Phase 3: RISC-V ALU, Decoder, Register File

EEL 4768 Computer Architecture: Spring 2026

Project Introduction

In this project phase, you will be building upon the previous phases of the project to build a single cycle RISC-V 32RI processor in Verilog. A skeleton for each module will be provided, along with a top module used to interconnect each module, which you ***must use*** or the autograder may not be able to process your files.

Your processor should ***only use instructions found in the provided RV32I reference card posted in webcourses***. Using any instructions outside of the base RV32I instruction set will result in a zero for the specific problem.

Submission Requirements

You are permitted to use generative artificial intelligence (e.g. ChatGPT); if you choose to do so, indicate where you use AI in your submission, and why you used it. Please ensure your usage of LLMs adheres to the course policies.

Submit the Verilog files as individual, unzipped files directly to Gradescope. In addition to the Verilog files, please include a word document or pdf detailing the contributions of you and your team members, along with another document indicating your use of AI.

- Verilog files
    - ALU.v
    - DECODER.v
    - REGISTER.v
    - Additional Verilog files
- group_contributions.pdf
- ai_report.pdf

In Gradescope, after submitting you can select your group for that submission. One submission is needed, but if you fail to select your group members in that submission, they may receive a zero for this phase.

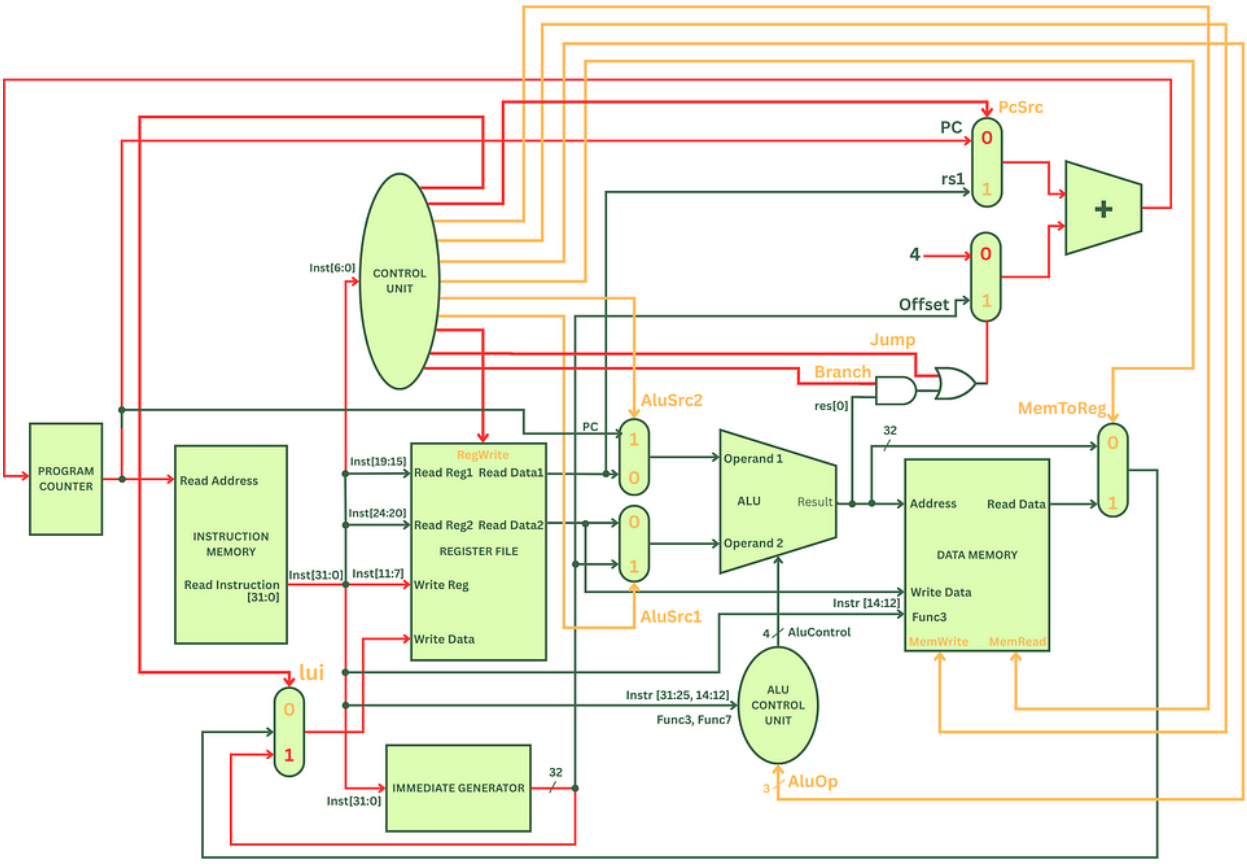
Environment setup

You may use any simulator as long as it's capable of simulating Verilog. If you want to use Vivado similar to how you used it in digital systems, you are more than welcome to.

Another simulator that is recommended to be used if Verilator, which compiles the simulation of the Verilog files with a C++ wrapper. For assistance with installation, please come to my office hours.

Icarus is another lightweight option.

Single Cycle Datapath



Here is an example of the single cycle datapath for a RISC-V processor. Please use this as a reference to help visualize the datapath and signals. There may be discrepancies between the image above and the skeleton files, so make sure not to directly copy the details of this figure and only use it as a reference.

Below will detail the submodules needed to implement the single cycle datapath.

ALU

We will add two instructions to the ALU, which are shown below.

BLTU	4'b1011
BGEU	4'b1111

The ALU skeleton provided will add these two instructions. Additionally, the input iAluOp has been changed to iAluCtrl. You do not need to use the blank skeleton and can simply add the changes to your existing code; it only exists for clarity.

Decoder and Register File

The decoder and register file will be reused from the previous phase. The skeletons will be provided again for clarity.

Control Unit

The control unit computes all the control signals within the processor. Below is a table that lists all of these and their uses.

Lui	Selects if loading from the upper immediate or from the output of the alu.
PC Src	Selects the source of the PC in the branch/jump unit, either directly from the current PC or from a register.
Memory Read	Enables reading from memory.
Memory Write	Enables writing to memory.
Alu Operation.	Passes the operation of the opcode to the ALU.
Memory to register.	Selects if either the memory value or alu output will be stored to a register.
Alu Src 1	Selects the input of ALU operand 1, either coming from the PC or RS1.
Alu Src 2	Selects the input of ALU operand 2, either coming from RS2 or an immediate value.
Register Write	Enables writing to the register file.
Branch selector.	Enables branching by comparing the output of the ALU and control unit branch.
Jump selector.	Enables jump by the control unit.

Branch and Jump Unit

The branch and jump unit are used to adjust the PC to go to different instruction address rather than just incrementing. The branch unit will always have an output, even if there is no branching or jumping, incrementing the PC to the next instruction. The branch and jump unit then can select to use an offset, which will be used in place of incrementing. Finally, the branch and jump unit can start with a different source address, selecting between the PC and an address from a register.

The output of the branch and jump unit is always the next instruction address to be processed.

ALU Control Unit

The ALU control unit processes the Alu operation, funct3 and funt7 inputs, and outputs the appropriate function for the ALU. The table below lists the output of the ALU Control Unit.

ADD	4'b0000
SUB	4'b1000
SLL	4'b0001
SRL	4'b1001
SRA	4'b1101
SLT	4'b0010
SLTU	4'b0011
XOR	4'b0100
OR	4'b0110
AND	4'b0111
BEQ	4'b1000
BNE	4'b1100
BLT	4'b1010
BGE	4'b1110
BLTU	4'b1011
BGEU	4'b1111

Data Memory Unit

The data memory unit is used to interact with off-chip data memory. This unit can read/write to off-chip memory to store and load values. This unit should not interact with instruction memory, which is separate from data memory.

Instruction Memory Unit

The instruction memory unit is used to interact with off-chip instruction memory. This unit can read from instruction memory with the instruction address (or the PC), and outputs the 32-bit instruction.

Additional modules

Additionally, multiple 2 to 1 multiplexer will be used around the datapath and must be implemented and called where needed. This module should parameterize the width of the two inputs and one output and use a single bit selection signal to select between the two inputs.

Top module

The top module will be used to tie all the modules together. There should be no logic in this top module besides wiring together modules.

Please see the skeletons for more details of the input/output of each module.