

## MILESTONE 1 WORK:

Monday, January 8, 2024:

- Refactored/finalized instruction set [1.5 hours]
- Determined instruction types [1.5 hours]
- Completed Instruction Set table [1 hour]

Tuesday, January 9, 2024:

- Wrote assembly examples for each instruction [1 hour]
- Translated relprime into machine code [1 hour]
- Wrote example common operation snippets [30 min]
- Finished up memory map, made modifications to expanded instruction set table [30 min]

## MILESTONE 2 WORK:

Thursday, January 11, 2024:

- Ethan gave hardware demo in Sandbox: showed locations+functionality of main register bus, special dedicated registers (sp/ra/pc)
  - Since instructions all have immediates in same location but of differening length, immediate genies for imm. instructions handled right after decoder, MUX selects them
- Began drafting document with RTL instructions, completed 10W

Friday, January 12, 2024:

- Ethan and I worked on refactoring document.
  - We decided on recursive calling conventions: local vars that need to be preserved between calls should be backed up+restored in saved registers, since this is more of a "callee responsibility"
  - Assembled example programs [20 mins]
  - I redid the memory map, Ethan fixed the example program instructions to follow it
  - Decided to refactor instruction formats to table format for readability
  - Examined verilog parts we plan to use: using equality that xors to a standard bitwidth reg rather than alu subtraction

Monday, January 15, 2024:

- Completed RTL for RETURN [10 min]
- Drafted component list [2 hrs]:

- We decided (for the time being, we will likely refactor into multicycle in the future and change things)
- there will be 4 separate ALUs:
  - One for simple 2-input logic instructions (Add, and, etc.)
  - One for branch instructions, as they output a single-bit flag + use gates rather than adders to do comparisons
  - One for PC/SP modification instructions, so output can be wired directly to pc/sp/ra
  - One for other instructions, that will likely be divided up in the future (since each of these instructions uses their own logic)
- Massive instruction decoder determines input flags for ALUs (might refactor this into opcodes in the future/clean it up)
- Cleaned up RTL based on new component list: fixed naming conventions to more clearly demonstrate what components were being used (we were using A/B/C letter variables for everything when these are reserved for loaded regs) [30 min]
- Drafted RTL error checking process and fixed some RTL errors based on it [30 min]

## MILESTONE 3 WORK:

### Thursday, January 18, 2024:

- Updated RTL, moving it away from trying to describe exact hardware components to more of a general overview of what flows to what in a single-cycle path.
  - Consolidated instructions that only differ by opcode (arithmetic + branching instructions) [30min]
- Updated RTL Error-checking process: We felt that drawing a diagram of the current states of the register files + stack frame would help determine if behavior is accurate, since these are the main components where instruction data is stored/retrieved (and the main places things could go wrong) [30min]
- Began writing L\_Register\_File [30min]

### Friday, January 19, 2024:

- Continued working on register file, fixed bugs -- Ethan refactored it to use existing components rather than hard-code everything like muxes [30min]

### Saturday, January 20, 2024:

- Began implementing L\_ALU\_11 in Verilog. L\_ALU\_11 handles the main arithmetic operations (Type 4WRR instructions) [1hr]

- We decided to stick the main instruction decoder unit inside of the ALU, and have the ALU not only handle all instruction operations but also the setting of read/write flags and immediate sign extension
- Instructions start by flowing through the decoder, then are sent to their appropriate sub-ALU. ALUs are divided based on where their outputs go and what output flags they need to set.
- We felt this would be more efficient since all of the instruction flags and whether immediates are SE'd are inevitably tied to what operation the ALU does

## Sunday, January 21, 2024:

- Began writing testbench for L\_ALU\_11: T\_L\_ALU\_11 [30 min]
- Completed bench but could not get it to work in ModelSim

## Monday, January 22, 2024:

- Refactored Verilog datapath diagram into a Google Drawings diagram for readability [2 hours]
- Worked on Datapath Implementation Plan in document. [1 hr]
  - I would like to include an additional diagram showing the ALU since it takes on so many responsibilities
- Reworked the L\_ALU\_11 test: ditched exhaustive testing because testing every single combination takes too long and is difficult to debug [1 hr]
  - Decided to instead test binary multiples of 8: this gives us tests of positive and negative numbers on both positions for every operation without making things too long
- Began updating component list with new ALU structures [30 mins]
- Added control signal descriptions [15 mins]
- 1 hour team meeting:**
  - Decided it's not worthwhile to do unit tests since our ALU components are so simple: we are focusing on integration testing instead
  - We decided to implement and test memory before Friday
  - Agnay, Wenzhi, are going to implement the ALU subcomponents next week
  - My task: modify assembler:
    - takes in program file (what you will run), put this in instruction space
    - init file that has a few instructions that initialize the environment before jumping to the program
    - outputs to a file

## MILESTONE 4 WORK:

Tuesday, January 23, 2024:

- Started modifying Python assembler to support memory initialization [1.5 hours]