Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Prof. Archana Chidanandan

Homework 3 - Solutions
(RTL, Datapath and Control Unit)
Maximum points: 75 points

Directions

This assignment is due Friday, Jan. 28th. 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

- Describe the implementation of machine language instructions using Register Transfer Language.
- Design datapaths to support machine language instruction sets by specifying the interconnections between components and the behavior of the associated control units.
- 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.
- If you are unsure of the RTL and datapath modifications that are required for Problems 1 and 2, refer to the class website for solutions to Homework 2.
- Figures for the 32-bit adder are also provided on the class website. Make modifications on these figures for Problem 4.

Problems

1. [20 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. `mfc0`

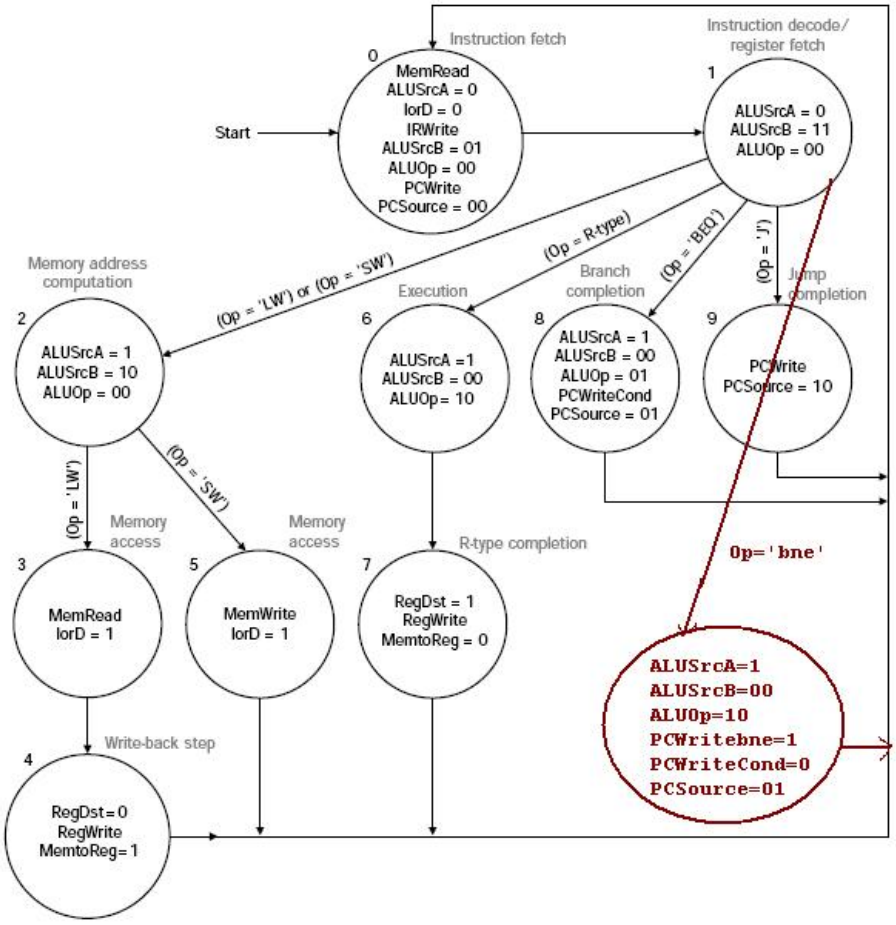
c. `mtc0`
d. `jal`

Figure 5.7.2 on the CD lists the control fields and their values that the microprogram uses.
The micro-code is as follows:

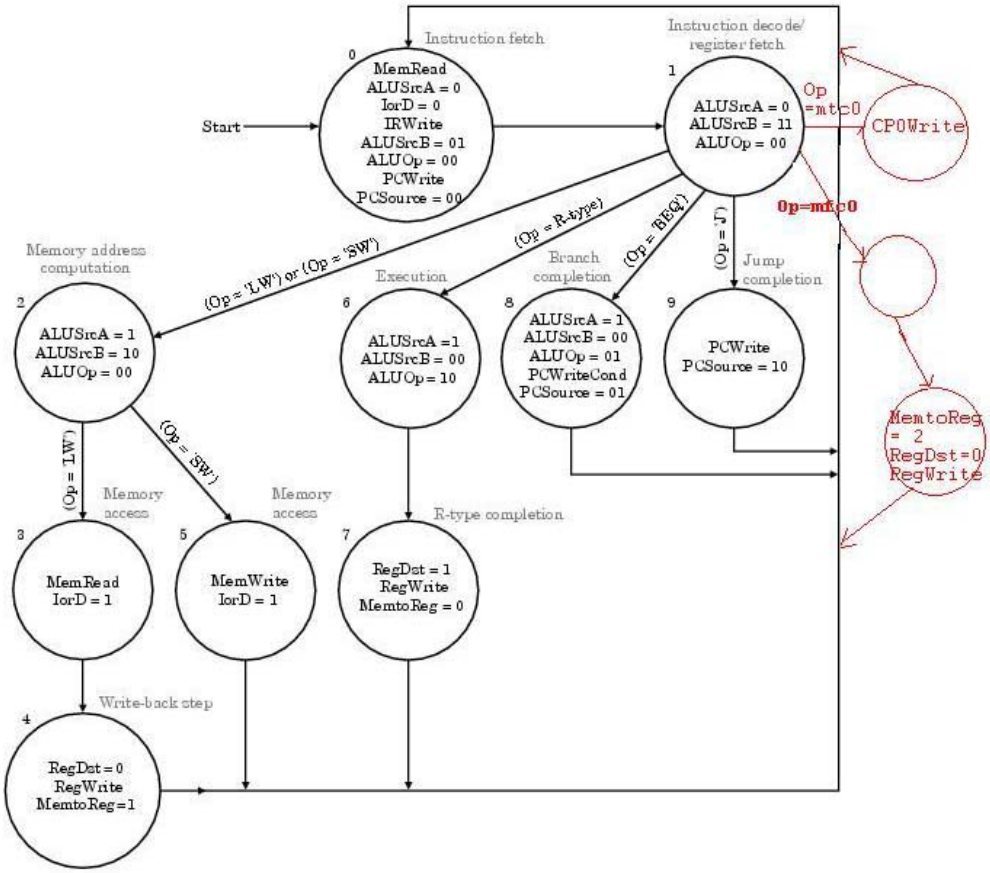
Label	ALU control	Src1	Src2	Register control and B input	Co-Processor Register control	Memory control	PCWrite control	Sequencing
Bne-1	Subt	A	B				ALU-Out-not Equal	Fetch
Mfc0-1						Read CoP		Seq
				Write D				Fetch
Mtc0-1					WriteCoP-B			
Jal-1				Write to Reg31			Jump Address	Fetch

2. [20 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.

a. `bne`

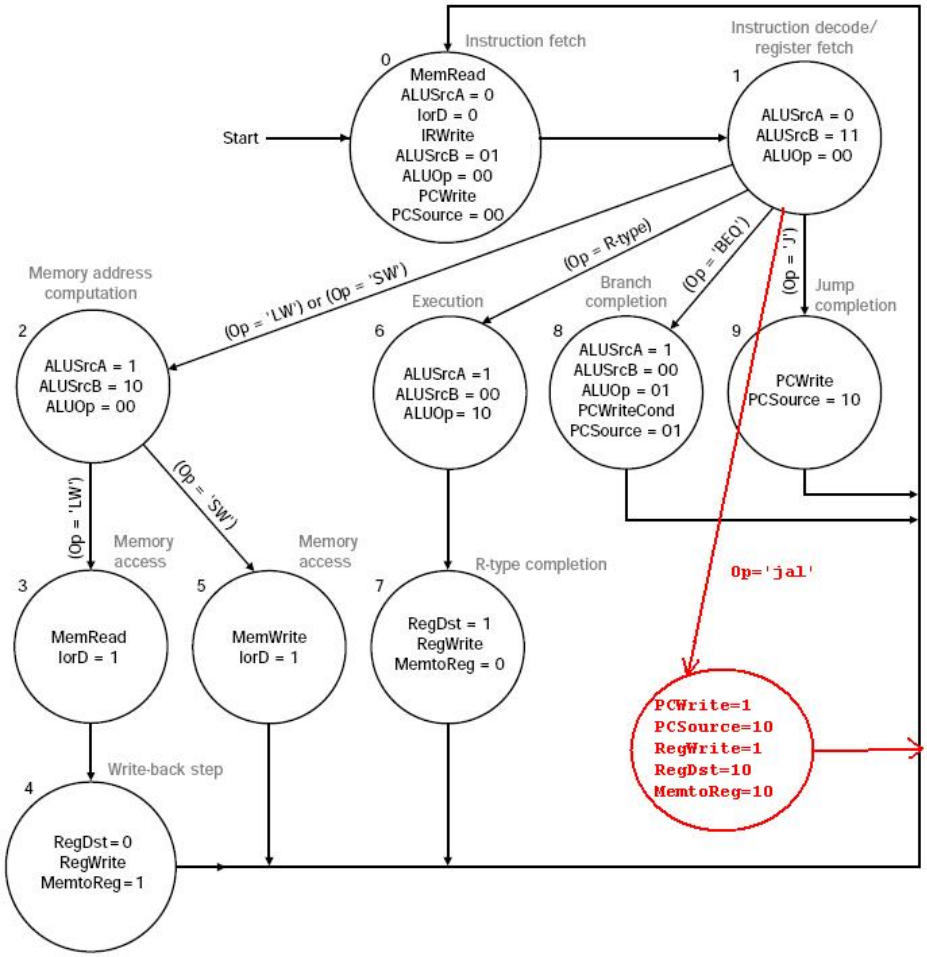


b. mfc0
c. mtc0



Control modified to implement the mfc0 and mtc0 instructions

d. jal


Start
Instruction fetch
0
MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 01
ALUOp = 00
PCWrite
PCSource = 00
Instruction decode/
register fetch
1
ALUSrcA = 0
ALUSrcB = 11
ALUOp = 00
(Op = 'LW') or (Op = 'SW')
(Op = R-type)
(Op = 'BEQ')
(Op = 'J')
Memory address
computation
2
ALUSrcA = 1
ALUSrcB = 10
ALUOp = 00
Execution
6
ALUSrcA =1
ALUSrcB = 00
ALUOp= 10
Branch
completion
8
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01
PCWriteCond
PCSource = 01
Jump
completion
9
PCWrite
PCSource = 10
(Op = 'LW')
(Op = 'SW')
Memory
access
Memory
access
R-type completion
3
MemRead
IorD = 1
5
MemWrite
IorD = 1
7
RegDst = 1
RegWrite
MemtoReg = 0
Op='jal'
PCWrite=1
PCSource=10
RegWrite=1
RegDst=10
MemtoReg=10
Write-back step
4
RegDst = 0
RegWrite
MemtoReg = 1

3. Consider the `addmem` instruction - The description is provided below.

`addmem rd, rs, rt`

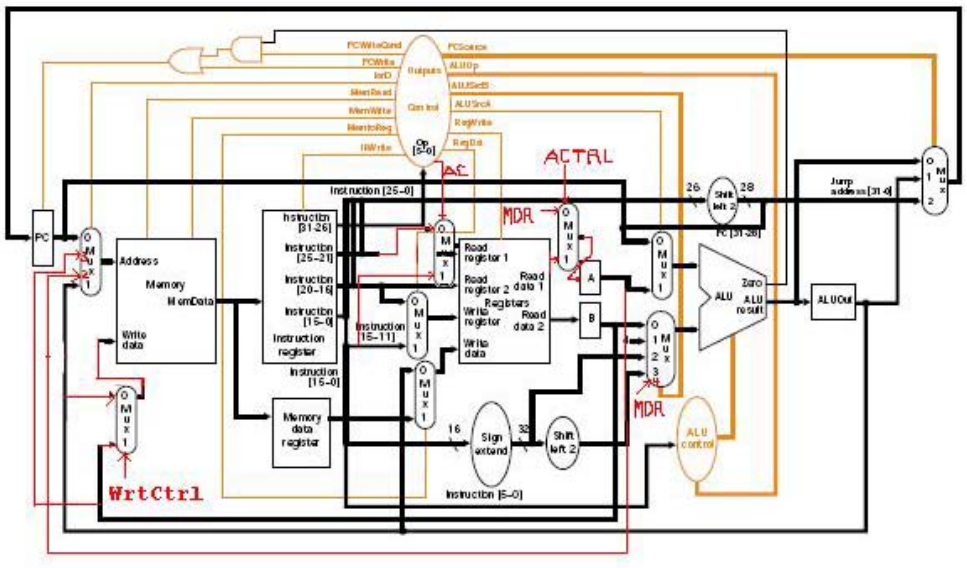
The instruction reads two values from memory, add the values and stores the result back in memory. The addresses of the values to be added are the contents of registers `rs` and `rt`. The address to which the result must be stored is available in register `rd`.

a. [5 points]Write a multicycle RTL description of an implementation of the `addmem` instruction that uses as few cycles as possible without extending the clock cycle of your design.

One possible solution is:

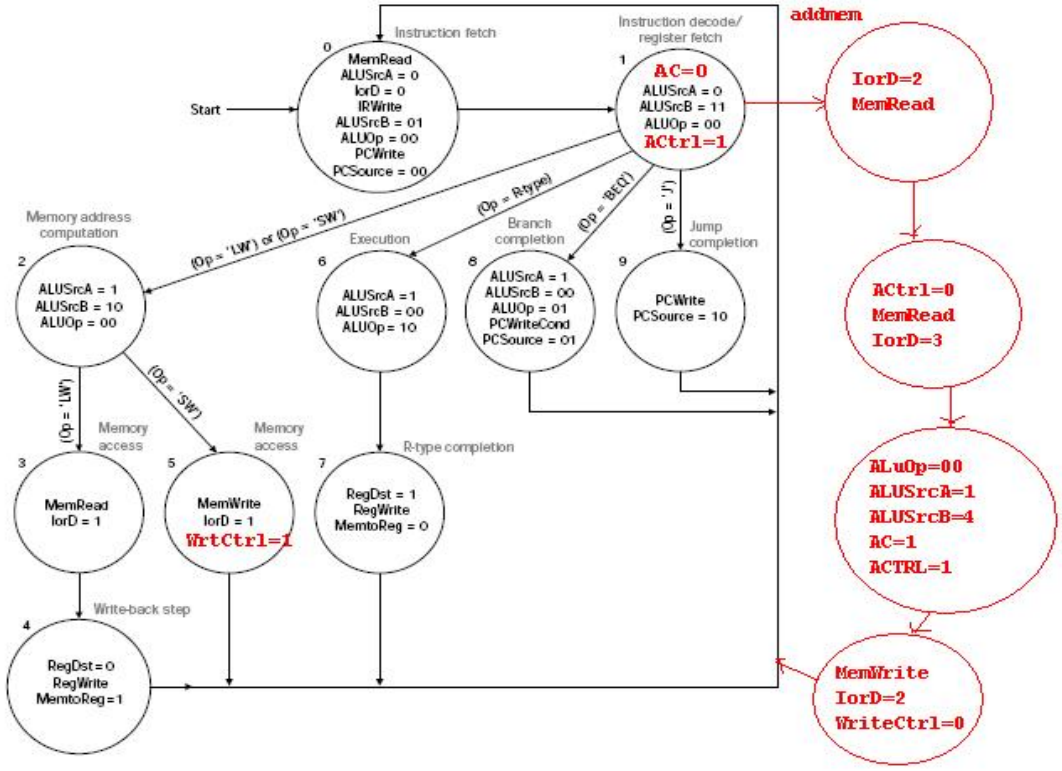
1. PC = PC + 4; IR = Mem[PC]
2. A = Reg[IR[25:21]]
   B = Reg[IR[20:16]]
   ALUOut = PC + (SE(IR[15-0]) << 2)
   If(opcode == Addmem) then
3. MDR = Mem[A]; B = Reg[IR[20:16]]
4. A = MDR; MDR = Mem[B];
5. ALUout = A + MDR; A = Reg[IR[15:11]]
6. Mem[A] = ALUOut

b. [5 points] Add any necessary datapath components and control signals to the multicycle datapath. A pdf version of Figure 5.28 is available on the class website(Homework).



Modfications to the datapath for the addmem instruction.

c. [5 points] Modify the textbook's multi-cycle control (Figure 5.38 on page 345) to implement the indicated instruction. Use a separate printout for the instruction.



d. [5 points]Add microcode to the program in Figure 5.7.3 to implement the `addmem` instruction.

Label	ALU control	Src1	Src2	Register control and B input	Co-Processor Register control	Memory control	PCWrite control	Sequencing
Addmem1						Read using A	-	Seq
				WriteMDRToA		ReadUsingB		Seq
	Add	A	MDR	ReadRd-and-WriteRegToA				Seq
						WriteALUOutAtA		Fetch

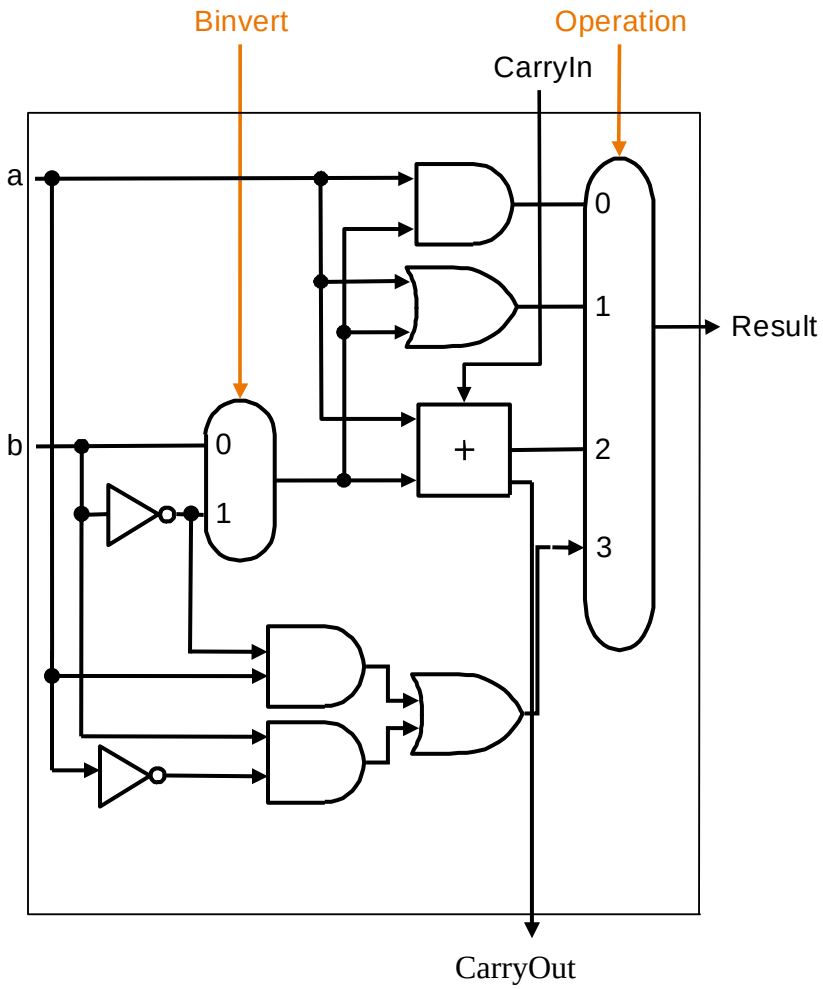
ReadUsingA – Read memory using the address in A.
WriteMDRToA- Write to Register A the value in MDR
ReadUsingB – Read memory using the address in B.
ReadRd-and-WriteRegToA – Read a value from the register using Rd as the address and write result to register A.
WriteALUOutAtA – Write to memory the value in ALUOut using the address specified by A.

4. [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:

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

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	$Less_0$ = Set
a>= b	0	0	0	0	0
a < b	0	0	1	0	1
a >= b	0	1	0	0	0
a >= b	0	1	1	1	0
a < b	1	0	0	1	1
a < b	1	0	1	0	1
a >= b	1	1	0	0	0
a < b	1	1	1	0	1

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

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

Since, overflow can be detected only after X_{31} is obtained and Less0 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}.