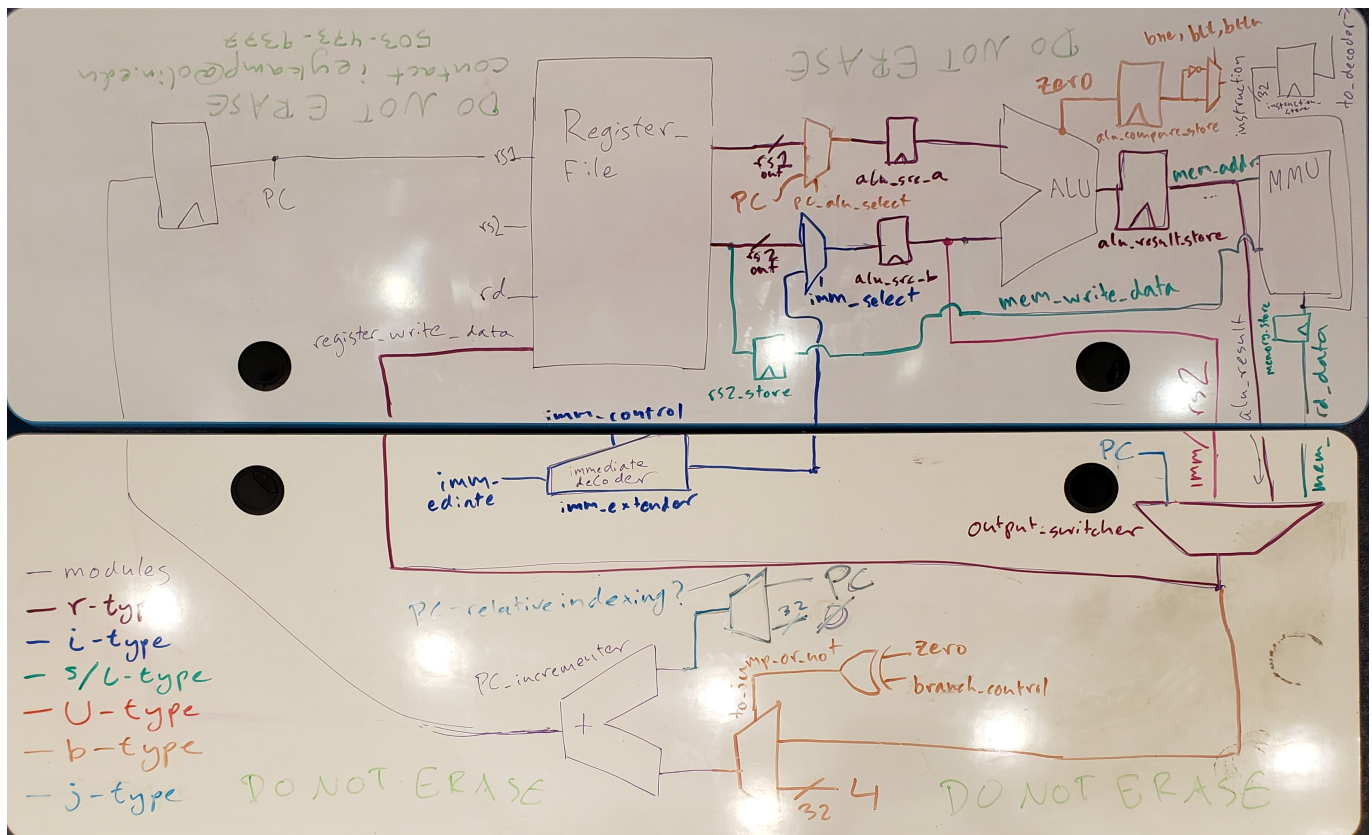# Lab 3 Submission

Ian Eykamp

Figure 1: Multicycle CPU schematic.

Base Modules (thin purple)

The main components of a multicycle cpu are the register file, ALU, and MMU. I chose to also instantiate a PC incrementer, which is just another adder. There is one register that holds the current Program Counter (PC), and another register that holds the current instruction, such that the cpu controller always refer to certain bits in the instruction. The other registers and muxes will be described in their specific sections.

Every instruction starts by incrementing the PC, resetting the control signals to of all the registers and muxes, then reading the instruction at address PC from memory and storing it in the instruction register. From there, the op-code determines the branch of the finite state machine to interpret the instruction correctly.

R-Type Instructions (dark purple)

An R-type instruction reads the two addresses `rs1` and `rs2` from the register file and stores them in `alu_src_a` and `alu_src_b`, respectively. On the next cycle, these values are read into the ALU and stored as `alu_result`. Then this value is written back to the register at the address `rd` specified in the instruction.

I-Type Instructions (dark blue)

For I-type instructions, the 12--or 20-bit immediate value is parsed by the decoder and fed into the immediate extender, which is a module I created. The `imm_control` signal determines which bit swizzling scheme to use depending on the instruction type, and the `imm_select` signal controls whether the immediate value or `rs2` should be read into `alu_src_b`.

U-Type Instructions (red)

For the U-type instruction `lui`, the appropriately extended immediate value is stored in `alu_src_b` and then immediately written into the register file at address `rd`.

S/L-Type Instructions (greenish)

For S/L-type instructions, additional connections are made to the MMU. For the store word instruction, the value from `rs2` is stored in its own register so that the immediate value can be added to `rs1` and used to set the memory address. The ALU output is connected to the `mem_addr` port, and the `mem_rd_data` value is stored in a register before being written back to the register file during the load word instruction.

B-Type Instructions (brown)

For B-type instructions, the `zero` output of the ALU is used to compare the register file values `rs1` and `rs2`. For the equality comparison, the ALU is set to subtraction; the values are equal if the result is zero. For the magnitude comparisons, the ALU is set to `SLT` or `SLTU`, and `rs1` is greaten than or equal to `rs2` if the result is zero. A simple not gate returns the opposite result for the `bne`, `blt`, and `bltu` instructions.

The output of the zero result controls whether the ALU result or simply 4 is added to the Program Counter before the next instruction.

J-Type Instructions (light blue/black)

For the `jal` instruction, the current PC is written to the register file. This can occur while the ALU result is still being calculated. The ALU adds an immediate value to the current PC and feeds that result into the PC incrementer

For the `jalr` instruction, the ALU adds an immediate value to the PC location stored in register `rs1`. Then the result is written to the PC.

For J-type instructions, the PC incrementer is configured to write the new value directly to the PC. This is accomplished with the `PC_relative_indexing` mux, which adds the increment to a constant `32'b0` instead of adding it to the current PC.

Implemented Instructions

R-types

- ☑ add
- ☑ sub
- ☑ xor
- ☑ or
- ☑ and
- ☑ sll

- ☑ srl
- ☑ sra
- ☑ slt
- ☑ sltu

I-types

- ☑ addi
- ☑ xori
- ☑ ori
- ☑ andi
- ☑ slli
- ☑ srli
- ☑ srai
- ☑ slti
- ☑ sltiu

Memory-Types (Loads/Stores)

- ☑ lw
- ☑ sw
- ☐ *lb*
- ☐ *lh*
- ☐ *lbu*
- ☐ *lhu*
- ☐ *sb*
- ☐ *sh*

B-types (Branches)

- ☑ beq
- ☑ bne
- ☑ *blt*
- ☑ *bge*
- ☑ *bltu*
- ☑ *bgeu*

J-types (Jumps)

- ☑ jal
- ☑ jalr (technically an i-type)

U-types (Upper immediates)

- ☑ *lui*
- ☐ *auipc*