Digital Electronics & Computer Engineering (E85)

Lab 2: MIPS Controller

Introduction

In labs 8-11 you will be constructing a simple microprocessor running a subset of the MIPS instruction set. One of the key components of the microprocessor is the controller, which receives an instruction encoded in binary and decodes it to produce appropriate control signals that direct the movement of data through the processor. In this lab, you will construct the controller given a truth table specifying its operation. In Chapter 5 you will learn more about the controller and what the outputs actually do; for now, you can treat the device as a black box. In this lab you will also learn more about the schematic editor and simulator, including how to draw busses and create formula test vectors.

As always, don't forget to refer to the "What to Turn In" section at the end of this lab before you begin.

Background

The MIPS instruction set uses a six-bit *operation code* (opcode, or just op) to specify the action a processor should perform, such as add, subtract, or load information from memory. Certain "R-type" instructions share the same operation code and are distinguished by an additional four-bit *function code* (funct).

For each instruction, the controller must produce a suitable set of control signals that direct the rest of the processor. The truth table below lists the controller outputs as a function of inputs. It is identical to Figure C.4 except that it adds support for the `j` and `addi` instruction. Note that ALUOp is not actually an output seen by the external world, but is actually an internal signal used by the controller to compute the ALUControl signals (this is indicated by the italics).

Op[5:0]	**Instruction**	**Control Outputs**								
		RegDst	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	Jump	*ALUOp[1:0]*
000000	R-type	1	0	0	1	0	0	0	0	*10*
001000	addi	0	1	0	1	0	0	0	0	*00*
100011	lw	0	1	1	1	1	0	0	0	*00*
101011	sw	X	1	X	0	0	1	0	0	*00*
000100	beq	X	0	X	0	0	0	1	0	*01*
000010	j	X	X	X	0	0	0	0	1	*00*

The controller must also generate three ALUControl signals using the Funct input and ALUOp output:

ALUOp[1:0]	Funct[3:0]	ALUControl[2:0]
00	XXXX	010
01	XXXX	110
1X	0000	010
1X	0010	110
1X	0100	000
1X	0101	001
1X	1010	111

You may assume the outputs are undefined for inputs not given in these tables.

1) Schematics

Your first task is to design the logic to compute the control outputs and draw it in the Foundation Schematic Editor. Remember that you should optimize for least design time, not for fewest number of gates. Don't get bogged down trying too much to simplify your logic. Open the Xilinx Project Manager with a new project named "lab2_xx" (where xx are your initials).

Note that many of the signals are multiple bits wide. Rather than, for example, drawing six lines to represent the single signal Op, you will learn how to draw a **bus** in the Schematic Editor. To draw a bus, first click on the "Draw Buses" button.

Then click in the schematic where you would like the bus to end. Move the mouse to the place where you would like the bus terminal for the Op input, and then right-click. Select "Add Bus Terminal" from the popup menu, and you will see the "Add Bus Terminal/Label" window. Specify "Op" as the bus name, "5:0" as the bus range, and a terminal marker type of "INPUT," then click OK.

The separate bits of the Op signal need to be wired to logic gates in the controller. To do this, first draw six wires starting in space and ending on the Op bus. Next you will need to draw bus "taps" to specify which bit of the Op signal each wire is actually connected to. Click on the "Draw Bus Taps" button.

Your mouse cursor should have changed, and you are now in the mode to draw bus taps. This mode works as follows: first click on the bus that you want to tap into. Then notice that the status bar at the bottom of the window indicates a specific bit of the bus, starting at the most significant bit Op5. Place your mouse cursor over a wire that taps into the bus that corresponds to the bit indicated. Each time you click, the bus line will decrement. Keep clicking on wires that tap into the bus until you have specified which bit each of them connects to. At any point, you can cancel any operation in the schematic editor by pressing 'Esc'. (For more information on the "Draw Bus Taps" mode, search for "Bus Taps" in the Schematic Editor's online help system.)

You will also need to create busses for Funct[3:0] and ALUControl[2:0]. Remember to make ALUControl an output. Finally, create regular hierarchy connectors for the single-bit outputs: RegDst, ALUSrc, MemtoReg, RegWrite, MemWrite, Branch, and Jump. When you have created the inputs and outputs, use any logic gates in the library to compute the outputs according to the truth tables given. You will probably need a B-sized drawing sheet to fit all of your gates; you can select the size from File•Page Setup. If you need to use a single wire for two purposes, you will discover the wire cannot have two names. To get around this problem, you can add buffers (BUF) to the design.

You may wish to label internal signals to help probe during simulation. For example, even though ALUOp is neither an input nor output of the controller, you may wish to be able to probe it later. To give a name to a wire, double-click on the wire. Then type in the net name. For example, name the ALUOp wires ALUOp1 and ALUOp0.

2) Simulation

When you are done drawing your control logic, your next step is to simulate and debug the logic. Bring up the Logic Simulator window and add at least the two input busses and all of the outputs. For debugging purposes, you may wish to add some internal signals as well.

We will use formulas to specify the Op and Funct inputs. Choose Signal•Add Stimulators, then click on the Formula button in the Stimulator Selector dialog to create new formulas. Double-click on the F0 line and enter the following formula:

[00]50[08]10[23]10[2b]10[04]10[02]10

The numbers in brackets are hexadecimal values to apply to the bus and the numbers after each bracket are durations to apply the value, measured in nanoseconds. In binary, this formula assigns the value 00000000 for 50 ns, 00001000 for 10 ns, 00100011 for 10 ns, 00101011 for 10ns, 00000100 for 10 ns, and 00000010 for 10 ns. We will use it as the stimulator for the Op input, which is only 6 bits, so the top two bits will be ignored. You can check that the remaining 6 bits correspond to rows of the Op truth table.

Similarly, create a formula F1 that will be used for the Funct input:

[0]10[2]10[4]10[5]10[a]10[0]50

Now assign these formulas to the inputs using the Stimulator Selector. Click on Op in the Waveform viewer, then click on the F0 yellow square in the Stimulator Selector.

Similarly, assign the next yellow button, F1, to the Funct input. When you have done this correctly, you should see the red letters F0 and F1 next to the inputs in the Waveform Viewer.

By default, bus values are displayed in hexadecimal. Since our truth tables are specified in binary, let's change the busses to display in binary as well. Click on a bus name in the Waveform Viewer (such as Op), then choose Signal•Bus•Display Binary. Repeat the process for the Funct and ALUControl busses.

Set the Simulation duration to 100 ns by typing 100 into the box on the Logic Simulator toolbar. Now press the Simulation Step button to run the simulation for 100 ns and check that the outputs agree with the truth table. If they do not, track down and fix your logic errors in the schematic. You may find it useful that the values of nodes displayed in the Waveform Viewer under the vertical cursor are also listed on the schematic.

What to Turn In

Please provide a hard copy of each of the following items:

1. Please indicate how many hours you spent on this lab. This will not affect your grade, but will be helpful for calibrating the workload for next semester's labs.
2. A printout of your schematic showing your controller logic.
3. A printout of your simulation waveforms showing correct operation for 100 ns.