ECS-154B, Spring 2013, Lab 4

Due by 11:55 p.m. on June 2, 2014

Via Smartsite

Objectives

- Build and test a pipelined MIPS CPU that implements a subset of the MIPS instruction set
- Handle hazards through forwarding and stalling

Description

In this Lab you will use logisim to design and test a pipelined MIPS CPU that implements a subset of the MIPS instruction set along with data hazard resolution. Make sure to use the given circuit for this assignment as the Register File included implements internal forwarding while the previous versions do not.

Details

Your CPU must implement all instructions from Lab 3. Specifically you must implement: ADD, SUB, ADDI, LW, SW, AND, OR, NOR, ANDI, ORI, BEQ, SLT, J, JAL, JR, SLL, and SRL. Your CPU should not have any branch delay slots and should use a branch not taken strategy for branch prediction. You may implement your control signals any way you want. You can use combinational logic, microcode, or a combination of the two.

Hazards

As you have learned in lecture, pipelining a CPU introduces the possibilities of hazards. Your CPU must be able to handle **all possible** hazards. Your CPU must use forwarding where possible and resort to stalling only where necessary. Below is a subset of the possible hazards you may encounter (This means that while all possible hazards may not be listed here, your CPU must still be able to handle all possible hazards).

1. Read after Write Hazards. Your CPU must perform forwarding on both ALU inputs to prevent Read after Write Hazards. Your CPU must handle the hazards in the following code segments without stalling.

Add $4, $5, $6	Add $4, $5, $6	Add $4, $5, $6	Add $4, $5, $6
Add $7, $4, $4	Add $8, $9, $10	Add $8, $9, $10	Lw $8, 0($4)
	Add $7, $4, $4	Add $7, $4, $8	

2. Load-Use Hazards. Your CPU must handle load-use hazards through stalling and forwarding. You may only stall when necessary. If you stall when forwarding would work, you will lose points.

Lw $8, 0($4)	Lw $8, 0($4)
Add $10, $9, $8	Add $4, $5, $6
	Add $10, $9, $8

3. SW hazards. Your CPU must handle all Read after Write Hazards associated with SW using forwarding.

Add $4, $5, $6	Lw $4, 0($0)	Lw $4, 0($0)
Sw $4, 0($4)	Sw $4, 10($0)	Sw $5, 10($4)

4. Control Flow Hazards. Read after write hazards can also occur with the BEQ and JR instructions.

The following hazards must be solved with forwarding.

Add $4, $5, $6	Add $4, $5, $6	Lw $10, 0($0)	Lw $10, 0($0)
Add $8, $9, $10	Add $8, $9, $10	Add $4, $5, $6	Add $4, $5, $6
Beq $0, $4, BranchAddr	Jr $4	Add $8, $9, $10	Add $8, $9, $10
		Beq $0, $10, BranchAddr	Jr $10

The following hazards should be resolved with stalls.

Add $4, $5, $6	Add $4, $5, $6	Lw $10, 0($0)	Lw $10, 0($0)	Lw $10, 0($0)	Lw $10, 0($0)
Beq $0, $4, BranchAddr	Jr $4	Add $4, $5, $6	Add $4, $5, $6	Beq $0, $10, BranchAddr	Jr $10
		Beq $0, $10, BranchAddr	Jr $10		

Branch Prediction

In order to reduce the number of stall cycles in our CPU we will be using a branch not taken prediction strategy. This means that if a branch is taken we will need to provide hardware to squash the incorrectly predicted instructions. For example

Beq $0, $0, BranchAddr	
Add $1, $1, $1	This instruction must be squashed

The number of instructions that must be squashed is dependent on where in the pipeline you evaluate your branch condition. Jump instructions can be viewed as branches that are always taken and therefore are able to have their “branch” conditions evaluated in the Decode stage.

Grading

Implementation 90%:

I will grade your implementation will be tested and graded as follows.

File Name	Percent of Implementation Grade	What it Contains
NoFowarding.rb	30%	A basic test of your pipelined CPU. No forwarding or stalling is required. Contains no control flow instructions.
Forwarding.rb	40%	A basic test of your forwarding logic. No stalling is needed. Contains no control flow instructions (This means it also does not test forwarding to control flow instructions).
FinalTest.rb	30.00%	Anything and everything possible. Requires both forwarding, stalling, and squashing. Contains control flow instructions.

For each test file I will look at the contents of your registers and memory to see if your CPU is performing correctly. NoFowarding.rb and Forwarding.rb do not have any infinite loops to terminate themselves with so you will have to step through those programs manually.

Interactive Grading: 10%

Submission

Submit your circuit on smartisite along with a README file.

In the README include:

- You and your partners' names
- If your circuit works and if it does not what it does not work on
- Any difficulties you had
- Anything else you feel I should know

Hints

- The pipelined CPU diagram in the book should be used as a guide and not a goal. It does not show everything you need to do to implement all of the instructions. Also it is very possible to improve on their design.
- In my old edition of the book there was an error in the forwarding logic. I am not sure if they fixed it in the current edition.
- Build and test in parts. I would recommend building a basic pipelined CPU. After that works add in the forwarding logic and test again. Finally add in the logic to stall and squash instructions.
- If you want to design your own tests you can use the program mips2mif located in

/home/cs154b/bin on the CSIF computers to generate your own mif file. From there your can run the python program, mif2logisim.py, to create memory initialization files that will work in logisim.