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

Homework 6 - Solutions
Computer Arithmetic and Performance
Maximum points: 60

Directions
This assignment is due Tuesday, 10th February 2004 by 5:00 PM for Sections 1 and 2.

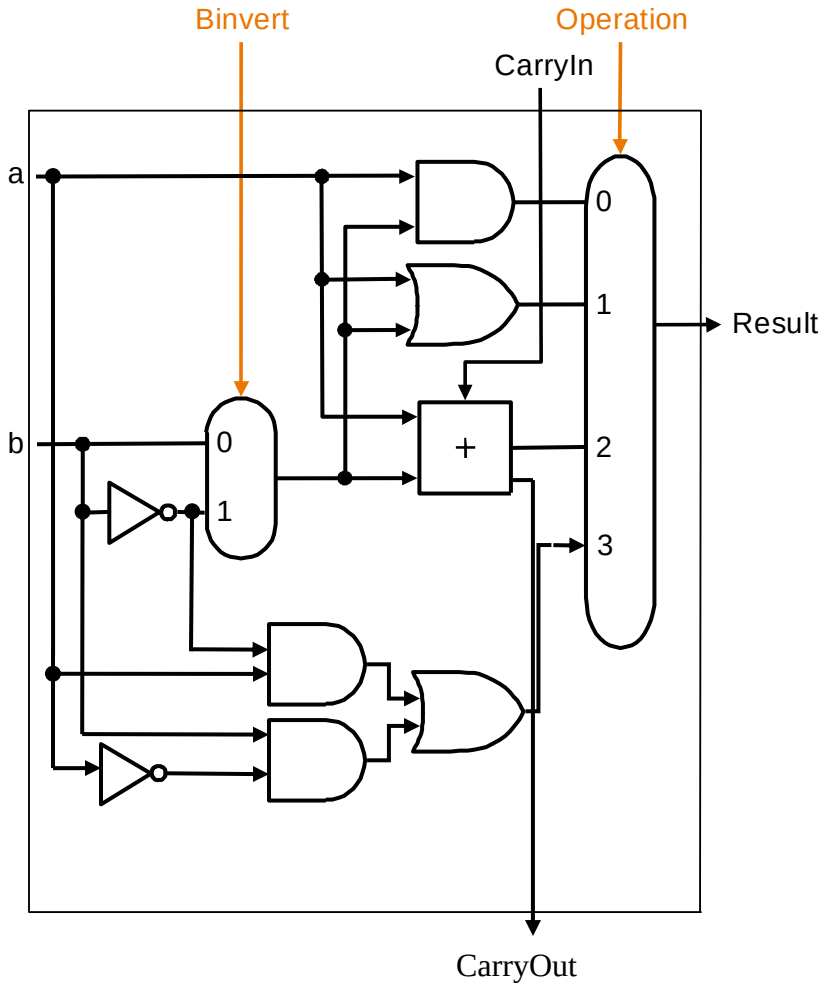
Learning Objectives
In the process of completing this homework assignment, students will develop their abilities to
- Design digital logic circuits for computer arithmetic.
- Predict the qualitative effect on clock cycle time of modifications to the design of digital logic circuits.
- Determine the absolute and relative performance of implementations of instruction set architectures.

General Instructions
1. Submit your solutions on a separate sheet of paper.
2. Remember that the figures in the book are available in electronic form on the book's website and on the class website(Resources).
3. Use only inverters, AND gates, OR gates, and multiplexers for problems 1 to 5.

Problems
1) [5 points] Modify the 32-bit ripple-carry ALU developed in class to support the MIPS XOR instruction. Describe any new or modified control signals.

The modified 1-bit ALU is shown on the next page. 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.



2) [5 points] Modify the 32-bit ripple-carry ALU developed in class 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). *Hint:* There are four cases to consider. One of them is the case in which both inputs are positive and the sum appears to be negative. Look up pages 221 and 222(Figure 4.4) of Hennessy and Patterson 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:

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

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

3) [5 points] Assume that you have a combinational logic unit that detects overflow, as you are required to design for Problem 2. Modify the 32-bit ripple-carry ALU developed in class so that it implements SLT correctly even when overflow occurs.

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}.

4) [5 points] Your design team makes frequent use of the MIPS pseudoinstruction ABS, which determines the absolute value of a value in a source register. You are considering modifying the processor to make it an actual instruction. Show the necessary modifications to the 32-bit ripple-carry ALU developed in class to support the new instruction. Describe any new or modified control signals.

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). b_{31}, therefore has to be a select signal to the MUX that chooses between b and –b. 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 first input will be zero, while for all other operations, it will be the *A* input. A new control signal “Abs” will be needed for this. “Abs” should be set such as to pick “A” as the input for all other instructions and pick “0” as the input for the ABS instruction.

2. Adding a MUX to select between the *Bnegate* control signal and the b_{31} signal as the control signal to the MUX that selects between b and -b. For the ABS operation, the signal selected by the MUX should be b_{31}, while for all other operations, it should be the *Bnegate* control signal of the ALU. The "abs" control signal should be used to control this MUX too.

The MUX introduced in Step 2 increases the delay.

Another possible solution: In this solution, the MUX introduced in Step 1 will not be required. Instead, when the control unit examines the opcode and determines that the instruction is the ABS instruction, it will use the ALUSrcA control signal to pick "0" as the input to the ALU. ALUSrcA may have to expand to 2 bits, to select the third input to the multiplexer. Step 2 remains the same.

5) [5 points] Assume that the ALU is currently on the critical path for your design. Determine whether or not each of your modifications would extend the clock cycle time. State any additional assumptions.
The answer to this question is stated in italics, in the previous pages.

6) [5 points] Problem 2.14 from Patterson and Hennessy.
Some possible combinations:
   a. {CPI, clock rate, # of instructions}
   b. {CPI, cycle time, # of instructions}
   c. {cycle time, # of cycles in a program}
   d. {clock rate, # of cycles in a program}

7) In this problem, you will analyze the performance of two implementations of the MULDER instruction set architecture.

   a. [5 points] For the FOX implementation, the clock rate is 1 GHz, and instructions fall into two categories. Category X instructions require 4 cycles to execute, while category Y instructions require 5 cycles. If the File program executes 1.27×10^{11} category X instructions and 1.01×10^{11} category Y instructions, how long will it take to execute? Express your answer in seconds.

$\text{CPI} = \sum_{i=1}^{N} IC_i * CPI_i$ where IC_i is the # of instructions of type "i" and CPI_i is the average CPI for instructions of type "i" for the architecture.

$$CPI = \frac{1.27x10^{11}x4 + 1.01x10^{11}x5}{1.27x10^{11} + 1.01x10^{11}}$$

$$t_E = \#ofinstructions \times CPI \times \frac{1}{ClockRate} = (1.27 \times 10^{11} + 1.01 \times 10^{11}) \times CPI \times \frac{1}{1 \times 10^9} = 1013 \text{seconds}$$

b. [5 points] The SPOOKY implementation is identical to the FOX implementation, except that the clock rate is 1.2 GHz and category Y instructions require 6 cycles. How long will the File program take to execute on this implementation?

$$CPI = \frac{1.27x10^{11}x4 + 1.01x10^{11}x6}{1.27x10^{11} + 1.01x10^{11}}$$

$$t_E = \#ofinstructions \times CPI \times \frac{1}{ClockRate} = (1.27 \times 10^{11} + 1.01 \times 10^{11}) \times CPI \times \frac{1}{1.2 \times 10^9} = 928 \text{seconds}$$

c. [5 points] The AGENT implementation is identical to the FOX implementation, except that it includes a category Z instruction that requires 8 cycles to execute and does the same thing as a common combination consisting of one category X instruction and one category Y instruction. This combination accounts for 10% of the category X instructions used by the File program on the FOX implementation. How long will the File program take to execute on the AGENT implementation?

# of Z type instructions = 10% x # of X type instructions = .1 x 1.27 x 10^{11} = 1.27 x 10^{10}
Total remaining # of X type instructions when Z type instructions are used
= 1.27 x 10^{11} – 1.27 x 10^{10} = 1.143 x 10^{11}
Total remaining # of Y type instructions when Z type instructions are used
= 1.01 x 10^{11} – 1.27 x 10^{10} = 0.883 x 10^{11}

$$CPI = \frac{1.27x10^{10} \times 8 + .883 \times 10^{11} \times 5 + 1.143 \times 4}{1.27 \times 10^{10} + .883 \times 10^{11} + 1.143 \times 10^{11}}$$

$$t_E = (1.143 \times 10^{11} + 0.883 \times 10^{11} \times 1.27 \times 10^{10}) \times CPI \times \frac{1}{1 \times 10^9} = 1000.3 \text{seconds}$$

d. [5 points] Between the FOX and SPOOKY implementations, which is faster for the File program, and by how much?

Comparing the execution times calculated, the SPOOKY implementation is faster by 85 seconds or 1.09 times as fast or 8.39% (85*100/1013) faster.

8) [10 points] Problem 2.13 from Patterson and Hennessy.

a. C1 on M1

CPI = 4 x .3 + 6 * .5 + 8 x .2 = 5.8

$t_{E\,1}$ = # of instructions x 5.8 x 1/400MHz

C1 on M2

CPI = 2 x .3 + 4 x .5 + 3 x.2 = 3.2

t_{E2} = # of instructions x 3.2 x 1/200MHz

t_{E2}/t_{E1} = 3.2/5.8 * 400/200 = 1.103

The manufacturers of M1 can claim that M1 is 1.103 times as fast as M2.

C2 on M1

CPI = 4 x .3 + 6 x .2 + 8 x .5 = 6.4

C2 on M2

CPI = 2 x .3 + 4 x.2 + 3 x .5 = 2.9

t_{E1}/t_{E2} = 6.4/2.9 * 200/400 = 1.103

With a different compiler, manufacturers of M2 can now claim that M2 is 1.103 times as fast as M1.

b. If we purchase M1, since the number of instructions(for the given program) and the clock rate will be the same, the only parameter that needs to be examined is the CPI. With C1, it is 5.8. With C2, it is 6.4.

With C3, CPI = .5 x 4 + .3 x 6 + .2 x 8 = 5.4

With C3, the lowest CPI is achieved. Therefore, C3 should be the choice of compiler for machine M1.

c. If we purchase M2, since the number of instructions(for the given program) and the clock rate will be the same, the only parameter that needs to be examined is the CPI. With C1, it is 3.2. With C2, it is 2.9.

With C3, CPI = .5 x 2 + .3 x 4 + .2 x 3 = 2.8

With C3, the lowest CPI is achieved. Therefore, C3 should be the choice of compiler for machine M2.

d. We have seen that for both machines C3 is the compiler of choice. Therefore, if we can determine the execution times for both machines with compiler C3, we could then decide which machine would be faster.

t_{E1} = # of instructions x 5.4/400 MHz = .0135 x 10^{-6} x # of instructions
t_{E2} = # of instructions x 2.7/200MHz = .014 x 10^{-6} x # of instructions

Machine M1 has a slightly better execution time and would be the machine to purchase.