My Final College Paper

A Thesis
Presented to
The Division of Mathematical and Natural Sciences
Reed College

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Arts

Sebastian Andrada Ottonello

May 2024

Approved for the Division
(Computer Science)

Charles McGuffey

Acknowledgements

I want to thank a few people.

Preface

This is an example of a thesis setup to use the reed thesis document class.

List of Abbreviations

You can always change the way your abbreviations are formatted. Play around with it yourself, use tables, or come to CUS if you'd like to change the way it looks. You can also completely remove this chapter if you have no need for a list of abbreviations. Here is an example of what this could look like:

ALU	Arithmetic Logic Unit
CISC	Complex Instruction Set Computer
CPU	Central Processing Unit
ISA	Instruction Set Architecture
RISC	Reduced Instruction Set Computer
RISC-V	Reduced Instruction Set Computer Five

Table of Contents

List of Tables

List of Figures

Abstract

The preface pretty much says it all.

Dedication

You can have a dedication here if you wish.

Introduction

Though there's countless pieces of hardware to study and delve into within this field, the CPU (Central Processing Unit) can be argued to be the most essential one. It is the heart of any computer, the piece that actually carries out the programs and computations that it was built for. It comes as no surprise, then, that most college-level Computer Systems courses begin by taking a close look at the inner workings of a simple CPU, so that they can then begin to understand how the other components of a computer support its functionality, and how it can produce computation out of just an assembly program, wires, and logic gates. Learning of this is all well and good, but getting hands-on experience with the topic is harder than one might expect: The powerful CPUs in the personal computers of students run on are all closed-source designs that obfuscate how they carry out their instructions; and even if they were transparent, the sheer breadth and depth of modifications they employ to get the performance they do would make their design completely indecipherable to a novice.

Therefore, the aim of this project is to develop a pedagogical simulator program, developed in the Rust programming language and capable of running on modern computers, that simulates a simple RISC-V CPU at the hardware level, running assembly code and then transparently displaying how the code and computed information traverse the inside of the processor. While simple, the simulated CPU implements a 5-stage pipeline design similar to the ones that are mandatory for modern CPUs. The project also seeks to include some user-facing features, like a graphical interface and the ability to step through and rewind through the execution of a program, to make the simulator more intuitive to use.

0.1 Background

0.1.1 What is an ISAs?

But first, what does it even mean that the CPU is RISC-V? RISC-V is one of many ISAs, belonging to the RISC family of design.

Starting from the top, ISA stands for Instruction Set Architecture; you may have also heard it reffered to as just a CPU's "architecture", or of popular ISAs like x86 and ARM. We will go into the details later, but for now an ISA is simply a list of machine-code instructions that a CPU can carry out: These instructions are things like basic arithmetic operations (addition, subtraction, etc), calls to load or store data from the memory, or even jump instructions to alter the flow of the program.

Ultimately, a program is just a list of these instructions, one after the other, that the CPU executes in order. A CPU is considered to use a certain ISA if it supports and has implemented all of its instructions. ISAs were created for the goal of hardware compatibility: As long as two different CPUs made by competing manufacturers both use the same ISA, any low-level program will run on both processors because they both guarantee that they'll support the same instruction set, no matter how different their underlying design is.

Ultimately, an ISA is a standard for how a CPU should take any given line of code from a low-level program - encoded as an integer in binary, this close to the hardware - and translate that into an actionable operation that the CPU should execute when it reaches that part of the program.

0.2 Why RISC-V?

Though the ISA doesn't ascribe an exact design for how its demands should be met(that's called the *implementation*, and is the domain of the CPU's manufacturer), it so heavily shapes the CPU's capabilities, and therefore indirectly its design, that choosing which ISA to implement is the most important decision one can make when designing a CPU, whether it's real or virtual. As alluded to in the introduction, there are multiple ISAs to choose from, and many more popular ones than RISC-V. So, what makes competing ISAs so different from one another, and why choose RISC-V for this project instead of any of the more popular options?

RISC-V is a modern ISA, albeit one that has not yet experienced widespread success; while it very rapidly growing in use, its two contemporaries, x86 and ARM, dominate the Personal Computer CPU and Mobile CPU market entirely for the time being. And yet, RISC-V has two major advantages over them that make it the ideal choice for this simulator.

The first is, simply, that RISC-V is fully open source. Unlike its two proprietary cousins, the full listings of all the RISC-V instructions are freely available, with details on what behavior is expected of each instruction and exactly how to glean an instruction's parameter from its 32-bit integer representation. Working with open-source sorftware and specifications is good by its very nature in that its development benefits everyone, not just its creators, but it's also a better choice practically.

The second difference is that RISC-V is a RISC ISA, while x86 is a CISC ISA; it's even in the name. These stands for *Reduced Instruction Set Computer* and *Complex Instruction Set Computer* respectively, and they are opposing schools of design for an ISA. The difference ultimately comes down to how many different instructions the ISA lists out: The idea of a CISC ISA is that by providing a very large variety of different instructions, most individual things you'd want a program to do can be performed with just one instruction each. By contrast, a RISC ISA aims to keep its total number of instructions low: While this would mean that code would need more total instructions to accomplish the same task, the idea is that keeping the demands simpler would allow RISC processors to have faster and more efficient designs that offset this cost [1]. This also coincidentally makes RISC processors a much better example for students to study, while still remaining a modern architecture that's growing in widespread use.

0.3 An abstract view at how a CPU works

In order to build and understand the design of a CPU, we must first understand what it, in general terms, does. We've already established that a CPU carries out a program consisting of instructions from its ISA. However, what exactly do those instructions demand the CPU do, and what does the CPU have access to to carry it out?

We will go over the exact design of the model of CPU shown here later, with all of its intricacies and small parts. But, in broad terms, the CPU has access to: an **Instruction Memory**, where the program to be executed is stored instruction-by-instruction. A **Decoder** that reads this raw 32-bit instruction and gleans the important details from it, like what instruction it is, what the inputs and outputs are, or other instruction-specific data. A **Register Memory**, which holds a small amount of numbered registers that store information; think of them as set mini-variables built directly into the CPU that it has quick and easy access to. A **Data Memory**, external and much larger than the previous two, in which to store larger values; and finally an **ALU**, Arithmetic Logic Unit, which is simply capable of performing arithmetic operations on the values it is given. This explanation leaves out many crucial components and smaller parts that make the whole thing possible, but they're enough to understand the CPU's general plan, detailed here:

1. Read the first instruction from the Instruction Memory.**(IF: Instruction Fetch)**

2. Decode the fetched instruction, and read the values of whichever registers the instruction calls for. **(ID: Instruction Decode)**

3. Perform whichever arithmetic operation the instruction calls for, using values from the registers or from the instruction itself as input. **(EX: Execute)**

4. If the instruction calls for a value to be read from or stored to the Data Memory, do so. **(MEM: Memory)**

5. Write back the end-result value of the instruction into the destination register it specified. **(WB: WriteBack)**

6. Read the next instruction from the Instruction Memory, and start over again at step 2. Repeat until end of program.

These five repeating steps are enough to accomplish everything that base version of RISC-V calls for, shockingly! It is enough for the CPU to successfully run low-level programs. How these simple instructions build up to the complex operating systems and graphical applications that most are familiar with today is beyond the scope of this project, but the important thing is that it is enough.

It is also enough to get a rough idea of how any given RISC-V instruction is carried out. The full list of instructions can be found on the official *RISC-V Instruction Set*

[1] Denning (1993)

Manual that is freely available online, or on a number of more user-friendly guides like Five EmbedDev's *RISC-V ISA Instruction Quick Reference*. A list of all the instructions in the most basic version of RISC-V, the RV32I Base Integer Instruction Set, can be found (INSERT HERE), but the instructions can be broadly divided into the following categories

Control Transfer: Instructions that call for the CPU to move on to a place in the code that is NOT the next instruction, potentially. These are what make the If-Then statements and Loops of high-level programming languages possible.

Store/Load: The purpose of these instructions are simply to interact with the Data Memory, either storing a value to a specific address or reading a value from it.

Register-Register Operation: These instructions take two registers as input, perform some sort of operation on their two values (an arithmetic operation like addition, or a logical operation like a bitwise AND or NOT), and store the result in a third register.

Register-Immediate Operation: These are the same as RR (Register-Register) instructions, but take as input one register and one hard-coded value. It's the difference between "z = x + y" and "z = x + 3".

0.4

0.5 Bibliographies

Of course you will need to cite things, and you will probably accumulate an armful of sources. This is why BibTeX was created. For more information about BibTeX and bibliographies, see our CUS site (`web.reed.edu/cis/help/latex/index.html`)[2]. There are three pages on this topic: *bibtex* (which talks about using BibTeX, at `/latex/bibtex.html`), *bibtexstyles* (about how to find and use the bibliography style that best suits your needs, at `/latex/bibtexstyles.html`) and *bibman* (which covers how to make and maintain a bibliography by hand, without BibTeX, at at `/latex/bibman.html`). The last page will not be useful unless you have only a few sources. There used to be APA stuff here, but we don't need it since I've fixed this with my apa-good natbib style file.

0.5.1 Tips for Bibliographies

1. Like with thesis formatting, the sooner you start compiling your bibliography for something as large as thesis, the better. Typing in source after source is mind-numbing enough; do you really want to do it for hours on end in late April? Think of it as procrastination.

2. The cite key (a citation's label) needs to be unique from the other entries.

[2] Reed College (2007)

3. When you have more than one author or editor, you need to separate each author's name by the word "and" e.g.
`Author = {Noble, Sam and Youngberg, Jessica},`.

4. Bibliographies made using BibTeX (whether manually or using a manager) accept LaTeX markup, so you can italicize and add symbols as necessary.

5. To force capitalization in an article title or where all lowercase is generally used, bracket the capital letter in curly braces.

6. You can add a Reed Thesis citation[3] option. The best way to do this is to use the phdthesis type of citation, and use the optional "type" field to enter "Reed thesis" or "Undergraduate thesis". Here's a test of Chicago, showing the second cite in a row[4] being different. Also the second time not in a row[5] should be different. Of course in other styles they'll all look the same.

0.6 Anything else?

If you'd like to see examples of other things in this template, please contact CUS (email cus@reed.edu) with your suggestions. We love to see people using LaTeX for their theses, and are happy to help.

[3] Noble (2002)
[4] Noble (2002)
[5] Reed College (2007)

Chapter 1

Mathematics and Science

1.1 Math

TEX is the best way to typeset mathematics. Donald Knuth designed TEX when he got frustrated at how long it was taking the typesetters to finish his book, which contained a lot of mathematics.

If you are doing a thesis that will involve lots of math, you will want to read the following section which has been commented out. If you're not going to use math, skip over this next big red section. (It's red in the .tex file but does not show up in the .pdf.)

1.2 Chemistry 101: Symbols

Chemical formulas will look best if they are not italicized. Get around math mode's automatic italicizing by using the argument `$\mathrm{formula here}$`, with your formula inside the curly brackets.

So, $\mathrm{Fe_2^{2+}Cr_2O_4}$ is written `$\mathrm{Fe_2^{2+}Cr_2O_4}$`
Exponent or Superscript: $\mathrm{O^-}$
Subscript: $\mathrm{CH_4}$

To stack numbers or letters as in $\mathrm{Fe_2^{2+}}$, the subscript is defined first, and then the superscript is defined.
Angstrom: Å
Bullet: $\mathrm{CuCl \bullet 7H_2O}$
Double Dagger: ‡
Delta: Δ
Reaction Arrows: $\longrightarrow$ or $\xrightarrow{solution}$
Resonance Arrows: $\leftrightarrow$
Reversible Reaction Arrows: $\rightleftharpoons$ or $\xrightleftharpoons{solution}$ (the latter requires the chemarr package)

1.2.1 Typesetting reactions

You may wish to put your reaction in a figure environment, which means that LaTeX will place the reaction where it fits and you can have a figure legend if desired:

$$C_6H_{12}O_6 + 6O_2 \longrightarrow 6CO_2 + 6H_2O$$

Figure 1.1: Combustion of glucose

1.2.2 Other examples of reactions

$NH_4Cl_{(s)} \rightleftharpoons NH_{3(g)} + HCl_{(g)}$
$MeCH_2Br + Mg \xrightarrow[below]{above} MeCH_2 \bullet Mg \bullet Br$

1.3 Physics

Many of the symbols you will need can be found on the math page (`http://web.reed.edu/cis/help/latex/math.html`) and the Comprehensive LaTeX Symbol Guide (enclosed in this template download). You may wish to create custom commands for commonly used symbols, phrases or equations, as described in Chapter **??**.

1.4 Biology

You will probably find the resources at `http://www.lecb.ncifcrf.gov/~toms/latex.html` helpful, particularly the links to bsts for various journals. You may also be interested in TeXShade for nucleotide typesetting (`http://homepages.uni-tuebingen.de/beitz/txe.html`). Be sure to read the proceeding chapter on graphics and tables, and remember that the thesis template has versions of Ecology and Science bsts which support webpage citation formats.

Chapter 2

Tables and Graphics

2.1 Tables

The following section contains examples of tables, most of which have been commented out for brevity. (They will show up in the .tex document in red, but not at all in the .pdf). For more help in constructing a table (or anything else in this document), please see the LaTeX pages on the CUS site.

Table 2.1: Correlation of Inheritance Factors between Parents and Child

Factors	Correlation between Parents & Child	Inherited
Education	-0.49	Yes
Socio-Economic Status	0.28	Slight
Income	0.08	No
Family Size	0.19	Slight
Occupational Prestige	0.21	Slight

If you want to make a table that is longer than a page, you will want to use the longtable environment. Uncomment the table below to see an example, or see our online documentation.

Table 2.2: Chromium Hexacarbonyl Data Collected in 1998–1999

Chromium Hexacarbonyl			
State	Laser wavelength	Buffer gas	Ratio of $\frac{\text{Intensity at vapor pressure}}{\text{Intensity at 240 Torr}}$
$z^7P_4^\circ$	266 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	0.57
$y^7P_3^\circ$	266 nm	Argon	1
$y^7P_3^\circ$	355 nm	Argon	0.14
$y^7P_2^\circ$	355 nm	Argon	0.14
$z^5P_3^\circ$	266 nm	Argon	1.2
$z^5P_3^\circ$	355 nm	Argon	0.04
$z^5P_3^\circ$	355 nm	Helium	0.02
$z^5P_2^\circ$	355 nm	Argon	0.07
$z^5P_1^\circ$	355 nm	Argon	0.05
$y^5P_3^\circ$	355 nm	Argon	0.05, 0.4
$y^5P_3^\circ$	355 nm	Helium	0.25
$z^5F_4^\circ$	266 nm	Argon	1.4
$z^5F_4^\circ$	355 nm	Argon	0.29
$z^5F_4^\circ$	355 nm	Helium	1.02
$z^5D_4^\circ$	355 nm	Argon	0.3
$z^5D_4^\circ$	355 nm	Helium	0.65
$y^5H_7^\circ$	266 nm	Argon	0.17
$y^5H_7^\circ$	355 nm	Argon	0.13
$y^5H_7^\circ$	355 nm	Helium	0.11
a^5D_3	266 nm	Argon	0.71
a^5D_2	266 nm	Argon	0.77
a^5D_2	355 nm	Argon	0.63
a^3D_3	355 nm	Argon	0.05
a^5S_2	266 nm	Argon	2
a^5S_2	355 nm	Argon	1.5
a^5G_6	355 nm	Argon	0.91
a^3G_4	355 nm	Argon	0.08
e^7D_5	355 nm	Helium	3.5
e^7D_3	355 nm	Helium	3
f^7D_5	355 nm	Helium	0.25
f^7D_5	355 nm	Argon	0.25
f^7D_4	355 nm	Argon	0.2
f^7D_4	355 nm	Helium	0.3
Propyl-ACT			

State	Laser wavelength	Buffer gas	Ratio of $\frac{\text{Intensity at vapor pressure}}{\text{Intensity at 240 Torr}}$
$z^7P_4^\circ$	355 nm	Argon	1.5
$z^7P_3^\circ$	355 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	1.25
$z^7F_5^\circ$	355 nm	Argon	2.85
$y^7P_4^\circ$	355 nm	Argon	0.07
$y^7P_3^\circ$	355 nm	Argon	0.06
$z^5P_3^\circ$	355 nm	Argon	0.12
$z^5P_2^\circ$	355 nm	Argon	0.13
$z^5P_1^\circ$	355 nm	Argon	0.14
Methyl-ACT			
$z^7P_4^\circ$	355 nm	Argon	1.6, 2.5
$z^7P_4^\circ$	355 nm	Helium	3
$z^7P_4^\circ$	266 nm	Argon	1.33
$z^7P_3^\circ$	355 nm	Argon	1.5
$z^7P_2^\circ$	355 nm	Argon	1.25, 1.3
$z^7F_5^\circ$	355 nm	Argon	3
$y^7P_4^\circ$	355 nm	Argon	0.07, 0.08
$y^7P_4^\circ$	355 nm	Helium	0.2
$y^7P_3^\circ$	266 nm	Argon	1.22
$y^7P_3^\circ$	355 nm	Argon	0.08
$y^7P_2^\circ$	355 nm	Argon	0.1
$z^5P_3^\circ$	266 nm	Argon	0.67
$z^5P_3^\circ$	355 nm	Argon	0.08, 0.17
$z^5P_3^\circ$	355 nm	Helium	0.12
$z^5P_2^\circ$	355 nm	Argon	0.13
$z^5P_1^\circ$	355 nm	Argon	0.09
$y^5H_7^\circ$	355 nm	Argon	0.06, 0.05
a^5D_3	266 nm	Argon	2.5
a^5D_2	266 nm	Argon	1.9
a^5D_2	355 nm	Argon	1.17
a^5S_2	266 nm	Argon	2.3
a^5S_2	355 nm	Argon	1.11
a^5G_6	355 nm	Argon	1.6
e^7D_5	355 nm	Argon	1

2.2 Figures

If your thesis has a lot of figures, LaTeX might behave better for you than that other word processor. One thing that may be annoying is the way it handles "floats" like tables and figures. LaTeX will try to find the best place to put your object based on the text around it and until you're really, truly done writing you should just leave it where it lies. There are some optional arguments to the figure and table environments to specify where you want it to appear; see the comments in the first figure.

If you need a graphic or tabular material to be part of the text, you can just put it inline. If you need it to appear in the list of figures or tables, it should be placed in the floating environment.

To get a figure from StatView, JMP, SPSS or other statistics program into a figure, you can print to pdf or save the image as a jpg or png. Precisely how you will do this depends on the program: you may need to copy-paste figures into Photoshop or other graphic program, then save in the appropriate format.

Below we have put a few examples of figures. For more help using graphics and the float environment, see our online documentation.

And this is how you add a figure with a graphic:

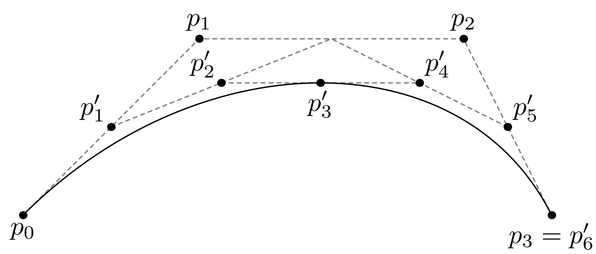

Figure 2.1: A Figure

2.3 More Figure Stuff

You can also scale and rotate figures.

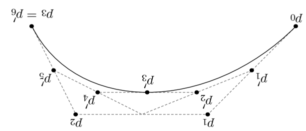

Figure 2.2: A Smaller Figure, Flipped Upside Down

2.4 Even More Figure Stuff

With some clever work you can crop a figure, which is handy if (for instance) your EPS or PDF is a little graphic on a whole sheet of paper. The viewport arguments are the lower-left and upper-right coordinates for the area you want to crop.

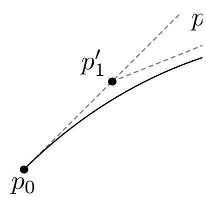

Figure 2.3: A Cropped Figure

2.4.1 Common Modifications

The following figure features the more popular changes thesis students want to their figures. This information is also on the web at `web.reed.edu/cis/help/latex/graphics.html`.

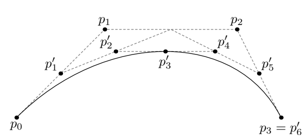

Figure 2.4: Subdivision of arc segments. You can see that $p_3 = p'_6$.

Conclusion

Here's a conclusion, demonstrating the use of all that manual incrementing and table of contents adding that has to happen if you use the starred form of the chapter command. The deal is, the chapter command in LaTeX does a lot of things: it increments the chapter counter, it resets the section counter to zero, it puts the name of the chapter into the table of contents and the running headers, and probably some other stuff.

So, if you remove all that stuff because you don't like it to say "Chapter 4: Conclusion", then you have to manually add all the things LaTeX would normally do for you. Maybe someday we'll write a new chapter macro that doesn't add "Chapter X" to the beginning of every chapter title.

4.1 More info

And here's some other random info: the first paragraph after a chapter title or section head *shouldn't be* indented, because indents are to tell the reader that you're starting a new paragraph. Since that's obvious after a chapter or section title, proper typesetting doesn't add an indent there.

Appendix A

The First Appendix

Appendix B

The Second Appendix, for Fun

References

Angel, E. (2000). *Interactive Computer Graphics : A Top-Down Approach with OpenGL*. Boston, MA: Addison Wesley Longman.

Angel, E. (2001a). *Batch-file Computer Graphics : A Bottom-Up Approach with QuickTime*. Boston, MA: Wesley Addison Longman.

Angel, E. (2001b). *test second book by angel*. Boston, MA: Wesley Addison Longman.

Denning, P. J. (1993). The science of computing: Risc architecture. *American Scientist*, *81*(1), 7–10. http://www.jstor.org/stable/29774812

Deussen, O., & Strothotte, T. (2000). Computer-generated pen-and-ink illustration of trees. *"Proceedings of" SIGGRAPH 2000*, (pp. 13–18).

Fisher, R., Perkins, S., Walker, A., & Wolfart, E. (1997). *Hypermedia Image Processing Reference*. New York, NY: John Wiley & Sons.

Gooch, B., & Gooch, A. (2001a). *Non-Photorealistic Rendering*. Natick, Massachusetts: A K Peters.

Gooch, B., & Gooch, A. (2001b). *Test second book by gooches*. Natick, Massachusetts: A K Peters.

Hertzmann, A., & Zorin, D. (2000). Illustrating smooth surfaces. *Proceedings of SIGGRAPH 2000*, *5*(17), 517–526.

Jain, A. K. (1989). *Fundamentals of Digital Image Processing*. Englewood Cliffs, New Jersey: Prentice-Hall.

Molina, S. T., & Borkovec, T. D. (1994). The Penn State worry questionnaire: Psychometric properties and associated characteristics. In G. C. L. Davey, & F. Tallis (Eds.), *Worrying: Perspectives on theory, assessment and treatment*, (pp. 265–283). New York: Wiley.

Noble, S. G. (2002). *Turning images into simple line-art*. Undergraduate thesis, Reed College.

Reed College (2007). Latex your document. http://web.reed.edu/cis/help/LaTeX/index.html

Russ, J. C. (1995). *The Image Processing Handbook, Second Edition*. Boca Raton, Florida: CRC Press.

Salisbury, M. P., Wong, M. T., Hughes, J. F., & Salesin, D. H. (1997). Orientable textures for image-based pen-and-ink illustration. *"Proceedings of" SIGGRAPH 97*, (pp. 401–406).

Savitch, W. (2001). *JAVA: An Introduction to Computer Science & Programming*. Upper Saddle River, New Jersey: Prentice Hall.

Wong, E. (1999). *Artistic Rendering of Portrait Photographs*. Master's thesis, Cornell University.