Rose-Hulman Institute of Technology
Computer Science and Software Engineering
Profs. Laurence Merkle and Archana Chidanandan

Homework 4 - Solutions (Datapath and Control Design, Computer Arithmetic)

1. For each of the problems below, modify the textbook's multi-cycle datapath and control (Figure 5.33 on page 383 and Figure 5.42 on page 396, respectively) to implement the indicated MIPS instruction.

   a. Modify the datapath and control to implement the MIPS `addi` instruction.
   b. Modify the datapath and control to implement the MIPS `mfc0` instruction.
   c. Modify the datapath and control to implement the MIPS `mtc0` instruction.

   The RTL is provided for clarity:

   a. `addi rt, rs, imm` - Put the sum of register `rs` and the sign-extended immediate into register `rt`.

```
1. PC = PC +4;
   IR = Mem[PC]
2. A = Reg[IR[25:21]];
   B = Reg[IR[20:16]];
   Sum = PC + (SE[IR[15:0]] << 2);
   If (IR[31:26] = = 15) then
3. ALUOut = A + SE[IR[15:0]]
4. Reg[IR[20:16]] = ALUOut
```

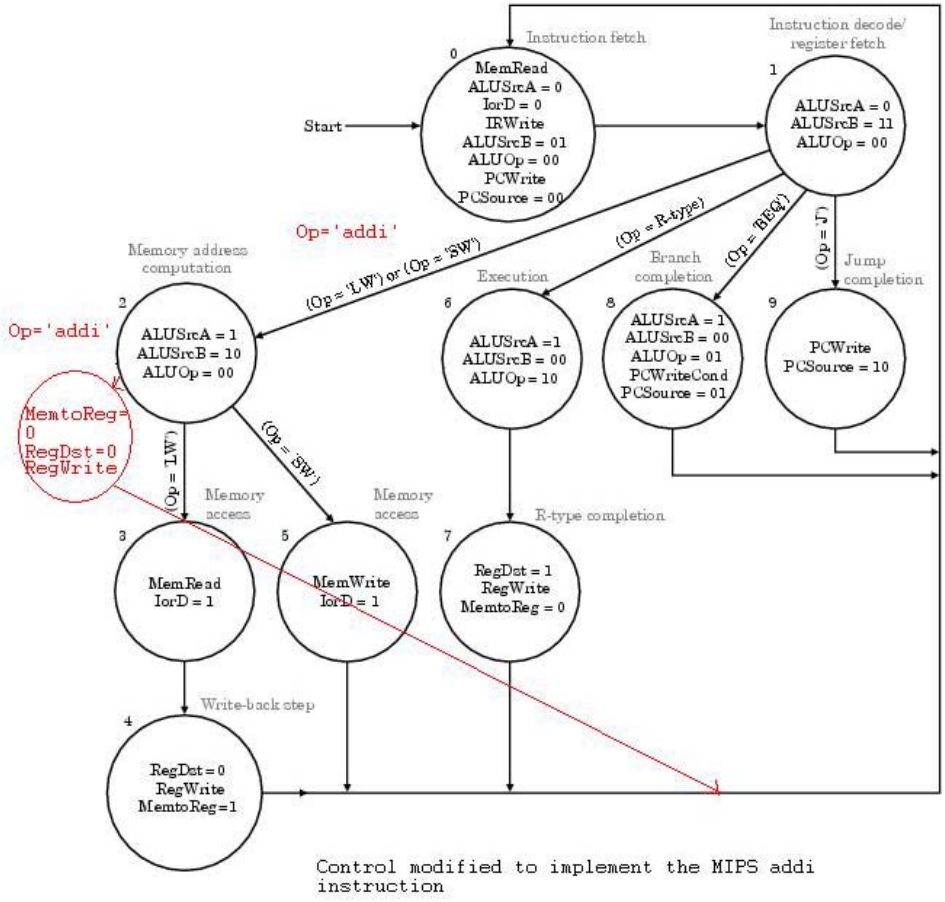
   b. `mfc0 rt, rd` - Move the contents of co-processor 0's register `rd` to register `rt`.

```
1. PC = PC +4;
   IR = Mem[PC]
2. A = Reg[IR[25:21]];
   B = Reg[IR[20:16]];
   Sum = PC + (SE[IR[15:0]] << 2);
   D = CoP0Reg[IR[15:11]];
   If((IR[31:26]==0)and(IR[25:21]==0)) then
3. Reg[IR[20:16]] = D;
```

   c. `mtc0 rt, rd` – Move the contents of register `rd` to co-processor 0's register `rt`.

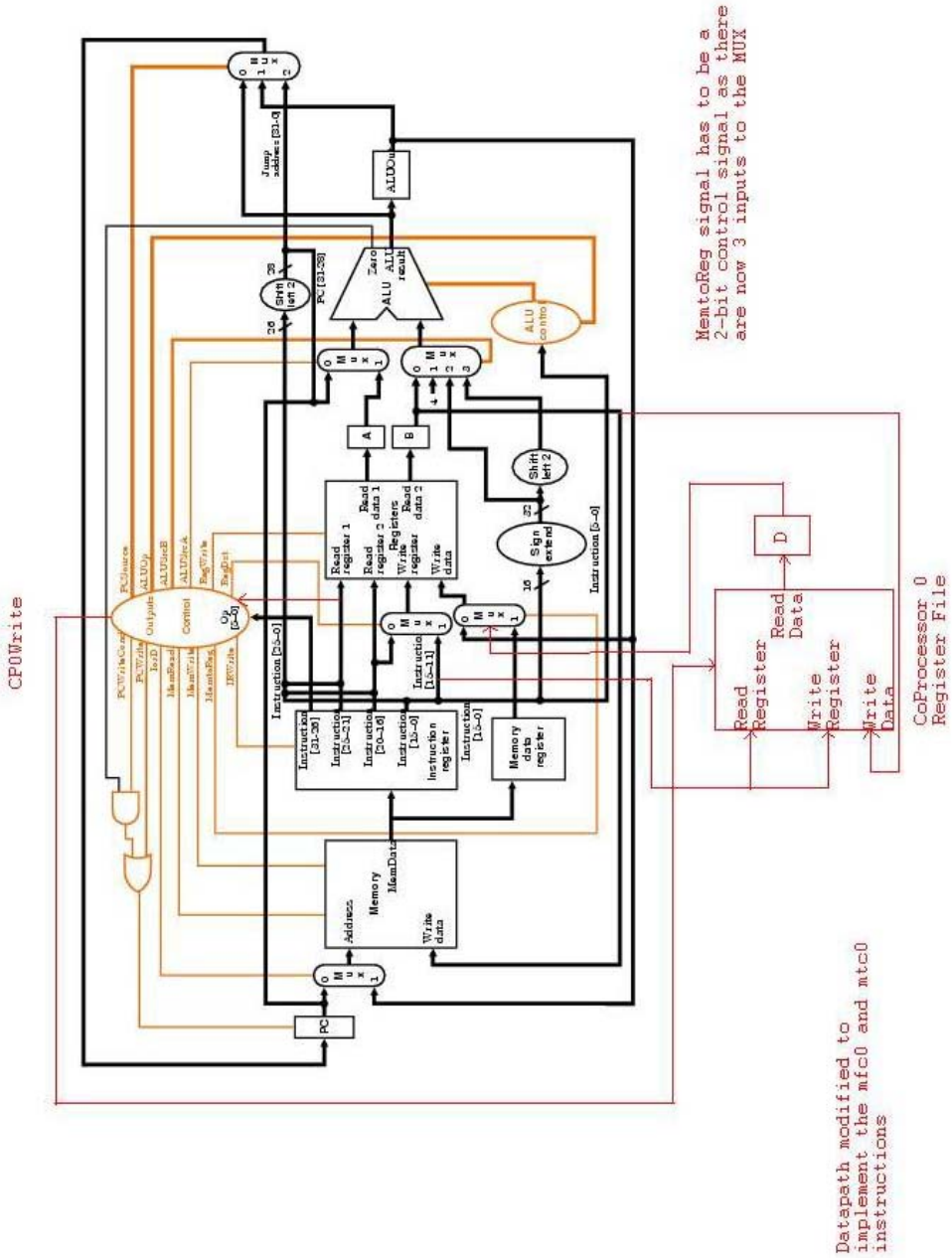
```
1. PC = PC+ 4;
   IR = Mem[PC];
2. A = Reg[IR[25:12]];
   B = Reg[IR[20:26]];
   Sum = PC + (SE[IR[15:0]] << 2);
   D = CoP0Reg[IR[15:11]];
   If ((IR[31:26]==0)and(IR[25:21]==4)) then
3. CoP0Reg[IR[15:11]] = B;
```

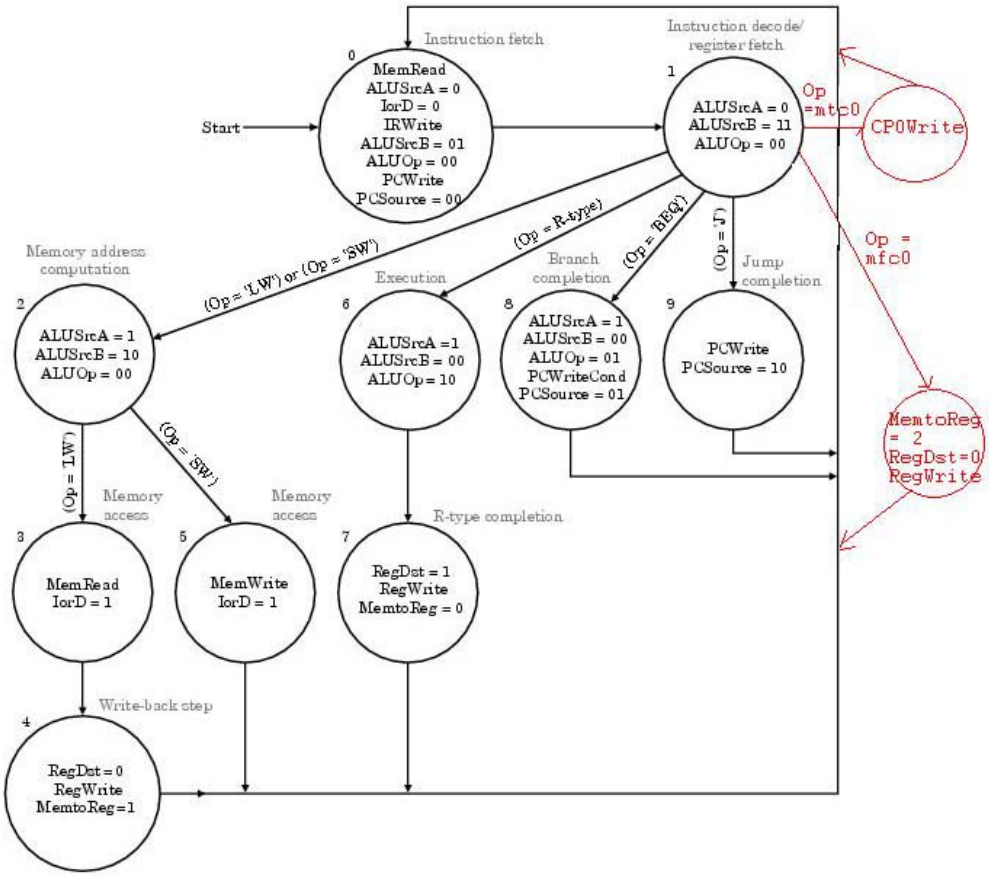
No modifications are required to the datapath for the `addi` instruction. The modified state diagram for the control unit is shown below. Another possible solution is to have a stage after stage 1 that is identical to stage 2, followed by the additional stage, as shown in red, in the figure below.



Control modified to implement the MIPS addi instruction

The following datapath and control diagrams show the changes required to implement the MIPS `mfc0` and `mtc0` instructions.

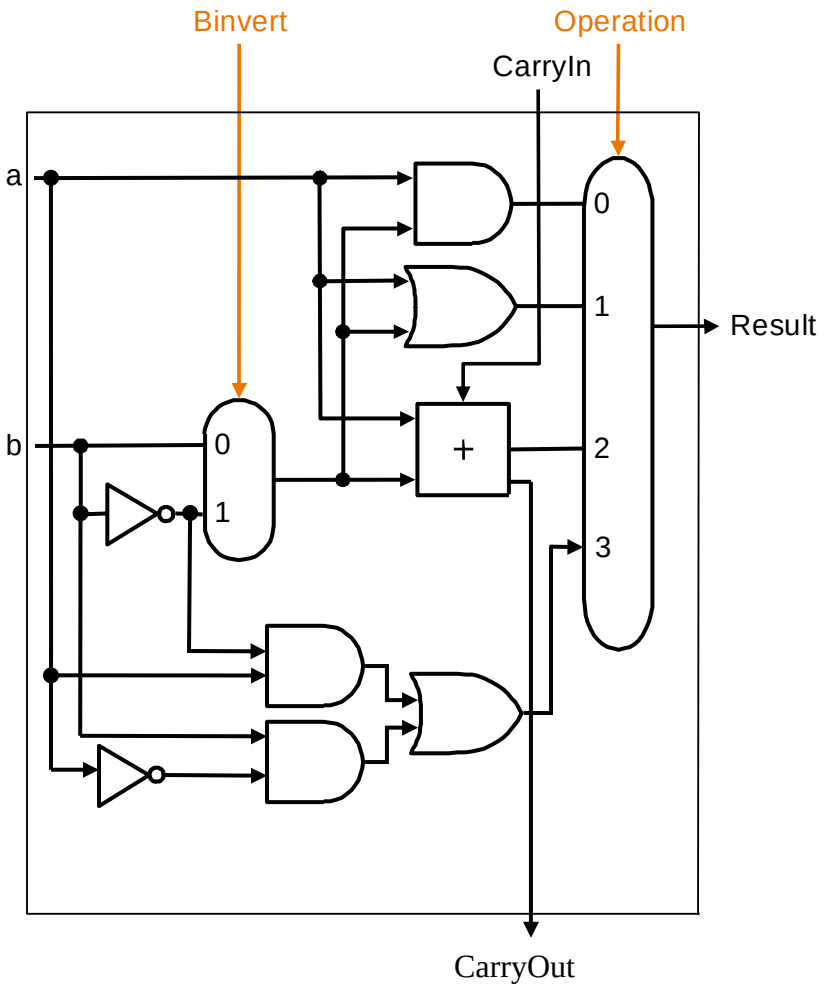




Control modified to implement the mfc0 and mtc0 instructions

2. 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 of 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 combination of the adder and the multiplexer on its second input will take longer than the XOR. Thus, the XOR will not extend the clock cycle.

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 if overflow occur.

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:

$$\begin{aligned} &\textit{Overflow} \\ &= \overline{Binvert} \cdot \overline{a_{31}} \cdot \overline{b_{31}} \cdot X_{31} + \overline{Binvert} \cdot a_{31} \cdot b_{31} \cdot \overline{X_{31}} \\ &+ Binvert \cdot \overline{a_{31}} \cdot b_{31} \cdot X_{31} + Binvert \cdot a_{31} \cdot \overline{b_{31}} \cdot \overline{X_{31}} \end{aligned}$$

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 X31 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 a31, b31 and X31.

d. Modify the ALU so that the MIPS pseudoinstruction abs could be implemented as an actual instruction.

To implement the ABS operation, we can use the operand as the b input to the ALU, and output either 0+b (if b is non-negative) or 0-b (if b is negative). This can be accomplished with two modifications.

1. Adding a MUX to select the first input to the adder of each bit of the ALU. For the ABS operation, the input will be zero, while for all other operations, it will be the *a* input to that bit of the ALU.

2. Adding a MUX to select the *Bnegate* control signal of each bit of the ALU. For the ABS operation, the signal will be b_{31}, while for all other operations (which is 1 when b is negative, and 0 when it is nonnegative), it will be the *Bnegate* control signal of the ALU.