

Louisiana Tech University

Louisiana Tech Digital Commons

Master's Theses

Graduate School

Summer 8-2020

The Megaprocessor as an Educational Tool Making the Abstract Concrete

Jonathon Beauregard II

Follow this and additional works at: <https://digitalcommons.latech.edu/theses>



Part of the [Computer Sciences Commons](#)

Recommended Citation

Beauregard, Jonathon II, "" (2020). *Thesis*. 44.
<https://digitalcommons.latech.edu/theses/44>

This Thesis is brought to you for free and open access by the Graduate School at Louisiana Tech Digital Commons. It has been accepted for inclusion in Master's Theses by an authorized administrator of Louisiana Tech Digital Commons. For more information, please contact digitalcommons@latech.edu.

**THE MEGAPROCESSOR AS AN EDUCATIONAL TOOL:
MAKING THE ABSTRACT CONCRETE**

by

Jonathon Beauregard II, B.S.

A Thesis Presented in Partial Fulfillment
of the Requirements of the Degree
Master of Science

COLLEGE OF ENGINEERING AND SCIENCE
LOUISIANA TECH UNIVERSITY

August 2020

LOUISIANA TECH UNIVERSITY
GRADUATE SCHOOL

April 22, 2020

Date of thesis defense

We hereby recommend that the thesis prepared by

Jonathon Wayne Beaugard II, B.S

entitled The Megaprocessor as an Educational Tool

Making the Abstract Concrete

be accepted in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science



Dr. Jean Gourd Supervisor of Thesis Research



Dr. Pradeep Chowriappa ,
Head of Computer Science


Members of the Thesis Committee:

Dr. Jean Gourd

Dr. Box Leangsuksun

Dr. Ben Choi

Approved:



Hisham Hegab
Dean of Engineering & Science

Approved:



Ramu Ramachandran
Dean of the Graduate School

ABSTRACT

Computer architecture courses can be difficult for students to engage with and learn from. This is because, unlike most core courses for a computer science student, learning architecture is an abstract process. To address this, universities have implemented methods for teaching course material other than purely descriptive methods. This typically means using simulations to model some aspect of a CPU or FPGA (field-programmable gate array) boards for hands-on experimentation in CPU design. However, there are issues with these tools. Simulations can only cover a few topics well, are prone to being abandoned, and introduce additional abstraction layers. FPGAs, while great for advanced topics and long class projects, are often best suited for senior and graduate level students. Both methods are useful, but neither offers a tangible learning experience, which is what the Megaprocessor can provide. The Megaprocessor is a room sized, general-purpose computer made from discrete components, whose architecture is comprised of primitive logic gates with LEDs on every input and output. The entire circuitry of the Megaprocessor is transparent to the users, with its entire state visible and unabstracted. Because of these properties, it is a great learning mechanism for computer architecture education. The Megaprocessor is a tool for hands on and project-based learning that can be used to span the learning gap between simulations and FPGAs.

APPROVAL FOR SCHOLARLY DISSEMINATION

The author grants to the Prescott Memorial Library of Louisiana Tech University the right to reproduce, by appropriate methods, upon request, any or all portions of this Thesis. It is understood that “proper request” consists of the agreement, on the part of the requesting party, that said reproduction is for his personal use and that subsequent reproduction will not occur without written approval of the author of this Thesis. Further, any portions of the Thesis used in books, papers, and other works must be appropriately referenced to this Thesis.

Finally, the author of this Thesis reserves the right to publish freely, in the literature, at any time, any or all portions of this Thesis.

Author _____

Date _____

DEDICATION

This thesis is dedicated to my wife, Rachel. I would not be where I am today without her, academically or professionally. She helped me to see something in myself that I would never have seen otherwise. She inspired me to pursue academia and to be a better version of myself. I am forever thankful.

TABLE OF CONTENTS

ABSTRACT.....	iii
APPROVAL FOR SCHOLARLY DISSEMINATION	iv
DEDICATION	v
LIST OF FIGURES	xii
LIST OF TABLES	xiv
ACKNOWLEDGMENTS	xv
CHAPTER 1 INTRODUCTION	1
1.1 The Current State of Computer Architecture Courses.....	1
1.2 An Overview of the Megaprocessor	2
1.3 Building a Megaprocessor	2
1.4 The Organization of This Thesis	3
CHAPTER 2 THE MEGAPROCESSOR.....	4
2.1 Details of the Megaprocessor	4
2.1.1 Design	4
2.1.2 Gates	5
2.1.3 Boards	5
2.1.4 Modules.....	6
2.1.5 Frames.....	6
Arithmetic and Logic Unit (ALU)	7
State and Status	10
General Purpose Registers	11

Input and Instruction Decoding	12
Special Purpose Registers	13
Memory	14
Control and I/O	15
Connectivity	16
2.1.6 Architecture.....	16
Instruction Set	17
Instruction Cycle.....	20
2.1.7 Highlights.....	21
2.2 The Need for the Megaprocessor.....	21
2.2.1 Computer Architecture is an Abstract Class	22
2.2.2 Computer Architecture is a Difficult Class.....	23
2.2.3 Computers Are Perceived As a ‘Magical Black Box’	23
2.3 What the Megaprocessor Provides	25
2.3.1 Broken Down to Raise Visibility.....	25
2.3.2 Uniquely Hands-On	26
CHAPTER 3 BACKGROUND	27
3.1 Guidelines	27
3.1.1 Curricula Recommendations.....	27
3.1.2 Class Objectives.....	28
The Science of Computing III.....	28
Digital Design	28
Computer Architecture.....	29
Advanced Computer Architecture	29
3.2 Simulations	29

3.2.1	Benefits of Simulations	29
3.2.2	Gaps with Simulations	30
3.2.3	Simulation at Louisiana Tech University	31
3.3	FPGAs.....	31
3.3.1	Excessive for Some Topics.....	31
3.3.2	Great for Advanced Topics.....	31
3.4	Architecture	32
3.4.1	Changing Course Content.....	32
3.4.2	Using RISC-V.....	32
3.5	Novel Approaches.....	33
3.6	Megaprocessor	33
3.6.1	Suitable Subjects.....	33
	The Science of Computing and Digital Design	33
	Computer Architecture.....	34
3.6.2	Unsuitable Subjects.....	34
	Advanced Computer Architecture	34
CHAPTER 4 MEGAPROCESSOR BUILD PLANNING.....		35
4.1	Establish Communication.....	35
4.2	Review Existing Documentation	35
4.3	Create New Documentation.....	36
4.4	Find and Correct Errors	36
4.5	Place Orders.....	36
4.6	Creation of Simulations	37
CHAPTER 5 BUILDING.....		41
5.1	Soldering.....	41

5.2	Testing	41
5.3	Laser Paneling.....	43
5.4	Metal Framing.....	43
5.5	Assembling	44
5.6	Awakening	46
CHAPTER 6 PRELIMINARY RESULTS AND CONTRIBUTIONS		47
6.1	Board Designs.....	47
6.2	Parts List	48
6.2.1	Board Components.....	48
6.2.2	Changes to LEDs	51
6.2.3	Soldering	53
6.2.4	Module Components	53
6.2.5	Frame Components	54
6.2.6	Part Count	55
6.2.7	Metal Framing.....	56
6.3	Test Plans	60
6.4	Build Plans	61
6.5	Digital Design Simulations.....	62
6.6	Open Repository for Distribution	62
6.7	Researching Application to Education	63
CHAPTER 7 DISCUSSION.....		64
7.1	Digital Design	64
7.2	Computer Architecture	65
7.2.1	Teacher Driven Lectures.....	65
7.2.2	Student Interaction	66

7.2.3	Soldering Labs	66
7.3	Accessibility Solutions	67
7.3.1	Divide into Labs.....	67
7.3.2	Photo and Video Repository	67
7.4	Schematics and Design	68
7.5	Megaprocessor Simulation	68
7.6	Measuring the Effectiveness of the Megaprocessor in the Classroom	69
7.6.1	Pedagogical Design.....	69
	Compare Sections	70
	Optional Additions.....	70
CHAPTER 8 CONCLUSION AND FUTURE WORK		71
8.1	Conclusions.....	71
8.2	Contributions of this Thesis	72
8.3	Future Work on Megaprocessor	72
8.3.1	Additional Registers.....	73
8.3.2	Bus Connection.....	73
8.3.3	Larger Bus.....	73
8.3.4	Instruction Set	74
8.3.5	Redesign RAM.....	74
8.4	Future Work on Megaprocessor I/O	75
8.4.1	Motherboard.....	75
8.4.2	Display	75
8.4.3	Keyboard Input	76
8.4.4	Storage	76
8.5	Operating System.....	76

8.6	Future Work on RISC-V.....	77
8.7	Expand Circuit Simulations.....	77
8.8	Remote Access.....	78
8.9	Cyber Security.....	78
8.10	Game Design.....	78
8.11	Conclusions Regarding Future Work.....	79
8.12	Final Remarks.....	79
APPENDIX A	Diagrams.....	80
A.1	Megaprocessor Connectivity Diagrams.....	80
BIBLIOGRAPHY	86

LIST OF FIGURES

Figure 2-1: A panorama photo of the Megaprocessor.	4
Figure 2-2: 4x2 AND/OR PCB.....	5
Figure 2-3: Flag Calculation module.	6
Figure 2-4: Layout of the processor frames. This excludes RAM and I/O.	7
Figure 2-5: The ALU frame. There is a space between the two modules that separates them.	8
Figure 2-6: The ALU module diagram. Shows the boards and their connections within the module. Note: the bus connections are not present in module diagram.	9
Figure 2-7: The ALU control module diagram. Note: the bus connections are not present in module diagram.....	9
Figure 2-8: State and Status frame.....	10
Figure 2-9: General Purpose Registers frame.....	11
Figure 2-10: Input and Instruction Decoding frame.	12
Figure 2-11: Special Purpose Registers frame.....	13
Figure 2-12: RAM frame.	14
Figure 2-13: Control and I/O frame.....	15
Figure 2-14: Typical processor layout.	17
Figure 2-15: 0x40 is the add opcode. The ‘D’ means ADD R1, R3. The full instruction is 4D.....	18
Figure 2-16: Information on the divide operation.	19
Figure 2-17: Examples of Divide operation.	20
Figure 2-18: Arcade stick connected to Control and I/O.....	26

Figure 4-1: The 3-input AND simulation. The top section is the circuit, the bottom is the encapsulated component.	38
Figure 4-2: The Full Adder simulation. The top section is the circuit, the bottom is the encapsulated component.	39
Figure 4-3: Transistor level RS Flip Flop.....	40
Figure 5-1: Test of laser printing of module graphic.....	43
Figure 5-2: The front of the State Machine Frame.....	44
Figure 5-3: The back of the State Machine Module.....	45
Figure 6-1: Isometric view of metal frame.....	57
Figure 6-2: Exploded view of metal frame.....	58
Figure 6-3: Full build of materials, including price.....	59
Figure 6-4: Screenshot of 3-D model of General Purpose Registers frame.....	60
Figure 6-5: Build guide for the Services Board. Includes parts number, the original description from Mr. Newman, and additional information to make the build process a self-guided one.....	62
Figure A-1: State and Status connectivity diagram.....	81
Figure A-2: Instruction and Decoding connectivity diagram.....	82
Figure A-3: Special Purpose Registers connectivity diagram.....	83
Figure A-4: ALU connectivity diagram.....	84
Figure A-5: General Purpose Registers connectivity diagram.....	85

LIST OF TABLES

Table 6-1: Margin Calculations. This is not the full table, it is only a small segment. ..	49
Table 6-2: Parts list for the 16-bit register board. The component names are simplified to fit here, and the full part names can be found in the documents in the open repository.....	50
Table 6-3: Build information for all 16-bit register boards needed.	51
Table 6-4: Red LEDs: 2mA, 1.7V.	52
Table 6-5: Green LEDs: 20mA, 2.1V.	52
Table 6-6: Blue LEDs: 20mA, 3.5V.	53
Table 6-7: Boards list for General Purpose Registers module.....	54
Table 6-8: Sample from parts list outside of board components.	55
Table 6-9: Sample from part counts for all boards.	56

ACKNOWLEDGMENTS

First, I would like to thank my teacher and advisor, Dr. Jean Gourd. Without him this project would not have happened, and I would likely not have pursued my master's degree. Dr. Gourd helped secure funding, spearhead the project, and guide me through this journey.

I would also like to extend my gratitude to Mr. James Newman, the first person to design and build the Megaprocessor. His permission and encouragement allowed this project to happen. Mr. Newman provided assistance and additional documentation throughout the build process.

Finally, I am sincerely grateful for the help of my good friend and former manager, Dr. David Lippert. His input on this thesis was vital. He has mentored me through the years, and I am thankful for him. Without his support, pursuing my master's degree while working full-time would have been impossible.

CHAPTER 1

INTRODUCTION

1.1 The Current State of Computer Architecture Courses

Computer Architecture can be a difficult subject for students. The classes are often taught from dryly written books and slide decks featuring many diagrams that many students often just memorize. Course material is abstract and because students are used to hands-on programming they often struggle [9]. The subject matter is difficult to understand and learn for many students. Unlike other core courses, where coding and hands-on projects are used, computer architecture lacks an intuitive project-based learning approach. Even if the course does not use a purely descriptive teaching method, there is still the issue of finding appropriate tools that allow students to interact with computer architecture concepts. Most solutions used at universities are typically simulations and FPGAs. Simulations of ranging functionality, from digital circuit design to full CPU emulation, have been designed and used for teaching. FPGA boards can be used for class projects and labs; however, there are issues with both tools. Simulations are often specialized to cover one or two topics, and FPGAs are only well suited for advanced topics. The Megaprocessor offers a solution for these issues.

1.2 An Overview of the Megaprocessor

The Megaprocessor is a general-purpose computer that is designed and built to maximize learning potential. All the components of the processor are built by hand using primitive logic gates, with each gates' input and output displayed with LEDs. This means that the entire processor is fully accessible and visible. All functionality and the full state of the processor are exposed with no interpretation layer. The Megaprocessor is a great educational tool for teaching computer architecture because it provides a learning experience that students get nowhere else. It truly allows them to see how a microprocessor works.

1.3 Building a Megaprocessor

The process of building a Megaprocessor at Louisiana Tech University is ongoing. Mr. James Newman was contacted for permission to use his designs at Louisiana Tech University. He was supportive, gave his permission, and provided additional documents. Unfortunately, there were many gaps in the documentation that prevented the immediate purchase of parts. This necessitated a reverse engineering of the Megaprocessor. The build phase is ongoing, and several inevitable setbacks occurred that have delayed the project. The build is anticipated to take at least another year before the Megaprocessor is fully operational. It took over two years of work to get to the build phase. All the work that was done to create a comprehensive parts list and build guide can be found in chapter six. Additionally, all contributions and documentation will be placed in an open repository to save others the time and effort involved in reverse engineering the Megaprocessor.

1.4 The Organization of This Thesis

In the second chapter the Megaprocessor is introduced in greater detail, the issues with computer architecture education that it solves are examined, and its most significant qualities are presented. In the third chapter the relevant research that others have done is discussed, as well as the tools currently in use and how they fall short. The fourth chapter contains details about the reverse engineering phase. The fifth chapter covers the build process, both what has been done and has yet to be done. Chapter six discusses the output of this project and specifies the contributions which advance the state of the art. Chapter seven discusses how the Megaprocessor can be applied to various computer architecture courses. Finally, chapter eight contains conclusions and ideas for future work.

CHAPTER 2

THE MEGAPROCESSOR

2.1 Details of the Megaprocessor

2.1.1 Design

The Megaprocessor is a general-purpose processor, built using discrete transistors, resistors, and LEDs. Instead of a processor on a silicon wafer, the Megaprocessor is built out of PCBs (printed circuit boards), ribbon cables, and metal framing. Processors are normally made to take up as little space as possible, while the Megaprocessor takes up 20 square meters. It is fully handmade, with over 200,000 solder joints, and over 100,000 components. It consists of seven frames, each of which contains modules that encapsulate some logical CPU component as shown in (**Figure 2-1**) [27].



Figure 2-1: A panorama photo of the Megaprocessor.

Each module is a collection of boards that are decomposed down to primitive logic gates, other than the Control and I/O frame.

2.1.2 Gates

Like all processors, the Megaprocessor's primitive logic gates are made from transistors and other components. What is unique about the Megaprocessor is that each gate has an LED on every input and output. This gives unprecedented insight into how the processor works. Every bit can be traced, and the state of the circuits within the processor are fully readable. The gates are also large enough to be read clearly by an observer, which makes for an exceptionally low density of components.

2.1.3 Boards

PCBs (printed circuit boards) are the basic building block of the Megaprocessor. The circuitry is silkscreened on each PCB so that students and observers can read them and understand each component's connections, an example is shown in (**Figure 2-2**) [28]. A silkscreen is typically a white graphic overlay showing the circuit design and labels that is printed on a PCB. The PCBs are soldered by hand and have varying levels of complexity. There are PCBs that contain exactly one gate, and there are PCBs that are more complex, made of many gates and other components.

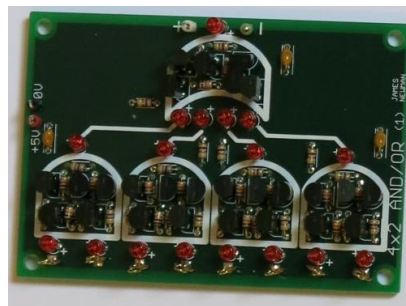


Figure 2-2: 4x2 AND/OR PCB.

2.1.4 Modules

The boards come together to create modules which function as a component of a processor, like an ALU. There are fourteen modules not counting RAM and I/O. Each module is comprised of gates, ribbon cables, and wires to connect the gates and other PCBs, as well as a module graphic. The module graphic contains the entire circuit diagram showing all connections and text describing the functionality of the module, an example can be seen in **(Figure 2-3)** [29].

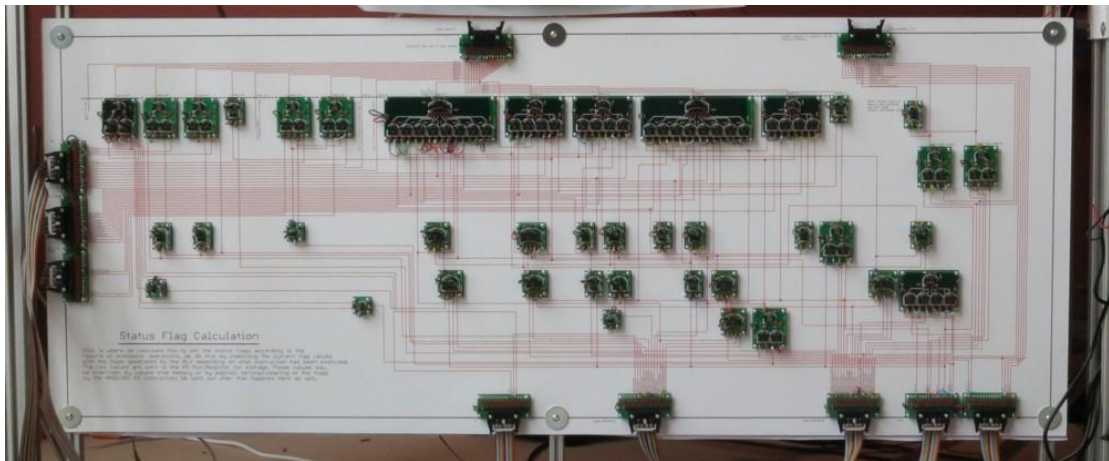


Figure 2-3: Flag Calculation module.

2.1.5 Frames

The Megaprocessor is comprised of seven frames connected by ribbon cables that carry the bus signals. The frames come together to create the completed Megaprocessor. Each frame has a different function and is described below. The layout of the processor frames are shown in **(Figure 2-4)** [30].

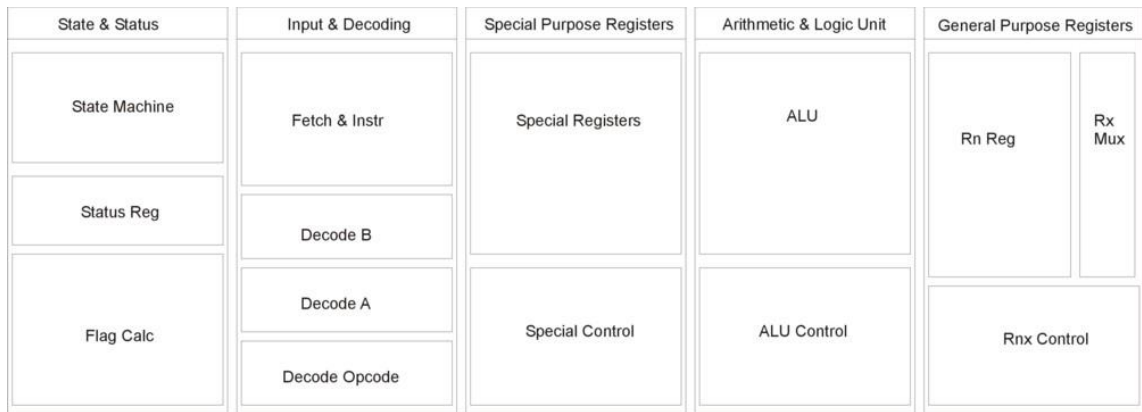


Figure 2-4: Layout of the processor frames. This excludes RAM and I/O.

Arithmetic and Logic Unit (ALU)

This frame is divided into two modules as shown in **(Figure 2-5)** [27]. The ALU module is at the top of the frame **(Figure 2-6)**, and the ALU control module is beneath it **(Figure 2-7)** [29]. Decoding the operation code to instruct the ALU on what operation to perform takes place on another frame. The negative, zero, and overflow flags generated by the ALU are sent to the Status Flags Calculation Module. A connectivity diagram for the ALU frame can be viewed in **(Figure A-4)** [27].

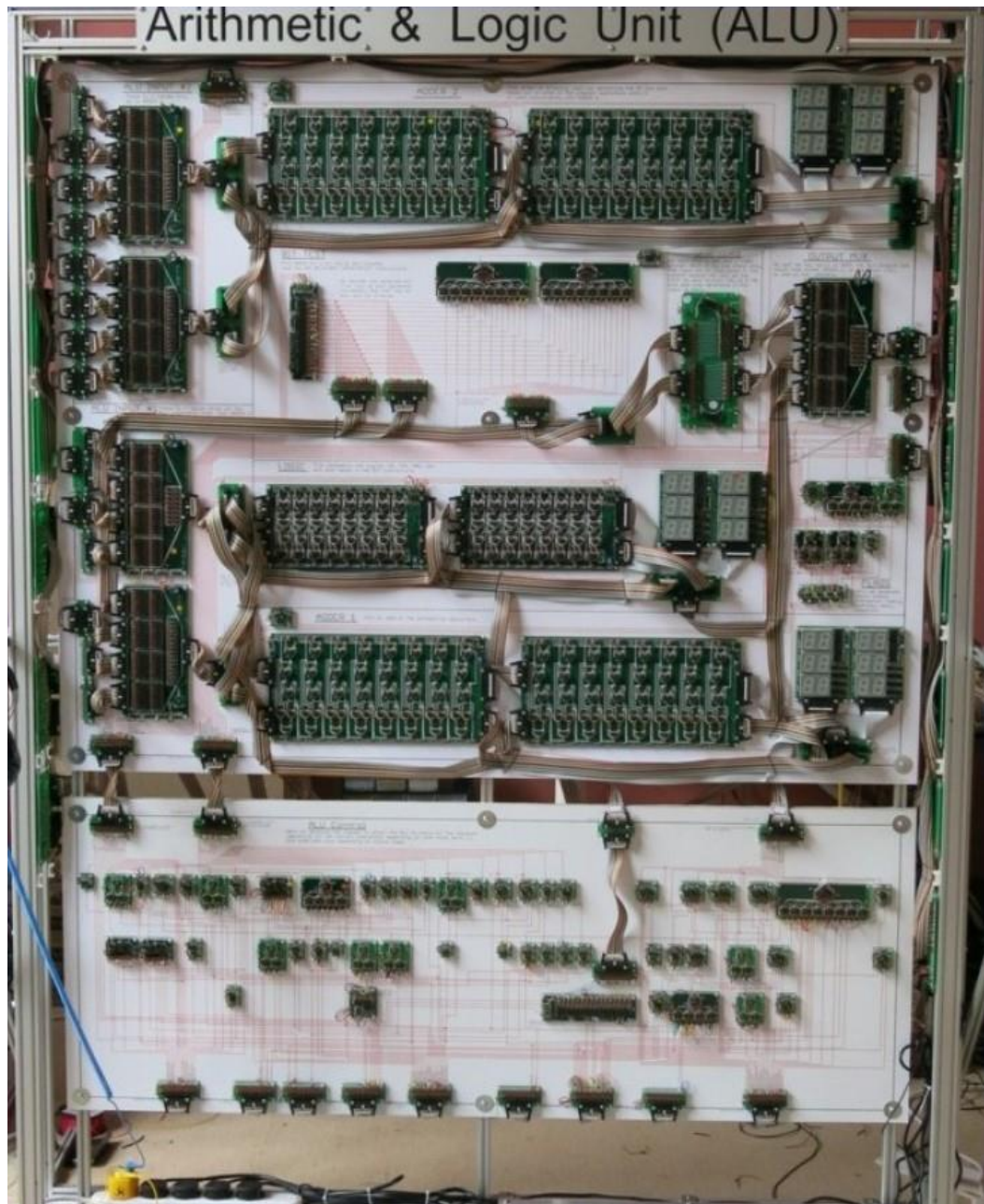


Figure 2-5: The ALU frame. There is a space between the two modules that separates them.

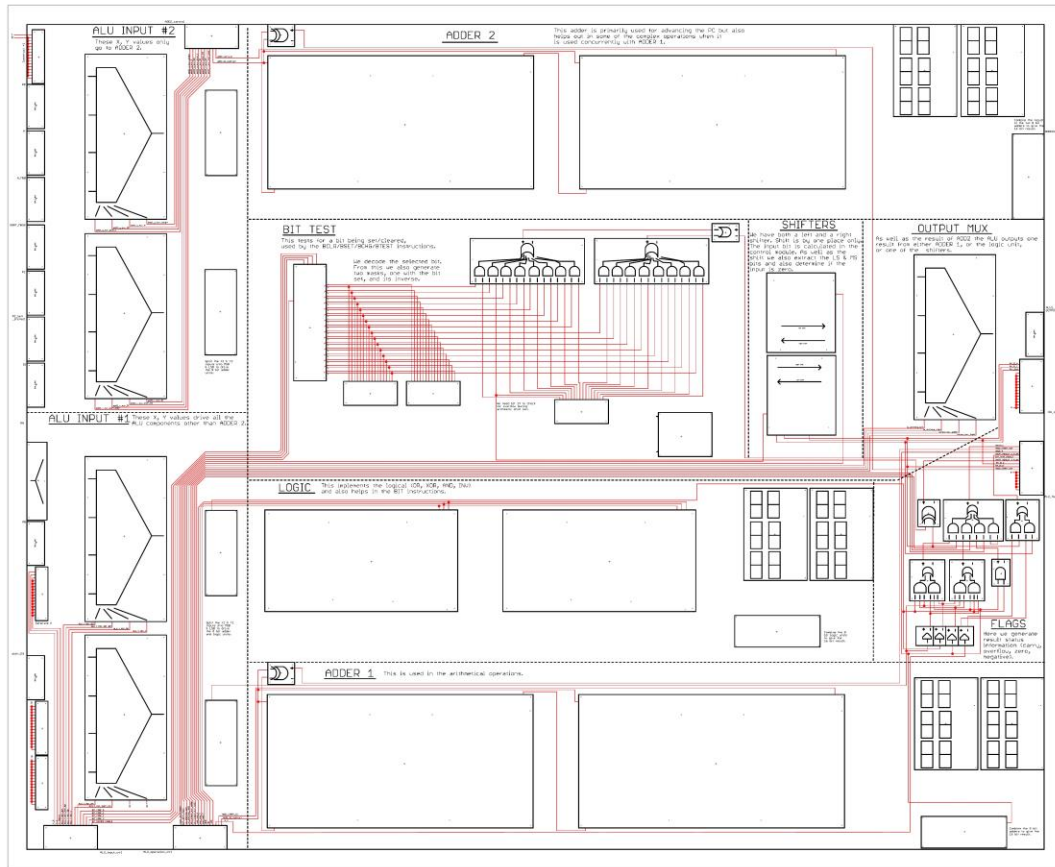


Figure 2-6: The ALU module diagram. Shows the boards and their connections within the module. Note: the bus connections are not present in module diagram.

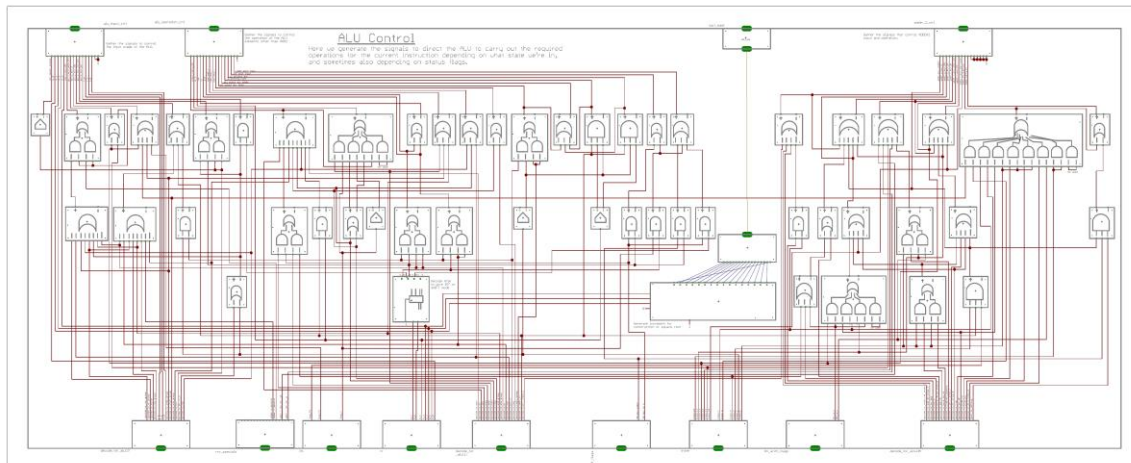


Figure 2-7: The ALU control module diagram. Note: the bus connections are not present in module diagram.

State and Status

The State and Status frame is composed of three modules (**Figure 2-8**) [27]. The top module is the State Machine module, which keeps track of what the CPU is currently doing and orchestrates what state the processor will go into next given the current state, flags, and outputs. The middle module is the Status Register which contains bits that inform the State Machine module. The bottom is the State and Flag Calculation module, which calculates the bits stored in the Status Register. These modules come together to create the state machine of the processor [31].

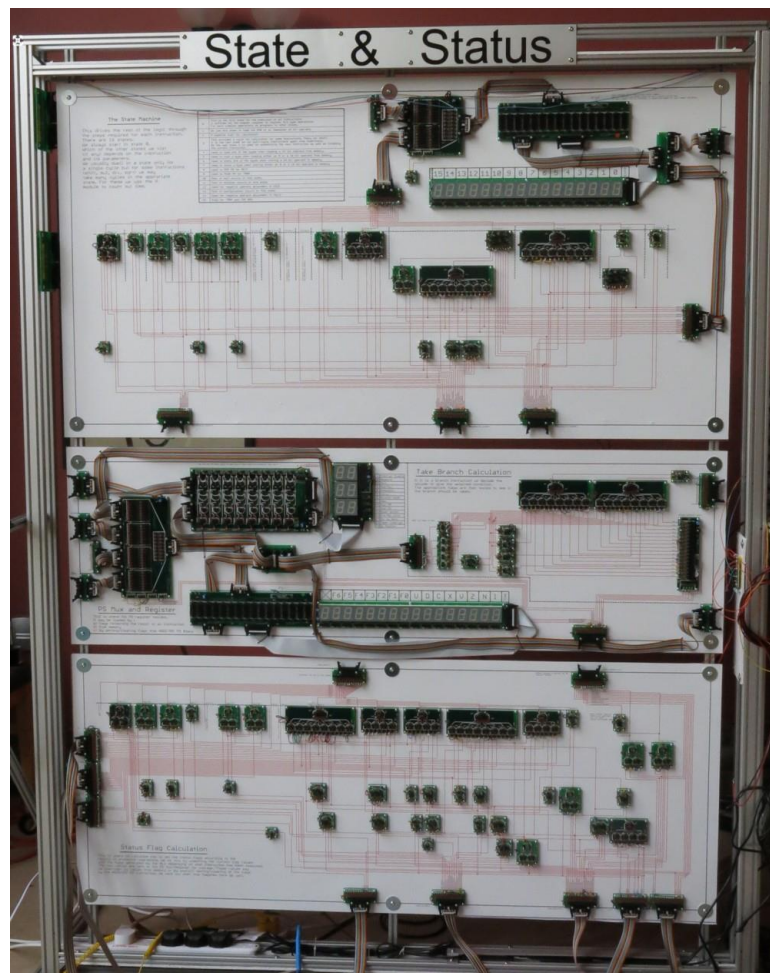


Figure 2-8: State and Status frame.

General Purpose Registers

This frame consists of three modules: the General Register module which contains four registers, the General Register Multiplexer module which contains the multiplexers for the registers, and the General Register Control module as seen in (Figure 2-9) [27].

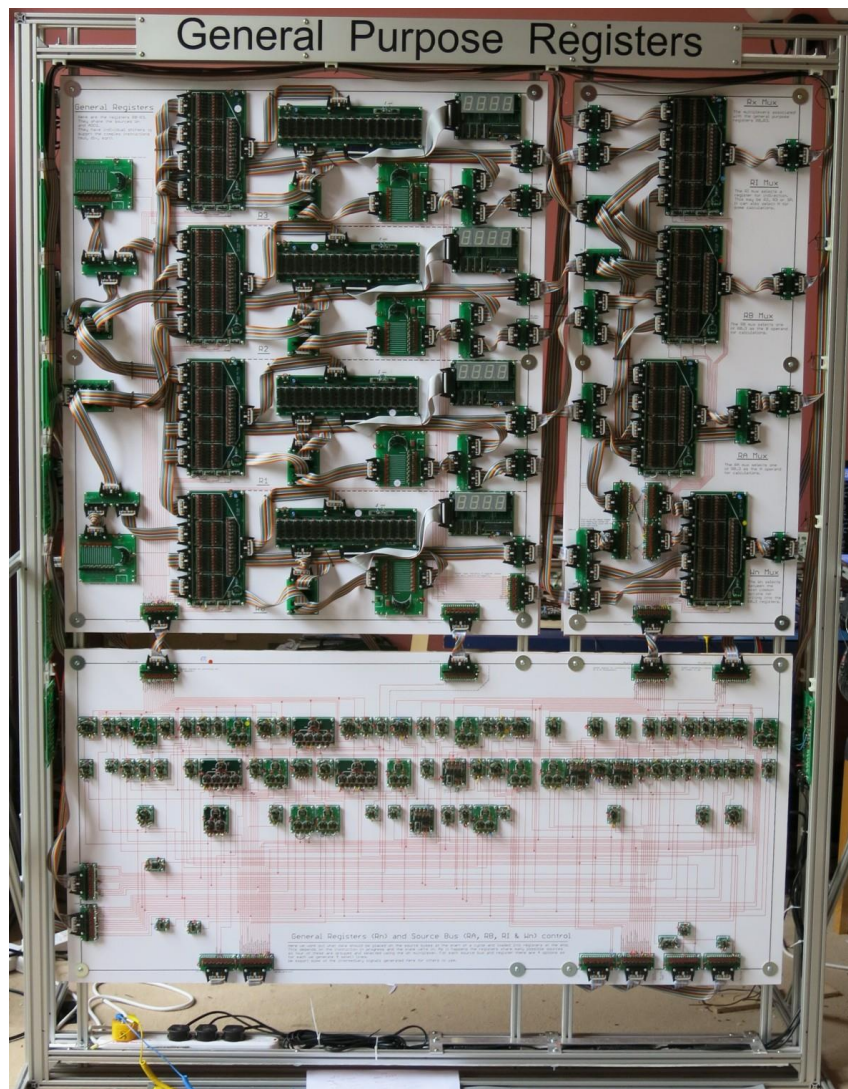


Figure 2-9: General Purpose Registers frame.

Input and Instruction Decoding

The Input and Instruction Decoding frame shown in **(Figure 2-10)** consists of four modules, one which handle decoding the instructions and another that decodes the operation codes (or opcodes). The other two modules generate control signals for the other frames. Based on the current opcode, different control signals will be sent. These control signals prepare the other frames for the instruction. For example, it may make different registers available to the ALU.

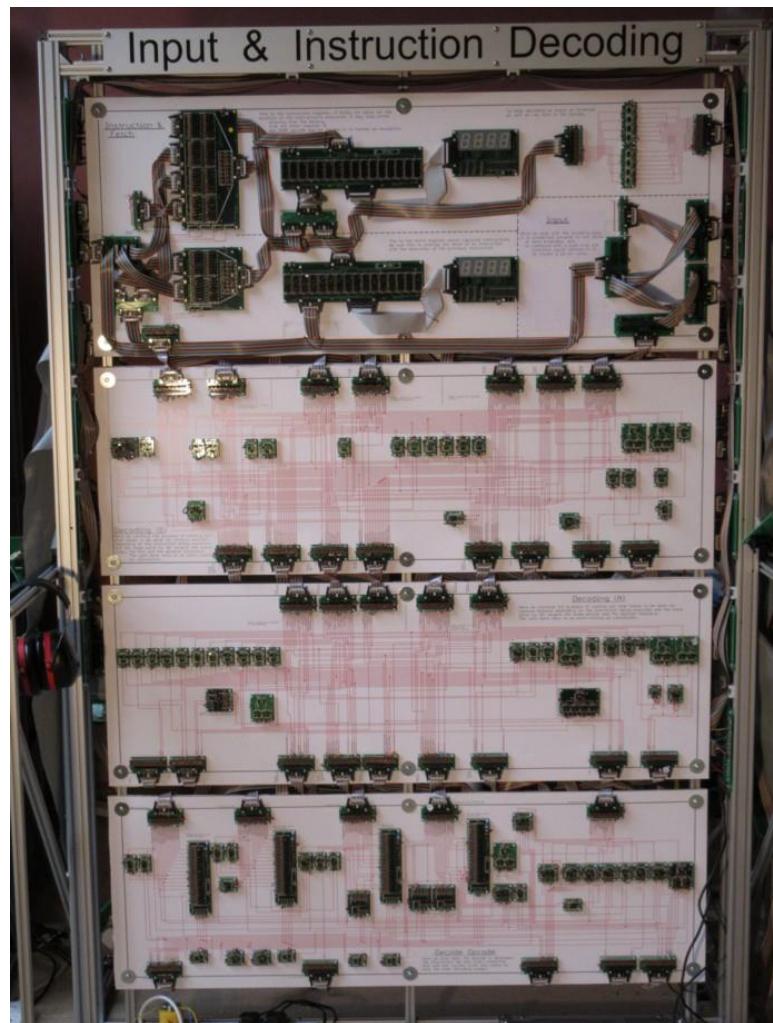


Figure 2-10: Input and Instruction Decoding frame.

Special Purpose Registers

The Special Purpose Registers frame consists of two modules seen in (Figure 2-11) [27]. The Special Registers module contains the stack pointer, external address, and program counter as well as their associated multiplexors. The lower module is the Special Register Control, which generates control signals.

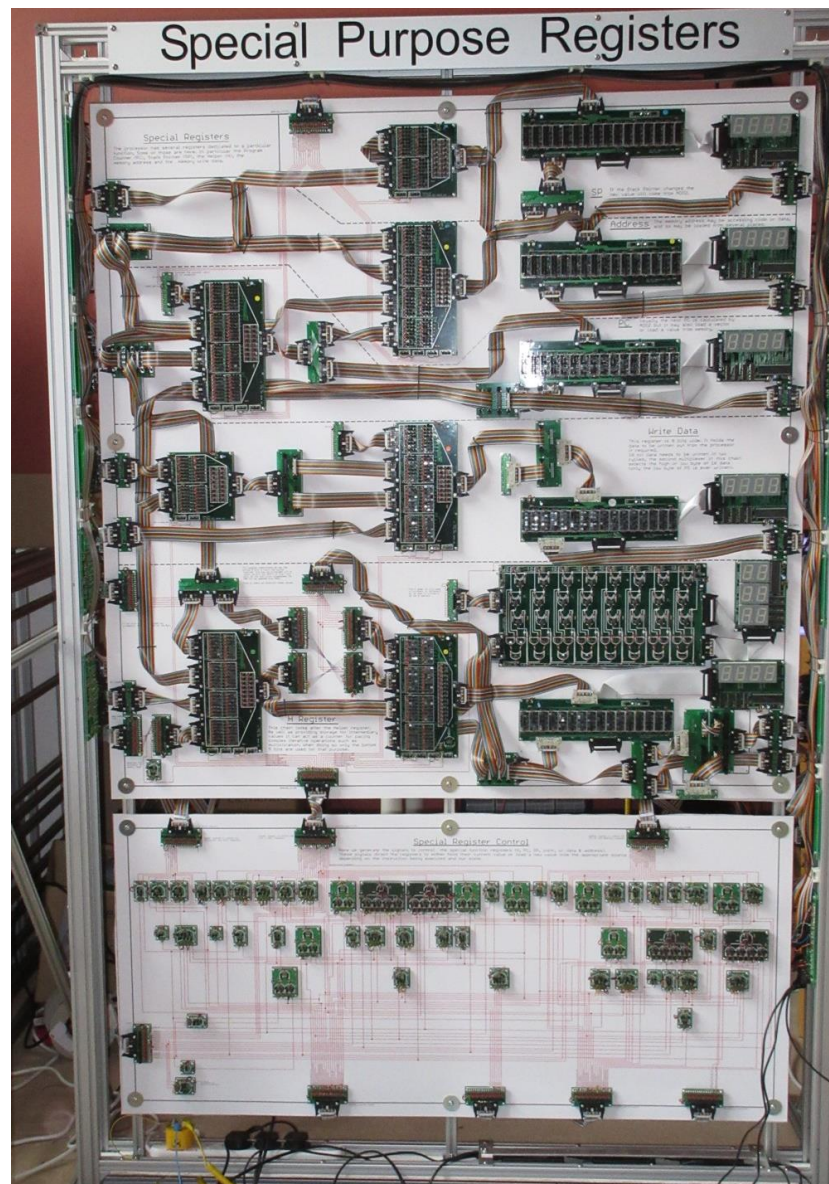


Figure 2-11: Special Purpose Registers frame.

Memory

The Memory frame shown in **(Figure 2-12)** contains 256 Bytes of RAM [27]. Each of the 2,048 total bits of RAM is illuminated with an LED. It can also function as a dot matrix display of 32 by 64 pixels, depending on the program and desired functionality.



Figure 2-12: RAM frame.

Control and I/O

More complex parts were used in the design and creation of the Control and I/O frame shown in **(Figure 2-13)** [27]. Part of the function of this frame is to be a side-by-side simulation of the Megaprocessor. The simulation compares its simulated output to the Megaprocessor's actual output. Should there be a difference, it halts the machine and dumps register values for debugging purposes. There is also a laptop that connects to the frame over serial interface to a control board to load programs to run on the Megaprocessor. The Megaprocessor simulation can be downloaded and used by students or faculty on their own computers for testing and learning.

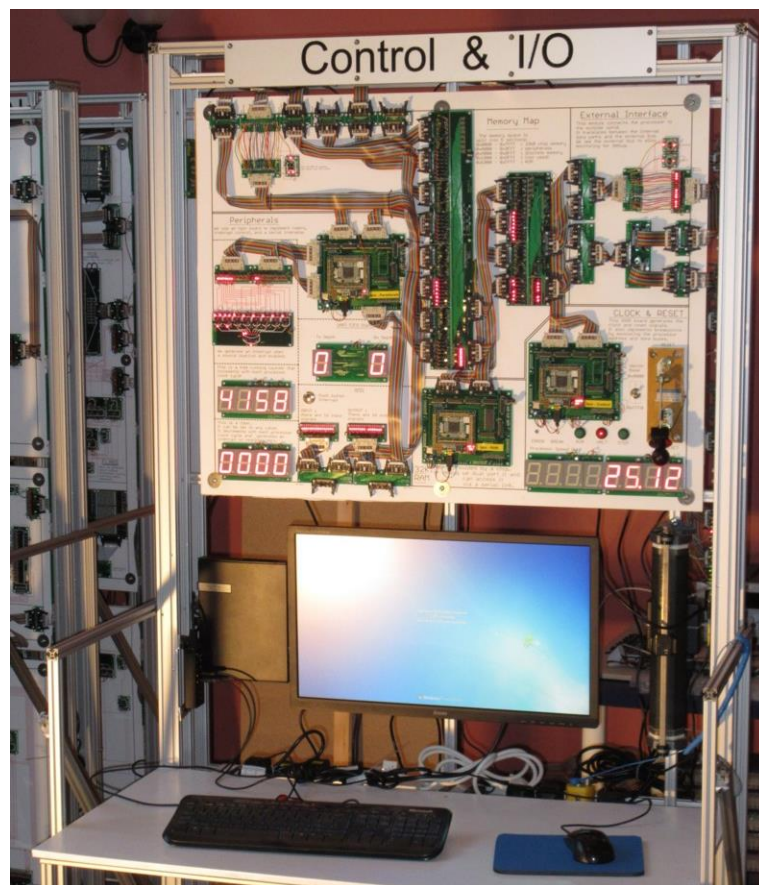


Figure 2-13: Control and I/O frame.

Connectivity

There is a diagram showing the connections between major components for the processor frames located in **(Figure A-1)**, **(Figure A-2)**, **(Figure A-3)**, **(Figure A-4)**, and **(Figure A-5)** [27]. Each frame will have its own image, and to create the full diagram, they can be laid side-by-side in the correct order.

The frames are connected to each other in series (see **Figure 2-4**) [30]. The States and Status frame connects to the Input and Decoding frame, and any data directed to other frames down the line get forwarded and manipulated by the frames it travels through.

2.1.6 Architecture

The Megaprocessor is a 16-bit processor with four general purpose registers (R0, R1, R2, R3) and three specialized registers: the program counter (PC), the stack pointer (SP), and the processor status register (PS). The components within the processor are connected using multiplexors that the Megaprocessor can configure to change the data paths depending on different instructions. The general layout of the processor is described in **(Figure 2-14)** [30].

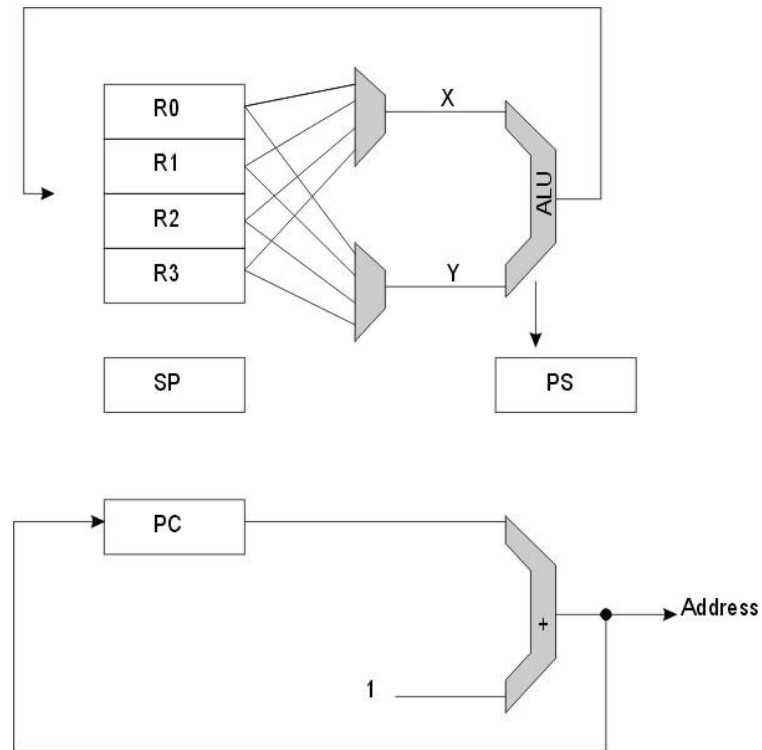


Figure 2-14: Typical processor layout.

While most instructions can use any register, there are slight restrictions to the process flow. For example, RAM based instructions have limitations on register usage. R0 or R1 may only be used as source or destination for the data being moved to and from RAM, and R2 or R3 store the index of the RAM location.

Instruction Set

The instruction set consists of 8-bit opcodes with up to two bytes of additional data. This additional data designates which registers are being used, where the new value will be stored, or if it is a sub-operation. There are a total of 256 possible instructions, and all are allocated for some use. An instruction is read from memory and sent to be decoded. Depending on the instruction, the processor will be configured to fulfill the

instruction. For most instructions, the first register is the destination and the second is the source.

An example of one of the instructions is the addition opcode 0x40 seen in (**Figure 2-15**) [32]. The two bytes that follow the opcode designate what two registers are being added, with the first register being where the new value is ultimately stored. Of note for this instruction is that it takes one clock cycle to complete. In the example, R3 is being added to R1. R1 contains the hex value 848B and R3 contains the value A6AB. These values added together are 12B36. This is greater than what can be stored in the register, so the overflow flag (V) and carry flag (C) are turned on, and 2B36 is stored in R1.

ADD RA, RB

Operation: $RA + RB \Rightarrow RA$

Format:

7	6	5	4	3	2	1	0
Code = 0x4, ADD				RB		RA	

Length/Cycles: 1 byte, 1 cycle

Condition Codes:

I - Not affected

N * Set if MSB set, cleared otherwise

Z * Set if result is zero, cleared otherwise

V * Set if overflow is generated. Cleared otherwise

X * As per carry bit.

C * Set if carry is generated. Cleared otherwise

Example:

R0[0000] R1[848B] R2[FFFF] R3[A6AB] PC[0075] SP[233B] PS[43(...NID.)]

0074: 4D: ADD R1,R3

R0[0000] R1[2B36] R2[FFFF] R3[A6AB] PC[0076] SP[233B] PS[79(CXV..ID.)]

Figure 2-15: 0x40 is the add opcode. The 'D' means ADD R1, R3. The full instruction is 4D.

Multiplication, division, and square root operations are implemented by iteration.

The following figures (**Figure 2-16**) and (**Figure 2-17**) provide information on the

division instruction [32] Unlike addition, which gets the opcode of 0x40, division is only comprised of two sub operations: 0xFA and 0xFB.

DIVS & DIVU

Operation:

```

if (R1 == 0)
    execute TRAP 1
else
    Quotient (R0 / R1) ⇒ R2
    Remainder (R0 / R1) ⇒ R3
end if
if (signed operation)
    abs( R1 ) ⇒ R1
end if

```

Description: If R1 is zero then TRAP 2 is taken. Otherwise R0 / R1 is calculated. The quotient is put in R2, the remainder. The arithmetic condition flags are cleared. There are two ways of defining signed division, the difference lying in whether or not negative remainders are allowed. Both are implemented, the method used is controlled by the D flag.

	D = 0	D = 1
13/3	Q = 4, R = 1	Q = 4, R = 1
13/-3	Q = -4, R = 1	Q = -4, R = 1
-13/3	Q = -4, R = -1	Q = -5, R = 2
-13/-3	Q = 4, R = -1	Q = 5, R = 2

In all cases the following holds:

$$\text{Dividend} = (\text{Quotient} \times \text{Divisor}) + \text{Remainder}$$

Format:

7	6	5	4	3	2	1	0
0xF, miscellaneous				1	0	1	sgn

Encoding:

	0	1
sgn	unsigned	signed

Length/Cycles:

1 byte, division by zero is 7 cycles, unsigned is 18 cycles, signed is 19 cycles

Condition Codes:

I - Not affected

N, Z, V, X, C - Set to zero

Figure 2-16: Information on the divide operation.

(Figure 2-17) illustrates several examples of the divide instruction [32]. All four registers are used in this instruction: R0 is the numerator, R1 is the denominator, R2 is the quotient, and R3 is the remainder.

```

Examples:
R0[000D] R1[0003] R2[0000] R3[0000] PC[0162] SP[8321] PS[02(.....N...)]
0161: FA:DIV.U
R0[000D] R1[0003] R2[0004] R3[0001] PC[0163] SP[8321] PS[00(.....)]
At the start of the operation R0=13, R1=3. After the calculation R2=4 (quotient) and
now R3=1 (remainder).

R0[000D] R1[FFFD] R2[0000] R3[0000] PC[0164] SP[8321] PS[00(.....)]
0162: FB:DIV.S
R0[000D] R1[0003] R2[FFFC] R3[0001] PC[0165] SP[8321] PS[00(.....)]
At the start of the signed division operation R0=13, R1=-3. After the calculation
R2=-4 and now R3=1.

R0[FFF3] R1[0003] R2[FFFC] R3[FFFF] PC[016A] SP[8321] PS[40(.....D.)]
0169: FB:DIV.S
R0[FFF3] R1[0003] R2[FFFB] R3[0002] PC[016B] SP[8321] PS[40(.....D.)]
At the start of the signed division operation with the signed encoding flag on (D flag
in PS), R0=-13, R1=3. After the calculation R2=-5 and now R3=2.

```

Figure 2-17: Examples of Divide operation.

Instruction Cycle

The Megaprocessor implements the same instruction cycle that traditional processors do: fetch, decode, read from RAM, and execute. The program counter is to be incremented at the end of the fetch cycle.

Additionally, there is some instruction pipelining that occurs within the Megaprocessor. Simple processors only perform one step in the clock cycle at a time. This is inefficient as each step of the clock cycle uses different portions of the processor, leaving the rest of the processor completely unutilized. A technique that modern processors utilize is instruction pipelining, where multiple steps of a clock cycle occur at

once. The processor is therefore doing work continuously. It has one stage of pipelining, in which the next instruction is fetched while the current one is being executed.

2.1.7 Highlights

The Megaprocessor has a user-controlled clock speed that can be slowed down so that each step takes seconds to perform. It has a maximum speed of 20kHz. The maximum speed is limited due to the design of the simple ripple adder combined with the type of discrete transistors used. Mr. Newman suggested using a different type of resistor in the ALU which should raise the maximum speed a bit [30].

Additionally, the Megaprocessor can be halted at any time by pushing a button that allows the current state of the processor to be observed and analyzed. Observers may also step through the program to explore each step of the execution by pressing another button on the Control and I/O frame. Once close examination is finished, the program can be continued with the push of a third button. There are system executions that take many instruction steps to complete, such as the divide operation. Even these basic operations can be stepped through and observed at the student's pace.

Finally, the Megaprocessor is the size of a room, 10 meters long and 2 meters tall. It would be possible to arrange the frames to surround its observers, as if the students are inside of the processor. It holds the Guinness World Record for the largest microprocessor [8].

2.2 The Need for the Megaprocessor

As previously mentioned, there are shortcomings with the current methods used in computer architecture education. For the students who take them, classes can be abstract and difficult in ways that they have not previously experienced. Strong students who

have benefited from hands-on methods in previous classes may find the abrupt switch to abstract learning difficult to adjust to [26].

There are teaching alternatives that go beyond books and diagrams that have been designed to try and resolve this issue. Academic methods for teaching computer architecture beyond a descriptive method will be discussed in chapter three.

2.2.1 Computer Architecture is an Abstract Class

Computer architecture, unlike most other core computer science classes, is full of concepts that cannot be explored by using software or writing code. Most other topics, including operating systems, object-oriented programming, computer networking, data mining, and cloud computing, can be learned in an interactive way by writing code or through direct interaction. It is not so easy to observe an ALU inside of a real processor and fully explore it as it operates.

Computer Architecture can be explored with software, but not natively. This software is a layer that can only illustrate concepts it was written to highlight. Because of this abstraction layer, some concepts may be hidden or difficult to grasp. Different software packages highlight different concepts and have different gaps, but none are complete. Simulations have been made that model various aspects of processors and computer hardware; however, these simulations often only cover one topic and can be difficult to run. Code is not a substitute for physical interaction in computer architecture. This concept will be discussed further in chapter three.

Because these classes discuss abstract subject matter, students often have difficulty engaging with them and thereby find them uninteresting [9]. The Megaprocessor provides them an accessible processor model to study that can help

motivate them to develop an interest in computer architecture. A processor that students can see, interact with, and touch with their own hands, can go a long way toward making computer architecture classes less abstract.

2.2.2 Computer Architecture is a Difficult Class

Because the material in these classes is abstract, it can be difficult for students to learn. This may be especially true for students who excel with project-based learning. Generally, the classes are structured as lectures, and the material is presented through diagrams that students often memorize. Instead of using traditional learning methods, the Megaprocessor provides access to a fully transparent CPU so that students can ‘get their hands dirty.’ It supplements lectures by allowing teachers to walk through the actions of the processor, highlight important information, and allow students to explore the lecture material, making the abstract more concrete. The stop button on the Megaprocessor allows an instructor to pause the processor in a desired state and ask students what the CPU is going to do next, moving the instruction from a lecture to an engaging discourse. Instructors can also guide students to follow the bits as they move through the data and control paths, anticipating where a bit is going or what action it is going to cause. There are more opportunities for student driven discovery that leads to better grasp of the material [40].

2.2.3 Computers Are Perceived As a ‘Magical Black Box’

It is not readily possible for a student to open a computer and tinker with the CPU. Getting any real idea of what a CPU does at any given time is generally unfeasible. Modern operating systems and processors run at blistering speeds, and the CPU mechanisms are abstracted into oblivion. Because of this, it is easy to think of a CPU as

a brain and leave it at that. The issue with this analogy is that it gives students an excuse to avoid learning complex concepts. If it is as complex as a human brain, which science still does not fully understand, why struggle to understand it? The analogy is problematic, both because the CPU is not like a brain, and because it is not out of reach for students to understand. Processors are fully designed by talented teams of humans, and the building blocks used to create modern processors can be learned. The Megaprocessor can help students understand the fundamentals of a processor. With it as a visible, concrete model, students can keep that insight with them throughout their careers because the method of learning was so memorable and remarkable. The Megaprocessor's architecture is drastically simpler than what is in use today; however, it contains the building blocks of modern microprocessors. Most classes utilize simplified, academic models because they are easier to learn from. At the same time, these models should be as close as possible to the real thing. The Megaprocessor meets both standards.

Unlike software that can be downloaded, extracted, decompiled, and experimented on, hardware is more complex to interact with. It is best understood through hands-on interaction; however, ironically the physical components of a computer can be harder for students to interact with than the software. While students are unable to modify the Megaprocessor once it is built and running, they can run different programs to experiment with its functions, as well as change the running speed and stop the processor to get a better visual understanding of how bits flow through the system. They can see how it works, and not merely read about it. It provides an immersive experience for students.

2.3 What the Megaprocessor Provides

The Megaprocessor provides a truly hands-on experience that allows students to learn about computer architecture. It provides a physical, working, general purpose processor that is tangibly accessible to students. The model of the processor is decomposed into discrete components, in a form factor that students can interact with.

2.3.1 Broken Down to Raise Visibility

The entire Megaprocessor is broken down to the transistor level, with an LED on every input and output of each component. Any segment of the processor can be approached and viewed at this level of detail. Gates and transistors are encapsulated to boards and modules, but nothing is hidden or abstracted away. If you get close enough, you can see all the discrete parts that go into a component, whether it is a register, a bit of RAM, or the ALU. Unlike most others, this processor can be slowed down to a crawl. Programs can be halted and stepped through to the granularity of a single clock cycle. Every action of the Megaprocessor executing the code can be observed and understood.

It is possible to make an 8-bit processor on a series of breadboards; however, those use far more complicated parts and are not normalized to the same level as the Megaprocessor. The chips on the breadboard creating the CPU can be complex. For example, just a single integrated circuit alone may contain the ALU. It is not possible to observe those chips and see exactly what they are doing. They can be useful for gaining a high-level view of what is occurring in a processor, but the same can be said about the Megaprocessor.

2.3.2 Uniquely Hands-On

The Megaprocessor can provide a completely hands on way to learn about computer architecture, from basic to complex concepts. Students can not only interact with it by writing and running code, but also by seeing details of the processor down to every gate. There is nothing hidden or abstracted away, other than the I/O. The main interface is an arcade stick that contains a joystick and buttons that serve as input to the Megaprocessor as seen in (**Figure 2-18**) [33]. This input can be used to interact with instructor provided or student written programs. Assembly code can be written by the student and then run on an actual general-purpose processor.

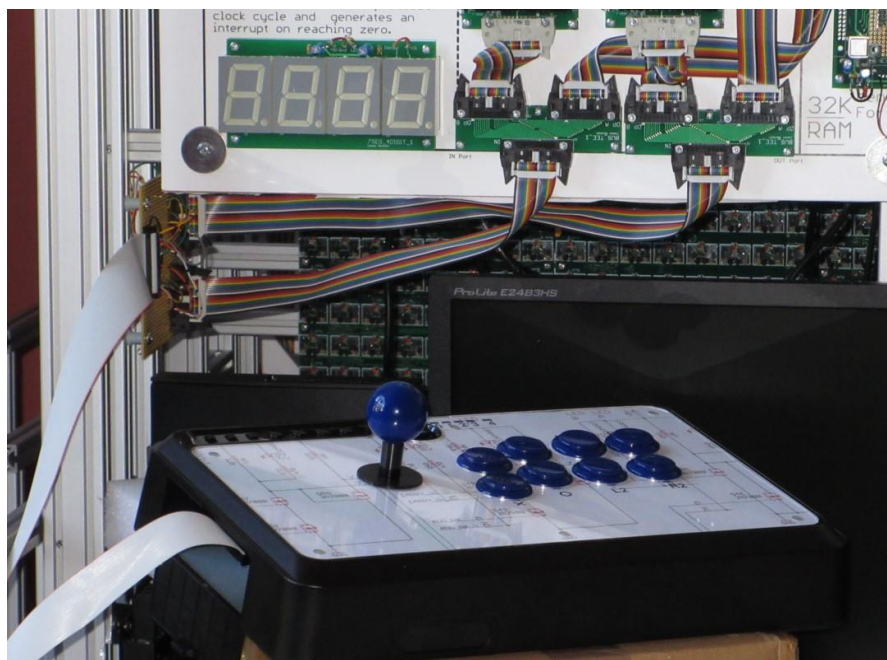


Figure 2-18: Arcade stick connected to Control and I/O.

CHAPTER 3

BACKGROUND

3.1 Guidelines

Before the different solutions for improving computer architecture education are compared, the grounds of comparison for the solution's effectiveness must be established. Most research done in computer architecture education discusses both computer engineering and computer science courses. Louisiana Tech University (the school that the author of this thesis attends) does not offer a computer engineering major. Because of this, the scope of classes will be restricted to three computer science courses offered at Louisiana Tech University. There is a newer class offered that will be covered as well.

3.1.1 Curricula Recommendations

The ACM, together with IEEE, issued a set of curricula recommendations for computer science [6]. Within the document that they released, there are many knowledge areas within the 'body of knowledge' that they deem should be covered [2]. Of interest is the Computer Architecture and Organization knowledge area, with acronyms AR within the guidelines document, and CAO in the larger body of research papers. Within the knowledge area of CAO there are many topics and learning outcomes. The knowledge

area of CAO and its topics can be used as a metric against the Megaprocessor and help guide discussion of content in university courses.

3.1.2 Class Objectives

The Science of Computing III

The Science of Computing III is a class at Louisiana Tech University that covers topics from CAO. It is part of the ‘Living with Cyber’ computing curriculum, which is the freshman year program that all computer science and cyber engineering students participate in. It has a project-based learning paradigm, and the contents of the courses are open to review here: <https://coes.latech.edu/living-with-cyber/>. Concepts from CAO are first introduced in The Science of Computing I; however, the first significant instruction on these topics take place in Science of Computing III.

According to the course syllabus [14], the following topics from CAO are covered: the fundamentals of signed and floating-point number; character representation in computers; and a basic understanding of how a computer is made (e.g., ALU, CPU, memory, I/O). This course covers topics other than CAO, and what it contains is very introductory.

Digital Design

This course is typically taken in a student’s sophomore year in the undergraduate program. According to the course catalog [22], the following topics from CAO are covered: introduction to digital design techniques, Boolean algebra, combinational logic, minimization techniques, simple arithmetic circuits, programmable logic, sequential circuit design, registers, and counters.

Computer Architecture

This course is typically covered in a student's junior or senior year in the undergraduate program. According to the course catalog [23], the following topics from CAO are covered: architecture and organization of computer systems, including the processor, control unit and microprogramming; computer arithmetic; memory hierarchy and memory management; input/output; and instruction sets.

Advanced Computer Architecture

This course is typically covered in the graduate program. According to the course catalog [24], the following topics from CAO are covered: pipeline systems design, processor design techniques (concepts, analysis, performance comparison, implementation, and commercial processors), memory system design, and interconnection media.

3.2 Simulations

Simulations of digital circuitry or CPU operation have become the go-to solution for use in classrooms and labs in CAO courses. Instructors and institutions have often defaulted to simulations because they have become accepted as better than a purely descriptive approach [40]. There are many types of simulations used for teaching computer architecture, ranging from simulations of simple digital circuitry up to full computer emulation [35].

3.2.1 Benefits of Simulations

Simulations can be used as great tools to teach the fundamentals of computer architecture and digital design [9,15]. Simulations for entry level topics, such as circuit design, allow students to experiment and learn basic concepts without having to use other

more complex tools that do not typically offer additional benefits [19]. These simpler simulations are often easier to setup and use, and serve as a learning tool with very low user friction [43]. Due to the complex and difficult nature of teaching computer architecture, attempts to develop other methods are abandoned for a simulation approach because it is more feasible [18,42].

3.2.2 Gaps with Simulations

Using computer simulations instead of a purely lecture based approach is an improvement; however, simulations typically have limitations that prevent them from being optimal for teaching. Simulations usually fall within two groups: specialized -- which do one or two things well -- and broad -- which give a better high-level view [19,35]. The broader, more feature rich simulations are harder to use, and many do not go into the detail desired; while simpler, more focused simulations have limited use [21,41]. It can be difficult to translate knowledge from a simulation to real hardware because simulations can be imperfect, simplified, and erroneous. During research for this thesis, many simulations displayed a pattern where development would halt after some period of time, leading to a build up a bugs and incompatibility issues when attempting to run the simulations [7,10,15,16,20,43]. Large, feature rich simulations are planned and development is started, but no update or release occurs [7]. There are countless simulators to choose from, which makes evaluating them daunting. In fact, many do not meet CAO learning outcomes [19]. Students may also become frustrated if more than one simulation is used. This is due to the difficulties of managing and operating finicky simulations, which may be necessary to cover learning gaps [19].

3.2.3 Simulation at Louisiana Tech University

A high-level CPU simulation is currently used for Louisiana Tech University's computer architecture course. It allows students to write, modify, and run assembly code. However, this type of simulation does not allow for any observation of the workings of a processor. Additionally, the simulation does not allow for any student exploration in CPU design.

3.3 FPGAs

Programs with more money at their disposal may look to FPGA boards to help students learn CAO topics.

3.3.1 Excessive for Some Topics

Without a sufficient background in computer architecture, students can find it difficult to use FPGAs as a learning medium without being frustrated [19]. Most successful adoptions of FPGAs as a teaching tool are aimed at senior undergraduate and graduate level students [19,21,37]. Due to the nature of FPGAs, they do not help in highlighting the link between system operations and underlying hardware circuitry. The problem with FPGAs is that they still introduce an abstraction layer, and while they help students apply advanced computer architecture concepts, learning may be hampered by the additional abstraction [20]. Additionally, FPGA boards become out of date and must be regularly maintained, replaced, and upgraded [19,39].

3.3.2 Great for Advanced Topics

FPGA boards are good for designing a CPU for an advanced, course long project, but not so much for teaching basic or fundamental topics. Advanced students prefer hands on learning and are not frustrated with the FPGAs because they have enough

background courses completed to leverage them effectively [21]. FPGAs can be leveraged to teach more advanced concepts, such as parallel architectures, performance measurements, caching and memory management, pipelining, and more [37]. Students can get hands-on experience with these concepts without having the large overhead needed to use advanced simulations. This learning continues the process of demystifying the computer and helps students understand the complexities of the computer.

3.4 Architecture

Changing the architecture taught in CAO courses has become an increasingly popular option to refresh the material [5,13,25,36]. However, changing the architecture in lectures or in project material would not address student learning success in the same way that the Megaprocessor does.

3.4.1 Changing Course Content

Changing the architecture only changes the content of the course. The tools and methods being used to teach the course remain the same whether they are simulations or utilize FPGAs. Because the course still relies on these tools, the deficiencies that come with using them still exist.

3.4.2 Using RISC-V

It has become very popular to switch over to RISC-V architecture for the computer and CPU model [13]. A list of tools and universities that use them can be found here: <https://riscv.org/educational-materials/> to further showcase. Most importantly, almost all universities listed as using RISC-V, if specified, are still using simulations and FPGAs. They typically use a simulation of a RISC-V based CPU [5,13,25] or FPGAs to design RISC-V based CPUs [36]. The issues with learning

through simulations and FPGAs still exist, even if the architecture has been switched to RISC-V.

3.5 Novel Approaches

Educators and universities continue to search for better ways to teach computer architecture. One university took issue with how simulations can be difficult to setup and operate. This university restored and now uses over 200 old computers to explore concepts in computer architecture [41]. Another university took issue with simulators, stating that while they are useful and one of the best approaches for teaching computer architecture, they fail to convey concrete concepts. This university now uses Nintendo DS systems as a learning platform [38]. This further illustrates the educational gap between simulations, FPGAs, and achieving learning objectives in CAO.

3.6 Megaprocessor

The Megaprocessor can be used to fill the learning gaps left by other tools. It can become part of a larger learning pathway where simulations are used for basic level topics and digital design, the Megaprocessor is used for introductory classes on CPUs and computer architecture, and FPGAs are used for advanced topics and large projects.

3.6.1 Suitable Subjects

The Science of Computing and Digital Design

Because the Megaprocessor is built from transistors, and no obfuscation is occurring, it can be leveraged for basic and foundational concepts in digital design. It is possible to view the various gates and the circuits they are composed of. This provides students with a simpler presentation to build foundational skills so that the scale and complexity of the Megaprocessor does not overwhelm them.

Computer Architecture

The Megaprocessor is a fully functional and self-contained CPU. Because of this, it has the complexity necessary for a computer architecture class. Students can write code, test it with the Megaprocessor simulation, and then run their code on the Megaprocessor. The completed design and schematics are fully accessible, easy to read, and transparent. Because of this, topics including the building blocks of a CPU and CPU design can be taught well. There is no abstraction layer between the student and the underlying architecture of the CPU that there is with simulations and FPGAs. Students can learn the CPU model and its components while having access to the real computer it is based on.

3.6.2 Unsuitable Subjects

Advanced Computer Architecture

Because the Megaprocessor is not easily modified, it cannot be used as a tool for student project-based learning on CPU building. Also, it cannot be used to cover topics in Advanced Computer Architecture such as pipelining, multicore, or advanced memory configurations. Admittedly, these topics are better covered with FPGA based projects. Students will be able to experiment with different processor designs and advanced features using FPGAs, as graduate students typically have the background necessary to use them efficiently. The Megaprocessor may be suitable for this course when new PCBs are being designed and created to upgrade it, which is further explained in chapter eight.

CHAPTER 4

MEGAPROCESSOR BUILD PLANNING

4.1 Establish Communication

The first step to build a Megaprocessor “clone” at Louisiana Tech required establishment of communication with the creator and designer of the original Megaprocessor, Mr. James Newman. Mr. Newman was kind enough to give his blessings and offered to assist in any way that he could. There was a great deal of communication between Louisiana Tech University and Mr. Newman in the initial months, often with a significant delay due to time zone differences (Louisiana Tech University is in Louisiana, USA, while Mr. Newman is in Cambridge, UK).

4.2 Review Existing Documentation

As the documentation published on Mr. Newman’s website, www.megaprocessor.com, was examined, it was discovered that there was a significant lack of documentation on the design and methods used to build the Megaprocessor. Furthermore, there would need to be a significant reverse engineering of the design before the build could proceed. Correspondence was maintained during this time, and Mr. Newman was gracious enough to provide many additional notes and supporting documents. Significant effort was put in to planning the project from the very beginning. Unfortunately, issues continued to arise despite careful and detailed plans.

4.3 Create New Documentation

All the PCBs schematics were inspected, and all PCB and other images were reviewed in order to create a complete list of parts that needed to be ordered. There were many additional parts required that would not be part of the PCBs, and much work was done to ensure that nothing was missed. Detailed notes were compiled and cross-examined by the computer science department chair. The better part of a year and a half was dedicated to reverse engineering and documenting improvements before it was possible to place orders for parts. By the end of the reverse engineering phase, thorough documents (largely in the form of spreadsheets) containing everything needed to build the Megaprocessor were created. All these documents will be part of the open repository detailed in chapter six.

4.4 Find and Correct Errors

During the reverse engineering process, issues with some of the PCB designs were discovered. Some of these defects were documented by Mr. Newman; however, they were only noticed during his build phase, and he corrected them with manually soldered wires. The circuit design files were corrected before the orders took place – one of the contributions to the Megaprocessor.

4.5 Place Orders

Placing orders was a monumental effort. Mr. Newman provided a certificate showing that one firm in the U.K. held the rights to manufacture the Megaprocessor boards. The total cost of the PCBs was approximately \$6,000 US. For the electronic components, a bidding process was required by the university. The cost of electrical components was approximately \$30,000 US. The hardware was obtained through a

university contract with Grainger. The total cost of hardware was approximately \$10,000 US. In total, the Megaprocessor build at Louisiana Tech University cost approximately \$50,000 US (so far).

4.6 Creation of Simulations

During the reverse engineering phase, something unexpected occurred. Upon reviewing the circuit designs, it became apparent that a better way to showcase them to students other than using the schematic files was necessary. As a result, a set of simulations were created (another contribution to the Megaprocessor) that utilize an online simulation tool called CircuitJS (see <https://www.falstad.com/circuit/circuitjs.html>, for example). A set of simulations for teaching fundamentals and gates was created. Additionally, several simulations of PCBs from the Megaprocessor were designed. The CircuitJS simulation allows for the design of complex circuits. A circuit design can be encapsulated into a component. These new components provide the functionality of the circuit but are easier for the simulation to run. The new component can then be selected like any other provided component and can be combined with other parts to create a more complex circuit. Like the rest of the outputs of the project, these simulations will be made available in an open repository.

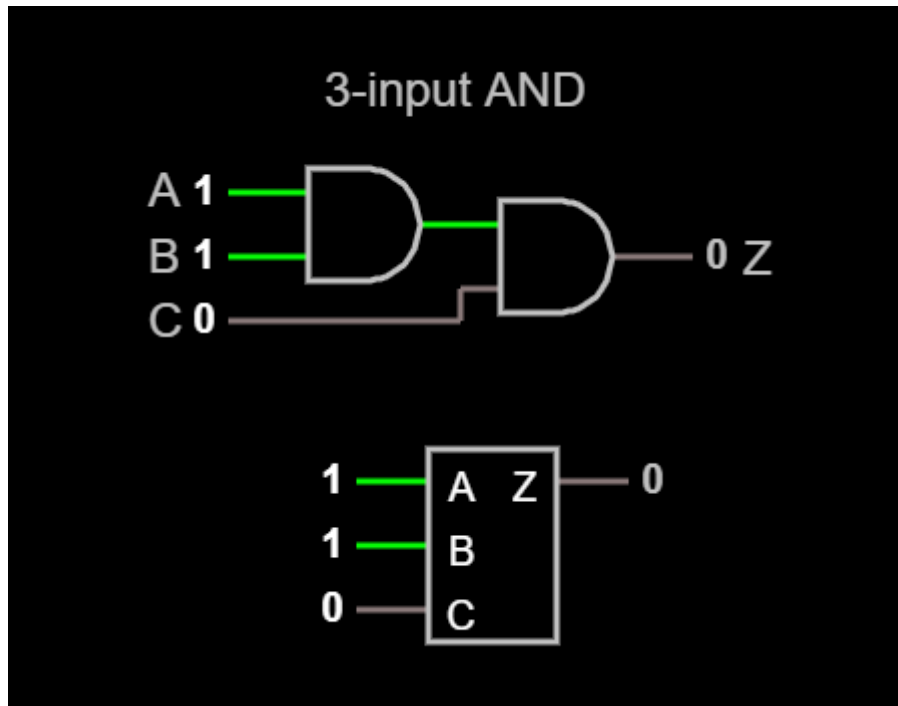


Figure 4-1: The 3-input AND simulation. The top section is the circuit, the bottom is the encapsulated component.

An example of one of these simulations is a 3-input AND gate as shown in (Figure 4-1), that contains a circuit for the gate and a component of the encapsulated circuit displayed below it.

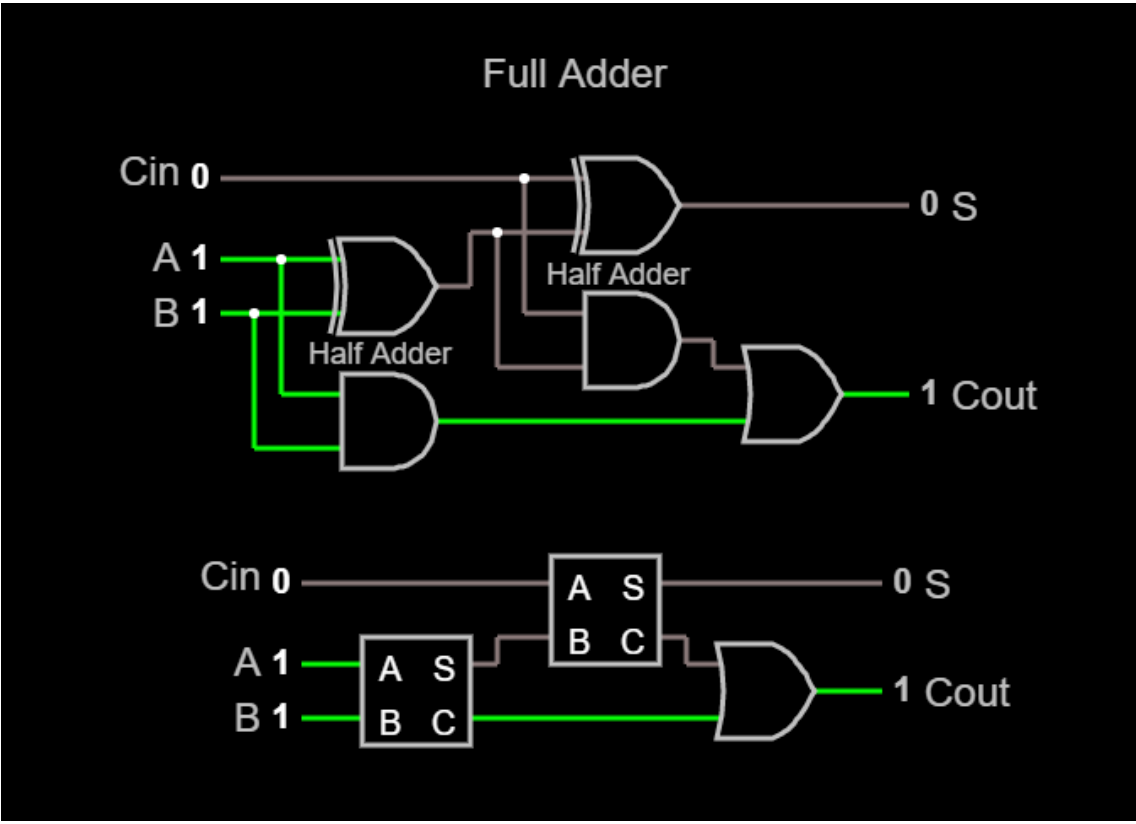


Figure 4-2: The Full Adder simulation. The top section is the circuit, the bottom is the encapsulated component.

Another example is a Full Adder which, like the previous example, has the gate based circuitry with an encapsulated component below it as shown in (Figure 4-2).

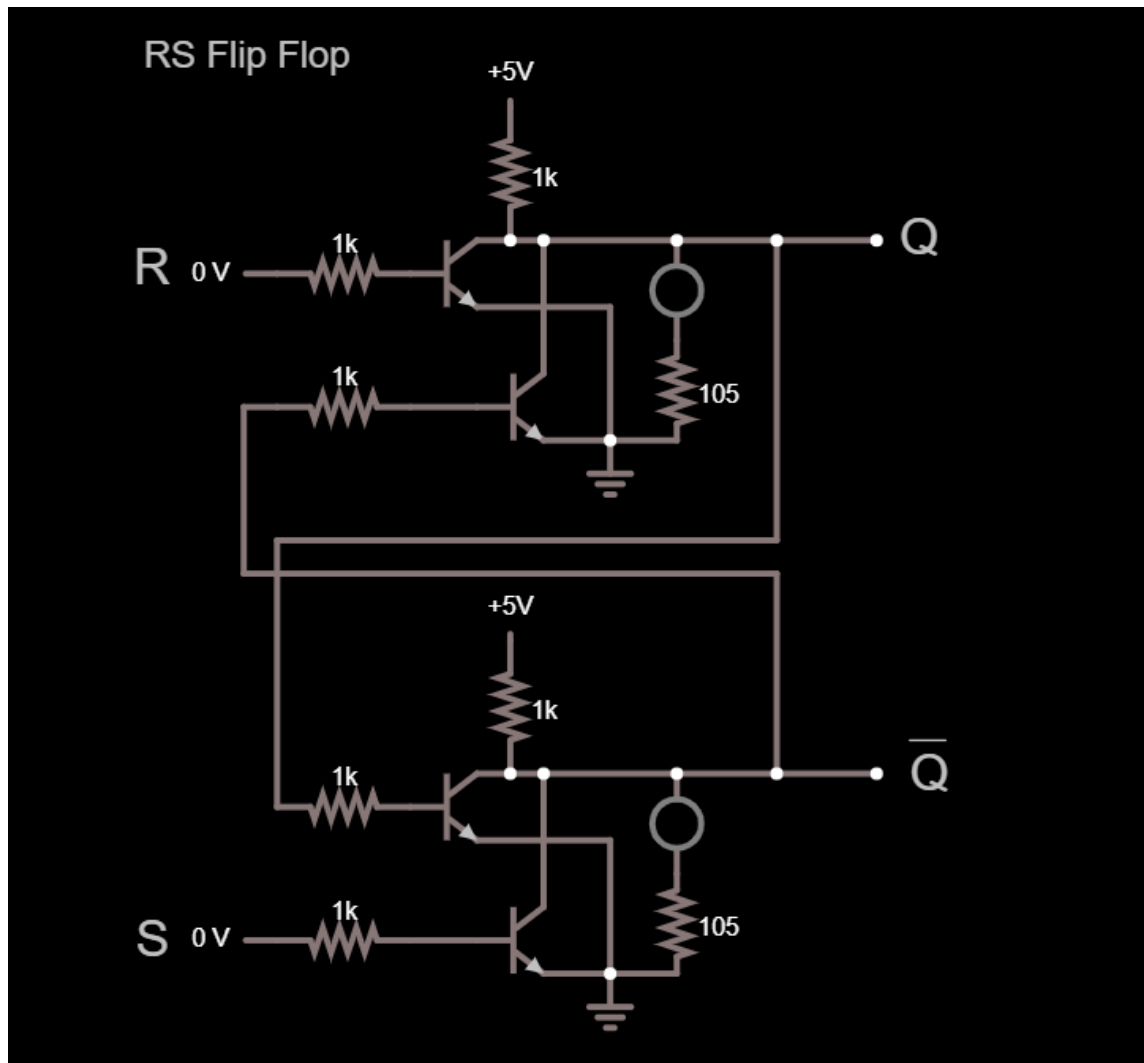


Figure 4-3: Transistor level RS Flip Flop.

As previously mentioned, it is possible to generate a circuit on the transistor level as seen in **(Figure 4-3)**.

CHAPTER 5

BUILDING

5.1 Soldering

In total, there are approximately 900 individual PCBs that form the Megaprocessor. To get these soldered required a significant team effort. A large room was located to house the Megaprocessor and provide a build environment, and ten soldering stations were setup. Initially, simple test PCBs were purchased from Adafruit to give students something to practice soldering on [3,4]. The initial soldering team was comprised of approximately 20 students. This number fluctuated as enthusiasm for the Megaprocessor project rose. There is a total of 300,000 solder joints for the build. This was estimated to take approximately 1,200 person-hours, provided the average solder joint takes 15 seconds (this was researched as well). The RAM is by far the most complex and difficult PCB to solder, and all 32 RAM boards account for over half of all the soldering.

5.2 Testing

Due to the large number of PCBs and the daunting prospect of debugging problem PCBs down the line, testing methods were meticulously designed. The goal was that PCBs would not be accepted as ‘completed’ until they were tested. This would help to avoid a mountain of problem PCBs to debug at the end of the soldering phase.

Initially, a testing harness was designed on large breadboards with pogo pins that could be inserted into the breadboard and stick out, exposing contact points. The PCBs would be centered in the breadboard, with pogo pins underneath each input, output, ground, and power input. A Raspberry Pi would serve as a driver, and have a ribbon cable connected to the breadboard, breaking out to all the test loops. However, an issue was discovered with this testing method. After PCBs were soldered, it was not possible to test in this way because of the test loops on the PCBs. To continue testing, the test loops would initially need to be excluded. The PCBs could then be tested, and subsequently the test loops could be soldered onto the PCBs. Ultimately, a different method was utilized that made use of testing clips (alligator clips) to connect to the test loops (that were soldered on all the inputs and outputs). Fortunately, this method became an easier testing method, though the cost of the pogo pins was ultimately lost.

Once a PCB was connected and ready to test, Python scripts that were designed at Louisiana Tech University were executed to test the PCB. These test scripts were written by students and faculty working on the project. These students also helped to identify issues in the initial testing setup and helped to create a working test setup as described above. The test scripts iterate through all the possible combinations of inputs and observe for the correct output(s). There is a known issue for larger PCBs that have too many inputs and outputs for a Raspberry Pi to connect to. To address this, multiplexors were used. Not all PCBs can be tested in this way, and time will tell if there are issues with these PCBs as the build progresses.

5.3 Laser Paneling

Instead of printed paper glued to wood panels for the modules (as Mr. Newman implemented in his original Megaprocessor), a laser etched design on lauan wood paneling was chosen for a more polished appearance. After some testing and debugging, an attractive product was attained (**Figure 5-1**).

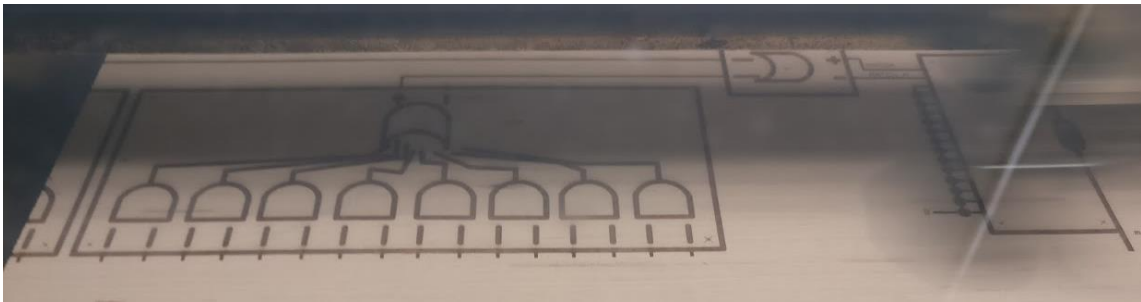


Figure 5-1: Test of laser printing of module graphic.

However, a problem exists with this approach: there is a maximum size that the laser printers on Louisiana Tech's campus can support (two by four feet), and some modules that exceed that size. However, modifications can be made to the metal framing to accommodate a split in oversized modules. This would allow smaller sections of a module to be lasered and subsequently assembled onto the frame. The process of building the frames is ongoing; therefore, results on such a method has not yet been determined.

5.4 Metal Framing

The metal framing was redesigned (a contribution to the Megaprocessor) that supports the ability to split the modules if needed. The metal framing for the Control and I/O frame was cut, counterbored, and assembled. To save time, the remaining metal

work may be outsourced to the distributor of the extruded profile aluminum that is used in the frames. The updated frame designs will also be included in the open repository.

5.5 Assembling

Once ready, PCBs will be mounted onto modules. The PCBs will then be connected using ribbon cables and individual wires as needed. Most modules that have connections to others have bus connectors at the perimeter. Modules will then be mounted onto the frames. They will then need to be connected and cable managed. Some cable management is done behind the frames such that the individual wires are routed behind the PCBs and modules so that they do not obstruct viewing. Due to the silk screening of circuits on the PCBs and lasering of connectivity on the modules, tracing the connectivity diagram is easy; therefore, the wires being routed behind and hidden provides a more professional presentation. This can be seen in **(Figure 5-2)** and **(Figure 5-3)** [29].

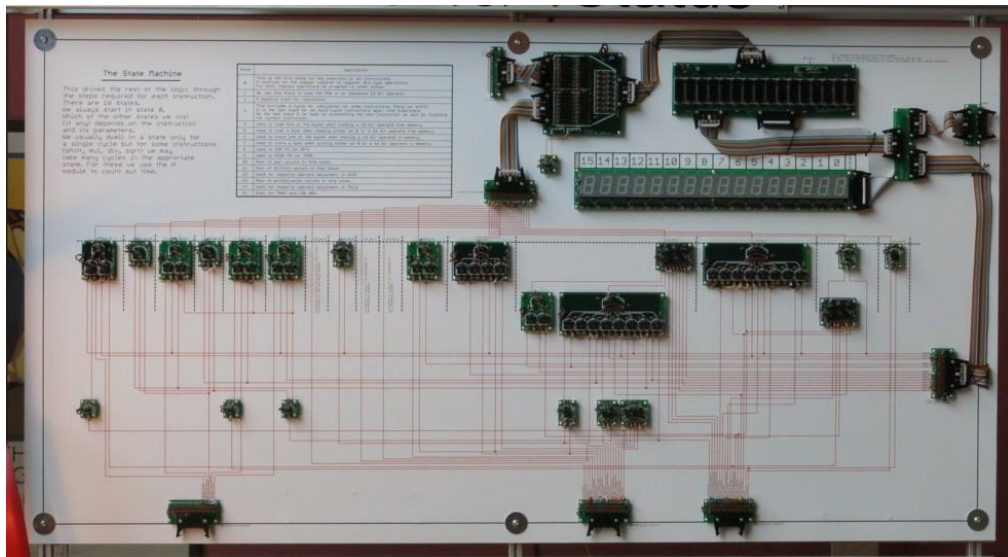


Figure 5-2: The front of the State Machine Frame.

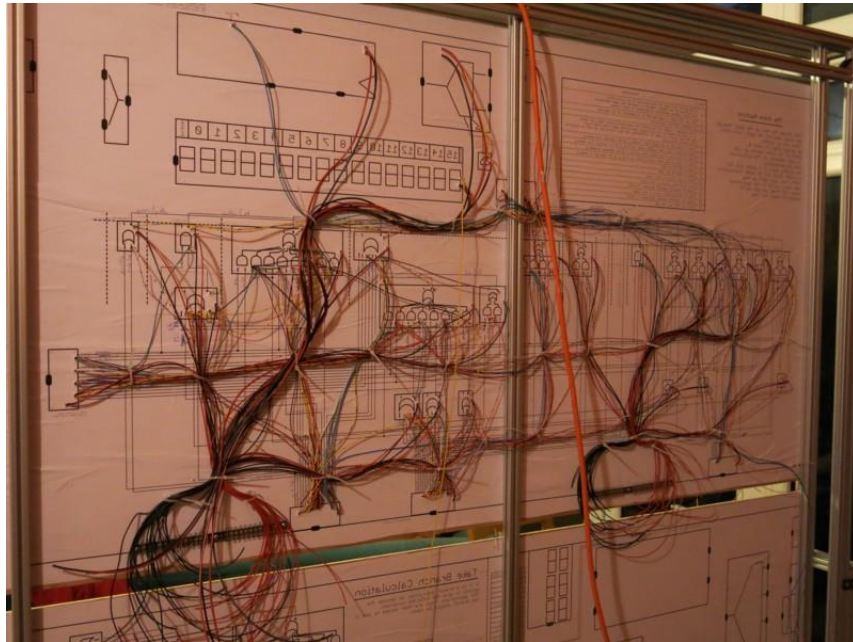


Figure 5-3: The back of the State Machine Module.

There is also power management to be considered at this point. Each frame can have as many as three 5-volt, 5-amp PSUs to power them. Most of the power needed for the Megaprocessor is for the 10,548 LEDs. Most connections between the frames are done over a 64-bit ribbon cable that combines buses together on one cable to allow for better cable management. All the connections will need to be made while double checking that there is not a bus being flipped and that all connections are properly seated. This, of course, is a task to be done after completion of the build phase.

Creating the custom ribbon cable should be done with caution, making sure that connectors are facing the correct way and are seated properly. A testing rig was setup to make sure that all the connections in the created ribbon cables are functional.

5.6 Awakening

After the Megaprocessor is built, it will take considerable effort to debug issues that arise once power is connected and it is turned on. Any number of problems could occur, including broken components, bits being flipped, and timing issues with the oscillators. Setting up the laptop and the programming of the three Igor boards on the Control and I/O boards is not a small matter. The programming of the PIC (peripheral interface controller) on the display boards, which drive the individual seven segment displays, is also something that has not yet been addressed.

CHAPTER 6

PRELIMINARY RESULTS AND CONTRIBUTIONS

This project is currently still in development. The end goal is a fully functioning Megaprocessor that can be used as part of the curriculum at Louisiana Tech University – and, eventually, at other institutions throughout the world. This chapter discusses contributions to advancing computer architecture education. In summary, it details the development of a replicable build process for the Megaprocessor, as well as the merits of its use in computer architecture education. Over two years of work has gone into this project so far, with the goal of others being able to utilize the work done here without having to do their own discovery and reverse engineering process.

6.1 Board Designs

All changes that were made to correct the PCB designs were documented into new PCB schematics. The revised designs are available to use for manufacturing additional PCBs.

Ten PCBs were updated: 2x2AND_OR, 4OR, 7SEG_3X2DIGIT, 7SEG_METER, 7SEG_STATUS, 8B_RAM, 8BIT_ADD_SUB, 8BIT_LOGIC, 8x2AND_OR, and 16REG. Some required creation or correction of traces. Others required more trivial updates, such as moving a component to a better location. Two modules were also updated: Instruction & Fetch and Special Purpose Registers. Upon double checking the designs, it was noticed that there were bus boards and connections

missing from the design file which were present in Mr. Newman's Megaprocessor. After consulting with him, the problems were resolved, and designs updated.

6.2 Parts List

The parts list is the most visible output from the significant reverse engineering process. A parts list did not exist before the start of the project. A full parts list for all the PCBs, frames, and additional hardware will be made available in an open repository. Through the build phase, many issues were encountered; therefore, many changes were made. These have all been recorded. It is the intention that the parts list that was designed be used as a basis for others to build their own Megaprocessor.

6.2.1 Board Components

The board components list took an estimated 100 hours to complete. Some information was extracted from the schematics provided by Mr. Newman; However, much information was missing. Therefore, many parts used in Mr. Newman's design were not identified and labeled. Consequently, these had to be determined. Verifying that everything was correct took many hours before expensive purchases were made. This included double checking that all the parts were indeed correct, that spacing of the components fit properly onto the PCBs, and that all the components would properly fit together on the PCBs. 1,277 PCBs (including margin of error, see next section) were purchased for this project; therefore, everything had to be double checked.

During the compiling of the parts list, different margins of error, or buffers, were considered and established. This was to address tolerance against part and soldering failures. Additionally, due to the way that the PCBs are manufactured, an entire sheet of PCB material is used for manufacturing any given PCB. The cost of the manufacturing is

equivalent to the cost of the entire sheet of PCB used. Therefore, the cost is the same whether one small PCB is manufactured from the single sheet and the rest is waste, or if the entire sheet is used to manufacture the maximum number of PCBs (that can fit on the sheet). This assisted in choosing the margins that were built into the order. These margins are designated within the documentation, with an excerpt shown in (**Table 6-1**). Others may decide to purchase based on the same margins or may adjust them as they see fit. 843 PCBs are needed for the Megaprocessor build, and 1,277 in total were ordered.

Table 6-1: Margin Calculations. This is not the full table, it is only a small segment.

Component	Original Quantity	25% Margin	50% Margin
16REG	13	17	20
1BUF	30	38	45
2OR	74	93	111
2X16MUX	54	68	81
2X2AND_OR	4	5	6
2X3AND_OR	48	60	72
3AND	6	8	9
3OR	19	24	29

Because of the manufacturing process, some margins are not as cleanly defined. For example, the number of RAM boards ordered was exactly the number needed. This is because, to add another PCB, an entire new PCB sheet is needed, which is expensive. Margins were checked against the size of the sheet being used in manufacturing and were only increased up to the point that the current sheet of PCB material was fully utilized.

One of the parts lists generated for each PCB can be seen in (**Table 6-2**) -- for the register PCB. There is a total of 491 components that go into a single register PCB. The total cost per PCB in this case was \$26.51 (at the time of purchase).

Table 6-2: Parts list for the 16-bit register board. The component names are simplified to fit here, and the full part names can be found in the documents in the open repository.

Component	Quantity	Pitch	Dimensions
100N Capacitor	7	5.08	
100UF Capacitor	1	2.54	12.7 mm diameter
Red LED	32	2.54	3.0 mm diameter
Blue LED	2	2.54	3.0 mm diameter
20 pin locking right-angle header	2	2.54	34.29 mm distance
30 pin locking right-angle header	1	2.54	
2N7000 Transistor	276	2.54	5.08 mm diameter
10K Resistor	132	5.08	
75 Resistor	2	5.08	
470 Resistor	0	5.08	
1.5K Resistor	32	5.08	
Test Loop	2	1.27	

There are 13 register boards needed for the Megaprocessor build, 20 including a buffer (or margin). The full number of components for these PCBs were then calculated in (**Table 6-3**). This gives a total of 26,260 solder joints, and a total cost for all register boards of \$530. And this is for one of 48 unique PCBs.

Table 6-3: Build information for all 16-bit register boards needed.

Component	Quantity	Solder Joints
100N Capacitor	140	280
100UF Capacitor	20	40
Red LED	640	1,280
Blue LED	40	80
20 pin locking right-angle header	40	800
30 pin locking right-angle header	20	600
2N7000 Transistor	5,520	16,560
10K Resistor	2,640	5,280
75 Resistor	40	80
470 Resistor	0	0
1.5K Resistor	640	1,280
Test Loop	40	80

6.2.2 Changes to LEDs

The LEDs and resistors were modified in order to increase power efficiency. Furthermore, different colors of LEDs were selected: red LEDs for inputs and outputs, green LEDs for selection of data paths, and blue LEDs for other different PCB semantics. The resistors chosen for each LED were determined by looking at how bright the LED was during testing, while attempting to keep power consumption down (i.e., utilizing Ohm's Law, shown in **Eq. 6-1**, to select appropriate resistors for the desired current flow through the LED).

$$\text{Voltage} = \text{Current} * \text{Resistance} \quad \text{Eq. 6-1}$$

Because voltage is constant, increasing the resistance lowers current.

Table 6-4: Red LEDs: 2mA, 1.7V.

<i>I</i> (mA)	<i>R</i> (ohm)	<i>P</i> (mW)
7.021	470	23.17
3.300	1000	10.89
2.200	1500	7.26
2.063	1600	6.81

For the red LEDs, 1.5K ohm resistors were chosen from (**Table 6-4**).

Table 6-5: Green LEDs: 20mA, 2.1V.

<i>I</i> (mA)	<i>R</i> (ohm)	<i>P</i> (mW)
19.333	150	56.07
13.182	220	38.23
6.170	370	17.89
2.900	1000	8.41
1.933	1500	5.61

For the green LEDs, 220-ohm resistors were chosen from (**Table 6-5**).

Table 6-6: Blue LEDs: 20mA, 3.5V.

<i>I</i> (mA)	<i>R</i> (ohm)	<i>P</i> (mW)
20.000	75	30.00
6.818	220	10.23
3.191	470	4.79
1.500	1000	2.25
1.000	1500	1.5

For the blue LEDs, 75-ohm resistors were chosen from (**Table 6-6**).

6.2.3 Soldering

Calculations were made in order to determine how much solder would be needed for the entire build phase (largely, to solder electronic components onto the PCBs). This was also done to ascertain how much soldering work was needed during the build phase. After some research, it was determined that each solder joint would require, on average, 0.00033 oz of solder [11]. The total maximum number of solder joints was calculated to be approximately 450,000. This was an overestimation that included student errors and margin. In total, approximately 9 pounds of solder was needed. As students and faculty gain experience soldering, it was estimated that an average solder joint takes approximately 10 seconds to solder. Consequently, the total time to complete all the soldering is approximately 1,240 hours.

6.2.4 Module Components

Each module needed to be checked for the number of PCBs as well as any additional parts. There are 14 different modules, each of which contain dozens of PCBs.

Each module connectivity diagram had to be checked against the finished product that Mr. Newman made, and some things were missed in his connectivity diagrams. An example of a module parts list can be seen in (**Table 6-7**), for the General Register Module.

Table 6-7: Boards list for General Purpose Registers module.

Boards	Quantity
BUS_TEE	5
BUS_4TEE	6
BUS_SPLIT_DISPLAY	2
SHIFT_ZERO_LEFT	4
SHIFT_ZERO_RIGHT	2
BUS2BUS	7
4X16MUX	5
16REG	5
7SEG_3X2DIGIT	5
BUS_SPLIT_DISPLAY	1

Each module has a list like the above table, that provides the total number of PCBs needed for the build.

6.2.5 Frame Components

Each frame is composed of the modules that make it up. In addition to that, there are extra boards on the frame that deliver power to all the boards and connect the frame to other frames. Furthermore, many other items needed to be located and accounted for;

for example, ribbon cable, wires, sockets, and many other extra parts needed for the build. A small fraction of that list is included in (**Table 6-8**).

Table 6-8: Sample from parts list outside of board components.

Part	Quantity	Notes	Unit Cost
20-pin ribbon cable	6	300-foot roll	\$94.07
30-pin ribbon cable	5	100-foot roll	\$58.73
20-pin IDC plug	1,192		\$0.90
20-pin IDC socket	1,876		\$0.90
18 AWG wire, solid, black	10	100-foot roll	\$35.48
DIP-20 socket	245		\$0.71
PCB test point	11,755	Various colors	\$0.12
4 MHz standard clock oscillator	5	5 VDC	\$1.76

6.2.6 Part Count

Once all the PCBs, modules, and frames were accounted for and double checked, a final parts list was created that contains all the components that needed to be ordered for the PCBs. (**Table 6-9**) illustrates an exceedingly small sample of the most numerous of the components for the entire Megaprocessor build – as well as their Mouser part numbers. Mouser is a large online distributor for electronic components.

Table 6-9: Sample from part counts for all boards.

Part	Quantity	Mouser Part	Total Cost
2N7000 Transistor	69,743	512-2N7000	\$3,208.18
10K Resistor	38,700	270-1K-RC	\$348.30
1.5K Resistor	15,945	270-1.5K-RC	\$159.45
Red LED	15,945	859-LTL-4221NLC	\$1,626.39
Diode	4,112	625-BAT42	\$4,112
1K Resistor	3,427	270-10K-RC	\$41.12
100N Capacitor	2,476	594-K104K15X7RF5UH5	\$99.04

6.2.7 Metal Framing

Mr. Newman's design includes a basic outline of the metal frame assembly; however, this was not enough to place an order for build parts. Additionally, extra space in the frame for building and laser engraved signage was needed. A full redesign of the metal frames, with full parts lists and build guides, were created. The frames are constructed from extruded profile aluminum using counter-bore anchor cam assemblies. They were redesigned in this way to allow flexibility of ordering from a readily available supplier to Louisiana Tech University. The metal framing can be ordered as whole pieces that can then be measured and cut, or full details can be provided to the distributor for cutting and counterboring. For the General Purpose Registers frame, the metal frame parts list (**Figure 6-3**) and build plans (**Figure 6-1**)(**Figure 6-2**) are a small sample of what can be found in the documentation contributing to the Megaprocessor [12].

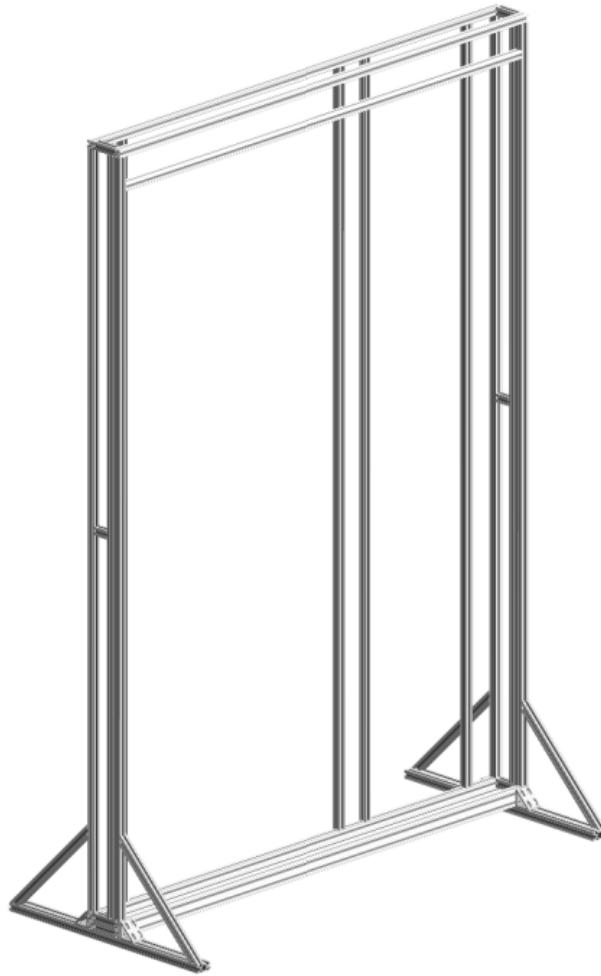


Figure 6-1: Isometric view of metal frame.

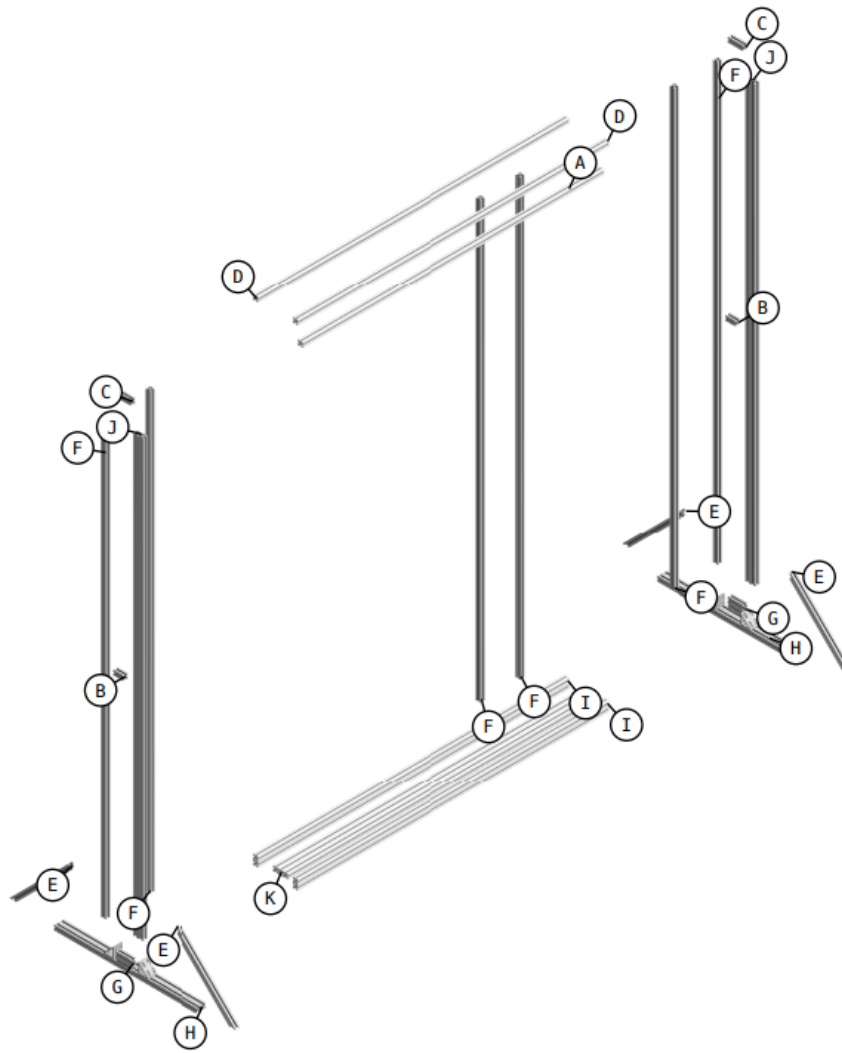


Figure 6-2: Exploded view of metal frame.

TAG	Part #	Qty	Length each (or area)	Units	Total wgt (kg)	Description (all extrusion dimensions at the left end)	Each \$	Price \$
A	1010	1	61.00	IN	1.22	7042 in slot B Both Ends;	14.03	14.03
B	1010	2	2.00	IN	0.08	7042 in slot B Both Ends;	0.46	0.92
C	1010	2	3.00	IN	0.12	7042 in slot D Both Ends;	0.69	1.38
D	1010	2	63.00	IN	2.51	1" X 1" T-SLOTTED PROFILE	14.49	28.98
E	1010	4	16.97	IN	1.35	7081 ***PER DRAWING 1***	3.90	15.61
F	1010	6	88.00	IN	10.54	7042 in slot B Both Ends;	20.24	121.44
G	1020	2	3.00	IN	0.21	7042 in slot J Both Ends; 7042 in slot I Both Ends;	1.17	2.34
H	1020	2	29.00	IN	2.05	1" X 2" T-SLOTTED PROFILE	11.31	22.62
I	1020	2	63.00	IN	4.46	1" X 2" T-SLOTTED PROFILE	24.57	49.14
J	1020	2	88.00	IN	6.23	7042 in slot G Both Ends; 7042 in slot I Both Ends; 7042 in slot E Both Ends;	34.32	68.64
K	1030	1	61.00	IN	3.15	7042 in slot H Both Ends; 7042 in slot E Both Ends;	33.55	33.55
L	4138	4	1.00	EA	0.33	10 S 8 HOLE INSIDE CORNER GUSSET	7.45	29.80
M	7000	8		EA		1" X 2" T-SLOT AND TUBE CUT TO LENGTH	1.95	15.60
N	7003	1		EA		1" X 3" T-SLOT AND TUBE CUT TO LENGTH	1.95	1.95
O	7005	13		EA		1" X 1" T-SLOT AND TUBE CUT TO LENGTH	1.95	25.35
P	7042	46		EA		10 S ANCHOR FASTENER COUNTERBORE	2.25	103.50
Q	7081	4		EA		MITER CUT & COUNTERBORE 1010	14.30	57.20
R	3061	32		EA	0.15	1/4-20 X .5" BHSCS	0.23	7.36
S	3395	46		EA	0.63	10 S ANCHOR FASTENER ASSEMBLY	2.90	133.40
T	3382	52		EA	0.24	10 S ECON T-NUT 1/4-20 THREAD	0.21	10.92
Total Weight:					33.26 kg (73.32 lbs)	Total Amount:		\$ 743.73

Figure 6-3: Full build of materials, including price.

In addition to these plans, the full 3D models for all frames as seen in (Figure 6-4) will be made openly available in an open repository [12]. The tool to load and explore these models is Frame Designer by Framexpert, which should be readily available through their web site (<https://www.frameexpert.com/products/framedesigner/>). Not only does this make building the frames easier, it also allows for easy modification of the frames as needed.

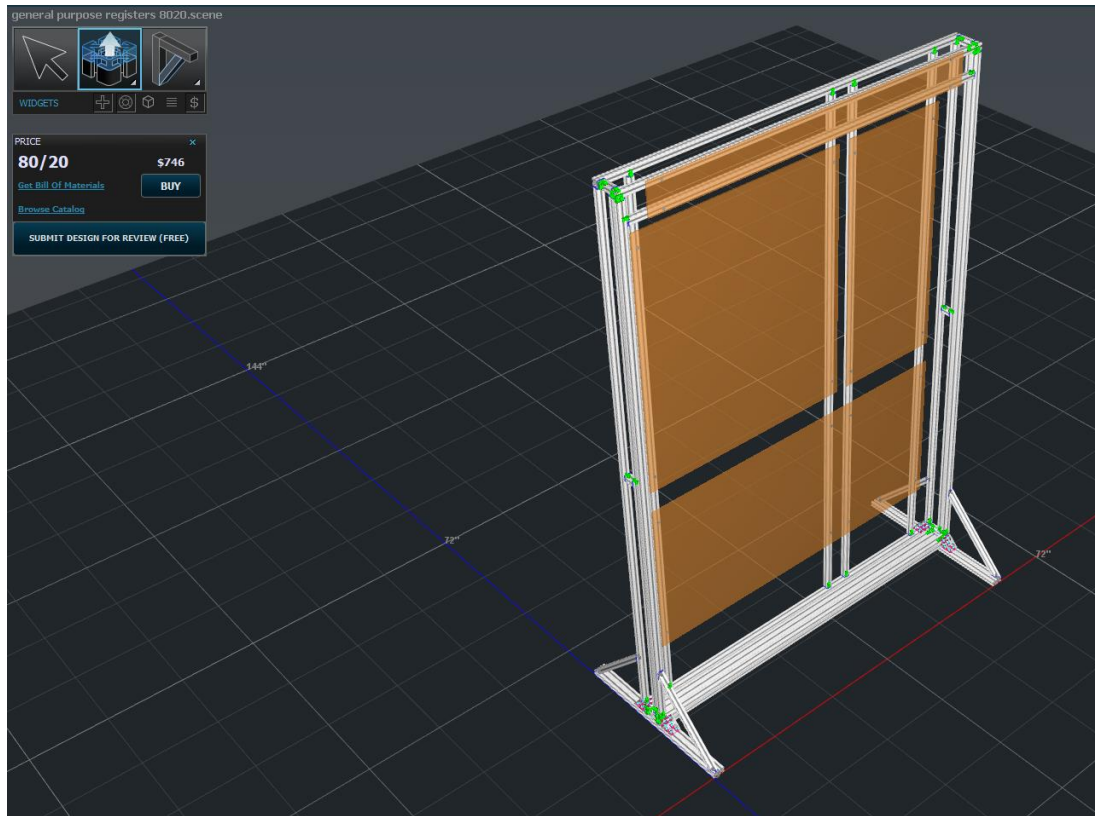


Figure 6-4: Screenshot of 3-D model of General Purpose Registers frame.

6.3 Test Plans

During the build phase, scripts for testing various PCBs were designed and created. Not all the PCBs have test scripts; however, what was created is documented and organized, including the full testing setup and other relevant information. More complicated testing setups with multiplexors is also documented so that replication is enabled. The test plans are important as they help to catch problems early on before problem PCBs become a part of the Megaprocessor, where (and when) problems will be much more difficult to diagnose.

6.4 Build Plans

A build plan for the Megaprocessor did not exist prior to the start of this project. Creating one took hundreds of hours of trial and error. Given that everything is documented from the reverse engineering phase, and all orders were fully documented, it will be easier for another Megaprocessor to be built by an institution.

This build was not without issues, and hopefully the contributed documentation can be of use to any future builders of a Megaprocessor. For example, some of the components (such as LED's and other sensitive components) may benefit from socketing. That is, soldering a socket that can accept an electronic component (without solder) would allow for dynamic changing of LED colors, for example, or perhaps to minimize damage to sensitive ICs (integrated circuits) or other discrete components (such as the transistors). Moreover, troubleshooting and changing components out later is easier. Although not all socketing plans were implemented, some were.

While the build is not yet complete (and by extension, neither are the guides), it will be quite easy to modify and implement at that time. For each PCB that is needed for the Megaprocessor, a build guide was designed for students and faculty to follow. An example of a build guide can be seen in **(Figure 6-5)**.

SERVICES

QTY	PART NUMBER	NEWMAN DESCRIPTION	CHEAT SHEET
1	1470-1477-5-ND (DigiKey) 919-RO-0509S/919-RO-0509S/P	IL0509S/IL0509S/SIL_IL	socket
6	859-LTL-4231	L-1334SRT/L-1334SRT/LED3MM-GREEN	LED (+ is long lead, - is short lead); 5VE through 5VA, 7V
5	593-VAOL-3LSBY2	L-1334SRT/L-1334SRT/LED3MM-BLUE	LED (+ is long lead, - is short lead); RESET, CLK1, CLK2, two others (to the right)
4	512-1N4148	1N4148/1N4148/DO34-5	diode (- is black bar on the body); labeled 4148
5	490-PJ-102AH	POWER_ENTRY/POWER_25/MYLIB	DC power jack
4	653-XG4A-2031	20_HDR_LOCKING/ 20_HDR_LOCKING/ 20_HDR_LOCKING	straight
15	512-2N7000	2N7000/2N7000/TO92-DGS	transistor
15	270-10K-RC	10K/R_P2INCH/R_P2INCH	10 K-ohm resistor (brown, black, black, red, brown)
6	270-220-RC	220/R_P2INCH/R_P2INCH	220 ohm resistor (red, red, black, black, brown); for green LEDs
5	270-75-RC	75/R_P2INCH/R_P2INCH	75 ohm resistor (violet, green, black, gold, brown); for blue LEDs
10	151-203-RC	BLACK/TEST_LOOP/TEST_LOOP	GND/0V, RESET, CLK1, CLK2, two others (to the right)
6	151-207A-RC	RED/TEST_LOOP/TEST_LOOP	+5V, +7V

NOTES

- 470 Ohm resistors in the original design are substituted with 75 Ohm resistors;
- LEDs for +V are GREEN;
- LEDs for RESET, CLK1, CLK2, and two others to the right are BLUE;
- Test points are soldered so that loops are on the FRONT of the PCB;
- Test points are soldered so that wires can be soldered to them without affecting components on the PCB;
- RED test points are for +5V and +7V;
- BLACK test points are for GND/0V, RESET, CLK1, CLK2, and two others to the right; and
- We are not currently soldering test points to the terminals near the ground pads or in the center of the PCB.

Figure 6-5: Build guide for the Services Board. Includes parts number, the original description from Mr. Newman, and additional information to make the build process a self-guided one.

6.5 Digital Design Simulations

As previously mentioned in chapter four, many useful simulations that are easy to understand and modify were created. These simulations will be available for student and outside use through the open repository. Furthermore, it is anticipated that additional simulations will be created as fabrication of the Megaprocessor continues.

6.6 Open Repository for Distribution

All the contributions listed above will be placed into a publicly accessible open repository. It is important that the contributed work specified in this thesis is available

for other universities to use. Since the build is not complete at the time of writing this thesis, more will be added to the repository as progress continues. The repository in use currently is privately hosted and not yet public. Once the build phase is done, it will be migrated to an open and publicly accessible repository.

6.7 Researching Application to Education

The research laid out in chapter three is original and shows that, based on the literature reviewed, there is not another tool like the Megaprocessor used in computer architecture education. The build is not finished yet and its efficacy is not tested. However, considering the learning objectives from course requirements and ACM guidelines, a strong argument has been made for the use of the Megaprocessor in computer science curricula; furthermore, that publicly providing build and other documentation can allow others an easier adoption path.

In chapter seven, possible options for applying the Megaprocessor to the classroom are discussed; furthermore, a plan to evaluate and assess its performance in curricula is specified in section 7.6. This is a good first step to integrating the Megaprocessor into computer science curricula.

Another aspect advocating for the adoption of the Megaprocessor is ABET accreditation [1]. ABET is an accreditation board for engineering and technology programs. The research behind this project, the pedagogy backing the Megaprocessor, and the innovative hands-on teaching methods that will come from the Megaprocessor, will be of great value to ABET-accredited programs.

CHAPTER 7

DISCUSSION

At the time of this thesis, the Megaprocessor at Louisiana Tech University is not completed. Once it is, however, it will be integrated into existing relevant courses. This chapter discusses several proposed methods for applying the Megaprocessor to computer architecture and related courses in a university setting.

7.1 Digital Design

The Digital Design course can be improved with the circuit design simulations contributed during this project. In these simulations basic gates are built from transistors, and gates can then be combined to make components discussed in digital design. These simulations can be used to help cover digital design concepts, as well as enable students to experiment with Megaprocessor boards.

The simulations and the Megaprocessor can be combined and used in class lectures. Topics to be covered can be created in one or more simulations and subsequently distributed to students for more hands-on exploration. If desired, teachers can create and use animated images or small recorded videos to place into a slide deck for a lecture. This is better than a simple diagram, and still images of circuit layouts are easy to make.

The simulations can also be used in a lab setting. A set of simulations can be setup for use in labs during the course, one setup for each lab, each with a set of items to be experimented with, arranged, or constructed by the students following teacher instruction or objectives. Labs or special classes can be held with the Megaprocessor, where a broad overview of the processor design is covered. A lab session could show, for example, how all the distinct components come together to create the Megaprocessor. It can also be used to show how a computer is built with simple parts such as transistors and other components. For many students, this will be the first time that they cover these topics. The simulations and Megaprocessor can help students gain an intuition with basic computer architecture concepts. This would equip students with a visual understanding of computer architecture and introduce them to foundational topics in the best way possible.

7.2 Computer Architecture

Course lessons can be redesigned to make use of the Megaprocessor. Below are suggested guidelines – a contribution of this work.

7.2.1 Teacher Driven Lectures

Teachers may run a program on the Megaprocessor and walk through it to teach the topic(s) of that day. They may also have a set of programs, from simple to complex, to use in class lectures. Teachers can walk through a program to show interactions within the Megaprocessor, such as: how data gets retrieved and stored within registers, how numbers get added in the ALU, or how the processor state machine drives the next step. Teachers could highlight the entire fetch, decode, and execute cycle of any single

operation. For example, they can step through the program as the next line of code is fetched and the add operation decoded. Then the execution step adds two numbers in the ALU before being stored in the destination register at the end of the execute cycle.

Teachers could also hold a class on each module or create a section of classes that cover each frame of the Megaprocessor. Instead of lessons centered around building an academic model (ALU, bus, counters, etc.), lessons can instead be centered around a component of the Megaprocessor, giving the students a physical model to learn from and interact with.

7.2.2 Student Interaction

The Megaprocessor simulations can be available to students. This would allow them to run teacher provided programs, modify them, or create their own. Students could submit their code for a class project after they have checked it with a simulation. Students could subsequently execute their code on the Megaprocessor and interact with it.

7.2.3 Soldering Labs

As the Megaprocessor is used more and more in the curriculum, and as it expands, new PCBs may need to be created, and existing PCBs may need to be updated and replaced. This can provide students with a tangible, hands-on experience in the classroom (for example, soldering replacement PCBs).

7.3 Accessibility Solutions

A limited number of students can have access to the Megaprocessor at any given time. Having an entire class vying for a good view could be difficult to equitably manage. However, there are several solutions that could help resolve this issue.

7.3.1 Divide into Labs

While slide-decks augmented by the Megaprocessor and simulations can be used for lectures, smaller labs could be held to allow students to get up close to the Megaprocessor. Questions can be asked and answered, programs run and analyzed, in addition to any other activity that would fit well into a small lab setting.

7.3.2 Photo and Video Repository

Instructors could create a set of photos and videos of the Megaprocessor. These images or videos would be of remarkably high quality, with the ability to zoom in to show details. Another option is to have an edited video where the zooming and highlighting takes place. These assets could be used in lectures so that, instead of a typical diagram for example, a video or image is used.

Since the processor can be run on a step-by-step basis, this could be leveraged to create information dense, but easy to understand, slide decks. Instructors could create these runtime slide decks by following the following steps.

1. Setup the Megaprocessor with the desired program;
2. Run/execute a step;
3. Take a photo of each Megaprocessor frame; and
4. Repeat until the program is completed/terminated.

The instructor would then have a series of photos that show the state of the Megaprocessor at each step, with no information being lost. Undoubtedly, This would take a lot of planning and effort to accomplish; however, once a repository of videos and slide decks is created, it should be manageable to continue to add to it and update it as ideas for additional class material arise. This collection of videos, photos, and programs could also be shared for use by other universities. An algorithm as simple as the addition of two numbers could be covered in new detail and easily shared.

7.4 Schematics and Design

Students can have access to the design files of the various PCBs and modules. These could be used in advanced lessons for exploring PCB and module design with the real-world limitations of PCB traces. Advanced Computer Architecture labs could be held on modifying the PBC schematics for more advanced features. The design files and schematics could also be used as lecture material to go along with the videos, images, and live demonstrations of the Megaprocessor.

7.5 Megaprocessor Simulation

Access to the Megaprocessor simulation could be provided to students, which would allow them to compile and run their own or instructor provided code. This allows for examination and assists with learning objectives. Students could modify code and observe differences in instructor provided programs. They could also write their own code and execute it, even as a class project or assignment.

7.6 Measuring the Effectiveness of the Megaprocessor in the Classroom

Determining whether the Megaprocessor is a good educational tool will require testing. This can be done by comparing it directly to whatever current learning tools and objectives are in use. The following method can be applied to the Computer Architecture and Digital Design courses to assess the Megaprocessors effectiveness.

7.6.1 Pedagogical Design

One of the following methods can be used to introduce the Megaprocessor into course usage. At the beginning, throughout, and at the end of the course students can be given questionnaires. These questionnaires, admittedly indirect in nature, can ask about the students' perceived proficiencies on a range of topics relevant to the course. There can be questions that allow students to rate different aspects of the tool(s) they used. A free-form section on the questionnaire that asks students for their general thoughts, including what improvements could be made, would be beneficial to assessment of the Megaprocessor's use in the classroom. The professor(s) teaching the various course sections should also document their assessment of the Megaprocessor in the context of the classroom. These can student surveys can be reviewed to attempt to gain some insight comparing the tools and gauging the students' reaction to the change [15,17,26]. Another metric that could prove useful is the course drop rate. For Computer Architecture and other high-level courses, there is some established, expected drop rate backed by university data. Observing a measurable decrease could indicate the Megaprocessor's effectiveness in the classroom.

Compare Sections

If two sections of a course are being offered within an academic term, adapt one of them to use the Megaprocessor. This can be done in whatever way the instructor believes is best for the course. The other course is to remain unaltered. The student surveys, average grades, and engagement can then be compared across the two sections. If the same professor teaches both sections, it may be possible to notice improvements or otherwise potentially caused by the Megaprocessor. If students respond well, the Megaprocessor could continue to be utilized in a course; alternatively, it could be appropriately scaled back to avoid causing negative impacts on students learning.

Optional Additions

For each section within a course, a set of optional labs or classes that utilize the Megaprocessor could be created. This additional teaching would be optional, with students who participate participating in questionnaires. The inherent problem with this is that positive results of the questionnaires and/or higher grades in the class could be due to additional instruction, for example, and not strictly the integration of the Megaprocessor in the classroom. Additional lectures and material can be given to the other group of students, but at that point it would probably be easier to divide the class into sections taught differently.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Conclusions

Undoubtedly, building, testing, and integrating the Megaprocessor in the classroom is a massive undertaking; however, the amount of work involved in this entire process was nonetheless underestimated. As of the writing of this thesis, it has been almost three years since the project began -- and it is still not finished. This delay in completion can be partially attributed to prioritization. Due to the diverse nature of the students and faculty participating on the project, there have been times when development efforts were slowed or put on hold. Even at the time of writing this thesis, the COVID-19 pandemic has halted the build progress. Admittedly, most of the participants are not experts in computer hardware or architecture. At the start, the desire to complete this project was driven by passion and interest. Upon learning about the original Megaprocessor, the possibility of revolutionizing computer architecture education was captivating; furthermore, its broader impacts and intellectual merit were evident. Although many challenges have been (and will undoubtedly continue to be) faced, the idea of providing students with an unparalleled education is motivating. While the build phase is not finished, research indicates that there is currently not another tool like it. The Megaprocessor has great potential to innovate computer architecture courses, and this is ultimately quite exciting.

8.2 Contributions of this Thesis

As presented in chapter six, there were several things contributed during this project that advance the state of the art. The massive reverse engineering processes resulted in the most visible contribution: a build plan and parts list that can be used by others to easily build their own Megaprocessor for use in the classroom. The reviewing of PCBs and module schematics resulted in several fixes and changes to the designs. The Megaprocessor is now more power efficient due to the replacement of LEDs and resistors. The redesigning of the metal framing resulted in an easier, more customizable build. The problems encountered, resolved, and documented, as well as the test suite contribute to a build plan that is more refined and implementable by others. In addition to the contribution of a build process, the research into the tools used for computer architecture education showcases a strong argument for the Megaprocessor. The guidelines for course integration are flexible with many options and pedagogy evaluation appear to be reasonable, given the seemingly straightforward hands-on learning provided by the Megaprocessor. Only time and an operational Megaprocessor will show affirmative results.

8.3 Future Work on Megaprocessor

Beyond the build phase of the Megaprocessor, future work is anticipated. One area involves adding functionality to the Megaprocessor. It is important to note that not all added functionality needs to be constructed from the transistor level. The level of granularity would certainly have to be considered based on build difficulty and the benefits. For example, the designer and original creator of the Megaprocessor allowed

himself the usage of more advanced parts and boards for the Control and I/O frame. The subsections below identify some potential directions in this area.

8.3.1 Additional Registers

There are only four general purpose registers. The addition of one or more registers could allow more advanced programs to be run. These would be addressable by teacher or student programs to keep track of more values while keeping the RAM free to act as a dot matrix display. These additional registers may also be needed for future complex instructions which need to use more than two registers at a time, the current limit on the instruction set for most instructions. These complex instructions could allow for more robust I/O or graphics driving. Additional registers may also be utilized in more advanced pipelining.

8.3.2 Bus Connection

The bus connectors that currently utilize ribbon cable of various widths could be converted to RJ45 (Ethernet). This conversion may make future builds easier and cheaper. This would also make the build tidier by having fewer and smaller cables. The tradeoff is that it would lose the generally pleasing aesthetic of the ribbon cables.

8.3.3 Larger Bus

A larger bus may be needed to accommodate more registers and additional features. This would not be a small endeavor, as it would basically amount to a moderate architecture redesign. The bus would be 'widened' throughout all the bus connections, leading to a redesign of registers and other components to take advantage of it, and most PCBs being modified to remain compatible to the bus connection.

8.3.4 Instruction Set

The above items may require expanding or consolidating the existing instruction set. Since all instructions in the available instruction space are allocated, increasing this space will be necessary to support additional instructions. There may also be ways to refine the instructions used to free up space. This would accommodate additional functionality such as I/O for keyboard, monitor, or external storage use. With changes to the bus, longer opcodes could be accommodated. These new instructions would need new circuitry in the Megaprocessor to do whatever is intended.

8.3.5 Redesign RAM

Over half of the soldering work during the build phase is on a single frame: the RAM frame. In the future, it may be worth modifying the design of the memory boards to enable finished boards, with all parts soldered on during the manufacturing of the PCBs (i.e., complete PCBs from a PCB fabricator). This would undoubtedly be much more expensive; however, it would allow for a more manageable build project. As discussed, some aspects of the build phase (even in the context of replacing potentially damaged parts on the Megaprocessor) could be used in the classroom. This questions whether all the PCBs should be redesigned such that they are received from the manufacturer in final form (i.e., with all electronic components pre-soldered). Obviously, assembly could then be done in shorter time; however, this should be considered only if it can be done without losing benefits of the Megaprocessor in the classroom. It may also be worth investigating a redesign of the RAM to be denser, as well as possibly having more RAM. Simply, this would create a larger dot matrix display. Maybe another RAM frame could be made to replace the internal RAM on one of the service boards on Control

and I/O, making that data more visible to students. Ultimately, this would allow the Megaprocessor to be much more scalable in terms of its use in computer architecture courses at other institutions.

8.4 Future Work on Megaprocessor I/O

The subsections below discuss a few additional ideas to extend the functionality of the Megaprocessor. These are more abstract ideas, with feasibility and potential benefits unknown.

8.4.1 Motherboard

The design of the Control and I/O frame could be evaluated to determine their potential to expand on the capabilities of the Megaprocessor. Mr. Newman had more elaborate plans for the I/O [27], and ultimately, the final design of the Control and I/O frame was chosen to make things as simple to build as possible – at the time. There are three FPGA service boards on the Control and I/O frame, affectionately referred to as Igor boards [34]. The layout and configuration of these Igor boards should be able to be reconfigured or replaced with a different solution as desired. Redesigning this frame to enable more capabilities should be possible as the connections to the CPU frames and RAM frame are established.

8.4.2 Display

It would be interesting to investigate what is necessary to effectively connect a monitor to the Megaprocessor to have true video output. This would improve general accessibility of the Megaprocessor. Mr. Newman initially planned to use the host computer to be a graphics output for the Megaprocessor [33]; however, this was not implemented in the final design.

8.4.3 Keyboard Input

If a display is possible, an investigation of what is needed to have a functional keyboard would become necessary. The keyboard should be able to interact with the display by displaying characters and navigating a keyboard cursor. This would allow students to explore the mechanisms that go into this type of I/O. This could be executed with a dedicated program that utilizes the display and keyboard as I/O. The program could listen for input from the keyboard, decode the signals coming from it, and display the characters, allowing students to interact with the Megaprocessor much like they do with traditional computing machines.

8.4.4 Storage

On the Control and I/O frame, the Megaprocessor includes integrated circuits that act as storage for the instructions that are executed. Currently, the Megaprocessor cannot directly access this storage (it must do so through the various Igor PCBs). An interesting question to consider is what is needed to enable connection of a storage device to the Megaprocessor that it can directly access? How would data be fetched from it, for example, by a student written program? A storage device could allow the Megaprocessor to handle large files. In addition, A floppy disk drive could offer a fun interaction for students.

8.5 Operating System

Depending on other functionalities added to the Megaprocessor, a basic operating system may help tie some of these new features and I/O together. A basic operating system would be needed to enable usage of any of the new I/O that is outside of a demonstration program. This could mean that the Megaprocessor would have use in

courses that are not necessarily computer architecture related. The development of an operating system or bash interpreter could be a class project for an operating systems class. Instead of writing a program for the Megaprocessor using the compiler, students could write a program in assembly as a challenge in operating systems or computer architecture classes.

8.6 Future Work on RISC-V

RISC-V is an open source processor and instruction set. Research that aims to determine if the Megaprocessor could be modified to implement RISC-V has intellectual merit. The creation of the plans, PCBs, and schematics would be a great contribution to computer architecture education. Using RISC-V, the Megaprocessor may be more easily adopted by universities that already use RISC-V, or could be a further incentive for universities to consider the switch to RISC-V. A RISC-V based Megaprocessor may potentially address any issues related to the implementation of additional functionality as described above.

8.7 Expand Circuit Simulations

Concerning the simulations that were designed during this work, it should be possible to continue to build up more complex parts and create a component for each Megaprocessor PCB. These simulations could then be used in lectures or made available to students. Admittedly, the success of this depends on how far the simulation environment can be pushed, based on how many components it can simulate at one time.

8.8 Remote Access

While discussing this project with a former professor, the question arose as to whether it would be possible to develop additional capabilities to allow remote access to the Megaprocessor. Modifications to the Control and I/O frame would be needed to add this functionality, and cameras setup to provide a live video feed to remote users. This could be useful to teachers, using remote access of the Megaprocessor to teach in a large lecture hall, for example. A system like this could be setup, and if robust enough, could be provided as a service to outside entities and universities.

8.9 Cyber Security

The Megaprocessor's potential for use in cyber security courses is evident. For example, it would be quite interesting to see if it is possible to demonstrate a simple attack on the Megaprocessor. For example, the Megaprocessor does some simple pipelining and speculation. The investigation and designing of an exploit to force a read from RAM sections which should have been protected, would be useful for course material.

8.10 Game Design

Given the properties of the Megaprocessor, it cannot currently support a complex game. However, there are games that can be written and executed, such as Tetris [33]. The use of this as a bonus assignment in computer architecture courses, for example, could be interesting to students and increase their interest in computer architecture. Students could create, submit, and play simple games. This would likely get students engaged and thinking hard about the constraints of the system. It could be the type of challenge to push many students to learn more.

8.11 Conclusions Regarding Future Work

The list of possible features is both daunting and exciting in possibility. Many of the considered future works would naturally be developed together and would take a lot of work. Some of these could be developed in a classroom setting as group-based projects. Others may need to be faculty designed and lead.

8.12 Final Remarks

The Megaprocessor is a remarkable tool with large untapped potential. The work done in this thesis seeks to begin tapping into that potential for academic use. The Megaprocessor and its introduction into the classroom can lead to a revolution in computer architecture education, where abstract models are no longer the only option.

APPENDIX A

DIAGRAMS

A.1 Megaprocessor Connectivity Diagrams

A.1.1 Organization of Diagrams

The Megaprocessor's full connectivity diagram cannot be fit onto a single page; therefore, they are included in this appendix [27]. To reconstruct the connectivity diagram, the following figures must be laid out in order of frame connection as specified in **(Figure 2-4)**. Considerations were made to make the diagrams as readable as possible, and the schematics for the connectivity diagrams will ultimately be available in the open repository.

State & Status

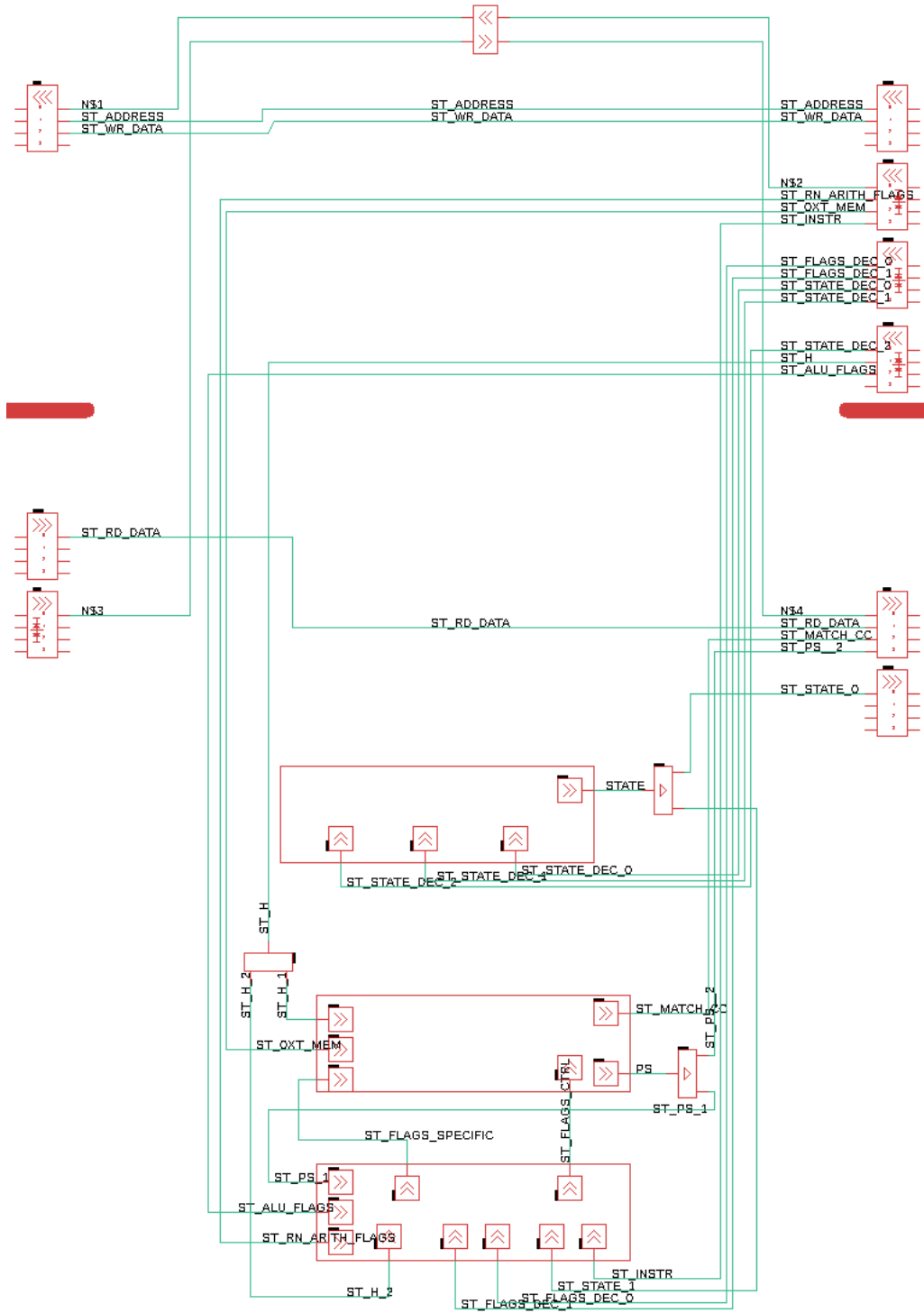


Figure A-1: State and Status connectivity diagram.

Special Purpose Registers

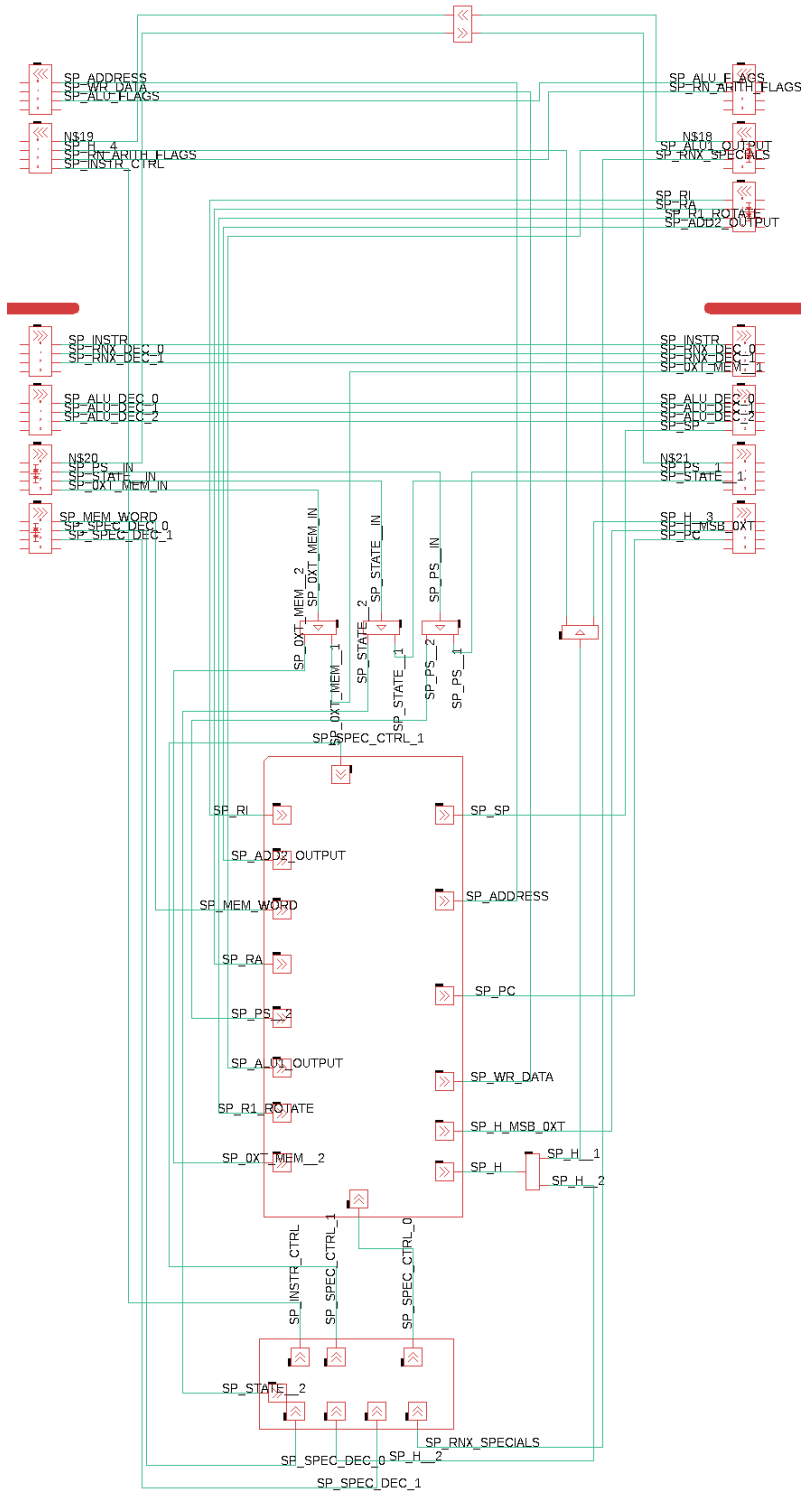


Figure A-3: Special Purpose Registers connectivity diagram.

ALU

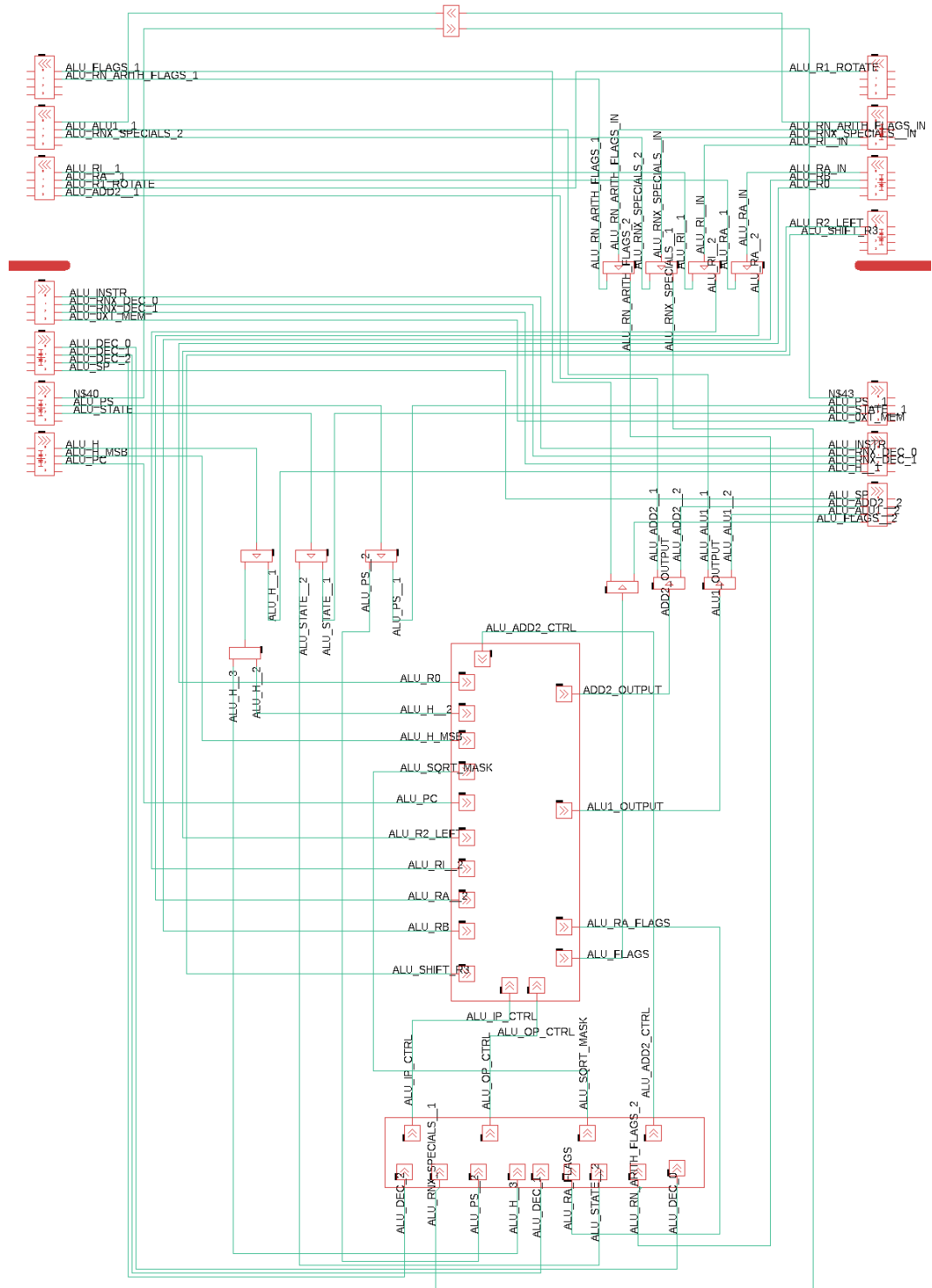


Figure A-4: ALU connectivity diagram.

General Purpose Registers

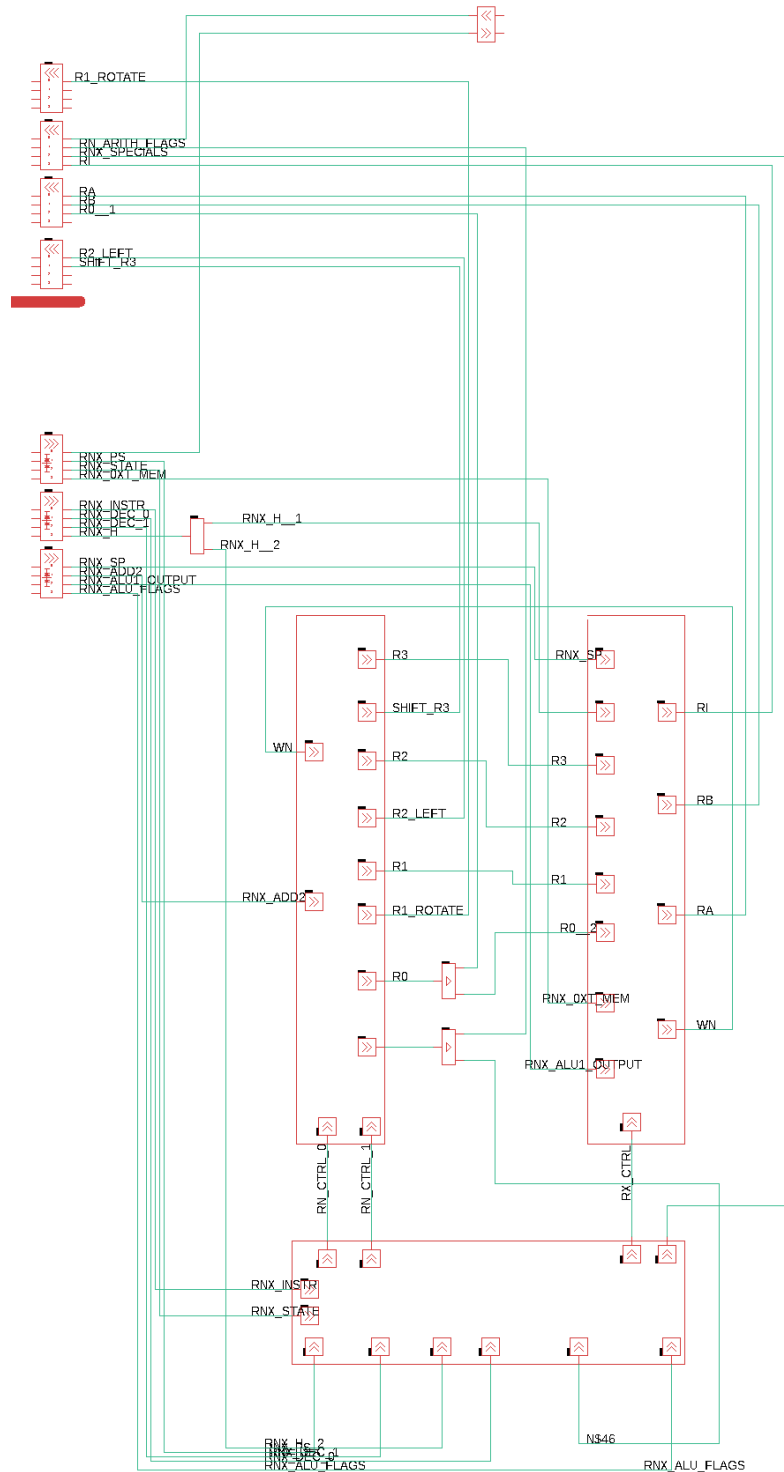


Figure A-5: General Purpose Registers connectivity diagram.

BIBLIOGRAPHY

- [1] ABET. ABET accreditation. Retrieved April 30, 2020 from <https://www.abet.org/>
- [2] ACM Computing Curricula Task Force (Ed.). 2013. *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. ACM, Inc. DOI:<https://doi.org/10.1145/2534860>
- [3] Adafruit. Conway's Game of Life Kit [v1.3] ID: 89. *adafruit.com*. Retrieved April 4, 2020 from <https://www.adafruit.com/product/89>
- [4] Adafruit. Drawdio kit [v1.1] ID: 124. *adafruit.com*. Retrieved April 4, 2020 from <https://www.adafruit.com/product/124>
- [5] R. Agrawal, S. Bandara, A. Ehret, M. Isakov, M. Mark, and M.A. Kinsy. 2019. The BRISC-V platform: A practical teaching approach for computer architecture. In *Proceedings of the Workshop on Computer Architecture Education, WCAE 2019*, Association for Computing Machinery, Inc. DOI:<https://doi.org/10.1145/3338698.3338891>
- [6] Association for Computing Machinery. 2016. Curricula Recommendations. *acm.org*. Retrieved February 15, 2020 from <https://www.acm.org/education/curricula-recommendations>
- [7] M. Black and P. Komala. 2011. A full system x86 simulator for teaching computer organization. In *SIGCSE'11 - Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, ACM Press, New York, New York, USA, 365–370. DOI:<https://doi.org/10.1145/1953163.1953272>
- [8] Centre for Computing History. 2017. Guinness World Record for Our MegaProcessor - Computing History. Retrieved February 18, 2020 from <https://www.computinghistory.org.uk/news/43619/Guinness-World-Record-for-Our-MegaProcessor/>
- [9] J.L. Dekeyser and A.S. Aljendi. 2015. Adopting new learning strategies for computer architecture in higher education - Case study: Building the S3 microprocessor in 24 hours. In *Workshop on Computer Architecture Education, WCAE 2015*, Association for Computing Machinery, Inc. DOI:<https://doi.org/10.1145/2795122.2795128>

- [10] J. Djordjevic, B. Nikolic, and A. Milenkovic. 2005. Flexible web-based educational system for teaching computer architecture and organization. *IEEE Transactions on Education* 48, 2 (2005), 264–273. DOI:<https://doi.org/10.1109/TE.2004.842918>
- [11] S. Driscoll. 2007. What kind of solder (rosin cored, etc. lead-free)? What is flux and when is it necessary? *CuriousInventor*. Retrieved April 30, 2020 from https://store.curiousinventor.com/guides/how_to_solder/kind_of_solder
- [12] FrameXpert. FrameDesigner. Retrieved April 30, 2020 from <https://www.frameexpert.com/products/framedesigner/>
- [13] R. Giorgi and G. Mariotti. 2019. WebriSC-V: A web-based education-oriented RISC-V pipeline simulation environment. In *Proceedings of the Workshop on Computer Architecture Education, WCAE 2019*, Association for Computing Machinery, Inc. DOI:<https://doi.org/10.1145/3338698.3338894>
- [14] J. Gourd. 2020. *The Science of Computing III Living with Cyber Student Edition*. Retrieved February 10, 2020 from <https://coes.latech.edu/wp-content/uploads/sites/7/2018/07/130-student.pdf>
- [15] C. Hacker and R. Sitte. 2004. Interactive teaching of elementary digital logic design with WinLogiLab. *IEEE Transactions on Education* 47, 2 (May 2004), 196–203. DOI:<https://doi.org/10.1109/TE.2004.824843>
- [16] M.H.H. Ichsan and W. Kurniawan. 2018. Design and implementation 8 bit CPU architecture on Logisim for undergraduate learning support. In *Proceedings - 2017 International Conference on Sustainable Information Engineering and Technology, SIET 2017*, Institute of Electrical and Electronics Engineers Inc., 132–137. DOI:<https://doi.org/10.1109/SIET.2017.8304123>
- [17] B.S. Jong, C.H. Lai, Y.T. Hsia, T.W. Lin, and C.Y. Lu. 2013. Using game-based cooperative learning to improve learning motivation: A study of online game use in an operating systems course. *IEEE Transactions on Education* 56, 2 (2013), 183–190. DOI:<https://doi.org/10.1109/TE.2012.2207959>
- [18] Z. Kurmas. 2017. ICOS: Support for “bare metal” computer architecture assignments. In *Proceedings of the 19th Workshop on Computer Architecture Education, WCAE 2017*, Association for Computing Machinery, Inc, 10–15. DOI:<https://doi.org/10.1145/3116214.3116240>
- [19] W. Kurniawan and M.H.H. Ichsan. 2017. Teaching and learning support for computer architecture and organization courses design on computer engineering and computer science for undergraduate: A review. In *2017 4th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, Institute of Electrical and Electronics Engineers (IEEE), 1–6. DOI:<https://doi.org/10.1109/eecsi.2017.8239076>

- [20] E. Larraza-Mendiluze and N. Garay-Vitoria. 2015. Approaches and tools used to teach the computer input/output subsystem: A survey. *IEEE Transactions on Education* 58, 1 (2015), 1–6. DOI:<https://doi.org/10.1109/TE.2014.2310711>
- [21] J.H. Lee, S.E. Lee, H.C. Yu, and T. Suh. 2012. Pipelined CPU design with FPGA in teaching computer architecture. *IEEE Transactions on Education* 55, 3 (2012), 341–348. DOI:<https://doi.org/10.1109/TE.2011.2175227>
- [22] Louisiana Tech University. 2020. CSC 265: Introduction To Digital Design - Acalog ACMS™. *University Course catalog*. Retrieved February 10, 2020 from https://catalog.latech.edu/preview_course_nopop.php?catoid=10&coid=34486
- [23] Louisiana Tech University. 2020. CSC 364: Computer Architecture - Acalog ACMS™. *University Course catalog*. Retrieved February 10, 2020 from https://catalog.latech.edu/preview_course_nopop.php?catoid=10&coid=34491
- [24] Louisiana Tech University. 2020. CSC 521: Advanced Computer Architecture - Acalog ACMS™. *University Course catalog*. Retrieved February 10, 2020 from https://catalog.latech.edu/preview_course_nopop.php?catoid=10&coid=34522
- [25] J. Lowe-Power and C. Nitta. 2019. The Davis In-Order (DINO) CPU A Teaching-focused RISC-V CPU Design. In *Proceedings of the Workshop on Computer Architecture Education, WCAE 2019*, Association for Computing Machinery, Inc. DOI:<https://doi.org/10.1145/3338698.3338892>
- [26] A. Martínez-Monés, E. Gómez-Sánchez, Y.A. Dimitriadis, I.M. Jorrín-Abellán, B. Rubia-Avi, and G. Vega-Gorgojo. 2005. Multiple case studies to enhance project-based learning in a computer architecture course. *IEEE Transactions on Education* 48, 3 (August 2005), 482–489. DOI:<https://doi.org/10.1109/TE.2005.849754>
- [27] J. Newman. 2016. Megaprocessor - Frames. *Megaprocessor.com*. Retrieved April 30, 2020 from <http://www.megaprocessor.com/frames.html>
- [28] J. Newman. 2016. Megaprocessor - Boards. *Megaprocessor.com*. Retrieved April 30, 2020 from <http://www.megaprocessor.com/boards.html>
- [29] J. Newman. 2016. Megaprocessor - Modules. *Megaprocessor.com*. Retrieved April 30, 2020 from <http://www.megaprocessor.com/modules.html>
- [30] J. Newman. 2016. Megaprocessor - Architecture. *Megaprocessor.com*. Retrieved April 8, 2020 from <http://www.megaprocessor.com/architecture.html>

- [31] J. Newman. 2016. Megaprocessor - State Machine. *Megaprocessor.com*. Retrieved April 8, 2020 from http://www.megaprocessor.com/arch_fsm.html
- [32] J. Newman. 2016. Megaprocessor- Instruction Set. *Megaprocessor.com*. Retrieved April 8, 2020 from http://www.megaprocessor.com/instruction_set.pdf
- [33] J. Newman. 2016. Megaprocessor - peripherals. *Megaprocessor.com*. Retrieved April 4, 2020 from <http://www.megaprocessor.com/peripherals.html>
- [34] J. Newman. 2016. Megaprocessor - Igor. *Megaprocessor.com*. Retrieved April 4, 2020 from <http://www.megaprocessor.com/igor.html>
- [35] B. Nikolic, Z. Radivojevic, J. Djordjevic, and V. Milutinovic. 2009. A survey and evaluation of simulators suitable for teaching courses in computer architecture and organization. *IEEE Transactions on Education* 52, 449–458. DOI:<https://doi.org/10.1109/TE.2008.930097>
- [36] B. Poduel, P. Kansakar, S.R. Chhetri, and S.R. Joshi. 2015. Design and Implementation of Synthesizable 32-bit Four Stage Pipelined RISC Processor in FPGA Using Verilog/VHDL. *Nepal Journal of Science and Technology* 15, 1 (February 2015), 81–88. DOI:<https://doi.org/10.3126/njst.v15i1.12021>
- [37] M. Reichenbach, B. Pfundt, and D. Fey. 2014. Designing and manufacturing of real embedded multi-core CPUs: A holistic teaching approach in computer architecture. In *10th European Workshop on Microelectronics Education, EWME 2014*, IEEE Computer Society, 213–218. DOI:<https://doi.org/10.1109/EWME.2014.6877428>
- [38] M.J. Santofimia and F. Moya. 2009. Nintendo DS: A Pedagogical Approach to Teach Computer Architecture. In *ESA*, 269–273. Retrieved February 8, 2020 from <https://www.researchgate.net/publication/221218080>
- [39] L. Shanshan and L. Weidong. 2019. THINPAD experimental platform for computer hardware experiment. In *14th International Conference on Computer Science and Education, ICCSE 2019*, Institute of Electrical and Electronics Engineers Inc., 792–795. DOI:<https://doi.org/10.1109/ICCSE.2019.8845431>
- [40] O.A. Siddiqui, S. Mahmood, R. Hasan, and A.R. Khan. 2012. Simulators as a teaching aid for computer architecture and organization. In *Proceedings of the 2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics, IHMSC 2012*, 110–113. DOI:<https://doi.org/10.1109/IHMSC.2012.33>
- [41] H. Tomari and K. Hiraki. 2015. Keeping old computers alive for deeper understanding of computer architecture. In *Workshop on Computer*

Architecture Education, WCAE 2015, Association for Computing Machinery, Inc. DOI:<https://doi.org/10.1145/2795122.2795127>

- [42] M. Walter and S. Karlsson. 2017. Software tools for low-level software and operating systems classes. In *Proceedings of the 19th Workshop on Computer Architecture Education, WCAE 2017*, Association for Computing Machinery, Inc, 16–23. DOI:<https://doi.org/10.1145/3116214.3116241>
- [43] G.S. Wolffe, W. Yurcik, H. Osborne, and M.A. Holliday. 2002. Teaching computer organization/architecture with limited resources using simulators. In *SIGCSE Bulletin (Association for Computing Machinery, Special Interest Group on Computer Science Education)*, ACM Press, New York, New York, USA, 176–180. DOI:<https://doi.org/10.1145/563517.563408>