EECE 3324	Computer Architecture and Organization	Final Project
	MIPS Architecture Implementation	Due on Dec. 5th (F) 11:59pm

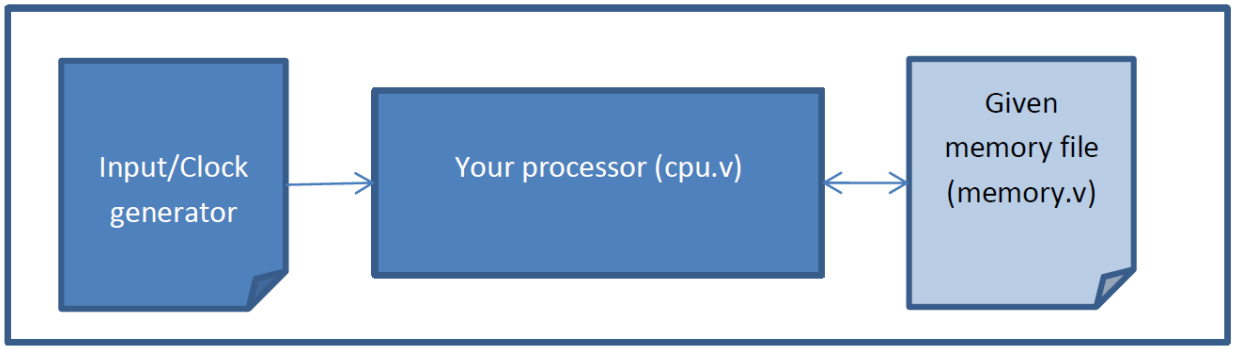
Basic project: single-cycle MIPS architecture implementation (worth 25% of the total course points)

1. Overview

For the EECE3324 project, you will implement the standard single-cycle MIPS architecture in Verilog. You are given a memory Verilog file which contains both text (program instructions) and data. You should write a processor Verilog file which contains all the modules for the processor datapath and controller. The processor module interacts with the memory module. Then 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.

What you will be given for memory modules are two files: memory.v and MinMax.hexdump. MinMax.hexdump contains all the binary instructions and data for a sample application, which will be read by memory.v into a memory array structure. In addition to these two files, you are also provided the application assembly file for understanding what the program does instruction by instruction (MinMax.asm).

The diagram below shows the structure of your testbench file. You testbench should include an input vector and clock generator, an instance of the processor (defined in cpu.v), and an instance of the memory module (defined in the given memory.v). The memory module has a set of 7 ports, the input ports are inst_addr (instruction memory read address), data_addr (data memory read/write address), mem_write (data memory write enable), mem_read (data memory read enable), data_in (data memory write value) and the output ports are instr (instruction memory read-out), data_out (data memory read-out).



I recommend using Modelsim as the HDL simulator. Instruction files on Modelsim have 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 can work in teams with no more than two students. The grading criteria are identical regardless of individual or team work.

You will be expected to maintain standards of academic integrity. Your group should do their own work.

It is okay, and actually encouraged, to discuss ideas with other groups for better understanding, 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, `$r0(=$zero)` 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 0x00003000 and all following instructions are filled sequentially after that. Data segment is in the range of 0x00000000 – 0x00002ffc.

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, addiu, slt`
- `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 Branches and Jumps

Branch instructions will follow exactly the MIPS format. The destination argument to the branch instruction (`d`) should be read as a word 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 taken is at memory address *PC+4+(4d)*.

Note that the Jump instruction is a little different from the MIPS Jump. It contains a raw target address (WORD address) as an argument. The byte address will be obtained by 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 the program execution should jump to address `0x00004040`.

4. Testing

Individual testbench: you have to write a 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. This is part of Homework 6.

For you final processor simulation, write a testbench file that contains an instance of processor, memory, and input vector and clock generator, and then simulate the design. Use the results below to verify the functionality of your implementation.

Total # of cycles: 149
Total # of instructions: 149
CPI: 1
Register file contents:

R0: 0x00000000 R1: 0x00000000
R2: 0x00000000 R3: 0x00000000
R4: 0x00000000 R5: 0x00000000
R6: 0x00000000 R7: 0x00000000
R8: 0x00000028 R9: 0x00000000
R10: 0xfffffff0 R11: 0x00000015
R12: 0x00000015 R13: 0x00000001
R14: 0x00000000 R15: 0x00000000
R16: 0x00000000 R17: 0x00000009
R18: 0x00000005 R19: 0x00000009
R20: 0x00000000 R21: 0x00000000
R22: 0x00000000 R23: 0x00000000
R24: 0x00000000 R25: 0x00000000
R26: 0x00000000 R27: 0x00000000
R28: 0x00000000 R29: 0x00000000
R30: 0x00000000 R31: 0x00000000

5. Project Timeline

The project implementation is based on homework 6. Here is the timeline:

1. By Nov. 12th (Wednesday) [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 Dec. 5th (Friday): Implement a single-cycle processor architecture shown in Figure 1.

6. Project Submission and Grading Criteria

By the project due date, you should turn in an archive zip file onto the COE server course directory containing the following files:

- A complete development package, including all Verilog module files, your top-level testbench file, and a brief README file to tell which testbench to simulate and run how long (no need to submit the individual testbenches for smaller modules).
- A 2-page report in pdf describing the development process, like how you designed your code including your choice of data structures and how you implemented the important control modules 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 for extra credits will be given to these factors:

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

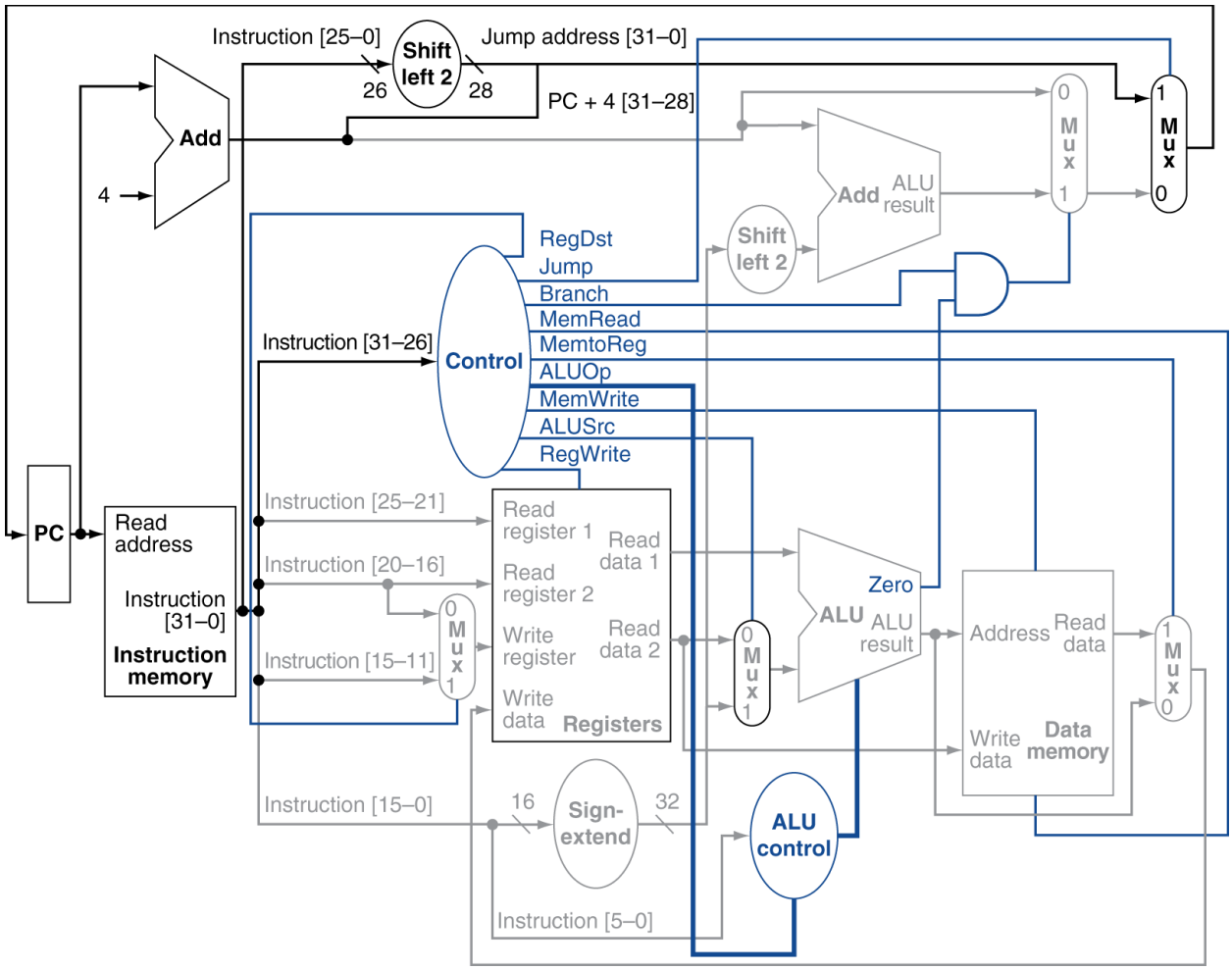


Figure 1. The processor diagram of a single-cycle implementation

Advanced project: five-cycle pipeline MIPS architecture implementation (worth 25% of the total course points, if you choose to continue to do the advanced project, the final exam is waived automatically)

In the advanced project, you will extend the single-cycle processor architecture to multi-cycle 5-stage pipeline architecture, shown in Figure 2. Add pipeline registers for both the datapath and control logic, forwarding detection logic, and hazard detection logic, put them all together, and simulate the 5-stage pipeline processor design. The simulation results will be different in performance from the single-cycle implementation, with the register file content not changing:

Total # of cycles: 183

Total # of instructions: 149

CPI: 1.23

Note that when implementing the pipeline processor, you may need to change some modules designed in the single-cycle processor. For example, now in one cycle, the register file can be both read and write (by different instructions), the writing has to occur earlier than the reading. Also you have to remove the

clock on register writing.

Submission: the advanced project is due on Dec. 7th (Sun) 11:59pm on COE server. You should submit another package (different from the basic one) which includes all the code files and a one-page report.

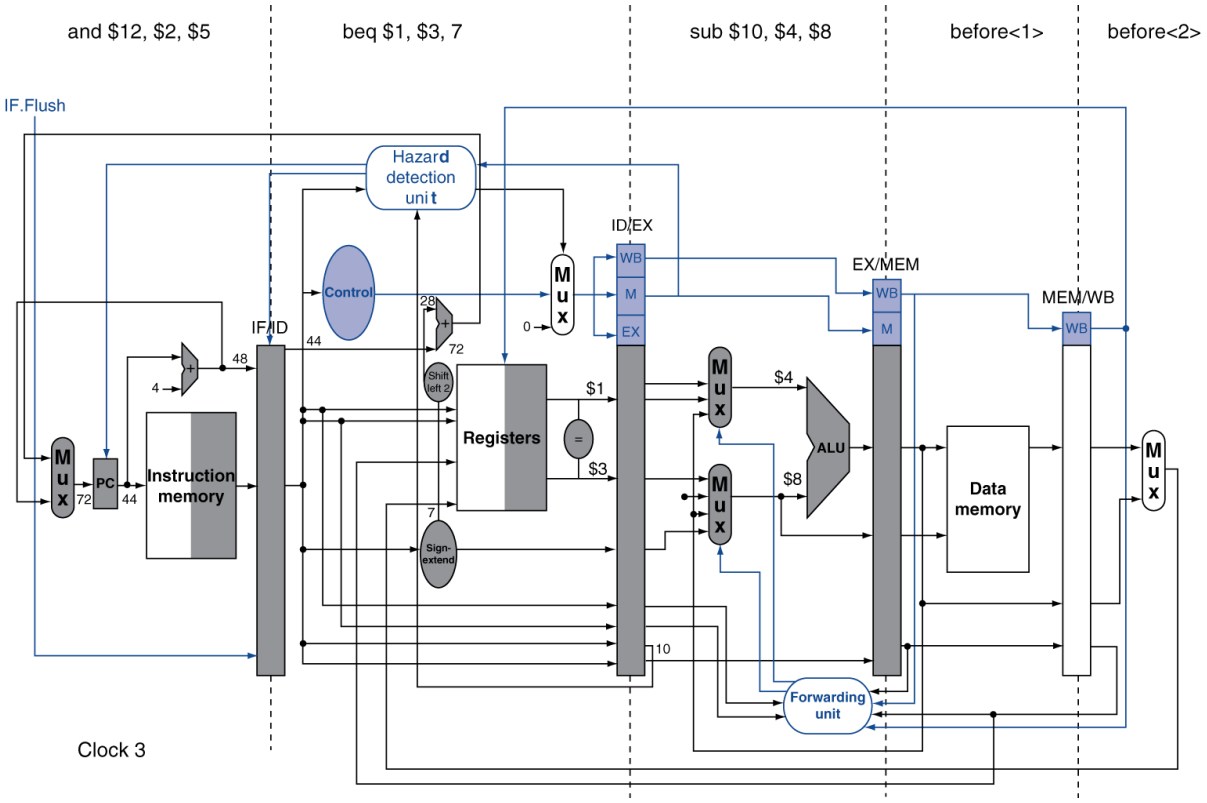


Figure 2. The processor diagram of 5-stage pipelined architecture