Digital Design and Computer Architecture

Lab 10: Multicycle Processor (Part 1)

Introduction

In this lab and the next, you will design and build your own multicycle MIPS processor. You will be much more on your own to complete these labs than you have been in the past, but you may reuse any of your hardware (SystemVerilog modules) from previous labs.

Your multicycle processor should match the design from the text, which is reprinted in Figure 1 for your convenience. It should handle the following instructions: `add`, `sub`, `and`, `or`, `slt`, `lw`, `sw`, `beq`, `addi`, and `j`. The multicycle processor is divided into three units: the `controller`, `datapath`, and `mem` (memory) units. Note that the `mem` unit contains the shared memory used to hold both data and instructions. Also note that the `controller` unit comprises both the `Main Decoder` that takes $OP_{5:0}$ as inputs and the `ALU Decoder` that takes as inputs $ALUOp_{1:0}$ and the $Funct_{5:0}$ code from the 6 least significant bits of the instruction. The `controller` unit also includes the gates needed to produce the write enable signal, `PCEn`, for the PC register.

In this lab you will design and test the controller.

© 2012 David Money Harris and Sarah L. Harris

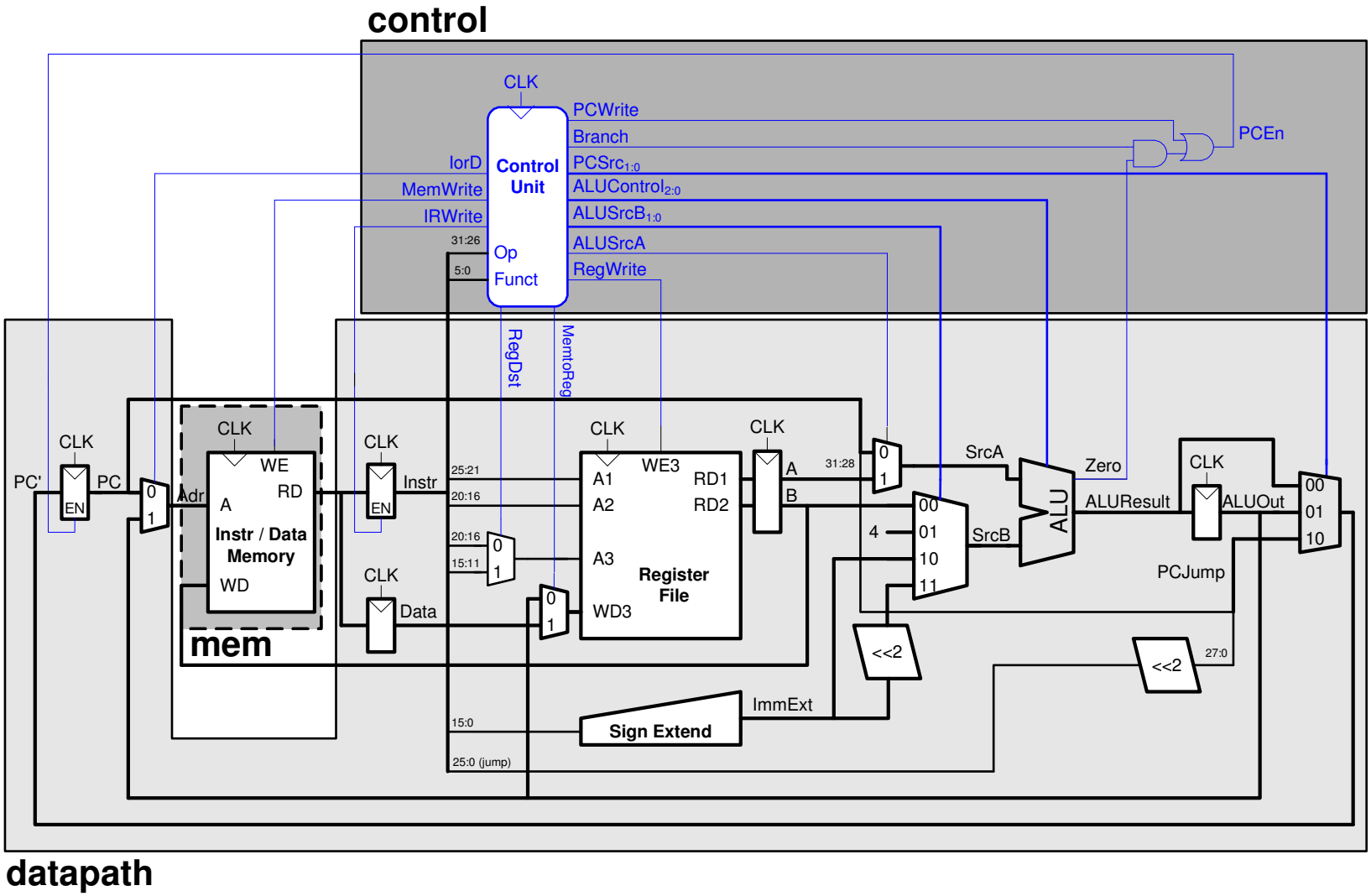


Figure 1. Multicycle Processor

© 2012 David Money Harris and Sarah L. Harris

Unit Overview

The three units have the following inputs and outputs. Although the signal names are in upper case here to match the diagram, remember to use lower case for all names in your SystemVerilog files.

CLK		Input
Reset		Input
Op	[5:0]	Input
Funct	[5:0]	Input
Zero		Input
IorD		Output
MemWrite		Output
IRWrite		Output
RegDst		Output
MemtoReg		Output
RegWrite		Output
ALUSrcA		Output
ALUSrcB	[1:0]	Output
ALUControl	[2:0]	Output
PCSrc	[1:0]	Output
PCEn		Output

Table 1. Controller

CLK		Input
Reset		Input
PCEn		Input
IorD		Input
IRWrite		Input
RegDst		Input
MemtoReg		Input
RegWrite		Input
ALUSrcA		Input
ALUSrcB	[1:0]	Input
ALUControl	[2:0]	Input
PCSrc	[1:0]	Input
ReadData	[31:0]	Input
Op	[5:0]	Output
Funct	[5:0]	Output
Zero		Output
Adr	[5:0]	Output
WriteData	[31:0]	Output

Table 2. Datapath

Note that *PCWrite* and *Branch* are internal signals (wires) within the controller.

CLK		Input
Reset		Input
MemWrite		Input
Adr	[5:0]	Input
WriteData	[31:0]	Input
ReadData	[31:0]	Output

Table 3. Memory (mem)

© 2012 David Money Harris and Sarah L. Harris

Generating Control Signals

Before you begin developing the hardware for your MIPS multicycle processor, you'll need to determine the correct control signals for each state in the multicycle processor's state transition diagram. This state transition diagram is shown in Figure 7.42 in the book. Complete the output table of the Main Decoder in Table 4 at the end of this handout. Give the FSM control word in hexadecimal for each state. The first two rows are filled in as examples. Be careful with this step. It takes much longer to debug an erroneous circuit than to design it correctly the first time.

Overall Design

Now you will begin the hardware implementation of your multicycle processor. First, copy `mipsmulti.sv` from the E85 Lab 10 directory on Charlie to your own directory and rename it `mipsmulti_xx.sv`.

The `mips` module instantiates both the `datapath` and control unit (called the `controller` module). The controller module in turn instantiates the main decoder module (`maindec`) and the ALU decoder module (`aludec`). You will design the controller in this lab. In the next lab, you will design the datapath. The memory is essentially identical to the data memory from Lab 9 and will be provided for you.

Control Unit Design

The control unit is the most complex part of the multicycle processor. It consists of two modules, the Main Decoder and the ALU Decoder. The Main Decoder, `maindec`, should take the Opcode input and produce the outputs described in Table 4. On reset, the control unit should start at State 0. The control unit should support the instructions from Figure 7.42 in the text. The state transition diagram is also given at the end of this handout.

Design your controller using an FSM for the Main Decoder and combinational logic for the ALU Decoder. Also include any additional logic needed to compute *PCEn* from the internal signals *PCWrite*, *Branch*, and *Zero*. The `controller`, `maindec`, and `aludec` headers are given showing the inputs and outputs for each module. A portion of the SystemVerilog code for the control unit has been given to you. Complete the SystemVerilog code to completely design the hardware of the controller and its submodules.

Create a `controllertest_xx` testbench for the `controller` module. Test each of the instructions that the processor should support (`add`, `sub`, `and`, `or`, `slt`, `lw`, `sw`, `beq`, `addi`, and `j`). Be sure to test both taken and nontaken branches. Remember that the `controller` inputs are: `clk`, `Reset`, `OP`, `Funct`, and `Zero`. Your test bench should apply the inputs. Visually inspect the states and outputs to verify that they match your expectations from Table 4. Also verify that *PCEn* performs correctly. If you find any errors, debug your circuit and correct the errors. Save a copy of your waveforms showing the inputs, state, and control outputs, and *PCEn* at each state.

© 2012 David Money Harris and Sarah L. Harris

What to Turn In

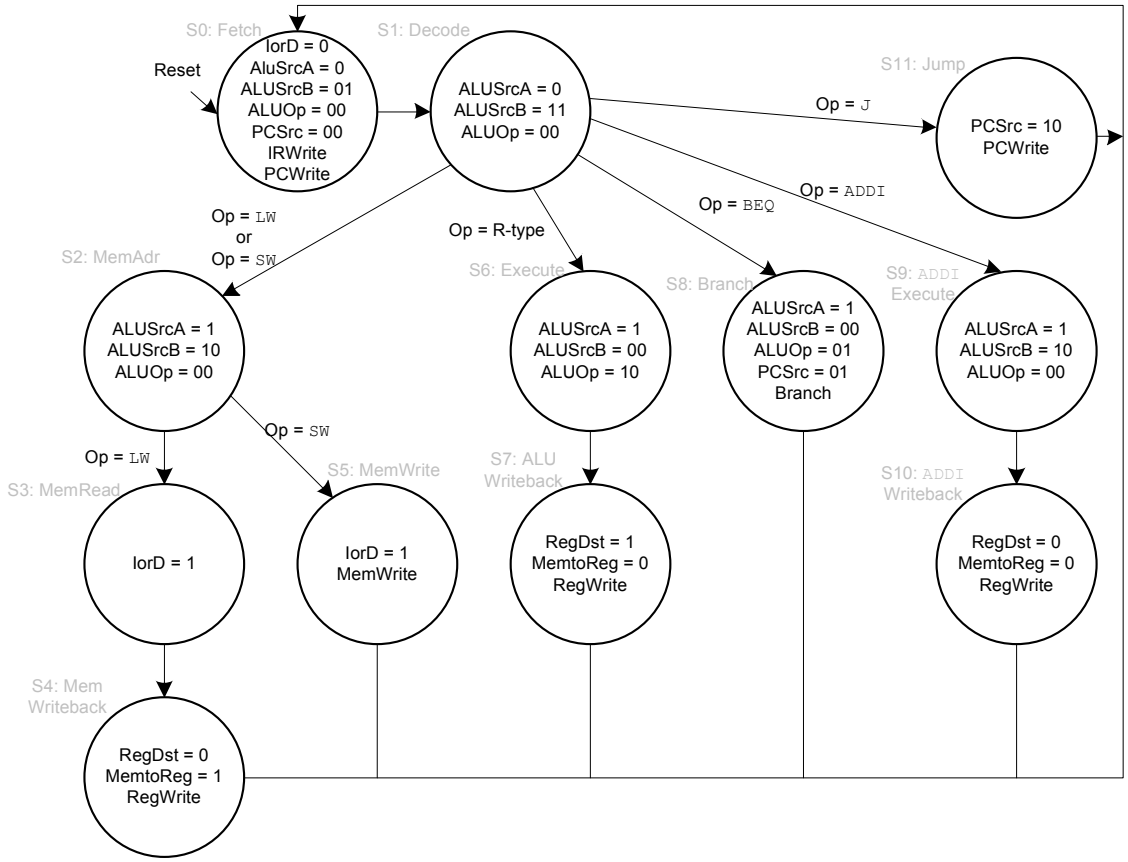
Submit the following elements **in the following order**. Clearly label each part by number. Poorly organized submissions will lose points.

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 completed Main Decoder output table (Table 4).
3. The SystemVerilog for your `controller`, `maindec`, and `aludec` modules.
4. Your `controllertest_xx` testbench.
5. Simulation waveforms of the controller module showing (in the given order): *CLK*, *Reset*, *OP*, *Funct*, *Zero*, the *state* (this is an internal registered signal), *ALUControl*, *PCEn*, and the entire control word (i.e. the 4-nibble word you entered in Table 4) demonstrating each instruction (including taken and non-taken branches). Display all signals in hexadecimal. Does it match your expectations?

© 2012 David Money Harris and Sarah L. Harris

State (Name)	PCWrite	MemWrite	IRWrite	RegWrite	ALUSrcA	Branch	IorD	MemtoReg	RegDst	ALUSrcB[1:0]	PCRsc[1:0]	ALUOp[1:0]	FSM Control Word
0 (Fetch)	1	0	1	0	0	0	0	0	0	01	00	00	0x5010
1 (Decode)	0	0	0	0	0	0	0	0	0	11	00	00	0x0030
2 (MemAdr)													
3 (MemRd)													
4 (MemWB)													
5 (MemWr)													
6 (RtypeEx)													
7 (RtypeWB)													
8 (BeqEx)													
9 (AddiEx)													
10 (AddiWB)													
11 (JEx)													

Table 4. **Main Decoder Control output**



© 2012 David Money Harris and Sarah L. Harris