EECE 444

LAB 6: MIPS Multi-Cycle Processor

Introduction

In this lab 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 (Verilog modules) from previous labs.

Your multicycle processor should match the design from the lecture, which is reprinted in Figure 1. 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 Part A of this lab you will design and test the controller. You will be finishing and testing the design of the whole microprocessor in part B.

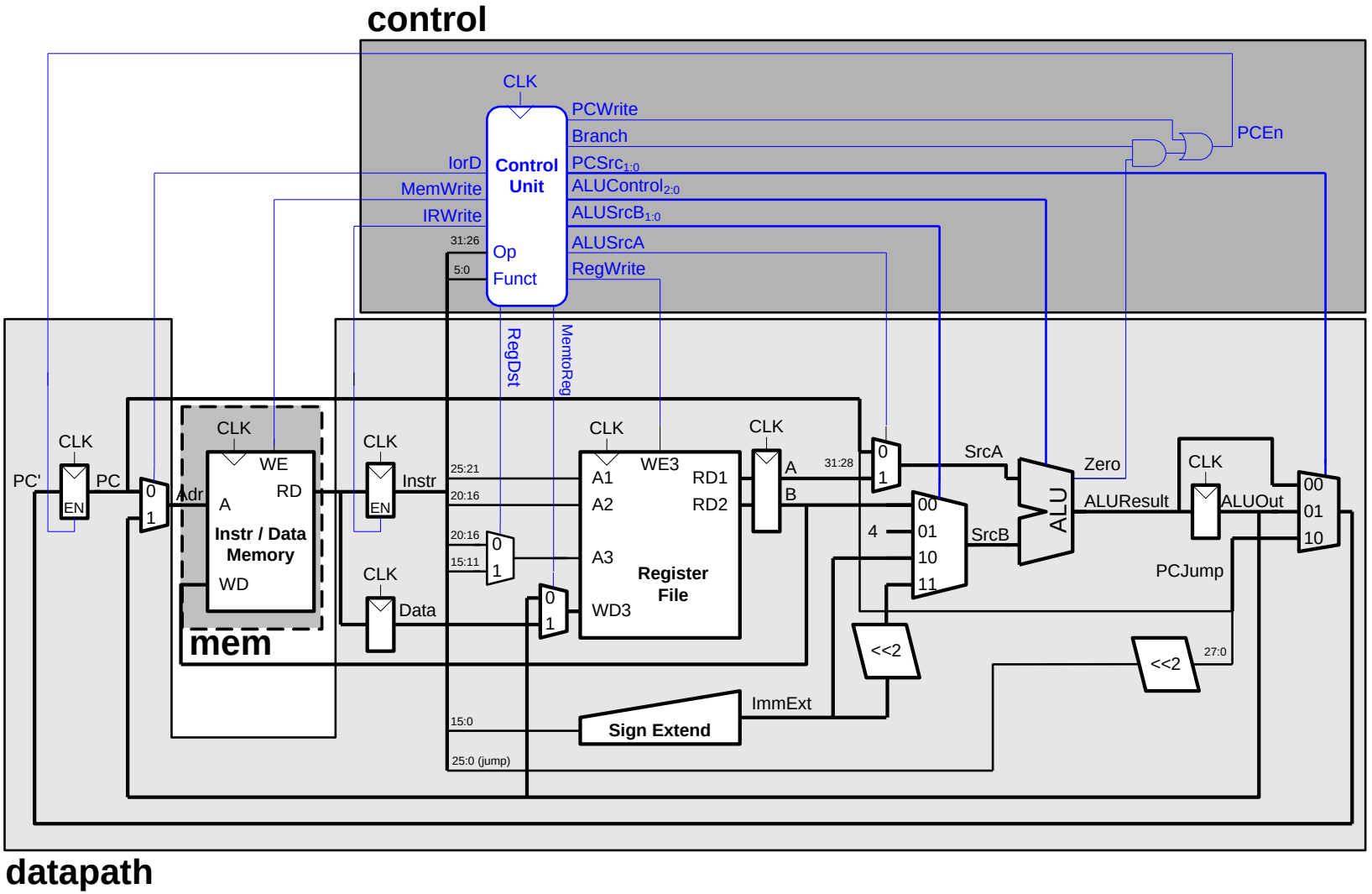


Figure 1. Multicycle Processor

Part A: Unit Overview

The three units have inputs and outputs as shown in Table1-3 below. Although the signal names are in upper case here to match the diagram, remember to use lower case for all names in your Verilog 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)

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 at the end of this lab. Complete the output table of the Main Decoder in Table 4 at the end of this lab 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.v` from the Lab Resources directory on Blackboard to your own directory and rename it `mipsmulti_xx.v`. (replace xx with your initials)

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 part of the lab, in prat B, you will design the datapath. The memory is essentially identical to the data memory from Lab 5 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 the state transition diagram given at the end of this handout.

Design your controller using a 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 Verilog code for the control unit has been given to you. Complete the Verilog 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.

Part B: Multicycle Processor Completion

In this part of the lab, you will complete your own multicycle MIPS processor. In part A, you created the control unit. In this part of the lab you will design the `datapath` and `mem` units and test your completed MIPS multicycle processor.

This lab uses the same testbench and generic parts as in Lab 5. Copy your `mipstest.v`, `mipsparts.v`, and `alu_xx.v` files from Lab 5 to your lab6B_xx directory. Also copy your `mipsmulti_xx.v` file containing your controller from Lab 6A.

Finally, copy the `topmulti.v` and `mipsmem.v` files containing the top-level module and the memory from the Blackboard. Look over the files and see how they work.

Test Program

Use the same test program and `memfile.dat` from the first part of Lab 5. Copy it to your lab6B_xx directory.

As in Lab 5, it is very helpful to first predict the results of a test program before running the program so that you know what to expect and can discover and track down discrepancies. Table 5: at the end of this lab, which is partially completed, lists the expected instruction trace while running the test program. Complete the remainder of the table. Do this before you run simulations so you have a set of expectations to check your results against; otherwise, it is easy to fool yourself into believing that erroneous simulations are correct.

Notice that the instruction (`instr`) is fetched during state 0 and therefore not updated until state 1 of each instruction.

When the ALUResult will not be used (e.g. in the Decode state of a nonbranch instruction, or the Writeback state of any instruction), you may indicate an 'x' for don't care rather than predicting the useless value that the processor will actually compute.

Datapath Design

Refer to Figure 1 for the hardware modules you need to set up your datapath. Design the datapath unit in Verilog.

Remember that you may reuse hardware from earlier labs (such as the ALU, multiplexers, registers, sign-extension hardware modules, register file, etc.) wherever possible.

All of your registers should take a *Reset* input to reset the initial value to a known state (0). The Instruction Register and PC also require enable inputs. Pay careful attention to bus connections; they are an easy place to make mistakes.

Simulate your processor using the testbench given (`mipstest.v`). The Reset signal is set high at first. Display, at a minimum, the *PC*, *Instr*, FSM *state* (from within your `controller` module), *SrcA* and *SrcB* (from within your datapath), *ALUResult*, and *Zero*, and the control word. You will likely want to add other signals to help debug.

Check that your results match the expectations from Table 5. If there are any mismatches, debug your design and fix the errors.

When you are finished – congratulations! You have built a microprocessor by yourself and have proven your mastery of microarchitecture, Verilog, FSMs, and logic design!

Debugging

Hopefully your lab will have at least one error so you will get to hone your debugging skills! Here are some hints:

- Be sure you thoroughly understand how the MIPS multicycle processor is supposed to work. This system is too complex to debug by trial and error. You should be able to predict what value every signal should be at every point in time while debugging.
- In general, trace problems by finding the first point in a simulation where a signal has an incorrect value. Don't worry about later problems because they could have been caused by the first error. Identify which circuit element is producing the bad output and add all its inputs to the simulation. Repeat until you have isolated the problem.

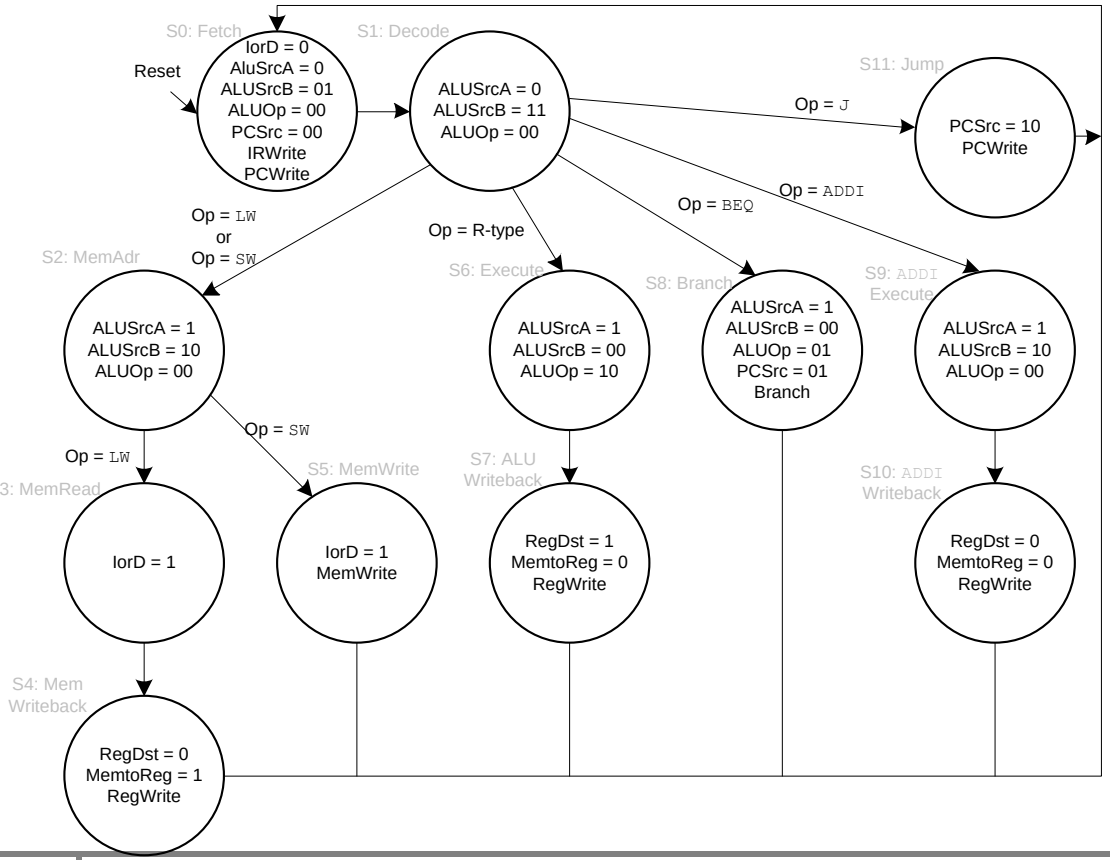
What to Turn In

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). (Part A)
3. The Verilog for your `controller`, `maindec`, and `aludec` modules. (Part A)
4. Your `controllertest_xx` testbench. (Part A)
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? (Part A)
6. A completed copy of Table 5 indicating the expected outcome of running the test program.
7. Verilog code of the datapath.
8. Simulation waveforms of the processor showing *CLK*, *Reset*, *state*, *PC*, *instr*, and *ALUResult* in this order while running the test program. As always, output the values in hex (or decimal if that is more readable) and make sure they are readable. Do the results match your expectations? Does the program indicate Simulation Succeeded?

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



Adopted and modified by Dr. Mustafa from David Money Harris and Sarah L. Harris

Cycle	Reset	PC	Instr	(FSM) state	SrcA	SrcB	ALUResult	Zero	Control Word
1	1	00	0	0	00	04	04	0	5010
2	0	04	addi 20020005	1	04	x	x	0	0030
3	0	04	addi 20020005	9	00	05	05	0	0420
4	0	04	addi 20020005	10	x	x	x	0	0800
5	0	04	addi 20020005	0	04	04	08	0	5010
6	0	08	addi 2003000c	1	08	x	x	0	0030
7	0	08	addi 2003000c	9	00	0c	0c	0	0420
8	0	08	addi 2003000c	10	x	x	x	0	0800
9	0								
10	0								
11	0								
12	0								
13	0								
14	0	10	or 00e22025	1	10	x	x	0	0030
15	0	10	or 00e22025	6	03	05	07	0	0402
16	0	10	or 00e22025	7	x	x	x	0	0840
17	0	10	or 00e22025	0	10	04	14	0	5010
18	0	14	and 00642824	1	14	x	x	0	0030
19	0	14	and 00642824	6	0c	07	04	0	0402
20	0	14	and 00642824	7	x	x	x	0	0840
21	0	14	and 00642824	0	14	04	18	0	5010
22	0	18	add 00a42820	1	18	x	x	0	0030
23	0	18	add 00a42820	6	04	07	0b	0	0402
24	0	18	add 00a42820	7	x	x	x	0	0840
25	0	18	add 00a42820	0	18	04	1c	0	5010
26	0								
27	0								
28	0								
29	0								
30	0								
31	0								
32	0								
33	0								
34	0								
35	0								
36	0								
37	0								
38	0								
39	0								
40	0	30	add 00853820	1	30	x	x	0	0030
41	0	30	add 00853820	6	01	0b	0c	0	0402
42	0	30	add 00853820	7	x	x	x	0	0840
43	0	30	add 00853820	0	30	04	34	0	5010
44	0	34	sub 00e23822	1	34	x	x	0	0030
45	0	34	sub 00e23822	6	0c	05	07	0	0402
46	0	34	sub 00e23822	7	x	x	x	0	0840
47	0	34	sub 00e23822	0	34	04	38	0	5010
48	0	38	sw ac670044	1	38	x	x	0	0030
49	0	38	sw ac670044	2	0c	44	50	0	0420
50	0	38	sw ac670044	5	x	x	x	0	2100
51	0	38	sw ac670044	0	38	04	3c	0	5010
52	0	3c	lw 8c020050	1	3c	x	x	0	0030
53	0	3c	lw 8c020050	2	00	50	50	0	0420
54	0	3c	lw 8c020050	3	x	x	x	0	0100
55	0	3c	lw 8c020050	4	x	x	x	0	0880
56	0	3c	lw 8c020050	0	3c	04	40	0	5010
57	0								
58	0								
59	0								
60	0								
61	0								
62	0								

Table 5: Part B Instructions