

## **Homework 5 - Solutions** **(Control Unit and ALU design)** **Maximum points: 45 points**

### **Directions**

This assignment is due Monday, Oct. 31st. Submit your solutions on a separate sheet of paper.

### **Learning Objectives**

In the process of completing this homework assignment, students will develop their abilities to

- Design control units to support machine language instruction sets by applying combinational and sequential digital logic design principles.
- Design digital logic circuits for computer arithmetic.
- Predict the qualitative effect on clock cycle time of modifications to the design of digital logic circuits

### **General Instructions**

- For the micro-programming problems, you do not have to re-produce the micro-program in Figure 5.7.3. Assume that that part of the micro-program already exists and write down only the new micro-instructions that are required. Also, if you create any new values for the fields or new fields, clearly specify what each new value/field denotes.
- Submit your answers on a separate sheet of paper.
- The state diagram for the FSM representation of the control unit (Figure 5.38) is provided on the class website.
- Figures 5.7.2 and 5.7.3 that deal with micro-programming are provided on the class website.
- Figures for the 32-bit adder are also provided on the class website. Make modifications on these figures for Problem 4.

### **Problems**

1. [15 points] Figure 5.7.3 in Hennessy and Patterson (the CD actually), shows the microprogram for the control unit of the MIPS architecture that can handle `lw`, `sw`, `beq`, `j` and R-type instructions. Add microcode to this program to implement the following MIPS instructions:

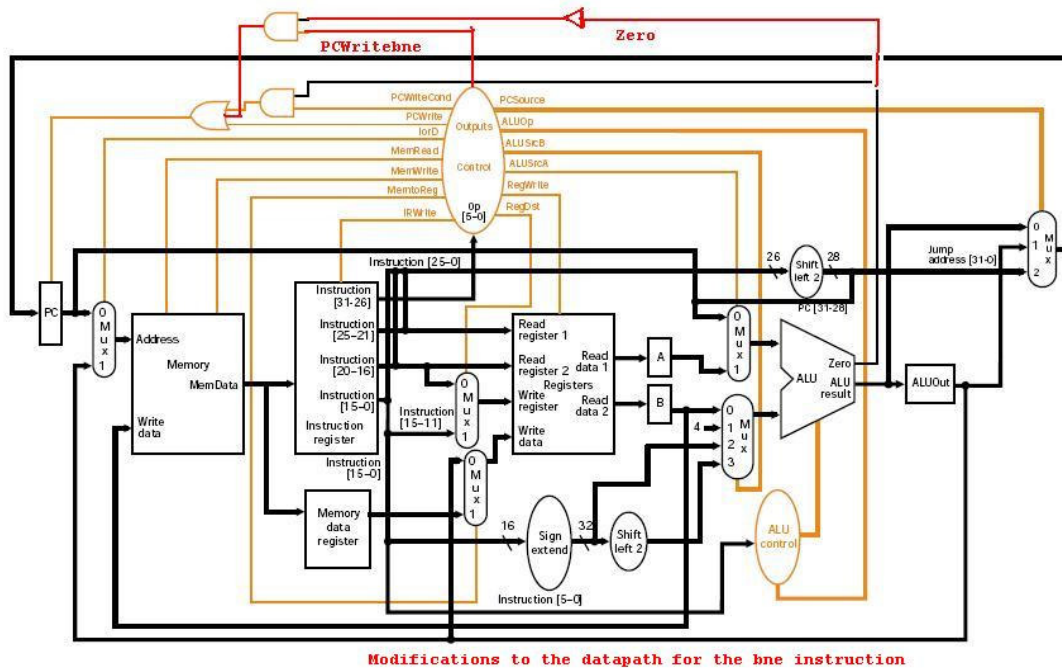
- a. `bne`
- b. `jr`
- c. `lui`

Figure 5.7.2 on the CD lists the control fields and their values that the microprogram uses.

The RTL for the bne is provided below:

1.  $PC = PC + 4$ ;  $IR = Mem[PC]$
2.  $A = Reg[IR[25:21]]$ ;  $B = Reg[IR[20:16]]$ ;  
 $ALUOut = PC + (SE[IR[15:0]] \ll 2)$ ;  
 If  $(IR[31:26] == 5)$  then
3. If  $(A - B \neq 0)$   $PC = ALUOut$

The modified datapath for bne is below:



The RTL and the modified datapaths for the jr and lui instructions are available as solutions to HW3 and HW4. You must use the provided solutions for questions 1 and 2 on this homework.

This table should be appended to table 5.7.3

Label	ALU-Control	Src1	Src2	RegisterControl	Memory	PCWrite-Control	Sequencing	ALUOut-Control
BNE1	Sub	A	B	-	-	ALUOut-cond-notZero	Fetch	From ALU
JR1	-	-	-	-	-	Write A	Fetch	
LUI1	-	-	-	-	-	-	Seq	From A
	-	-	-	WriteALU	-	-	Fetch	

New fields and values:

The PCWriteControl field has a new value ALUOut-cond-notZero whose function is “If the Zero output of the ALU is zero, write the PC with the contents of the register ALUOut”

The PCWriteControl field has another new value Write A whose function is “Write the output of register A into the PC.”

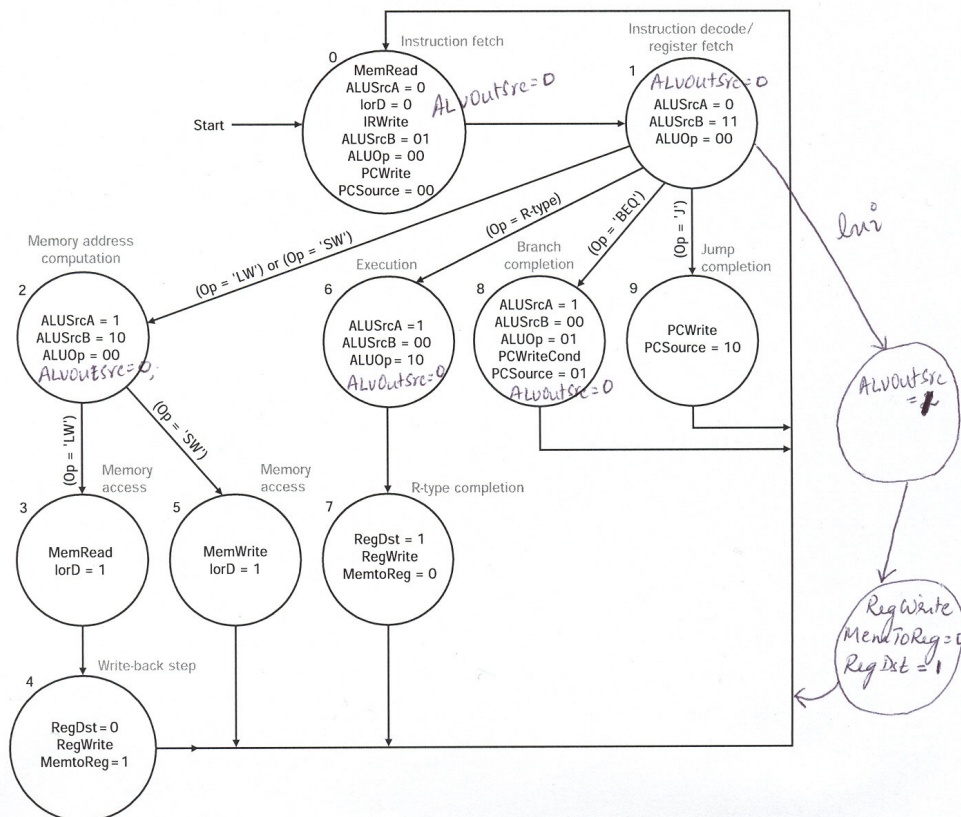
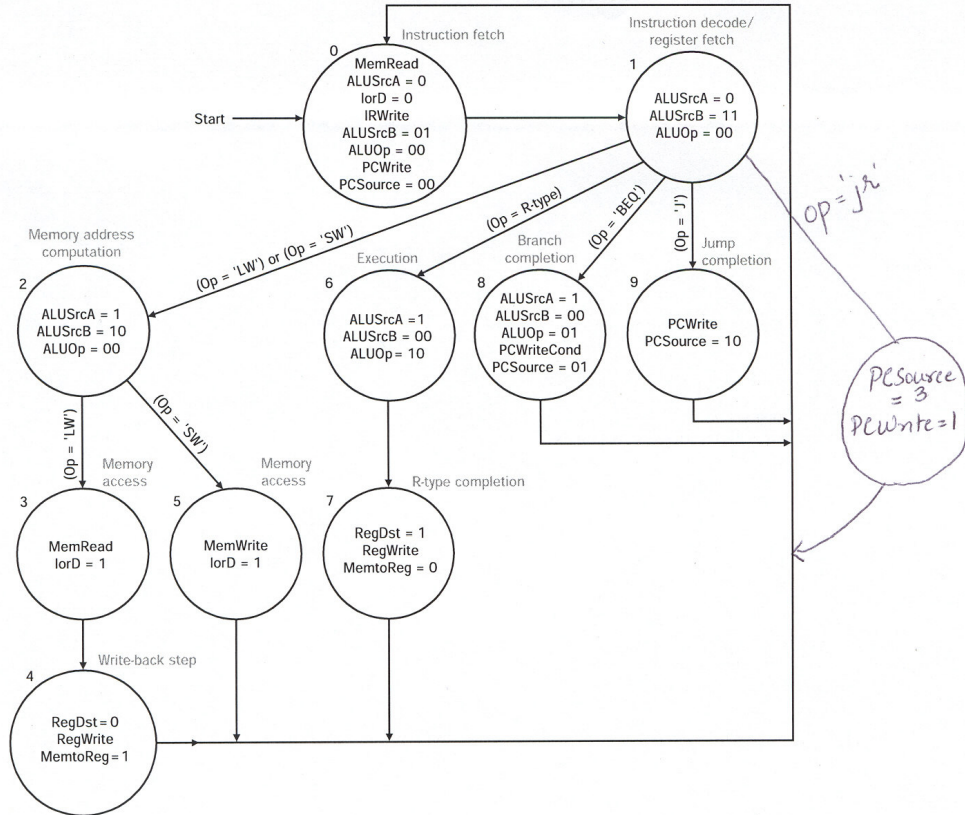
A new field ALUOut-Control is added. This field takes two values:

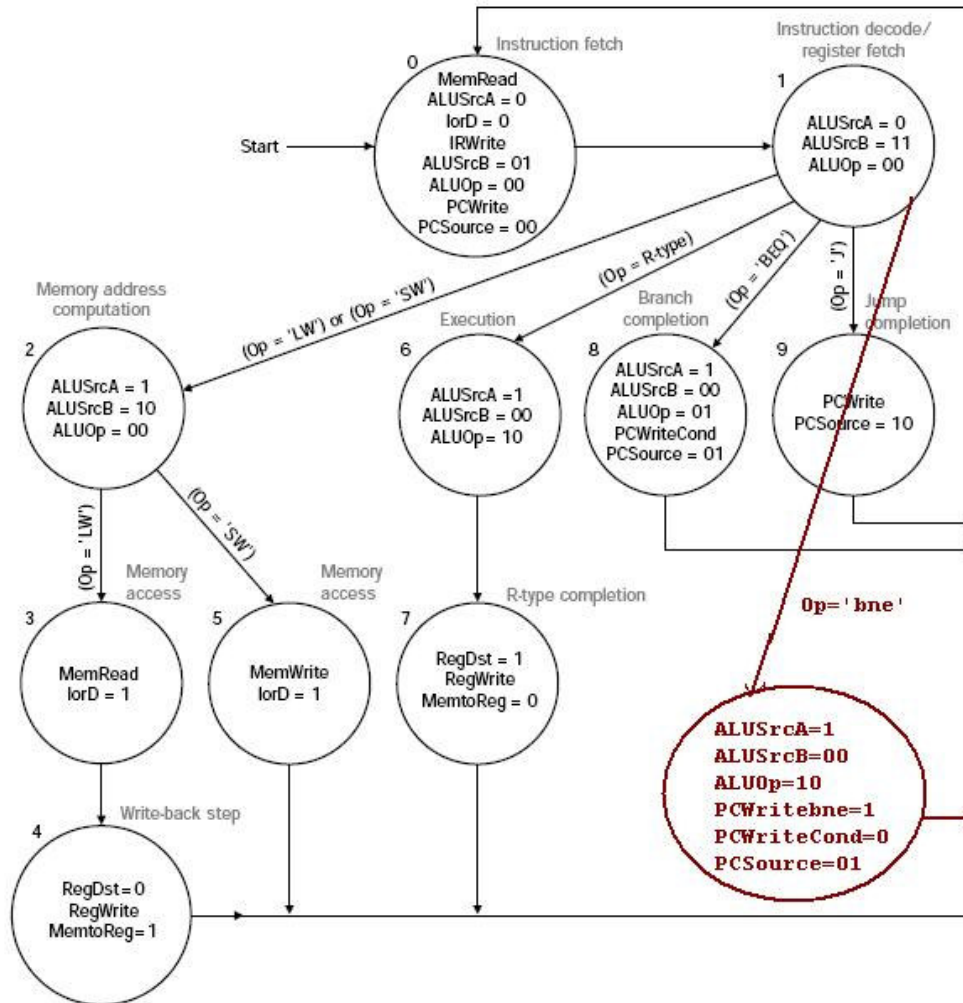
- a. From ALU – Write the output of the ALU to the ALUOut register.
- b. From A – Write the output of register A to the ALUOut register.

This new field will result in changes to the instructions labeled Fetch, the next instruction, Mem1, R-format1, and Beq1. For all these instructions where ALUOut is the intended output of the ALU, the ALUOut-Control field's value will be From ALU.

2. [15 points] For each of the problems below, modify the textbook's multi-cycle control (Figure 5.38 on page 345) to implement the indicated MIPS instruction. Use a separate printout for each of the instructions. A copy of Figure 5.38 is provided on the class website.

- a. bne
- b. jr
- c. lui



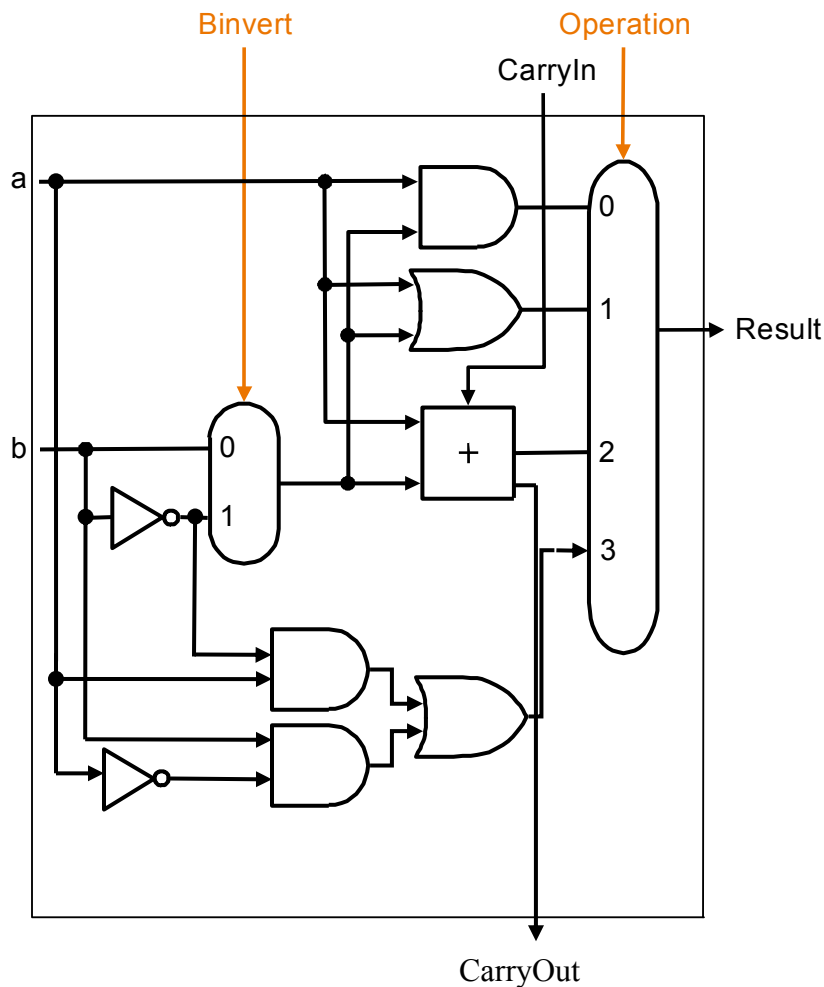


3. [5 points each] For each of the problems below, modify the 32-bit ripple-carry ALU developed in class as indicated. Use only inverters, AND gates, OR gates, and multiplexers. In each case, describe any new or modified control signals. Also, assuming that the ALU is currently on the critical path for your design, determine whether or not your modifications would extend the clock cycle time.

a. Modify the ALU to support the MIPS xor instruction.

The modified 1-bit ALU is shown below. The Operation control signal will have to be “3” to select the XOR output. The rest of the control signals are don’t-cares.

*There is no additional delay introduced with the inclusion of the XOR gate.*



b. Modify the ALU so that it detects overflow for both addition and subtraction (i.e. so that it has a new 1-bit output called Overflow which is asserted iff overflow occurs). Read pages 170-172 of your textbook and Figure 3.3 for more information.

Two possible ways to determine overflow are explained below. Other solutions are also possible.

1. If the CarryIn to the most significant bit's adder module is different from its CarryOut, overflow has occurred. Therefore, for a 32-bit ALU, to check for overflow, in the 32nd ALU module, add an EXOR gate to evaluate the overflow.

$$\text{Overflow} = \text{CarryIn}_{31} \oplus \text{CarryOut}_{31}$$

2. A more intuitive way to determine overflow, is by examining the signs of the numbers being added/subtracted and the sign of the output obtained. For a 32-bit number  $a$ , represented in 2's complement,  $a_{31}$  is the sign bit. Using Figure 4.4 from Hennessy and Patterson, the following truth table can be obtained.

Binvert (= 1 => subtraction; = 0 => addition)	$a_{31}$ (= 1 => a is negative, else positive)	$b_{31}$ (= 1 => b is negative, else positive)	$X_{31}$ (= 1 => Result is negative, else positive)	Overflow
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

From this table, we can obtain the following logic equation:

*Overflow*

$$= \overline{\text{Binvert}} \cdot \overline{a_{31}} \cdot \overline{b_{31}} \cdot X_{31} + \overline{\text{Binvert}} \cdot a_{31} \cdot b_{31} \cdot \overline{X_{31}}$$

$$+ \text{Binvert} \cdot \overline{a_{31}} \cdot b_{31} \cdot X_{31} + \text{Binvert} \cdot a_{31} \cdot \overline{b_{31}} \cdot \overline{X_{31}}$$

*In either case, since overflow can be detected only after the 32nd ALU module has finished its computation, the clock cycle time is extended.*

- c. Modify the ALU so that it implements slt correctly even when overflow occurs. Assume that you have a combinational logic unit that detects overflow, as you are required to design for the previous problem.

To implement slt correctly, we need to introduce the logic equation obtained from the following truth table (for values not displayed in the table, Set has to be zero), to the ALU module of the MSB.

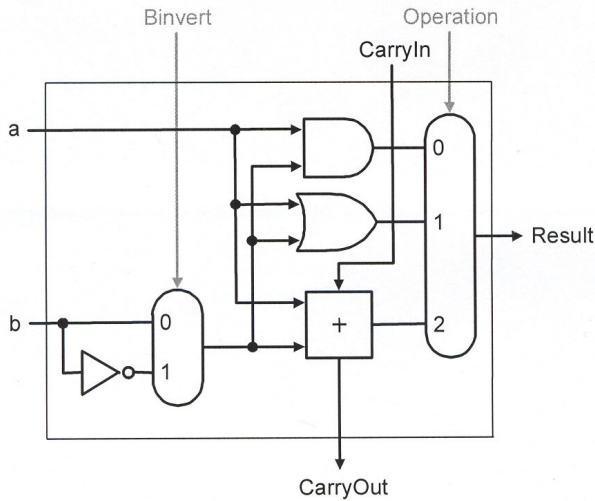
	$a_{31}$	$b_{31}$	$X_{31}$	Overflow	Less <sub>0</sub> = Set
$a \geq b$	0	0	0	0	0
$a < b$	0	0	1	0	1
$a \geq b$	0	1	0	0	0
$a \geq b$	0	1	1	1	0
$a < b$	1	0	0	1	1
$a < b$	1	0	1	0	1
$a \geq b$	1	1	0	0	0
$a < b$	1	1	1	0	1

From the truth table, we get the following logic equation:

$$\text{Set} = X_{31} \cdot \overline{\text{Overflow}} + \overline{X_{31}} \cdot \text{Overflow}$$

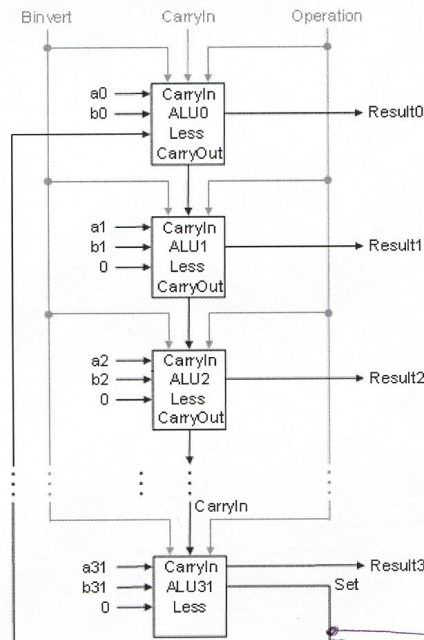
*Since, overflow can be detected only after  $X_{31}$  is obtained and Less<sub>0</sub> can be obtained after overflow has been detected, the clock cycle time is increased.*

It is also possible to obtain the correct value of Set, without the Overflow signal, by instead examining  $a_{31}$ ,  $b_{31}$  and  $X_{31}$ .



1-bit ALU

*Modifications to handle "slt" correctly even if overflow occurs.*



32-bit ALU

*Overflow detector*

*overflow = 0 ⇒ No overflow  
 = 1 ⇒ Yes, overflow*