Digital Electronics & Computer Engineering (E85)

Lab 10: Multicycle Processor (Part 1)

Introduction

In this lab and the next, you will design and build your own multicycle MIPS processor!

Your processor should match the design from the text reprinted below. It is divided into four units: the cunit (control), eunit (execution), iunit (instructions), and munit (memory). In this lab we will refer to the sections as units rather than boxes to avoid conflicting with names from your previous labs. Note that the munit contains the shared memory used to hold both data and instructions. Also note that the cunit comprises both the decoder using Op[5:0] and the ALUControl logic taking ALUOp[1:0] and the Funct code from the low bits of the instruction.

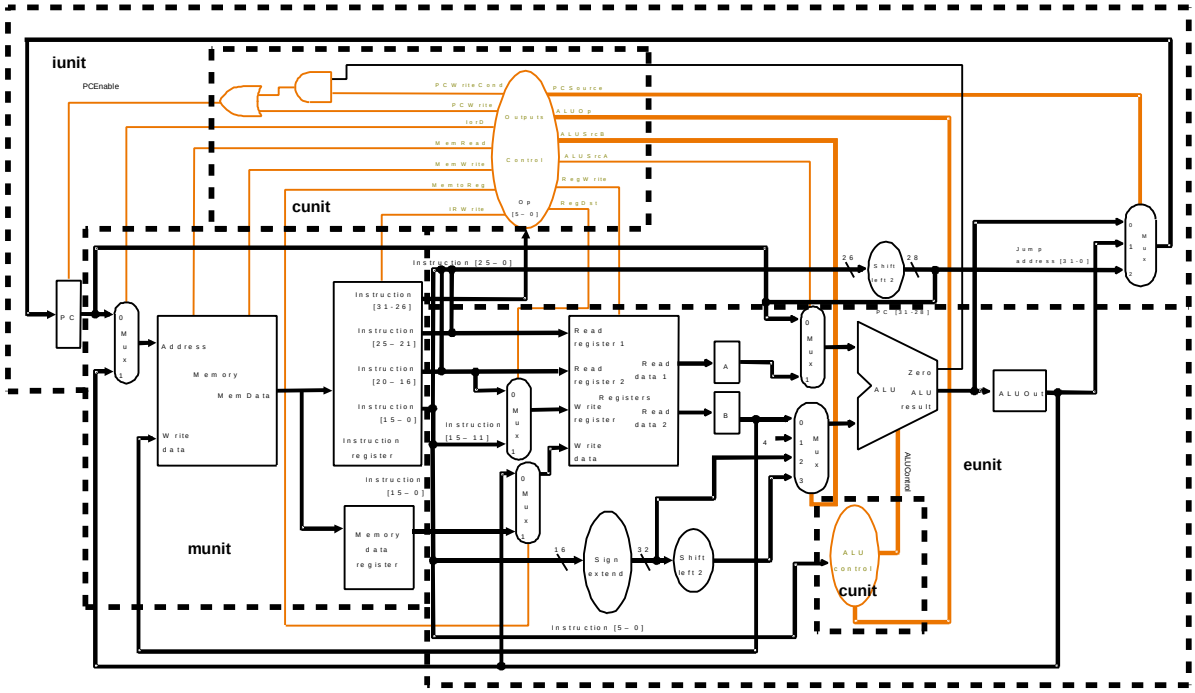

Figure 1: Multicycle Processor Datapath

Your task in this lab will be to design and test a microcoded state machine for the cunit and to work out a test program for the processor as a whole. In the next lab, you will design and test the remainder of the multicycle processor. You will be much more on your own to complete these labs than you have been in the past, but may reuse any of your hardware from previous labs.

Unit Overview

The units have the following inputs and outputs:

Clk		input
Reset		input
Op	[5:0]	input
Funct	[3:0]	input
Zero		input
PCSource	[1:0]	output
ALUControl	[2:0]	output
ALUSrcB	[1:0]	output
ALUSrcA		output
RegWrite		output
RegDst		output
IorD		output
MemRead		output
MemWrite		output
MemtoReg		output
IRWrite		output
PCEnable		output

Table 1: cunit interface signals

Clk		input
Reset		input
Instruction	[31:0]	input
ALUResult	[31:0]	input
ALUOut	[31:0]	input
PCEnable		input
PCSource	[1:0]	input
PC	[31:0]	output

Table 2: iunit interface signals

Clk		input
Reset		input
PC	[31:0]	input
ALUOut	[31:0]	input
WriteData	[31:0]	input
IorD		input
MemRead		input
MemWrite		input
IRWrite		input
Instruction	[31:0]	output
MemoryData	[31:0]	output

Table 3: munit interface signals

Clk		input
Reset		input
Instruction	[31:0]	input
MemoryData	[31:0]	input
PC	[31:0]	input
RegDst		input
MemtoReg		input
RegWrite		input
ALUSrcA		input
ALUSrcB	[1:0]	input
ALUControl	[2:0]	input
Zero		output
ALUResult	[31:0]	output
ALUOut	[31:0]	output
WriteData	[31:0]	output

Table 4: eunit interface signals

Test Program

Your first task will be to prepare a simple test program, shown below, to test all of the instructions. The program doesn't do anything terribly interesting, but will prove your processor is working correctly if it ends in an infinite loop at done with the proper value in $t4. Translate the program to machine language and predict the values of major signals after each cycle. You will use this program in the next lab as you debug your processor. Note that the constants 42, 2C, and 28 are all in hexadecimal, not decimal. Start the program at address 0 in memory. Recall that branches are counted relative to the next instruction, to the beq branches back by –7 instructions (FFF9).

```
          addi $t0, $0, 42
          j    later
earlier:  addi $t1, $0, 4
          sub  $t2, $t0, $t1
          or   $t3, $t2, $t0
          sw   $t3, 2C($0)
          lw   $t4, 28($t1)
done:     j    done
later:    beq  $0, $0, earlier
```

What result should be in $t4 when the program reaches done?

Before debugging your multicycle processor, you will need to have a good idea of what to expect out of the processor. Complete Table 6 at the end of this lab showing the values of the FSM state, PC, Instruction, SrcA, SrcB, ALUResult, and Zero at each cycle for your program. The FSM states are numbered in Figure 5.42 of your textbook; addi requires States 10 and 11 as shown in Table 5 later in this lab. When Table 5 indicates a don't care, you can leave a ? in Table 6 indicating the result is implementation-dependent. Notice that the instruction code is fetched during state 0 and therefore not updated until state 1 of each instruction.

cunit design

The cunit is the most complex part of the multicycle processor. It should take the inputs and produce the outputs described in Table 1. On Reset, the cunit should start at State 0. The cunit should support the instructions from Figure 5.42 plus `addi`, as we added in class. The state transition table is shown in Table 5. Recall that Seq of 00 means next state, 10 means Dispatch 1, 11 means Dispatch 2, and 01 means Fetch, as defined in Figure 5.46.

Design your controller using a microcode sequencer, as shown in Figure 2. The microcode storage is a ROM taking a 4-bit state as input and producing 18 bits of output.

Translate the state transition table in Table 5 into a series of 5-nibble hexadecimal values (treat the upper two bits as 0) that can be entered as the contents of the microcode ROM.

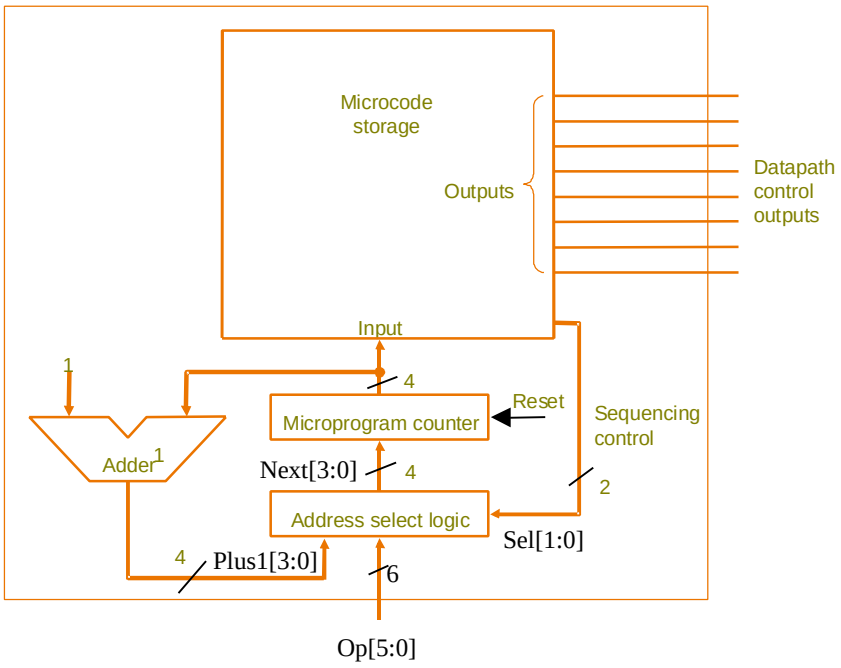

Figure 2: Microcode Sequencer

State	ALUOp[1:0]	ALUSrcA	ALUSrcB[1:0]	RegWrite	RegDst	MemtoReg	IorD	MemRead	MemWrite	IRWrite	PCSource[1:0]	PCWrite	PCWriteCond	Seq[1:0]	ROM Contents
0	00	0	01	0	x	x	0	1	0	1	00	1	0	00	02148
1	00	0	11	0	x	x	x	0	0	0	xx	0	0	10	
2	00	1	10	0	x	x	x	0	0	0	xx	0	0	11	
3	xx	x	xx	0	x	x	1	1	0	0	xx	0	0	00	
4	xx	x	xx	1	0	1	x	0	0	0	xx	0	0	01	
5	xx	x	xx	0	x	x	1	0	1	0	xx	0	0	01	
6	10	1	00	0	x	x	x	0	0	0	xx	0	0	00	
7	xx	x	xx	1	1	0	x	0	0	0	xx	0	0	01	
8	01	1	00	0	x	x	x	0	0	0	01	0	1	01	
9	xx	x	xx	0	x	x	x	0	0	0	10	1	0	01	
A	00	1	10	0	x	x	x	0	0	0	xx	0	0	00	
B	xx	x	xx	1	0	0	x	0	0	0	xx	0	0	01	

Table 5: Microcode ROM Contents

The address select logic of the Microcode Sequencer is shown in Figure 3. The dispatch tables are ROMs that determine next addresses based on the opcode. For example, the Dispatch2 ROM should go to state 3 on a lw (opcode=23_{16}) or state 5 on a opcode=$2B_{16}$).

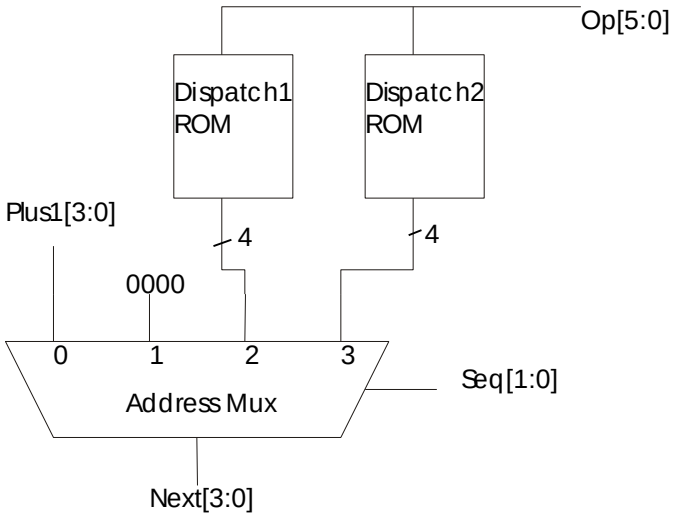

Figure 3: Address Select Logic

Create a schematic for your cunit. The schematic should include the microcode sequencer from Figure 2 along with logic to compute ALUControl[2:0] from ALUOp[1:0] (you can copy this from your Lab 2) and logic to compute PCEnable from PCWrite, PCWriteCond, and Zero.

You will want to use LogiBLOX extensively in your design. You can make a resettable flip-flop by specifying a LogiBLOX data register with an ASYNC CONTROL input and an ASYNC VALUE of 0; the ASYNC CONTROL is the reset input and the value is the value to which the register is set on reset. You can also use LogiBLOX to create four-bit constants like 0 and 1 as inputs to the adder and address mux. You can avoid a very messy combinational logic design problem by using ROMs for the microcode and dispatch tables. Place the contents of the ROM after the DATA line in the .mem file. When the ROM contains only a few nonzero entries, you can specify addresses for particular contents to avoid listing lots of 0's. For example, a Dispatch2 ROM .mem file is shown below, with the two lines you need to add indicated in bold. Observe that the ROM width is 4 bits (because it puts out a 4-bit output) and depth is 64 (6 bits of opcode address one of $2^6 = 64$ words).

```
;
; memfile dispatch2.mem for LogiBLOX symbol dispatc2
; Created on Tuesday, April 11, 2000 15:45:48
;
; Header Section
RADIX 10
DEPTH 64
WIDTH 4
DEFAULT 0
;
; Data Section
; Specifies data to be stored in different addresses
; e.g., DATA 0:A, 1:0
RADIX 16
DATA
23:3
2B:5
; end of LogiBLOX memfile
```

There are a number of ways to connect the microcode ROM output bus to the individual outputs. One of the simpler ways is to draw 18 bus taps and hook each bit of the bus to a buffer, which in turn can drive the output with appropriate name.

When you have completed your schematics, simulate the cunit. Develop stimulus for the Clk, Reset, Op, Funct, and Zero inputs based on your Table 6. For example, the first part of each waveform is shown below; the rest of Op and Funct is yours to determine from the appropriate bits of your Table 6. You may find it helpful to refer to your Lab 8 when generating the stimulus. If you use a LogiBLOX ROM for your dispatch logic, you may get a peculiar complaint about a bus conflict on your dispatch ROMs. This is caused by a bug in the simulator that doesn't properly understand the LogiBLOX ROM. You can eliminate the error by modifying the "Multiplexer Style" field of the LogiBLOX ROM to be "Normal Gates" rather than "Maximum Speed." Print out your waveforms showing all of the control outputs at each state. Make sure the outputs match your expectations (you haven't written these expectations, so you'll have to work them out as you go by looking at the current state and Table 5). If you find any errors, debug your circuit.

```
CLK:        B0
RESET:      [1]10[0]1000
OP:         [08]57[02]30[04]30 …
FUNCT:      [2]57[8]30[9]30 …
ZERO:       [0]97[1]10[0]1000
```

What to Turn In

Please turn in 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. Your test program, written in machine language.
3. Your expected results from the test program:
   - the value of $t4 upon reaching done
   - A completed Table 6 showing the FSM state, PC, Instruction, SrcA, SrcB, ALUResult, and Zero, of your program at each cycle until reaching done.
4. List the contents (in hexadecimal) of the microcode ROM from Table 5 and dispatch ROMs.
5. A printout of your schematics from the cunit.
6. Simulation waveforms of the cunit showing all the inputs and outputs while running the test program. Be sure the results match your expectations.

Cycle	Reset	PC	Instruction	FSM State	SrcA	SrcB	ALUResult	Zero
0	1	00	0	0	00	04	04	0
1	1	00	0	0	00	04	04	0
2	0	04	addi 20080042	1	04	108	10C	0
3	0	04	addi 20080042	A	00	42	42	0
4	0	04	addi 20080042	B	??	??	??	?
5	0	04	addi 20080042	0	04	04	08	0
6	0	08	j 08000008	1	08	20	28	0
7	0	08	j 08000008	9	??	??	??	?
8	0	20	j 08000008	0	20	04		0
9	0	24	beq 1000fff9	1	24	FFFF FFE4	08	0
10	0	24	beq 1000fff9	8	00	00	00	1
11	0	08	beq 1000fff9		08	04	0C	0
12	0	0C	addi 20090004					0
13	0	0C	addi 20090004	A	00	04	04	0
14	0	0C	addi 20090004	B	??	??	??	?
15	0							
16	0							
17	0							
18	0							
19	0	10	sub 01095022	0	10	04	14	0
20	0							
21	0							
22	0							
23	0							
24	0	18	sw ac0b002c	1	18	B0	C8	0
25	0							
26	0							
27	0							
28	0	1C	lw 8d2c0028	1	1C	A0	BC	0
29	0							
30	0							
31	0	1C	lw 8d2c0028	4	??	??	??	?
32	0	1C	lw 8d2c0028	0	1C	04	20	0
33	0	20	j 08000007	1	20	1C	3C	0
34	0	20	j 08000007	9	??	??	??	?
35	0	1C	j 08000007	0	1C	04	20	0
36	0	20	j 08000007	1	20	1C	3C	0
37	0	20	j 08000007	9	??	??	??	?

Table 6: Expected Instruction Trace