ENCM 369 Winter 2018 Lab 7 for the Week of March 5

Steve Norman
Department of Electrical & Computer Engineering
University of Calgary

March 2018

Lab instructions and other documents for ENCM 369 can be found at
http://people.ucalgary.ca/~norman/encm369winter2018/

Administrative details

You may work in pairs on this assignment

You may complete this assignment individually or with *one* partner.

Students working in pairs must make sure both partners understand *all* of the exercises being handed in. The point is to help each other learn *all* of the lab material, not to allow each partner to learn only half of it! Please keep in mind that you will not be able to rely on a partner to do work for you on midterm #2 or the final exam.

Two students working together should hand in a *single assignment* with names and lab section numbers for both students on the cover page. Names should be complete and spelled correctly. If you as an individual are making the cover page, please get the information you need from your partner. For partners who are not both in the same lab section, please hand in the assignment to the collection box for the student whose last name comes first in alphabetical order.

Due Date (no Late Due Date!)

The Due Date for this assignment is 3:30pm Friday, March 9.

Because of Midterm #2 on Monday March 12, solutions to marked exercises will be posted just after 3:30pm on March 9, and there will be no Late Due Date.

Marking scheme

A	6 marks
B	4 marks
C	4 marks
TOTAL	14 marks

How to package and hand in your assignments

Please see the Lab 1 instructions.

Exercise A: Datapath and control signals

Read This First

The point of this exercise is to make a detailed examination of the behaviour of the single-cycle processor circuit of Figure 7.11 in the course textbook.

What to Do, Part I

Suppose the instruction

```
lw $t8, 40($sp)
```

is located at address `0x0040_0154` in instruction memory. Suppose that just before the instruction is executed, the following GPRs have the given values:

```
$sp = 0x7fff_fe90
$t7 = 0x0005_4321
```

Finally suppose that data memory contains the following words:

```
ADDRESS       DATA
0x7fff_feb0   0x0001_2345
0x7fff_feb4   0x0002_3456
0x7fff_feb8   0x0003_4567
0x7fff_febc   0x0004_5678
0x7fff_fec0   0x0005_6789
0x7fff_fec4   0x0006_789a
0x7fff_fec8   0x0007_89ab
```

(Note: You have been given more information than you actually need to solve the problem.)

During the clock cycle in which the instruction is executed, most of the signals in the circuit will change values. This problem asks you to find the values of some of these signals *just before the end of the clock cycle*, when all of the signals will have stabilized to their final values for the clock cycle.

Determine and write out the values of these signals:

- The values of the control signals MemtoReg, MemWrite, Branch, ALUControl, ALUSrc, RegDst, and RegWrite. Use base two for the 3-bit ALUControl signal.
- The values of the 5-bit A1, A2, and A3 inputs to the Register File, as base two numbers.
- The values of the 32-bit ALU input signals and the ALUResult signal, in hexadecimal format.
- The value of the 32-bit WD3 input to the Register File, in hexadecimal format.
- The value, in hexadecimal format, of the 32-bit output of the adder that computes PCBranch, the branch target address. (The branch target address is useless in an `lw` instruction, of course, but this circuit will compute it anyway.)

(If you don't have a calculator that can display numbers in hexadecimal, you may want to use the C programs in `encm369w18lab07/exA` to help generate some of bit patterns you will need.)

What to Do, Part II

Repeat Part I, but this time assume that the instruction address is `0x0040_0158` and the instruction is

```
and $s2, $s2, $t3
```

Use these starting values for the source registers:

```
$s1 = 0x1002_d364
$t5 = 0xffff_ffc0
```

What to Hand In

Hand in your answers to Parts I and II. They may be either typed or neatly handwritten, whichever you find easier.

Exercise B: Immediate-mode instructions in the single-cycle machine

Read This First

A good way to see whether you understand the design of the circuit of Figure 7.11 in your textbook is to try to enhance it to support more instructions. When doing this kind of exercise it is important to remember this:

It's not enough to modify the datapath and control so that the new instructions work—the eight instructions already implemented must continue to work properly!

Let's consider adding support for the following useful instructions: `addi`, `slti`, `andi`, and `ori`.

It turns out that including just `addi` is easy. The datapath is already set up to have the ALU add a GPR and a sign-extended 16-bit constant. All we have to do is add a row for `addi` to the table that specifies the main decoder within the Control Unit. This is done in Section 7.3.3 of the textbook.

Since the datapath doesn't have to change at all, and no new control signals have been added, it's obvious that the eight existing instructions will still work.

However, the problem becomes more difficult if we move on to `slti`, `andi`, and `ori`. For the `andi` and `ori` instructions, the 16-bit constant is zero-extended to 32 bits, not sign-extended. For all three new instructions, the existing communication between the main decoder and the ALU decoder is not capable of getting the ALU to do an SLT, AND, or OR operation.

What to Do

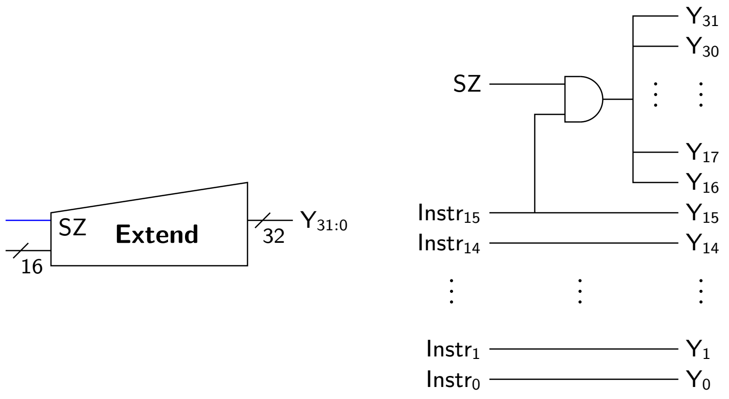
Suppose the Sign Extend unit of Figure 7.11 is replaced with the unit shown in Figure 1.

Suppose also that the ALUOp signal in textbook Figure 7.12 is widened from 2 bits to 3 bits. Bit patterns 000, 001, and 010 should mean what 00, 01, and 10 meant in the original design, and other 3-bit patterns can be used to support `slti`, `andi`, and `ori`.

Show how the above two modifications will allow support for this set of twelve instructions:

```
add, sub, and, or, slt,
lw, sw, beq,
addi, slti, andi, ori
```

Figure 1: A circuit to do either sign-extension, when SZ = 1, or zero-extension, when SZ = 0. A symbol is on the left, and a simple implementation with an AND gate is on the right.



Your answer should be:

- a few sentences to explain how the main decoder and ALU decoder within the Control Unit are modified, and what the SZ input of the Extend unit should be connected to;
- two tables, similar to textbook Tables 7.2 and 7.3, to completely specify how the redesigned main decoder and ALU decoder should behave.

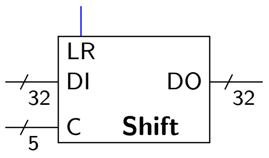
What to Hand In

Your answer, as described in "What to Do".

Exercise C: Support for shift instructions

Read This First

Let's return to the processor of Figure 7.11, forget about the immediate-mode instructions of Exercise B, and think about adding support for `sll` and `srl`. Suppose the following combinational logic unit is available:

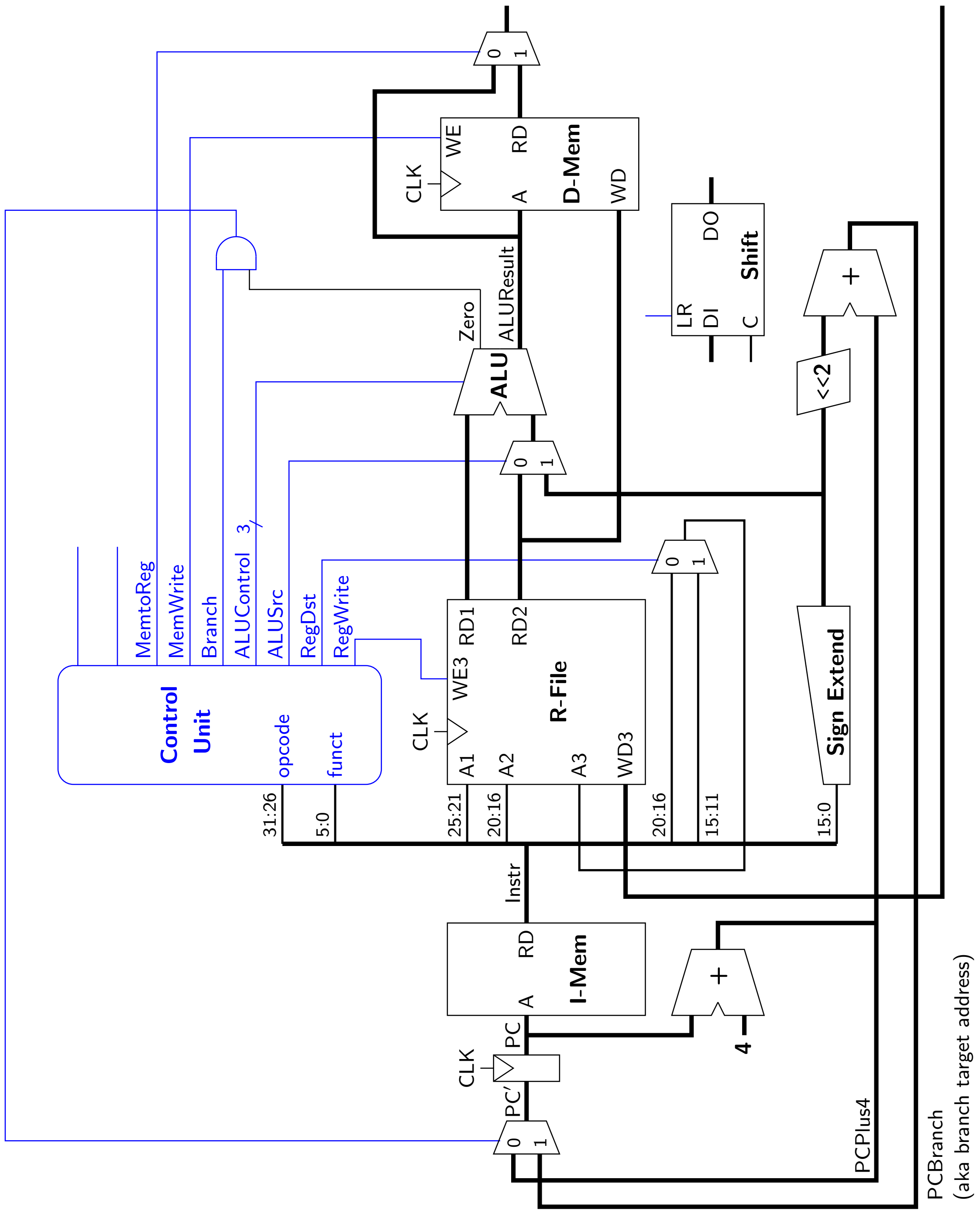


DI stands for "Data In" and DO stands for "Data Out". If the control input LR is 1, the unit does a left shift, and if it's 0, the unit does a right shift. The input C is a 5-bit shift count—for example, if C = 00010_{two}, the shift amount is two bit positions.

Note that the instruction formats are as follows, where `sssss` is a 5-bit code for the source GPR, `ddddd` is a 5-bit code for the destination GPR, and `ccccc` is a 5-bit unsigned shift count:

```
sll:  000000_sssss_00000_ddddd_ccccc_000000
srl:  000000_sssss_00000_ddddd_ccccc_000010
```

Figure 2: Incomplete single-cycle processor for Exercise C.



What to Do

Print page 5, so that you have a hard copy of Figure 2. This is a modified version of textbook Figure 7.11—the Shift unit has been added, along with two new unnamed control signals. As you can see, there are a number of unconnected wires.

Choose helpful names for the new control signals.

Add connections and functional units (hint: a new mux might be helpful) to the schematic to show how `sll` and `srl` can be supported, while allowing all of the eight original instructions to run correctly. Be as tidy and clear as you can with your wiring and labelling of wires.

For the Control Unit, it turns out to be tedious to work through all of the changes needed to the two-level main-decoder/ALU-decoder structure. Instead, let's just specify the Control Unit as a single element. To do that, make a table that looks like this:

Instruction			**RegWrite**	**RegDst**	**ALUSrc**	**ALUControl**	**Branch**	**MemWrite**	**MemtoReg**
ADD			1	1	0	010	0	0	0
SUB			1	1	0	110	0	0	0
AND			1	1	0	000	0	0	0
OR			1	1	0	001	0	0	0
SLT			1	1	0	111	0	0	0
LW			1	0	1	010	0	0	1
SW			0	X	1	010	0	1	X
BEQ			0	X	0	110	1	0	X
SLL									
SRL									

Write in the names you've chosen for your new control signals, and fill in the two blank columns and two blank rows with 0's, 1's, and X's, so that all ten instructions will work correctly.

What to Hand In

Hand in your completed schematic and your completed table for the Control Unit.

Exercise D: A single-cycle machine for a different instruction set

Attention!

This exercise will not be marked, because it's pretty easy to find a solution for it using previous years' ENCM 369 web pages. *However, please do not think of this as an optional exercise. Doing the work may be very helpful as you prepare for Midterm #2.*

Read This First

This exercise is adapted from a problem on the Winter 2006 final exam.

The Exam16 ISA (instruction set architecture) describes a system in which addresses, instructions, and data words are all 16 bits wide. It has sixteen 16-bit general purpose registers, and a 16-bit PC. The instructions of Exam16 are given in the table in Figure 3. Note that unlike with MIPS, there are no offsets built into Exam16 load and store instructions.

Figure 4 is a nearly-complete datapath for a computer that implements the Exam16 ISA. It is very much in the style of the ***single-cycle*** MIPS subset implementation studied in ENCM 369. Note that there are two 16-bit adders and a 16-bit ALU. The ALUOp signal works as follows: 00 asks for addition, 01 for subtraction, and 10 for set-on-less-than. The circuit labeled "All Bits 0?" is a big NOR gate—the 1-bit output is 1 if all 16 input bits are 0, and is 0 otherwise.

Data Memory in Figure 4 works very slightly differently from Data Memory in examples in the 2018 textbook and 2018 lectures. In Figure 4, MemRead and MemWrite should both be zero for instructions that do not access memory, to avoid wasting energy. Values of MemRead and MemWrite are obvious for loads and stores, I hope!

Figure 3: Exam16 instruction set.

Mnemonic	Format	Description
`add`	`0000_ssss_tttt_dddd`	Add source registers selected by bits `ssss` and `tttt`, put result in register selected by bits `dddd`.
`sub`	`0001_ssss_tttt_dddd`	Same as `add`, except ALU operation is subtraction.
`slt`	`0010_ssss_tttt_dddd`	Same as `add`, except ALU operation is set-on-less-than.
`brz`	`0011_ssss_oooo_oooo`	Branch if register is zero: If register selected by bits `ssss` contains zero, branch forward or backward by number of instructions in 8-bit 2's-complement offset `oooo_oooo`.
`lw`	`0100_0000_aaaa_dddd`	Using register selected by bits `aaaa` as an address, load word from data memory into register selected by bits `dddd`.
`sw`	`0101_ssss_aaaa_0000`	Using register selected by bits `aaaa` as an address, store word from register selected by bits `ssss` into data memory.

What to Do

In some of the following parts, it may be helpful to draw a simple diagram or two to go along with the text you write to answer the question being asked.

Part a. The Address and Write Data inputs to the Data Memory are not connected to anything in Figure 4. What signals should be sent to these inputs? Explain why.

Part b. The WD3 input to the Register File is not connected to anything. *How should this signal be driven?* Here is a hint: Introduce a new control signal, give it a name, and use it to control a multiplexer. *Briefly give reasons to support your design.*

Part c. The input to the PC register is not connected to anything. *How should this signal be driven?* A new control signal, a new multiplexer, and perhaps some other new, simple logic element will be needed. *Briefly give reasons to support your design.*

Figure 4: Nearly complete datapath for Exam16 single-cycle implementation.

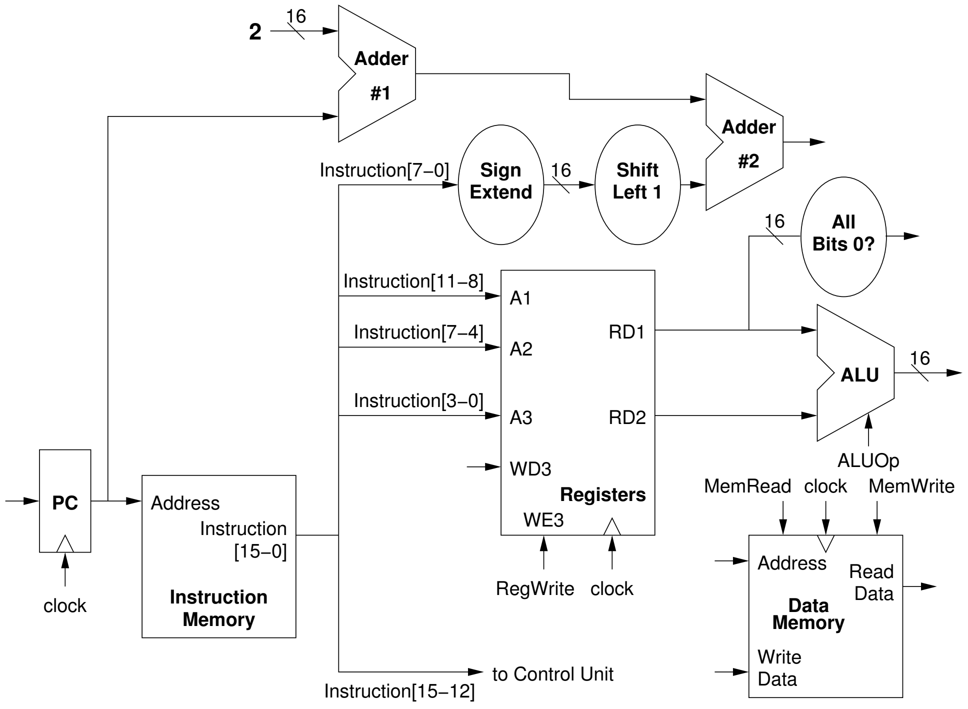


Figure 5: Table of control signals for **part d** of Exercise D.

Instruction	MemRead	MemWrite	RegWrite	ALUOp		
`add`						
`sub`						
`slt`						
`brz`						
`lw`						
`sw`						

Part d. Make a copy of the table from Figure 5, or print the page that it is on. Then fill in all the blank cells with the correct values of control signals for the given instructions.

The last two columns are reserved for the new control signals you introduced in **parts b and c**—please write in the names of these signals.

Use "X" in table cells to indicate that a particular control signal is a "don't care" for a particular instruction.

Part e. Suppose you want to extend the Exam16 ISA to include an `addc` ("add constant") instruction with the following format:

    1000_ssss_cccc_dddd

`ssss` encodes the source register, `dddd` encodes the destination register, and `cccc` encodes a constant in the range from 0 to 15. Describe all the datapath changes (not control changes) that would be needed to add support for `addc` while continuing to support the original six Exam16 instructions.

What to Hand In

Nothing. You can check your answers against solutions that will be posted well in advance of Midterm #2.