CSE141L Lab 4: Single-Cycle MIPS Branches

Due Jul 25 (6:00pm)

The programs that you will generate for Lab 4 will be too large to fit in a 1K instruction ROM. Therefore, please modify your datapath to have an address width of 10 bits. For example, in my processor.v module, I instantiate the instruction ROM like this:

```
// Instruction ROM Instantiation
inst_rom #(
        .ADDR_WIDTH(10),
        .INIT_PROGRAM("../memh/lab3.inst_rom.memh")
) inst_rom_inst (
        .clock(clock),
        .reset(reset),
        .addr_in(pc_next),
        .data_out(instruction)
);
```

If you get errors about a memh file being too large, this is probably your problem. Let us know if you have any problems with this.

Overview

In Lab 4, you will finalize the design of your single-cycle MIPS processor. Specifically, you will be adding support for branch and jump instructions. In addition, you will add support for a sufficient subset of the MIPS ISA to enable you to run many common applications.

This lab (and the next two) will require much more work than the previous labs and will probably take longer than you expect, so start early.

General Notes

- All Verilog used in these labs (except for test benches) should be synthesizable.
- Please remember that you must conform to the class Verilog Coding Standards.
- Use the same group you used for Lab 3. You can split up, but you cannot merge.

MIPS ISA Subset

In Lab 3, you added a control unit that supported a few basic instructions: **LW, SW, ADD, ADDI, SUB, AND, OR, NOR, XOR**. However, we will need to add a few more instructions to run a typical application.

This is the subset of the MIPS ISA that you will need to implement by the end of this lab. You will need to modify many parts of your datapath, as well as expanding your control unit. Depending on your implementation, you may need to modify your ALU. You will need to create new modules (i.e., a 4-input MUX, a zero-extender, etc.). We will not be providing a schematic for you for this lab. Before testing your project with the example applications below, you will be required to implement the remaining instructions. Use page A-50 of your textbook to see how each instruction is encoded.

Subset of MIPS ISA

Branches & Jumps	Loads & Stores	Arithmetic & Logic
BEQ	LB	ADDU
BNE	LH	ADDIU
BLTZ	SB	SUBU
BGEZ	SH	ANDI
BLEZ	LBU	ORI
BGTZ	LHU	XORI
JUMP		LUI
JR		SLT
JAL		SLTU
JALR		

Branches & Jumps

There are many types of the branch instruction, but their basic function is to change the program counter (PC) based on the evaluation of some condition. This instruction enables the if-else, while loop, and for loop contructs in typical programming languages. The jump instruction is similar in that it is capable of changing the PC, but does not require evaluating any condition. The jump-and-link instructions are used for function calls.

Your design will not include a branch delay slot.

As a reference, you should use Figure 4.24 on page 271 in the textbook. The top third of the figure includes most of the logic that you will need to implement. **Be advised that the figure does not include the logic necessary to implement JR, JAL, and JALR.**

Store Byte and Store Half

The SB instruction stores the low-byte of a register to memory. The SH instruction is similar, except it stores the low two bytes of a register to memory. We have already included support in the async_memory module for the SB and SH instruction. The data_memory module has a port called size_in. You will have to modify your control unit to set this signal appropriately. It is a 2-bit input, but only accepts three possible values:

- 2'b00: A byte write. It will use the low byte of the write data input and store it in the correct location.
- 2'b01: A half write. It will use the lower two bytes of the write data input and store it in the correct location.
- 2'b11: A normal word write. This is what you set it to in Lab 2.

Load Byte and Load Half

the byte at address 0x03 and the word at that location is 0x80000000, I should return 0xFFFFFF80. The LH word is similar except it returns a shifted, sign-extended two-byte half instead of a single byte.

The LBU instruction is similar to the LB instruction, except the byte is zero-extended. Therefore, in the previous example, if I request the byte at address 0x03 and the word at that location is 0x80000000, I should return 0x00000080. The LHU instruction acts similarly except it returns a half-word (two bytes) instead of a byte.

Your design will not include a load delay slot.

Task 1: Update your schematic from Lab 3 or draw a new schematic that shows the additional logic you will need to add to your processor to implement the remaining instructions in the MIPS ISA Subset.

Task 2: Create a table that outlines the values of each of your control signals with respect to the current instruction. In the first column, include a row for each output signal your control unit will have. Label the top of each other column by the instruction(s). Fill in the table with the values of the control output signals for each intruction or group of instructions. If you recognize that a group of instructions share the same sontrol signal output values, you may consolidate them into a single column to save space. (You could reference the lab preview slides)

Testing The Whole Processor

In order to simulate the programs you will create for this lab, you should follow the same instructions you used for Lab 3. Namely, you should remember to set the INIT_PROGRAM parameter for the instruction ROM and data memory.

Unit Testing

In this lab, we provided three benchmark programs in lab4-files-2.zip: No branch hello world, Hello world, and fibonacci number. We also provide a simpler test bench lab4-test-3.zip. The simpler test bench contains an assembly file that is compatible with PCSpim and should print out "Hello world".

Task 3: Simulate your processor for each instruction. Observe the waveform that shows all of the internal signals in your processor module. Make sure each clock cycle with the operation is performed correctly(i.e., "ADDI", "BEQ", etc.).

When you are satisfied that your control unit is behaving correctly, its time to move on to testing larger programs.

~~"Hello World" with Branches~~

~~In Lab 3, you ran a simplified version of "Hello World" that avoided the use of any branch or jump instructions. We were able to do this by not testing the serial interface to see if it was ready to transmit data. Now that we have branches, however, we can test the serial interface by reading the serial output status register.~~

"Fibonacci number"

We wrote a program that prints out the first 10 Fibonacci numbers (0,1,1,2,3,5,8,13,21,34) using the iterative algorithm.

Task 4: Run the "helloworld" and "Fibonacci number" program contained in the zip file we provided. Simulate your processor until the whole program finishes. Measure the instruction count and number of cycles.

Wrap-up

Task 5: Once your processor is working correctly, run the whole synthesis and place and route flow and include the Fmax (and minimum period in nanoseconds), Total logic elements, and the Total registers statistics from your design.

Task 6: First, try to guess what the critical path in your design is. Say where your critical path starts, where it goes to, and where it ends. For example, you may say that the critical path is from the output of the PC register, through the adder, and back to the register. Now, try to use the tools to identify the critical path in your design. Instructions are on the Wiki. If your initial guess was wrong, try to explain why the path the tools report is correct. Think in terms of the amount of logic that is happening in a path, how many gates a wire might be connecting to, etc.

Interview Questions

- Show us the schematic of your design
- Show us that your processor can execute the three benchmarks provided in this lab in ModelSim
- Complete the following table.

  The Cycles Per Instruction (CPI) of a program is computed by calculating the total number of instructions that processor executed divided by the number of cycles it took to execute those instructions.

	No Branch Hello, world	~~Hello, world~~	Fib
Number of Instructions (excluding NOPs) in *.dis file			
Number of Cycles			
Cycles Per Instruction (CPI)			

- Tell us the critical path, Fmax in your design

Due: July 25