i281CPU

Design Document

Senior Design CPR E 491

Client: Professor Alexander Stoytchev

Team 14 – Hardware Implementation of i281 Processor

Team Members:

Daryl Damman	Team & Documentation Lead and Client Point of Contact
Logan Lee	Scheduling and Control Management
Grant Nordling	Quality Control and Parts Manager
Braxton Rokos	PCB Design & Routing Lead and Testing
Gavin Tersteeg	Technical Lead, Quality Control, and Project Assembly Lead

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

Contact: sdmay24-14@iastate.edu

Revised: 2023-12-03

Executive Summary

Development Standards & Practices Used

Standards and practices of the i281 CPU project establish protocols and procedures that we must follow to make the i281 CPU as close as possible to industry standards. By including them, we can ensure that the design is easy for industry leaders and professionals to understand our design along with less experienced individuals such as students. Some of the standards and practices assist with mitigating potential error by using the practices described by the Institute of Electrical and Electronics Engineers (IEEE).

- IEEE 162-1963
- IEEE 370-2020
- IEEE 2716-2022
- IEEE 696-1983

Summary of Requirements

- The primary CPU components must be explainable to CPR E 281 students;
- Memory and control flow must be visualized through LEDs and other indicators;
- CPU execution must allow for single-instruction or continuous execution;
- CPU must allow writing custom programs via switches, loading example programs, and playing PONG;
- CPU design must be as close as possible to the original Verilog design.

Applicable Courses from Iowa State University Curriculum

Major	Course Number	Course Name
EE	201	Electric Circuits
	230	Electronic Circuits and Systems
	330	Integrated Electronics
	333	Electronic Systems Design
CPR E	281	Digital Logic
	381	Computer Organization and Assembly Level Programming
S E	329	Software Project Management

Table 1 - Applicable courses for the i281 Hardware Implementation project

Acquired Skills

The team has learned many new skills while designing the project and developing the breadboard prototype. Members have learned how to wire large-scale breadboards, draw up schematics in KiCad, and hold technical reviews between our client and outside parties.

Table of Contents

List of Figures

List of Tables

Table of Acronyms

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

Table 2 - Acronym Definition List

1. Team

1.1 Team Members

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

Table 3 - Member List with Majors

1.2 Required Skill Sets

Skill Set	Rationale
PCB Design	The project's second phase is centered around developing PCBs.
Layout	PCB layout is an important factor in designing and creating PCBs.
Routing	Routing is a difficult task that is essential to the success of a PCB and determines if the board can be printed.
Assembly	The first phase of the project includes physically building the circuits using breadboard and potentially other materials to create the product.
Digital Logic	The project is centered around constructing a processor using topics almost exclusively from the CPR E Digital Logic class.
System Testing	Both phases of the project include testing our designs against every possibility to ensure proper function.
Assembly Programming	The i281 as a custom instruction set architecture (ISA) and particular programs must be written for functional tests.

Table 4 - Required Skill Sets for the Project

1.3 Team Skill Sets

Member	Usable Skill Sets
Logan Lee	Circuit design and digital logic simulation.
Grant Nordling	PCB and circuit testing, part ordering, enclosure design and wiring
Braxton Rokos	Circuit design, PCB layout and testing, and circuit simulations.
Gavin Tersteeg	PCB, digital logic, and homebrew computer design.
Daryl Damman	Project management and software/assembly programming

Table 5 - Skill Sets Brought in by Team Members

1.4 Project Management Style

The chosen Project Management Style is Agile; however, it is loosely adopted. There exist certain actions and procedures that we perform that are more akin to Waterfall. With the scope of this project and course, a full implementation of the Agile methodology was deemed too burdening. This was further amplified with the class and work schedule of each member and the client. Our notable differences from Agile are as follows:

1. There are no retrospectives held after sprints (or milestones, in this case).
2. Milestones don't adhere to a particular schedule and are neither weekly nor monthly.
3. Daily standups are not held due to time constraints of all members.

Through our project management style, tasks are identified individually and grouped into appropriate milestones. These milestones are observed, tracked, and held against our total project plan.

Meetings are held weekly on Wednesday night, with the adoption of including a work session on Mondays. During our weekly meeting, we will review these tasks and determine the appropriate steps forward. At the end of our weekly meeting, we begin our client meeting to discuss current project progress, hold technical discussions, and affirm project requirements.

1.4.1 Results of Project Management Style Adoption

At the end of the first semester, the Agile style slowly faded out to prioritize finishing particular tasks and preparing for faculty panel review. Milestones are still tracked and monitored but tasks were not recorded as accurately. The primary project management style will be more thoroughly enforced at the start of the second semester.

1.4.2 Project Tracking

For additional information regarding how we track stories and communication, please refer to the Project Management section in Section 3, Project Plan.

1.5 Initial Project Management Roles

Member	Usable Skill Sets
Logan Lee	Schedule Consultant and Testing Co-Lead
Grant Nordling	Project Assembly Lead and Quality Control Co-Lead
Braxton Rokos	PCB Design & Routing Lead, Prototyping Lead, and Testing Co-lead
Gavin Tersteeg	Quality Control Co-Lead and Testing Co-Lead
Daryl Damman	Team Lead and Client Point of Contact

Table 6 - Initial project management roles for 491

1.6 Evolved Project Management Roles

Member	Usable Skill Sets
Logan Lee	Schedule Consultant and Control Management
Grant Nordling	Quality Control Co-Lead and Parts Manager
Braxton Rokos	PCB Design & Routing Lead and Testing Co-Lead
Gavin Tersteeg	Technical Lead, Quality Control Co-Lead, and Project Assembly Lead
Daryl Damman	Team Lead, Documentation Lead, and Client Point of Contact

Table 7 - Evolved project management roles at the end of 491

2. Requirements

2.1 Problem Statement

Currently, the students in the digital logic course do not have a physical design of an i281 CPU they can use to apply their learning. This project aims to make a raw physical design of this CPU on breadboards that students can interact and learn with and a slick PCB physical design with LEDs to show the processes as they happen.

2.2 Design Requirements

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

2.2.1 Functional Requirements

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

Table 8 - Functional Requirements

2.2.2 Qualitative/Subjective Requirements

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

SR-10	CPU must be readable from one viewing direction
SR-11	CPU must be fully labeled
SR-12	CPU design must be as close as possible to the original Verilog design
SR-13	Bus entry and exit need to be clearly labeled with direction and connection type
SR-14	LEDs and related visualizers must be color-coded
SR-15	Singular direction (northern indicator) must be standardized and rigorously followed for all CPU components and visualizers
SR-16	A standard cable and LED color code must be used to distinguish

Table 9 - Qualitative Requirements

2.2.3 Quantitative Requirements

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

Table 10 - Quantitative Requirements

2.3 Design Constraints

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

Table 11 - Design constraints

2.4 Engineering Standards

- IEEE 162-1963
  - IEEE 162-1963 describes the standard definitions and terms for digital computers. As the project is intended to be educational, using the appropriate terminology for digital computing and related components is paramount for a comprehensive curriculum. [2]
- IEEE 370-2020
  - IEEE 370-2020 describes a standard for predicting electrical characteristics on printed circuit boards and other related interconnects at frequencies up to 50 GHz.

This is relevant to our project because we will need to handle signals running at up to 1 MHz on our printed circuit boards for the final product. [3]

- IEEE 2716-2022
    - IEEE 2716-2022 provides a guide for characterizing the effectiveness of printed circuit board level shielding. In our project, we will have a dozen or so PCBs all connected with discrete cables. We will need to take shielding into account, so we don't encounter noise-related problems. [4]
- IEEE 696-1983
    - IEEE 696-1983 describes a computer bus architecture for 8-bit computers running at TTL logic levels. Knowledge of how to avoid signal noise, arbitrate device access, and distribute power to all subsystems will come into use for our own project. [5]

2.5 Intended Users and Uses

The intended users of the i281 CPU project are Professor Stoytchev and college students who take his courses. Professor Stoytchev may create a class dedicated to recreating portions of the CPU design using breadboards. The students will connect their designs to the i281 CPU using the ribbon cables to test their designs.

3. Project Plan

3.1 Project Management/Tracking Procedures

3.1.1 Project Management

Our project is based around an Agile methodology but does not fully implement it. Tasks are identified individually and grouped into appropriate milestones. During our weekly meeting, we will review these tasks and determine the appropriate steps forward. Our further differences from Agile are as follows:

- There are no retrospectives held after sprints (or milestones, in this case).
- Daily standups are not held due to time constraints of all members.

3.1.2 Tracking Procedures

Provided by the university, Gitlab has been used throughout this semester and will continue to be used into the final semester. Gitlab is being used to track tasks and milestones of the project, as outlined in the project plan. Email and Discord are also being used to communicate and provide a written record that will be backed up into Gitlab when appropriate, as outlined in the Team Contract.

3.2 Project Proposed Milestones, Metrics, and Evaluation Criteria

From our project, there are two types of milestones we have: overarching and requirements. Overarching milestones are those tracked throughout the project life cycle as the primary milestones. Requirement milestones will track additional tasks that are done pseudo-independently of the project which satisfy the requirements of the project.

3.2.1 Overarching Milestones

Overarching Milestone	Description
Initial Complex Designs	Certain CPU components cannot be directly translated from FPGA simulation to hardware. As such, these components will need technical discussions with the client and members alike.
Breadboard Implementation	Per the client's requirement, the CPU will be implemented on breadboard before constructing a PCB design/set-up. This will be the proof-of-concept using the converted FPGA simulation designs.
PCB Implementation	Once the breadboard implementation is complete, PCBs must be designed and ordered. These are required to be hot-swappable with breadboards.

Finalized PCB Testing	While PCB testing will be performed throughout implementation, dedicated time has been reserved for ironing out all potential kinks in the PCB system.

Table 12 - Overarching Milestones

3.2.1 Requirements Milestones

Requirements Milestone	Description
Visualization Standardization	Design elements of the breadboard and PCB visualizations need to be standardized. These include: • LEDs • Direction This must be completed before the Breadboard Implementation milestone is completed
Clock Speed	With the requirement of the CPU reaching 1MHz clock speed, testing must be performed at incrementally increasing clock speeds to ensure the system is capable of handling speeds up to and including 1MHz.
Technical Binder	The client has requested a complete copy of *all* documentation for the entire project in a physical binder as an additional deliverable.

Table 13 - Requirements Milestones

3.3 Task Decomposition

The tasks for our project have been broken down into respective overarching milestones. Additional tasks related to requirement milestones are also listed but not documented on our Gantt charts as they are tracked independently of the hardware implementation process.

3.3.1 Initial Complex Design

Task #	Task Description	Description	Dependency
1-1	Data Memory Design	Design for the DMEM module	
1-2	Code Memory Design	Design for the CMEM module	
1-3	Boot Sequence	Process for booting the i281 CPU	1-2
1-4	Control Table	Developing the control lookup table for control lines	
1-5	MUX Design	Implementing the first ever bus multiplexer	

Table 14 - Initial Complex Design Tasks List

3.3.2 Breadboard Implementation

The following components will require one or more schematics to be drawn and implementations of the schematics on breadboard to be completed.

Task #	Task Description	Dependency
2-1	MUXs	1-5
2-2	Program Counter (PC)	
2-3	PC Update Logic	2-2
2-4	ALU	
2-5	Control Table	
2-6	Register Files	
2-7	Flag Register	
2-8	Video Card	2-10
2-9	Code Memory	
2-10	Data Memory	
2-11	Interface Box	
2-12	Bus Management	
2-13	Power Management	
2-14	Control Line Management	
2-15	Clock	
2-15	Breadboard System Assembly	All other breadboard tasks

Table 15 - Breadboard Implementation Tasks List

3.3.2 PCB Implementation

The following components will require one or more PCBs to be routed and ordered for physical production. These tasks are 1:1 from the Breadboard Implementation milestone.

Task #	Task Description	Dependency
3-1	MUXs	2-1
3-2	Program Counter (PC)	2-2
3-3	PC Update Logic	2-3
3-4	ALU	2-4
3-5	Control Table	2-5
3-6	Register Files	2-6
3-7	Flag Register	2-7
3-8	Video Card	2-8
3-9	Code Memory	2-9
3-10	Data Memory	2-10
3-11	Interface Box	2-11
3-12	Bus Management	2-12
3-13	Power Management	2-13
3-14	Control Line Management	2-14
3-15	Clock	2-15 and 6-5
3-16	PCB System Assembly	All other PCB tasks

Table 16 - PCB Implementation Tasks List

3.3.4 Final PCB Testing

Task #	Task Description	Description	Dependency
4-1	Full System Test	Once all PCBs have been assembled and individually tested, all components will be brought together to test as a whole system	PCB Milestone Completed
4-2	PONG System Test	PONG must run on the CPU as required by the client. As such, a PONG test must be performed on the PCB system.	
4-3	Integrity Test	With the PCB system enclosed, the system should be run as if it would be used in a laboratory environment to check for faults in assembly.	

4-4	Program Test	Custom programs will be written to test the full functionality of the PCB system.	

Table 17 - Final PCB Testing Tasks List

3.3.5 Visualization Standardization

Task #	Task Description	Description	Dependency
5-1	LED Standardization	Standardize which LEDs are used for what purpose and location.	None
5-2	MSB Standardization	Standardize if the Most Significant Bit (MSB) is specifically one color of LED or not.	
5-3	Board Direction	Standardize how the boards should be aligned	
5-4	Power Regulation	Power needs to be from one or more sources across either implemented system. This power source must be standardized.	

Table 18 - Visualization Standardization Tasks List

3.3.6 Clock Speed

Task #	Task Description	Description	Dependency
6-1	Clock Testing	Using the Arduino Nano Every, use a C program to test different clock speeds on breadboard designs to find appropriate clock speeds	
6-2	Clock Speed 0.25MHz	Get the Clock Speed on the chips up to or exceeding 0.25MHz	
6-3	Clock Speed 0.50MHz	Get the Clock Speed on the chips up to or exceeding 0.5MHz	6-2
6-4	Clock Speed 1MHz	Get the Clock Speed on the chips up to or exceeding 1MHz	6-3
6-5	Full System Clock Check	Using the Arduino Nano Every again, check if the clock speeds theorized are appropriate for the final design.	6-1
6-6	Manual Clock Control	The CPU design must accept a single button press to perform a cycle on the CPU.	

Table 19 - Clock Speed Tasks List

3.3.7 Technical Binder

There are no specific tasks assigned to the technical binder but rather a continuous effort throughout the semesters. Each time a part is used in a design, either breadboard or PCB, the first page of the datasheet must be included in the binder. Additional documents:

- Meeting Notes (Client, team, etc.)
- Schematics and PCB routing
- Diagrams and scratch notes
- Design document
- Notable presentations outside of lightning talks
- Bills of Materials
- Any other notable piece of documentation

3.4 Project Timeline/Schedule

The first set of deliverables we have been working on since the start of our project is working out the hardware design based on the previous computer models. Our split Gantt chart includes this project as “hardware level design” and subtask underneath. For this deliverable, we are working with our client to understand what functionality is needed in the CPU and work through brainstorming ideas to implement. This information is required to complete the critical components' physical breadboard and PCB designs. The focus of this design stage is the DMEM, CMEM, the boot sequence of the CPU, and any changes necessary for hardware-level functionality that differ from the software version.

Our second deliverable is a breadboard implementation of the i281 CPU. This will be where we spend most of our time since we must work out any problems with our systems here. We have started building systems for the breadboard and ordering more parts to use in our breadboard CPU. The step of our project is included in our Gantt chart as “Breadboard Implementation” sub-tasks are marked with “(BB).”

The last deliverable we will work on for this project is the PCB implementation of the i281 CPU. This will be the final form of the CPU we work on throughout the project. To prepare for this section of our project, hardware schematics will be created and finalized. Further, we will work on any problems of our design as well as on breadboards to simplify the PCB design process. By getting a working PCB, we should put ourselves in a good position to work on the PCB part.

While not denoted directly on the Gantt charts, Breadboard testing and PCB testing will be an ongoing task that will be performed during all phases of production on the respective CPU components.

See the Gantt charts on the following pages.

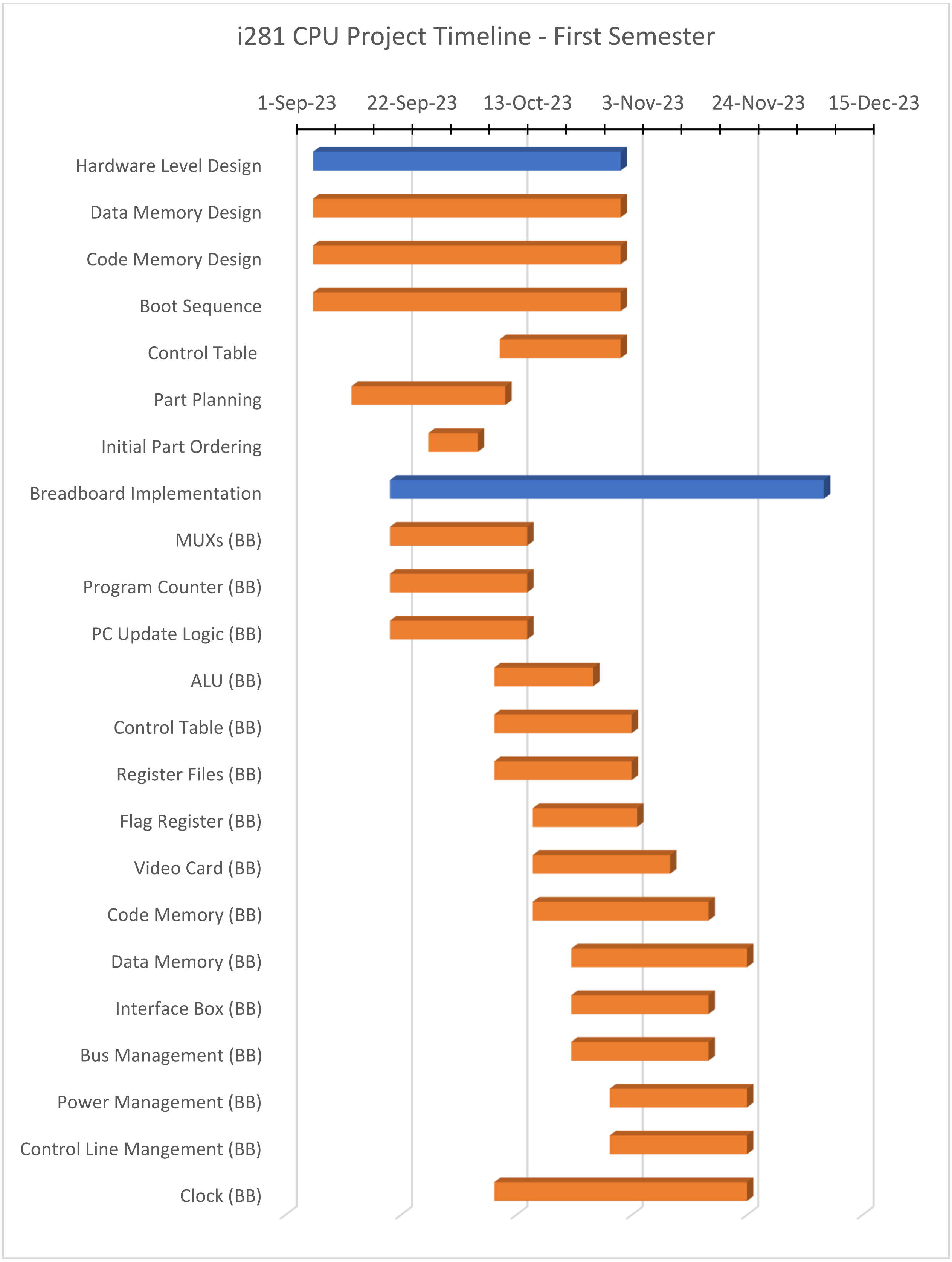


Chart 1 — Gantt Chart of First Semester (EE 491)

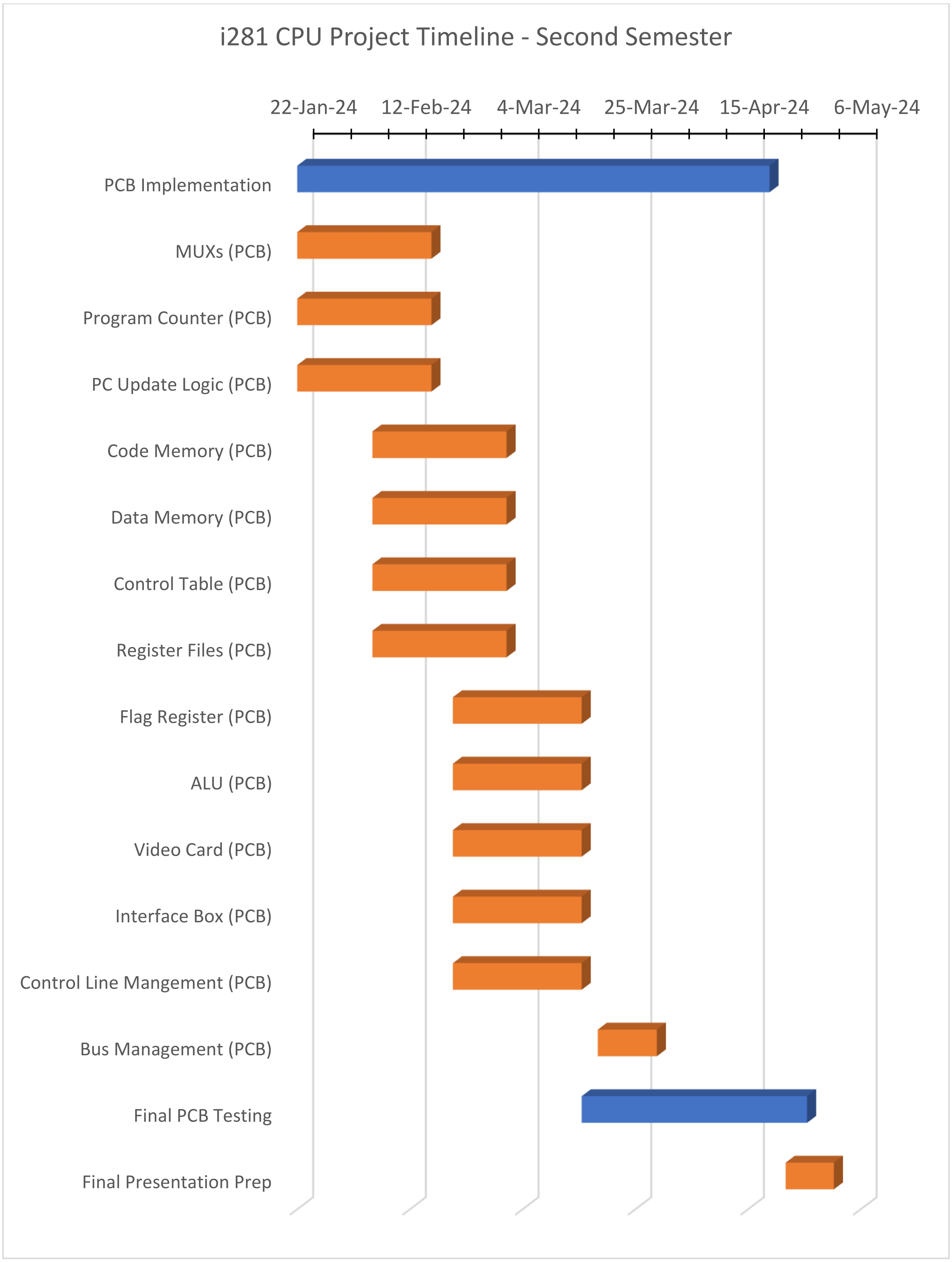


Chart 2 — Gantt Chart of Second Semester (EE 492)

3.5 Risks and Risk Management/Mitigation

Learning from the past group's attempt at this project, we have already considered problems with our hardware-level CPU and are working through those to resolve them early. Every project carries a risk of its plan going wrong, and to mitigate risk, we continuously keep records of our work. This is most obvious in the schematics we create to keep written records of our circuits. Schematics are a helpful tool in our risk management since, if needed, they allow another team to pick up from where we left off. By leveling a clear written record of our design process, another team can use our design and finish any work that might not have been finished in time.

We will deal with a common problem throughout our project: shipping times and availability. We must always be aware of these delays when getting new parts. Although this can increase the build parts of our breadboard version, we are not overly picky on specific components, allowing us to get many parts that fit our usage. The time to get the PCB components will be quite long compared to normal off-the-shelf components; this will be a concern when in the final stages of our project. We hope to have all the information prepared at this point, so if we cannot get the final PCB, they should not be difficult to get and use.

Lastly, we face the hurdles of being senior students finishing our degree program(s). While our academics are our highest priority, all of us are taking 300 or 400+ classes that require our upmost attention as well. Balancing work, life, and academics offers risk to miss tasks and deadlines if not treated with respect. As such, our goal to mitigate such risks is to monitor the progress of all members and hold ourselves accountable. Furthermore, we have regular times to perform our work that cannot be swept away from other responsibilities.

3.6 Personnel Effort Requirements

Courtesy of completing a large portion of the breadboard implementation, rough time estimates were compiled and shown against the estimations listed. Certain components have not yet been implemented and thus Not Applicable (N/A). In-progress is a plus "+".

Task Name	Approx. Hours Estimated	Approx. Hours Spent
Data Memory Design	2	N/A
Code Memory Design	2	4
Boot Sequence	5	8
Control Table Design	2	2
Part Planning	10	N/A
MUXs (BB)	12	13.5
Program Counter (BB)	3	4
PC Update Logic (BB)	5	5
ALU (BB)	10	22
Control Table (BB)	10	12
Register Files (BB)	10	42
Flag Register (BB)	10	3
Video Card (BB)	10	N/A
Code Memory (BB)	10	10
Data Memory (BB)	10	N/A
Interface Box (BB)	10	20
Bus Management (BB)	10	4
Power Management (BB)	10	1+
Control Line Management (BB)	10	1+
Clock (BB)	10	5
MUXs (PCB)	5	N/A
PC Update Logic (PCB)	5	N/A
Program Counter (PCB)	5	N/A
Code Memory (PCB)	20	N/A
Data Memory (PCB)	10	N/A
Control Table (PCB)	15	N/A
Register Files (PCB)	20	N/A
Flag Register (PCB)	5	N/A
ALU (PCB)	5	N/A
Video Card (PCB)	8	N/A
Interface Box (PCB)	10	N/A
Control Line Management (PCB)	5	N/A
Bus Management (PCB)	10	N/A
Final PCB Testing	80	N/A
Final Presentation Preparation	40	N/A

Table 20 - Personal Effort Requirements Towards Each Component

3.7 Other Resource Requirements

Compared to other projects, implementing the i281 CPU requires numerous resources.

1. The Electronics and Technology Group (ETG) is our distributor of parts and maintenance group for the Senior Design lab. ETG will order our parts provided we supply a Bill of Materials (BOM) and aid us with any questions we may have.
2. Professor Stoytchev, our client, is one of the founding designers of the i281 CPU. As we are implementing his design, we will require his guidance and requirements throughout the entire project.

3.8 Financial Requirements

The project, according to our project proposal, is using departmental funding. Our approximate budget is $1,000 USD with the ability to use less or more depending on appropriate reasoning. The only financial requirement for the project is to stay within our approximate budget and save money where both possible and appropriate.

It would be considered inappropriate to reduce costs if the part(s) being purchased are of reduced quality in such a way that may damage or hinder the project.

4. Design

4.1 Design Content

The design of the i281 CPU includes a multitude of components that will all interconnect and work together to process and run programs. This project requires that we build all the circuitry on breadboards a have ribbon cables leading from subcircuit to subcircuit to make the design easier to read. One of the main goals is to make the design readable and easy to understand, so a class of students could easily understand the design.

4.2 Design Complexity

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

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

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

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

These components do not match or exceed modern solutions. Modern processors are built on silicon chips through lithography, ion doping, electroplating, and etcetera. These processors can and will perform far better in all capacities compared to our project. While we cannot match or exceed modern solutions, the project trumps complexity of a modern processor by being built on breadboards using integrated components (ICs) and wiring rather than being a design on a computer that will be sent to a manufacturing facility.

4.3 Modern Engineering Tools

Our tools can be split into a few categories: design, develop, and deploy. Design tools are used to produce schematics, diagrams, and other relevant components to aid or produce a design for one or more CPU components. Development tools are used to produce more tangible results from designs. Deployment tools are for producing a final product.

Some tools may fall into multiple categories due to their wide range of capabilities.

Engineering Tool	Category	Purpose
KiCad Schematic Editor	Design/Develop	Drawing technical schematics of all CPU components.
KiCad PCB Editor	Develop/Deploy	Producing Gerber files to send off for PCB printing. Utilizes the schematics and footprints from other KiCad programs.
Inkscape	Design	Creating diagrams and figures for explaining and outlining high-level discussions of a CPU component design. Additionally used for graphical support.
Excel	Design	Creating part order lists and keeping track of various pieces of data.
GitLab	Design	Keeping track of milestones and design goals.
Word	Design	Creating documentation and explanations for the design.

Table 21 - Tool List for the i281 CPU Design

4.4 Design Context

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

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

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

4.5 Prior Work/Solutions

As mentioned in the subsection regarding Design Complexity, a homebrew computing kit exists on the market to build 8-bit retro computers [1]. As a comparison between the product and ours:

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

Table 22 - Comparison Between Ben Eater's CPU and the i281 CPU

This project marks our second endeavor in implementing the i281 processor on both breadboard and PCB. The previous Senior Design team (sddec22-20) achieved a proof-of-concept by the end of their second semester. Leveraging insights from their experience, our team has made informed decisions early in the project timeline to avoid their past mistakes, enhancing the likelihood of successful completion. Our timeline also encompasses the second phase of the project, focusing on designing and producing PCBs using KiCad. Unlike the previous team's design, our PCBs aim to showcase the computer's processes, serving as an educational tool rather than a disassembling learning tool.

To learn from the previous team's experience, our approach involves categorization, general organization, and careful consideration of the intricacies that the previous team found challenging. Knowledge transfer from the earlier team guided our decisions, emphasizing the importance of avoiding design inconsistencies and treating circuits with utmost care. To maintain constructive feedback, Dr. Stoytchev provided remarks on what worked and didn't work in the last project, offering valuable insights without denigrating the efforts of the previous team. The prior project faced challenges, resulting in a non-functional breadboard computer due to design inconsistencies and insufficient circuit care. Our team is committed to learning from these experiences to enhance the success of our implementation.

4.6 Design Decisions

4.6.1 TTL Chip Selection

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

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

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

4.6.2 Design Layout

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

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

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

4.6.3 Read/Write of RAM

The original i281 design was built using Verilog for FPGA devices. As such, certain constraints could be ignored One of these major constraints was reading and writing to RAM in a single clock cycle. Nearly all RAM on the market is considered “single-port” RAM. The data lines for reading and writing are on the same pins of the IC. Given this restriction, reading and writing to RAM must be performed in two separate clock cycles which conflicts with the requirements of the project. This only occurs if RAM passes an instruction that would alter the contents of RAM in the same cycle.

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

4.6.4 Boot Sequence

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

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

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

4.7 Proposed Design

4.7.1 Design 0 (Initial Design)

4.7.1.1 Design Visual and Description

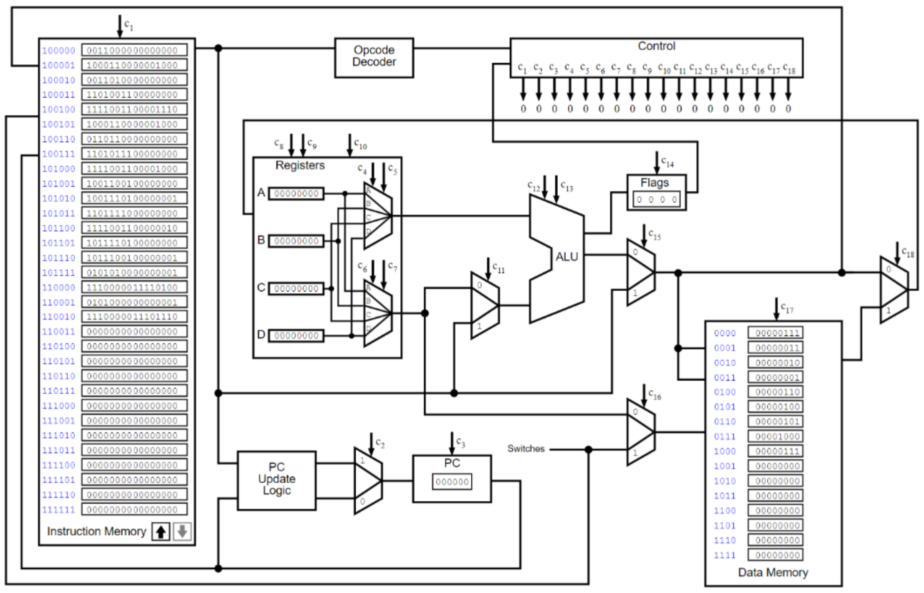


Figure 1 - i281 CPU Simulator Design

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

4.7.1.2 BIOS

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

4.7.1.3 Code Memory

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

4.7.1.4 Register Files

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

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

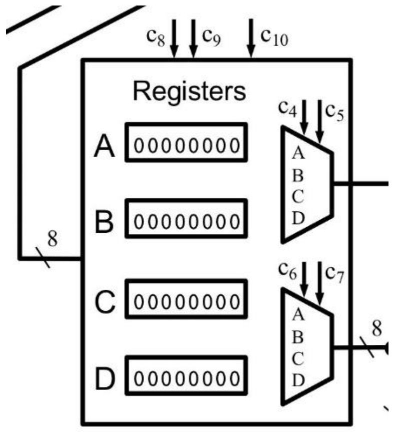


Figure 2 - Register Block From Simulator

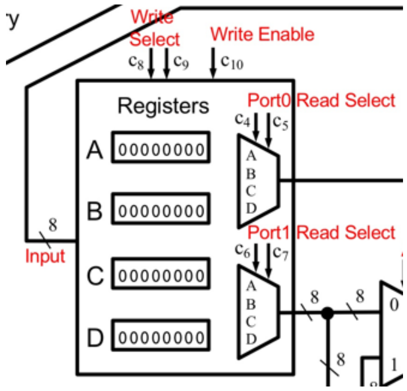


Figure 3 - Register Block from Simulator

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

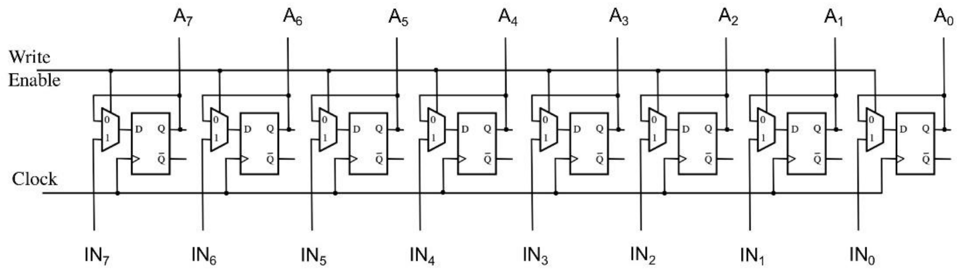


Figure 4 - Register File from Simulator

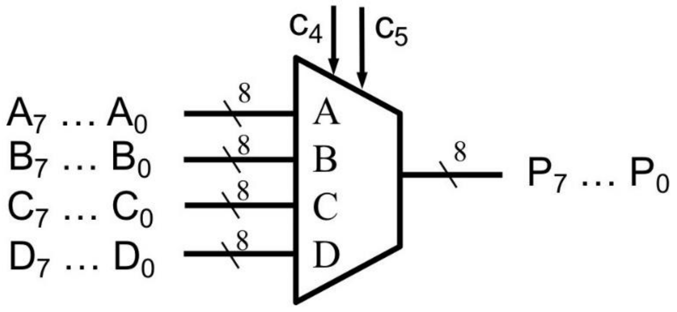


Figure 5 - 8-bit 4-1 Mux

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

4.7.1.5 Arithmetic Logic Unit

For Design 0 of the Arithmetic Logic Unit (ALU), we were given an initial design that was used to create both the simulator and the FPGA design. As we can see from the image below, the ALU contains a few subcircuits: the 8-bit shifter, 8-bit adder, a few multiplexers, and a flag calculator. For our design, we will also be adding the flag registers as part of the ALU design which is just a 4-bit register file. The design employs three control signals. Two of the control signals are labeled ALU_SELECT0 and ALU_SELECT1 in the picture and they determine the output of the ALU (the second picture shows the different combinations that lead to different opcodes). The third one is to control the flag register. This design will output one 8-bit bus (ALU_RESULT in the picture) and one 4-bit bus (the flag register outputs). The flag register outputs the carry, negative, overflow, and zero flag.

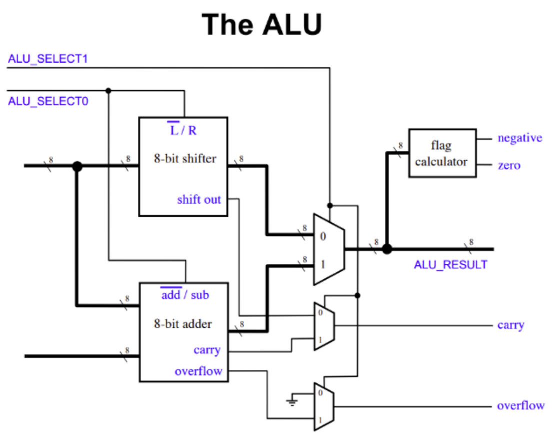


Figure 6 - ALU Subcomponents Diagram

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

Figure 7 - ALU Arithmetic Mode Table

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

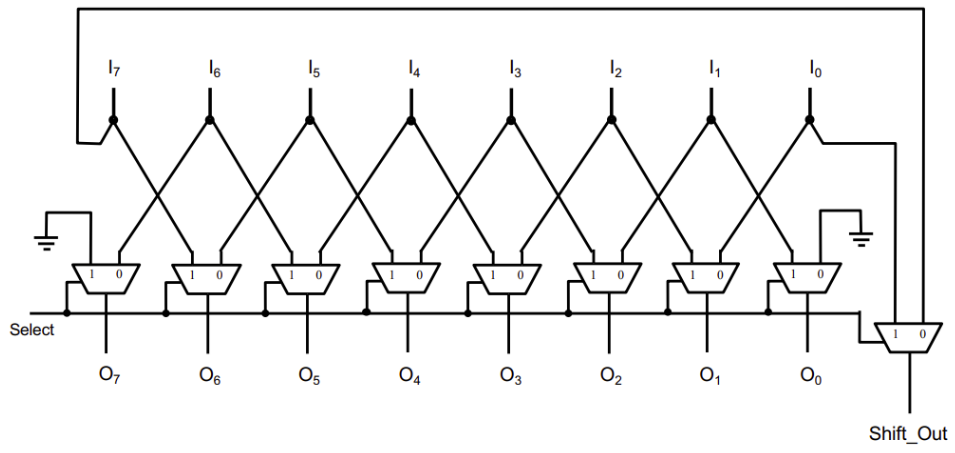


Figure 8 - ALU Shifter Design

The 8-bit adder/subtractor circuit is capable of both adding and subtracting two 8-bit numbers. When the ALU_SELECT0 control line is at a logic low, the circuit is in addition mode. When it is at a logic high, then it is in subtraction mode. This circuit is essentially eight full adders tied together in a line. This circuit also outputs a carry bit and a negative bit. All the output bits labeled S_{0-7} can be put into an 8-bit NOR to determine if the value is zero. If it is, it triggers the zero flag. Lastly, the C_7 and C_8 carry bits can be put into an XOR to show if the circuit has overflowed, meaning the result was too large to fit in the number of bits provided.

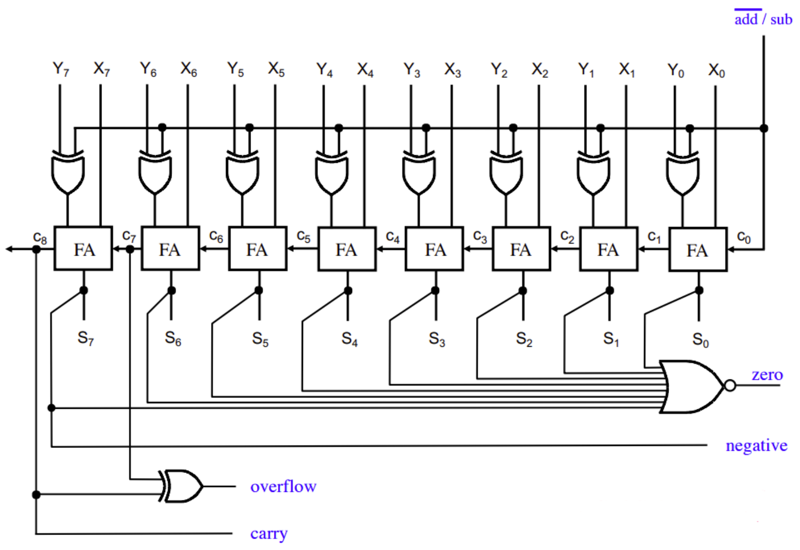

add / sub
Y7 X7 Y6 X6 Y5 X5 Y4 X4 Y3 X3 Y2 X2 Y1 X1 Y0 X0
c8 c7 c6 c5 c4 c3 c2 c1 c0
FA
S7 S6 S5 S4 S3 S2 S1 S0
zero
negative
overflow
carry

Figure 9 - ALU Addition/Subtraction Design with Flags

4.7.1.6 Program Counter

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

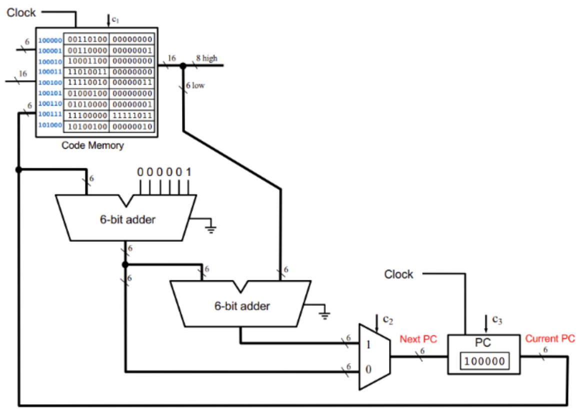


Figure 10 - Program Counter Design

4.7.1.7 Data Memory

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

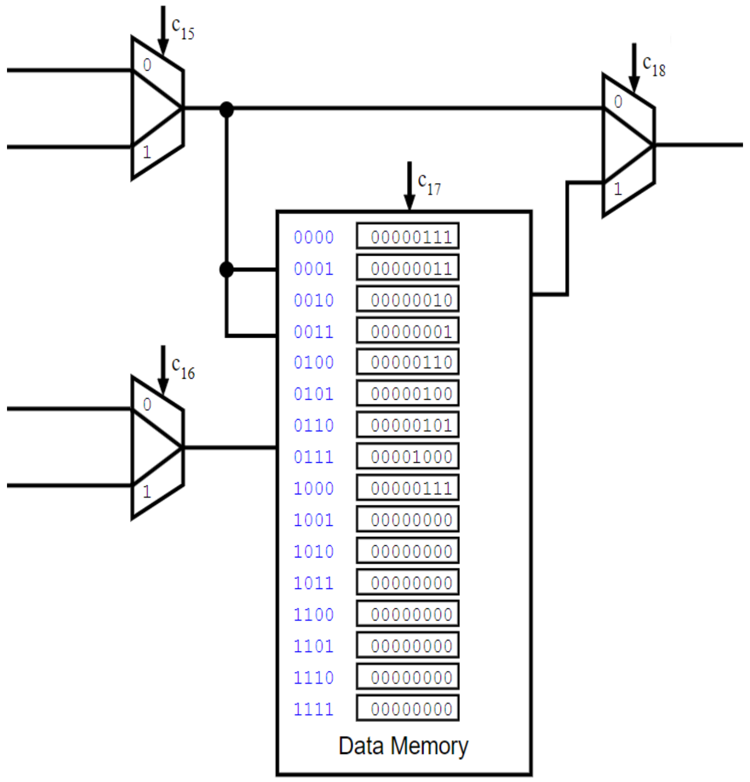


Figure 11 - Data Memory diagram from the simulator

4.7.1.8 Video Card

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

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

4.7.1.9 Control

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

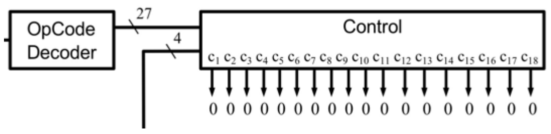

OpCode Decoder
27
4
Control
c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12 c13 c14 c15 c16 c17 c18
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Figure 12 - Control Logic Blocks

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

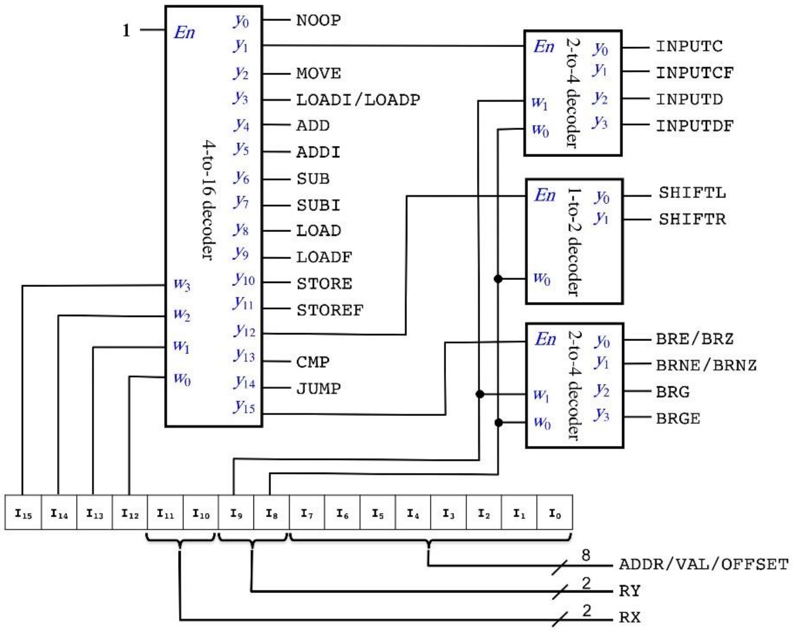
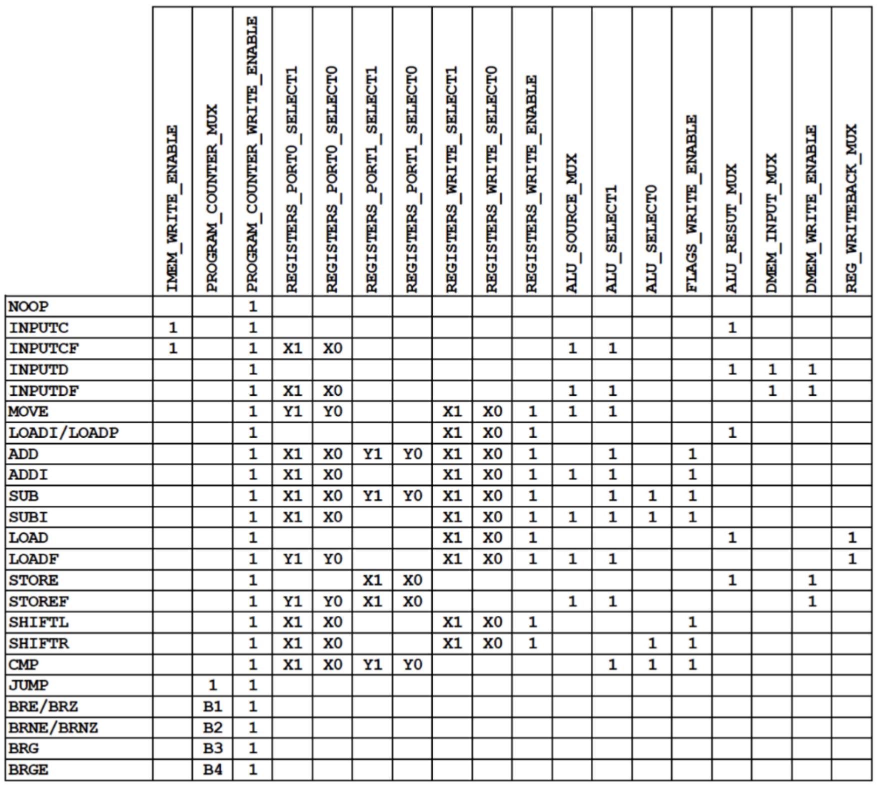


Figure 13 - OpCode Decoder Logic

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



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

Figure 14 - Control Lines Table

4.7.2 Design 1

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

4.7.2.1 BIOS

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

4.7.2.3 Code Memory

To avoid issues with needing to modify or load a program in the same chip as the BIOS, a ROM and RAM chip will be distinguished. In addition to containing ROM and RAM, which holds the BIOS, Code Memory handles all interactions with program loading and execution. Instructions for running the user's program will be stored and executed from the RAM. This will be filled in with the necessary instructions during boot. Either the RAM will be filled from a storage chip on the device storing sample programs or filled using the switches manually (via BIOS loader).

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

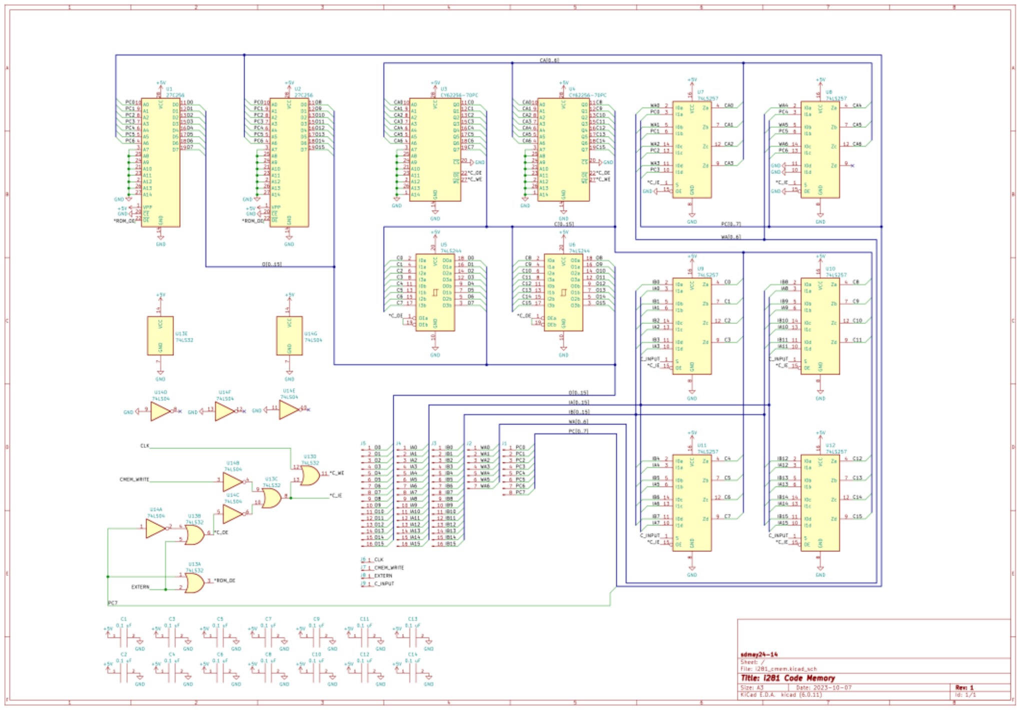


Figure 15 - Code Memory Schematic

4.7.2.4 Register Files

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

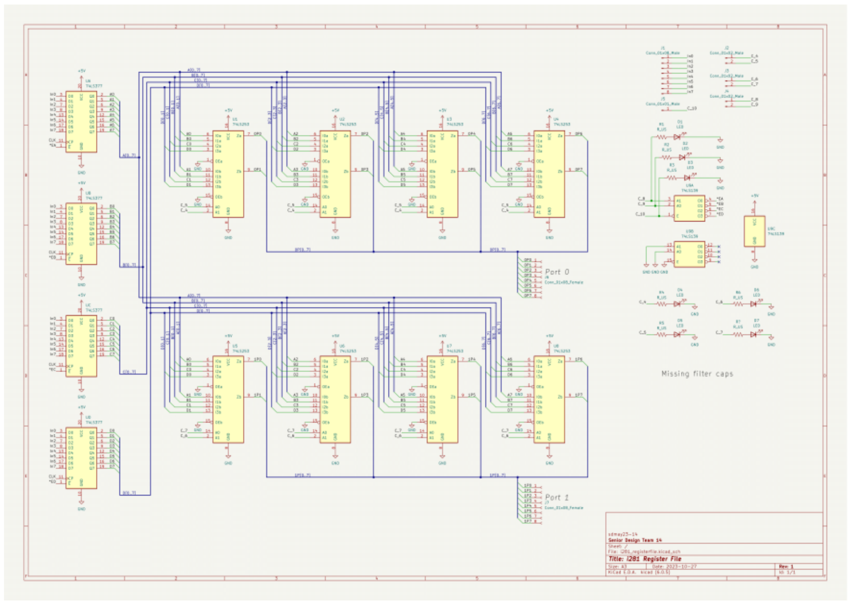


Figure 16 - Register File Schematic

4.7.2.5 Arithmetic Logic Unit

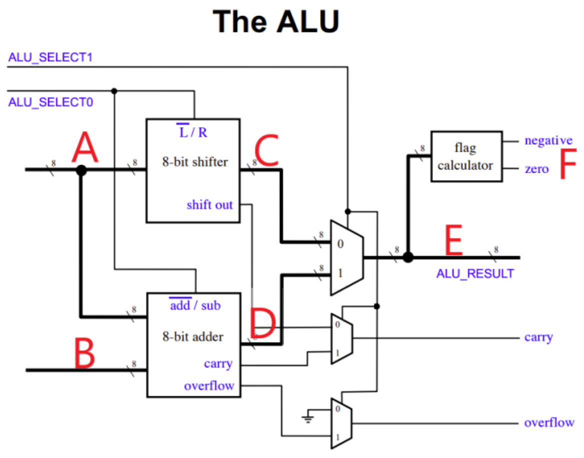


Figure 17 - ALU Ports Labeled on Diagram

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

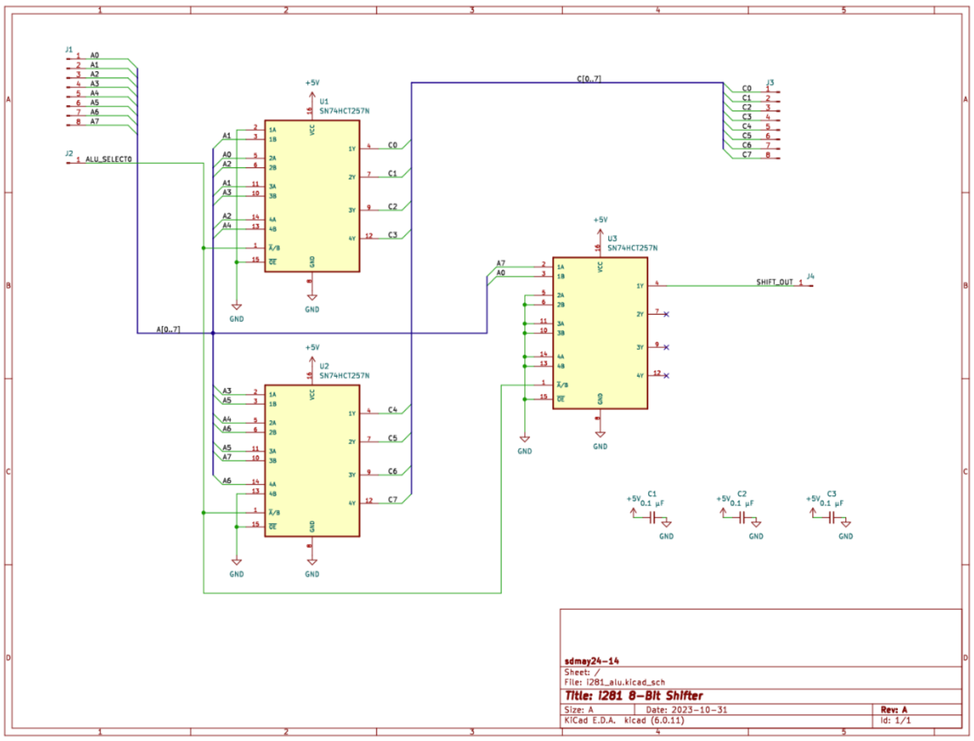


Figure 18 - Initial Schematic Design of ALU 8-Bit Shifter

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

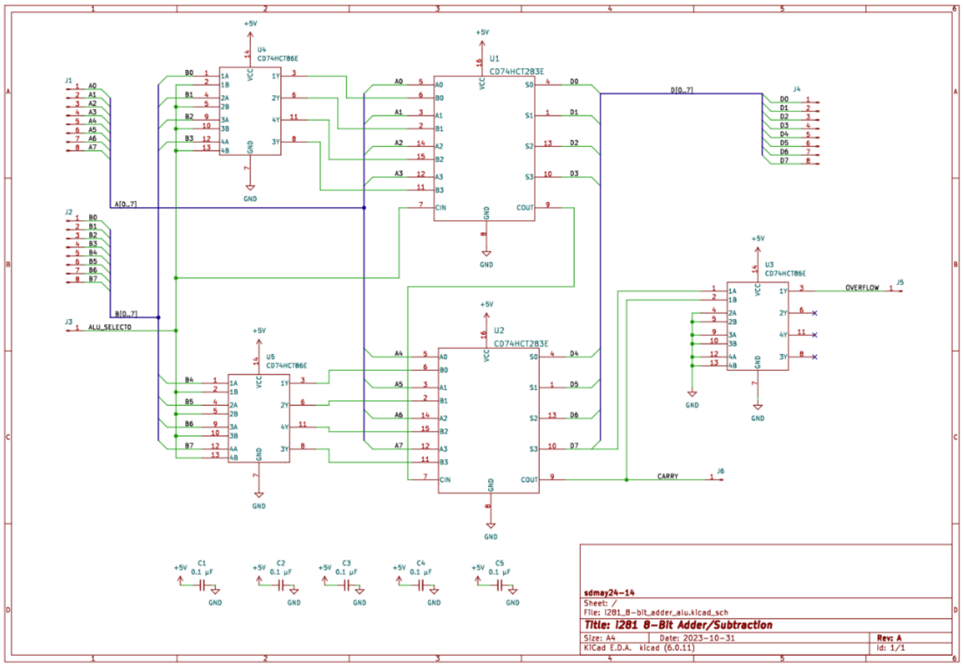


Figure 19 - Initial Schematic Design of 8-Bit Addition/Subtraction Component Schematic

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

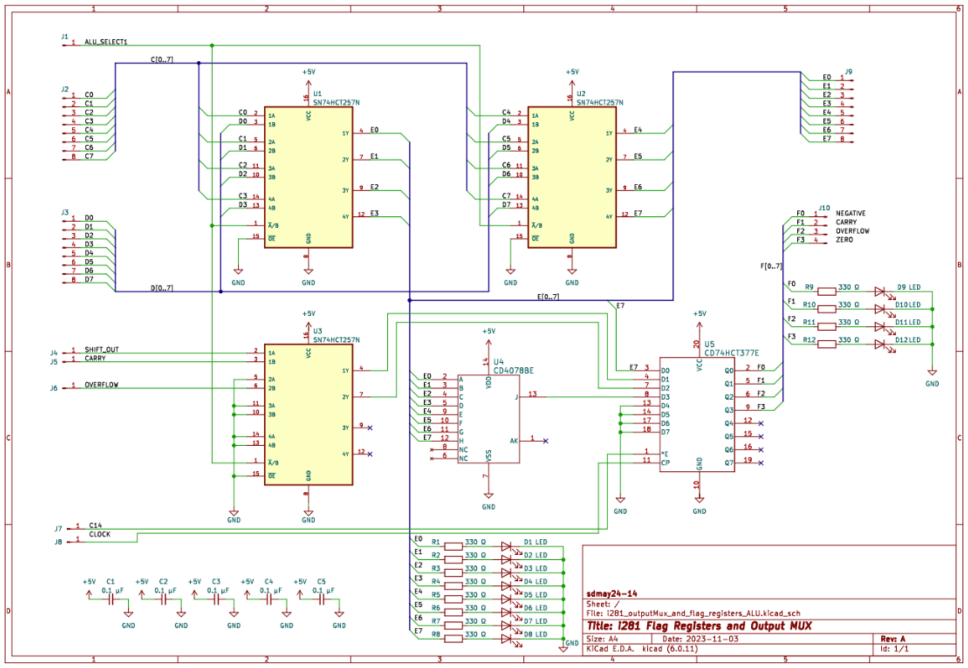


Figure 20 - ALU Flag Registers and Output MUX Schematic

4.7.2.6 Program Counter

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

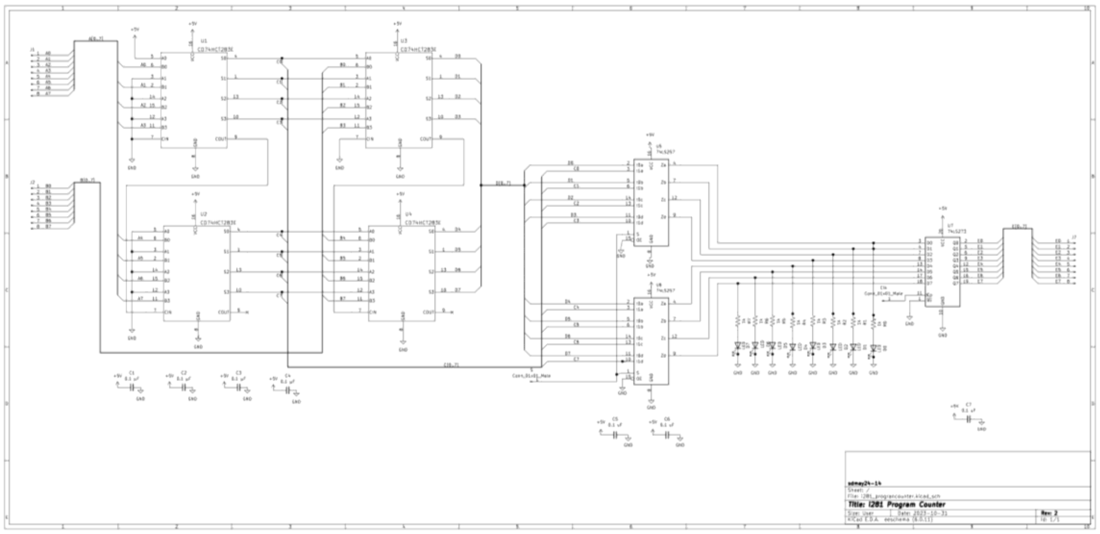


Figure 21 - Program Counter schematic

4.7.2.7 Control Table

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

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

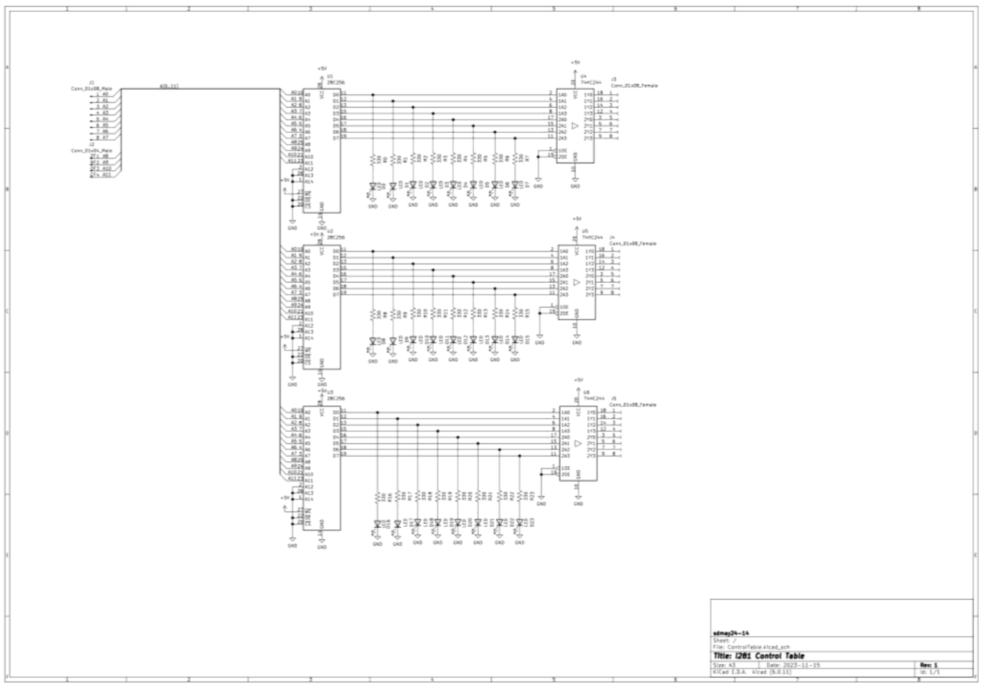


Figure 22 - Control table schematic

4.7.3 Design 2

Certain components were iterated upon in this second design.

4.7.3.2 Code Memory

A visualization panel was added in extension to the component to showcase the current instruction and program address.

4.7.3.4 Arithmetic Logic Unit

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

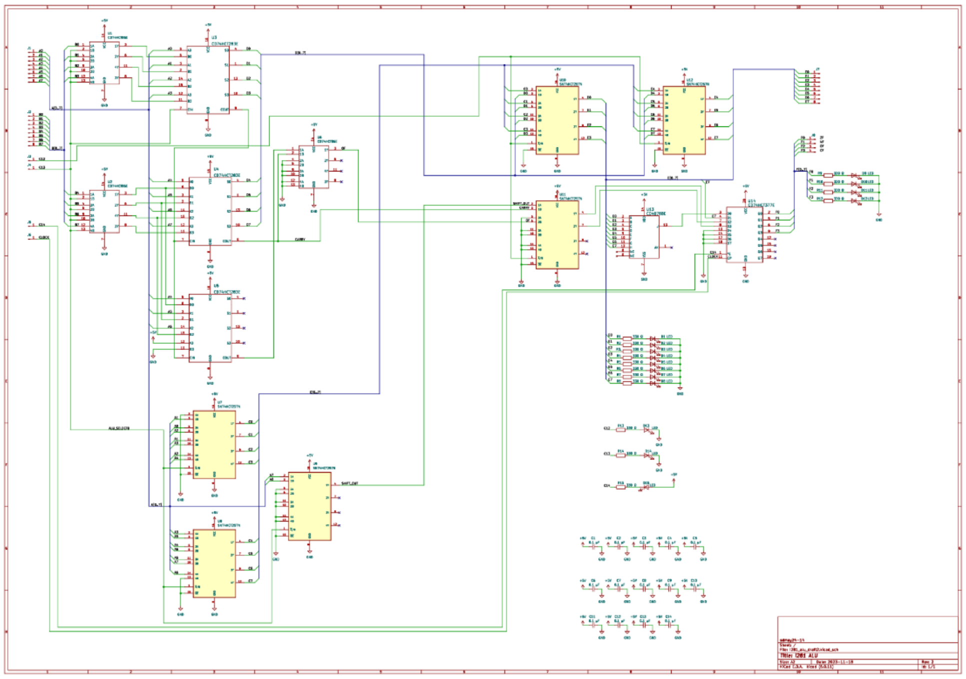


Figure 23 - ALU Subcomponents Final Schematic

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

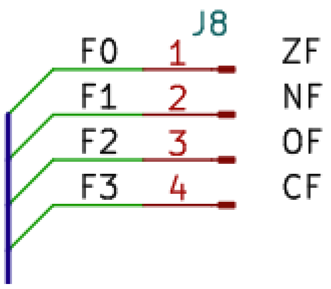


Figure 24 - Output Flags on Schematic

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

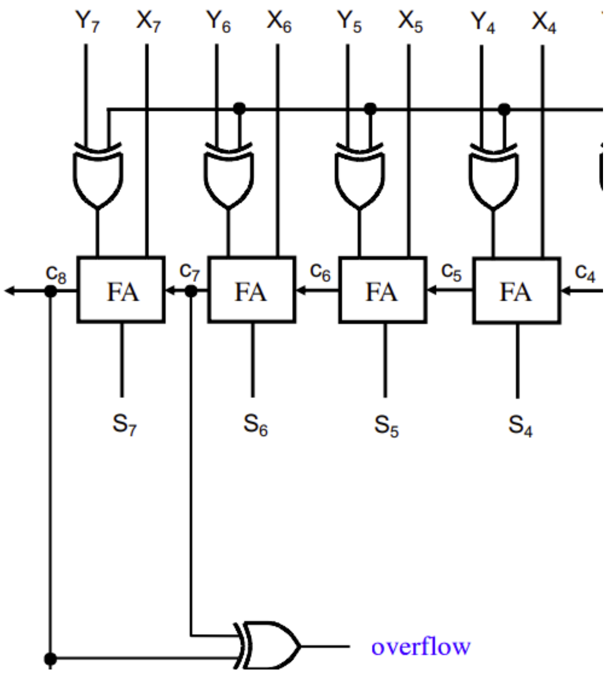


Figure 25 - Adder Circuit Zoomed in on Last Few Bits

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

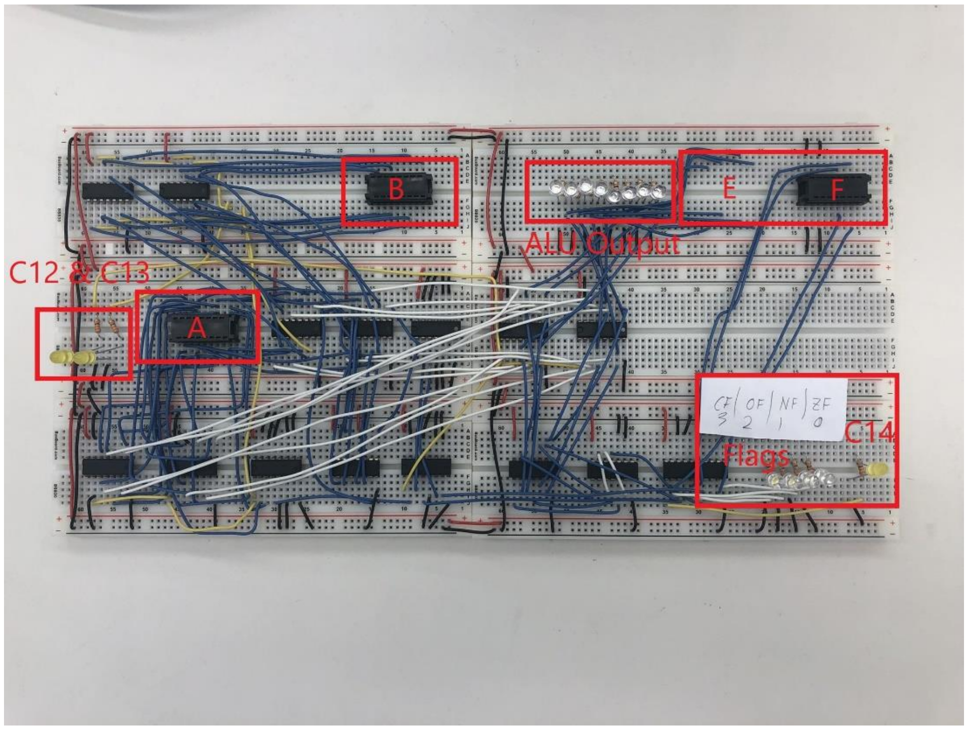


Figure 26 - Implementation of the ALU on Breadboards

4.7.4 Functionality

Describe how your design is intended to operate in its user and/or real-world context. This description can be supplemented by a visual, such as a timeline, storyboard, or sketch.

How well does the current design satisfy functional and non-functional requirements?

4.7.5 Design Visual and Description

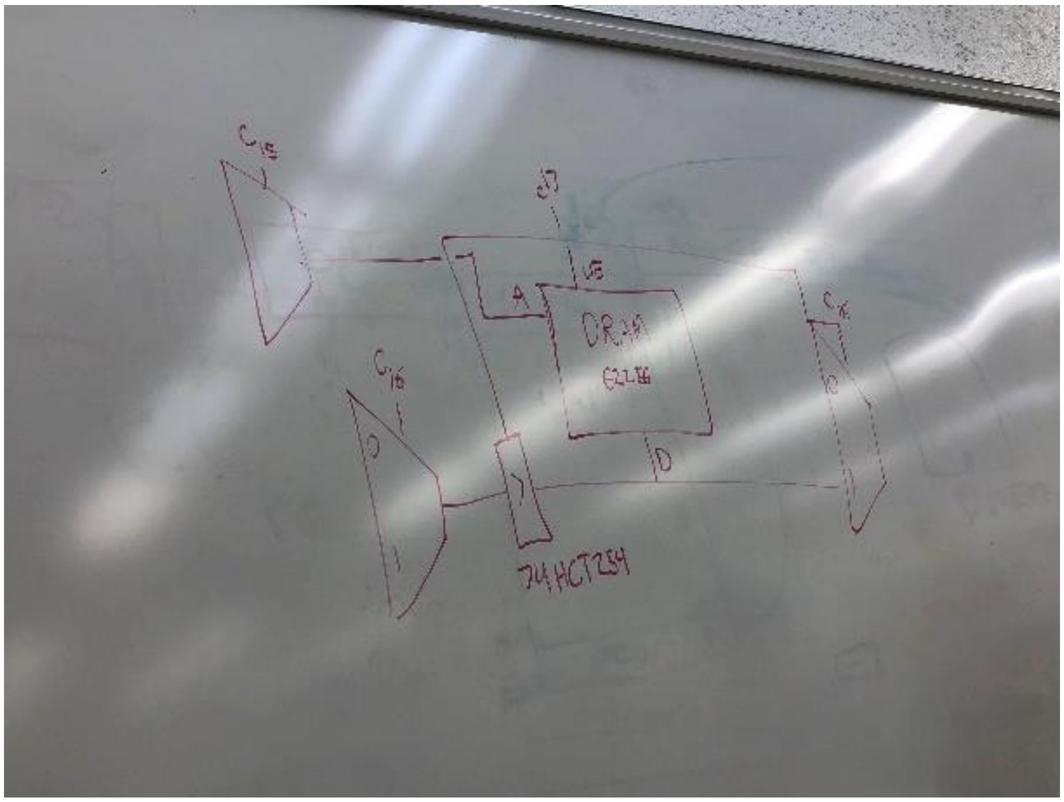


Figure 27 - Data memory modifications for physical hardware implementation

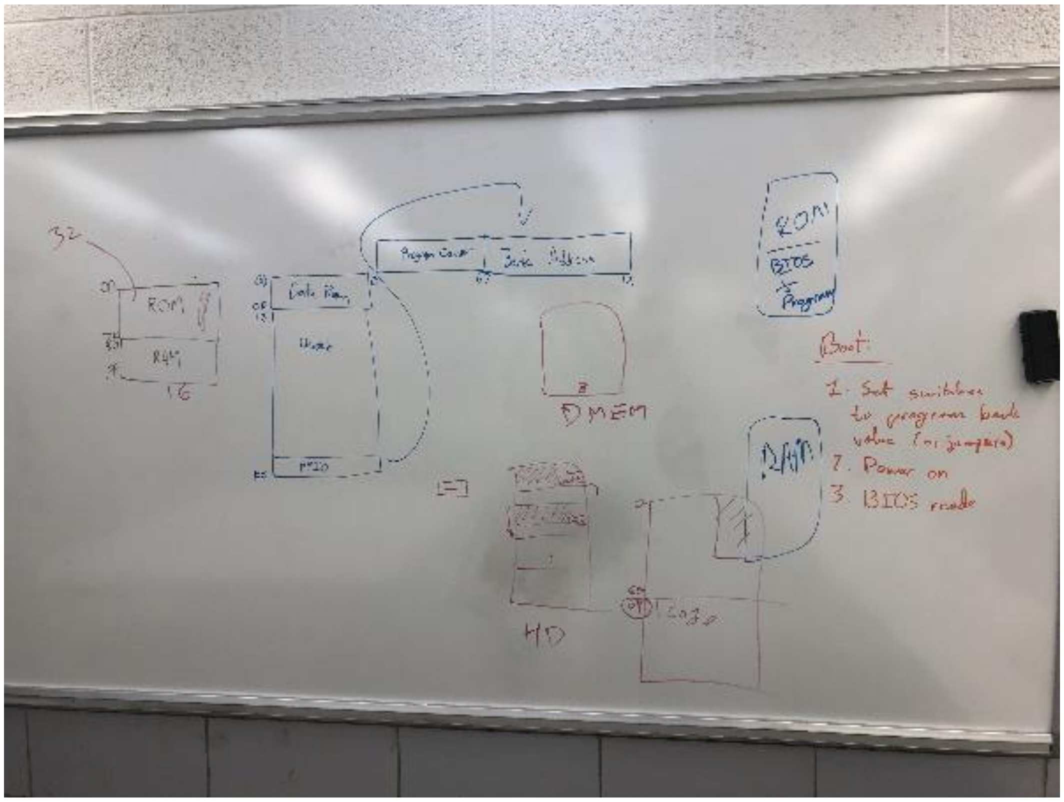


Figure 28 - Code memory discussion

4.8 Technology Considerations

For a successful breadboard project, it's crucial to ensure the compatibility of selected components in terms of voltage, current, and signal requirements. Verify the stability and sufficiency of the power supply to prevent issues related to voltage fluctuations or inadequate current for the components. Pay close attention to signal integrity to avoid interference, noise, or crosstalk that could impact the circuit's proper functioning. Select appropriate documentation tools to maintain clear and comprehensive records for effective collaboration and future reference. Consider the availability and functionality of testing equipment, such as multimeters and oscilloscopes, for thorough testing and troubleshooting during the design and implementation phases. Additionally, using KiCad, a free and accessible design software, is advantageous for schematic and PCB design, though addressing the learning curve for less experienced users is essential to optimize its usage for the project.

4.9 Design Analysis

The initial design draft from Section 4.7 showed theoretical promise but required adjustments for practical implementation. Not all elements from the simulator or FPGA design could be directly translated into ICs. Due to the project's limited timeframe, we streamlined the design for efficiency and ease of testing, focusing on chip efficiency. This involved evaluating the balance between component complexity and functionality, ensuring alignment with project objectives, and staying within scope and timeline constraints.

5. Testing

5.1 Unit Testing

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

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

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

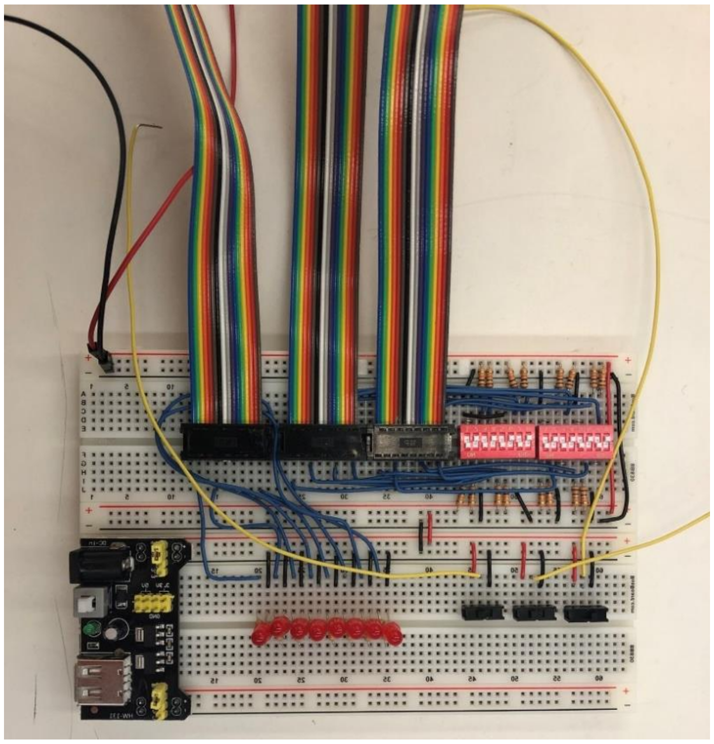


Figure 29 - Testing Rig

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

5.2 Interface Testing

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

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

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

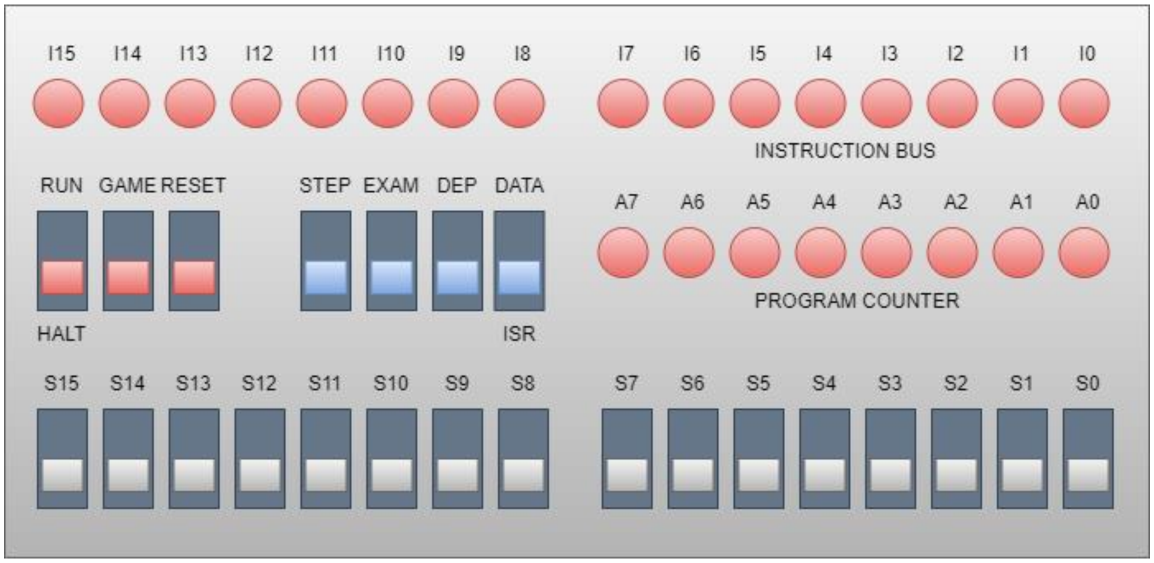


Figure 30 - User Panel Design

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

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

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

Table 23 - Switch Types and Applications

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

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

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

5.3 Integration Testing

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

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

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

Importantly, the MVP explicitly excludes:

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

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

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

5.4 System Testing

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

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

5.5 Regression Testing

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

5.6 Acceptance Testing

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

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

5.7 Security Testing

Security is not a concern for this project. While we have made considerations about security, none were implemented for the sake of project complexity and the lack of requirements. Security features and further considerations will be a topic in the far future for students examining the hardware implementation of i281 processor.

5.8 Results

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

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

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

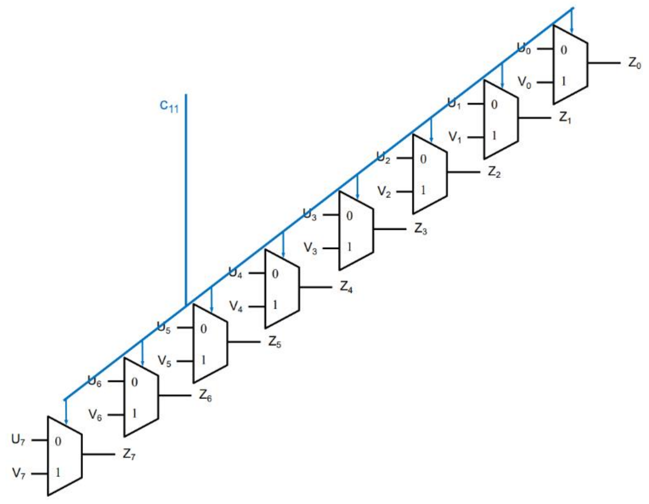


Figure 31 - 8-Bit 2-1 Multiplexer Design

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

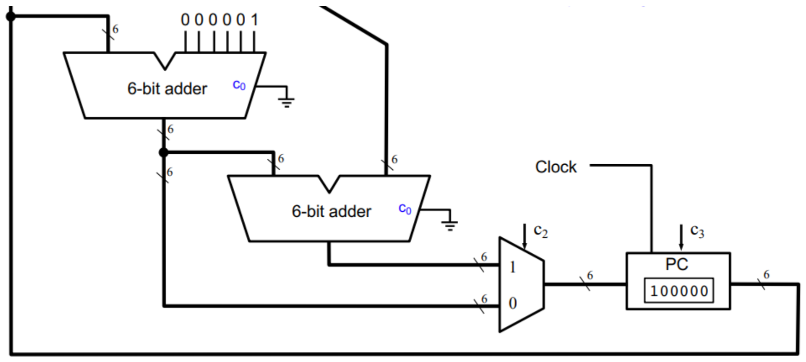


Figure 32 - Program Counter Design

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

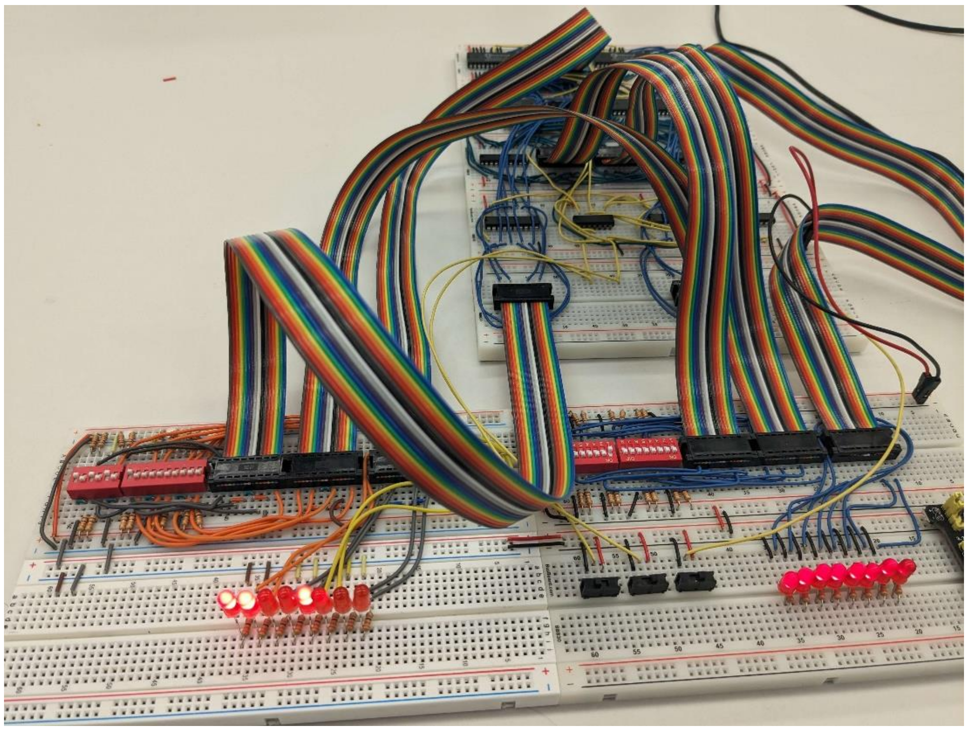


Figure 33 - Testing Rig Connected to the RAM and ROM

We tested the ALU in a very similar way. We used one of the testing rigs to hook up to the ALU since they were designed to have two 8-bit input busses and one 8-bit output bus. The ALU had three control signals, so we connected each of those two a switch. We also used the Arduino Nano clock board to simulate the clock signal in the flag register chip. By changing the values of C_{12} and C_{13}, we were able to switch the mode the ALU was in. We exhaustively tested each mode with multiple different potential inputs. To see if the design was working, we looked at the output LEDs. Based on our inputs, we predicted the outputs and checked if they matched the LEDs. For the addition and subtraction arithmetic, we used inputs from Professor Stoytchev's CPR E 281 class slides to verify our results. In addition to the outputs, we used the slides to check if the Zero, Negative, Overflow, and Carry flags were triggered. When we initially tested the design, we found a variety of errors such as an incorrect orientation of the output LEDs, incorrect overflow errors, and issues with the shifter circuit. These issues were then debugged, fixed, and some led to Design iteration two. After the issues were addressed, the design was retested and passed.

6. Implementation

6.1 Implementation Plan

Next semester's implementation plan involves the finalization of our breadboard design, which is currently in progress. Activities include the completion of outstanding tasks, comprehensive design reviews, and addressing any identified issues for optimization. As we work towards concluding the breadboard phase, our focus will transition to the preliminary stages of the i281 CPU's PCB design. This transition includes developing an initial PCB layout based on the breadboard design, gathering feedback for refinement, and identifying and sourcing components for the upcoming manufacturing phase. Consultations with experienced professionals will be sought to optimize the PCB layout, while simultaneously, a comprehensive testing plan will be developed to ensure the functionality and quality of the design. Throughout this process, we will continually assess and adjust the budget based on refined estimates for PCB manufacturing and assembly.

6.2 Financial Spending

Throughout our first semester working on this project, we placed three-part orders and kept track of our spending. The tables below outline the financial details for each order, providing a transparent breakdown of costs, quantities, and sources for each electronic part.

Part Order #1

Part Name	**Quantity**	**Price Per Unit**	**Total Cost**
Solderless Breadboard	50	2.95	147.5
Arduino Nano	1	14.7	14.7
8-Position DIP Switch	10	0.277	2.77
0.1uF Bypass Caps	100	0.028	2.8
Hex NOT Gates	10	0.55	5.5
Quad AND Gates	10	0.51	5.1
Quad OR Gates	10	0.51	5.1
Quad XOR Gates	10	0.77	7.7
Quad NAND Gates	10	0.55	5.5
Quad NOR Gates	10	0.55	5.5
Octal Latch + Enable	20	0.822	16.44
Quad 2-1 Mux	20	0.737	14.74
4-Bit Adders	10	0.864	8.64
10x 27C256 ROMs	1	9.95	9.95

Octal Bus Buffers	10	0.45	4.5
Grey Ribbon Cable	3	5.95	17.85
7-Segment Display	10	0.401	4.01
6 Color Wire (100 FT) [22]	1	39.95	39.95
LED Green 5mm	>10	0.15	1.5
LED Red 5mm	>10	0.12	1.2
LED Blue 5mm	>10	0.17	1.7
LED Yellow 5mm	>10	0.15	1.5
LED Orange 5mm	>10	0.115	1.15
LED White 5mm	>10	0.19	1.9

Table 24 - Part Order #1 parts and costs

Order Cost: **$327.20**

Part Order #2

Quantity	**Part Number**	**Manufacturer Part Number**	**Description**	**Cost**
1	B09136G5JL	Oumefarezyh3f70kw312-01	EPROM Eraser 120-220V	27.69
25	38C9328	MCM 26-580	Round Paddle On-On Toggle Switch	24.25
10	98K4970	1MS2T1B1M1QE	Round Paddle On-(On) Toggle Switch	22.90
20	2267079	103-7014-EVX	Rocker Paddle On-On Switch	7.80
10	2267039	103-7008-EVX	Rocker Paddle On-(On) Switch	4.90
10	296-8406-5-ND	SN74HCT257N	Quad 2-1 Mux	7.37
5	296-1613-5-ND	SN74HCT273N	Octal Latch + Clear	4.45
60	42674	8200-16-R	DIP Package Cable Connectors	53.4
1	36822	782201 GR005-JVP	22 AWG Green Wire 100 Feet	9.95
1	36768	782201 BL005-JVP	22 AWG Blue Wire 100 Feet	9.95
1	99363	ICS-01-R	IC Pin Straightener	7.95
50	34761	LG3330	LED Green 5mm	7.50

50	333973	UT1871-81-M1-R	LED Red 5mm	6.00
50	2234071	LL-50ABD2E-017	LED Blue 5mm	8.50
50	34825	LY3330	LED Yellow 5mm	7.50
50	2290247	C512A-WNN-B0-WM4-28	LED Orange 5mm	9.50
1	643858	3365/16 100-JVP	16 Conductor Gray Flat Ribbon Cable 100 Feet	44.95
100	690742	CF1/4W331JRC	Resistor Carbon Film 330 Ohm 1/4 Watt 5%	2.50

Table 25 - Part Order #2 parts and costs

Order Cost: $278.88

Part Order #3

Quantity	Part Number	Manufacturer Part Number	Description	Cost
5	296-1608-5-ND	SN74HCT138N	3-8 Active-Low Decoder/Demux	4.20
5	12950	CD4040BE	12-Stage Binary Ripple Counter	3.45
5	27924	MXO45-3C-2M0000-JVP	2 MHz Full Can TTL Crystal Oscillator	9.75
3	30AC8774	NR01105ANG13	Rotary Switch, 5 Position	17.01
1	--	--	Sheet Protector, 200 cnt	15.50
100	2267079	103-7014-EVX	Rocker Paddle On-On Switch	25.00
25	2267039	103-7008-EVX	Rocker Paddle On-(On) Switch	12.25
10	45022	74HCT244	Octal 3-State Buffer	3.90
7	595-CD4078BE	CD4078BE	8-Input NOR	4.90
1	2257539	LRS-50-5	5V 10A Power Supply	13.50
1	102007	2381.12	10 Ft Power Cable	5.95
1	36768	782201 BL005-JVP	22 AWG Blue Wire 100 Feet	9.95

Table 26 - Part Order #3 parts and costs

Order Cost: $125.36

Total Spending: $731.44

These details give a straightforward view of how we spent our budget during the semester. By tracking expenses for each part in a clear table, we've made it easy to manage our budget and plan for future spending. This open approach helps us stay accountable and makes it simpler to make well-informed decisions as we move into the next stages of our project.

Our preliminary estimate for the PCB design of our breadboard computer project is around $600. While this is a rough approximation, it encompasses factors such as design intricacies, component costs, manufacturing, testing, and potential iterations. As we progress, we'll refine this estimate through quotes from manufacturers. This initial projection provides a baseline for financial planning as we move forward with the project.

7. Professionalism

This discussion is with respect to the paper titled "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment".

7.1 Areas of Responsibility

Area of responsibility	Definition	NSPE Canon	IEEE Code of Ethics
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence	Perform services only in areas of their competence; Avoid deceptive acts.	"uphold the highest standards of integrity, responsible behavior, and ethical conduct in professional activities"
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs	Act for each employer or client as faithful agents or trustees.	N/A
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	Responsibility to disclose dangers to the public and to see honest criticism of one's own work.
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders	Hold paramount the safety, health, and welfare of the public	"to hold paramount the safety, health, and welfare of the public"
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	"to avoid unlawful conduct in professional activities"
Sustainability	Protect environment and natural resources locally and globally.		"to strive to comply with [...] sustainable development practices"
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	"to improve the understanding by individuals and society [...] emerging technologies"

Table 27 - The seven areas of professional responsibility in the assessment instrument

Work Competence is important to any worker and most of IEEE's first code of ethics applies to different types of this. IEEE's Code of Ethics does not give guidance about the finical responsibly of engineering. NSPE does give guidance, saying being fair to all parties involved. Both codes talk about honest communication with the public. IEEE also wants engineers to be open to criticism when directly about them and their own work. They have the same quote for health, safety, and well-being. Topics about property ownership is a bit of a stretch for IEEE's code of ethics, only a broad no illegal things applies. NSPE has the same answer as financial responsibility. Whereas NSPE does not talk about sustainability, IEEE ensure we strive for ethical designs and sustainable practices. When looking at the social responsibility IEEE algins with the definition giving, to help make a better tomorrow. NSPE's answer is more about being a good person and engineer.

7.2 Project Specific Professional Responsibility Areas

Work Competence is among the most important areas of professional responsibility for out project. We are constantly working on designing or building a component. These need to be made well and correct in a timely manner so that we can use them in the final protypes. We are constantly being counted on by other groupmates and must perform to our own standards.

The areas of professional responsibility that are higher priority for this project include: Financial Responsibility, Communication Honesty, and Health, Safety, Well-Being. These are being throughout our project, and we should not take action to oppose them, but they are not driving our project's progress. For Financial Responsibility, we stay responsible for the purchase order and only getting what we need to complete the requirements. The ultimate goal of this project is to make a learn tool, so we need to ensure that the computer can be used without anyone putting themselves in danger. We are trying to limit the risk of injuring when using the computer.

Lower priority professional responsibilities include Property Ownership and Sustainability. These are not very important for us. Since our design is made by our project professor we do not need to worry about the ownership of that material. We are not purpose being wasteful throughout this project and are practicing sustainable practices throughout the project.

This project in not a great step forward in society nor with it by used by and massive amount of people outside of an academic environment. Therefore, Social Responsibility is not of great importance throughout our project.

7.3 Most Applicable Professional Responsibility Area

Work Competence is the most applicable professional responsibility area to our project, the i281 CPU.

8. Closing Material

8.1 Discussion

The primary outcomes of our project involve the successful assembly and initial testing of the breadboard design. While we achieved the milestone of running a two-instruction program, we encountered a challenge with our largest connector cable, resulting in two pins touching. This issue affected the switch input and the functionality of specific instructions. Despite these challenges, the project has provided valuable insights into the functionality and potential improvements needed for the breadboard design. The next steps will involve addressing these issues, refining the design, and proceeding with the PCB design phase to enhance the overall performance and robustness of our i281 CPU project.

8.2 Conclusion

In our project's current phase, we have successfully undertaken most of the breadboard design, achieving a significant milestone by running a two-instruction program. Our overarching goal for this semester was twofold: first, to complete the breadboard design, and second, to create a comprehensive design document outlining project requirements and considerations for the subsequent implementation phase in the second semester. Concurrently, we aimed to apply our newly acquired skills while refining existing ones through the design process.

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

Outside of having schedule changes and a better ethic toward the project management style, the project operated smoother than expected and we will continue to strive for improvements as our project evolves and concludes.

8.3 References

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

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

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

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

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

9. Appendix A

Project-Specific Standards

10.1 Design Definitions

In addition to all the terms and acronyms listed throughout the document, additional definitions and concepts are left in the appendices for further reading.

Breadboards

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

Scoping

There are two types of “scope”: global and local.

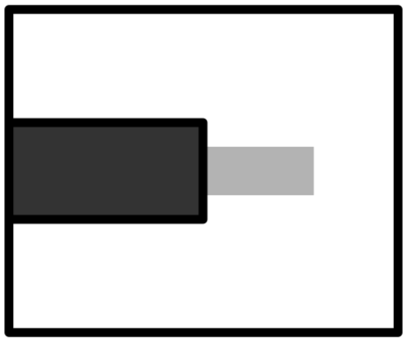
Global scope, or globally scoped, mandates the requirements across all breadboards or PCBs of a particular component, wire, etc. For example, if a wire is globally scoped, it should be the only wire color used for a particular case in all scenarios regardless of breadboard or PCB.

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

10.2 Wiring Standards

Wire Color Scheme

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

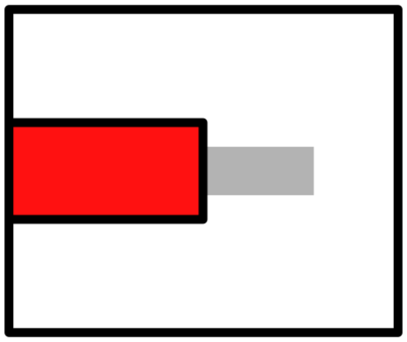


Black

Purpose: ground (GND)

Scope: global

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

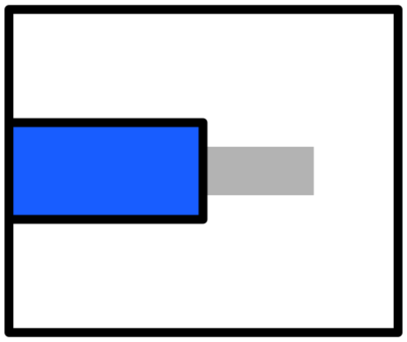


Red

Purpose: +5V power line

Scope: global

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

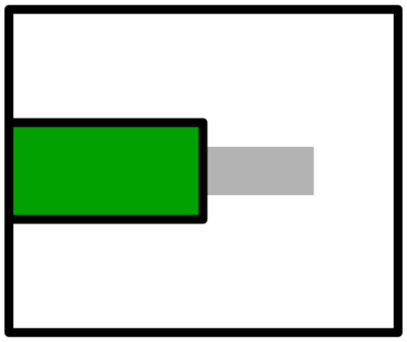


Blue

Purpose: data line, primary

Scope: global

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

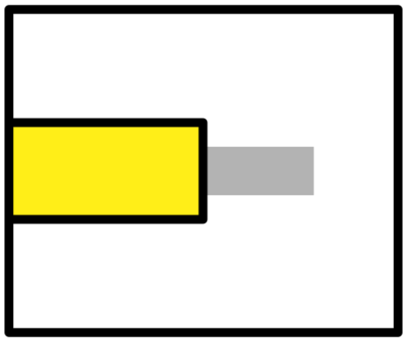


Green

Purpose: address line

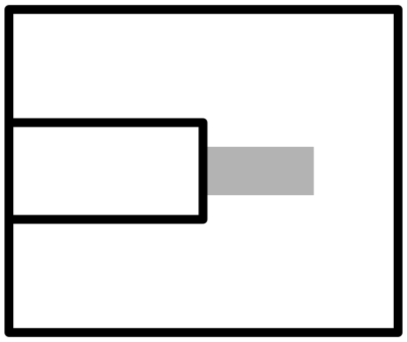
Scope: global

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



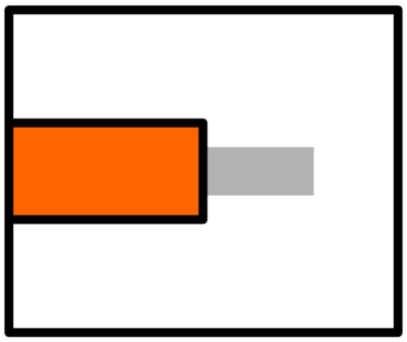
Yellow
Purpose: control line, clock pulse
Scope: global

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



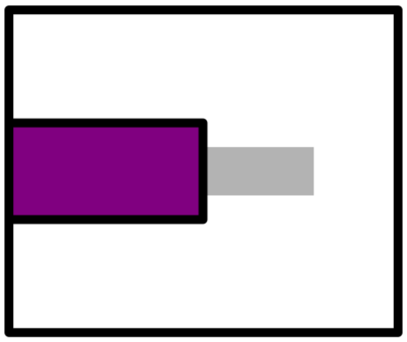
White
Purpose: data line, secondary
Scope: local

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



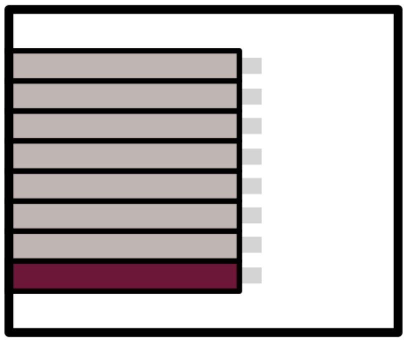
Orange
Purpose: data line, ternary
Scope: local

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



Purple
Purpose: data line, ternary
Scope: local

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



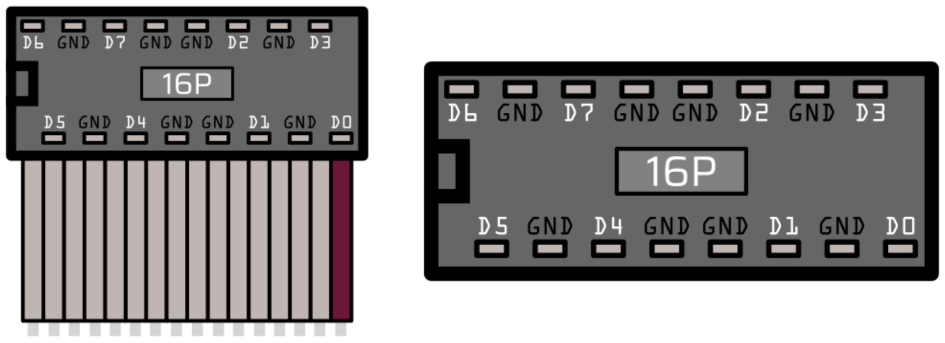
Gray Ribbon Cable

Purpose: bus data line

Scope: global

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

Connector for Bus Data Lines



D6 GND D7 GND GND D2 GND D3
16P
D5 GND D4 GND GND D1 GND D0
D6 GND D7 GND GND D2 GND D3
16P
D5 GND D4 GND GND D1 GND D0

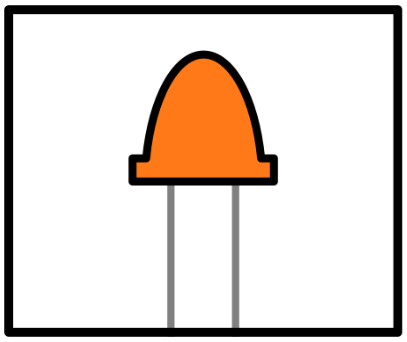
Pin Description

D0-D7	Data lines 0-7
GND	Ground

The primary indicator for the connector is that the zeroth bit line (non-gray wire) must be on the right side of the connector when facing the i281 processor.

10.3 Visualization Standards

An initial 6 colors were purchased for the i281 CPU. Colors are not as restrictive as wires; however, there are some reserved colors.

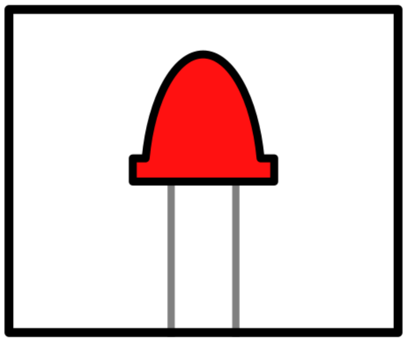


Orange

Purpose: register storage

Scope: global

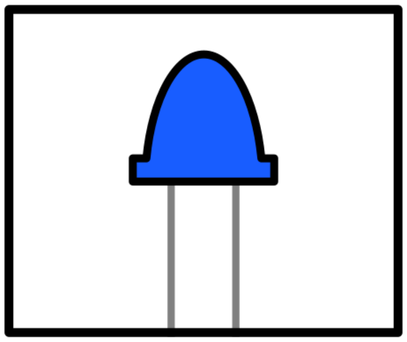
Meant to exclusively be used for representing what is currently stored in the individual registers of the Register File.



Red

Purpose: instruction representation

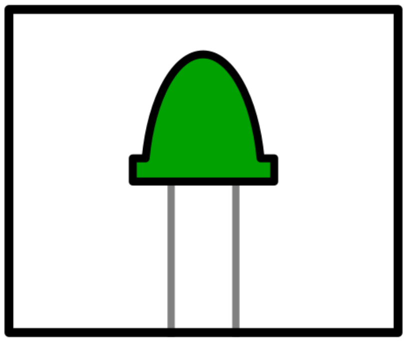
Scope: local



Blue

Purpose: open

Scope: local

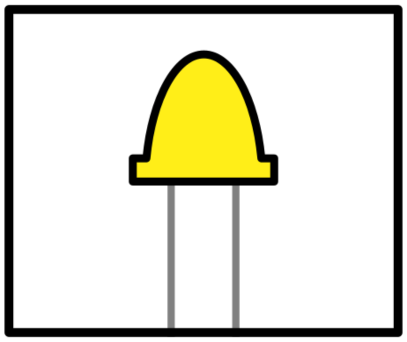


Green

Purpose: program address

Scope: global

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

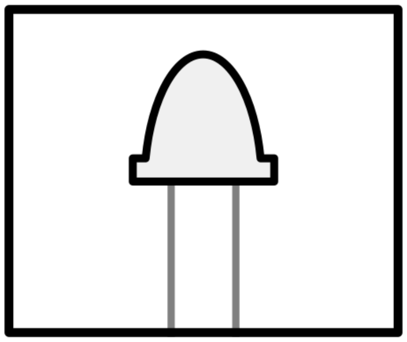


Yellow

Purpose: control line indicator

Scope: global

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



White

Purpose: flags register

Scope: local

10. Appendix B

Team Contract

Team Name _i281 CPU Hardware Implementation_____

Team Members:

1) _Logan Lee_________________ 2) _Braxton Rokos_____________

3) _Brandt Daryl Damman_________ 4) _Grant Nordling______________

5) _Gavin Tersteeg______________

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

   Team meetings will be held every Wednesday night from 5:30pm until 6pm. These meetings will precede the meeting with the client, Dr. Alex Stoytchev, from 6pm to 7pm.

   Team and client meetings are to be held face-to-face unless a popularity of members is unwell or unable to physically meet. In this scenario, an attempt to meet virtually will be made. Additional meetings may be scheduled on other days and times when the project requires such.

   Work meetings between a portion of the team will be scheduled on a needed basis to ensure physical hardware work is performed. Work will be performed in the Senior Design lab in these scenarios.

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

   Texting, Discord, and Email. Texting is reserved for timely communication. The Discord app is for all relevant and non-relevant information about the project. Email will be used for discussions that need to be well-documented and client interactions.

   Gitlab will also be used for issues, reminders, and project updates related to issues.

3. Decision-making policy (e.g., consensus, majority vote):

   For most decisions, a group consensus must be held to pass a decision. For controversial or long decisions, popularity must be met to pass a decision.

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

   During client meetings, Daryl Damman will record information regarding interchanges between client and team. These notes will be stored on the team's Gitlab instance for version controlling and easy accessibility.

   For other meetings, all other members will volunteer on a meeting basis to document information related to project advancement, discourse, and other relevant information. This information will be stored in Gitlab.

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

   All meetings between the client or separately as a team require full attendance. Absences from these meetings are permitted with a notice from each respective absent member with a sound reason.

   Work outside of meetings does not always require the full team unless directed and agreed upon by all members.

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

   Unless noted in advance, where possible, all members are expected to deliver and demonstrate on their respective timelines, deadlines, and assignments promptly.

   Failure to meet deadlines will result in team discussion to resolve the work that has fallen behind. Continued failure to meet the deadlines set will result in an infraction of the contract.

3. Expected level of communication with other team members:

   Team members are expected to monitor communications channels regularly. Members should promptly respond to any questions or inquiries. Additionally, the group should meet in person or virtually at least once a week to share progress and make plans for future work.

Not meeting during a week (as a team, not with the client) must be agreed upon by the full team in writing via text or email.

Client meetings are to be weekly or bi-monthly unless otherwise agreed upon by the whole team and client.

4. Expected level of commitment to team decisions and tasks:

   Team members are expected to dedicate ample time to accomplish the weekly or monthly milestones. Team members are also expected to communicate in team decisions and share any concerns or answer any questions.

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):
   - Daryl Damman has taken the role of organizing the team, contacting the client (Dr. Stoytchev).
   - Logan Lee has taken the role of monitoring scheduling for all meetings and overall testing procedures.
   - Grant Nordling has taken the role of client interaction, testing, project assembly, and quality control for circuits and PCB design.
   - Braxton Rokos has taken the role of testing, breadboarding, PCB design, and PCB routing.
   - Gavin Tersteeg has taken the role of testing and quality control for digital logic.

   Responsibilities regarding the project components will be divided on a milestone basis as the project progresses.

2. Strategies for supporting and guiding the work of all team members:

   Gratitude toward each team member for all accomplishments and achievements. As noted in an earlier section, issues and milestones will be used to track required project work. From those same issues and milestones, members are expected to aid each other, when possible, to support each other.

3. Strategies for recognizing the contributions of all team members:

   Messages in phone texting will be seen by all members to highlight their specific achievement. Emails with the client and/or professor may include notable remarks for one or more members between milestones.

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.
    - Daryl Damman has experience in project management, software development, and assembly architecture. Some mild experience in firmware development.
    - Logan Lee has experience in designing circuits, building circuits, and experience with simulating digital logic.
    - Grant Nordling has experience in PCB testing, ordering parts, chip testing, and enclosure wiring and design;
    - Braxton Rokos has experience in circuit design, PCB testing and layouts, cable design and creation, soldering, and running simulations.
    - Gavin Tersteeg has experience in PCB layouts, digital logic design, and homebrew computer design.

2. Strategies for encouraging and support contributions and ideas from all team members:

    Ensuring we acknowledge all ideas brought up by team members will ensure a positive environment to continue contributing to the team. We will ask for feedback from other team members to ensure everyone voices their ideas.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

    Communication is a major factor of identification and resolution. If there are issues with involvement, all team members must do their part to reduce the barrier of entry or eliminate obstacles.

    A team member who is experiencing issues with the team environment and complications with their ability to contribute should state their concerns via text message or Discord message to the whole team. If a scenario arises where the team member is not comfortable bringing up the problem to the whole team, they must notify two other members of the team to discuss their concerns. The three members involved will then bring forth the concern together.

    In extraordinary circumstances where a member is not comfortable with discussing the concern with any other member on the team for any sound reason, they must contact the TA (Abir Mojumder) for guidance on going forward.

Goal setting, Planning, and Execution

1. Team goals for this semester:
    1. Creating a fully flushed out design document that will thoroughly denote project requirements and considerations for full implementation during the second semester.
    2. Apply newly acquired skills through the project design and honing old skills.

2. Strategies for planning and assigning individual and teamwork:

    We are going to use GitLab to keep track of work done and documentation. We will have a list of milestones and tasks that we can all work towards together when available. We will set some deadlines to keep the project on track. Team members will assign their own work to work towards our milestones.

3. Strategies for keeping on task:

    Weekly objectives with long term goals to ensure we know what we are working on. Milestones will be installed by the team organizer to ensure longer term goals.

    Progress on the physical hardware project should be shared with the whole team, especially if not all members are present, to boost morale and display the achievements the team as a whole is making.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

   Team members that violate the contract in part or whole will be subject to discussion from all team members and follow-up with the client, if necessary. The severity of the infraction is to be determined by all members that have not violated the contract. If the infraction is determined to be minor and should be instead considered an honest mistake, the infraction is to be nullified upon discussion amongst members.

2. What will your team do if the infractions continue?

   Further discussions with the client and team members alike to understand the source of infractions of a specific member will be held. A resolution must be made with the team member who is violating the contract to avoid extraordinary circumstances.

   In extraordinary circumstances, the professor will be involved to mitigate further infractions and discuss the future project involvement of a team member.

****

a) *I participated in formulating the standards, roles, and procedures as stated in this contract.*

b) *I understand that I am obligated to abide by these terms and conditions.*

c) *I understand that if I do not abide by these terms and conditions, I will suffer the consequences as stated in this contract.*

1) __Logan Lee____________________________ DATE __9/10/23________

2) __Braxton Rokos__________________________ DATE __9/10/23________

3) __Grant Nordling__________________________ DATE __9/10/23________

4) __Brandt Daryl Damman____________________ DATE __9/10/23________

5) __Gavin Tersteeg__________________________ DATE __9/10/23________