EECE 3324 Computer Architecture and Organization Final Project

5-stage Pipeline Architecture Implementation

Due on Dec.6th, at 11:59pm

1. Overview

For the EECE3324 project, you will implement part of the standard 5-stage MIPS architecture in Verilog. You are given a memory Verilog file which contains a program and its data, you should write a processor Verilog file which contains many other modules for the 5-stage pipeline architecture. The processor module interacts with the memory module. You should write your own testbench file to simulate the processor and memory. Finally, you should calculate the CPI of the provided program from the Verilog simulator.

I recommend use Modelsim as the HDL simulator. A separate instruction file on Modelsim installation and usage has been posted on BB. You can also use others, like ISE simulator, vcs, etc., if you are familiar with them. However, you have to let the TA and me know if you are using other simulators.

2. Collaboration

You will be expected to work in teams with no more than two students. The grading criteria are identical regardless.

You will be expected to maintain standards of academic integrity. Your group should do their own work. It is okay to discuss ideas with other groups but you must write your own code. The TA and I have collected various reference implementations and we will run thorough code checking. If we find any submission similar to the references, your group will lose all the points for the final project.

3. ISA Features

You are implementing the MIPS instruction set, which means that the basic features of your architecture should be the same: fixed-length instruction words, 32 registers, byte-addressed memory, `$r0` always contains the value 0, machine instruction formats, opcodes, and function codes, etc.

3.1 Loading the program

The given program in the memory module contains both instructions and data, in **little endian** format. You should assume that the first instruction you read is stored in the instruction memory at address 0x00001000 and all following instructions are filled sequentially after that. Data segment is in the range of 0x00000000 – 0x00000ffc.

3.2 Instructions to implement

As the focus of this project is on implementing and simulating the architecture, we pick a very small ISA subset for example. Your simulator must execute the following MIPS instructions:

- `add, addi`
- `beq, j`
- `lw, sw`

There is no OS support required, so we now add one special instruction "`HLT`" to end the program and simulation. The opcode for "`HLT`" is 0x3f and therefore the machine instruction for it is 0xfc000000.

3.3 Jumps and Branches

Note that the Jump instruction is different from the MIPS Jump. It contains a raw target address (WORD address) as an argument. The byte address will be obtained by just left shifting the WORD address by 2. You do not need to consider the top 4 bits of the current PC appended to the left-2 shifted immediate, like MIPS jump does. For example, if you see `j 0x00001010` in a machine instruction, then you should branch to address `0x00004040`.

Branch instructions will follow the MIPS format as accurately as possible. The destination argument to the branch instruction (`d`) should be read as an instruction offset from the next instruction to be executed. In other words, if the branch instruction is at address *PC*, then the first instruction executed after the branch is at memory address *PC+4+(4d).*

3.4 Simulating memory

Data operations will be limited to the range 0x00000000-0x00000ffc.
You may assume that the access time for the memories (instruction and data) is one cycle.
Your register file should be implemented such that writes occur before reads.

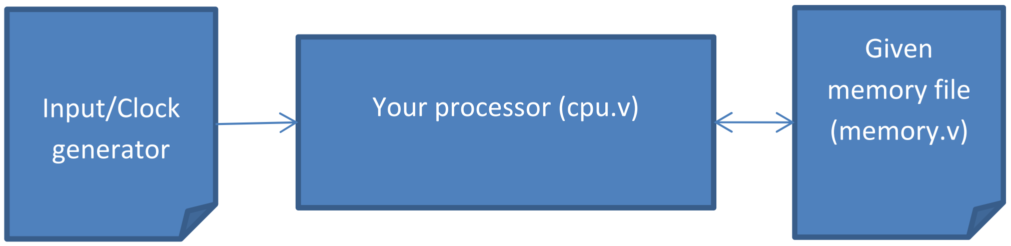
3.5 Individual testbench

I strongly encourage you to write testbench for each individual module before you put them in a large Verilog file for the processor. Simulate each module and make sure it works as you intend.

4. Testing

In addition to memory.v file, the sample program will be provided for you to understand the instructions contained in the program. The test program is a matrix computation program, including the assembly file, the object file, and the hexdump file (matrix.s, matrix.bin. matrix.hexdump).

The diagram below shows the structure of your testbench file. The memory.v file given to you have a set of 7 ports, the input ports are IRA (instruction memory read address), DA (data memory address), DMWE (data memory write enable), DMRE (data memory read enable), DWD (data memory write value) and the output ports are IRO (instruction memory read-out), DRO (data memory read out).



Use the results below to verify the functionality of your implementation.

Total # of cycles: 123
Total # of instructions: 100
CPI: 1.23
Register file contents:

R0: 0x00000000 R1: 0x00000000
R2: 0x00000020 R3: 0x00000012
R4: 0x00000000 R5: 0x00000000
R6: 0x00000000 R7: 0x00000000
R8: 0x00000003 R9: 0x00000006

R10: 0x00000009 R11: 0x0000000C
R12: 0x0000000F R13: 0x00000012
R14: 0x00000015 R15: 0x00000018
R16: 0x00000000 R17: 0x00000000
R18: 0x00000000 R19: 0x00000000
R20: 0x00000000 R21: 0x00000000
R22: 0x00000000 R23: 0x00000000
R24: 0x0000001B R25: 0x00000024
R26: 0x00000000 R27: 0x00000000
R28: 0x00000000 R29: 0x00000000
R30: 0x00000000 R31: 0x00000000

5. Project Timeline

The project implementation will be divided to three steps and here is the timeline and the tasks.

1. By Nov. 8th (Thursday) [Homework 6]: You are asked to implement several important modules in MIPS processor datapath, and write your own testbench for each module and simulate with Modelsim.
2. By Nov. 19th (Monday) [60 points]: Implement a single-cycle processor architecture shown in Figure 1.
3. By Dec. 6th (Wed.) [40 points]: Extend the single-cycle processor architecture to multi-cycle 5-stage pipeline architecture, shown in Figure 2. Add forwarding detection logic and hazard detection logic, and finish the project.

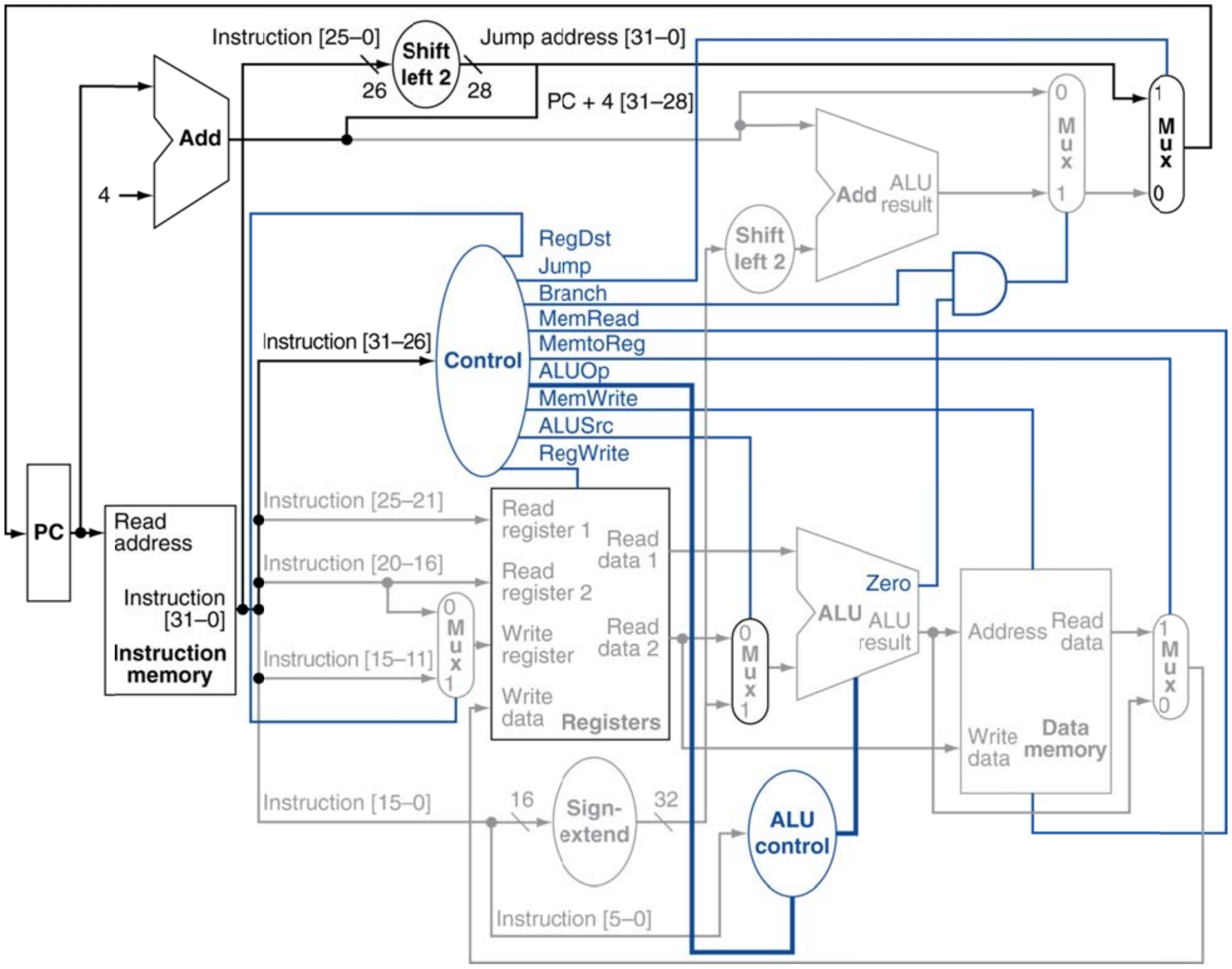


Figure 1 The processor diagram of a single-cycle implementation

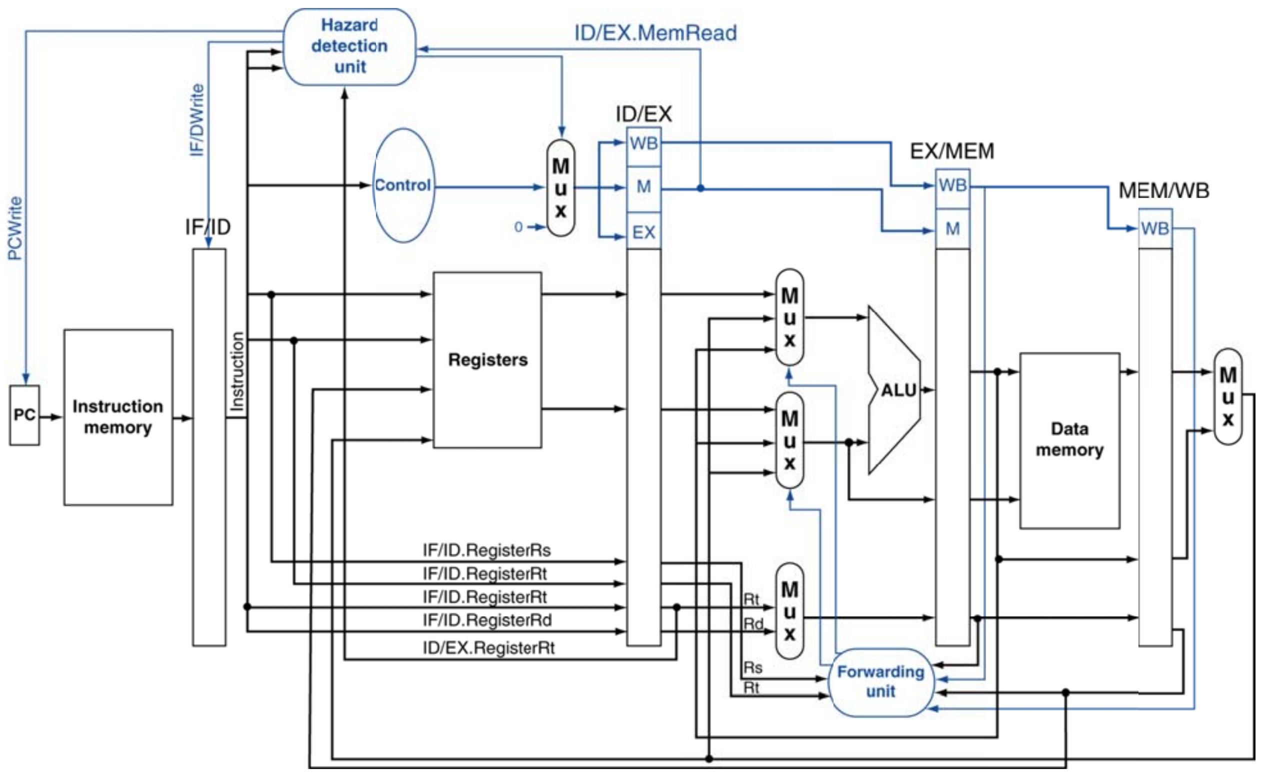


Figure 2. The processor diagram of 5-stage pipelined architecture

6. Project Submission and Grading Criteria

For each of the step, you should submit all the Verilog code with the intermediate results to the COE server course directory. On the final due date, you should turn in an archive onto the COE server course directory containing the following files:

- A complete development package, including all module Verilog files, your testbench file, and a README file if possible.
- A 2-page report describing the development process, like how you designed your code (including your choice of data structures and how you implemented the important control modules like forwarding unit and hazard detection unit), and the final simulation results (number of cycles, register contents, number of instructions, etc.)

The final grade will be based on:

- Successful simulation of your implementation by the instructor and the TA.
- The organization of your material: the code (including comments and documentation), and the report.

Additional consideration will be given to these factors:

- Clean, well-designed code.
- Any performance-enhancing optimizations.