Lab 2: MIPS Controller

Introduction

In Labs 8 onwards 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 of *Computer Organization and Design* 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.

Background

The MIPS instruction set uses a six-bit *operation code* (OPCODE, or just OP (5:0)) 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 (3:0)).

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.2.4 in Appendic C on the CD for *Computer Organization and Design*, 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 an internal signal used by the controller to compute the ALUCONTROL signals. (This is indicated by the italics.)

		CONTROL OUTPUTS								
OP (5:0)	INSTRUCTION	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 four ALUCONTROL signals using the FUNCT input and ALUOP output:

ALUOP(1:0)	FUNCT(3:0)	ALUCONTROL(3:0)
00	XXXX	0010
01	XXXX	0110
1X	0000	0010
1X	0010	0110
1X	0100	0000
1X	0101	0001
1X	1010	0111

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

Part I

Generate ten output signals (from REGDST to JUMP plus ALUOP(1) and ALUOP(0)) as functions of the six input variables from the bus OP(5:0), according to the first truth table above. Although there are 2^6=64 combination terms, only the six listed in the truth table are actually used. No Karnaugh map simplification is needed here. For each function, you only need to "OR" those terms corresponding to function value 1 (e.g., OP(5:0)=000000 for REGDST, OP(5:0)=001000 OR 100011 OR 101011 for ALUSRC, etc.).

Part II

The inputs include the four variables from a 4-bit bus called FUNCT(3:0) and the two functions ALUOP(1) and ALUOP(0) generated in Part I. Based on these six variables, you are to generate three functions, ALUCONTROL(3:0), according to the second truth table above. The functions are put together as a 3-bit bus. Again, as only very few of the combinations are actually used, you don't need to use a Karnaugh map for simplification. Do the same as in Part I to generate the outputs.

Schematics

Your first task is to design the logic to compute the control outputs and draw it in the Xilinx ECS (schematic editor). Create a new project in Xilinx Project Navigator with a project name "lab2_*xx*" (where *xx* are your initials). To create your schematic, go to Projects→New Source, choose Schematic, name your schematic and click OK. Remember that you should optimize for least design time, not for fewest number of gates. Don't get bogged down trying too hard to simplify your logic.

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 represent the multi-bit signal. To do so, you need to first draw a wire, add an I/O marker to the input and name the port, for example, OP(5:0). Your bus should be changed to a thick wire. On the other side of the bus, place six bus taps:

If you need to rotate your bus taps, you can use "Ctrl-R" or the Rotate icon that can be found in the toolbar. Make the tap side (left side as shown) of each bus tap attach to the bus line. On the other side of each tap, extend the wire. These wires are related to the bits of the bus and will be connected to the logic gates. You have to name each of the wires after the bus tap, for example, OP(5), OP(4), OP(3), OP(2), OP(1) and OP(0), to represent the corresponding bit. To do so, go Add→Net Names or click the Add Net Names icon in the toolbar. In the same toolbar, select Name Branch and type in OP(5) in the next empty block. Move the cursor to the wire which represents OP(5) and click to place the name of the wire. Similarly name branches OP(4) to OP(0). The finished bus should look like Figure 1.

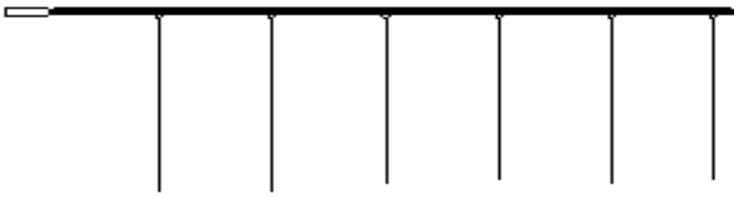

FIGURE 1 Bus with width 6.

At any point, you can cancel any operation in the schematic editor by pressing Ctrl-Z. 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(3:0). Remember to make ALUCONTROL an output port. Finally, create regular hierarchy connectors for the single-bit outputs: REGDST, ALUSRC, MEMTOREG, REGWRITE, MEMWRITE, BRANCH, and JUMP. *Note (weird but important):* this software may

use the name BRANCH for special purposes, so name yours as BRANCH0 instead. Otherwise, the schematic files will not be able to re-open after you exit the editor. When you have created the inputs and outputs, use 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. To select the size, double-click on any part of the schematic where there are no objects. Select the desired paper size from the Schematic Properties, click OK and Yes.

If you need to use a single wire for two purposes, you will discover the wire cannot have two different names. To get around this problem, you can add buffers (BUF) to the design. You may also wish to monitor the internal signals to help you debug during the simulation. To do so, you need to add additional output ports to the internal paths you are interested in. This is similar to using a probe pin in a circuit when you debug your hardware. For example, even though ALUOP is neither an input nor output of the controller, you may wish to be able to probe it later. After you are done with your schematic, check for errors. Choose Tools→Check Schematic. If there is an error, you can use Edit→Find to help you find the nets and debug your schematic before simulation. When you work with the schematic, remember to save the file regularly. Before exiting the editor, save the final version of the schematic and print it if you think it's the final correct version.

Simulation

When you are done drawing your control logic, your next step is to simulate and debug the logic. First you need to create a test bench waveform as you did in Lab 1 by clicking Project→New Source→Testbench waveform. Name your waveform such as "testlab2" and click Next, Next and Finish.

Your HDL Bencher opens and you can assign your input-output time constraints. Use 5ns for both Check Outputs and Assign Inputs in the combinatorial timing for the simulation. Select the Display as Hex. To enter the inputs, simply click on each blue cell and type in the value in hex. The following values will be entered as the OP (5:0) input:

```
[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 number after each bracket is the duration 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. You will use it as the stimulus for the OP input, which is only six bits, so the top two bits will be ignored. You can check that the remaining six bits correspond to rows of the OP truth table.

Similarly, enter the following inputs to be used for the FUNCT(3:0) input:

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

Save the waveform and close the testbench. The *testlab2.tbw* file is added in the Sources in Project window. As you did in Lab 1, choose ModelSim Simulator from the process window. Double-click Simulate Behavioral Verilog Model to launch ModelSim and click the Zoom Full icon in the taskbar to see the whole waveform of the simulation results. Both input and output bus values can be set to be displayed in Hex, Decimal or Binary. Choose Radix→Binary/Decimal/Hexadecimal. The finished waveform should look similar to Figure 2.

Check and see if 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. In the output, you will find very narrow spikes at the output of ALUSRC and REGWRITE around 60 ns. If you zoom in, you will see that instead of having continuous 1 for those time instants, the ALUSRC and REGWRITE are actually zero during a very narrow period of time. Why is this so?

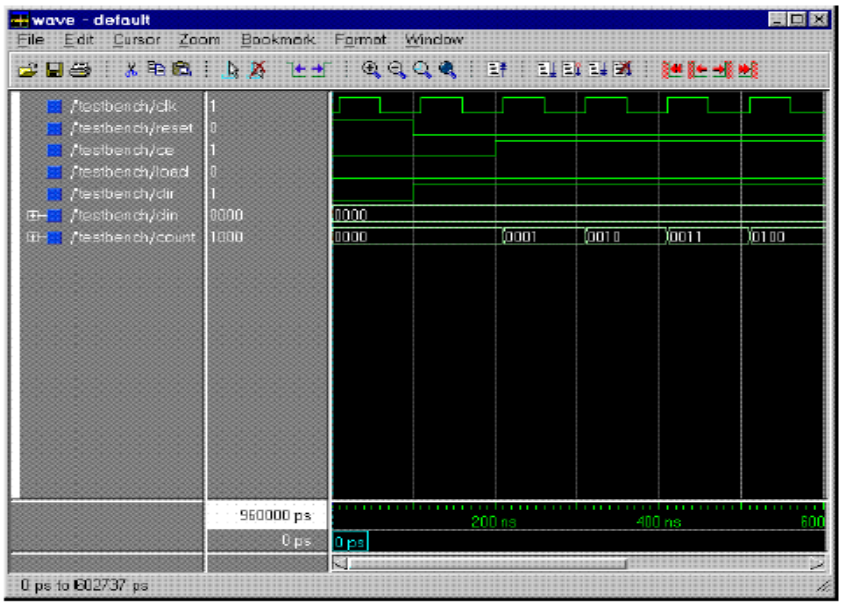

FIGURE 2 Finished waveform.