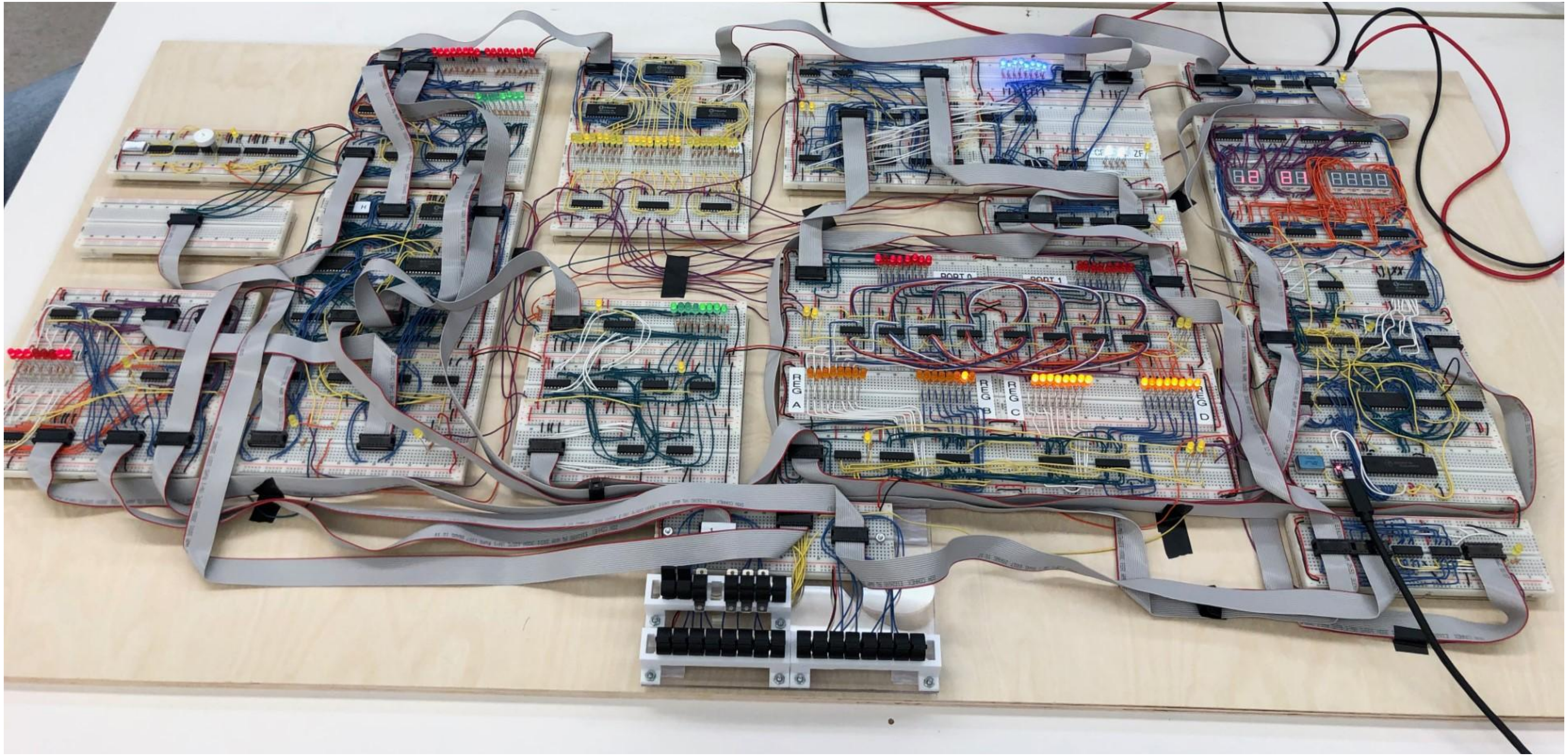
Minimum Viable Processor (Fall 2023)



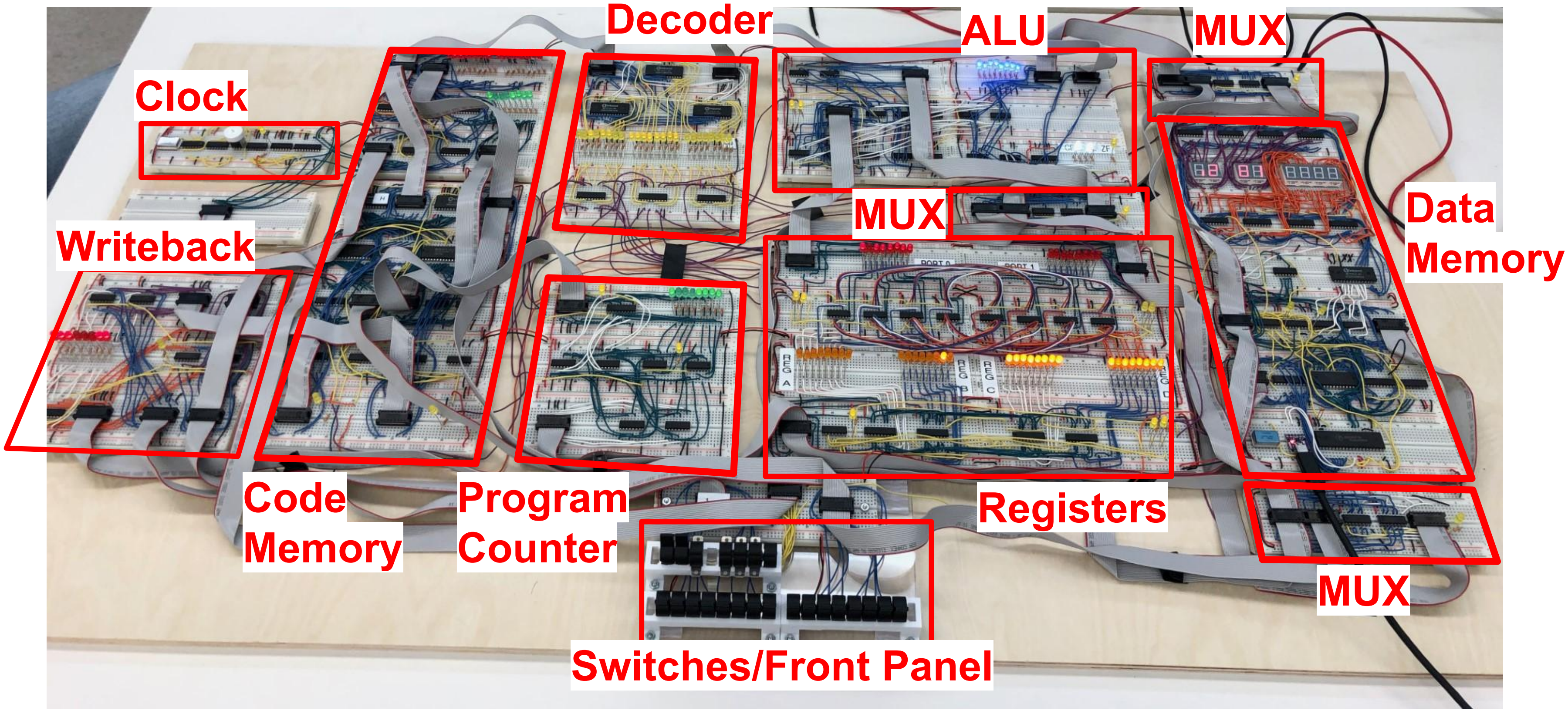
Expansion of the Original Design



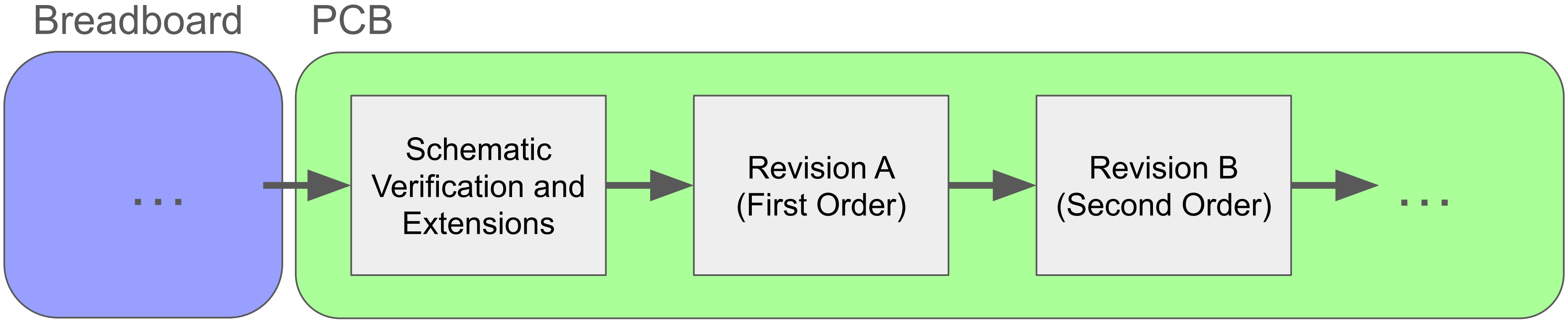
Breadboard Machine (Early Spring 2024)



Breadboard Machine (Early Spring 2024)

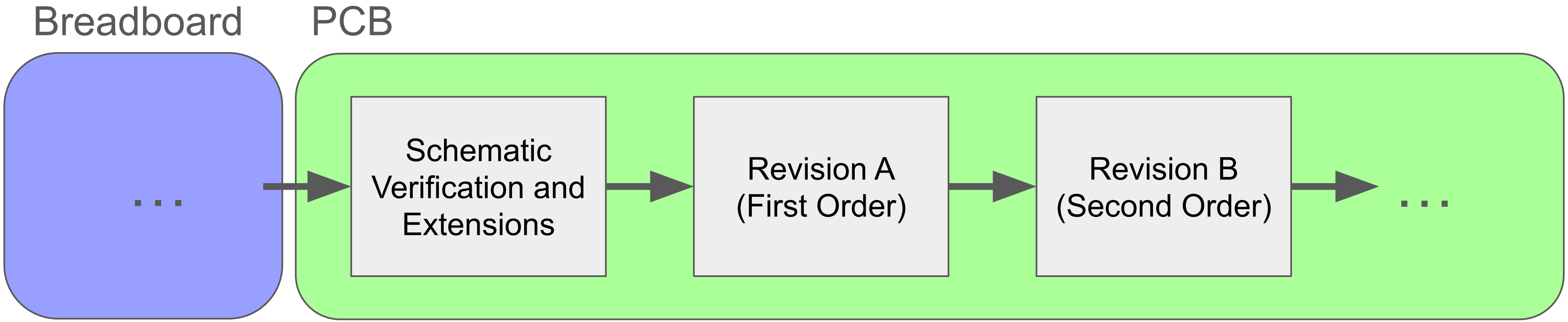


Design Iterations



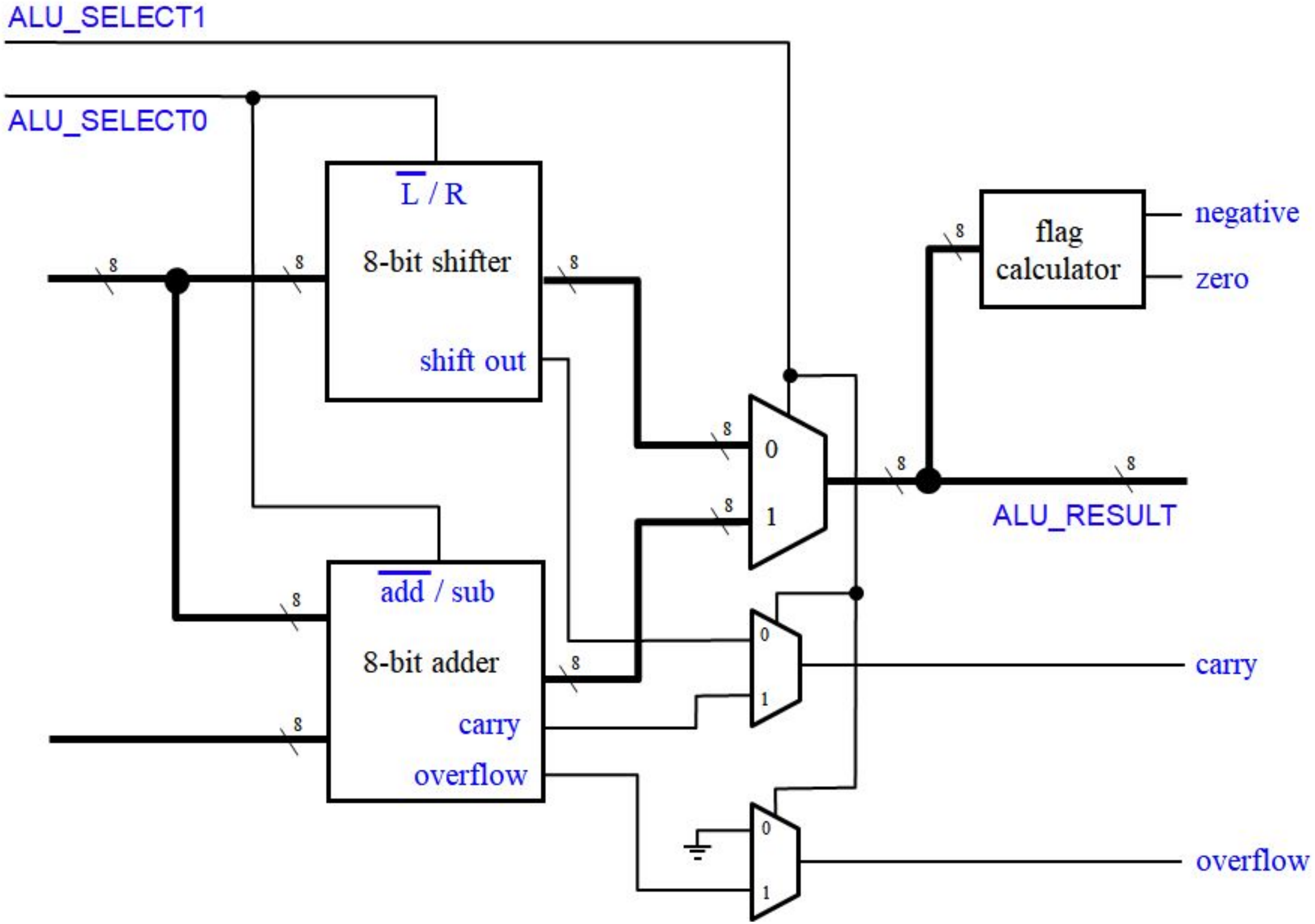
Breadboard
PCB
...
Schematic Verification and Extensions
Revision A (First Order)
Revision B (Second Order)
...

Design Iterations

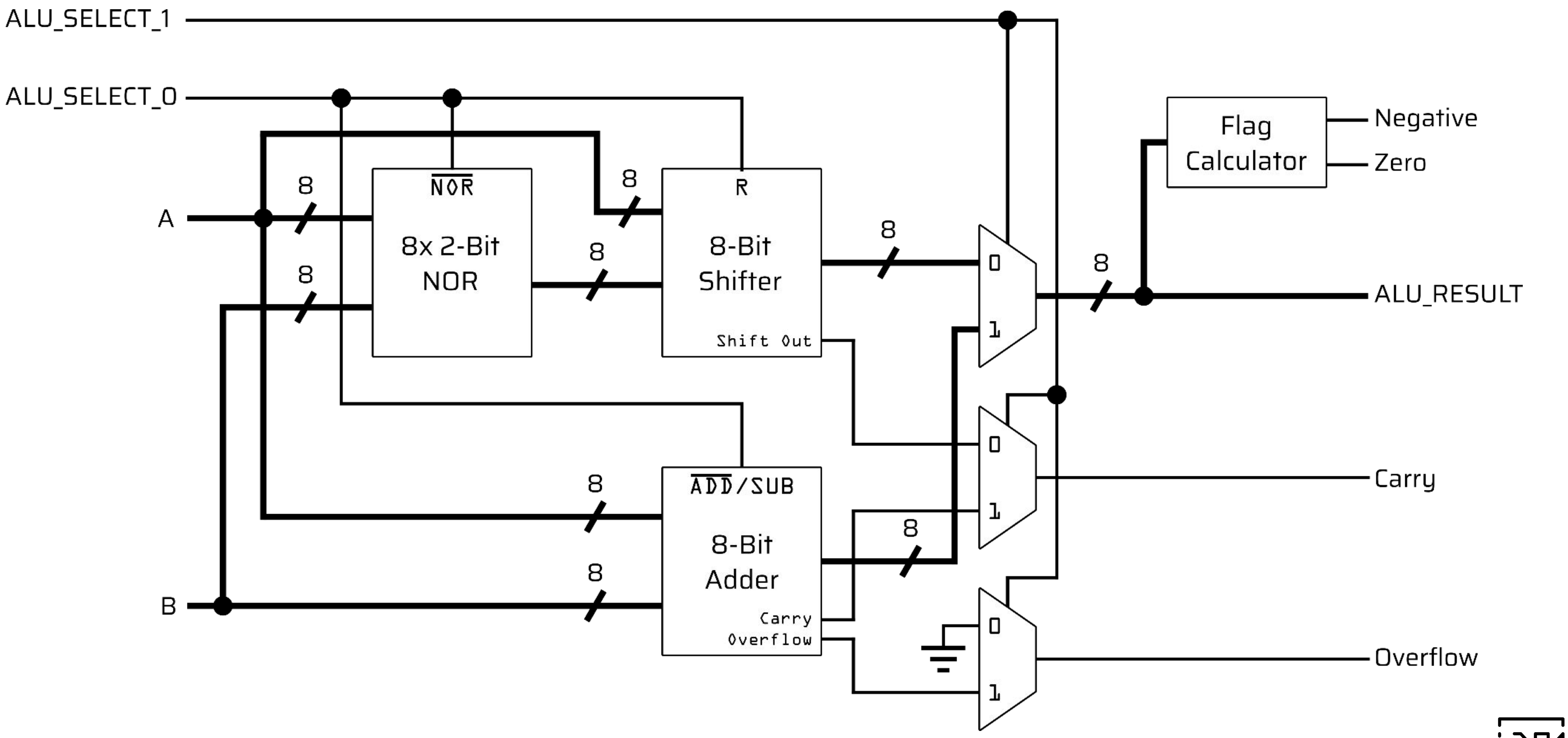

Breadboard
PCB
...
Schematic Verification and Extensions
Revision A (First Order)
Revision B (Second Order)
...

Let's look at the Arithmetic Logic Unit (ALU) again, for which we built two alternative PCB designs.

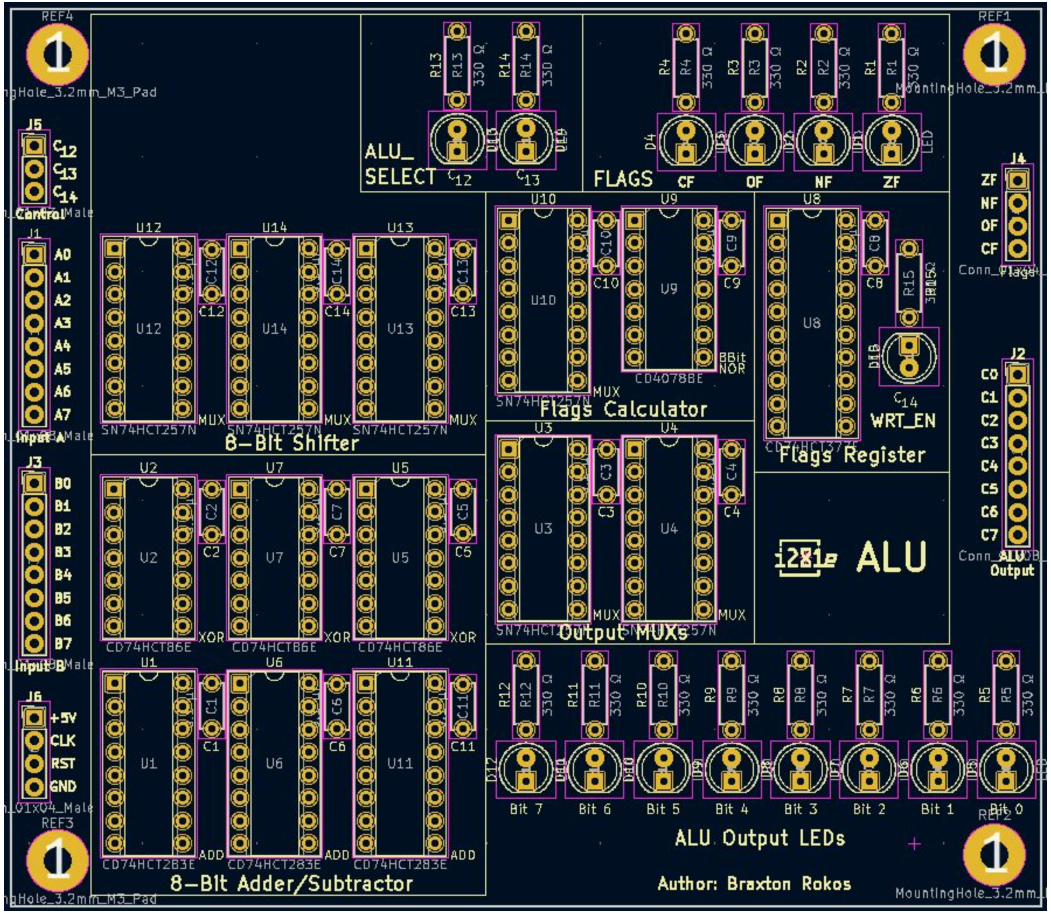
ALU Logic


ALU_SELECT1
ALU_SELECT0
$\overline{L}$ / R
8-bit shifter
shift out
$\overline{add}$ / sub
8-bit adder
carry
overflow
flag calculator
negative
zero
ALU_RESULT
carry
overflow
0
1
0
1
0
1
8
8
8
8
8
8
8
8
8
8
8
8

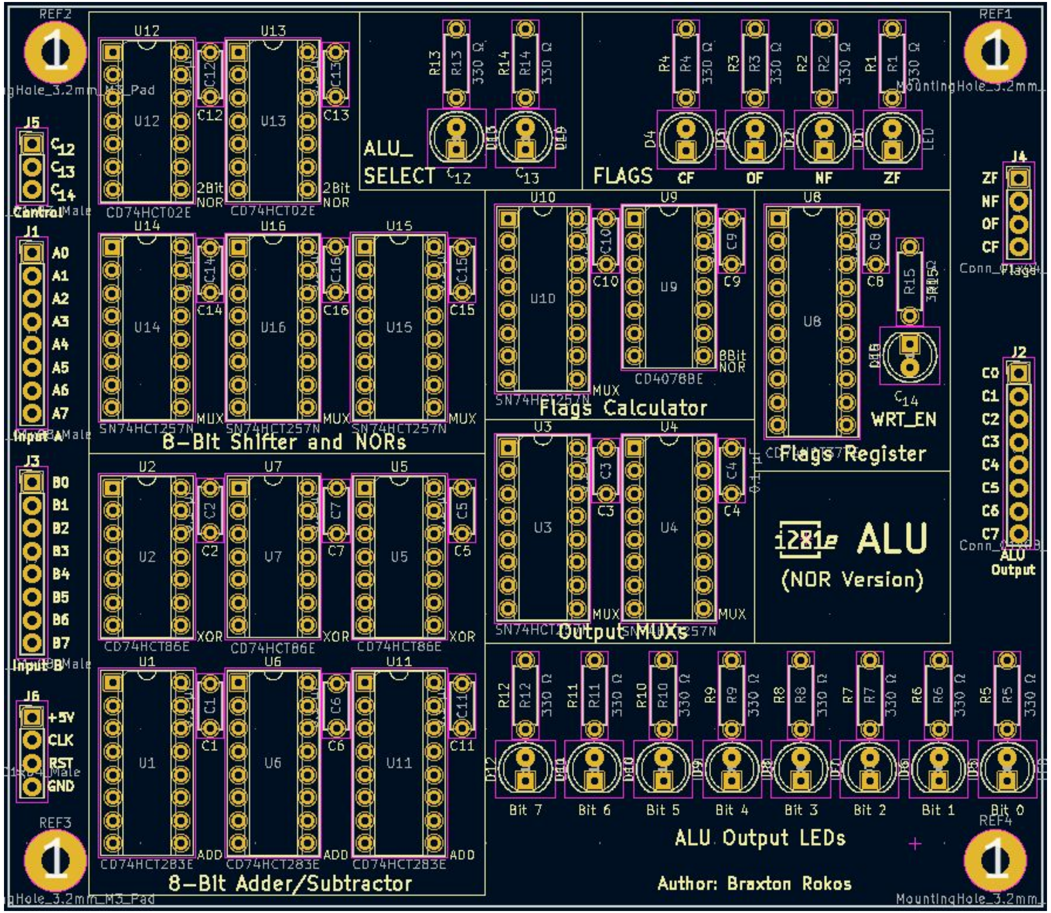
ALU Logic (NOR Version)



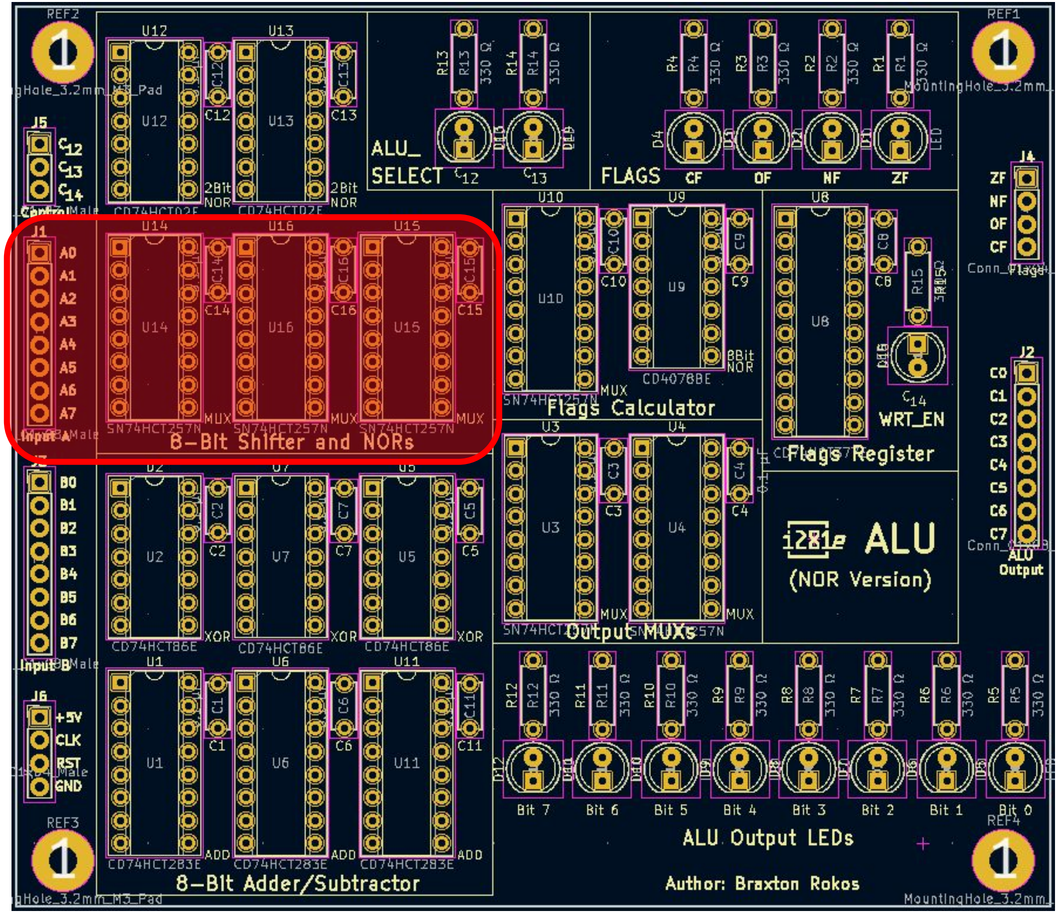
ALU Layout (First Version)



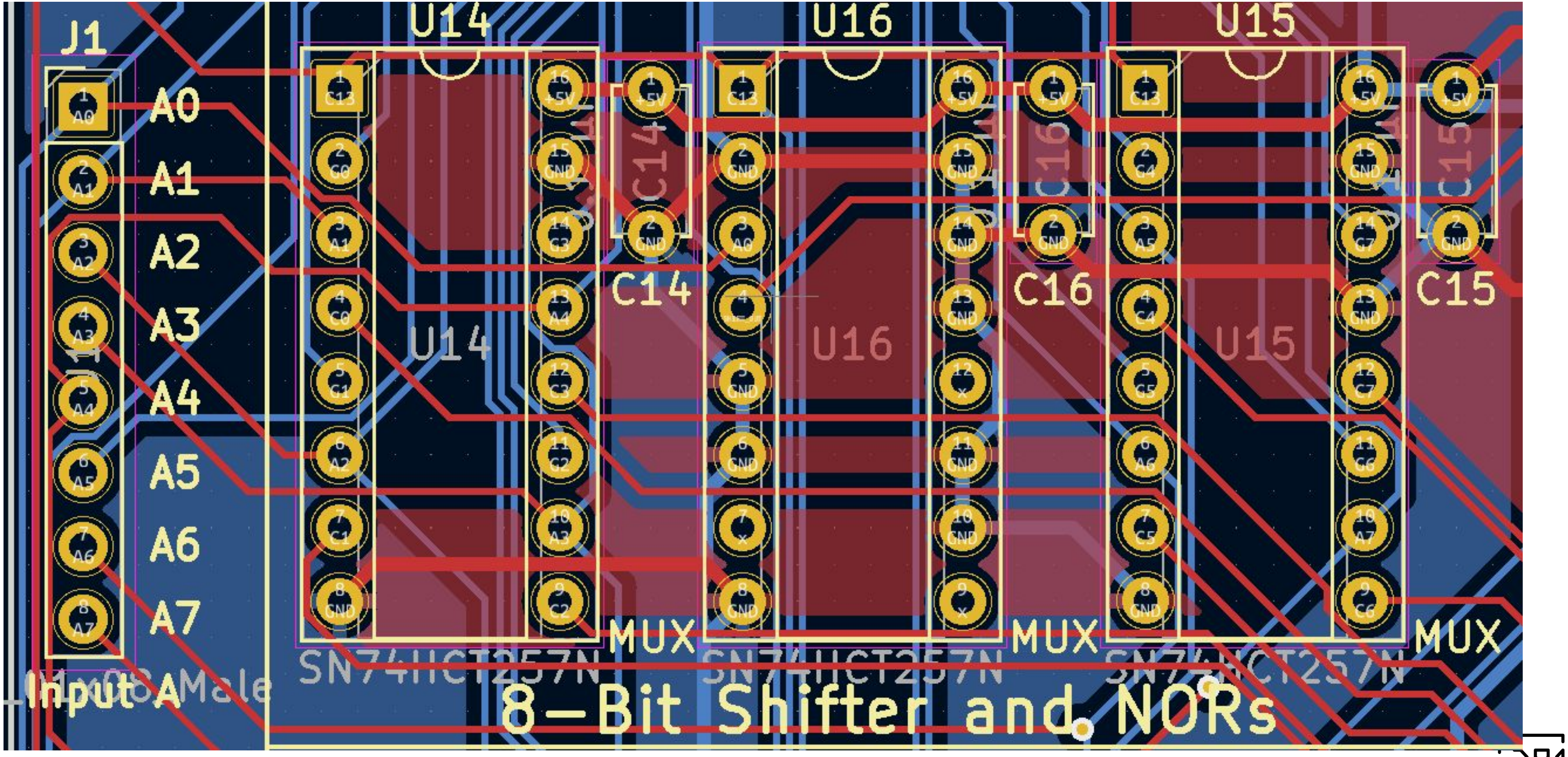
ALU Layout (Second Version)



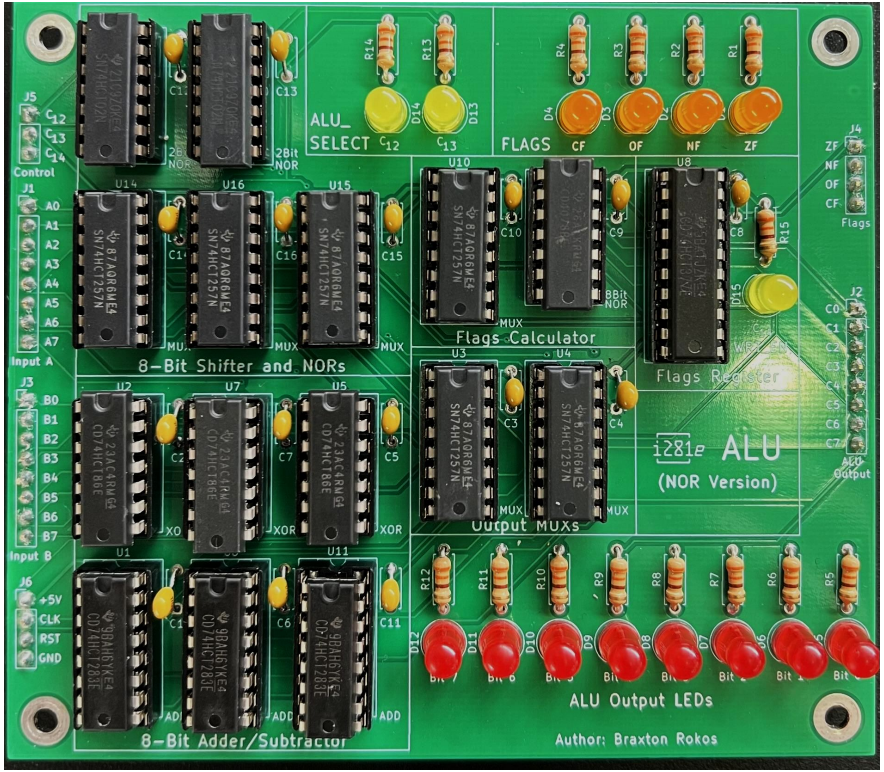
ALU Layout (Second Version)

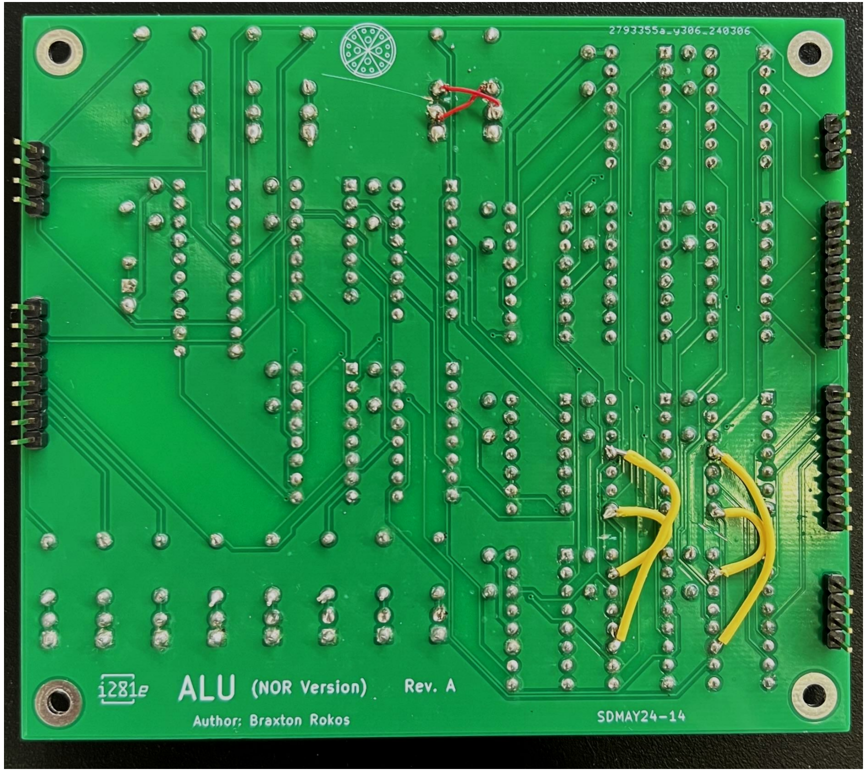


ALU Routing

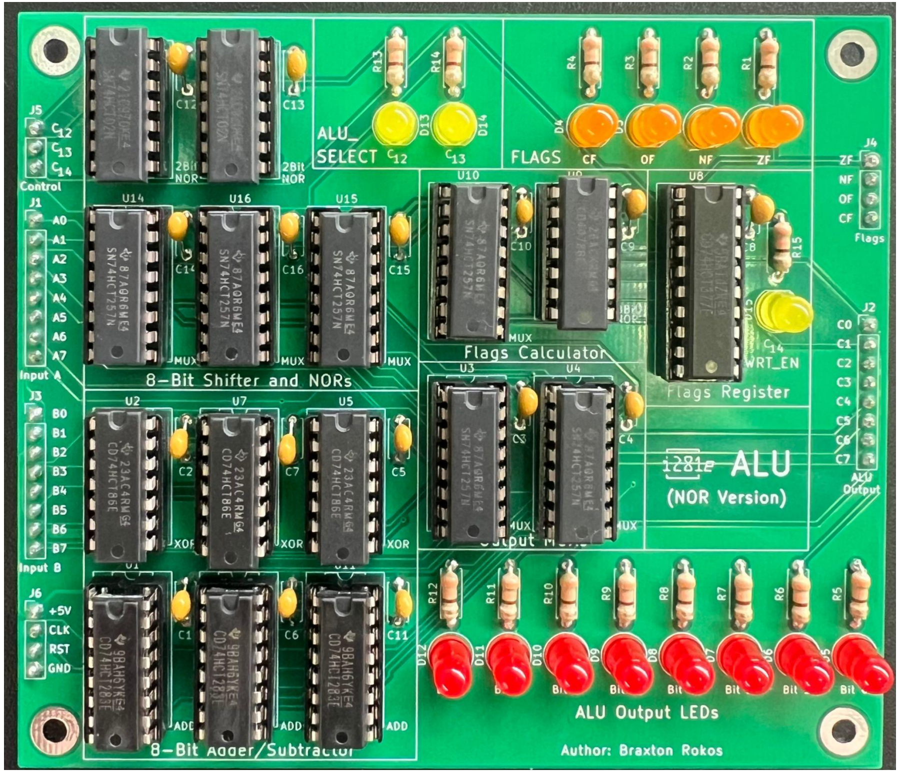

J1
A0
A1
A2
A3
A4
A5
A6
A7
Input A
U14
U16
U15
C14
C16
C15
MUX
MUX
MUX
SN74HCT257N
SN74HCT257N
SN74HCT257N
8-Bit Shifter and NORs

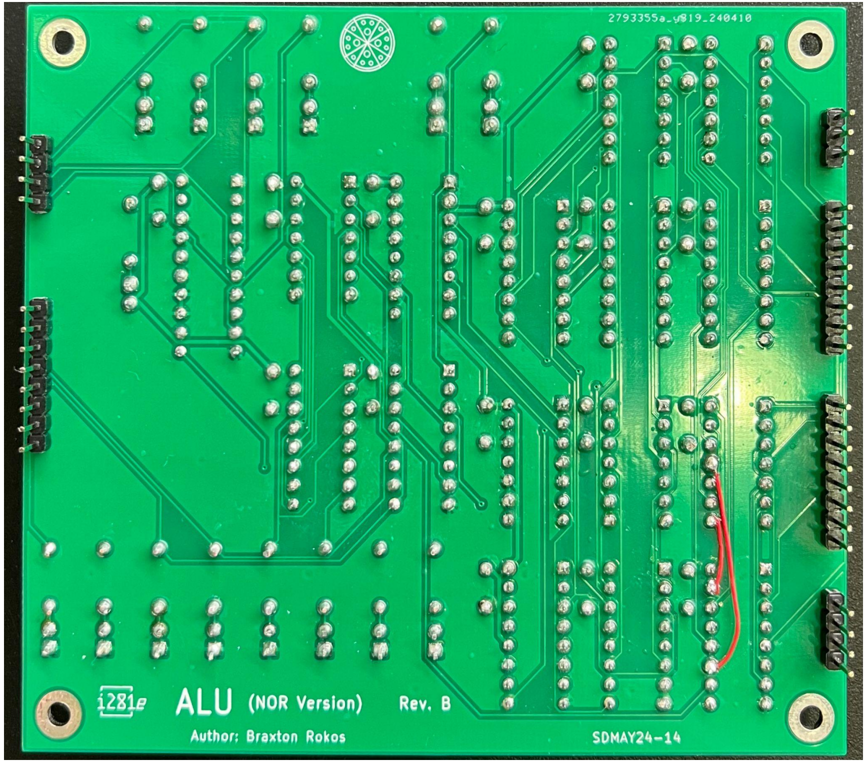
ALU PCB Implementation (Revision A)

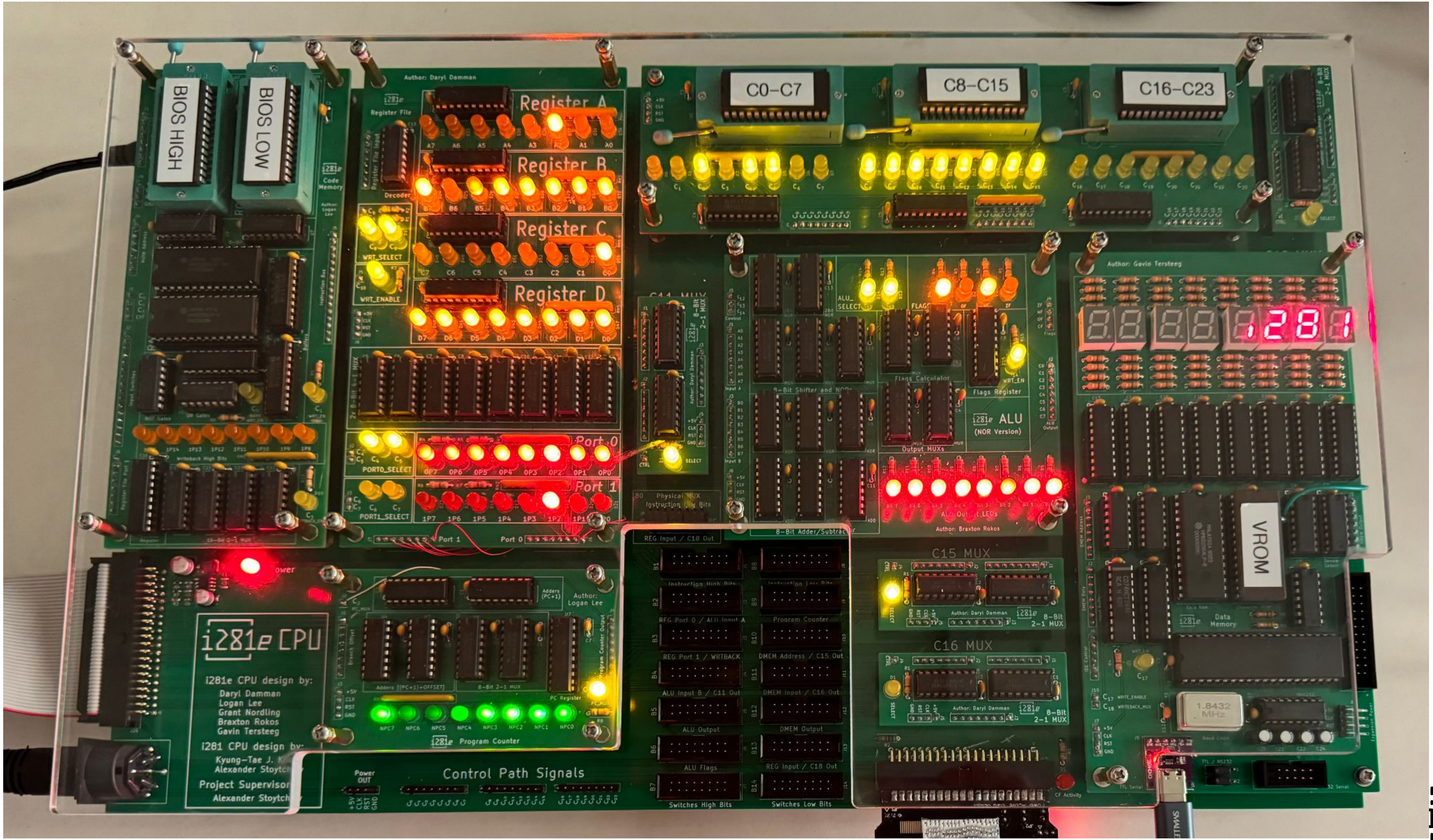


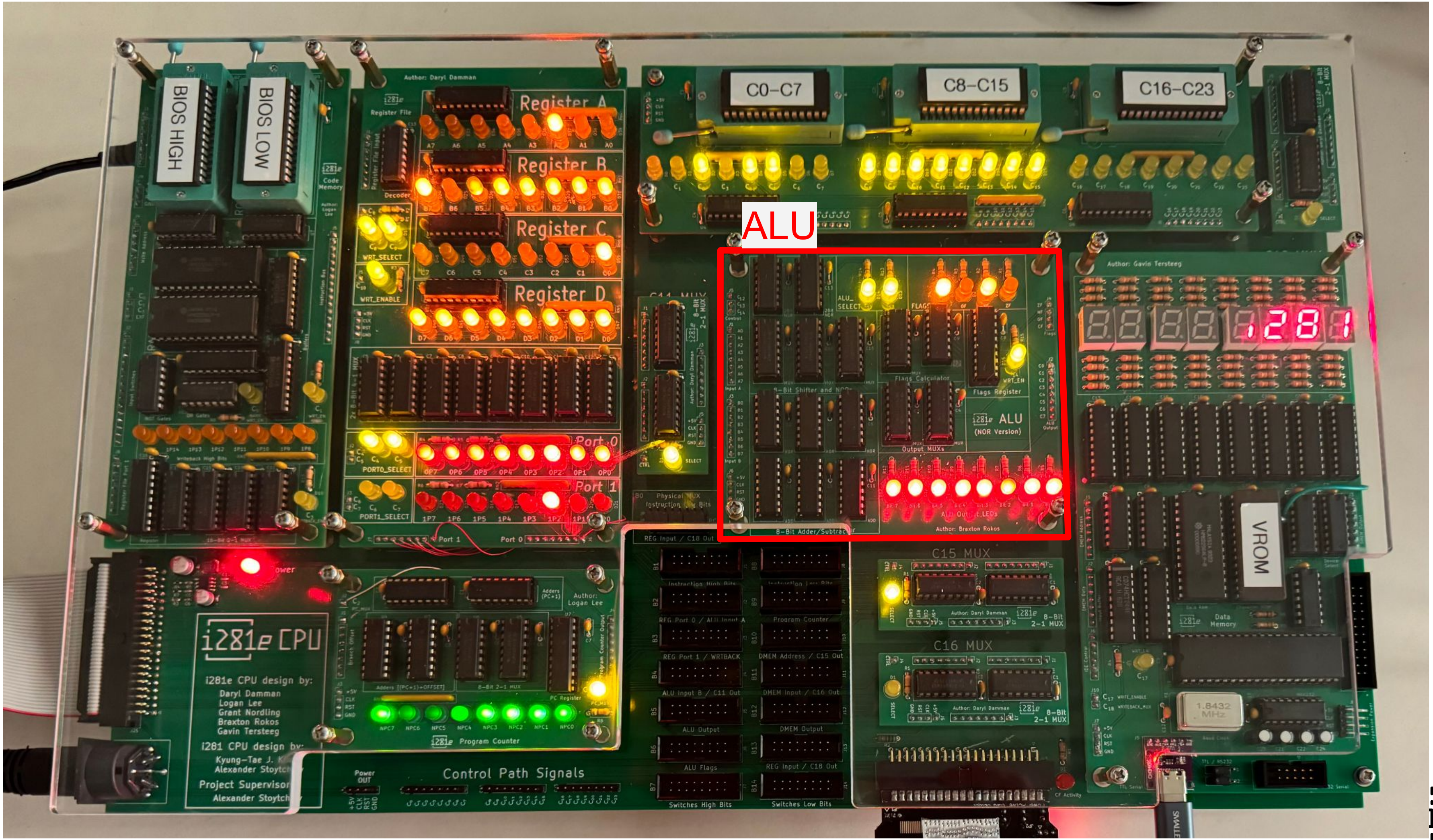


ALU PCB Implementation (Revision B)



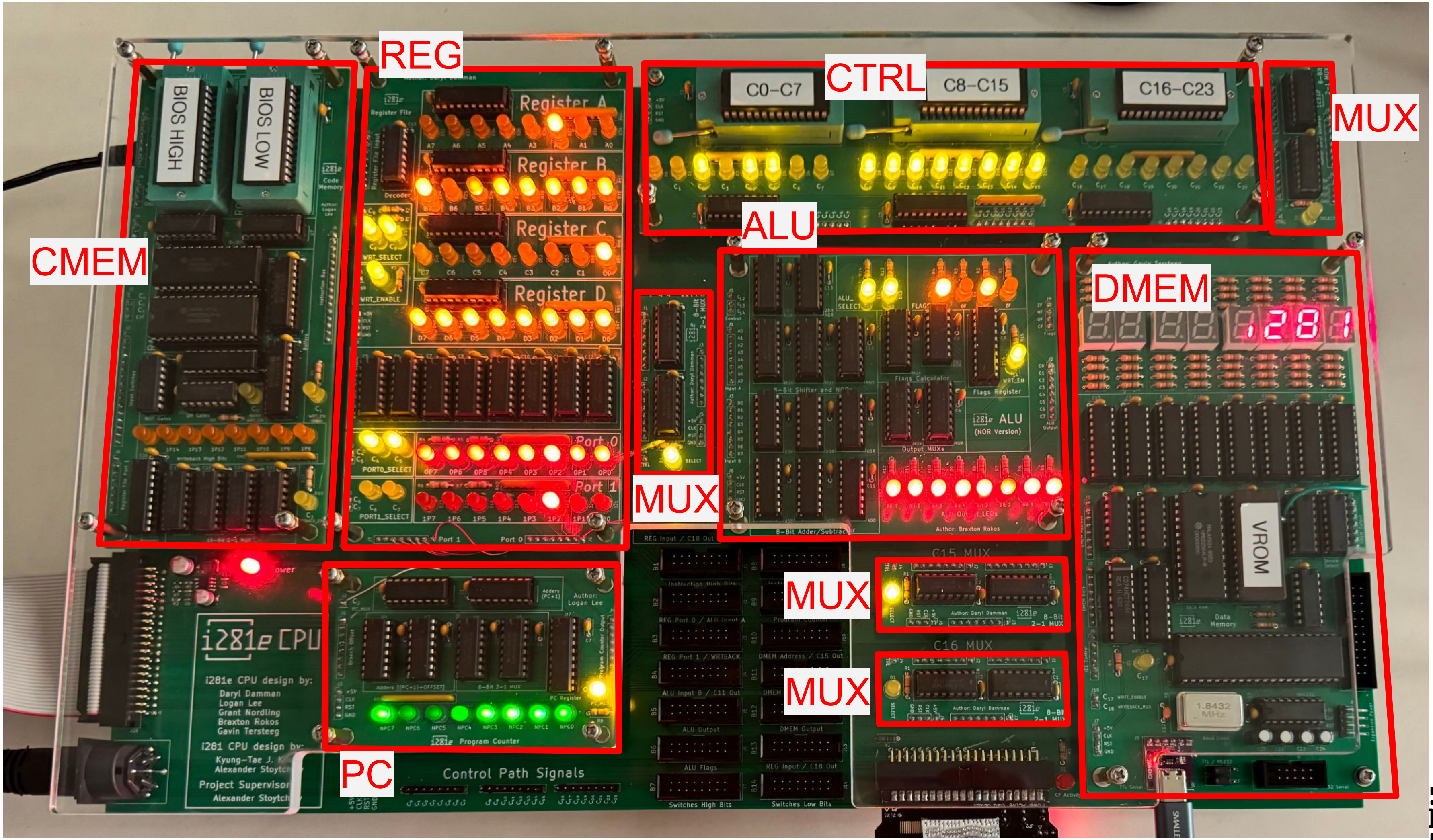


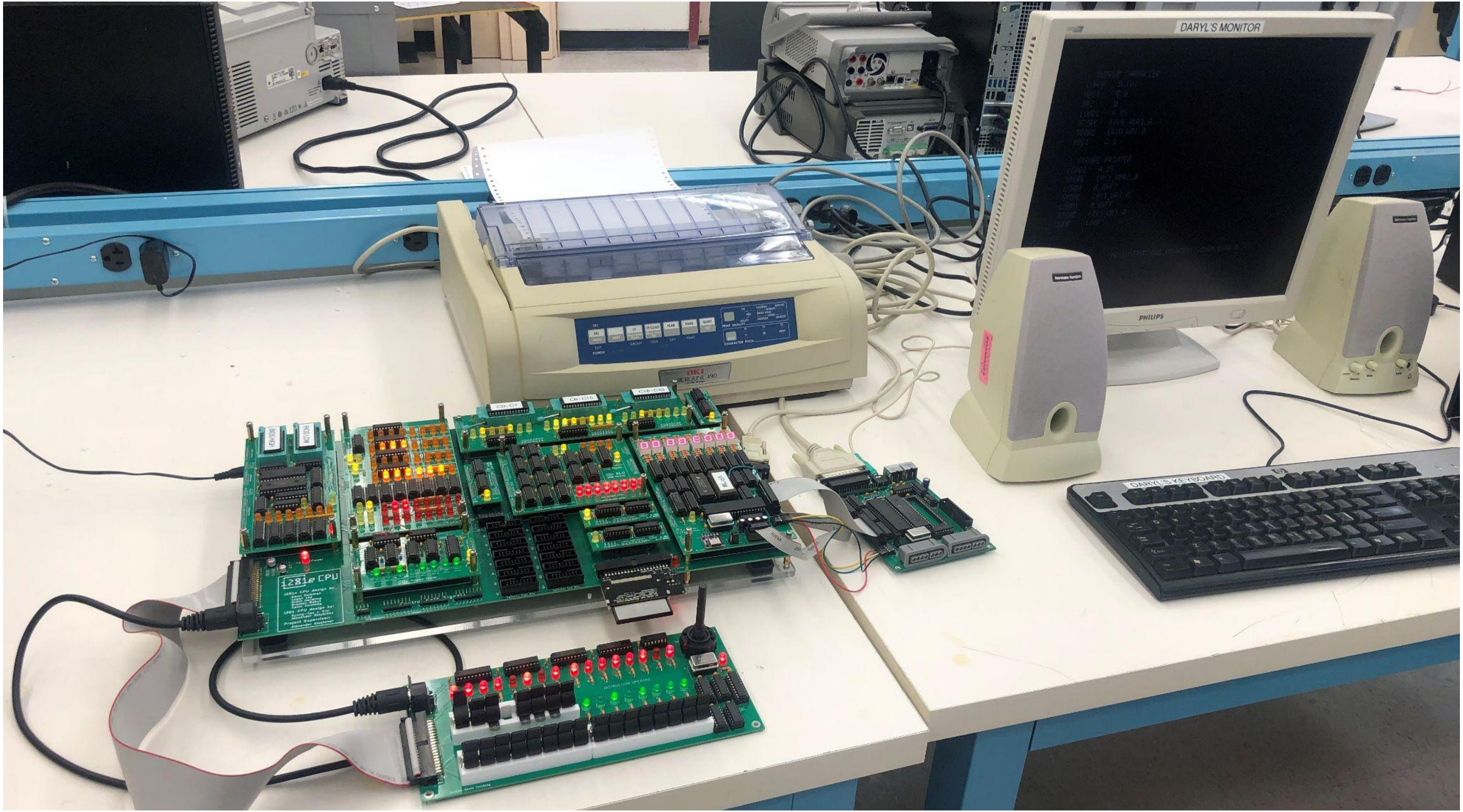

BIOS HIGH
BIOS LOW
Author: Daryl Damman
Register File
Code Memory
Register A
Register B
Register C
Register D
WRT_SELECT
WRT_ENABLE
Port 0
Port 1
PORT0_SELECT
PORT1_SELECT
C0–C7
C8–C15
C16–C23
C11 MUX
8-Bit 2-1 MUX
Physical MUX Instruction Low Bits
ALU_SELECT
FLAGS
Flags Calculator
Flags Register
8-Bit Shifter and NOR
i281e ALU
(NOR Version)
Output MUXs
ALU Output LEDs
8-Bit Adder/Subtractor
Author: Braxton Rokos
Author: Gavin Tersteeg
i281
VROM
Data Memory
1.8432 MHz
C15 MUX
C16 MUX
Author: Daryl Damman
8-Bit 2-1 MUX
Power
Author: Logan Lee
i281e Program Counter
i281e CPU
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281 CPU design by:
Kyung-Tae J. K
Alexander Stoytch
Project Supervisor
Alexander Stoytch
Power OUT
Control Path Signals
REG Input / C18 Out
Instruction High Bits
Instruction Low Bits
REG Port 0 / ALU Input A
Program Counter
REG Port 1 / WRTBACK
DMEM Address / C15 Out
ALU Input B / C11 Out
DMEM Input / C16 Out
ALU Output
DMEM Output
ALU Flags
REG Input / C18 Out
Switches High Bits
Switches Low Bits

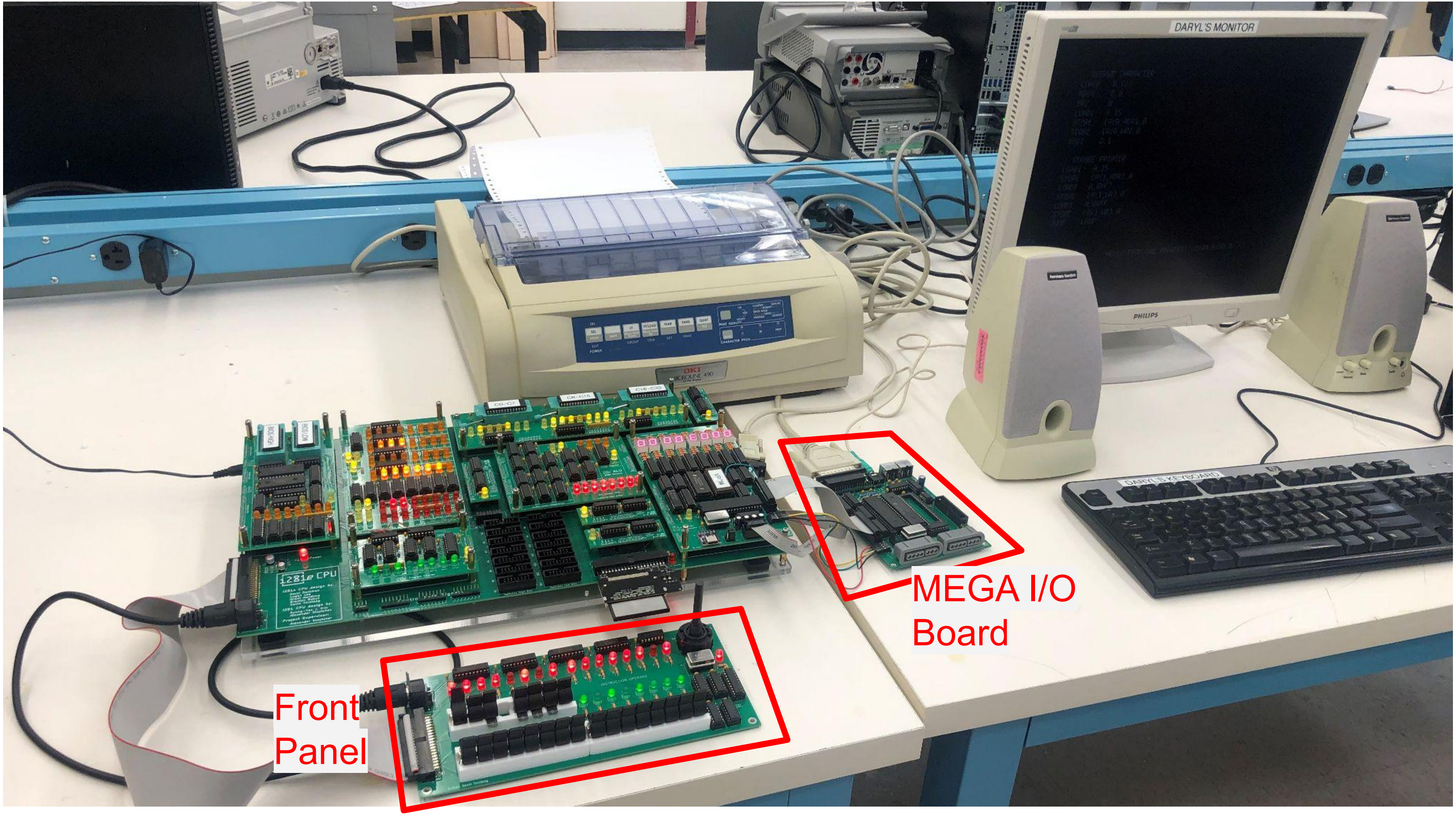


ALU

BIOS HIGH
BIOS LOW
Register A
Register B
Register C
Register D
Port 0
Port 1
C0–C7
C8–C15
C16–C23
C11 MUX
i281e ALU
(NOR Version)
Flags Register
Flags Calculator
Output MUXs
ALU Output LEDs
8-Bit Adder/Subtractor
8-Bit Shifter and NOR
Author: Braxton Rokos
Author: Gavin Tersteeg
Author: Daryl Damman
C15 MUX
C16 MUX
VROM
Data Memory
i281e CPU
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281 CPU design by:
Kyung-Tae J. K
Alexander Stoytch
Project Supervisor
Alexander Stoytch
Program Counter
Control Path Signals
Instruction High Bits
Instruction Low Bits
RFG Port 0 / ALU Input A
Program Counter
REG Port 1 / WRTBACK
DMEM Address / C15 Out
ALU Input B / C11 Out
DMEM Input / C16 Out
ALU Output
DMEM Output
ALU Flags
REG Input / C18 Out
Switches High Bits
Switches Low Bits
1.8432 MHz


REG
CTRL
MUX
CMEM
ALU
DMEM
MUX
MUX
MUX
PC
BIOS HIGH
BIOS LOW
C0–C7
C8–C15
C16–C23
Register A
Register B
Register C
Register D
Port 0
Port 1
VROM
C15 MUX
C16 MUX
Control Path Signals
i281e CPU
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281 CPU design by:
Kyung-Tae J. K
Alexander Stoytc
Project Supervisor
Alexander Stoytch
i281e Program Counter
i281e ALU (NOR Version)
Data Memory


DARYL'S MONITOR
PHILIPS
OKI
MICROLINE 490
C0-C7
C8-C15
C16-C23
ALU
128le CPU
DARYL'S KEYBOARD


DARYL'S MONITOR
PHILIPS
DARYL'S KEYBOARD
OKI
MICROLINE 490
128le CPU
MEGA I/O Board
Front Panel

Technical Details

Gavin Tersteeg
Software Engineering

Loading User Programs

Original i281 CPU

- “Magically” loads programs/BIOS into memory
- No boot sequence/procedures
- User program loading done by “external” mechanisms

New i281e CPU

- External program load mechanism too complex to physically implement
- System must be able to load its own programs

Expanded Memory

Original i281 CPU

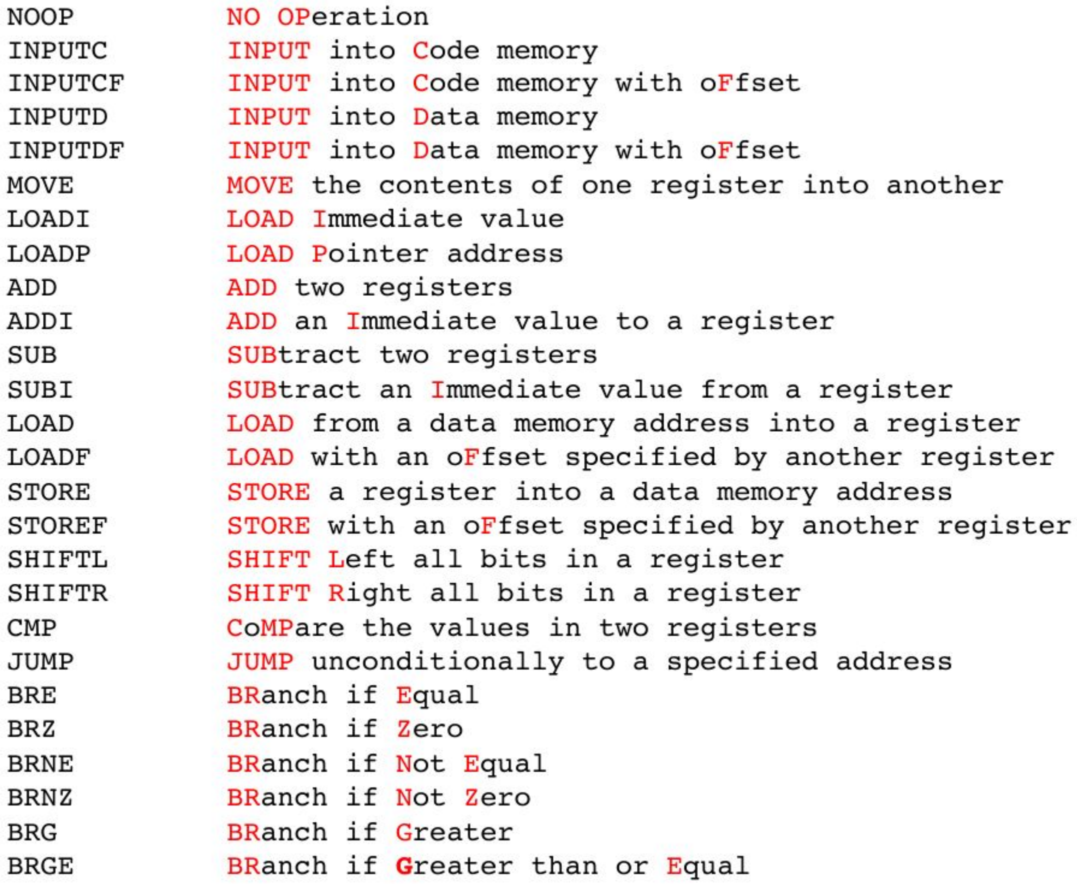
32 words	- Code ROM (BIOS)
32 words	- Code RAM
16 bytes	- Data Memory

No Mass Storage / Hard Disk

Expanded Memory

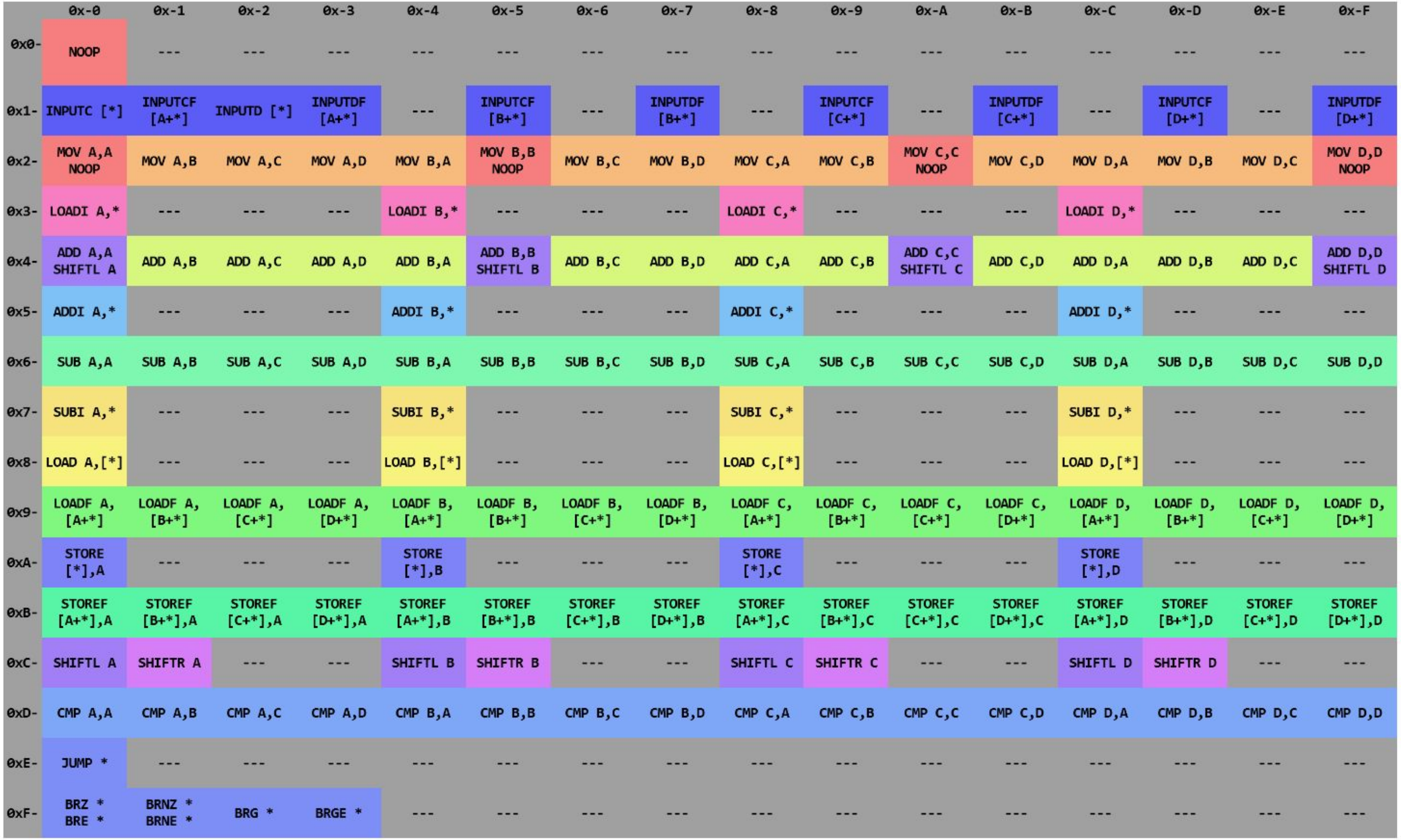
Original i281 CPU			New i281e CPU	
32 words	- Code ROM (BIOS)	=>	128 words	- Code ROM (BIOS)
32 words	- Code RAM	=>	32 KW	- Code RAM
16 bytes	- Data Memory	=>	32 KB	- Data Memory
No Mass Storage / Hard Disk		=>	32 MB	- Compact Flash Memory “Hard Disk”

Original Opcodes



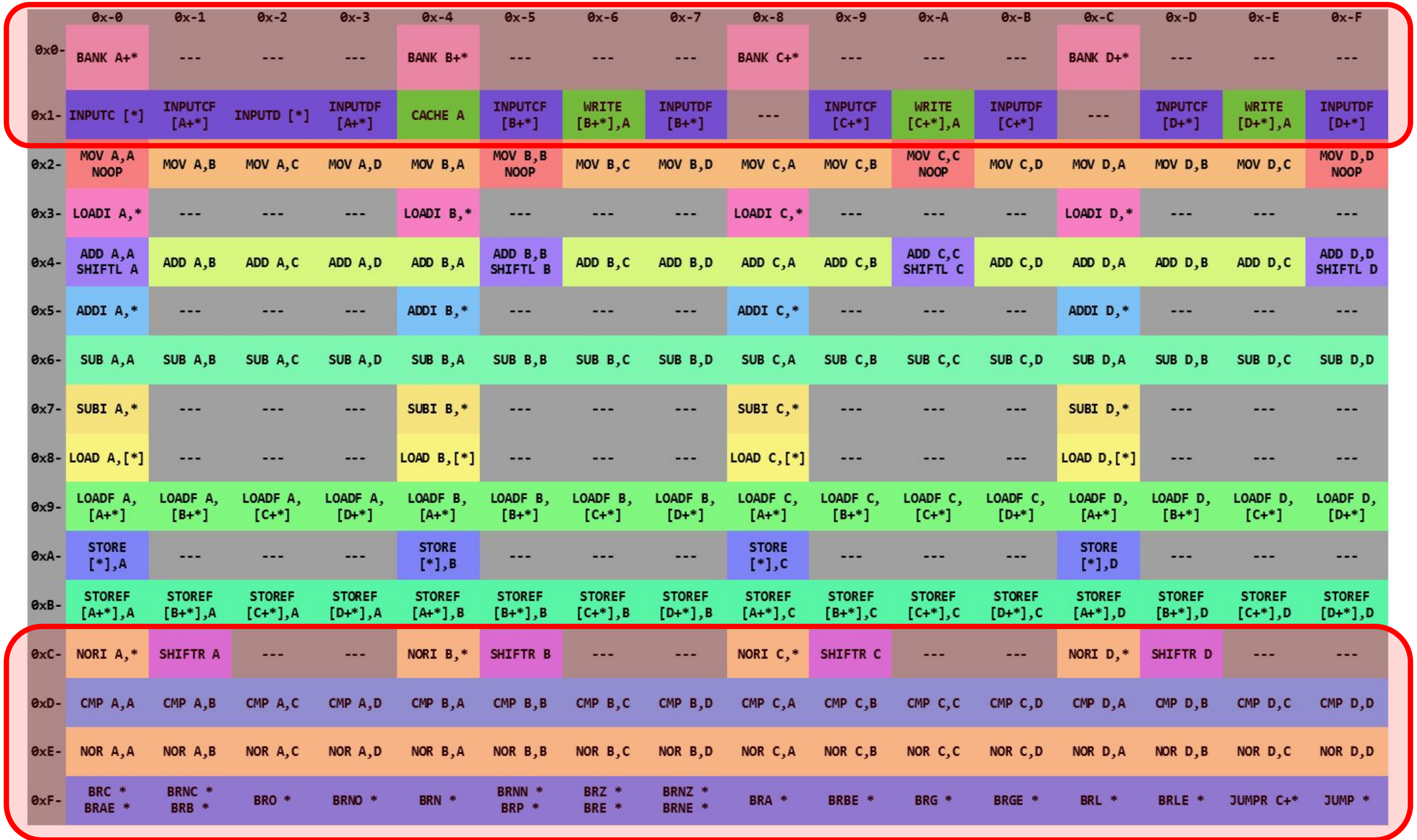
NOOP	NO OPeration
INPUTC	INPUT into Code memory
INPUTCF	INPUT into Code memory with oFfset
INPUTD	INPUT into Data memory
INPUTDF	INPUT into Data memory with oFfset
MOVE	MOVE the contents of one register into another
LOADI	LOAD Immediate value
LOADP	LOAD Pointer address
ADD	ADD two registers
ADDI	ADD an Immediate value to a register
SUB	SUBtract two registers
SUBI	SUBtract an Immediate value from a register
LOAD	LOAD from a data memory address into a register
LOADF	LOAD with an oFfset specified by another register
STORE	STORE a register into a data memory address
STOREF	STORE with an oFfset specified by another register
SHIFTL	SHIFT Left all bits in a register
SHIFTR	SHIFT Right all bits in a register
CMP	CoMPare the values in two registers
JUMP	JUMP unconditionally to a specified address
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal

Original i281 Instruction Set Table



	0x-0	0x-1	0x-2	0x-3	0x-4	0x-5	0x-6	0x-7	0x-8	0x-9	0x-A	0x-B	0x-C	0x-D	0x-E	0x-F
0x0-	NOOP	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
0x1-	INPUTC [*]	INPUTCF [A+*]	INPUTD [*]	INPUTDF [A+*]	---	INPUTCF [B+*]	---	INPUTDF [B+*]	---	INPUTCF [C+*]	---	INPUTDF [C+*]	---	INPUTCF [D+*]	---	INPUTDF [D+*]
0x2-	MOV A,A NOOP	MOV A,B	MOV A,C	MOV A,D	MOV B,A	MOV B,B NOOP	MOV B,C	MOV B,D	MOV C,A	MOV C,B	MOV C,C NOOP	MOV C,D	MOV D,A	MOV D,B	MOV D,C	MOV D,D NOOP
0x3-	LOADI A,*	---	---	---	LOADI B,*	---	---	---	LOADI C,*	---	---	---	LOADI D,*	---	---	---
0x4-	ADD A,A SHIFTL A	ADD A,B	ADD A,C	ADD A,D	ADD B,A	ADD B,B SHIFTL B	ADD B,C	ADD B,D	ADD C,A	ADD C,B	ADD C,C SHIFTL C	ADD C,D	ADD D,A	ADD D,B	ADD D,C	ADD D,D SHIFTL D
0x5-	ADDI A,*	---	---	---	ADDI B,*	---	---	---	ADDI C,*	---	---	---	ADDI D,*	---	---	---
0x6-	SUB A,A	SUB A,B	SUB A,C	SUB A,D	SUB B,A	SUB B,B	SUB B,C	SUB B,D	SUB C,A	SUB C,B	SUB C,C	SUB C,D	SUB D,A	SUB D,B	SUB D,C	SUB D,D
0x7-	SUBI A,*	---	---	---	SUBI B,*	---	---	---	SUBI C,*	---	---	---	SUBI D,*	---	---	---
0x8-	LOAD A,[*]	---	---	---	LOAD B,[*]	---	---	---	LOAD C,[*]	---	---	---	LOAD D,[*]	---	---	---
0x9-	LOADF A, [A+*]	LOADF A, [B+*]	LOADF A, [C+*]	LOADF A, [D+*]	LOADF B, [A+*]	LOADF B, [B+*]	LOADF B, [C+*]	LOADF B, [D+*]	LOADF C, [A+*]	LOADF C, [B+*]	LOADF C, [C+*]	LOADF C, [D+*]	LOADF D, [A+*]	LOADF D, [B+*]	LOADF D, [C+*]	LOADF D, [D+*]
0xA-	STORE [*],A	---	---	---	STORE [*],B	---	---	---	STORE [*],C	---	---	---	STORE [*],D	---	---	---
0xB-	STOREF [A+*],A	STOREF [B+*],A	STOREF [C+*],A	STOREF [D+*],A	STOREF [A+*],B	STOREF [B+*],B	STOREF [C+*],B	STOREF [D+*],B	STOREF [A+*],C	STOREF [B+*],C	STOREF [C+*],C	STOREF [D+*],C	STOREF [A+*],D	STOREF [B+*],D	STOREF [C+*],D	STOREF [D+*],D
0xC-	SHIFTL A	SHIFTR A	---	---	SHIFTL B	SHIFTR B	---	---	SHIFTL C	SHIFTR C	---	---	SHIFTL D	SHIFTR D	---	---
0xD-	CMP A,A	CMP A,B	CMP A,C	CMP A,D	CMP B,A	CMP B,B	CMP B,C	CMP B,D	CMP C,A	CMP C,B	CMP C,C	CMP C,D	CMP D,A	CMP D,B	CMP D,C	CMP D,D
0xE-	JUMP *	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---
0xF-	BRZ * BRE *	BRNZ * BRNE *	BRG *	BRGE *	---	---	---	---	---	---	---	---	---	---	---	---

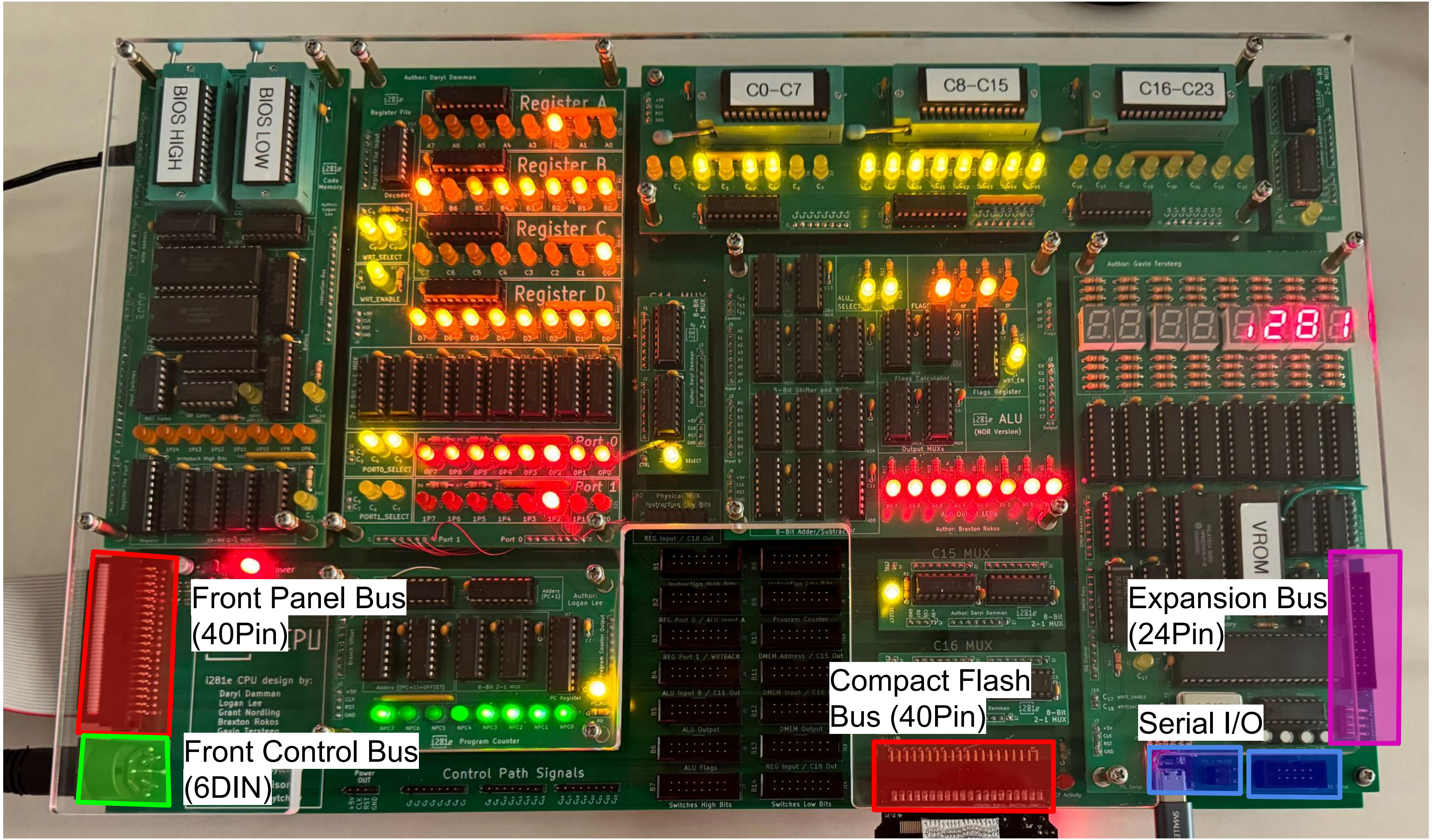
Expanded i281e Instruction Set Table



	0x-0	0x-1	0x-2	0x-3	0x-4	0x-5	0x-6	0x-7	0x-8	0x-9	0x-A	0x-B	0x-C	0x-D	0x-E	0x-F
0x0-	BANK A+*	---	---	---	BANK B+*	---	---	---	BANK C+*	---	---	---	BANK D+*	---	---	---
0x1-	INPUTC [*]	INPUTCF [A+*]	INPUTD [*]	INPUTDF [A+*]	CACHE A	INPUTCF [B+*]	WRITE [B+*],A	INPUTDF [B+*]	---	INPUTCF [C+*]	WRITE [C+*],A	INPUTDF [C+*]	---	INPUTCF [D+*]	WRITE [D+*],A	INPUTDF [D+*]
0x2-	MOV A,A NOOP	MOV A,B	MOV A,C	MOV A,D	MOV B,A	MOV B,B NOOP	MOV B,C	MOV B,D	MOV C,A	MOV C,B	MOV C,C NOOP	MOV C,D	MOV D,A	MOV D,B	MOV D,C	MOV D,D NOOP
0x3-	LOADI A,*	---	---	---	LOADI B,*	---	---	---	LOADI C,*	---	---	---	LOADI D,*	---	---	---
0x4-	ADD A,A SHIFTL A	ADD A,B	ADD A,C	ADD A,D	ADD B,A	ADD B,B SHIFTL B	ADD B,C	ADD B,D	ADD C,A	ADD C,B	ADD C,C SHIFTL C	ADD C,D	ADD D,A	ADD D,B	ADD D,C	ADD D,D SHIFTL D
0x5-	ADDI A,*	---	---	---	ADDI B,*	---	---	---	ADDI C,*	---	---	---	ADDI D,*	---	---	---
0x6-	SUB A,A	SUB A,B	SUB A,C	SUB A,D	SUB B,A	SUB B,B	SUB B,C	SUB B,D	SUB C,A	SUB C,B	SUB C,C	SUB C,D	SUB D,A	SUB D,B	SUB D,C	SUB D,D
0x7-	SUBI A,*	---	---	---	SUBI B,*	---	---	---	SUBI C,*	---	---	---	SUBI D,*	---	---	---
0x8-	LOAD A,[*]	---	---	---	LOAD B,[*]	---	---	---	LOAD C,[*]	---	---	---	LOAD D,[*]	---	---	---
0x9-	LOADF A,[A+*]	LOADF A,[B+*]	LOADF A,[C+*]	LOADF A,[D+*]	LOADF B,[A+*]	LOADF B,[B+*]	LOADF B,[C+*]	LOADF B,[D+*]	LOADF C,[A+*]	LOADF C,[B+*]	LOADF C,[C+*]	LOADF C,[D+*]	LOADF D,[A+*]	LOADF D,[B+*]	LOADF D,[C+*]	LOADF D,[D+*]
0xA-	STORE [*],A	---	---	---	STORE [*],B	---	---	---	STORE [*],C	---	---	---	STORE [*],D	---	---	---
0xB-	STOREF [A+*],A	STOREF [B+*],A	STOREF [C+*],A	STOREF [D+*],A	STOREF [A+*],B	STOREF [B+*],B	STOREF [C+*],B	STOREF [D+*],B	STOREF [A+*],C	STOREF [B+*],C	STOREF [C+*],C	STOREF [D+*],C	STOREF [A+*],D	STOREF [B+*],D	STOREF [C+*],D	STOREF [D+*],D
0xC-	NORI A,*	SHIFTR A	---	---	NORI B,*	SHIFTR B	---	---	NORI C,*	SHIFTR C	---	---	NORI D,*	SHIFTR D	---	---
0xD-	CMP A,A	CMP A,B	CMP A,C	CMP A,D	CMP B,A	CMP B,B	CMP B,C	CMP B,D	CMP C,A	CMP C,B	CMP C,C	CMP C,D	CMP D,A	CMP D,B	CMP D,C	CMP D,D
0xE-	NOR A,A	NOR A,B	NOR A,C	NOR A,D	NOR B,A	NOR B,B	NOR B,C	NOR B,D	NOR C,A	NOR C,B	NOR C,C	NOR C,D	NOR D,A	NOR D,B	NOR D,C	NOR D,D
0xF-	BRC * BRAE *	BRNC * BRB *	BRO *	BRNO *	BRN *	BRNN * BRP *	BRZ * BRE *	BRNZ * BRNE *	BRA *	BRBE *	BRG *	BRGE *	BRL *	BRLE *	JUMPR C+*	JUMP *

External Connectors

- Front Panel / Control Bus
    - 40-Pin IDC and 6-Pin DIN for providing clock and switch controls
- Serial I/O
    - TTL-Logic or RS-232 serial connection to on-board UART (for DOS/281 or MONITOR)
- Compact Flash
    - “Hard Disk” of the system, using Compact Flash bus protocols
- Expansion Bus
    - Smaller version of S-100 Bus (IEEE-696) connection for managing external peripherals


BIOS HIGH
BIOS LOW
Register A
Register B
Register C
Register D
C0–C7
C8–C15
C16–C23
Port 0
Port 1
i281e ALU
(NOR Version)
VROM
C15 MUX
C16 MUX
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281e Program Counter
Control Path Signals
Front Panel Bus
(40Pin)
Front Control Bus
(6DIN)
Compact Flash
Bus (40Pin)
Expansion Bus
(24Pin)
Serial I/O

What’s left?

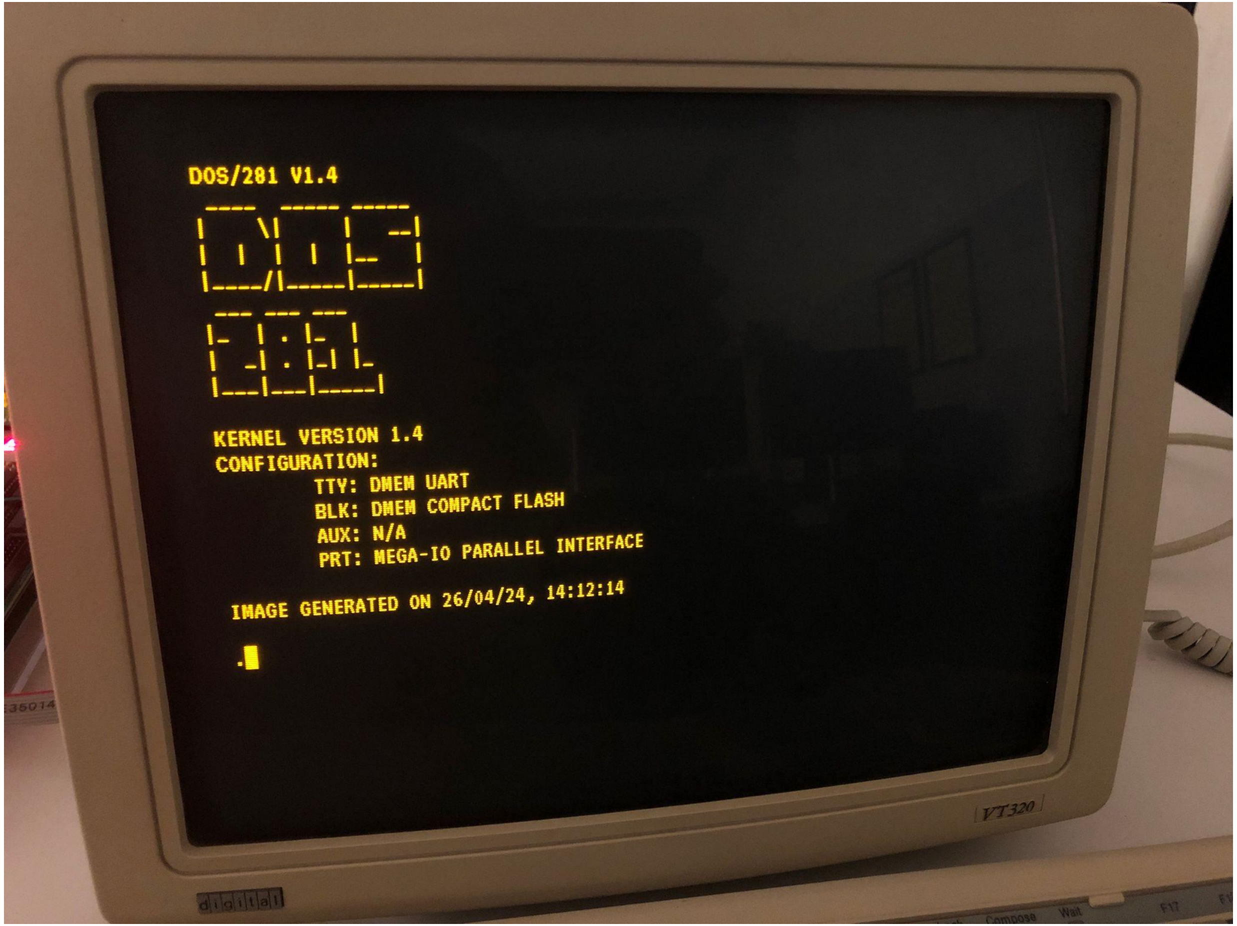
- Expanded memory allows for boot software to execute
- i281e CPU can now directly access mass storage to read programs
- New opcodes to take data and automatically write it to memory

What's left?

- Expanded memory allows for boot software to execute
- i281e CPU can now directly access mass storage to read programs
- New opcodes to take data and automatically write it to memory

All that remains now is the software.

DOS/281



```
DOS/281 V1.4
 ____  _____ _____
|    \|     |   __|
|  |  |  |  |__   |
|____/|_____|_____|
 ___ ___ ___
|_  | . |_  |
|  _| . |_| |_
|___|___|_____|

KERNEL VERSION 1.4
CONFIGURATION:
        TTY: DMEM UART
        BLK: DMEM COMPACT FLASH
        AUX: N/A
        PRT: MEGA-IO PARALLEL INTERFACE

 IMAGE GENERATED ON 26/04/24, 14:12:14

 .
```

DOS/281 Features

- Loading user programs
- Downloading and saving new user programs
- Managing program and non-program files on the compact flash card
- Allowing editing, assembly, and debugging of user programs locally
- Providing API interfaces for more complex i281 applications

DOS/281 is implemented in ~3,000 lines of assembly code, and occupies 5 KB of memory.

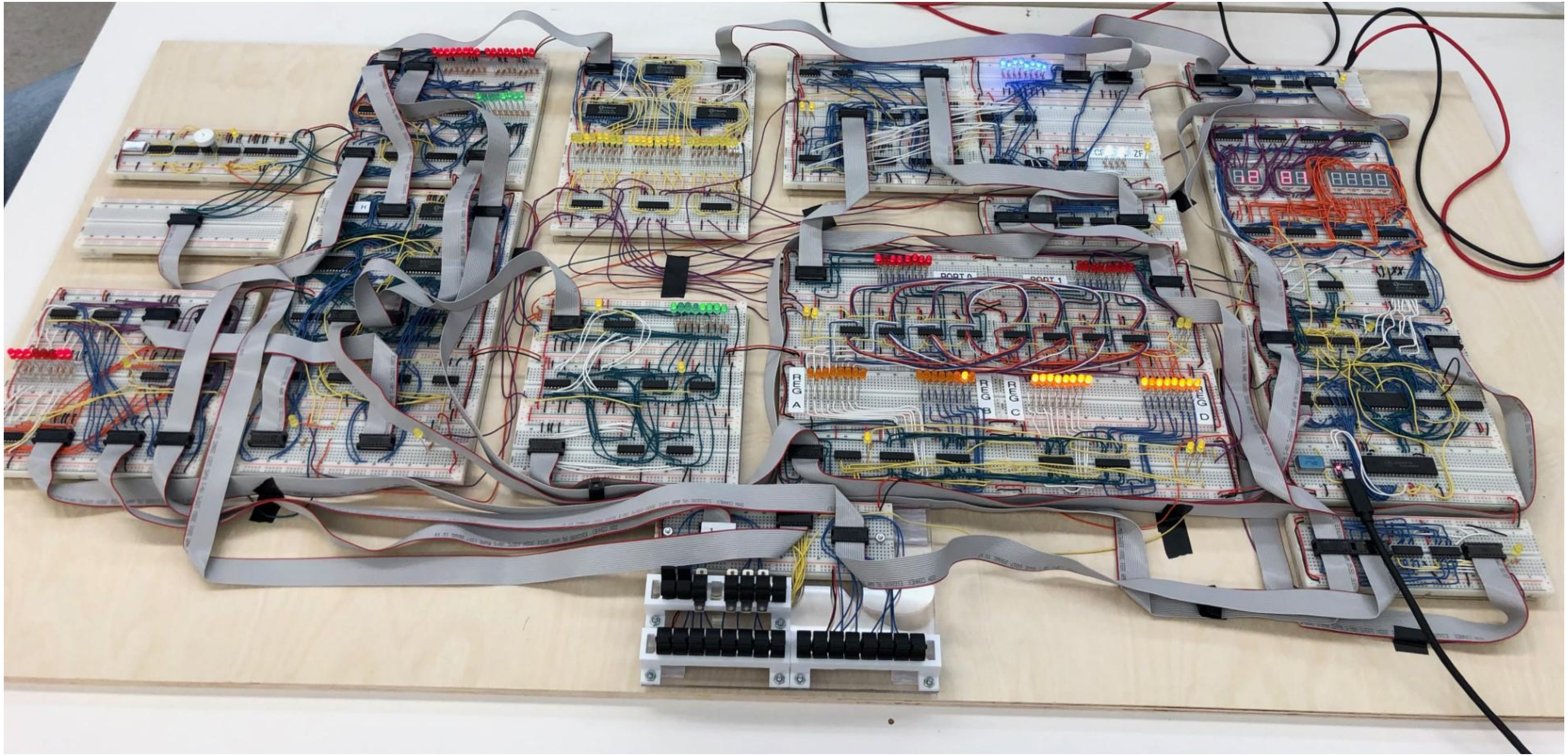
This is about **0.011%** of the size of the Linux kernel.

0000E000
Z81e CPU

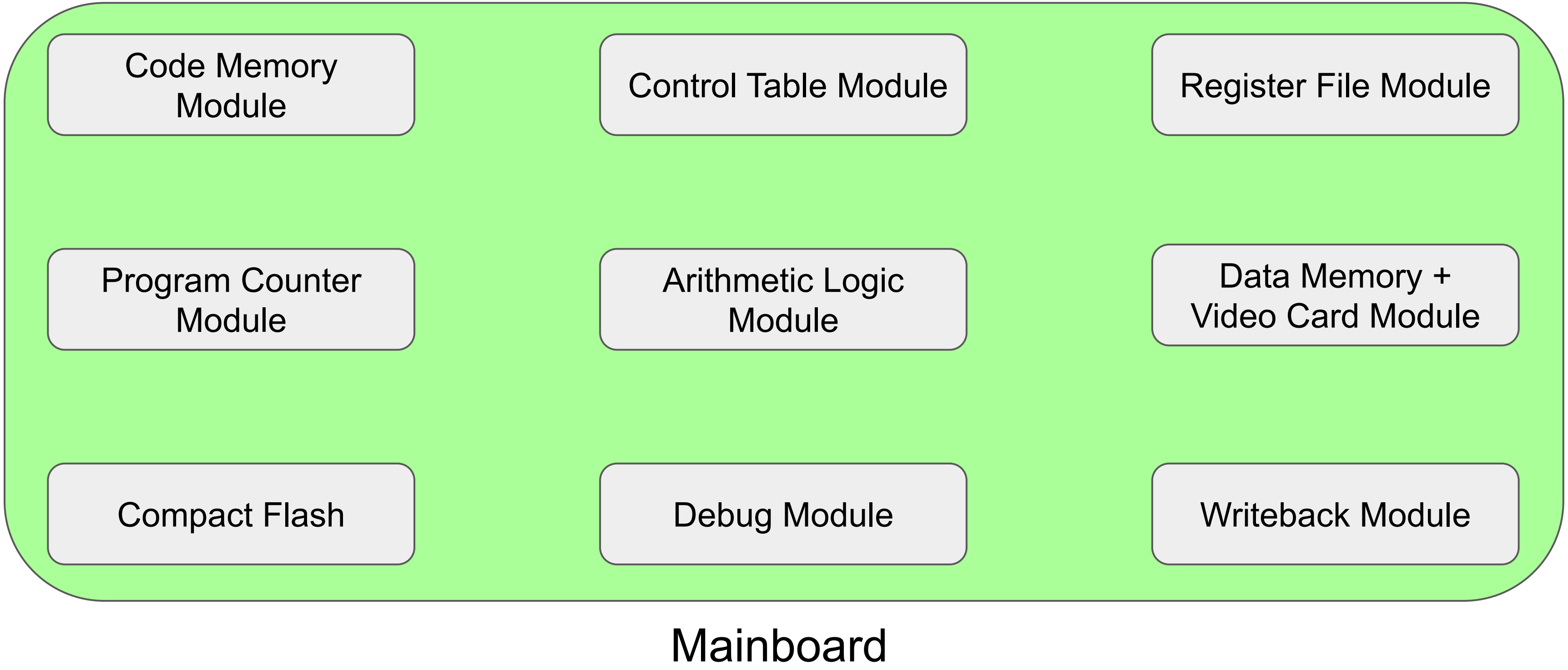
Expandability

Gavin Tersteeg
Software Engineering

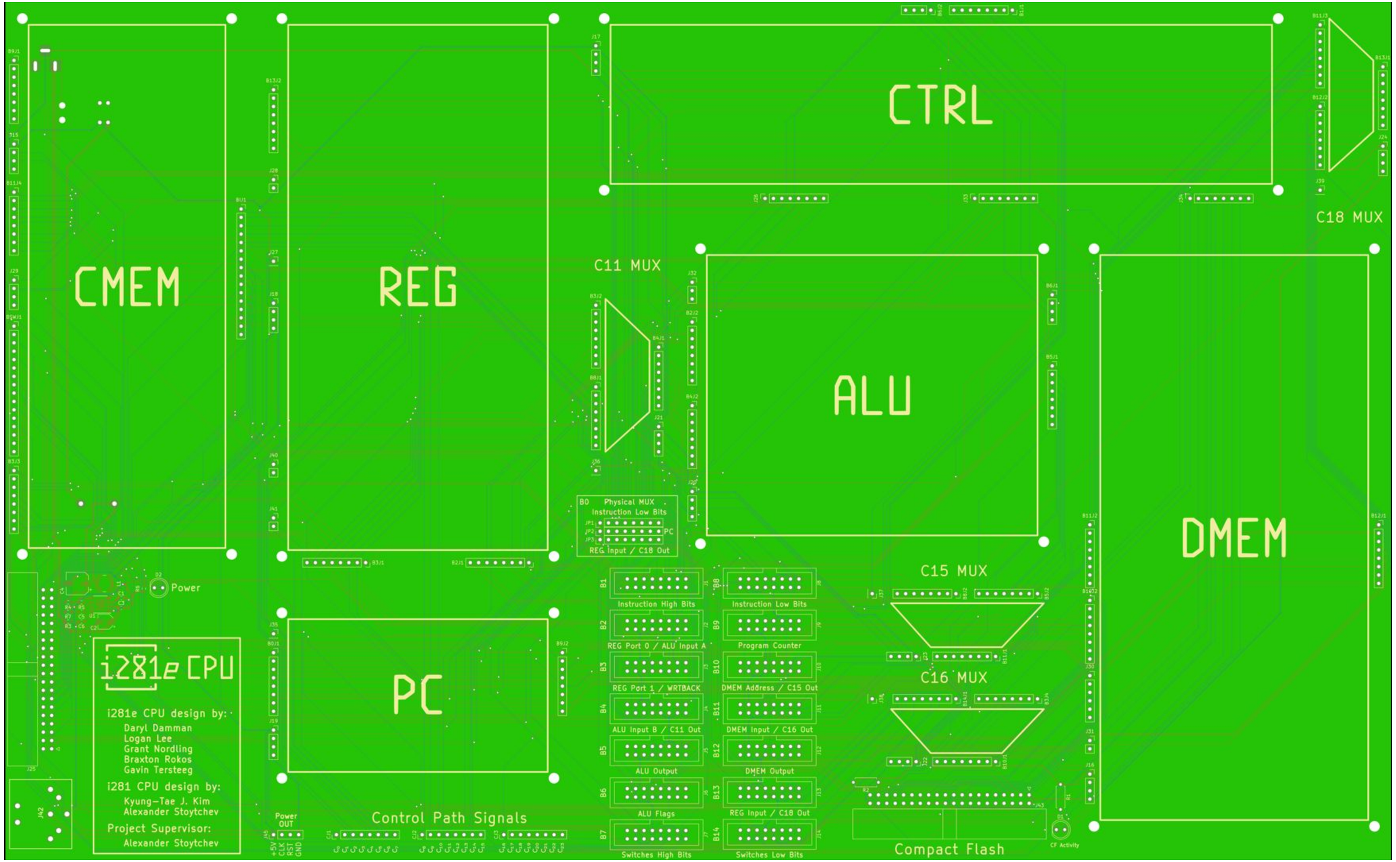
Each Ribbon Cable is an 8-bit Bus



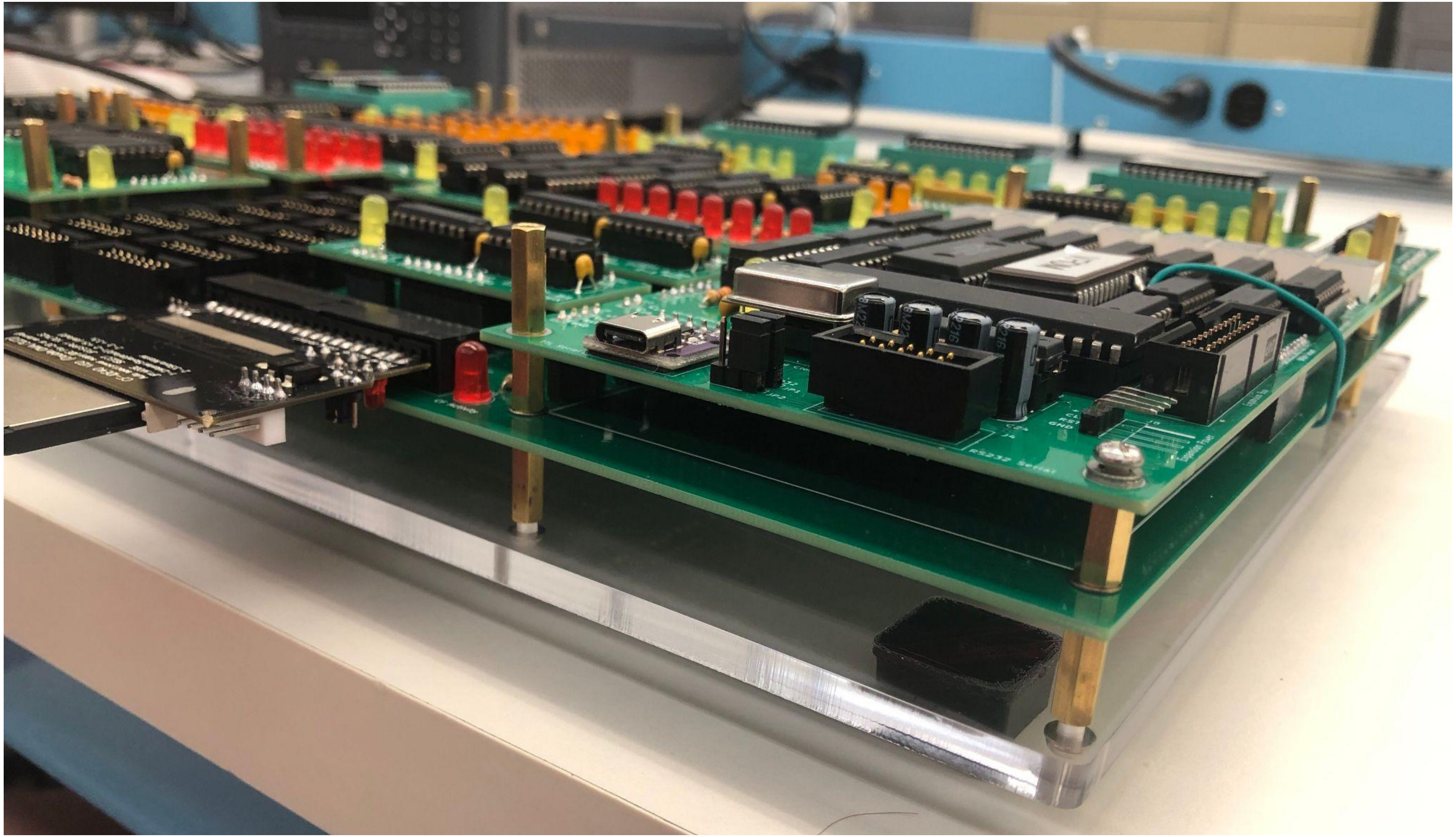
Module Separation

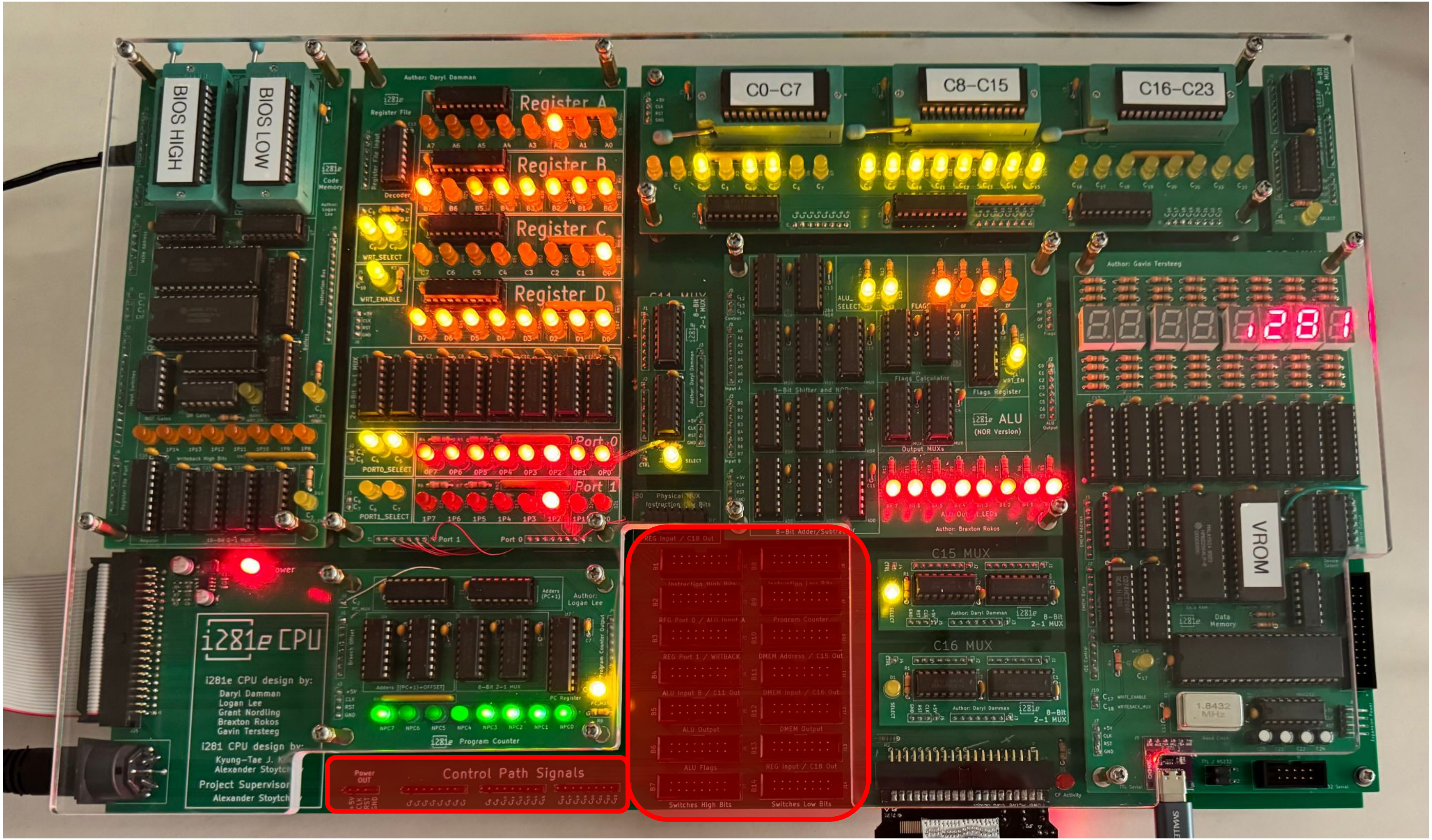


Mainboard PCB Diagram

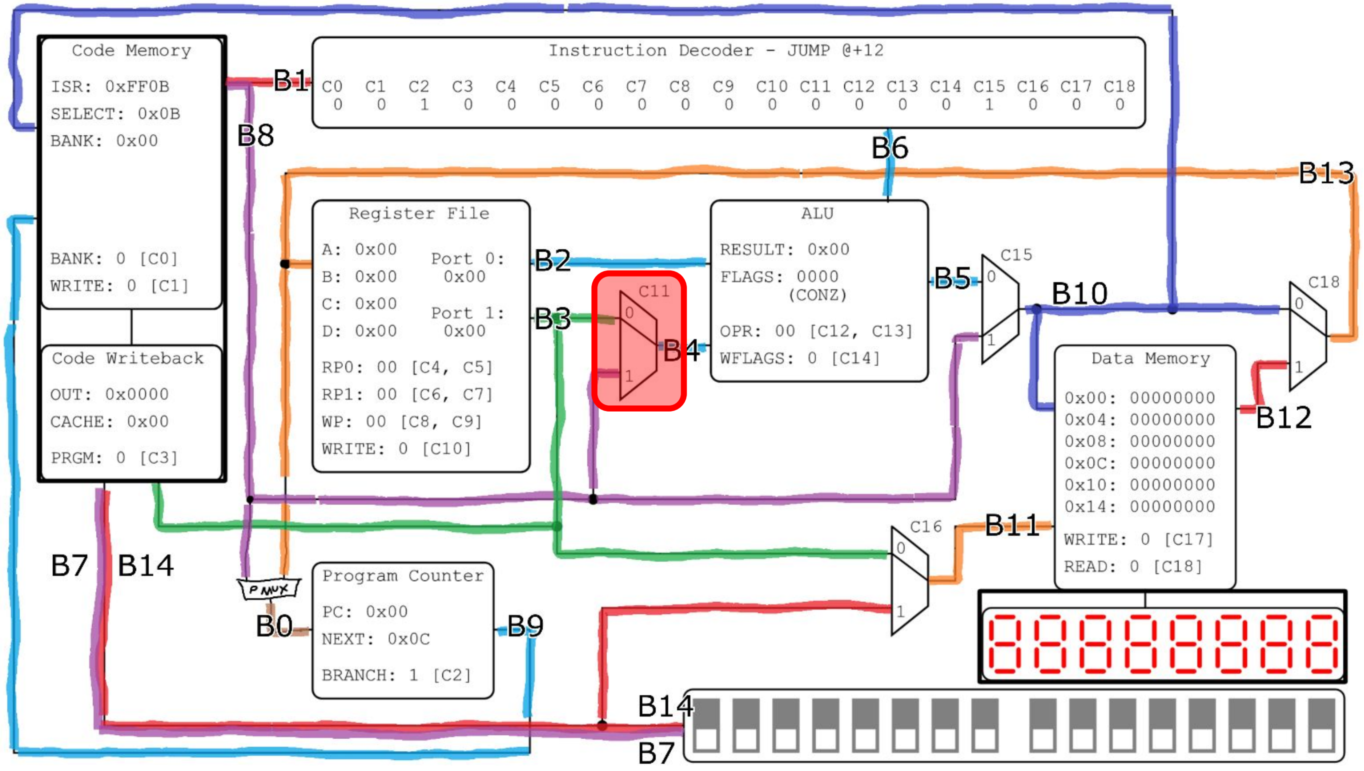


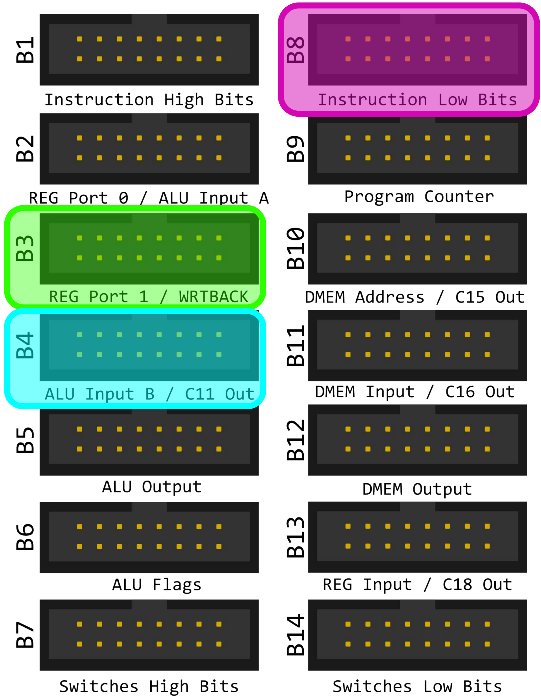
Two Layer Design



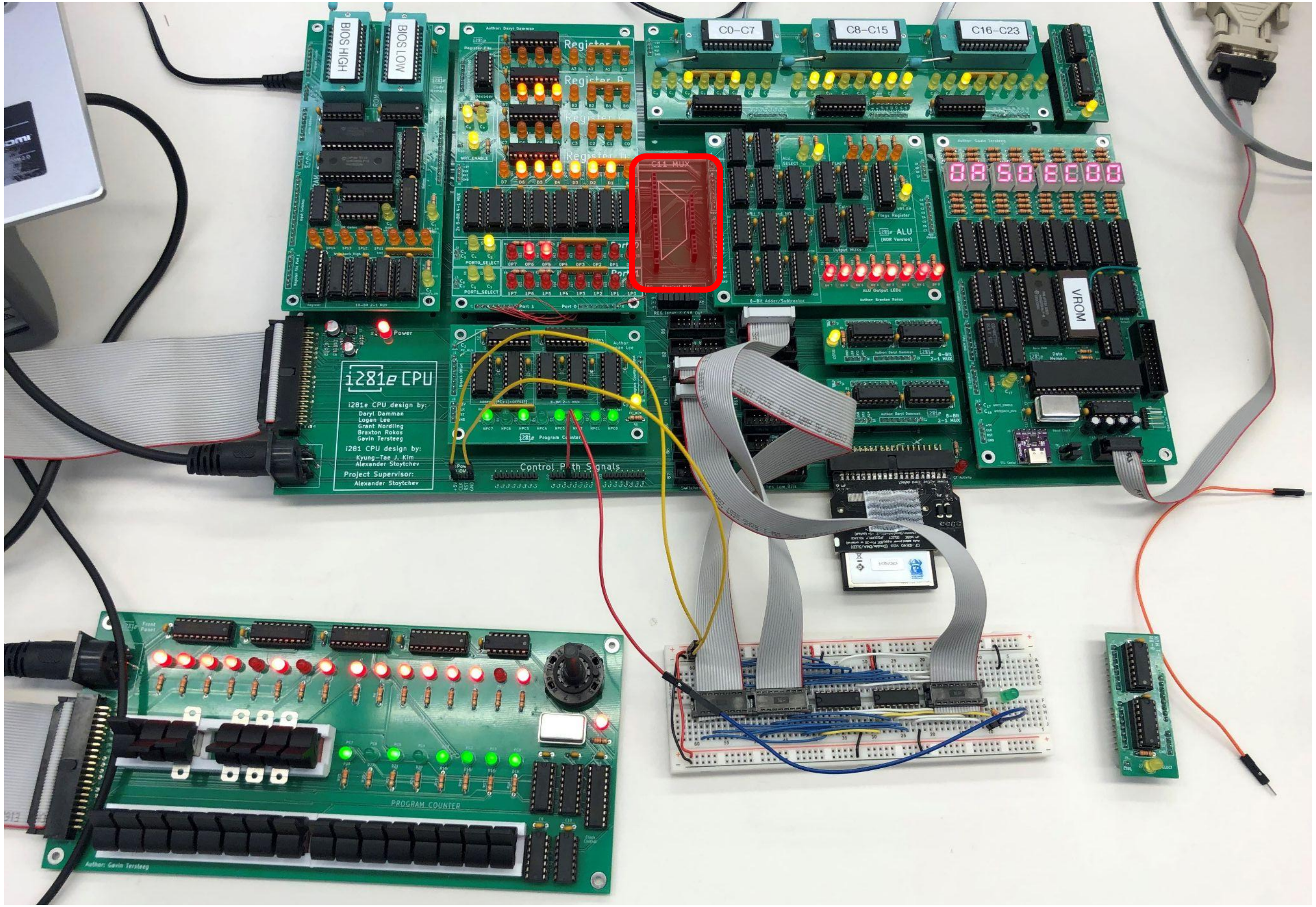

BIOS HIGH
BIOS LOW
Code Memory
Author: Logan Lee
Author: Daryl Damman
i281e Register File
Register A
Register B
Register C
Register D
WRT_SELECT
WRT_ENABLE
Port 0
Port 1
PORT0_SELECT
PORT1_SELECT
C0–C7
C8–C15
C16–C23
Physical MUX Instruction Low Bits
ALU_SELECT
FLAGS
Flags Calculator
Flags Register
8-Bit Shifter and NOR
i281e ALU (NOR Version)
Output MUXs
ALU Output LEDs
Author: Braxton Rokos
8-Bit Adder/Subtractor
Author: Gavin Tersteeg
VROM
Data Memory
1.8432 MHz
C15 MUX
C16 MUX
8-Bit 2-1 MUX
Power
i281e CPU
i281e CPU design by:
Daryl Damman
Logan Lee
Grant Nordling
Braxton Rokos
Gavin Tersteeg
i281 CPU design by:
Kyung-Tae J. K
Alexander Stoytc
Project Supervisor
Alexander Stoytch
Author: Logan Lee
Adders (PC+1)
8-Bit 2-1 MUX
PC Register
i281e Program Counter
NPC7
NPC6
NPC5
NPC4
NPC3
NPC2
NPC1
NPC0
Control Path Signals
Power OUT
REG Input / C18 Out
Instruction High Bits
Instruction Low Bits
REG Port 0 / ALU Input A
Program Counter
REG Port 1 / WRTBACK
DMEM Address / C15 Out
ALU Input B / C11 Out
DMEM Input / C16 Out
ALU Output
DMEM Output
ALU Flags
REG Input / C18 Out
Switches High Bits
Switches Low Bits

Mainboard Bus Connectors

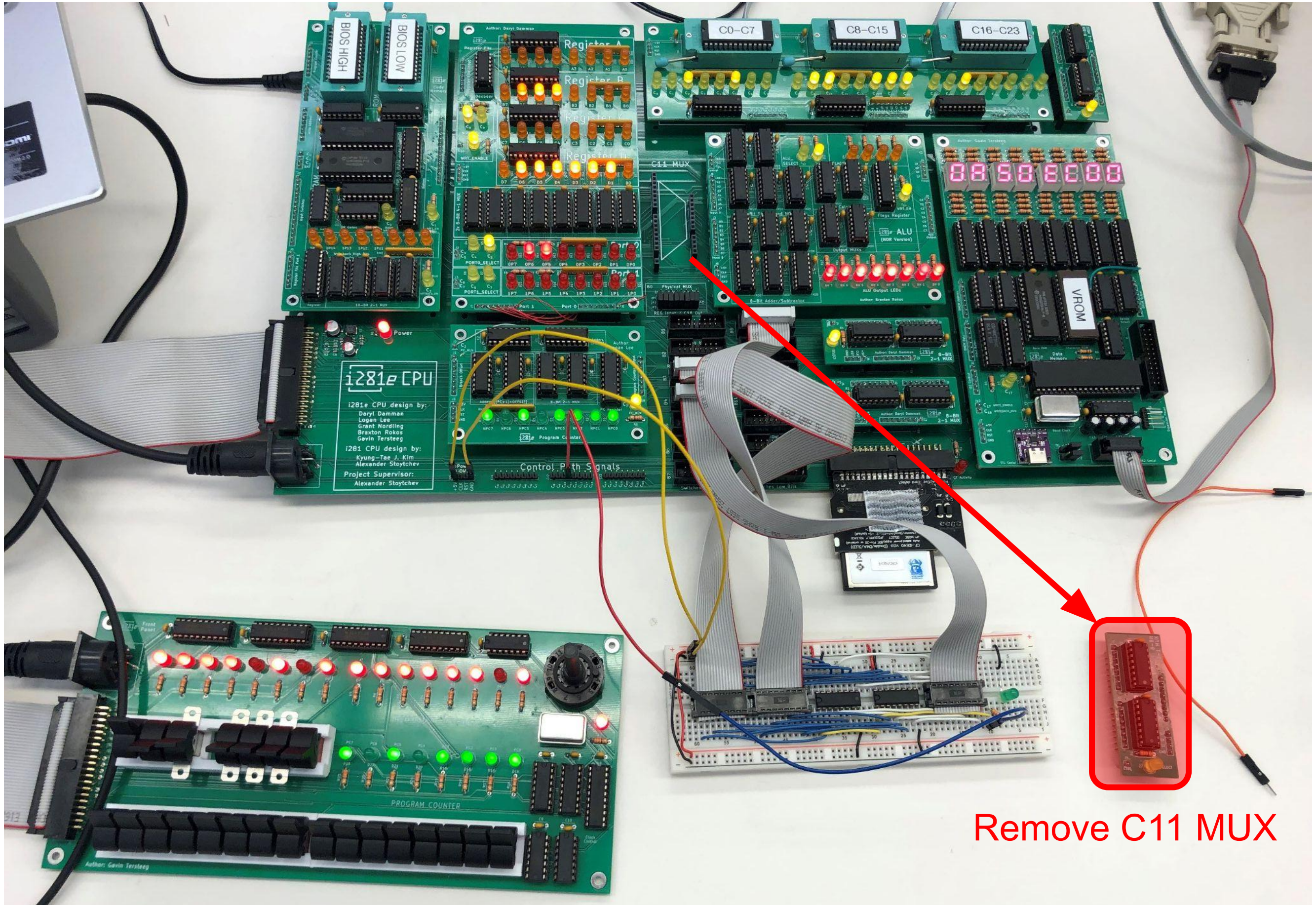




Additional Implementation

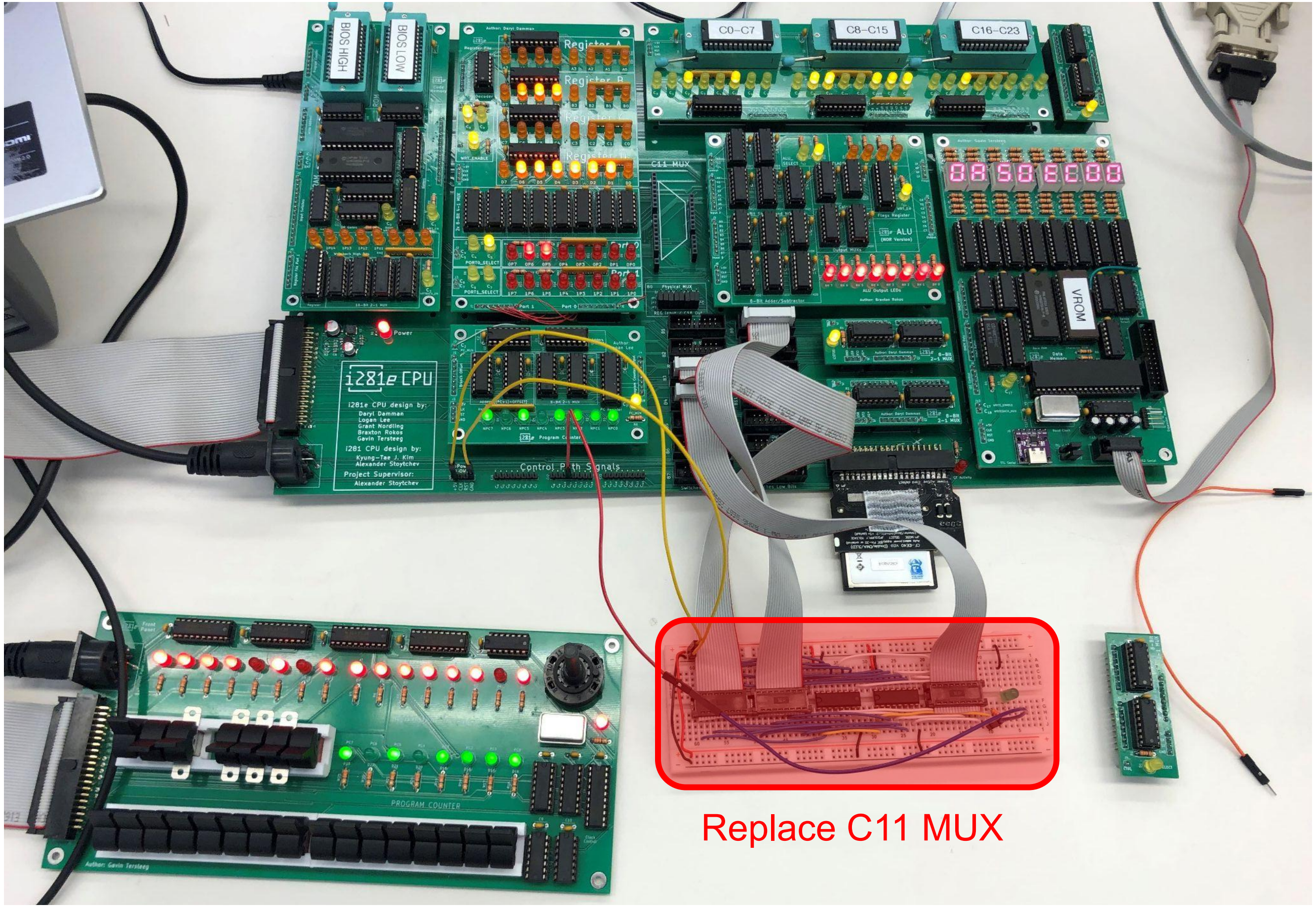


Additional Implementation



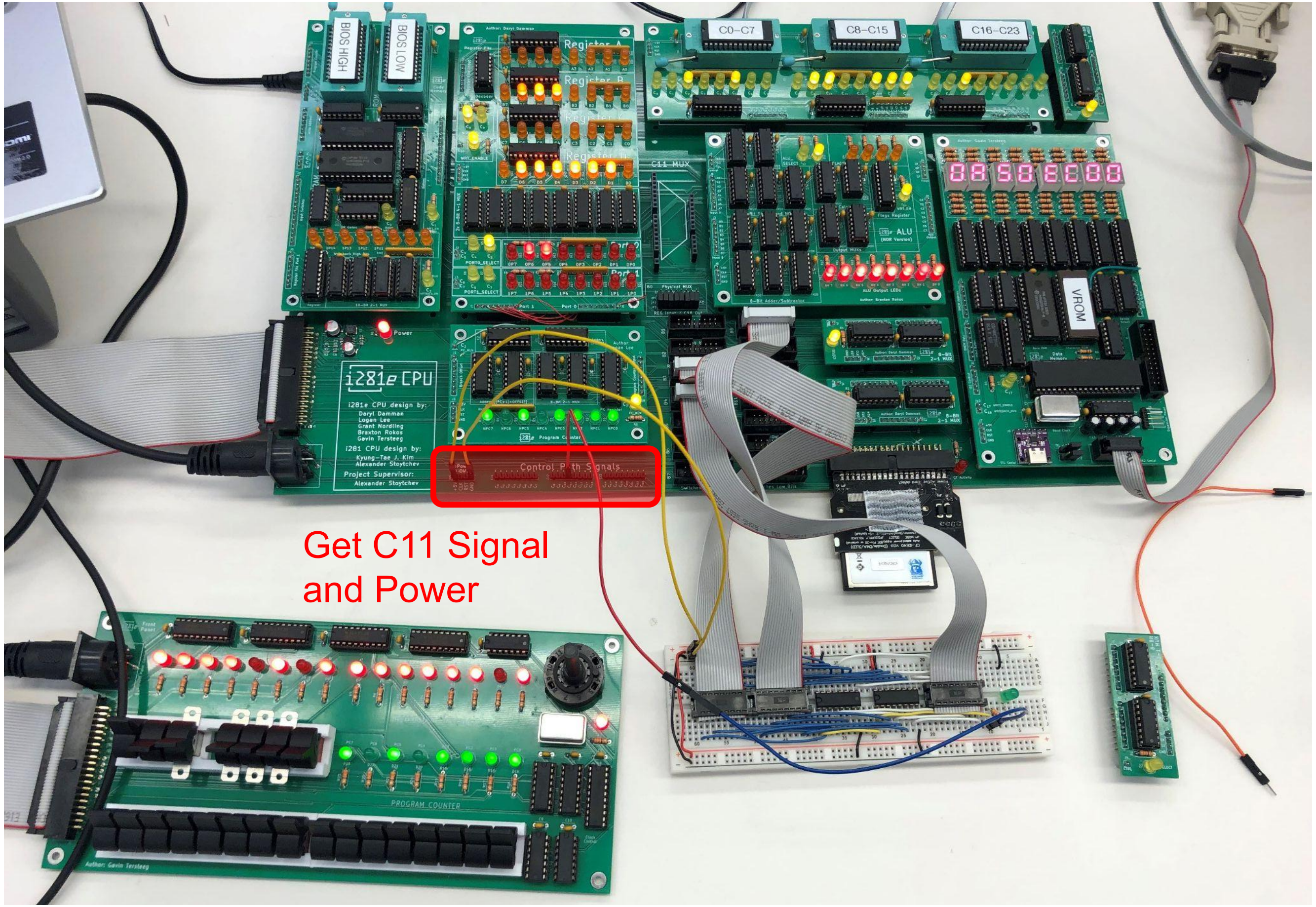
Remove C11 MUX

Additional Implementation



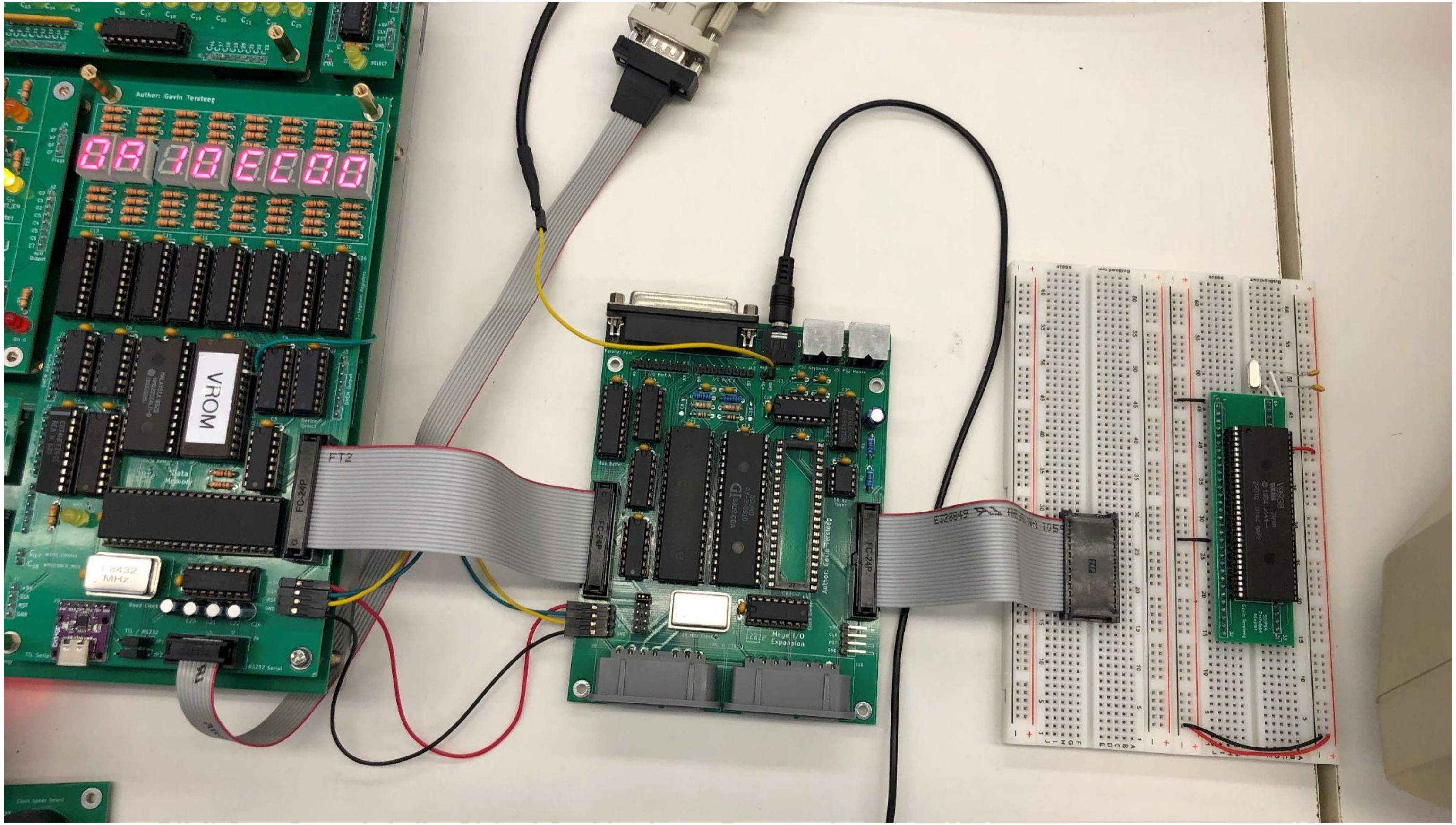
Replace C11 MUX

Additional Implementation



Get C11 Signal and Power

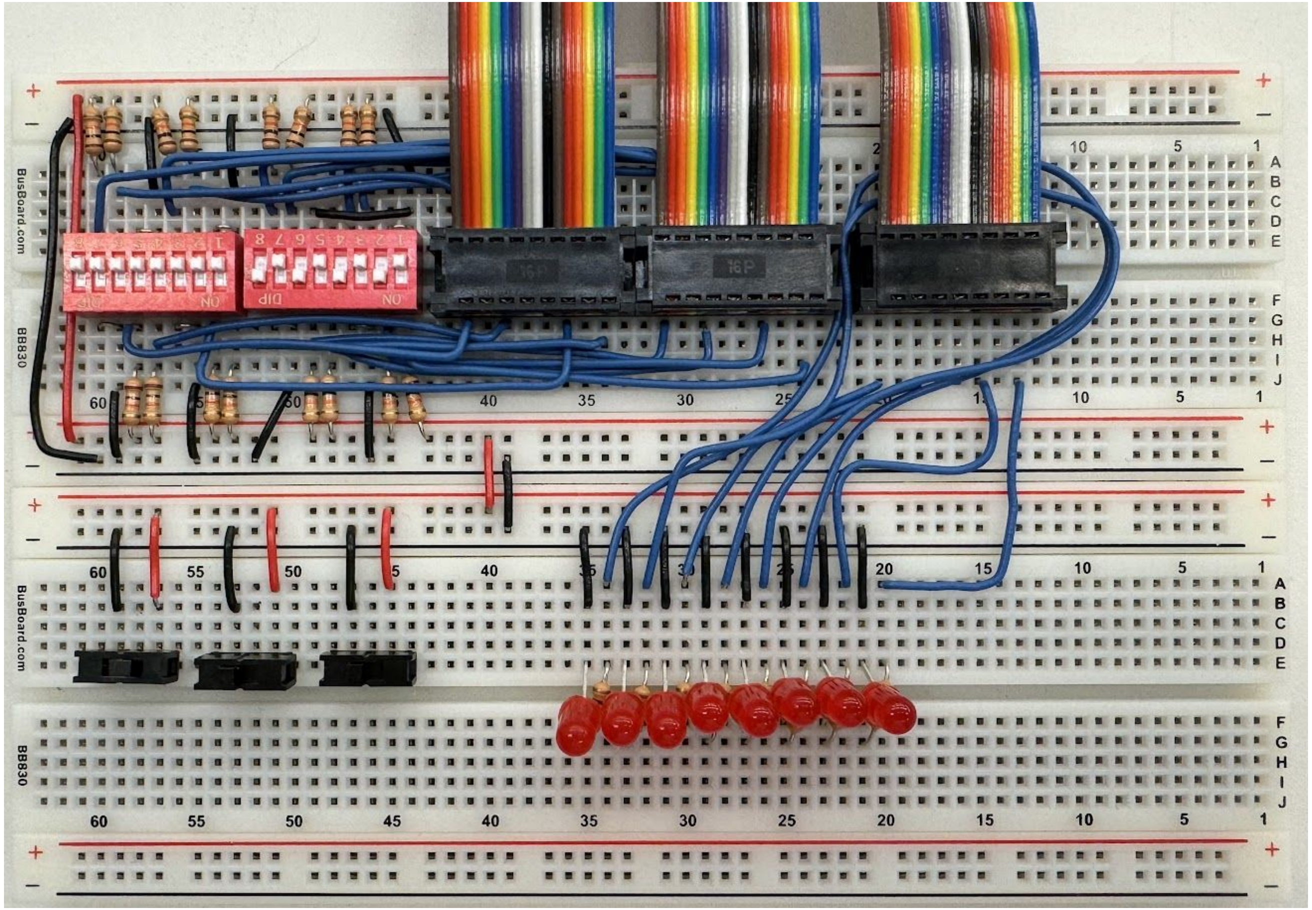
Expansion Capability



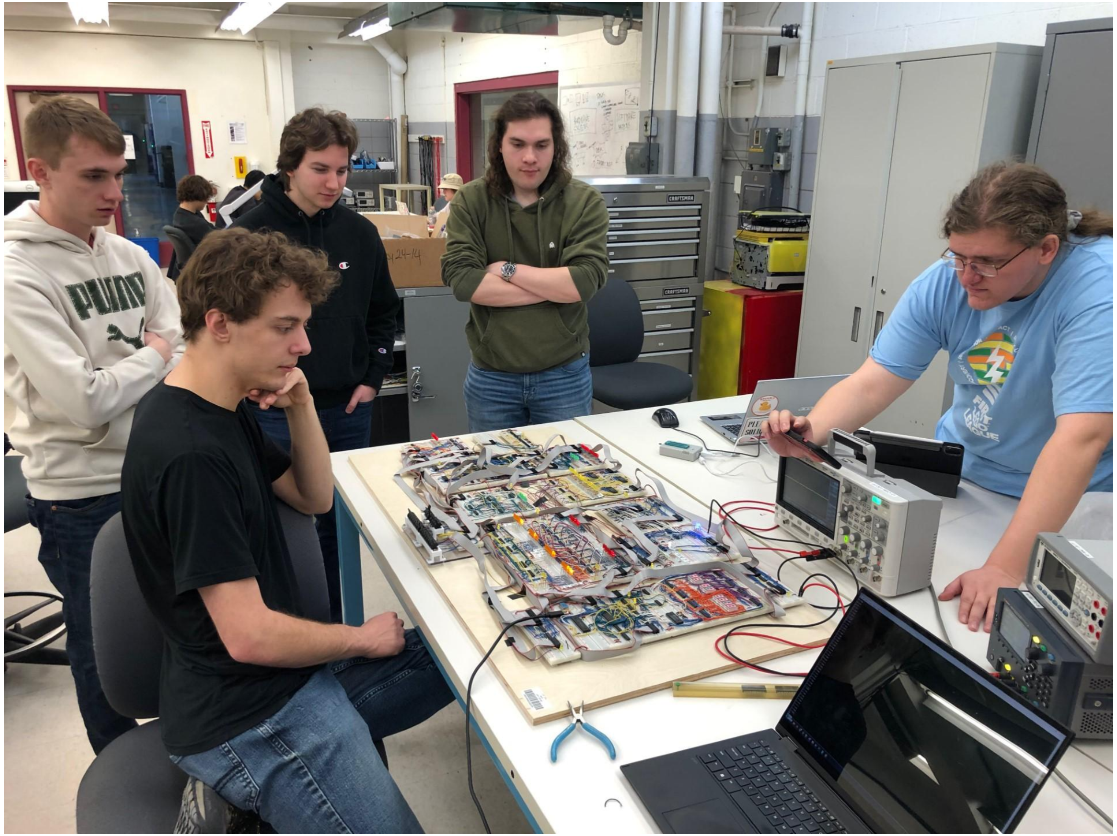
Testing

Braxton Rokos
Electrical Engineering

Breadboard Testing



Breadboard Stability



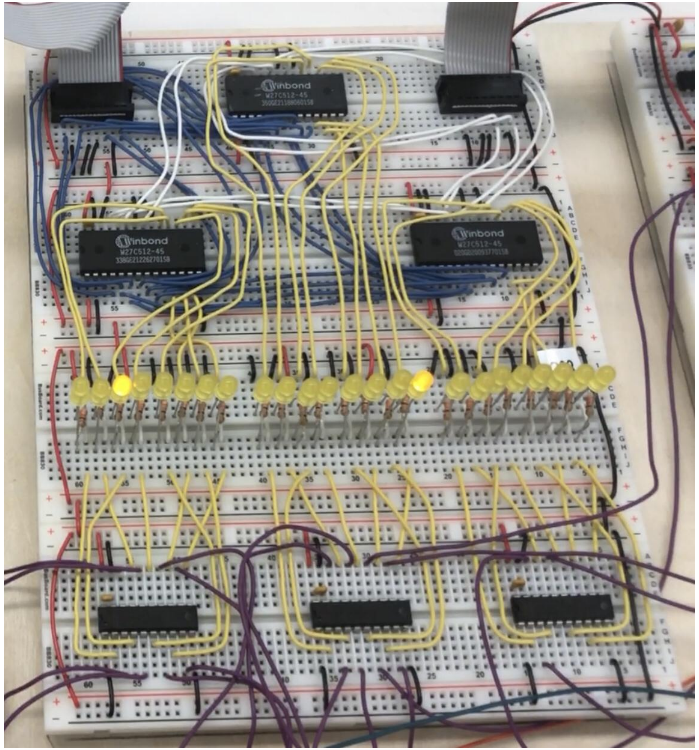
PCB Testing

- Self-Check Program for Several Opcodes
- Reassurances Testing
- Power Analysis
- Clock Frequency Limit Tests
- Verification Testing
- Full System Integration Testing
- Comparing Simulator Results to Hardware Results

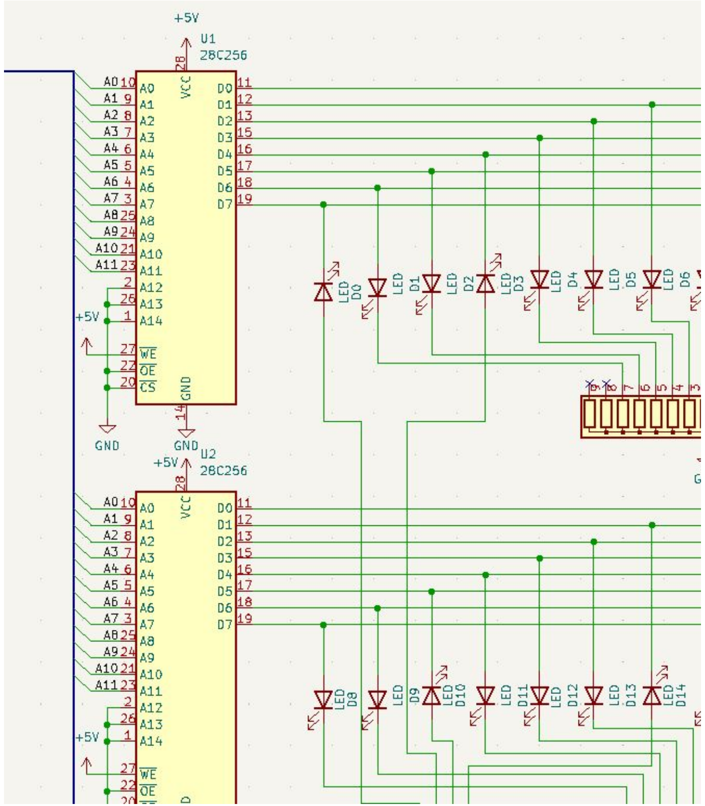
Lessons Learned

Grant Nordling
Electrical Engineering

Circuit Validation

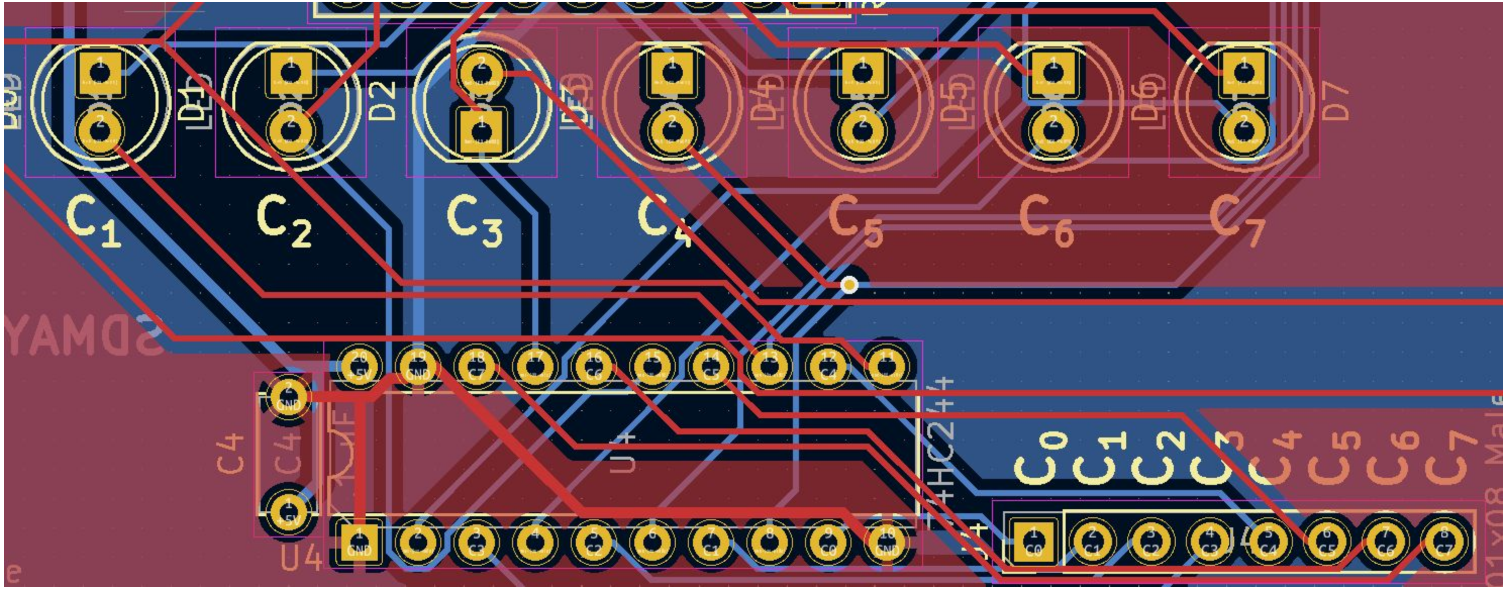


⇒

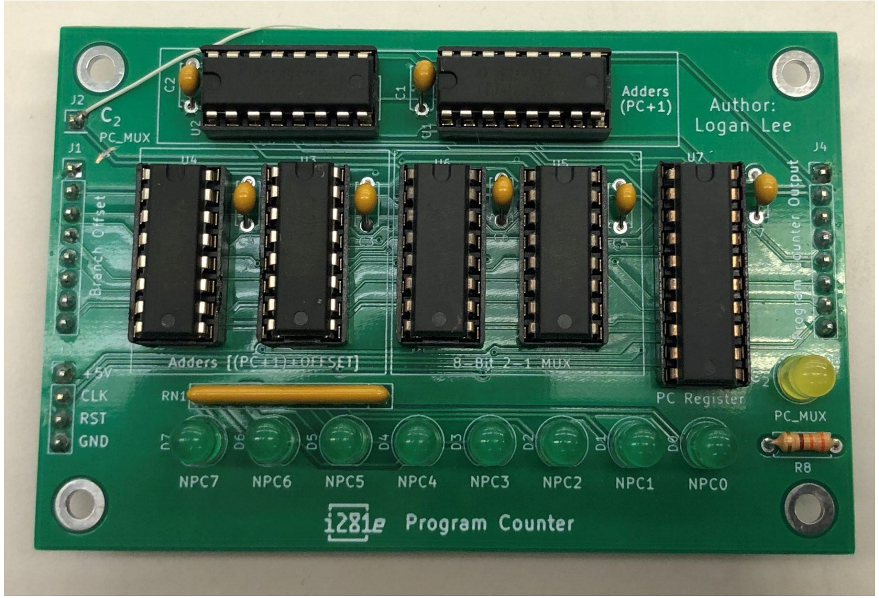


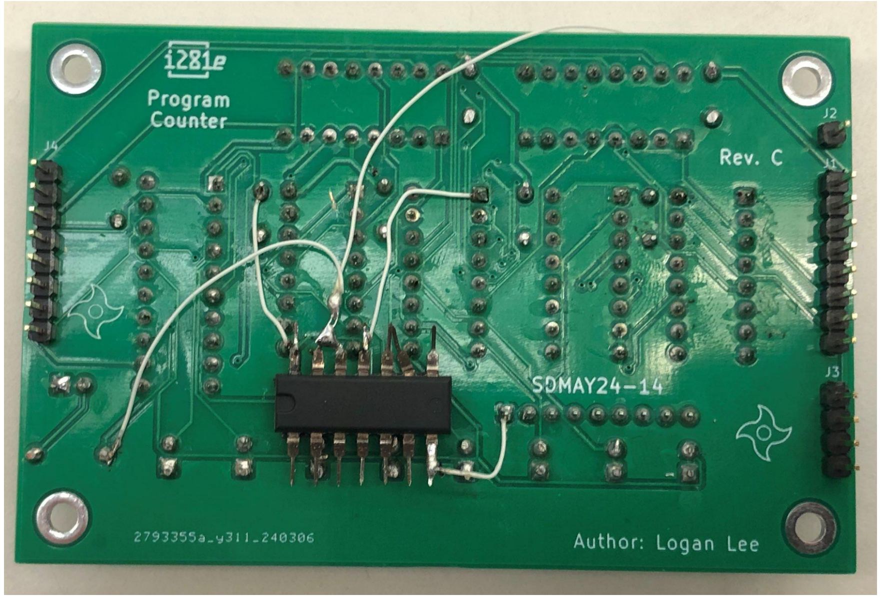
PCB Routing

freerouting

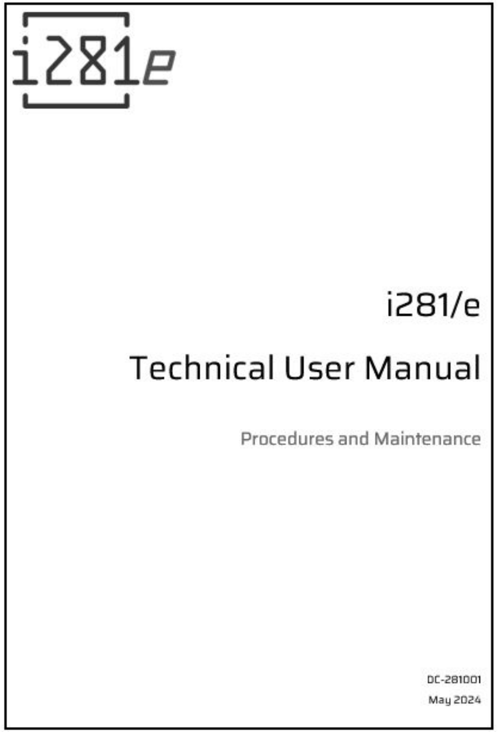


Fixing PCBs: Inverted Select Bit fixed with NOT gate IC





Documentation & Bill of Materials



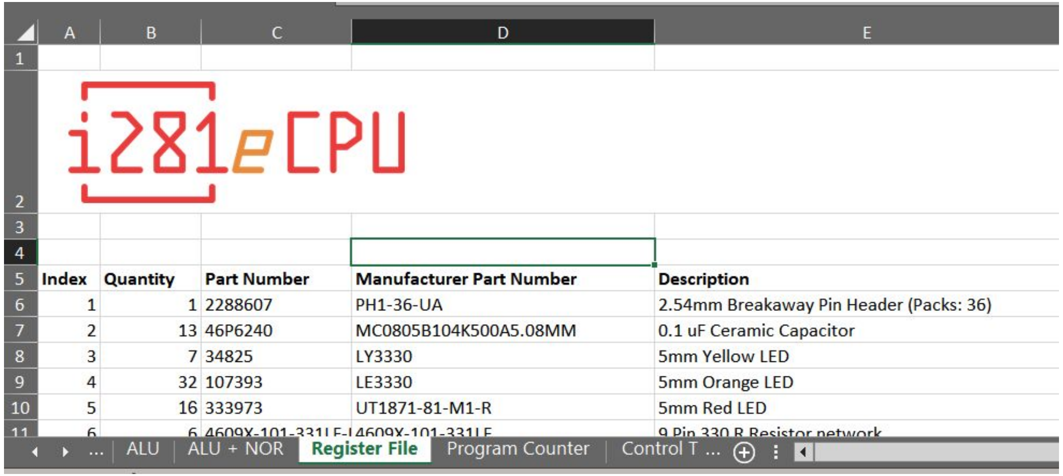
i281/e

Technical User Manual

Procedures and Maintenance

DC-281001

May 2024



i281e CPU

Index	Quantity	Part Number	Manufacturer Part Number	Description
1	1	2288607	PH1-36-UA	2.54mm Breakaway Pin Header (Packs: 36)
2	13	46P6240	MC0805B104K500A5.08MM	0.1 uF Ceramic Capacitor
3	7	34825	LY3330	5mm Yellow LED
4	32	107393	LE3330	5mm Orange LED
5	16	333973	UT1871-81-M1-R	5mm Red LED
6	6	4609X-101-331LF-I	4609X-101-331LF	9 Pin 330 R Resistor network

Acknowledgements

- Matt Post, Lee Harker, and ETG Student Staff
- ECPE Department
- Dr. Alexander Stoytchev

i281e CPU

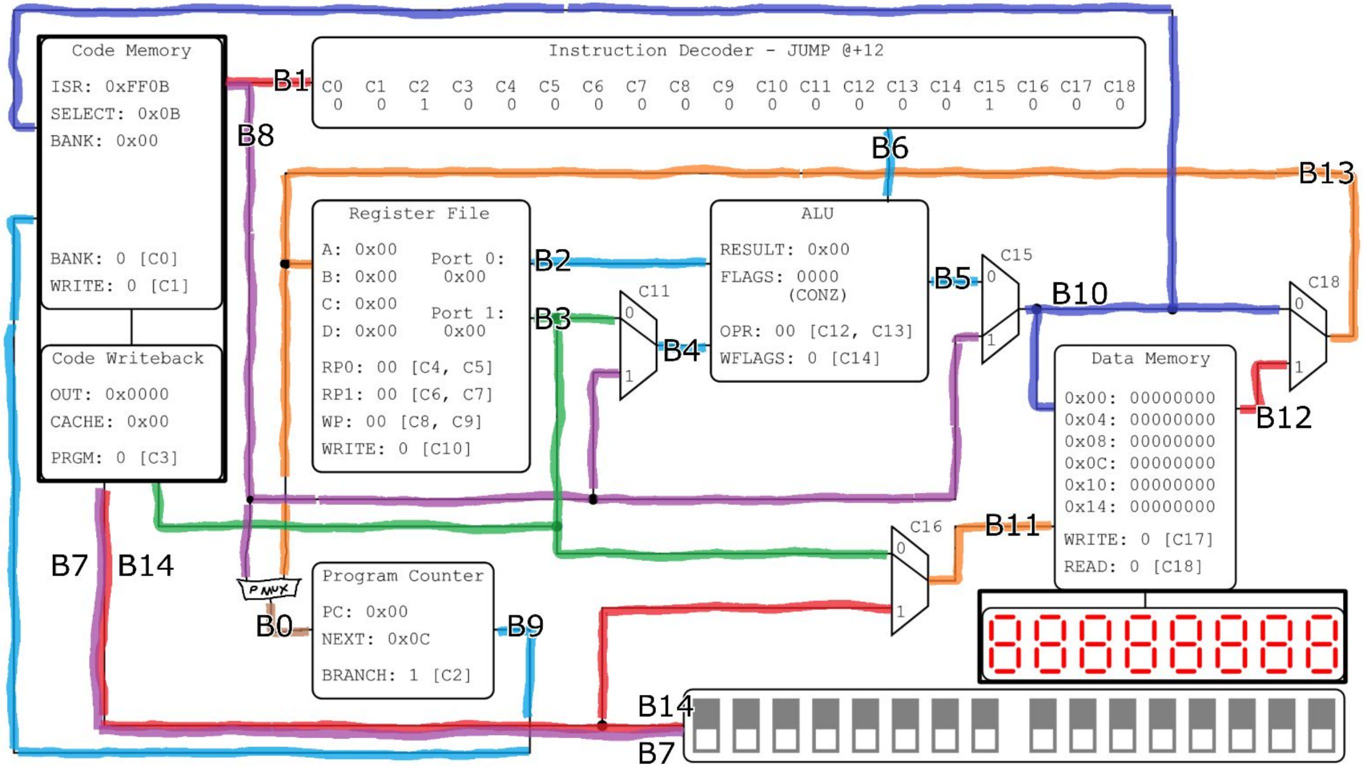
Discover the *extended* difference.

Questions?

THE END

Backup Slides

Mainboard Bus Connectors



Code Memory
ISR: 0xFF0B
SELECT: 0x0B
BANK: 0x00
BANK: 0 [C0]
WRITE: 0 [C1]

Code Writeback
OUT: 0x0000
CACHE: 0x00
PRGM: 0 [C3]

Instruction Decoder - JUMP @+12

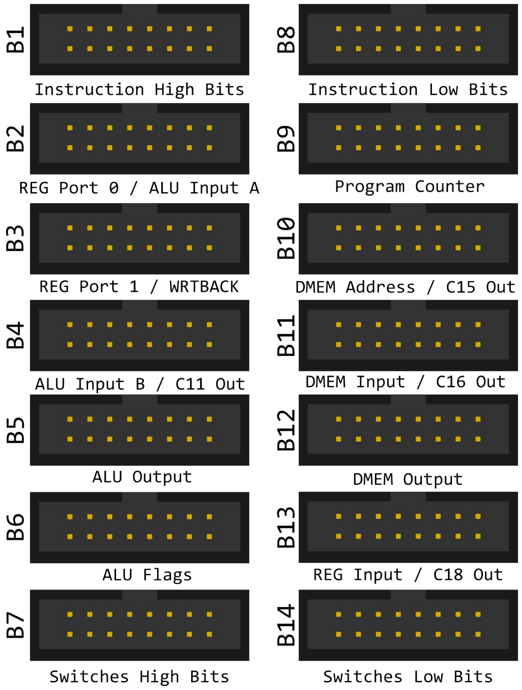
C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Register File
A: 0x00
B: 0x00
C: 0x00
D: 0x00
Port 0: 0x00
Port 1: 0x00
RP0: 00 [C4, C5]
RP1: 00 [C6, C7]
WP: 00 [C8, C9]
WRITE: 0 [C10]

ALU
RESULT: 0x00
FLAGS: 0000 (CONZ)
OPR: 00 [C12, C13]
WFLAGS: 0 [C14]

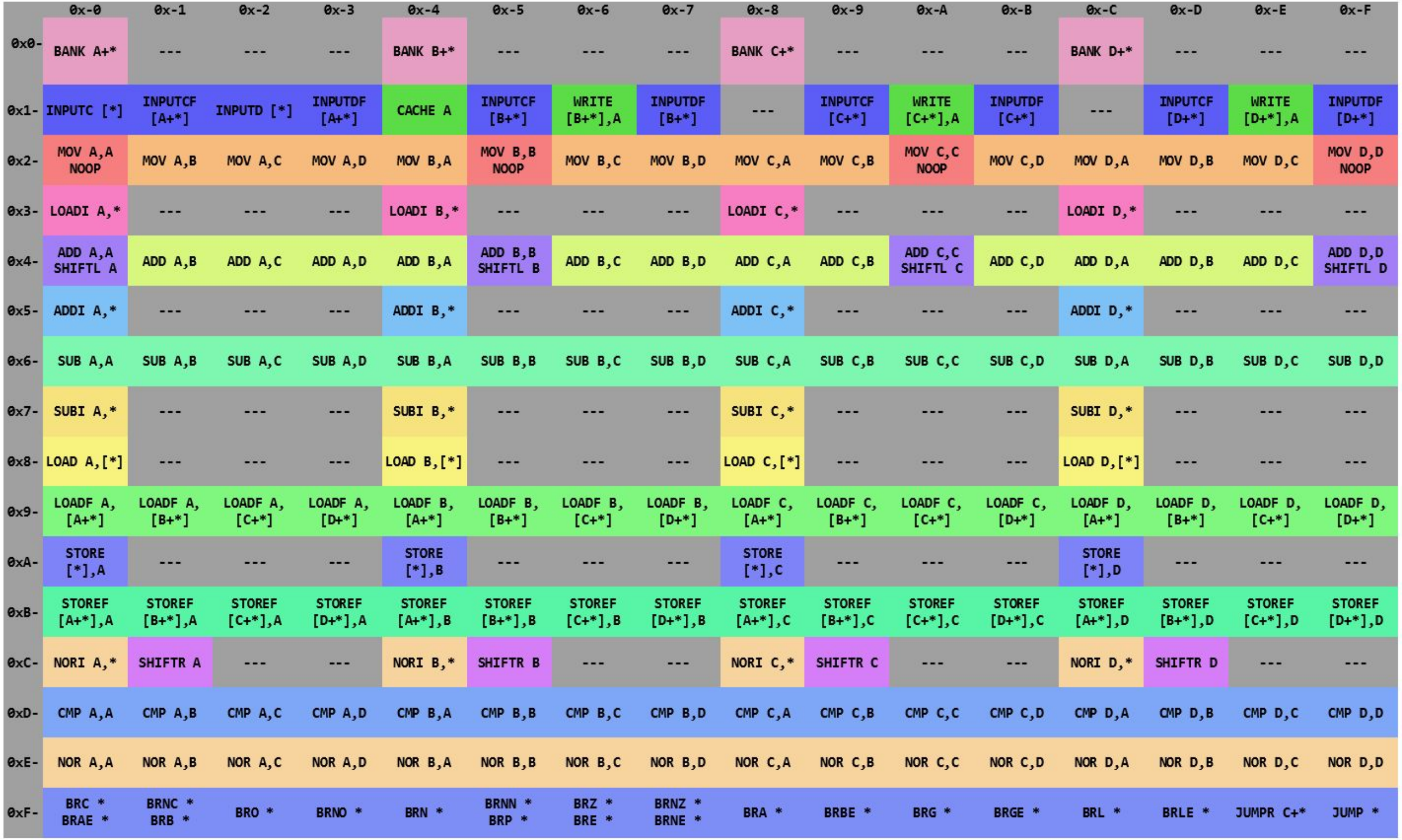
Data Memory
0x00: 00000000
0x04: 00000000
0x08: 00000000
0x0C: 00000000
0x10: 00000000
0x14: 00000000
WRITE: 0 [C17]
READ: 0 [C18]

Program Counter
PC: 0x00
NEXT: 0x0C
BRANCH: 1 [C2]



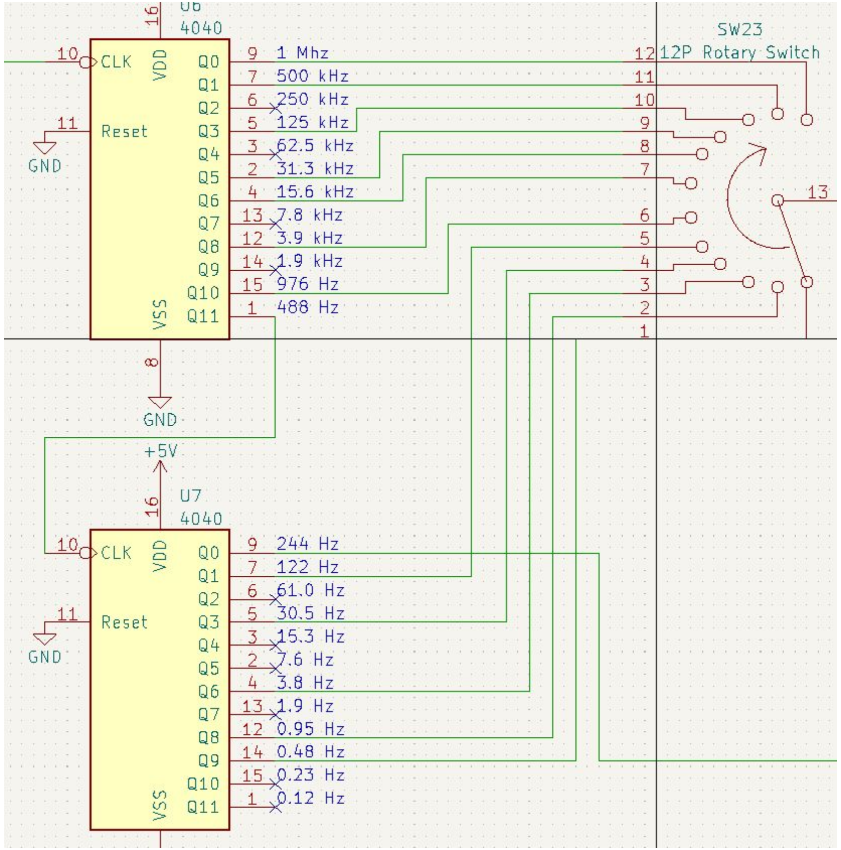
i281e

Expanded i281e Instruction Set Table



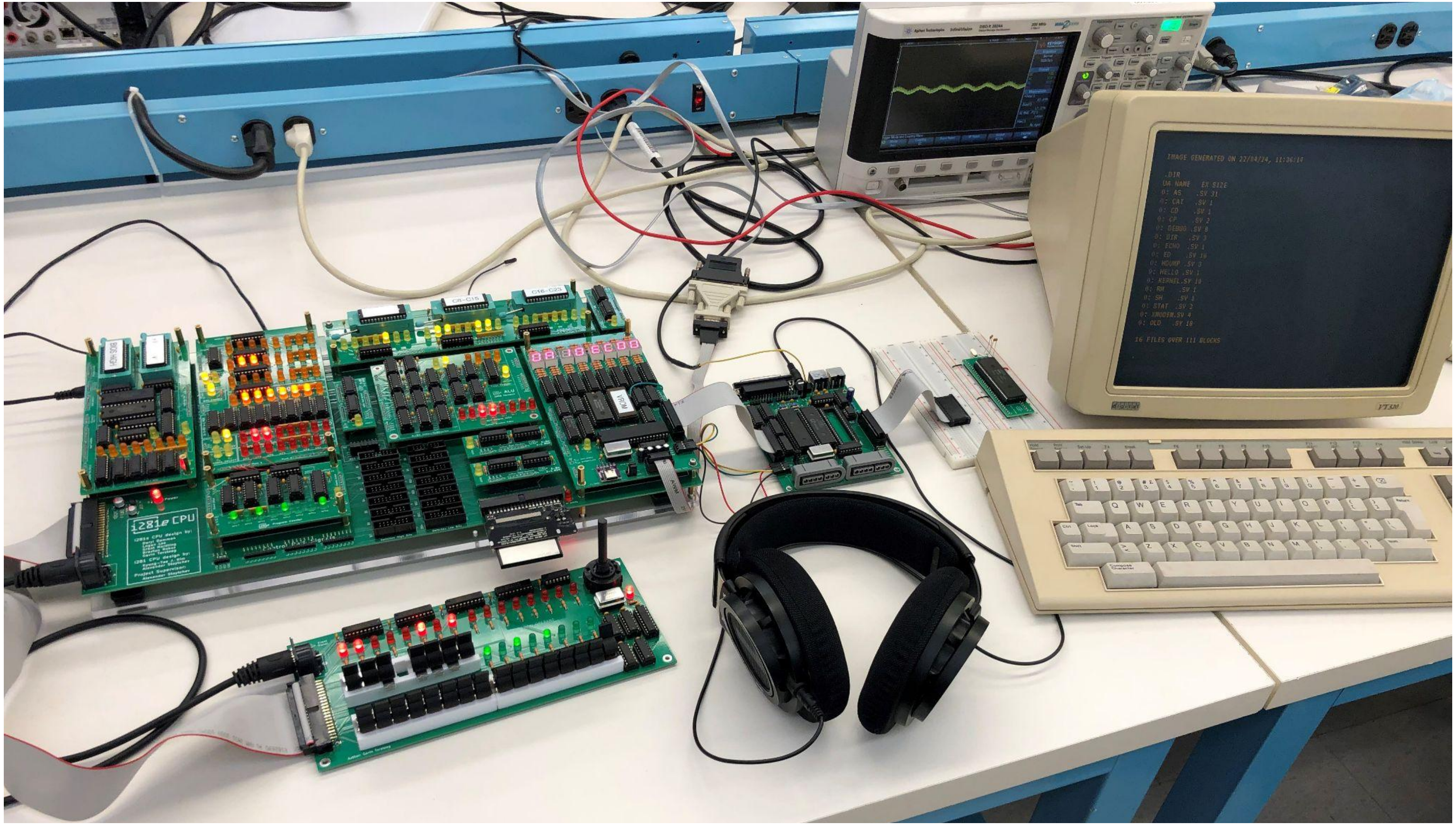
	0x-0	0x-1	0x-2	0x-3	0x-4	0x-5	0x-6	0x-7	0x-8	0x-9	0x-A	0x-B	0x-C	0x-D	0x-E	0x-F
0x0-	BANK A+*	---	---	---	BANK B+*	---	---	---	BANK C+*	---	---	---	BANK D+*	---	---	---
0x1-	INPUTC [*]	INPUTCF [A+*]	INPUTD [*]	INPUTDF [A+*]	CACHE A	INPUTCF [B+*]	WRITE [B+*],A	INPUTDF [B+*]	---	INPUTCF [C+*]	WRITE [C+*],A	INPUTDF [C+*]	---	INPUTCF [D+*]	WRITE [D+*],A	INPUTDF [D+*]
0x2-	MOV A,A NOOP	MOV A,B	MOV A,C	MOV A,D	MOV B,A	MOV B,B NOOP	MOV B,C	MOV B,D	MOV C,A	MOV C,B	MOV C,C NOOP	MOV C,D	MOV D,A	MOV D,B	MOV D,C	MOV D,D NOOP
0x3-	LOADI A,*	---	---	---	LOADI B,*	---	---	---	LOADI C,*	---	---	---	LOADI D,*	---	---	---
0x4-	ADD A,A SHIFTL A	ADD A,B	ADD A,C	ADD A,D	ADD B,A	ADD B,B SHIFTL B	ADD B,C	ADD B,D	ADD C,A	ADD C,B	ADD C,C SHIFTL C	ADD C,D	ADD D,A	ADD D,B	ADD D,C	ADD D,D SHIFTL D
0x5-	ADDI A,*	---	---	---	ADDI B,*	---	---	---	ADDI C,*	---	---	---	ADDI D,*	---	---	---
0x6-	SUB A,A	SUB A,B	SUB A,C	SUB A,D	SUB B,A	SUB B,B	SUB B,C	SUB B,D	SUB C,A	SUB C,B	SUB C,C	SUB C,D	SUB D,A	SUB D,B	SUB D,C	SUB D,D
0x7-	SUBI A,*	---	---	---	SUBI B,*	---	---	---	SUBI C,*	---	---	---	SUBI D,*	---	---	---
0x8-	LOAD A,[*]	---	---	---	LOAD B,[*]	---	---	---	LOAD C,[*]	---	---	---	LOAD D,[*]	---	---	---
0x9-	LOADF A, [A+*]	LOADF A, [B+*]	LOADF A, [C+*]	LOADF A, [D+*]	LOADF B, [A+*]	LOADF B, [B+*]	LOADF B, [C+*]	LOADF B, [D+*]	LOADF C, [A+*]	LOADF C, [B+*]	LOADF C, [C+*]	LOADF C, [D+*]	LOADF D, [A+*]	LOADF D, [B+*]	LOADF D, [C+*]	LOADF D, [D+*]
0xA-	STORE [*],A	---	---	---	STORE [*],B	---	---	---	STORE [*],C	---	---	---	STORE [*],D	---	---	---
0xB-	STOREF [A+*],A	STOREF [B+*],A	STOREF [C+*],A	STOREF [D+*],A	STOREF [A+*],B	STOREF [B+*],B	STOREF [C+*],B	STOREF [D+*],B	STOREF [A+*],C	STOREF [B+*],C	STOREF [C+*],C	STOREF [D+*],C	STOREF [A+*],D	STOREF [B+*],D	STOREF [C+*],D	STOREF [D+*],D
0xC-	NORI A,*	SHIFTR A	---	---	NORI B,*	SHIFTR B	---	---	NORI C,*	SHIFTR C	---	---	NORI D,*	SHIFTR D	---	---
0xD-	CMP A,A	CMP A,B	CMP A,C	CMP A,D	CMP B,A	CMP B,B	CMP B,C	CMP B,D	CMP C,A	CMP C,B	CMP C,C	CMP C,D	CMP D,A	CMP D,B	CMP D,C	CMP D,D
0xE-	NOR A,A	NOR A,B	NOR A,C	NOR A,D	NOR B,A	NOR B,B	NOR B,C	NOR B,D	NOR C,A	NOR C,B	NOR C,C	NOR C,D	NOR D,A	NOR D,B	NOR D,C	NOR D,D
0xF-	BRC * BRAE *	BRNC * BRB *	BRO *	BRNO *	BRN *	BRNN * BRP *	BRZ * BRE *	BRNZ * BRNE *	BRA *	BRBE *	BRG *	BRGE *	BRL *	BRLE *	JUMPR C+*	JUMP *

Adjustable Clock



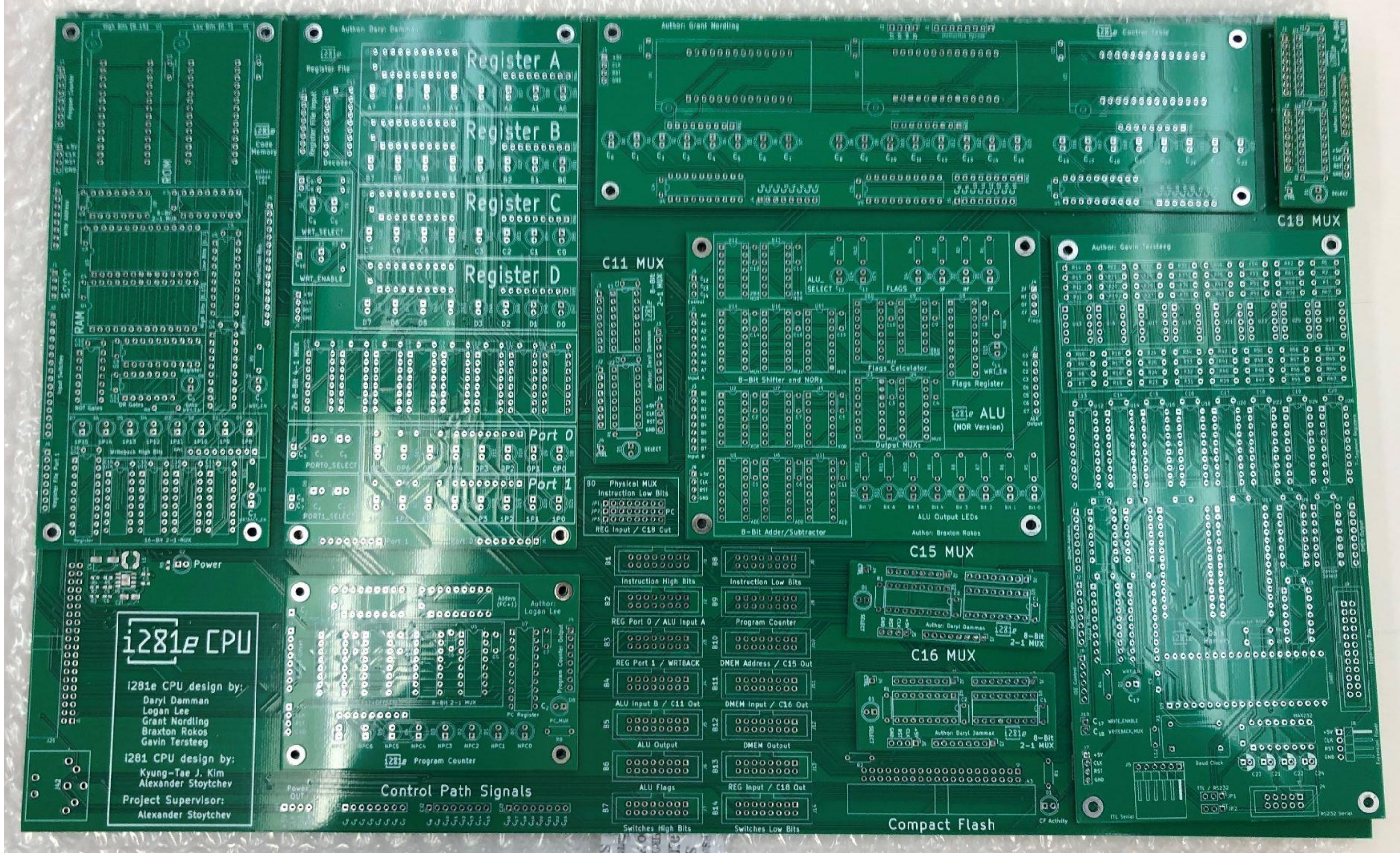
Adjustable CPU clock to enable stepping through individual instructions and operating at different clock frequencies.

Expansion Capability



```
IMAGE GENERATED ON 22/04/24, 11:36:14

.DIR
UA NAME   EX SIZE
0: AS     .SV 31
0: CAT    .SV 1
0: CD     .SV 1
0: CP     .SV 2
0: DEBUG  .SV 8
0: DIR    .SV 3
0: ECHO   .SV 1
0: ED     .SV 10
0: HDUMP  .SV 3
0: HELLO  .SV 1
0: KERNEL.SV 10
0: RM     .SV 1
0: SH     .SV 1
0: STAT   .SV 2
0: XMODEM.SV 4
0: OLD    .SV 10

16 FILES OVER 111 BLOCKS
```

Mainboard PCB with Overlaid Modules



Project Vision

The i281 CPU was designed to support the curriculum in CprE 281: Digital Logic. The CPU has no physical form capable of creating hands-on experiences for students. The CPU must be created in hardware and be similar to the FPGA and Simulator designs. The project consists of two stages: Breadboard and Printed Circuit Board. The Breadboard version must be created first to prove the concept and debug any issues. The Printed Circuit Board version will use the designs from the Breadboard version and implement them in a fashion that is easier to reproduce and learn from.

Project Vision

The students in the CPR E 281 digital logic course lack a physical design of an i281 CPU they can use to apply their learning.

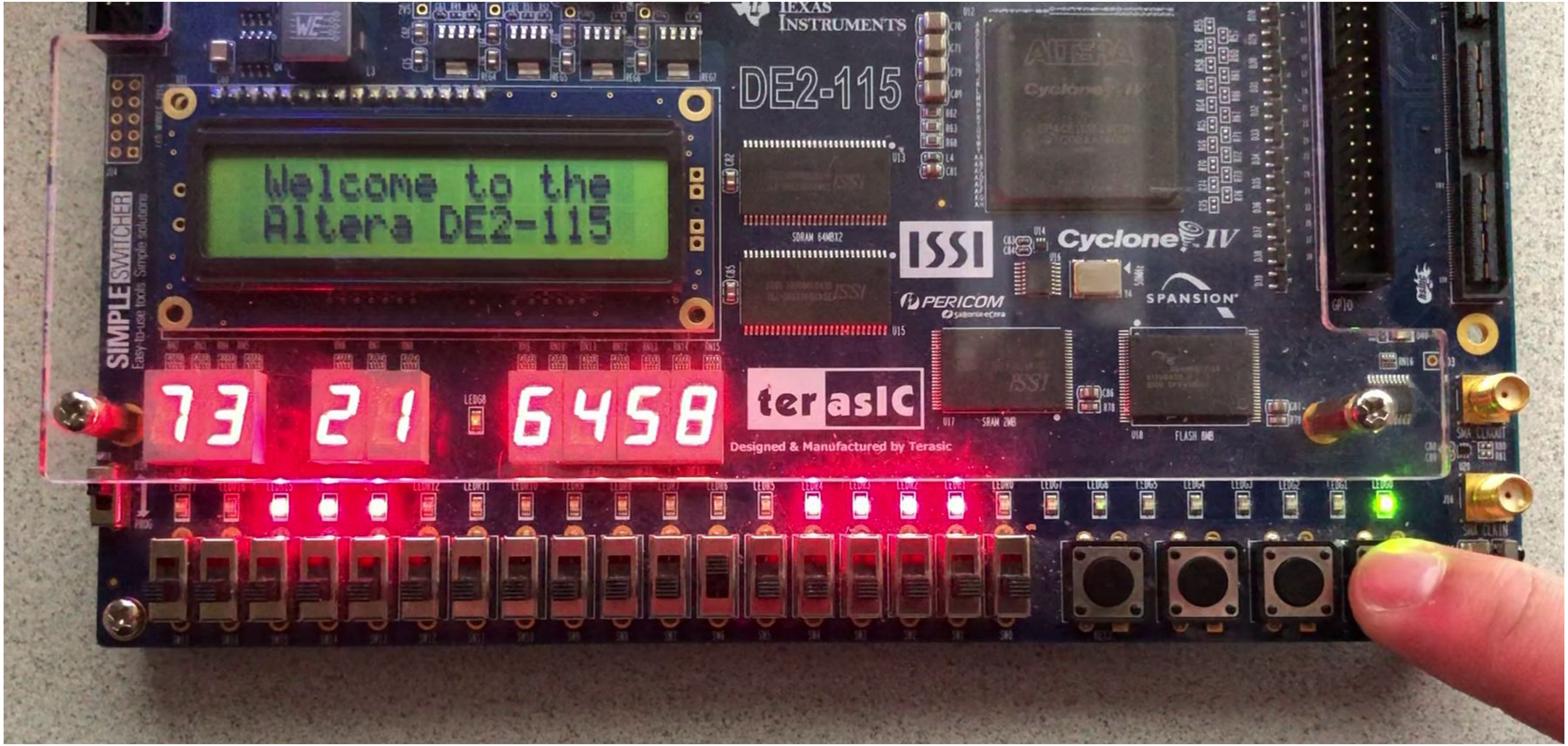
When we originally laid out the curriculum, there were three classes (X81). The motivation for the project is to better prepare students for 381; however, after all these projects, etc. it was determined the students have no idea how to use breadboards and PCB. There does not exist material and a fully-fledged physical computer for 481, a non-existent class.

This processor is meant to be a sandbox to allow computer engineering students learn digital design experience. Used as a teaching platform for both 281 and 481. The original design is mysterious. This design takes way the confusion and showcases the internals in full.

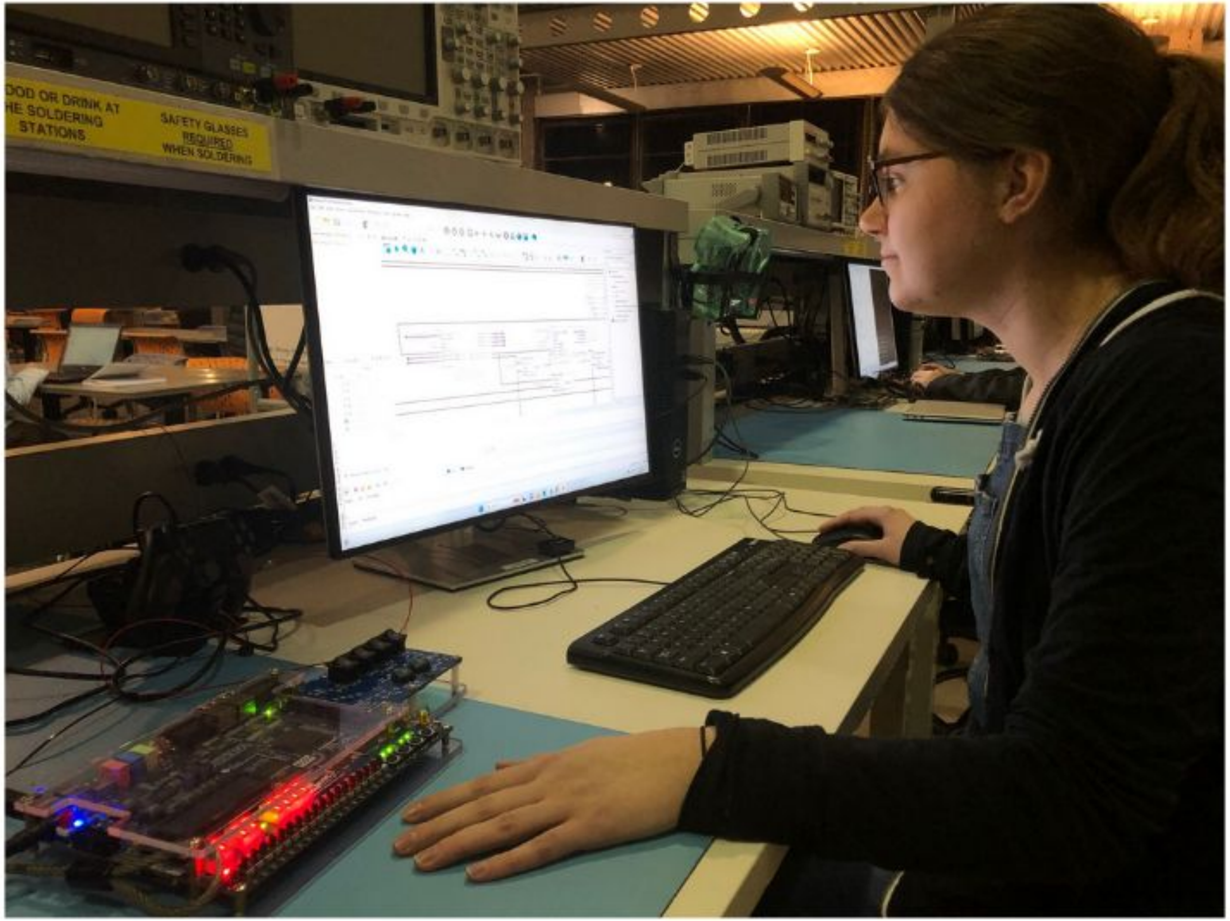
The CPR E 281 Digital Logic Course demonstrates the i281 CPU on both FPGA and Simulators, yet lacks a hardware implementation of it. This project aims to physically build the hardware version of the design to emphasize the branch between software and hardware. - Braxton :)

Breadboard Machine (Early Spring 2024)

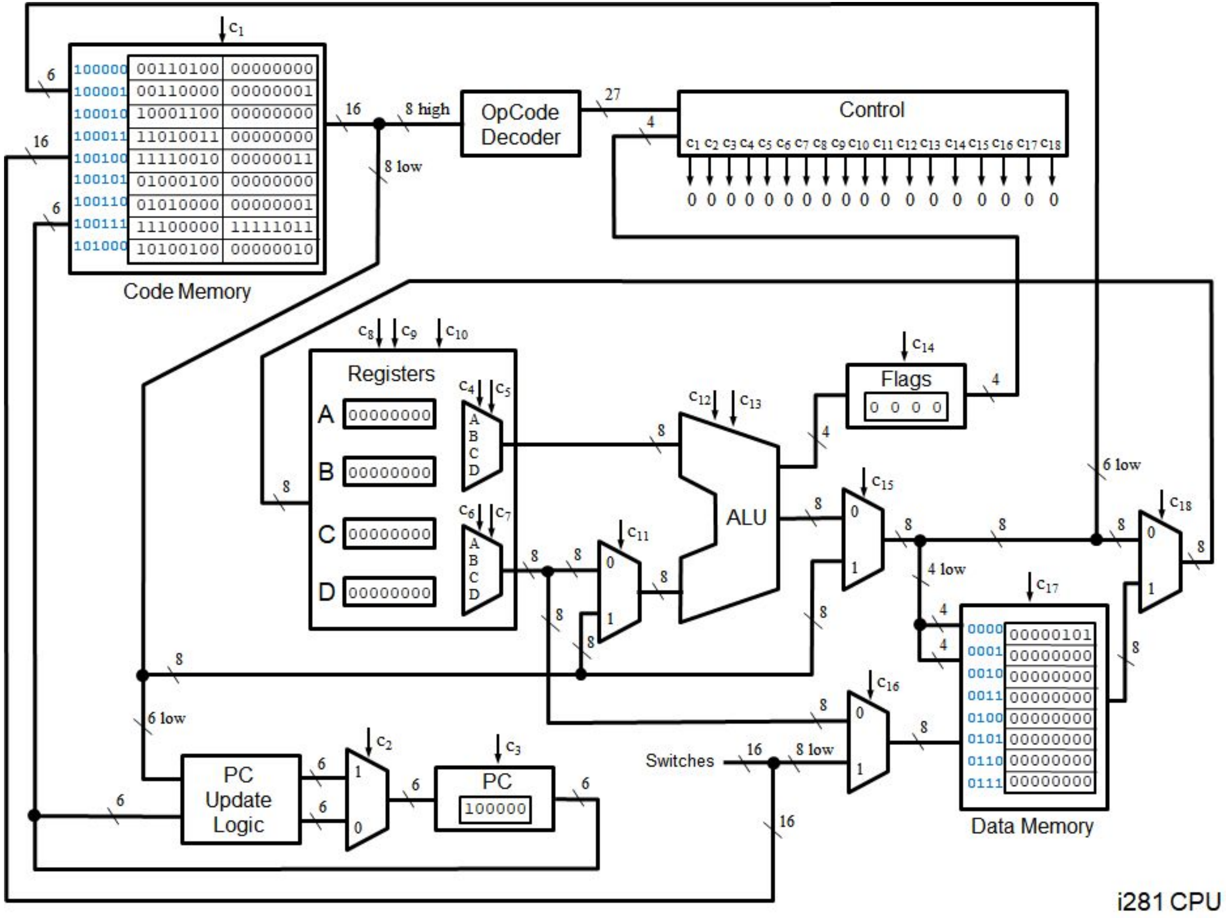
FPGA Implementation (Summer 2019)



i281 CPU covered in CprE 281 labs (Fall 2019 — Present)



i281 CPU covered in CprE 281 lectures (Fall 2019 — Present)



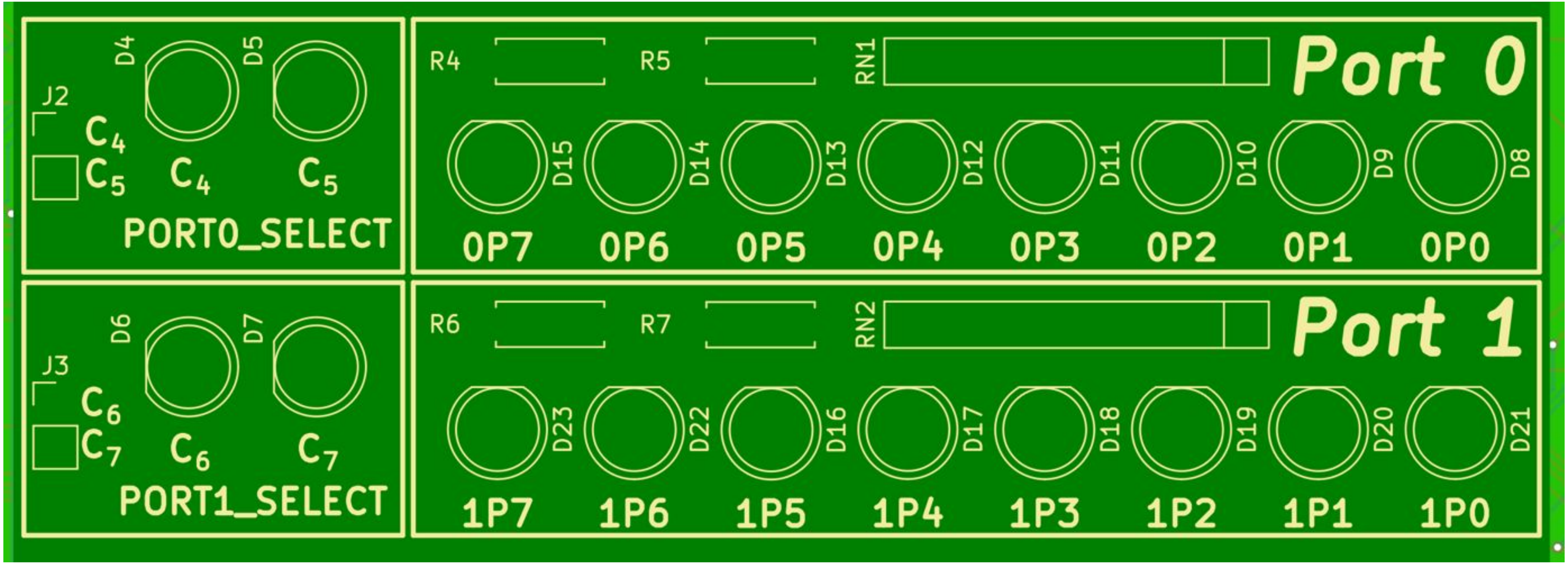
Code Memory

100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

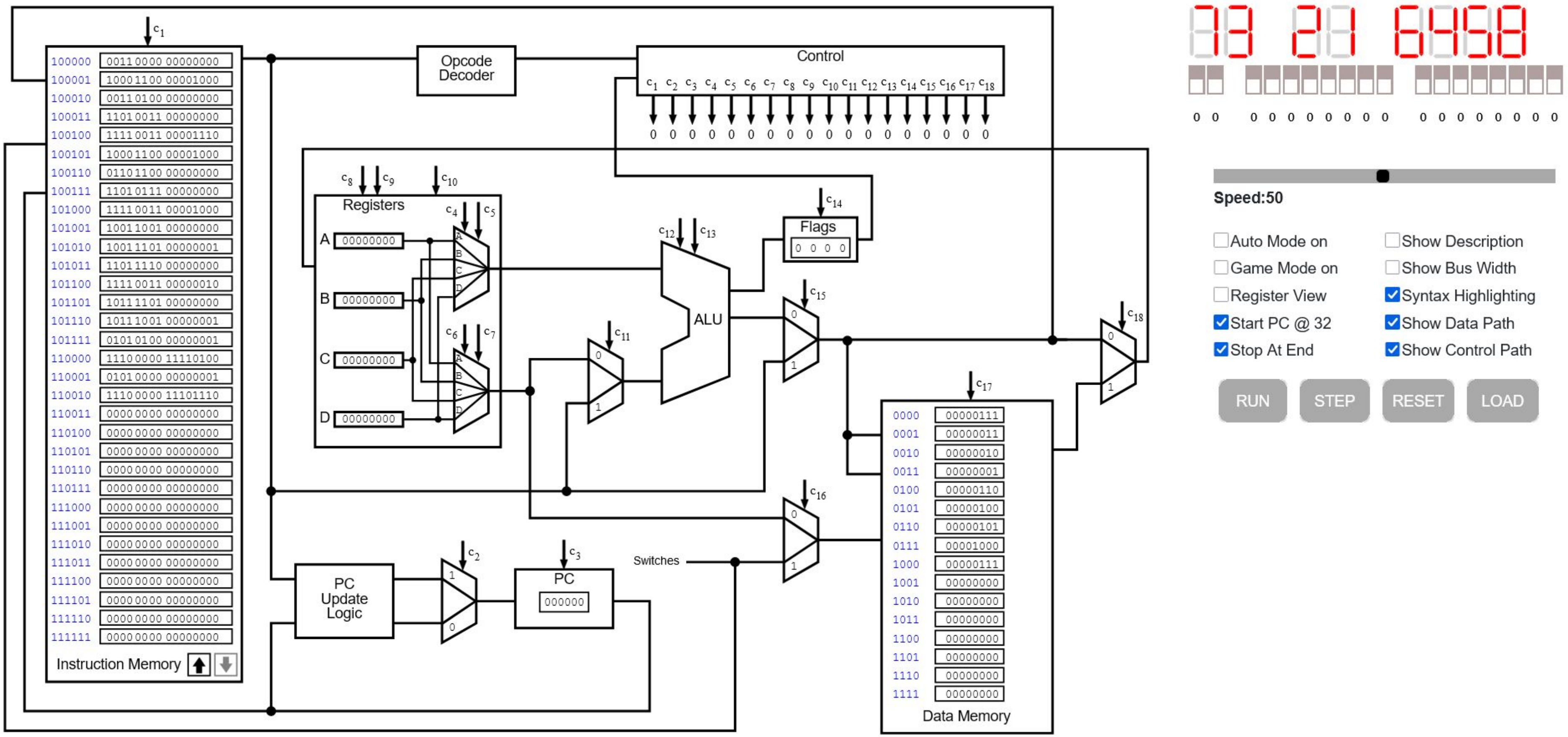
Data Memory

0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

Mindful Layout & Silkscreen



i281 CPU Web Simulator (Spring 2021 — Present)



```
100000  0011 0000 00000000
100001  1000 1100 00001000
100010  0011 0100 00000000
100011  1101 0011 00000000
100100  1111 0011 00001110
100101  1000 1100 00001000
100110  0110 1100 00000000
100111  1101 0111 00000000
101000  1111 0011 00001000
101001  1001 1001 00000000
101010  1001 1101 00000001
101011  1101 1110 00000000
101100  1111 0011 00000010
101101  1011 1101 00000000
101110  1011 1001 00000001
101111  0101 0100 00000001
110000  1110 0000 11110100
110001  0101 0000 00000001
110010  1110 0000 11101110
110011  0000 0000 00000000
110100  0000 0000 00000000
110101  0000 0000 00000000
110110  0000 0000 00000000
110111  0000 0000 00000000
111000  0000 0000 00000000
111001  0000 0000 00000000
111010  0000 0000 00000000
111011  0000 0000 00000000
111100  0000 0000 00000000
111101  0000 0000 00000000
111110  0000 0000 00000000
111111  0000 0000 00000000
```

Instruction Memory

Registers

A 00000000

B 00000000

C 00000000

D 00000000

Flags 0 0 0 0

PC 000000

```
0000  00000111
0001  00000011
0010  00000010
0011  00000001
0100  00000110
0101  00000100
0110  00000101
0111  00001000
1000  00000111
1001  00000000
1010  00000000
1011  00000000
1100  00000000
1101  00000000
1110  00000000
1111  00000000
```

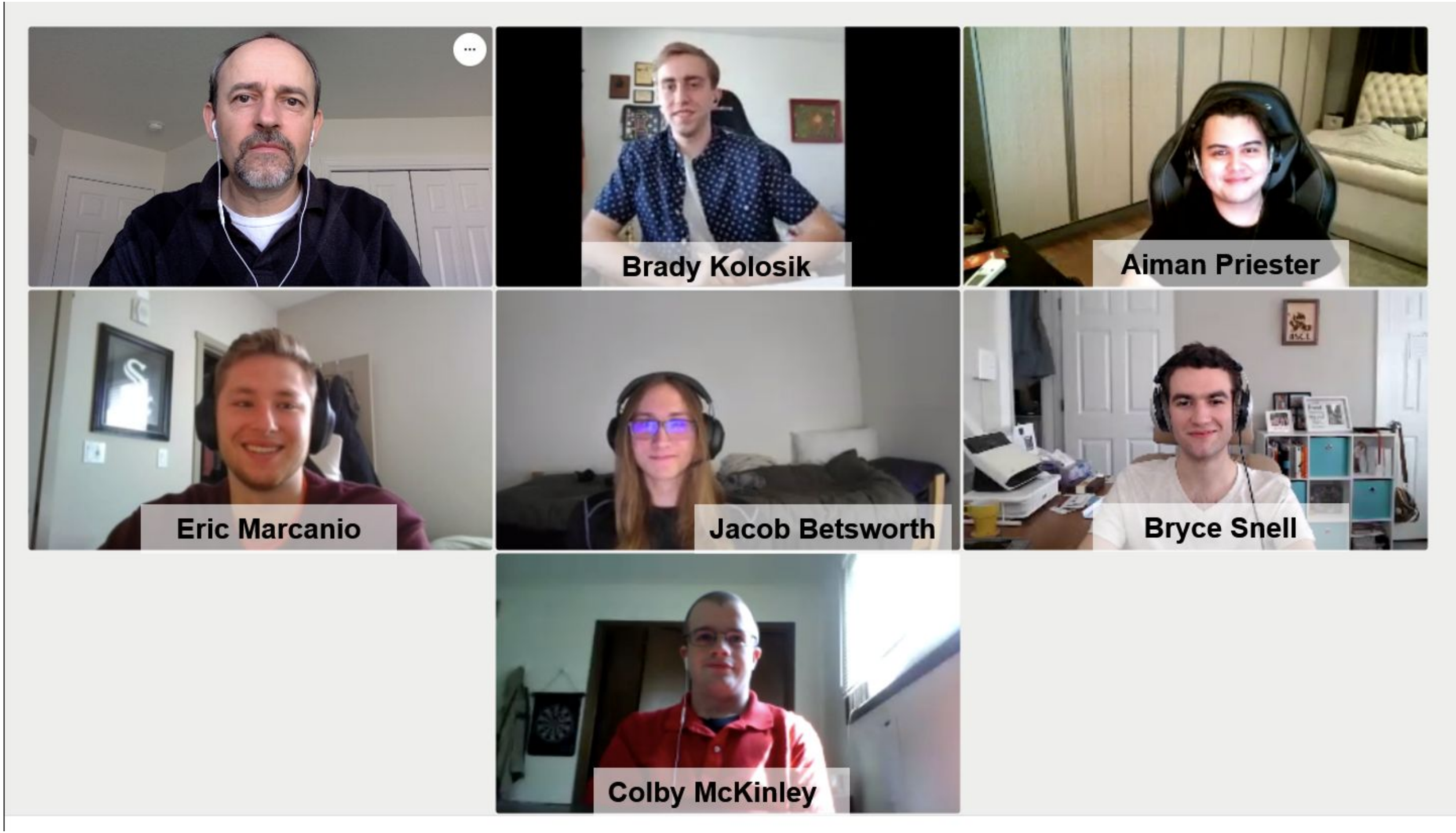
Data Memory

Control: c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18 = 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

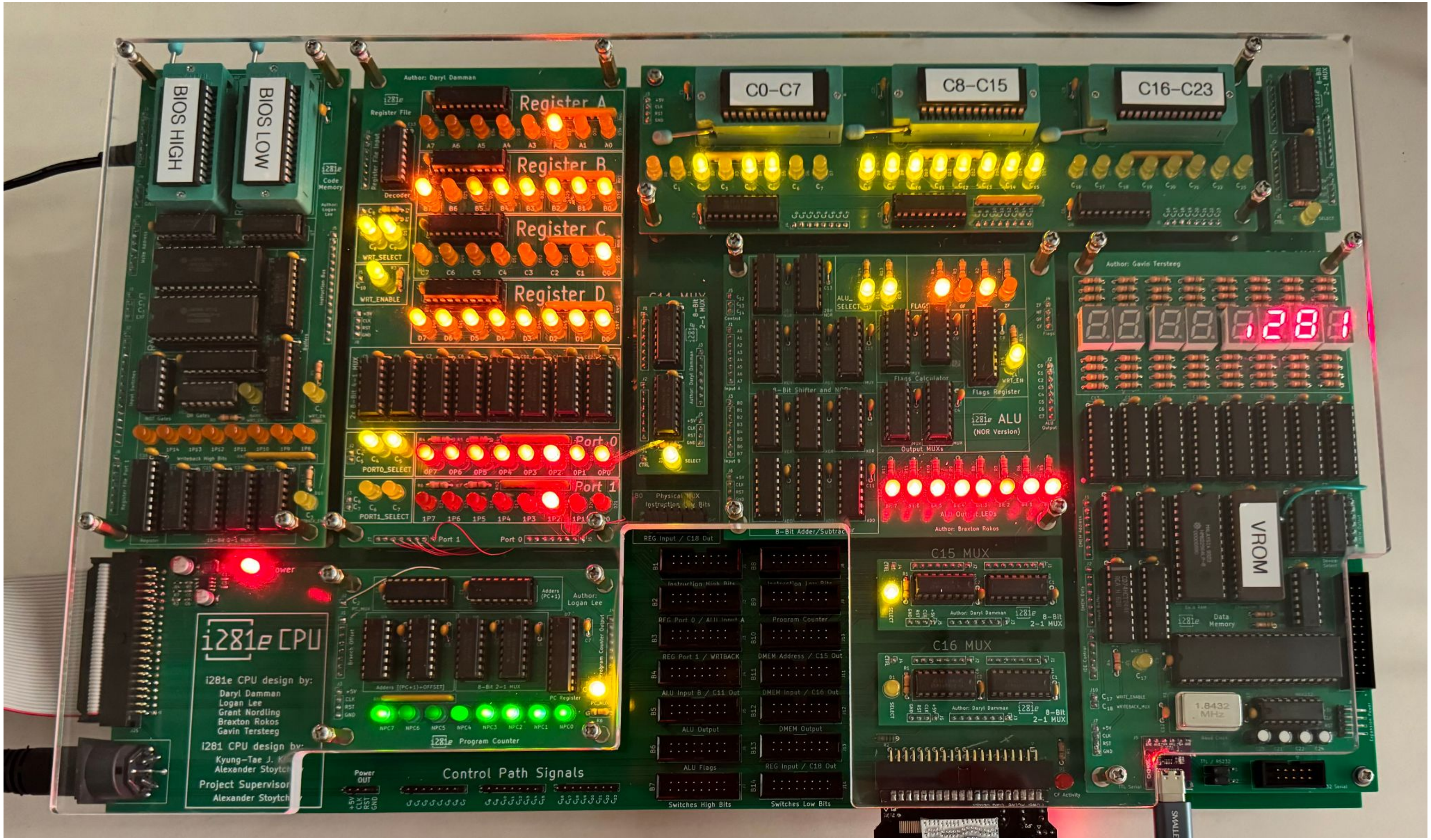
Speed:50

- ☐ Auto Mode on
- ☐ Game Mode on
- ☐ Register View
- ☑ Start PC @ 32
- ☑ Stop At End
- ☐ Show Description
- ☐ Show Bus Width
- ☑ Syntax Highlighting
- ☑ Show Data Path
- ☑ Show Control Path

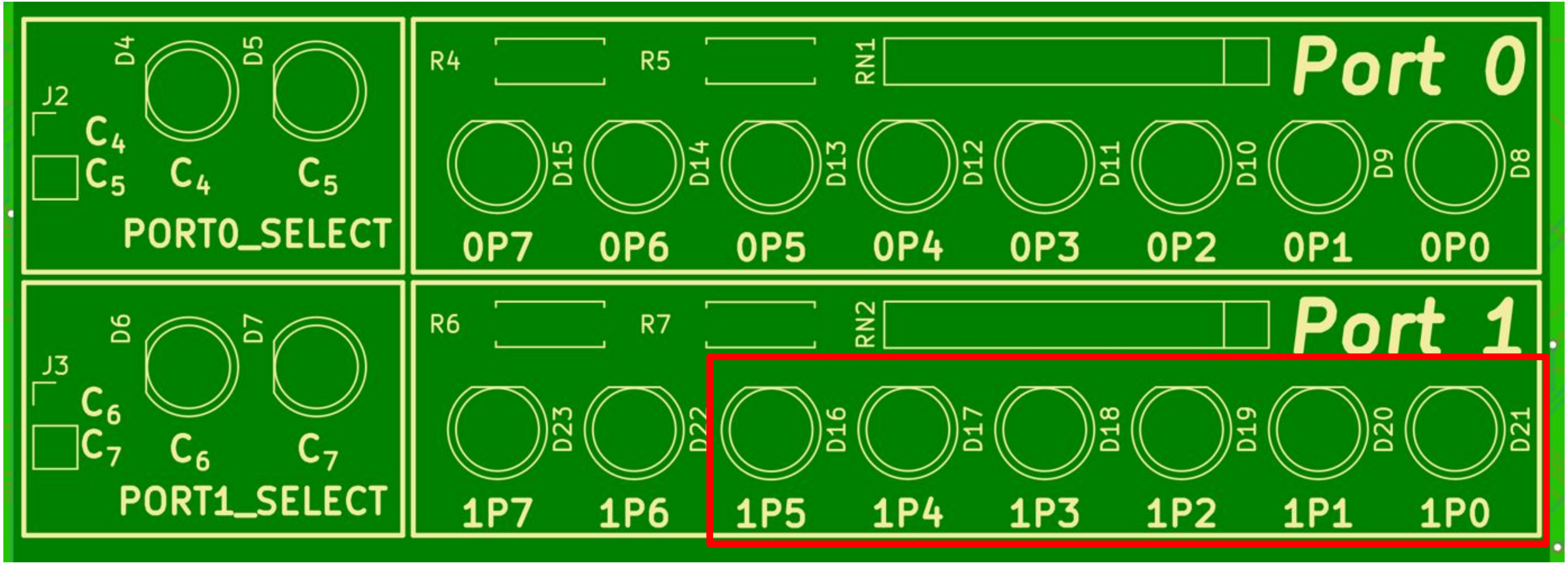
i281 CPU Web Simulator (Finished in Spring 2021)



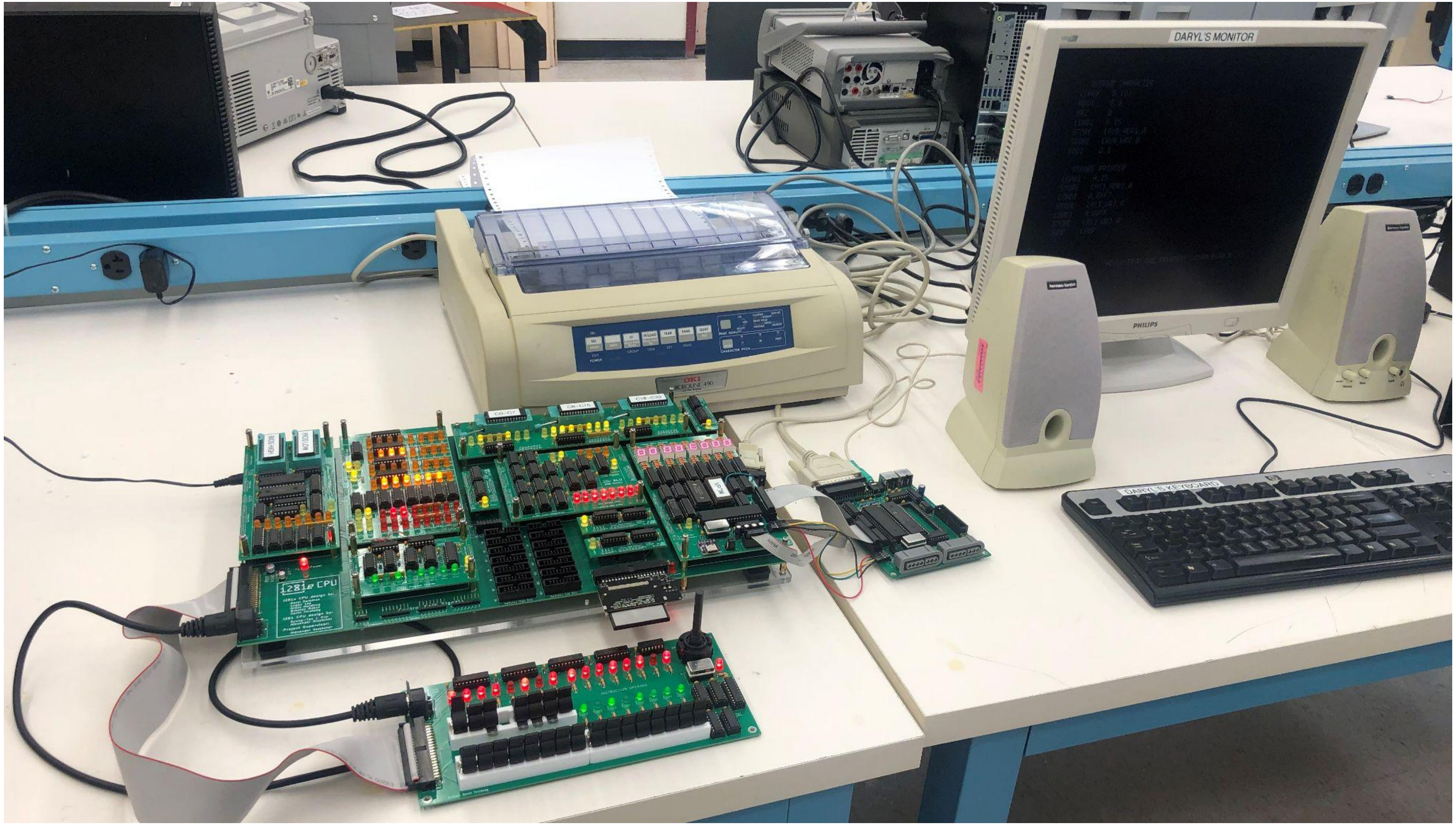
Our Project i281e CPU (Spring 2024)



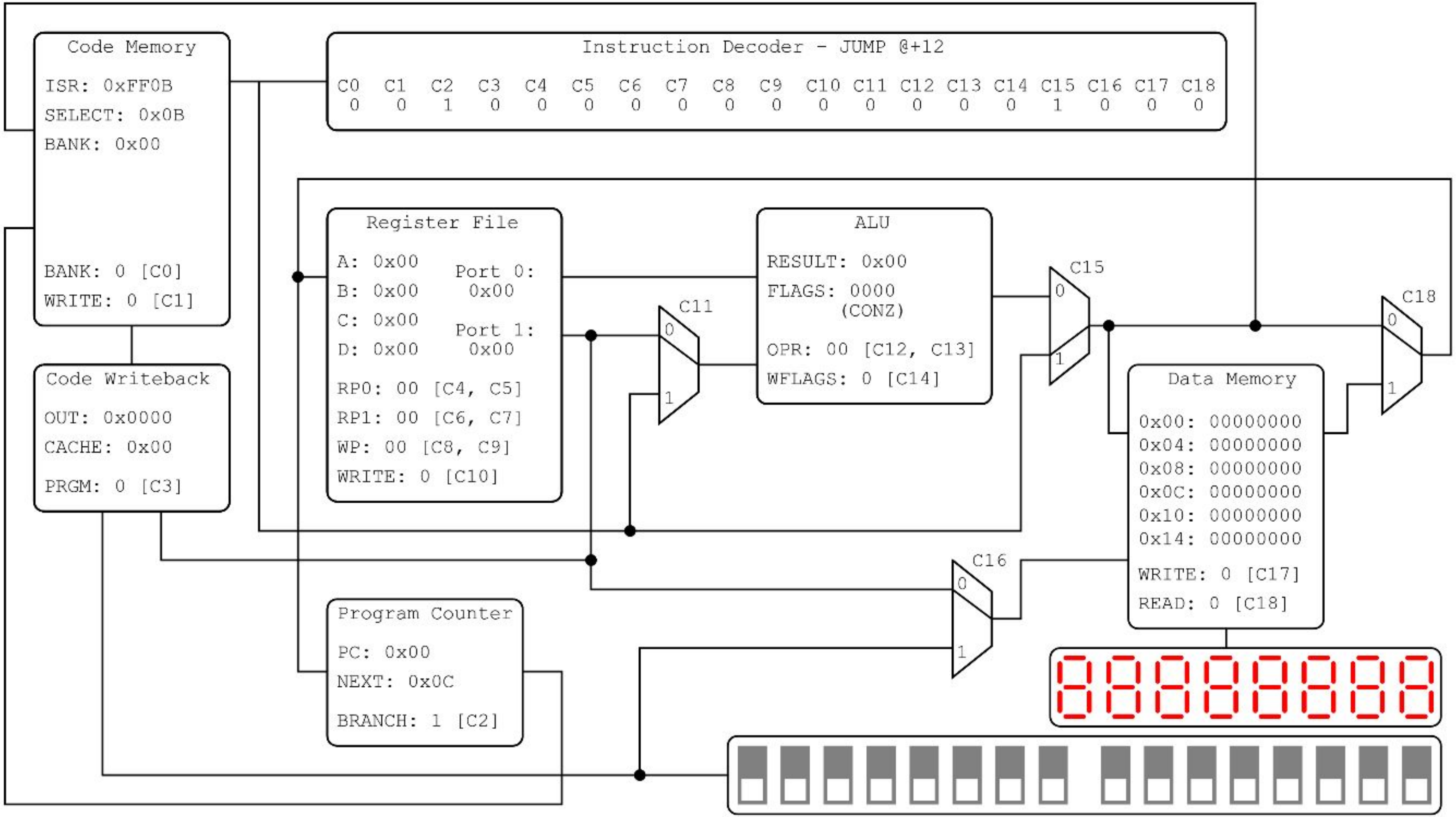
Mindful Layout



Expansion Capability



Web Simulator for the i281e CPU



Code Memory

ISR: 0xFF0B

SELECT: 0x0B

BANK: 0x00

BANK: 0 [C0]

WRITE: 0 [C1]

Instruction Decoder - JUMP @+12

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Register File

A: 0x00

B: 0x00

C: 0x00

D: 0x00

Port 0: 0x00

Port 1: 0x00

RP0: 00 [C4, C5]

RP1: 00 [C6, C7]

WP: 00 [C8, C9]

WRITE: 0 [C10]

ALU

RESULT: 0x00

FLAGS: 0000 (CONZ)

OPR: 00 [C12, C13]

WFLAGS: 0 [C14]

Code Writeback

OUT: 0x0000

CACHE: 0x00

PRGM: 0 [C3]

Data Memory

0x00: 00000000

0x04: 00000000

0x08: 00000000

0x0C: 00000000

0x10: 00000000

0x14: 00000000

WRITE: 0 [C17]

READ: 0 [C18]

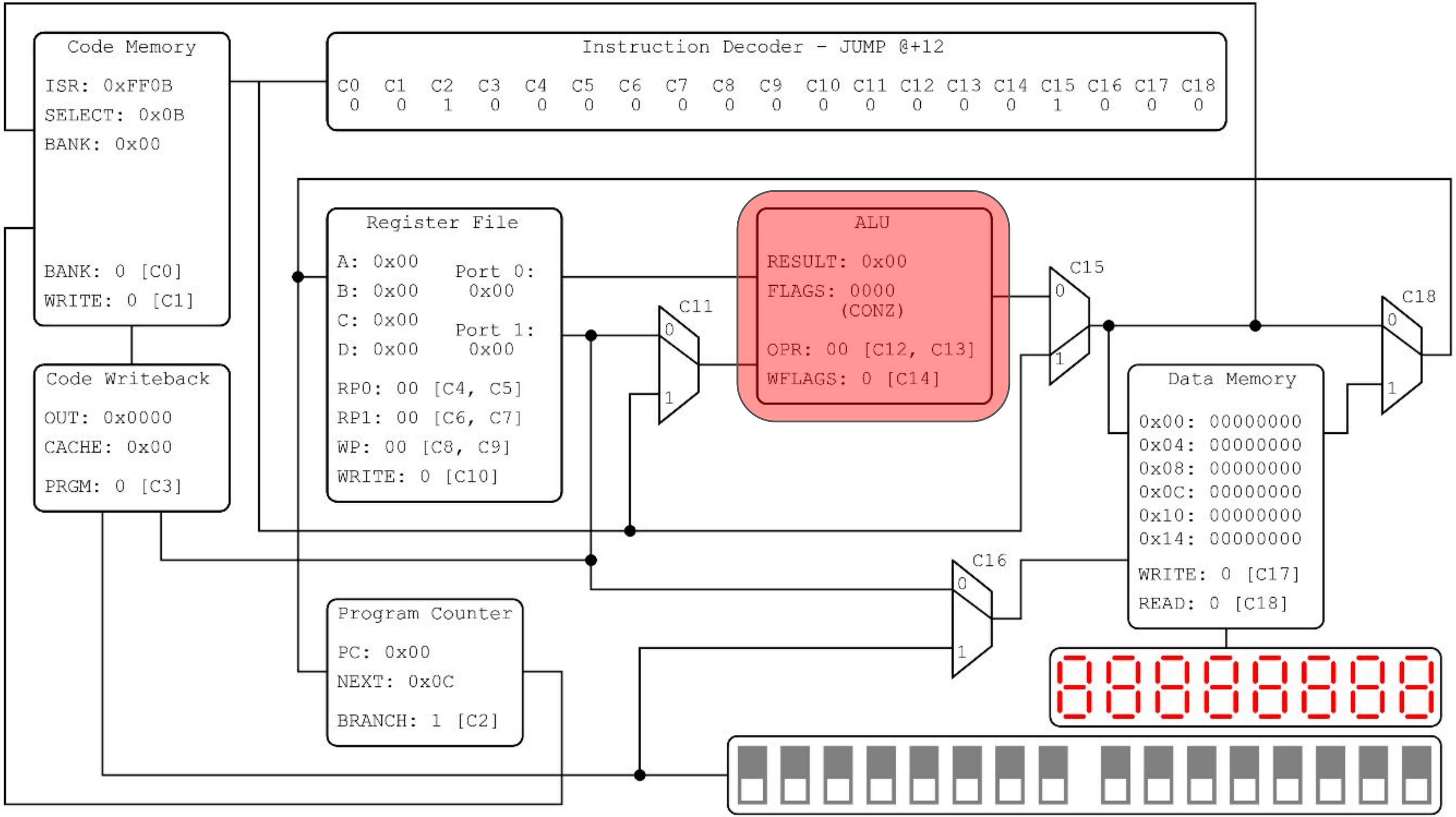
Program Counter

PC: 0x00

NEXT: 0x0C

BRANCH: 1 [C2]

Web Simulator for the i281e CPU



Code Memory

ISR: 0xFF0B

SELECT: 0x0B

BANK: 0x00

BANK: 0 [C0]

WRITE: 0 [C1]

Instruction Decoder - JUMP @+12

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0

Register File

A: 0x00

B: 0x00

C: 0x00

D: 0x00

Port 0:
0x00

Port 1:
0x00

RP0: 00 [C4, C5]

RP1: 00 [C6, C7]

WP: 00 [C8, C9]

WRITE: 0 [C10]

ALU

RESULT: 0x00

FLAGS: 0000
(CONZ)

OPR: 00 [C12, C13]

WFLAGS: 0 [C14]

Code Writeback

OUT: 0x0000

CACHE: 0x00

PRGM: 0 [C3]

Data Memory

0x00: 00000000

0x04: 00000000

0x08: 00000000

0x0C: 00000000

0x10: 00000000

0x14: 00000000

WRITE: 0 [C17]

READ: 0 [C18]

Program Counter

PC: 0x00

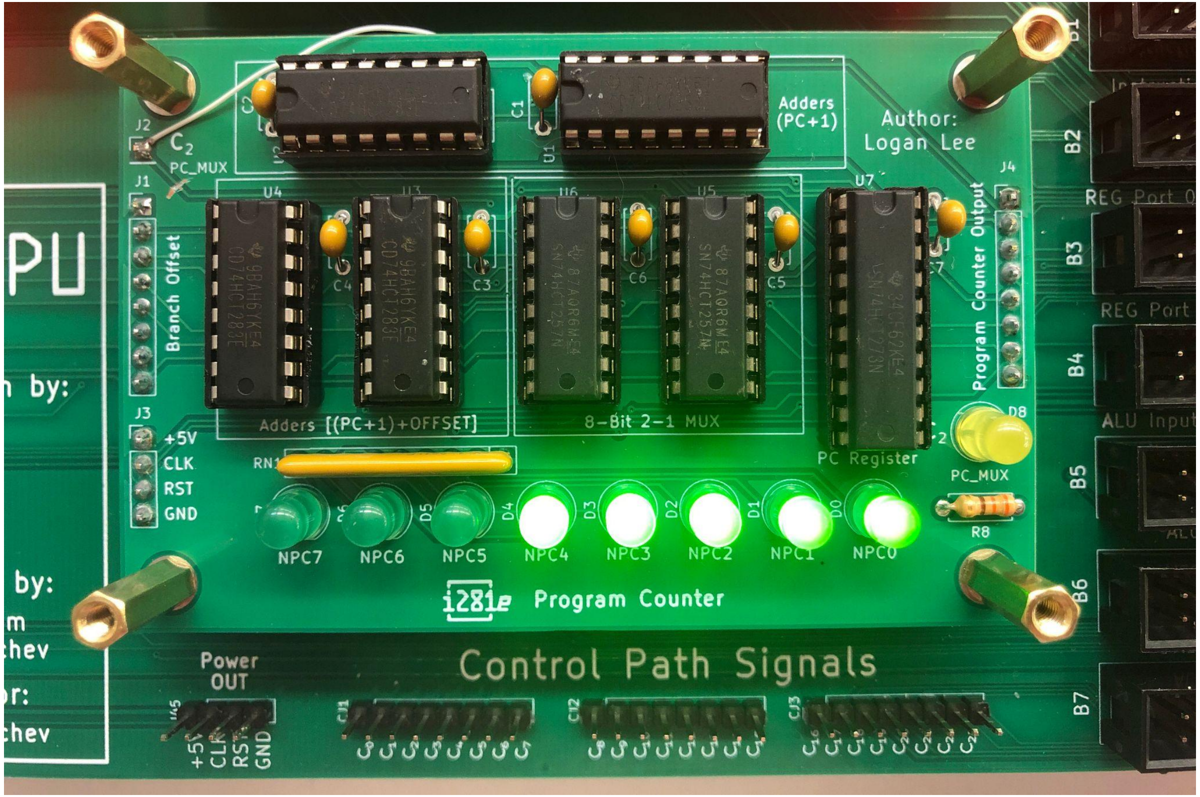
NEXT: 0x0C

BRANCH: 1 [C2]

i281e

Banking

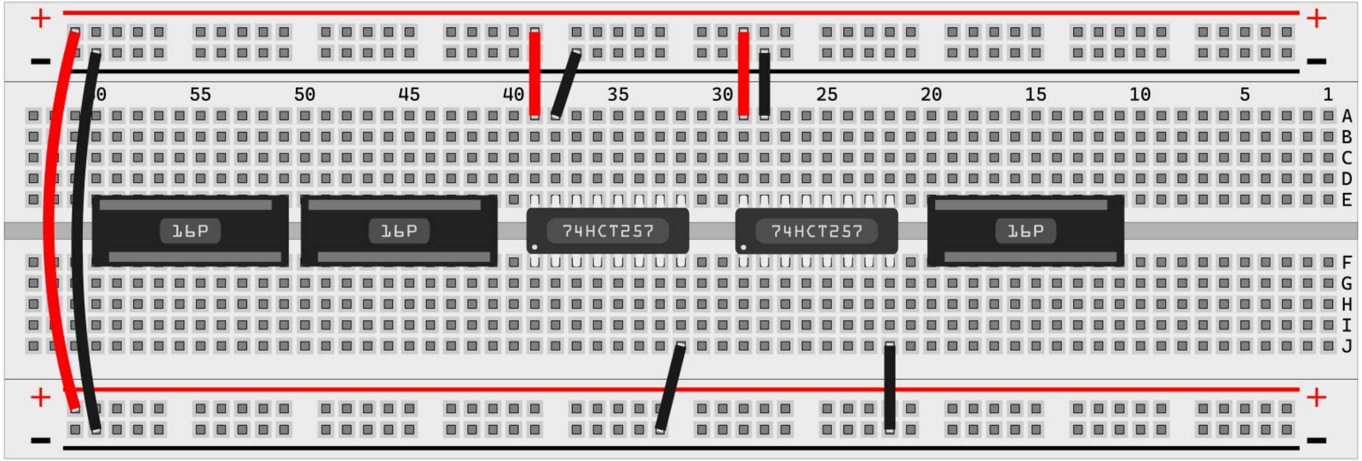
Using banking techniques		Accessible Memory:
32 Kw - Code Memory	=>	256 Banks - Code Memory
32 KB - Data Memory	=>	256 Banks - Data Memory
32 MB - Compact Flash (CF) Memory	=>	65,536 Blocks - CF Memory



CPU data path and control path must be visualized through LEDs.

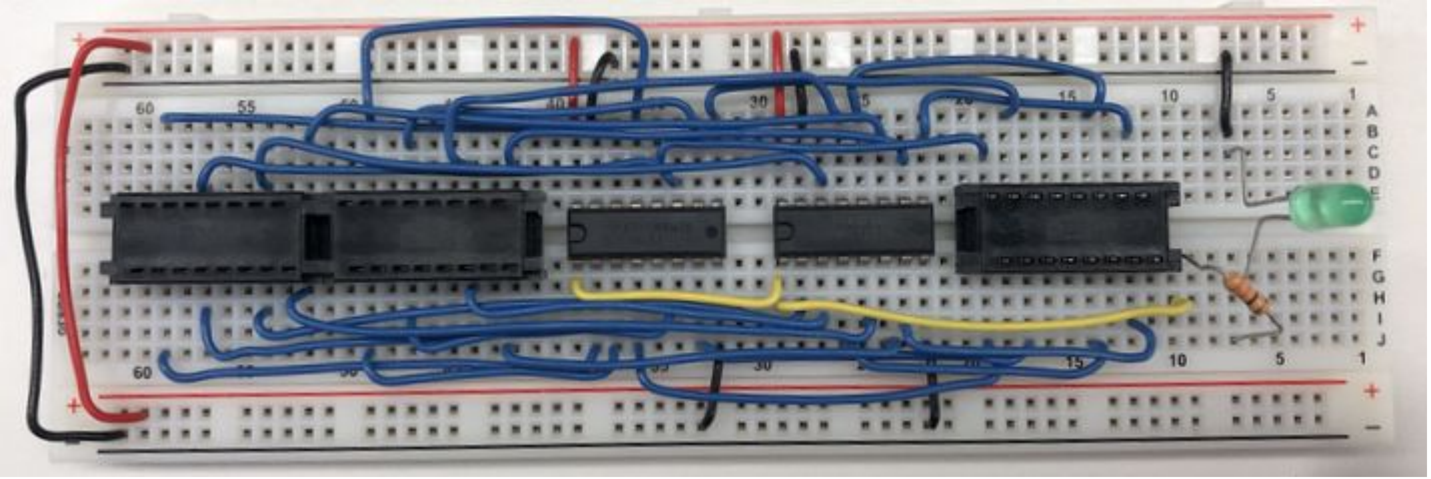
System Design

- After the logical schematic is finished, it must be translated into something that can be built on a breadboard.
- A rough component outline of each module is created before physical assembly can begin.



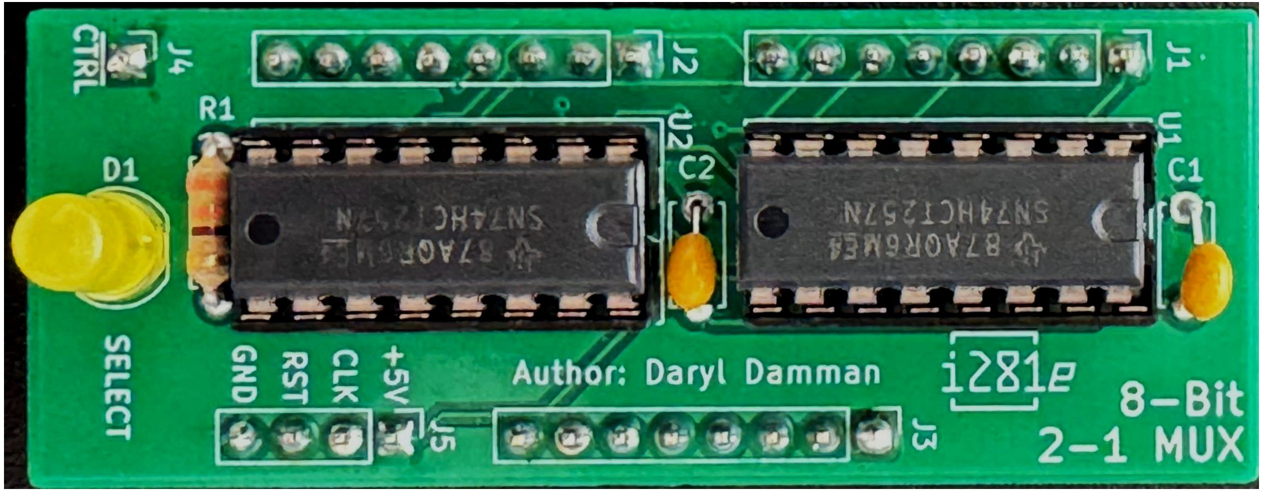
System Design

- Once a breadboard layout is finished, it can be physically implemented.
- Ribbon cable is used with the black connectors to transfer data across modules.



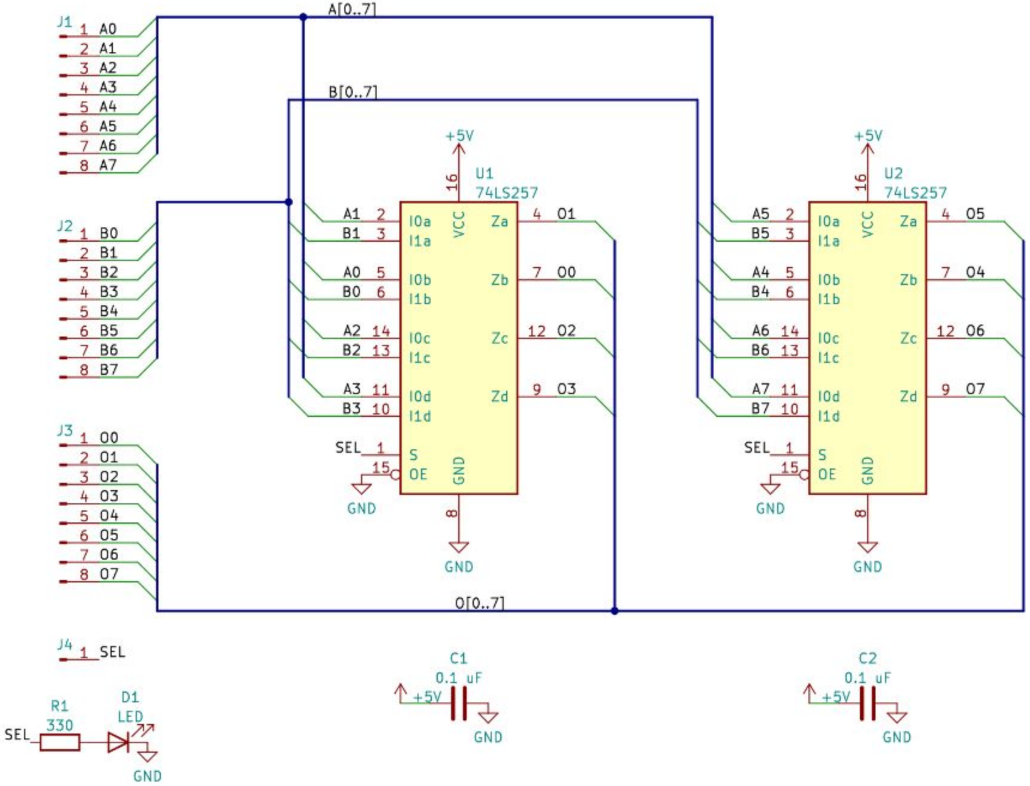
System Design

- PCB info?
- Mention how it connects to the Mainboard?



System Design

- Each module starts as a chip-level electrical schematic.
- Pictures is the schematic for the MUX

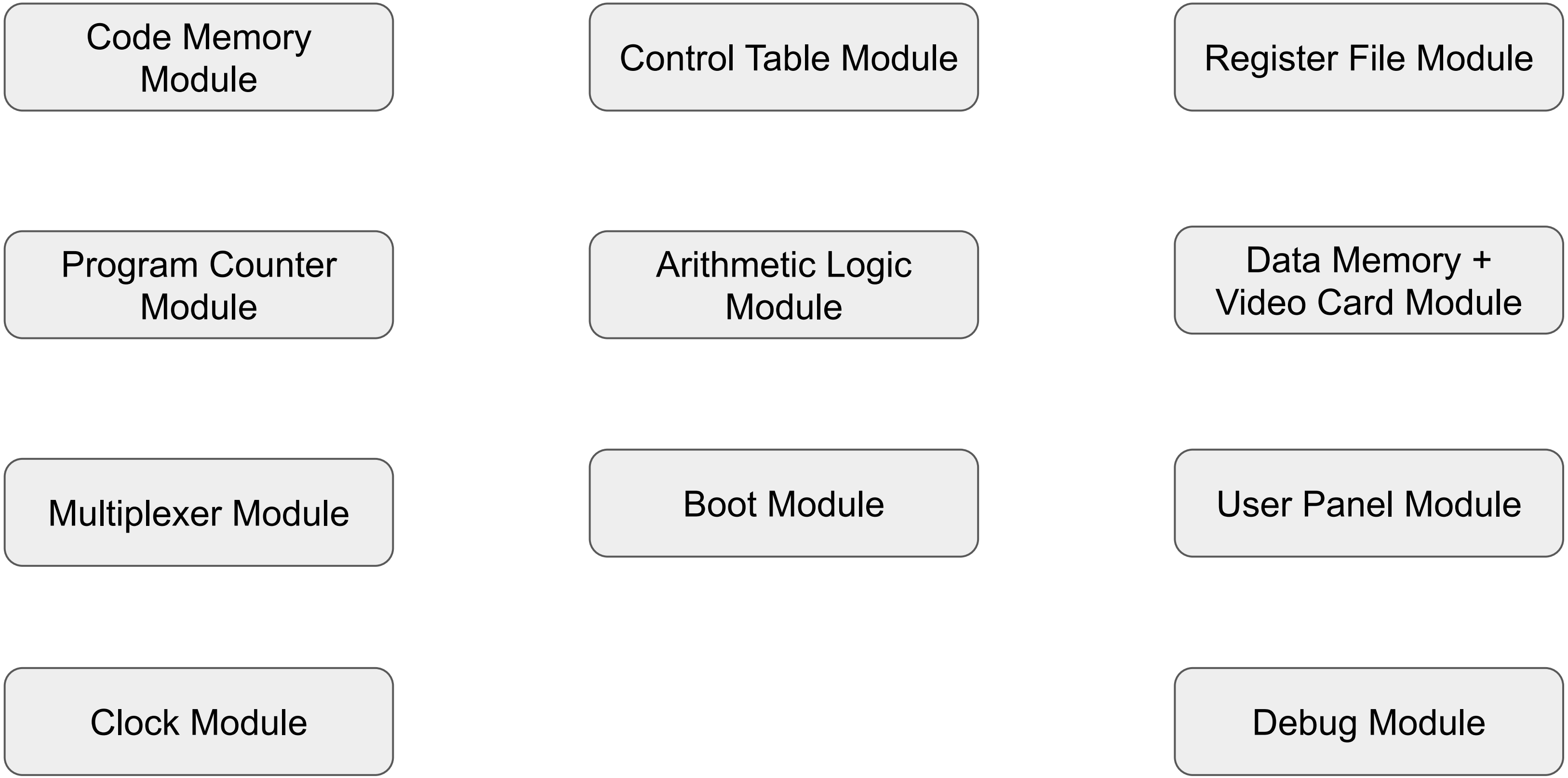


Unadded Opcodes (Blue)

Signed VS Unsigned

Signed	
BRE	BRanch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRG	BRanch if Greater
BRGE	BRanch if Greater than or Equal
BRL	BRanch if Less
BRLE	BRanch if Less than or Equal

Unsigned	
BRE	Ranch if Equal
BRZ	BRanch if Zero
BRNE	BRanch if Not Equal
BRNZ	BRanch if Not Zero
BRA	BRanch if Above
BRAE	BRanch if Above or Equal
BRB	BRanch if Below
BRBE	BRanch if Below or Equal



Code Memory Module

Control Table Module

Register File Module

Program Counter Module

Arithmetic Logic Module

Data Memory + Video Card Module

Multiplexer Module

Boot Module

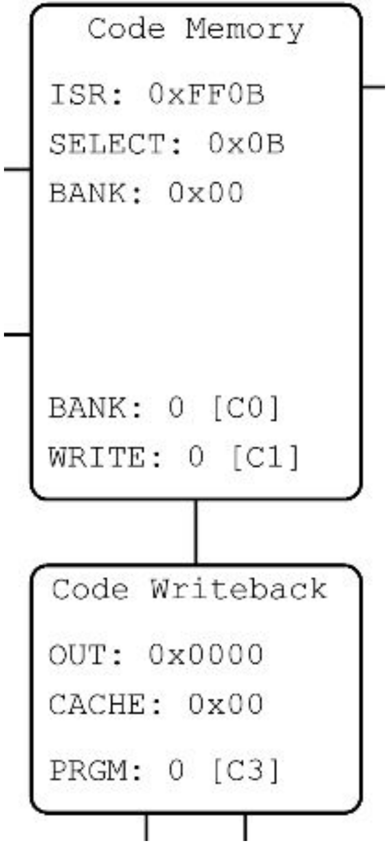
User Panel Module

Clock Module

Debug Module

System Modules

Code Memory



Code Memory

ISR: 0xFF0B

SELECT: 0x0B

BANK: 0x00

BANK: 0 [C0]

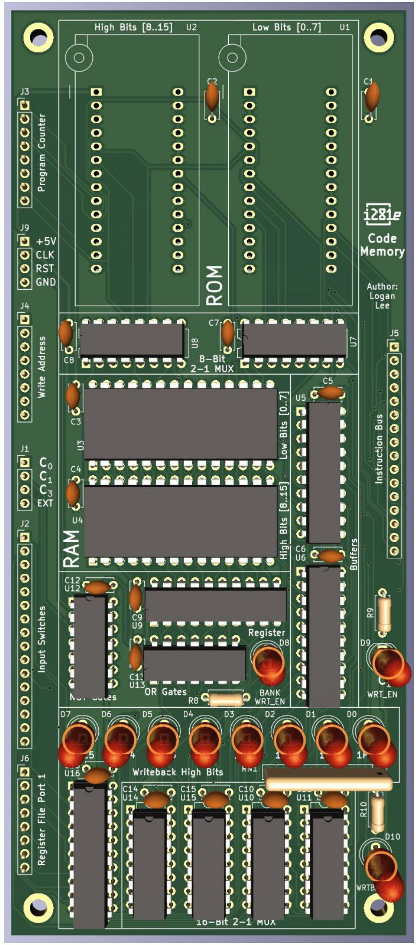
WRITE: 0 [C1]

Code Writeback

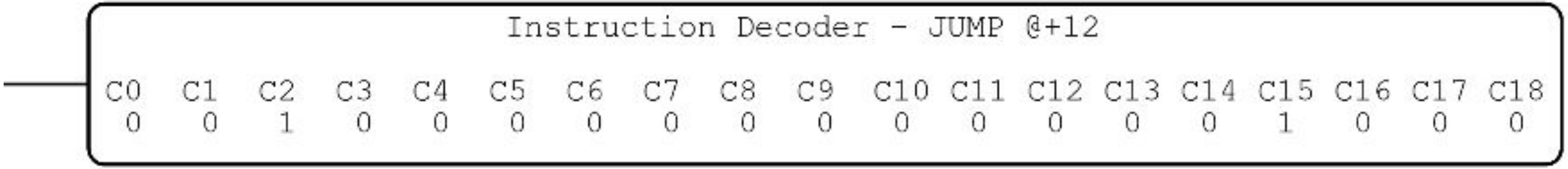
OUT: 0x0000

CACHE: 0x00

PRGM: 0 [C3]

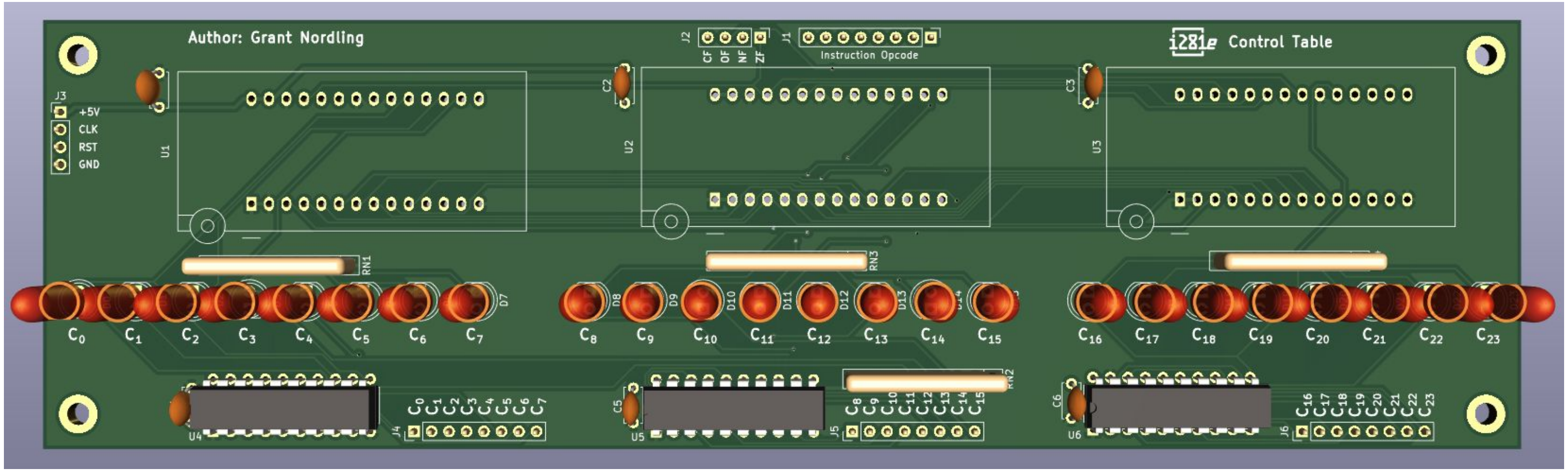


Control Table / Instruction Decoder

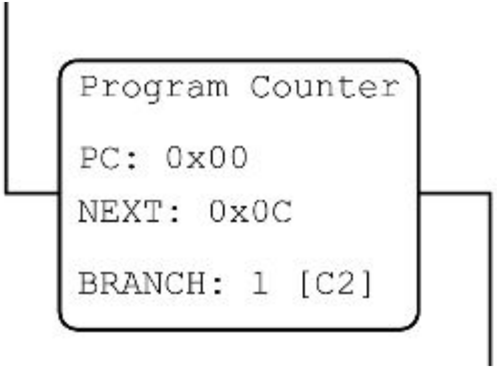


Instruction Decoder - JUMP @+12

C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0


Author: Grant Nordling
i281e Control Table
J2
CF
OF
NF
ZF
J1
Instruction Opcode
J3
+5V
CLK
RST
GND
U1
U2
U3
C2
C3
RN1
RN3
RN2
D7
D8
D9
D10
D11
D12
D13
D14
C_0 C_1 C_2 C_3 C_4 C_5 C_6 C_7
C_8 C_9 C_{10} C_{11} C_{12} C_{13} C_{14} C_{15}
C_{16} C_{17} C_{18} C_{19} C_{20} C_{21} C_{22} C_{23}
U4
J4
C5
U5
J5
C6
U6
J6

Program Counter

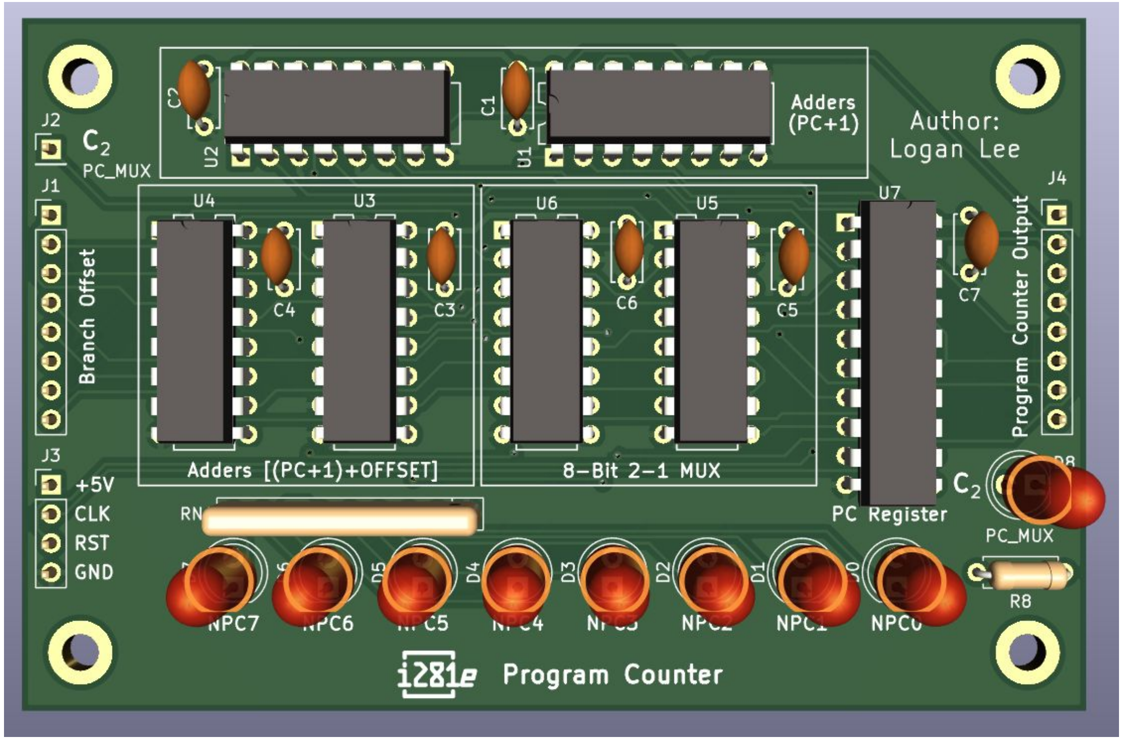


Program Counter

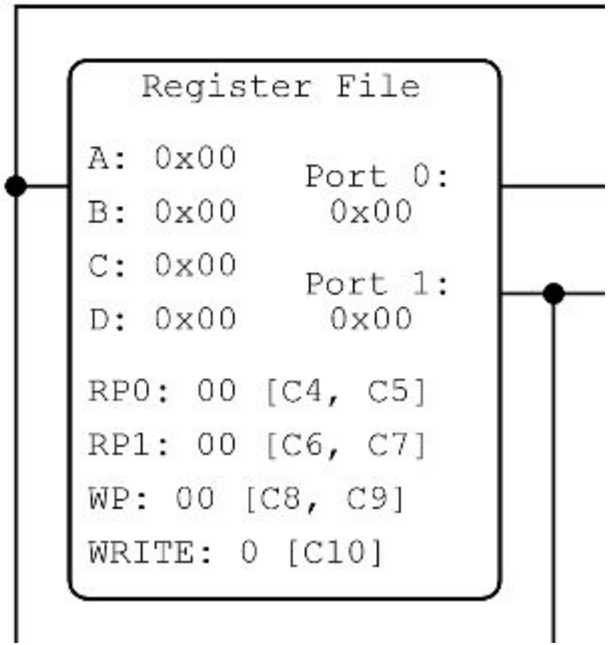
PC: 0x00

NEXT: 0x0C

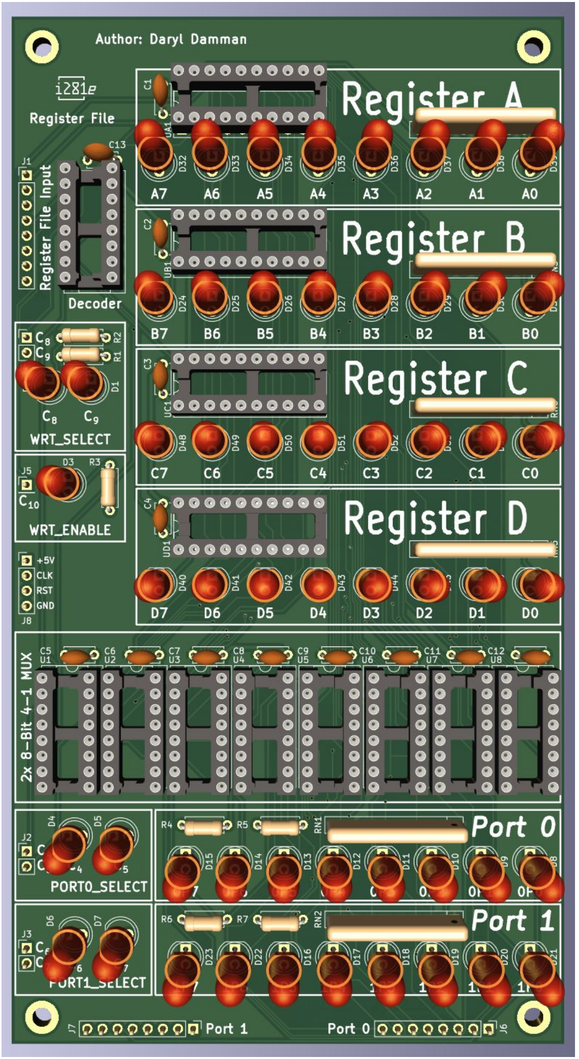
BRANCH: 1 [C2]



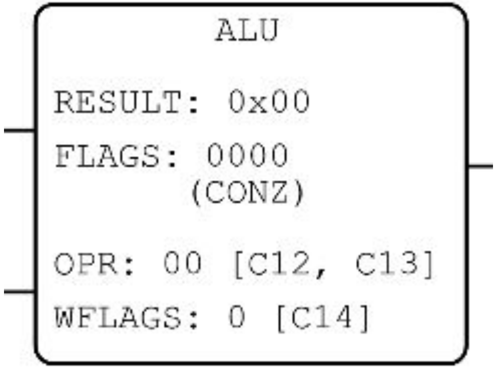
Register File



```
Register File

A: 0x00          Port 0:
B: 0x00           0x00
C: 0x00
                 Port 1:
D: 0x00           0x00

RP0: 00 [C4, C5]
RP1: 00 [C6, C7]
WP: 00 [C8, C9]
WRITE: 0 [C10]
```



ALU with Flags Register



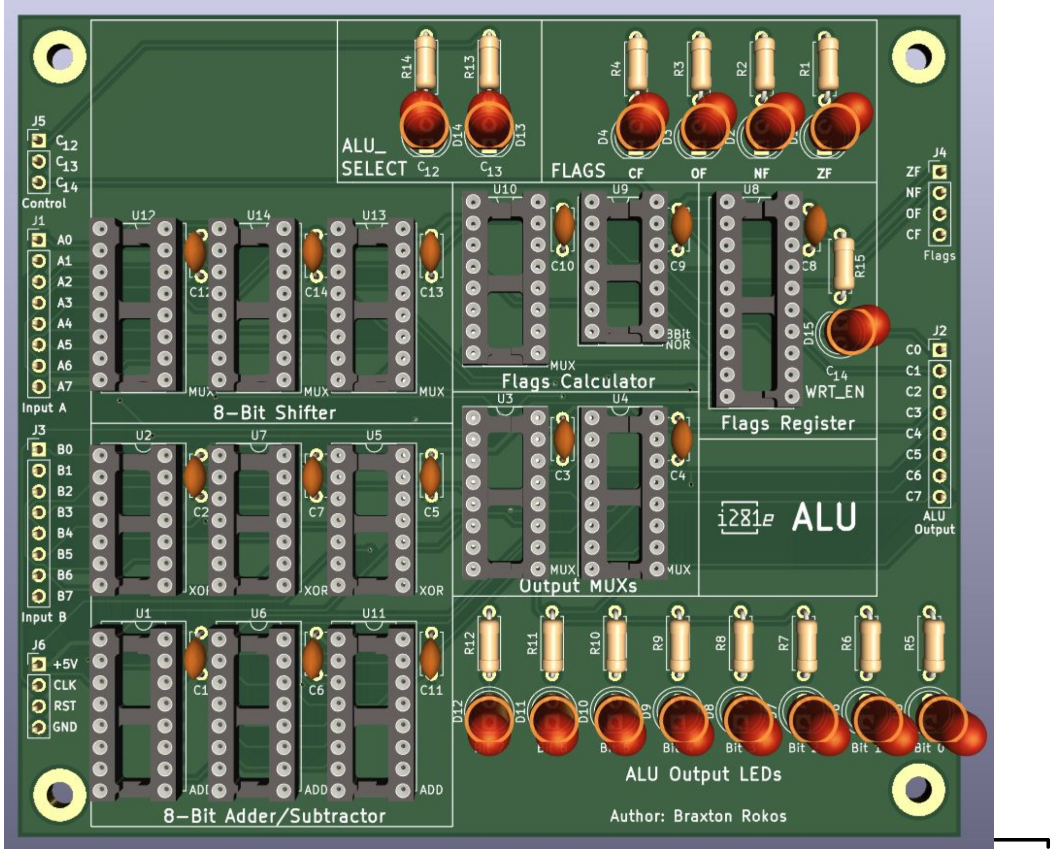
ALU

RESULT: 0x00

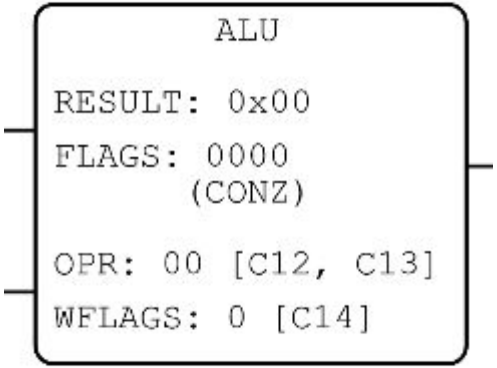
FLAGS: 0000
(CONZ)

OPR: 00 [C12, C13]

WFLAGS: 0 [C14]



ALU (NOR version)



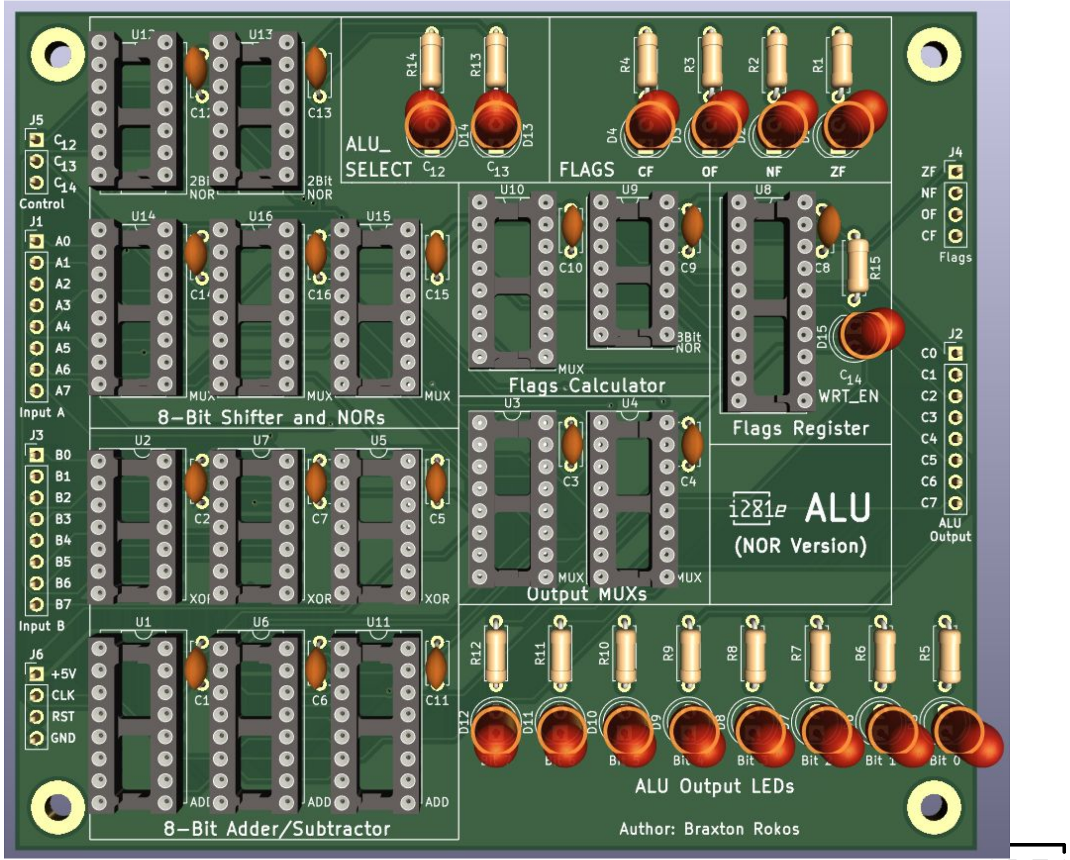
ALU

RESULT: 0x00

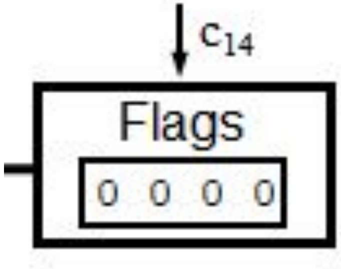
FLAGS: 0000
(CONZ)

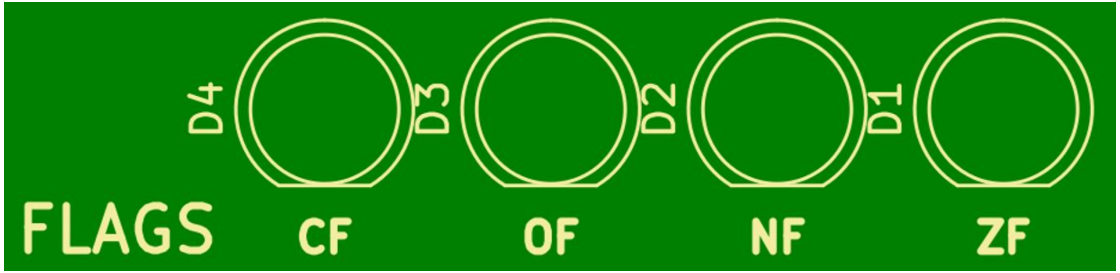
OPR: 00 [C12, C13]

WFLAGS: 0 [C14]

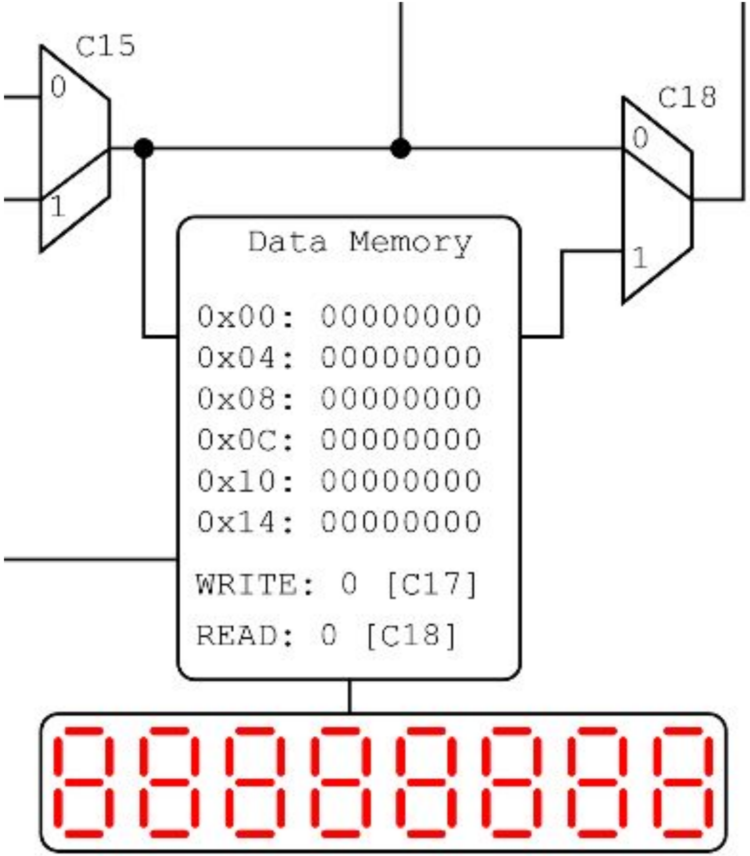


Flags Register

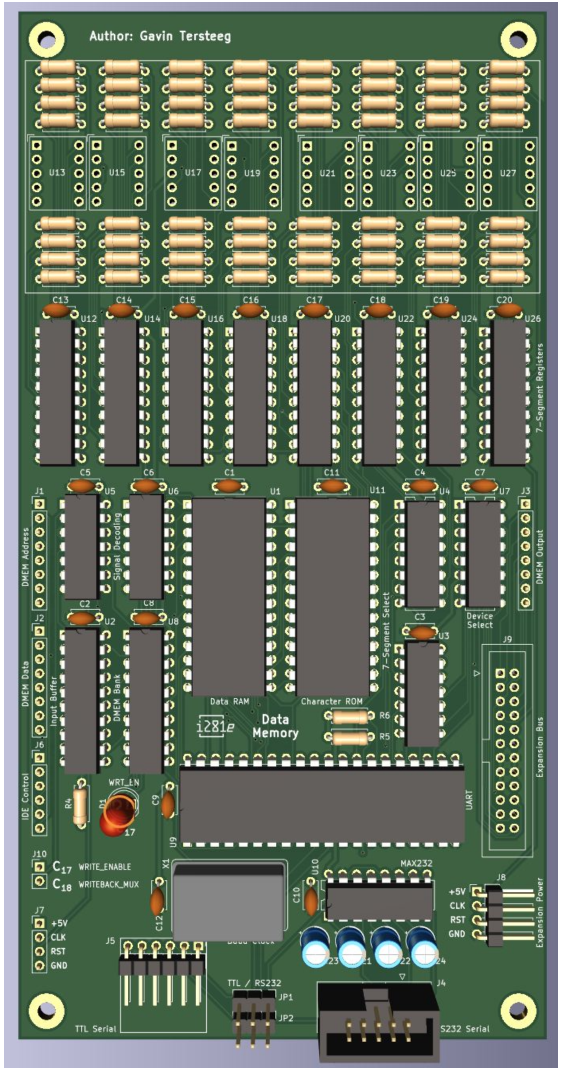




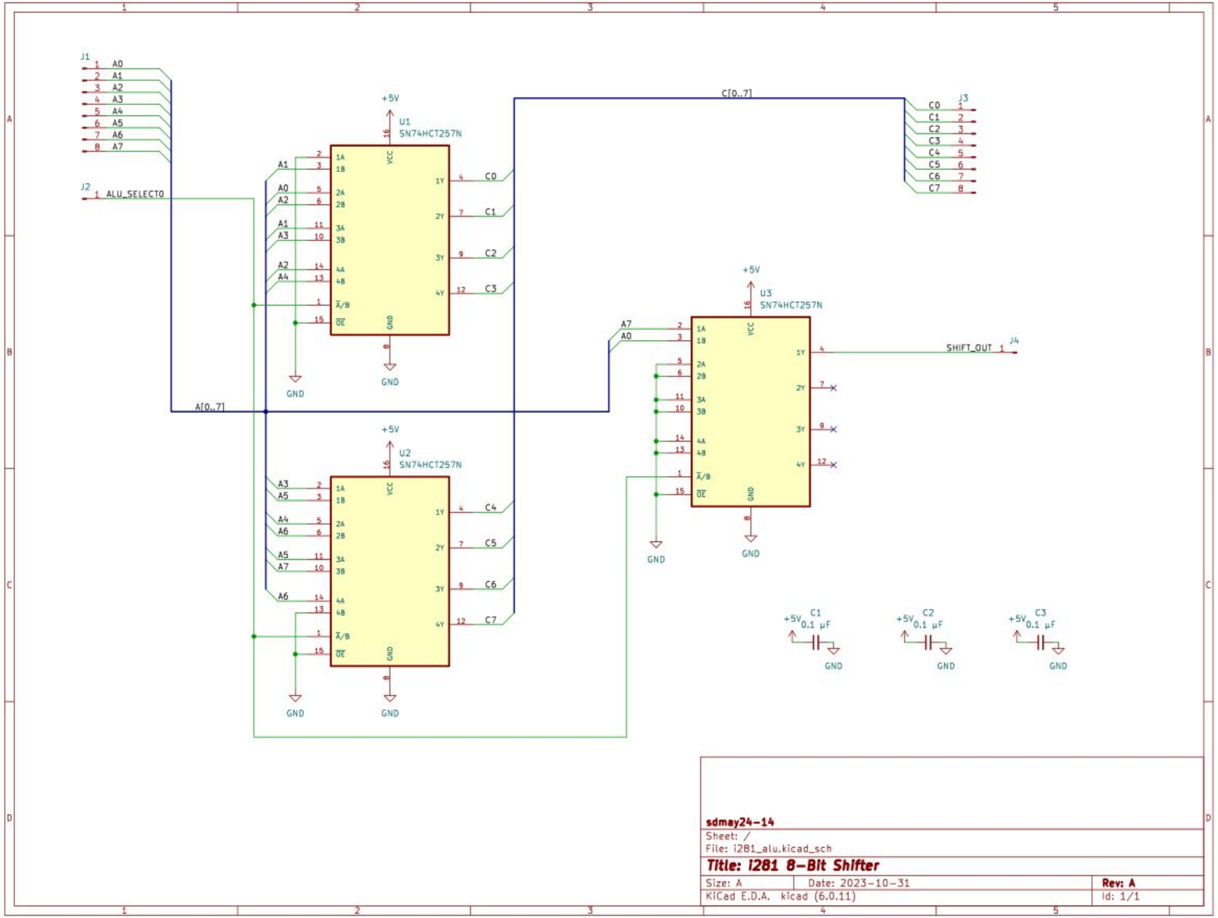
Data Memory and Video Card



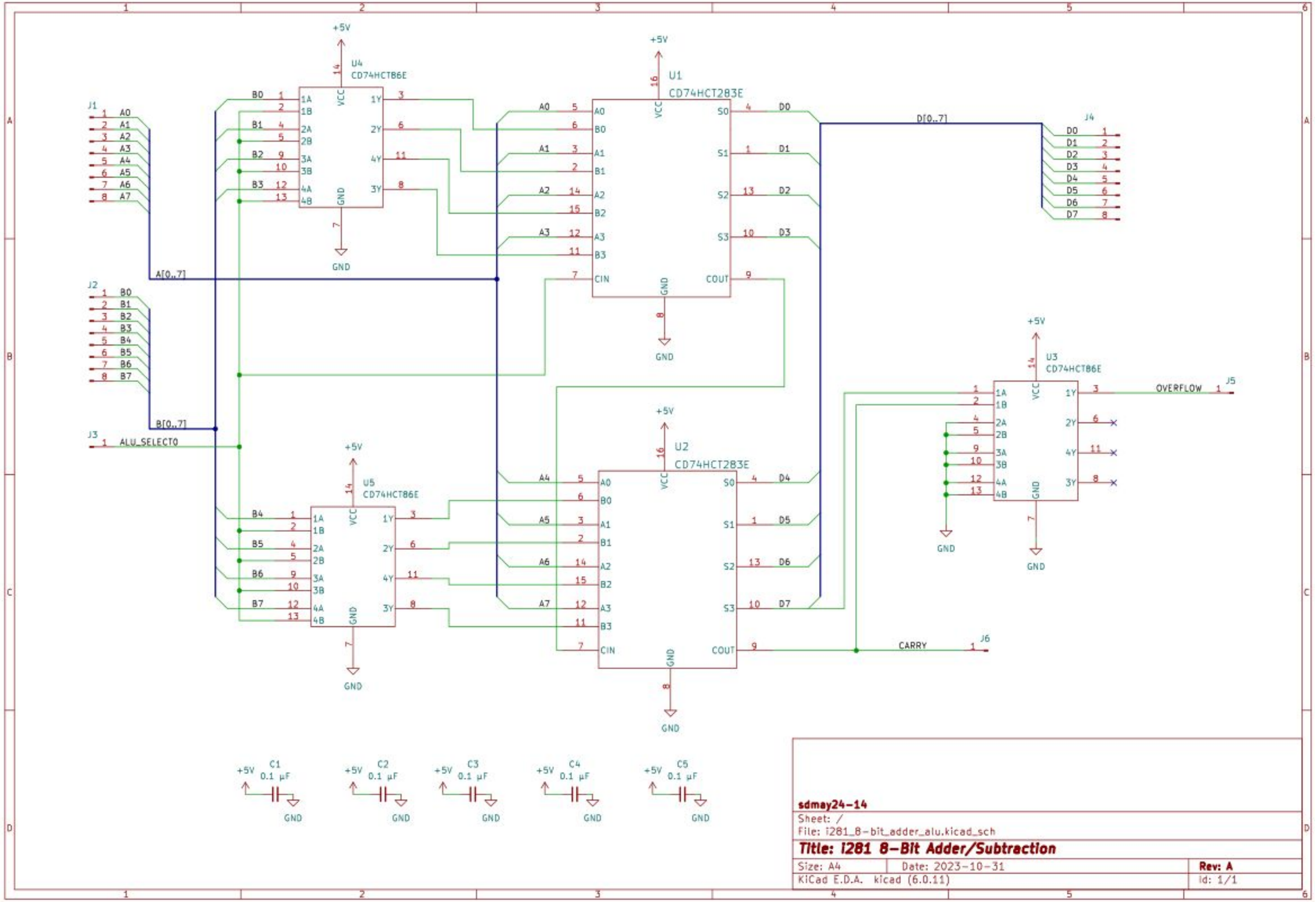
```
Data Memory

0x00: 00000000
0x04: 00000000
0x08: 00000000
0x0C: 00000000
0x10: 00000000
0x14: 00000000

WRITE: 0 [C17]
READ: 0 [C18]
```



ALU Schematic (2 of 3)

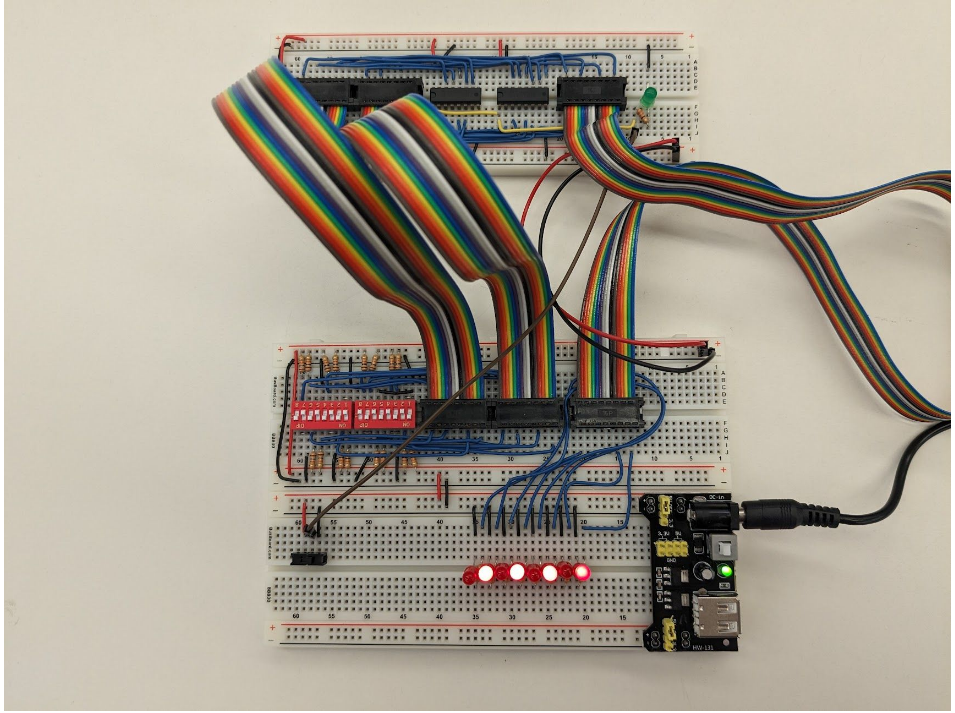


ALU Schematic (3 of 3)

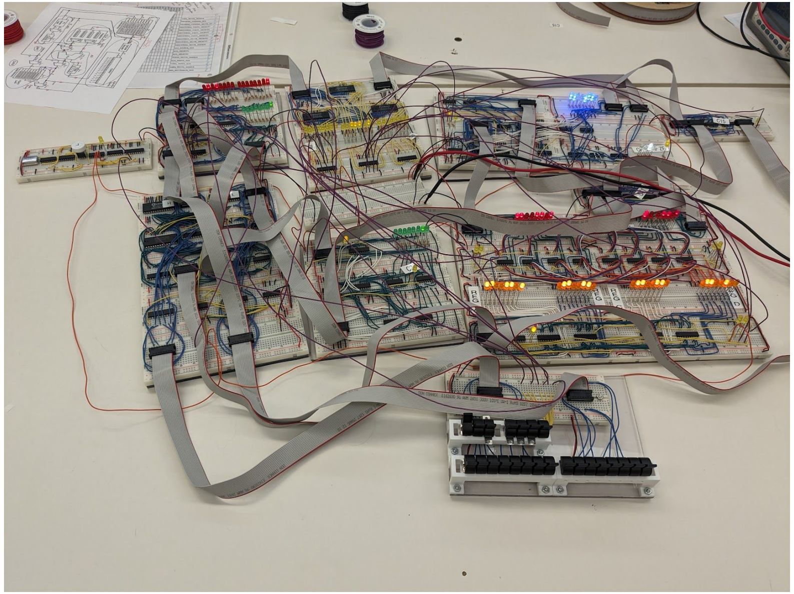


Prototyping

- Since this project is more implementation heavy, most of our first semester was focused on constructing our prototype.
- The project requirements call for the first version of the i281 CPU to be built on solderless breadboards. This allowed us to build each module and test as we progressed.
- When an issue with a module was found, it could easily be corrected, and it's KiCAD design schematics updated.



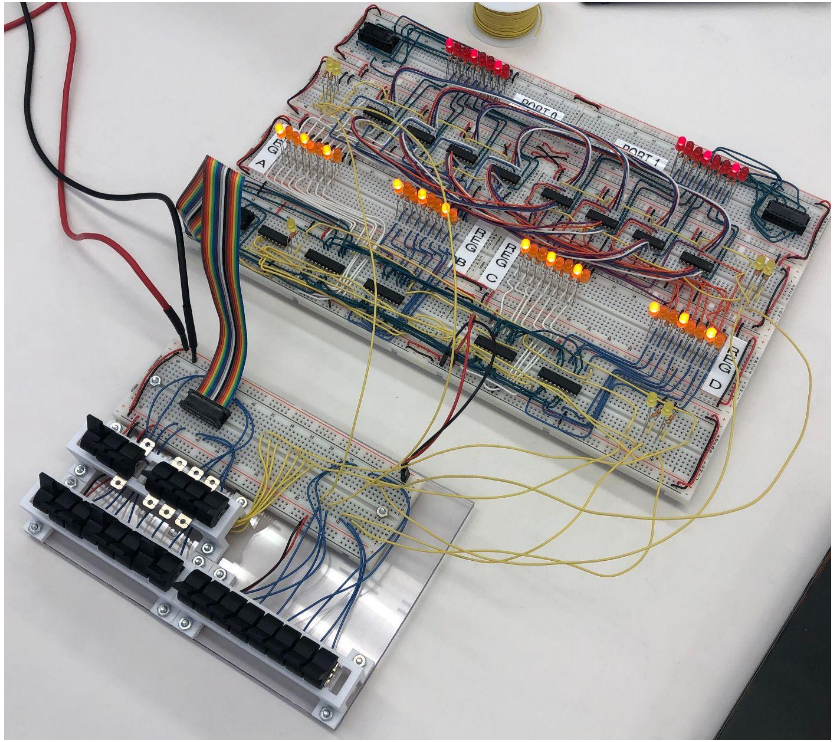
Prototype Implementations



- As the semester progressed, we implemented enough of the prototype modules that we were able to construct a minimally viable processor (MVP).
- The MVP lacks a number of features that will be present on the finished product, but it allows most of the critical modules to be integration tested using simple test programs.

Test Plan

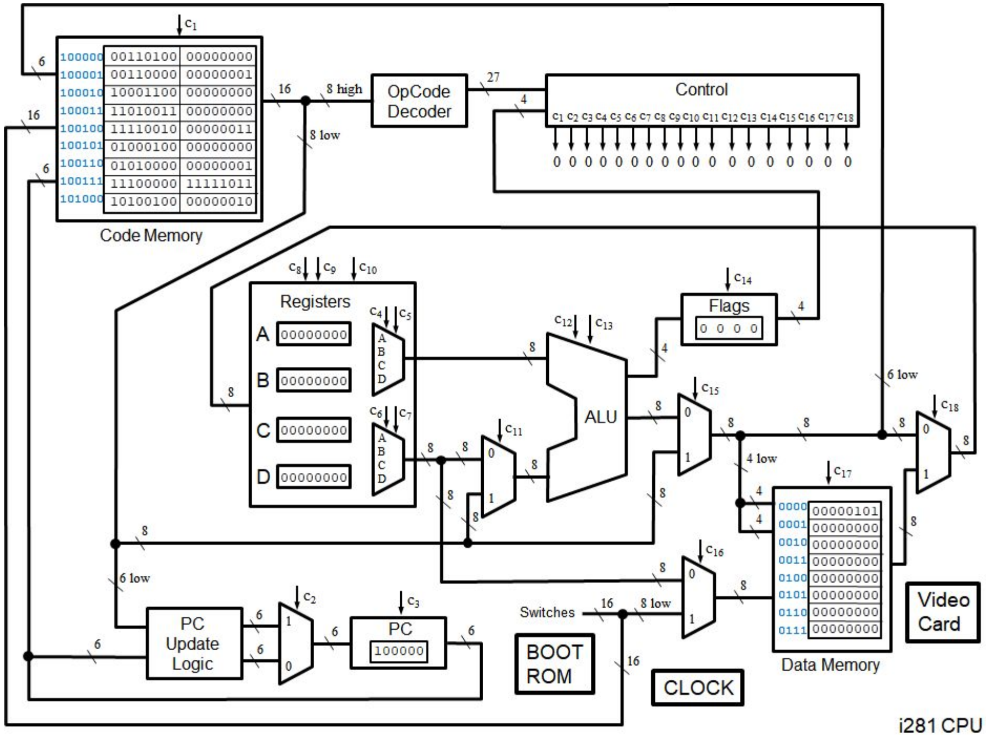
- **Acceptance Testing**
  - Testing electrical properties
    - Ex. short circuits
- **Integration Testing**
  - Entire system testing & running program
- **Interface Testing**
  - Interface panel designed with basic input and control switches



Breadboard Implementation

Design Complexity

- Breadboards and Wires
- Hardware Design/Constraints
  - Limited By Breadboards
  - Wiring Complexity
  - Educational Emphasis
- Obtainable Parts
  - Available parts



Code Memory

100000	00110100	00000000
100001	00110000	00000001
100010	10001100	00000000
100011	11010011	00000000
100100	11110010	00000011
100101	01000100	00000000
100110	01010000	00000001
100111	11100000	11111011
101000	10100100	00000010

Data Memory

0000	00000101
0001	00000000
0010	00000000
0011	00000000
0100	00000000
0101	00000000
0110	00000000
0111	00000000

i281 CPU

Design Iterations

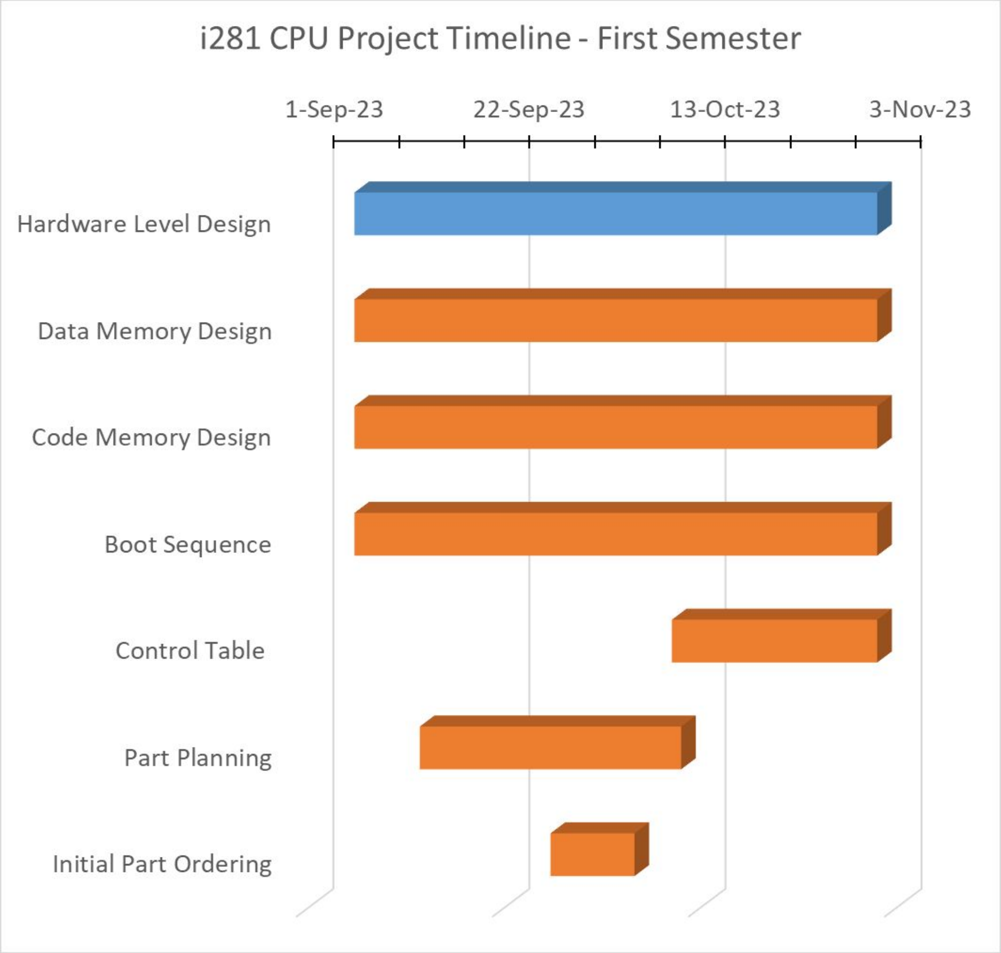
- i281 CPU Simulation and FPGA Design
- FPGA Design + Our Design Decisions
- Fixed Errors From Debugging & Redesigns

i281e

Project Schedule
First Semester

- **First Metric**
  - Initial Designs

- **First Milestone**
  - Hardware Level Design

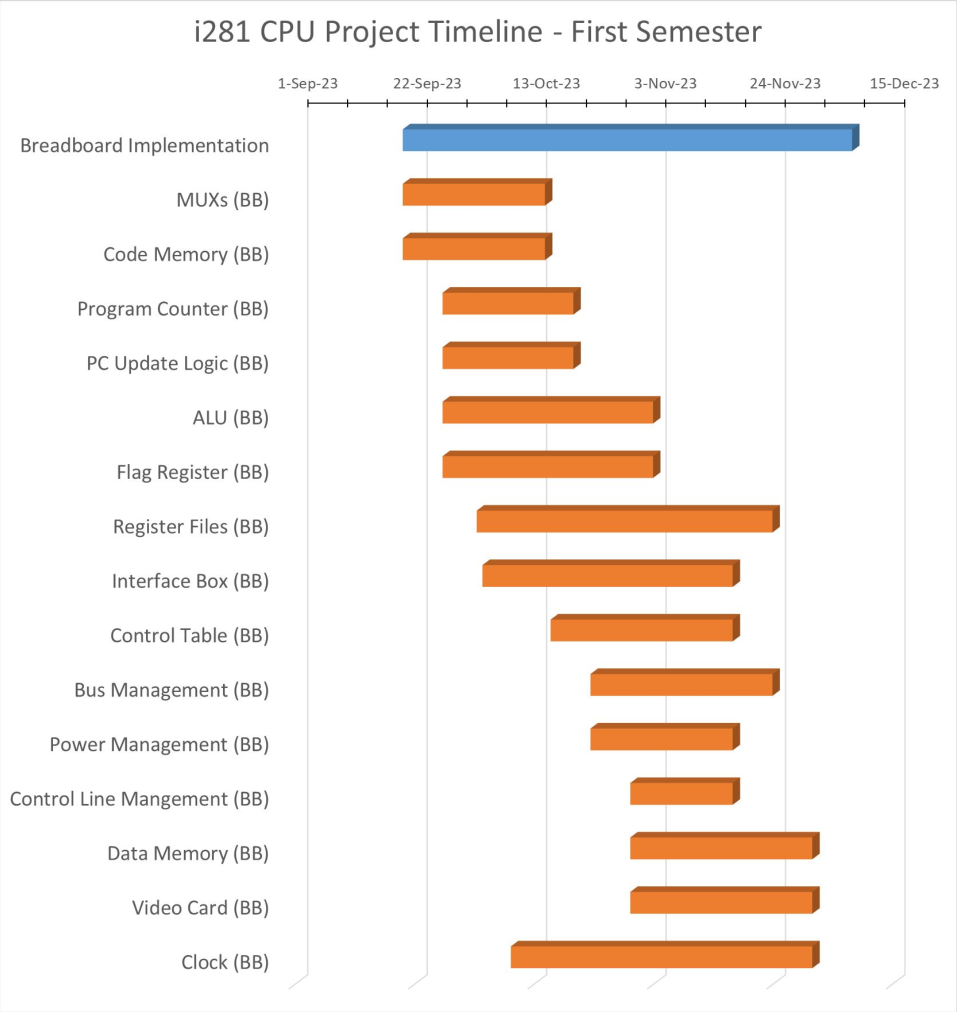


i281e

Project Schedule
First Semester

- **Second Metric**
    - Breadboards (BB)

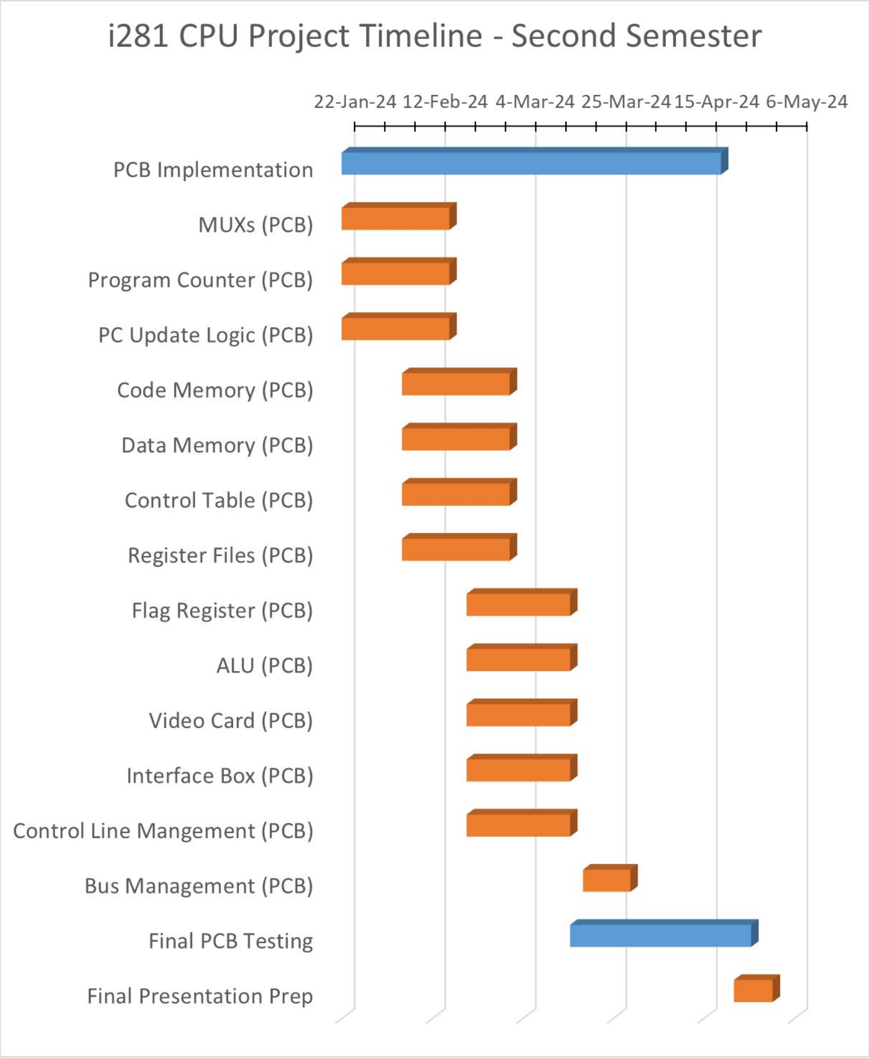
- **Second Milestone**
    - Breadboard Implementation



Project Schedule
Second Semester

- **Metric**
  - Printed Circuit Board (PCB)

- **Milestones**
  - PCB Implementation
  - Final PCB Testing



THE END++