CSSE 232 – Computer Architecture I
Rose-Hulman Institute of Technology
Computer Science and Software Engineering Department

Homework 4

This assignment is due 03 November 2004.

1. (7 points) Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 4 ns and a CPI of 2.0 for some program, and computer B has a clock cycle time of 2 ns and a CPI of 1.2 for the same program. Which computer is faster for this program?

   $time_A$ = cycle time of A x times IC x times CPI of A $= 4 \times 2 \times IC = 8 \times IC$ ns

   $time_B$ = cycle time of B x times IC x times CPI of B $= 2 \times 1.2 \times IC = 2.4 \times IC$ ns

   B is faster than A.

2. (8 points) Suppose we could improve the speed of the CPU in a computer by a factor of 5 (without affecting I/O performance) for 5 times the cost. Also assume that the CPU is used 50of the time, and the rest of the time the CPU is waiting for I/O. If the CPU is one-third of the total cost of the computer, is increasing the CPU speed by a factor of 5 a good investment from a cost/performance viewpoint?

   Using Amdahl's rule to determine the overall speedup:

$$\text{Overall Speedup} = \frac{1}{(1-f)+\frac{f}{s}} = \frac{1}{0.5+\frac{0.5}{5}} = 1.667 = 166.7\%$$

   Assuming that the original cost of the computer is x.
   Original CPU cost = $\frac{1}{3}$x
   New CPU cost = 5 $\times\frac{1}{3}$x

   New cost of computer = (5 $\times\frac{1}{3} + \frac{2}{3}$)x = 2.67 $\times$ x

   Overall increase in cost = 267%

3. (Extra credit) (15 points) Problem 4.10 from Hennessy and Patterson.

   Using C1, the average CPI for I1 is (.4 * 2 + .4 * 3 + .2 * 5) = 3, and the average CPI for I2 is (.4 * 1 + .2 * 2 + .4 * 2) = 1.6.

   Thus, with C1, I1 is ((6 10^9 cycles/second)/(3 cycles/instruction))/((3 10^9 cycles/second)/(1.6 cycles/instruction)) = 16/15 times as fast as I2.

   Using C2, the average CPI for I2 is (.4 * 2 + .2 * 3 + .4 * 5) = 3.4, and the average CPI for I2 is (.4 * 1 + .4 * 2 + .2 * 2) = 1.6.

   So with C2, I2 is faster than I1 by factor of ((3 10^9 cycles/second)/(1.6 cycles/instruction))/((6 10^9 cycles/second)/(3.4 cycles/instruction)) = 17/16.

   For the rest of the questions, it will be necessary to have the CPIs of I1 and I2 on programs compiled by C3.

   For I1, C3 produces programs with CPI (.6 * 2 + .15 * 3 + .25 * 5) = 2.9. I2 has CPI (.6 * 1 + .15 * 2 + .25 * 2) = 1.4.

   The best compiler for each machine is the one which produces programs with the lowest average CPI.

   Thus, if you purchased either I1 or I2, you would use C3. Then performance of I1 in comparison to I2 using their optimal compiler (C3) is ((6$\times 10^9$ cycles/second)/(2.9 cycles/instruction))/((3 $\times 10^9$ cycles/second)/(1.4 cycles/instruction)) = 28/29.

   Thus, I2 has better performance and is the one you should purchase.

4. (15 points) Problem 3.23 (page IMD 3.11-5) from your text. Prove that this modified algorithm would cause at most $n/2$ adds, regardless of the multiplier, where n is the number of bits in the numbers being multiplied.

a_{i+1}	a_i	a_{i-1}	Operation	Reason
0	0	0	noop	String of 0's
0	0	1	+mult	Same as 2 bit algorithm
0	1	0	+mult	Subtraction (-mult) followed by a shift and add (+2*mult) is equivalent to +mult
0	1	1	+2*mult	Shift and add gives 2*mult
1	0	0	-2*mult	Shift and sub gives 2*mult
1	0	1	-mult	Add (+mult) followed by a shift and sub (-2*mult) is equivalent to mult
1	1	0	-mult	Same as 2 bit algorithm
1	1	1	noop	String of 1's

010101 3 operations with the 3 bit algorithm as opposed to 6 operations with the 2 bit version.

We do the multiplication 2 bits at a time and a most 1 operation is done each step, therefore at most n/2 operations.

5. (15 points) Use the 2-bit Booth's algorithm to multiply 21 (multiplicand) and 27 (multiplier). Show all of your work and verify that the result is +567.

   21 = 010101 -21 = 101011 27 = 011011

0	000000	011011	0	start
1	101011	011011	0	subtract
	110101	101101	1	shift right
2				noop
	111010	110110	1	shift right
3	001111	110110	1	add
	000111	111011	0	shift right
4	110010	111011	0	subtract
	111001	011101	1	shift right
5				noop
	111100	101110	1	shift right
6	010001	101110	1	add
	001000	110111	1	shift right

   001000 110111 = 567

6. (15 points) Use the modified Booth's algorithm (designed in Problem **??**) to multiply 21 (multiplicand) and 27 (multiplier). Show all of your work and result is +567. Compare the 3-bit Booth's algorithm to the 2-bit.

   21 = 010101 -21 = 101011 27 = 011011
   42 = 00101010 -42 = 11010110

0	00000000	00011011	0	start
1	11101011	00011011	0	-mult
	11111010	11000110	1	shift right 2
2	11100101	11000110	1	-mult
	11111001	01110001	1	shift right 2
3	00100011	01110001	1	+2*mult
	00001000	11011100	0	shift right 2
4				noop
	00000010	00110111	0	shift right 2

   00000010 00110111 = 567

7. (15 points) Use the restoring division algorithm described in class to compute the quotient (Q) and remainder (R) if the dividend (DN) is 130 and the divisor (DR) is 11. For full credit do all of the following:

   - Express DR and DN as 8-bit unsigned numbers.
   - Initialize the 16-bit double register RDN.
   - Show the contents of RDN at each step of the algorithm.
   - On each iteration you should: indicate the bit of RDN under scrutiny; show the action taken, if any, with an explanation; and, shift the contents of RDN.
   - Verify that the result (Q, R) is correct.

   130 = 1000 0010 11 = 0000 1011 -11 = 1111 0101

0	0000 0000	1000 0010	
	0000 0001	0000 0100	shift left
1			sub and restore
	0000 0010	0000 1000	shift left
2			sub and restore
	0000 0100	0001 0000	shift left
3			sub and restore
	0000 1000	0010 0000	shift left
4			sub and restore
	0001 0000	0100 0000	shift left
5	0000 0101	0100 0000	sub
	0000 1010	1000 0001	shift in 1
6			sub and restore
	0001 0101	0000 0010	shift left
7	0000 1010	0000 0010	sub
	0001 0100	0000 0101	shift in 1
8	0000 1001	0000 0101	sub
	0001 0010	0000 1011	shift in 1
9	0000 1001	0000 1011	shift rem right

   remainder: 1001 = 9 quotient: 1011 = 11

8. (10 points) Convert the following decimal numbers to single precision IEEE 754 floating point format:

   (a) $21.625 \Rightarrow 10101.101$
   $(-1)^1 \times (1.01011010) \times 2^4$, exp = 131
   1 1000 0011 0101 1010 0000 0000 0000 000

   (b) $9.25 \Rightarrow 1001.01$
   $(-1)^0 \times (1.00101) \times 2^3$, exp = 130
   0 1000 0010 0010 1000 0000 0000 0000 000

9. (15 points) The following bit patterns are floating point numbers in single precision IEEE 754 format. Convert them to decimal:

   (a) $10011000 \Rightarrow$ exp = 25
   $1.011011 \Rightarrow 1.421875$
   $(-1)^1 \times 1.421875 \times 2^{25} = -47,710,208$

   (b) $01111011 \Rightarrow$ exp = -4
   $1.0011 \Rightarrow 1.1875$
   $(-1)^0 \times 1.1875 \times 2^{-4} = 0.07421875$

   (c) $00000000 \Rightarrow$ exp = -126
   $0.0111 \Rightarrow 0.4375$
   $(-1)^0 \times 0.4375 \times 2^{-126} = 5.1427877 \times 10^{-39}$