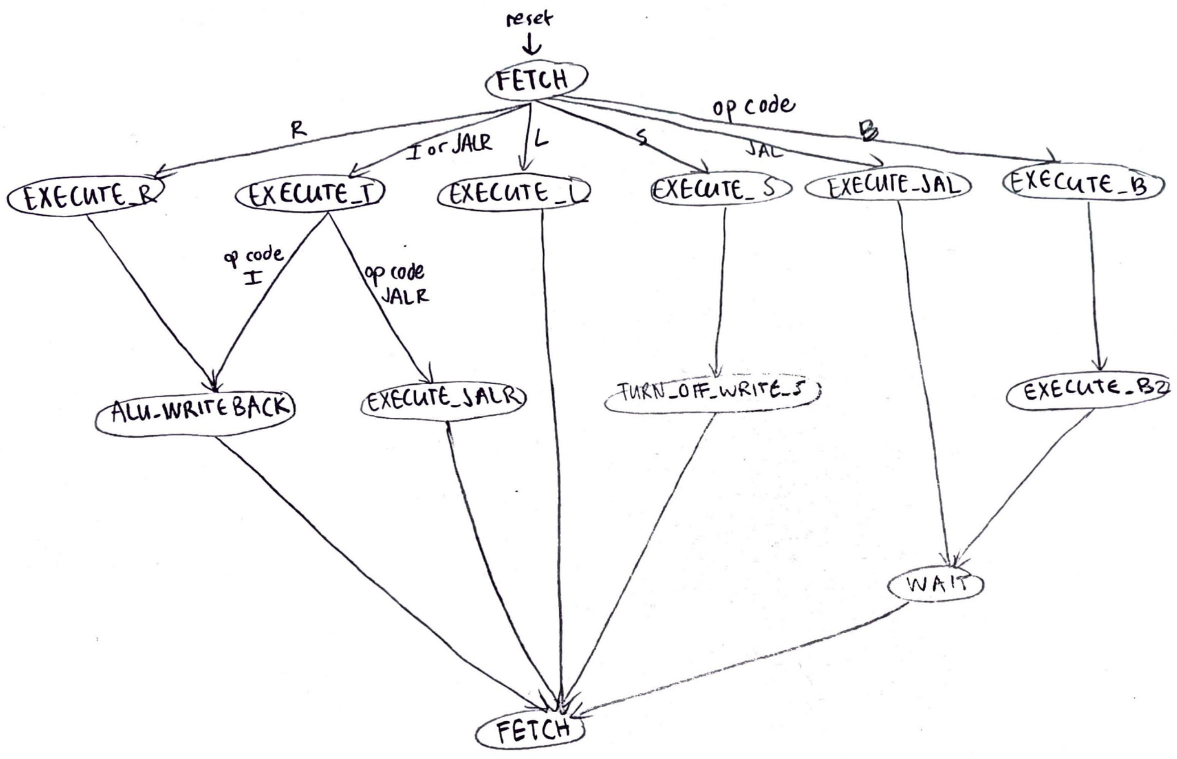
RISC-V CPU

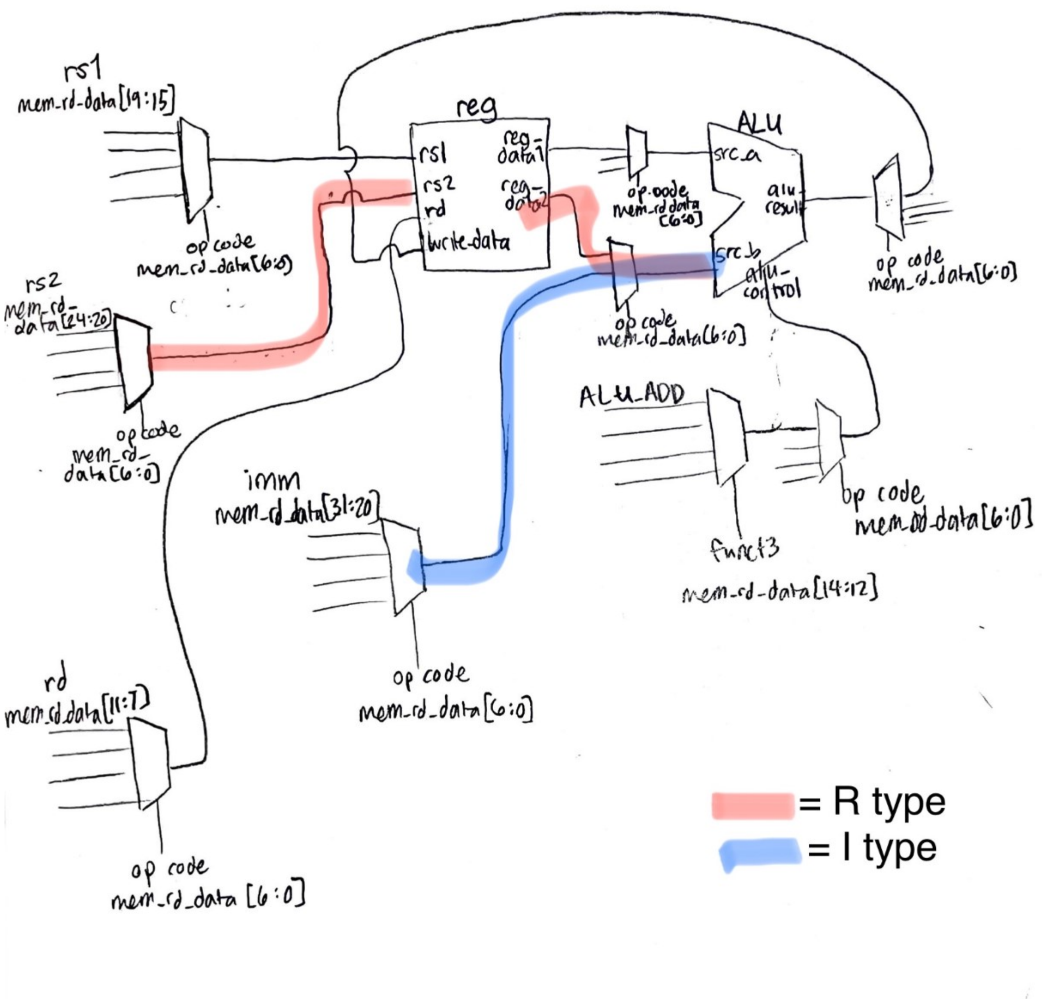
Computer Architecture Lab 3, Fall 2022
Lilo Heinrich and Emma Mack

Main FSM

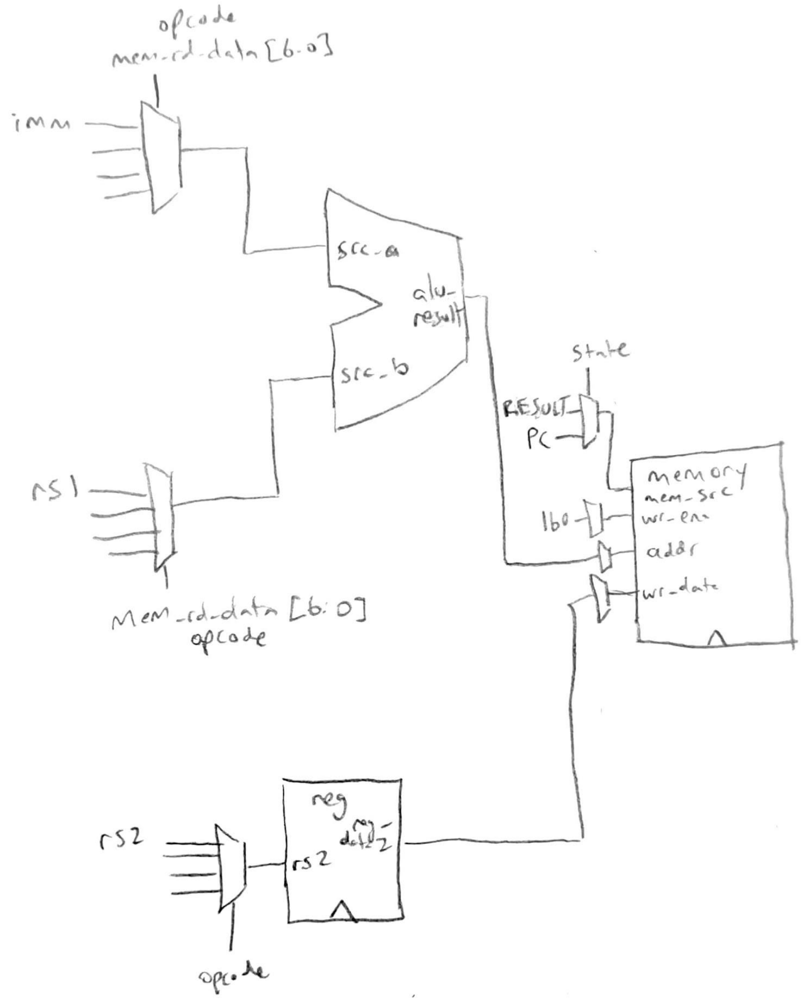


Reset takes the state into FETCH, which has case statements on the assembly instruction to determine which opcode is next. Based on the opcode, FETCH sets up the initial steps of the instruction, then directs the state to that of the opcode. Most opcodes are not able to accomplish everything in one state, due to needing to wait for registers or the ALU to return values, thus some direct into states like ALU_WRITEBACK. EXECUTE_JALR and EXECUTE_B2 both need to direct into a WAIT state before returning to FETCH, otherwise the PC experiences issues with being updated at the wrong time. Eventually, all state paths lead back to FETCH, and the cycle repeats.

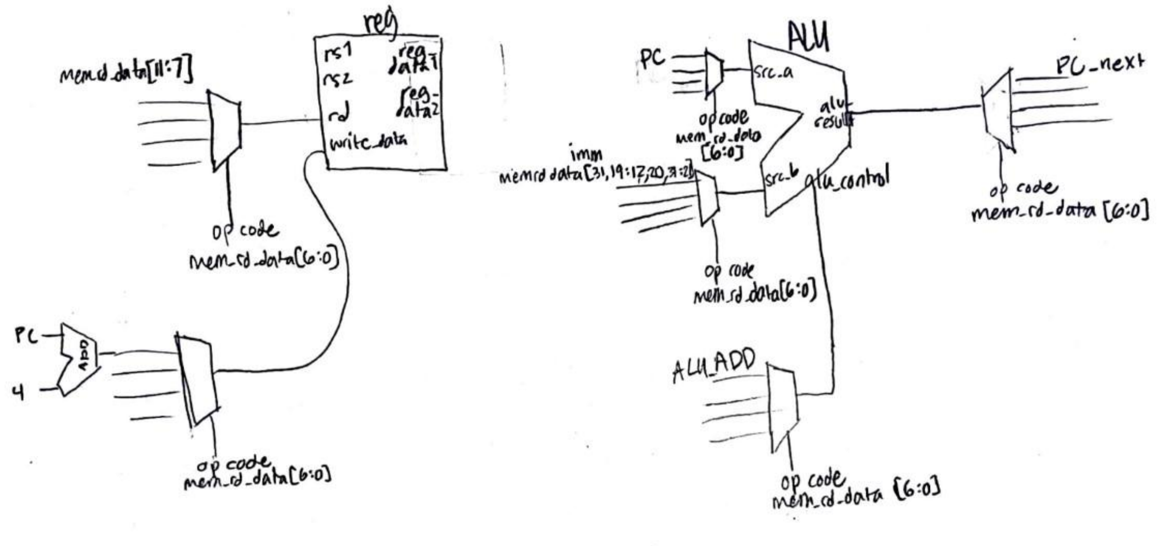
Datapaths



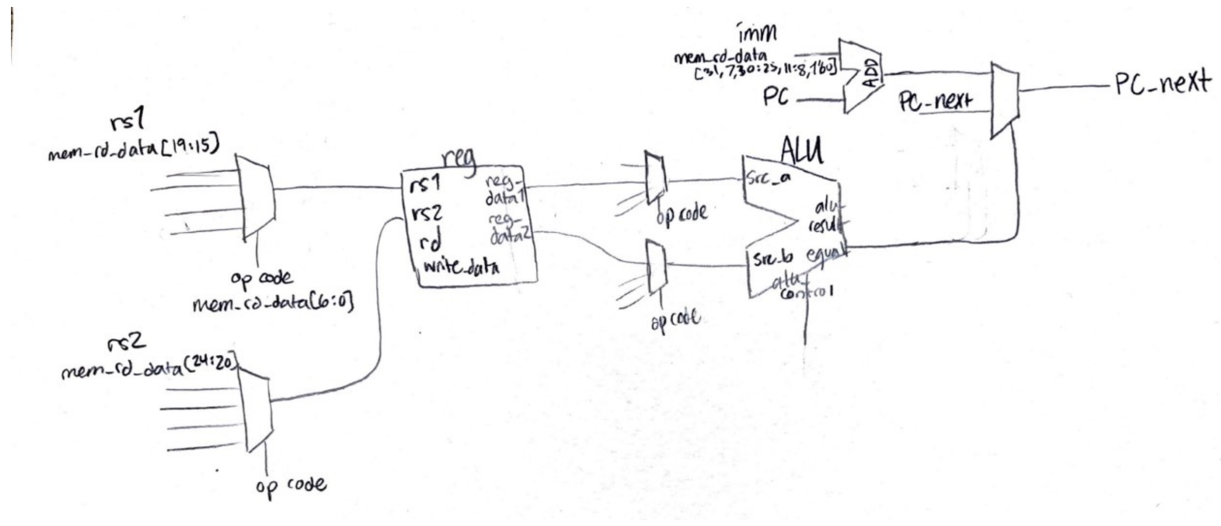
R- and I- type datapaths



Store word (SW) datapath



Jump (JAL) datapath



Branch (BEQ) datapath

Tests we ran

- asm/ritypes.s, to test r-types
- asm/itypes.s, to test i-types
- asm/lstypes.s, to test loads and stores
- asm/jb_types.s, to test jumps and branches

We verified that instructions were working by inputting these assembly files to the CPU in simulation, then using GTKWave to see that the CPU behaved as expected. We looked to see that the program counter and the main FSM progressed through states as expected, and that the values we expected were deposited into registers/memory. These tests are incomplete: we changed them over time to test different values and situations, rather than compiling one complete testbench. However, you can still see a demonstration of some instructions of each type by running make test_rv32i_ri_types, make test_rv32i_itypes, make test_rv32i_ls_types, and make test_rv32i_jb_types.

Instructions we implemented

R-types

- [x] ~~add~~
- [ ] sub
- [x] ~~xor~~
- [x] ~~or~~
- [x] ~~and~~
- [x] ~~sll~~
- [x] ~~srl~~
- [ ] sra
- [x] ~~slt~~
- [x] ~~sltu~~

I-types

- [x] ~~addi~~

- [x] ~~xori~~
- [x] ~~ori~~
- [x] ~~andi~~
- [x] ~~slli~~
- [x] ~~srli~~
- [ ] srai
- [x] ~~slti~~
- [x] ~~sltiu~~

Memory-Types (Loads/Stores)

- [x] ~~lw~~
- [x] ~~sw~~
- [ ] lb*
- [ ] lh*
- [ ] lbu*
- [ ] lhu*
- [ ] sb*
- [ ] sh*

B-types (Branches)

- [x] ~~beq~~
- [ ] bne
- [ ] blt*
- [ ] bge*
- [ ] bltu*
- [ ] bgeu*

J-types (Jumps)

- [x] ~~jal~~
- [x] ~~jalr (technically an i-type)~~

U-types (Upper immediates)

- [ ] lui*
- [ ] auipc*

* means optional.
We did not implement sub and sra due to needing to parse the additional funct7 code. We also did not implement bne, although we did implement beq which is similar. We did not feel the need to implement more branch instructions because it would not add significantly to our learning.

Code

GitHub: https://github.com/liloheinrich/RISC-V-CPU