# Pipelined MIPS: Project 1

## Overview

In the first and second projects we will implement a subset of the MIPS32 architecture in Logisim:

<http://www.cburch.com/logisim/index.html>

The TA will give an introduction to the use of the tool in the class meeting.

By the end of these two assignments we will implement a 32-bit pipelined version of the MIPS architecture. We will ignore more advanced features such as the MIPS coprocessor instructions and traps/exceptions for these assignments.

In project 1 you will implement a functioning outline of the pipelined processor for a small set of instructions, including: decoding all the instructions you will encounter in projects 1 and 2, implementing most of the MIPS pipeline, correct implementation of arithmetic and logic operations, and simple hazard avoidance for these instructions.

In project 2 you will implement all a more complete set of instructions, and add more complete hazard logic.

## MIPS Pipeline

You will implement a five-stage MIPS pipeline, which is the most common organization for MIPS and is similar to what is described in the book and in class:

1. Fetch
2. Decode
3. Execute
4. Memory
5. Writeback

The MIPS standard, volume I, section 2.6, describe several possible MIPS pipelines, none of which exactly match the five-stage pipeline that we will implement.

**Memory stage.** For project 1, the memory stage of the pipeline will just be a passthrough doing nothing. In project 2 you will implement load and store memory operations and populate the memory stage part of the pipeline.

**Delay slot.** In this assignment you will not implement the branch/jump delay slot or memory load delay slot. That will be added in project 2.

## What to Implement

Implement a five-stage pipelined MIPS processor in Logisim. Your design should contain a program counter, a read-only program memory (ROM), a register file, *our* ALU, and any other components needed, along with the instruction decode and control circuits needed to connect them all together. The pipeline should: fetch instructions to execute from the program ROM and increment the program counter by 4; decode each instruction; select arguments from the register file; compute results; do nothing in the memory stage; and store results back in the register file.

Your pipeline must correctly execute all of the instructions in Table A:

|  |
| --- |
| **Table A** |
| Immediate arithmetic | ADDIU, ANDI, ORI, XORI, SLTI, SLTIU |
| Register arithmetic | ADDU, SUBU, AND, OR, XOR, NOR, SLT, SLTU |
| Move | MOVN, MOVZ |
| Shifts (constant and variable) | SLL, SRL, SRA, SLLV, SRLV, SRAV |
| Immediate load | LUI |

Note that LUI doesn't access memory and so, despite its name, is more similar to an arithmetic or logic instruction than a memory load instruction.

Furthermore, your instruction decoding and control logic must correctly decode all the instructions in table B. You will implement the functionality of these instructions in project 2, so decoding them properly now will save you from having to completely re-implement your decoding and control logic later.

|  |
| --- |
| **Table B** |
| Jumps (with one delay slot) | J, JR, JAL, JALR |
| Branches (with one delay slot) | BEQ, BNE, BLEZ, BGTZ, BLTZ, BGEZ |
| Memory Load/Store (little endian) | LW, LB, LBU, SW, SB |

Our testing programs for project 1 will include a mixture of instructions from both Table A and Table B. When processing one of the instructions of Table B, your implementation is free to do anything as long as: (1) no value in the register file changes and (2) the execution of instructions from Table A are not interfered with in any way. Think carefully about your decode logic to make sure this second condition works out.

**Deviation from the MIPS standard:** You can ignore any MIPS instruction or feature not mentioned in this document, such as traps, exceptions, and system calls. You can also assume that your processor will never encounter anything but legal instructions from the lists above.

Refer to the MIPS manual, Volume 2, linked on the course web site for a full specification of what each of these operations does. Except where noted in here, the MIPS manual is the authoritative specification for this project: information you find elsewhere (e.g. Wikipedia or the book) doesn't count if it contradicts the MIPS manual.

### Processor Components

Build your MIPS processor as a single Logisim circuit file. We provide a Logisim JAR library which you can load using Logisim's *Project > Load Library > JAR Library* command. This is the only Logisim library you should use. **Do not** use Logisim's *Project > Load Library > Logisim Library* command to import circuits from other Logisim files.

The ***EEL4713.jar*** Logisim library contains several components you will need, such as a register file, an ALU, an incrementer, and a program memory. *You must use these components where applicable, but you may only use one of each.*

**Register file.** Don't attempt to create your own register file out of regular Logisim Registers. This one makes debugging and simulation much easier.

**ALU.** *You must use the ALU from this library, rather than your ALU from homework 1.* Your design may only contain one ALU.

**MIPS Program ROM.** We will implement a Harvard Architecture design: The Program ROM component will store a read-only copy of the program instructions, and (in project 2) you will use a separate Logisim RAM to store data. The Program ROM component can load and display MIPS assembly language, which helps with debugging and saves you from having to write test programs in machine language.

**Incrementer.** You must use one (and only one) of these to increment the PC. Note that the incrementer only computes +1, whereas the PC increments by 4.

**Logisim components.** You can additionally use any of the components that come with Logisim, such as a Register for the PC, multiplexers, and so on.

## Testing

Write a test program in MIPS assembly that fully tests all of the features you have implemented. This is a critical step, and you should spend a fair amount of time developing a comprehensive program that exercises all of the different instructions and features of your processor. The program should be well commented, indicating what it is doing and what results should be expected when running the program, so that the course TA is convinced of the correctness of your processor.

## Documentation

The design document should include a block diagram showing all the major components in the processor (ALU, Register File, PC, pipeline registers, etc.), and the datapath connections between them. You need not completely draw wires for control logic signals, but should indicate which components take control inputs, and give names to all control signals. Overall, the diagram should be very similar to (and not much more complicated than) the diagrams used in lecture, but with labels for control signals and any relevant data signals. You should provide an explanation for any parts of your processor or any subcomponents that are unusual or potentially confusing.

Also include a description of your control and instruction decoding logic. For each control logic signal (or group of related control logic signals) you should provide (a) a brief description of what the signal does, e.g. what the values of the control signal mean; and (b) a truth table showing what value the signal takes for each possible opcode.

Update your design document as you work on your circuit. Documentation provides you another avenue of communicating your intended design and implementation in the case your circuit has mistakes. For a well-designed, clearly labeled and nicely laid out, completely correct solution, a screenshot of the Logisim circuit itself can serve as a diagram, but, otherwise, use the documentation to clarify and explain any parts of your circuit we might find confusing.

## What to Submit

* A single Logisim project file containing your processor and all needed subcomponents.
* A text file containing your well-commented MIPS assembly test program.
* A updated design document of your processor.

## Help and Hints

**Help.** Ask the instructor, TAs and consultants for help. We also expect to see most students in office hours during the course of the project. Extra hours will be scheduled as needed.

**Bugs.** If you suspect a bug in Logisim or the **EEL4713 library**, ask the course TA for help. There is a known bug having to do with bus splitters when the simulation is running. It is a good idea to turn the simulator off when editing the wire ordering on a bus splitter. This does not cause any data loss, but you might have to restart Logisim.

**Really, it shows.** Do a pencil-and-paper design before implementing in Logisim.

**What to do first.** Don't wait to get started. However, because we are still covering pipelining and pipeline hazards in lecture, you probably will not want to start right in on the final design. The following order of operations might be sensible:

1. Read the MIPS manual page for every instruction (some have unusual quirks).
2. Start writing test programs, including code both with and without hazards.
3. (After pipelining is covered in lecture) Design on paper, build in Logisim, and begin to test a pipelined CPU that doesn't handle pipeline hazards. Write the documentation as you go.
4. (After pipeline hazards are covered in lecture) Modify the datapath and control logic to handle pipeline hazards.