Table of Contents

Milestone 1

Dec 19, 2023 9:00 AM | Comp Arch CSSE 232

Notes

- Day 1 of Project: Milestone 1
- Setup GitHub repo
- Discussed some basic ideas of what Memory-Memory would look like

Action items

- ☐ Meeting later today to discuss how to make instructions for memory-to-memory

Dec 19, 2023 6:00 PM | Comp Arch Meeting

Notes

- We have transitioned from a memory-to-memory model to an accumulator-based architecture due to the constraint of 16-bit instructions
- Most of our instructions will follow a similar structure allotting 4 bits for the opcode and 12 bits for the arguments
- I came up with the idea to implement stack-based instructions, namely loadsp, and storesp, which utilize the Last-In-First-Out (LIFO) principle. This choice was made to restrict the total number of instructions to 16, enabling us to keep 4-bit size opcodes and 12 bits for address space
- After further discussion, we've settled on using 5 bits for the opcode and 11 for addressing. This setup enables us to create clearer instructions and eliminates the need for a Last-In-First-Out (LIFO) stack approach. We're now able to handle the stack

similarly to how it's done in RISC-V, a shift made possible since we needed space for 'jal' and 'jalr'.

Action items

- [ ] Try to run through the relPrime code to see if there are types/instructions we have not thought about

Dec 20, 2023 12:00 PM | Comp Arch

Notes

- When Deciding how we wanted to do the SLT instruction, we started talking about having multiple accumulators
- Decided that we would have four different accumulators
- This means we need to add a 2-bit ID to certain instructions to specify where to store certain operation value
- We decided to make accumulator 2 the branching accumulator. This means any comparisons must use Accumulator 2. This allows us to keep the branch address reasonably sized
- We decided to keep arguments on the stack. Since we have an arbitrary amount of arguments for a function, it would be the easiest thing to do.
- We are making it a requirement that the user always stores ra in the stack first before going to a procedure. They must store the arguments IN ORDER. This allows our stack operations to open up bigger chunks of space. Before, if you had to specify where in the stack you wanted to access an element, that means you could open up more space on the stack than you could access. We wanted to avoid this.
- jal can jump even further. We will implicitly override the value of RA and expect the user to store RA on the stack before doing a jal anywhere
- StoreSp and LoadSp are only used for placing arguments and return addresses on the stack

Action items

- [ ] Finish off making the types for the rest of our instructions
- [ ] Keep looking through relPrime to see if there is anything we still need to account for.
- [ ] Run By some of our ideas to Professor Williamson

Dec 21, 2023 6:00 PM | Comp Arch Team Meeting

Notes

- We talked about how to fix the StoreSp and LoadSp problems. We aren’t letting the user tell us where they want to store/load values as of now. This is so they can access the

same amount of bits that they open. Decided that lui would always load an 11-bit immediate into R[2]. If we let the user specify the register, they would be unable to store as big of values.
- Made a doc for questions to ask Professor
- Added the shift right arith, ori, xori, andi, slli, srli, srai
- Jay updated all the opcodes for the instructions; Liz and I wrote descriptions for the instructions and code examples
- Havalock did a lot on the reference sheet outside of the meeting since he would not be able to join this meeting.
- Wrote the register sheet
- Register 1 is called reggie1, register 2 is called branch since it is used for branching, register 3 is called reggie3, and register 4 is called reggie4
- Typed the high-level description while Jay and Liz throw

Action items
- ☐ Talk to the professor tomorrow
- ☐ Go through relPrime
- ☐ FINISH

Dec 22, 2023 | Team Meeting with Professor

Notes
- We talked about our register design and what each is used for
- Talked about how the return address and the inputs to a function will be used on the stack
- Need to show some code examples of our architecture such as for loops, recursive, while loops, etc.
- Memory allocation will be very similar to RISC-V but needs to adjust since there are 32-bits
- Need to take a look at a reserved area
- Convert all examples and realprime to binary

Action items
- ☐ Write assembler
- ☐ Write common code examples in our architecture

Jan 4, 2024 | Comp Arch Work

Notes

- Worked on the assembler and some working examples on the design document
- Found some cases we did not touch on (.globl, .data, etc)
- I plan to share the assembler on GitHub so all my partners can use it
- Able to read from an Excel file or Txt file
- Correctly parses I-types
- Correctly parses S-types
- Other types need some more team discussion

Action items

- ☐ Account for the other types
- ☐ Add more to the design document

Jan 5, 2024 | Comp Arch

Notes

- Working on relPrime conversion
- Wondering if we should do our stack differently. Instead of putting the ra first, put it last. This would make more sense since the 0sp would be arg1, 4sp arg2, and so on, then once all are used, the last value will just be the ra
- We need to discuss where our open and close will happen. Will the user close right after a method is done?
- Where do return values get stored?
- I am thinking maybe we should let a user specify where in the stack to retrieve and store elements

Action items

- ☐ Figure out how return values will be dealt with
- ☐ Bring up other ideas of how to deal with procedures
- ☐ Finish up assembler

Jan 8, 2024 | Comp Arch

Notes

- We changed how we are doing stack operations. Previously we decided to not allow the user to decide where to store/load stuff on the stack. We are now following Risk-V Conventions to allow the user to specify where on the stack they are storing and loading values. This makes it easier for the user and us to keep track of the stack. This also makes it so that procedures can only have 7 arguments.

- Updated reference sheet fixing some inconsistencies we had
- We reserved Reggie 4 (R[3]) as the register that holds return values. Any other values that need to be returned will be put on the stack.
- Users will be responsible for storing registers that hold important values in memory on their own. Once we make the memory map, we will allocate 4 spots for them to do this
- Working on code fragments in our architecture
- We added a jalr procedure. This would take no arguments as we always store ra at 0x on the stack. This means we will add a new type.
- This means when a user goes to a procedure, the first thing they store is the Ra, and after that is the arguments in order
- Assembler is on GitHub to make machine-coding translations
- The team finished up code Fragments and Memory Allocation

Action items

- ☐ Finish Up the Assembler

Jan 9, 2024 | Comp Arch

Notes

- Fixing lots of problems in the reference sheet.
- Decided on starting sp at 0xFFF8 so that we can use the extra space above it to store constants 1, 0, and -1 which are commonly used.
- Using a .word keyword that we expect the assembler to handle. This will tell the assembler that a variable is to be stored in memory. Static data starts at 0x1000
- Decided to use the heap to store reggie values. The heap starts at 0x2000. This will allow the user to keep track of saved data. Reggie[3] won't be saved since this holds return values and I want the storing of reggies and loading of reggies to be controlled by us. Not the user.
- Plan to bring up ideas for other instructions we could add to make some operations easier
- Mostly just a lot of work on code fragment updates.
- I believe adding a swap instruction between registers will help with some operations. Sometimes we need certain values that are stored in register x to be in register y. A swap makes it easier for the user to interchange them
- Added an alter type. This will take in 2 registers and a return register, and do some operation between them. There are a lot of instances where an operation is done between registers, and we don’t want to complicate things by moving stuff to memory and doing an operation.
- This all took about 3 hours
- After finishing relPrime, I started to convert to Machine Code. The assembler did most of the work. Preparing some drawings for a group meeting so everyone is on the same page

- We need to either fix our memory map or start using lui more to load in bigger addresses to use as arguments. Highlighted problems in red

Action items

- ☐ Finish up translations for team meeting
- ☐ Work on assembler

Milestone 2

Jan 10, 2024 | Comp Arch

Notes

- Worked some more on the assembler
- hexToBinary and decToBinary now fully account for positive and negative values. I added some unit tests to make sure. Once we figure out our memory map, I can finish the last few types and pre-process data

Action items

- ☐ Talk to the professor and team about the Memory Map
- ☐ Walk through Rel Prime Again

Jan 10, 2024 6:00 PM | Comp Arch Team Meeting

Notes

- Walked through relPrime with new information
- Alter and Swap are official instructions
- Alter can add, subtract, load0, and load1 onto a register
- Whenever we have a jal to a non-procedure, we need to have some space in the stack for the jal to store the RA. So we decided on always opening up a spot on the stack whenever we know we will have such a jal

Action items

- ☐ Work on Assembler
- ☐ Prepare for the meeting tomorrow
- ☐ Milestone 2 soon

Jan 11, 2024 | Comp Arch CSSE 232

Notes

- More work on Assembler
- Can print out addresses with the binary values
- Need to account for new types
- Need to fix branching

Action items

- [ ] Fix Assembler

Jan 11, 2024 | Comp Arch CSSE 232

Notes

- Meeting with Professor about M1
- Jay did note-taking
- Mostly just missing info on our design document. We need to put everything in one place (reference Jay's doc)
- Upload PDFs into git-repo

Action items

- [ ] Finish Assembler today
- [ ] Look over RTL before the M2 Meeting

Jan 11, 2024 8:00 PM | Comp Arch Meeting

Notes

- ASSEMBLER DONE 🦍🦍🦍🦍🦍🦍(hopefully)
- The code structure is as follows:
    - main.py
        - Deals with user input and starts up assembler class
        - Gives the user the option to input a Txt or Excel file
        - .txt files come with options of removing spacing and adding the addresses that each instruction is run on and where static variables are stored in static memory

        - Excel files let the user choose which column in a MICROSOFT Excel to read from, and what column to write the data on. Columns must exist. The results will be written on a new sheet named Assembler Results
        - assembleInstruction method has a dictionary of all our operations with their binary code and type. Takes an Assembler instance and a string representation of the instruction. Based on the type of instruction, it will tell the Assembler class to handle it appropriately
    - Assembler.py
        - Where most logic happens
        - Is a class file
        - Keeps track of current address, static address, alter codes, Reggie conversions, and symbols (such as tags and static variable names)
        - Has a preprocessing method for Txt files to find symbols before being parsed to binary
        - Holds methods to handle every type of instruction. Catches most faults that would come from user error (improper naming, listing instructions that don't exist, etc.)
    - helpers.py
        - Holds functions that do conversions of decimal, hexadecimal, and binary values
        - Learned Python is NOT great for this
        - Some helpful functions included hexToBinary, hexToDec, isHex, isDec, and withinBitLength
    - Tests folder
        - Just some unit tests for the helper functions and the pre-processing of data. Wanted to test these separately
    - Also gives the option to print out binaries spaced out by operands, the total number of bits each instruction is(hopefully 16), with the address they would be in code (assuming they are the first lines ran on the program), and keeping in-line comments. The program will prompt the user if they want a detailed output. Mostly for debugging purposes
- Worked on Milestone 2 with team
- No major decisions were made at this meeting. Just focused on making RTL instructions

Action items

- [ ] Work more on M2
- [ ] Going to New York this weekend so the workload will probably see a decline

Milestone 3

Jan 16, 2024 | Comp Arch meeting

Notes

- Went over points we were not too confident about in M2. Mostly some RTL instructions. These are thighs we will get feedback from in our group meeting with the professor
- Started working on making RTL instructions for Multi-Processing
- Was not able to contribute much since I was not hear for the multicycle processor lecture. Will hopefully be caught up after copying the notes today and setting up a meeting for tomorrow
- Made the design document neat by adding a fleshed-out table of contents 😅

Action items

- ☐ Understand Multi-Cycle Processing

Jan 17, 2024 | Comp Arch Lab CSSE 232

Notes

- In the second half of class, we worked on our RTL instructions.
- No major decisions were made this weekend
- We are leaning towards a multi-cycle processor, due to the fact we have some instructions that do a lot of mini instructions behind the scenes. Using a single-cycle processor would cause these instructions to take a very long time and slow down our processor. No final decision made yet
- Will have a meeting with the professor later today to touch on some iffy concepts in multi-cycle processing

Action items

- ☐ Meeting with prof at 2:30 PM

Jan 17, 2024 | Comp Arch Team Meeting

Notes

- Decided on a Multi-Cycle processor. The biggest drivers for this decision were instructions like jal and Jb.

- Ran into a problem with our jb and jal instructions where we were moving sp and hp(stack and heap pointers) each cycle back to where they started. The easiest solution to this problem is to have some variable take whatever is in the value of sp and hp and increment that variable instead of hp and sp directly.
- This also meant we would need some global way to track where sp and hp are in memory. The easiest solution would be to have each one as a register. The user would have 0 access to these registers, but it would be the easiest way to keep track of the address of both the sp and hp

Action items

- ☐ Milestone meeting

Jan 18, 2024 | Comp Arch CSSE 232

Notes

- Milestone 2 meeting with professor
- We decided to keep sp and hp in memory since we are trying to balance the cost and efficiency of our processor. We save a lot of money not having 2 more registers, and the only times we mess with sp and hp are when we are about to jump to other procedures. This isn't super common so we can afford the time it takes to read/write to memory the new values of the addresses
- Started talking about addressing the issues in our M2 and moving on with M3 (notes in Jay's Journal ~ Jan 18th, 2024)

Action items

- ☐ Fix the Professor's suggestion
- ☐ Work on M3

Jan 18, 2024 | Comp Arch

Notes

- Small team meeting to finish off RTL

- Liz and Havalock worked on finishing all of the professor's suggestions from the earlier meeting, while Jay and I finished up multi-cycle RTL and components with the professor's suggestion in mind
- Created a shared Visio so we can start creating the datapath next meeting

Jan 21, 2024 | Comp Arch

Notes

- Me and Jay worked on the datapath while Liz and Havalock worked on Quartus
- Wrote the data path to work for out R, I, B, and J types. Decided on the ALU to have 1 output that decides on when we branch, and 4 separate opcodes for the operation it should do for each branch. This makes it easier for us to deal with branching by just putting the responsibility on the ALU
- Noticed some inconsistencies with our RTL instructions and what we intended for certain operations to do. I went through and fixed some logic with jal jalr to make them follow PC relative rules and was able to reduce a cycle on jal
- Fixed load, store, loadsp, and storesp. They were messing with the sp directly instead of storing values into ALUout so as not to affect the original. Also wasn't broken down enough

Action items

- ☐ Finish off datapath
- ☐ Walk though rtl to make sure everything makes sense

Jan 22, 2024 | Comp Arch

Notes

- Me and Jay worked on the datapath while Liz and Havalock worked on Quartus just like the day before
- Noticed more problems in our RTL where we were putting things that should take 2 + cycles and labeling them as 1
- Finishing off S types, A types and C types for our datapath

DATAPATH NOTES(Because I keep forgetting what certain wires were for)

R Types:

- Things being decoded
  - rID ← Reg[inst[6:5]] main register being dealt with
  - MDR ← Mem[inst[15:7]] address passed in
  - offset ←SE(inst[15:7]) For load and stores to offset sp
- SRC Mux changes
  - ALUSrcA takes the imm
  - ALUSrcB takes MDR
- Extra Notes
  - Whenever a register gets written onto memory, we assume it will always be in r1
  - Added mem2reg mux for load instruction

I Types:

- Nothing notable

B Types:

- Note
  - The mux has 1 signal that deals with all branches
  - Since PC would be calculated on cycle 2, cycle 3 will do the branch logic and use the PC from previous cycle to set current PC if necessary.
  - Does this using branch and shouldBranch signals

J Types:

- Note
  - Added the IorD signal to deal with the heap needing to be constantly written too when doing a jal
  - Needed a second MDR since we are changing the stack and the heap
- SRC Mux Changes
  - Switched what I said in R types
  - ALUSrcB takes the imm
  - ALUSrcA takes MDR
  - This is because we have a cycle that does MDR + 2, so to save a wire instead of giving SrcA the imm, we can instead give it MDR since 2 is already in our second mux
  - ALUSrcA takes ALUOut to increment/decrement by 2 when messing with the heap address

S Types:

- SRC Mux Changes
  - Had to make ALUSrcB take a 3-bit signal
  - Needed to hard code 5 as a value to do a bit shift with LUI so ALUSrcB has 5 inputs

C Types:

- Mem2Reg mux has to be changed since we needed to have a reg2reg operation. It is now a RegFileSrc signal

A Types:

- Notes
    - Had to match the alter opcodes with the original ALUOpcodes to avoid adding a second ALU

Action items

- [ ] Look at Quartus Stuff with Havalock and Liz

Jan 23, 2024 | Comp Arch CSSE 232

Notes

- Team Meeting in the classroom
- Fixed the github by adding a .gitignore and removed files that were causing merge conflicts
- Updated the design document with the current datapath, our decision of implementing a multi-cycle processor, and some templates to fill out later
- Need to fix the assembler. The addresses in jump instructions are not PC relative in the assembler

Action items

- [ ] Fix Assembler
- [ ] Catch up on M#

Milestone 4

Jan 25, 2024 | Comp Arch

Notes

- Worked on figuring out a plan to fix everything wrong with M3 such as our RTL not following the rules of our design document, the datapath not working like we excepted it too, and implementing tests without actually having the test plan in our document 😅
- Added different types of immediate extensions depending on the opcode of our instruction
- I added better documentation for our addressing mode to further explain the different types of immediate extensions
- As a team we made a gameplan for how we want the next few days to look:

    - Friday: Fix RTL. Potentially Data path
    - Saturday: Fix data path. Assign 2 people to start M4 and the other 2 to do lab 07
- Added a syntax and schematics section in the design

Action items

- [ ] Fix descriptions in the design document

Jan 26, 2024 | Comp Arch with Havalock

Notes

- Me and Havalock went through and fixed the multi-cycle RTL
- We decided to make to immediate generators. One for sign extending, and one for what we call address extending for R-types
- Finished up the first 2 cycles for all RTL instructions. All of that is in the reference sheet
- Decided that our sp and hp would be registers that would start off at a hardcoded value because it would be easier to deal with

Jan 27, 2024 | Comp Arch

Notes

- This was a group meeting to work on the datapath
- We are deciding if an ALU control is needed. I am in favor of it. Having everything in the main control will be hard to deal with. I plan on doing some more research to see why Risc-V does it
- Exaplined the immediate Generator to the group(which was explained to me during a meeting)
- Started delegating some more responsibilities:
    - Me - Will work on the Assembler today. Tomorrow around noon I will be working on the datapath with Jay and I plan on working on Lab07
    - Jay - Working with me on the datapath and will also work on the control unit with Liz
    - Havalock - Will work on Lab07 with me, and working on writing some Verilog components
    - Liz - Working on the control unit, Integration testing plan, and Wriitng our component testing in the design document

Jan 27, 2024 | Personal Comp Arch Work

Attendees:

Notes

- Updating the assembler since we have changed things about our instruction set
- Doing some research on the ALU control and why it's used
    - Learned that the general reason is a separation of concerns and helps lighten the complexity of the main Control unit

Action items

- [ ] Assembler and Datapath

Jan 28, 2024 | Comp Arch

Notes

- These are notes for the whole weekend
- Went back and updated the assembler since we've made a decent amount of changes since the last time we worked on this. It took about 4-5 hours to get everything to work
- Worked on our Datapath. Added the sp and hp registers and the wiring to get R-types and I-types to work.
- Added what the control values should be at each cycle of these two instruction types
    - All this was about 2-3 hours
- There are some bugs we need to talk about in person for assembler *lui and static vars)

Action items

- [ ] Finish Data path with the group

Jan 29, 2024 | Comp Arch

Notes

- Almost finished datapath
- The reason it is not fully finished, is because we potentially want to delete HP and move its responsibilities to SP. This would mean we have 1 less register, and all we need to do is adjust some addressing modes. This will reduce a lot of the hardware, in general,

- did not want to make this decision without the whole team (Havalock was sick)
- Worked some more on the assembler and noticed that our idea of having local variables would not work. At least, not how we thought they would
    - For now, we will focus on getting something to work, then maybe come back to this
- Finished up the control diagram for everything but jal/jb
- Jay and Liz were responsible for the code

Action items

- ☐ Update the design document to have everything it needs for M4

Jan 30, 2024 4:00 PM | Comp Arch CSSE 232

Notes

- Fixing up the design document before it is due
- Adding a truth table for ALUOut & fixing up all our descriptions
- Making a backup datapath, RTL, and FSM for if we decide to remove the HP and replace it with the SP, which I think will end up being our decision. Also, note to add this into our addressing mode. They will not appear on submission for M4. The reason for removing HP is to reduce the cost of the datapath.
- Also updating the documentation on signals since a few were added since they were last documented

Jan 30, 2024 7:00 PM | Comp Arch

Notes

- Went over the choice of removing HP. Ended up going through. Split of the responsibilities as follows:
    - Havelock updated all documentation and controls
    - Jay updated the RTL
    - Liz updated any verilog code that depended on HP/Needs to test more
    - I updated the Control and Datapath

Action items

- ☐ Finish the Control and Datapath

Jan 30, 2024 | Personal Comp Arch Work

Notes

- Finished the datapath and control
- Added a new alu operation: add6+ which adds ALUSrc1 and ALUSrc2 and 6. This is for addressing. It will save us a cycle on storesp and loadsp
- Updated the design doc as much as possible. Tried to make sure everything reflected the changes

Changes Made:

<u>New signals:</u>

DataSrc - picks what data memory reads
0 = r1
1 = PC

What this affects:
R-types
Jal
IorD now has a value for 11 (SP)

Changes in control:
R types now need to account for the DataSrc signal. So DataSrc = 0 whenever they are writing to memory.
jal will have DataSrc = 1

OperandSrc - picks what data gets loaded into r1(not the accumulator)
00 = inst[6:5]
01 = r0
10 = r1
11 = r2

What this affects:
All nodes

Changes in control:
Almost all insturctions will specify the OperandSrc to be 0. Technically this is the default value for signals anyways
Jal will have this signal change per cycle starting from cycle 4

Branch is now 2 bits!!
For Jump back since we set PC to whatever is on the stack, our mux will have a new input
This affects B types and J types since they are now 2 bits. Also our cycle 2 (Which every instruction uses)

ReturnSRC is now 3 bits!!
This affects instructions that write back onto a register

Action items

- ☐ Review design document and Control to make sure everything is correct.

Feb 1, 2024 | Comp Arch CSSE 232

Notes

- Milestone 4 meeting
- Things we need to do:
    - A way to take input and output. Will probably remove some instructions (The non-immediate shifts lol)
    - What do we want to do when a program is “Finished”
    - Verilog needs to initiate our registers properly
    - Should diagram a reset signal

Action items

- ☐ Meeting later tonite to address these issues
- ☐ Hopefully can start working on lab 7 with Havalock

Milestone 5

Feb 1, 2024 | Comp Arch

Notes

- Started coming up with solutions form the M4 meeting earlier in the day
- I went in and updated the control to make sure signals were all right, and set some to 0 as a reminder (since they were 1 from previous cycles)
- Jay and Liz worked on fixing the verilog code for the control unit
- Me and Havalcok went over how we wanted to deal with inputs, outputs, and stopping a program.
    - Combined opensp and closesp to just be on instruction (movesp) that takes a positive/negative immediate to move the sp
    - Moved sra to our alter instruction. Decided on this one since we don't see it being used often, and won't affect the datapath or control. If a user wants to do a shra, they will have to load both values into registers now
    - With 3 spots open for instructions, we added an input, output, and stop instruction (which are all J-types)
- I am going through all the design documents, reference sheets, and the Assembler to reflect the update
- Havalock is updating the datapath and control to reflect the update.

Action items

- ☐ Finishing updating all the documents and assembler

Feb 2, 2024 | Comp Arch

Notes

- Finished updating all the documents and assembler
- Found some inconsistencies that we need to address for our Sunday meeting

Action items

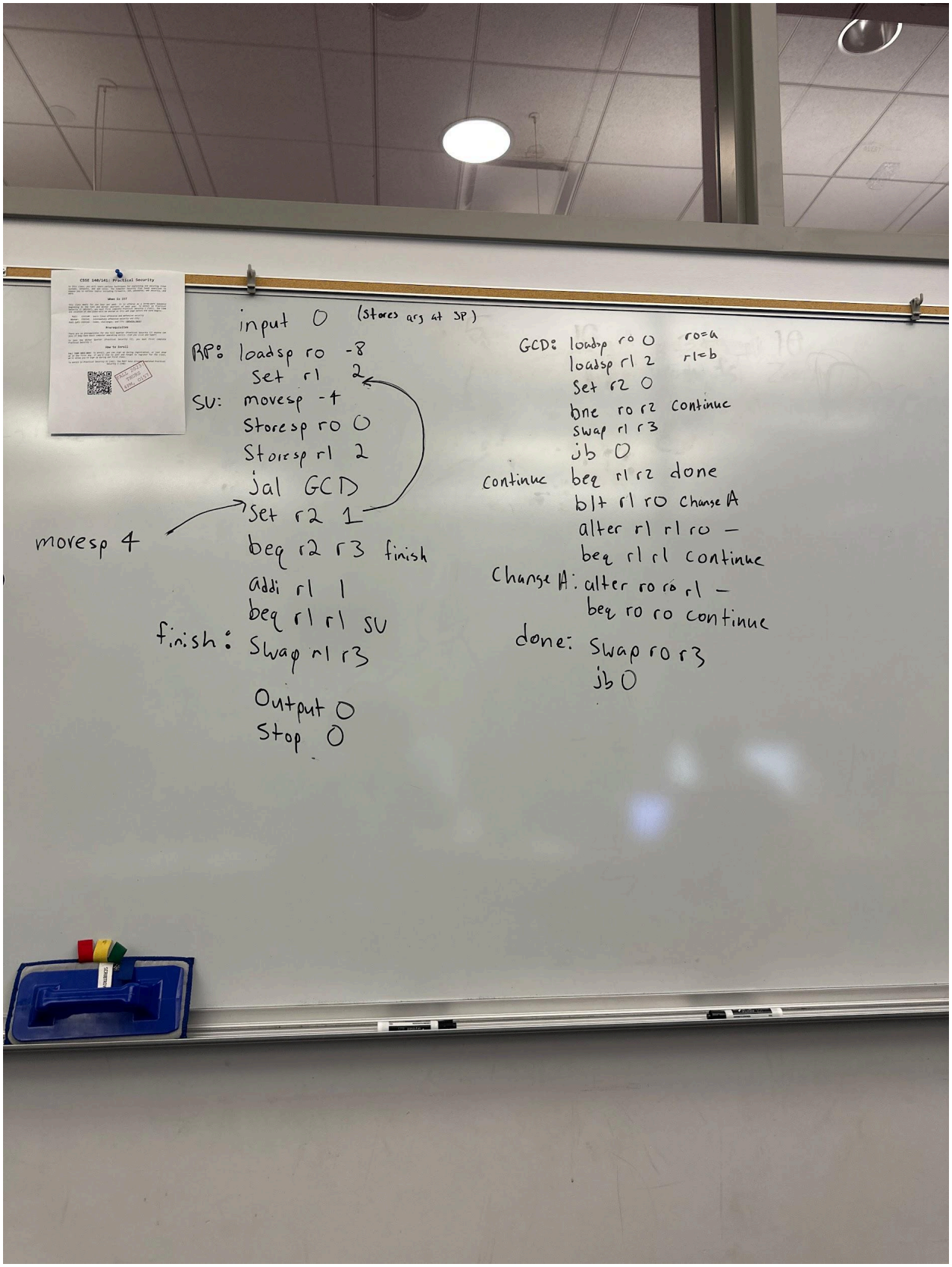
- ☐ Coordinate with Havalock about lab07

Feb 4, 2024 | Comp Arch

Notes

- All hands on deck team meeting
- Liz and Jay worked on the test implementation for some components in Verilog

- Me and Havalock worked on adding input, output, and stop to the datapath, control, and design documents
- We fixed some inconsistencies with jal and jb and how they were working
- Fixed our addressing mode to add 8 instead of 6. We are now going to make jal automatically add space for ra instead of just storing ra. This allows for less overhead by the user
- We added a set instruction and moved srl to alter. Since we don't have a load immediate, the set instruction makes setting registers to an immediate value a lot easier. Alter can handle what srl does with 1 extra instruction, but it does not get used often anyway.
- Me and Havalock went over relPrime again with our new instructions. It takes fewer lines than it did before


CSSE 140/141: Practical Security
input 0 (stores arg at SP)
RP: loadsp r0 -8
Set r1 2
SU: movesp -4
Storesp r0 0
Storesp r1 2
jal GCD
Set r2 1
beq r2 r3 finish
addi r1 1
beq r1 r1 SU
finish: Swap r1 r3
Output 0
Stop 0
movesp 4
GCD: loadsp r0 0 r0=a
loadsp r1 2 r1=b
Set r2 0
bne r0 r2 continue
swap r1 r3
jb 0
continue beq r1 r2 done
blt r1 r0 changeA
alter r1 r1 r0 -
beq r1 r1 continue
changeA: alter r0 r0 r1 -
beq r0 r0 continue
done: Swap r0 r3
jb 0

- Lots of changes will need to be made. Since we have replaced certain instructions with different types, and moved things around, certain opcodes in components need to be updated. We have divided the work as follows:
    - I am updating the assembler
    - Havalock is updating the immediate generator
    - Liz is updating the ALU and ALU Control
    - Jay is updating the Control

Action items

- [ ] Update the assembler

Feb 5, 2024 | Comp Arch

Notes

- Spent the whole meeting fixing the assembler and making sure relPrime would be parsed correctly
- Assembler is officially done. Almost every instruction has been unit-tested, and relPrime was tested by hand.
- The assembler can also give outputs in hex values for even easier debugging purposes
- The rest of the team worked on Verilog

Action items

- [ ] Start working on M6

Milestone 6

Feb 8, 2024 | Comp Arch Project

Feb 9, 2024 | Comp Arch Project

Notes

- Journal entry for both the 8th and 9th
- Worked on combining the RegFile, Imm generator, the stack pointer, and all appropriate muxes/wiring
- Finished making the component and made a test bench
- Ran into some bugs while testing. Both the imm generator component and my assembler has something to fix
    - My assembler was not catching errors when a user tried to branch/jump too

    - The imm generator was not calculating the correct immediate

Action items

- [ ] Fix both bugs you are running into

Feb 12, 2024 | Comp Arch CSSE 232

Attendees:

Notes

- Fixed both my assembler and the testbench for the component I have been working on. The assembler now tells the user if they jump too far, and the immediate generator sends back the right immediates. The problem was with how it bit-shifted -> sign extended immediates.

Feb 14, 2024 | Comp Arch Project

Notes

- Worked on testbench that has everything for our processor but out of control
- Specifically worked on checking J-types
- I am pretty sure the RTL for Lui we have is wrong, as it takes 2 cycles for LUI to fully progate. Have to figure out what we are doing wrong
- Made it to where we can run multi-line programs through our test benches by moving blocks of code to tasks

Action items

- [ ]

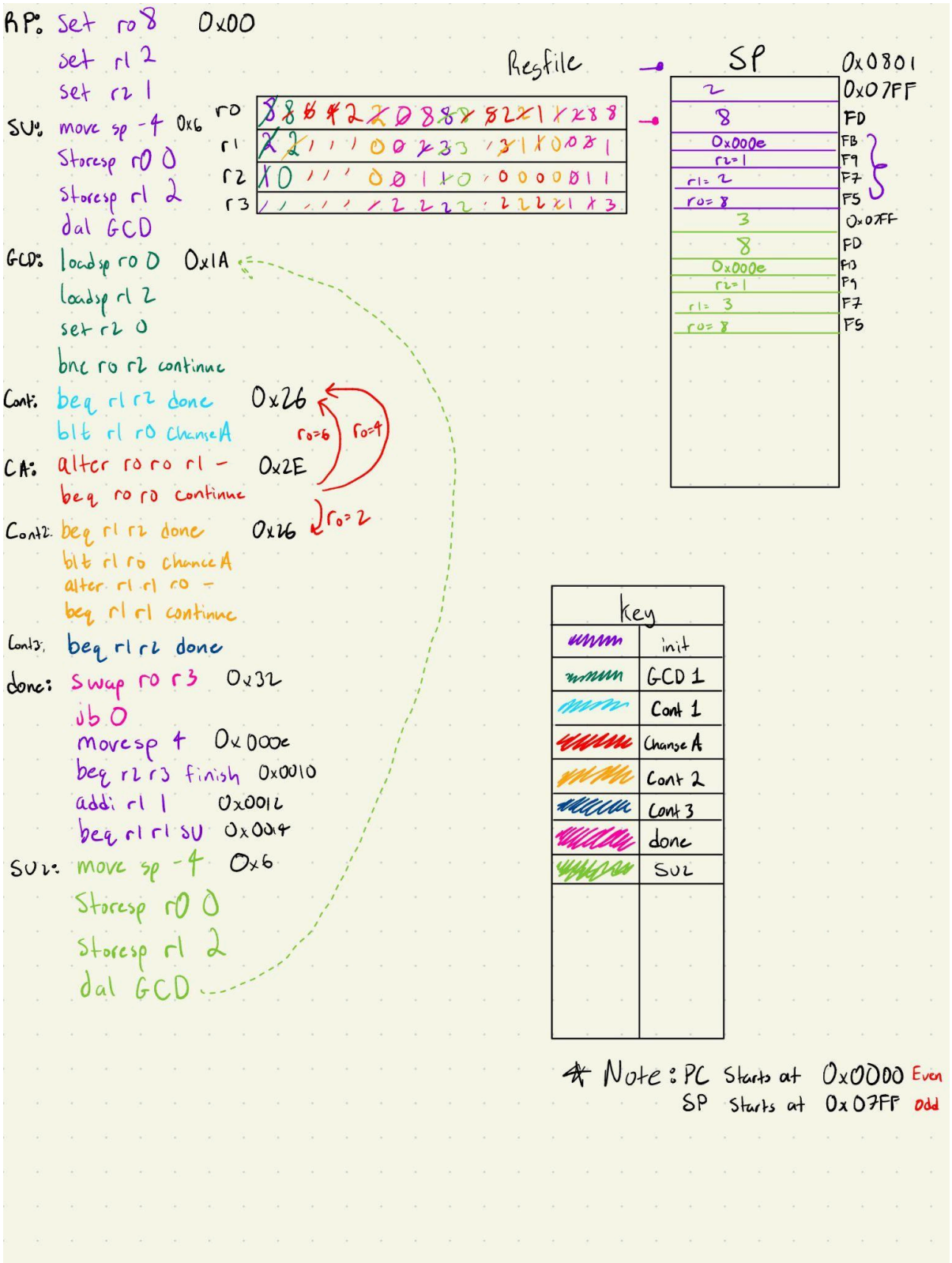
Feb 15, 2024 | Comp Arch Project

Notes

- Grind out day
- Spending all day getting relPrime to work without the control (called headless files)
- Ran into plenty of bugs that me and Havalock mostly fixed

- Spent more hours than I'd like to say doing this
- Made a nice integration plan that makes what we are testing clear. Trashed what we were doing yesterday because there wasn't a good test plan in place.
- Used the assembler to come up with what memory should look like, while changing it a little bit. Since our cycles always added 2 to PC, that would mean a branch/jal would always go one line too far since the assembler calculated the number of lines from the jal/branch call. Not the next line
- .
- .
- .
- .
- Got it done 😎

Below is a document I used to help keep track of values for the integration plan and coding purposes:



- Ended up writing 70 tests (All fully documented in the phase 3 integration test plan)
- I also tested some R-Types since those weren't used in RelPrime. Both in the integration plan and made some extra tests.

Action items

- [ ] Connect the Control!!!

Feb 16, 2024 | Comp Arch

Notes

- Me and Havalock worked on getting I/O to work
- I worked on adding test cases for it in the headless file while Havalock worked on connecting the control
- We ran into a few bugs, but got I/O to work in the headless file pretty easily
- We then started working on the full processor
    - Ran into multiple bugs with the control 🥲
    - Debugging took a few hours
- WE GOT RELPRIME WORKING DONE

Action items

- [ ] Finish M6 (Performance) and update any documentation and our slides
- [ ] Plan on giving tasks tomorrow to clean up documentation

Feb 17, 2024 | Comp Arch Project Meeting

Notes

- Explained the processor to Liz and Jay
- Liz said she would take on M6 and add some info to slides
- Jay was assigned to update the code snippets on our design document
- Havalock was going to update the memory section on the design document
- I am not doing anything

Action items

- [ ] Meeting tomorrow to go over slides and anything we could be missing

Feb 19, 2024 | Comp Arch

Notes

- Tiyding up all the design documents, Tedious work

Feb 20, 2024 | COMP ARCH WHAT

Notes

- Unexpected all hands on deck comp arch meeting
- We thought the processor wasn't working when we tried to receive a second input…
- Our relPrime code was wrong 🥲🥲🥲🥲
- Fixed relPrime, tidied up slides, and prepared for presentation

Action items

- ☐ 🦍