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

Homework 4 - Solutions
(Floating point representation, Performance, Recursion and Stacks)
Maximum points: 80 points

Directions
This assignment is due Friday, Feb. 11th. 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:

- Determine the performance of a processor.
- Determine how different parameters affect the performance of a machine
- Apply Amdahl's law to determine how much speedup can be obtained by improving an architecture.
- Interpret and represent floating point numbers
- Write recursive procedures.

Problems
1) 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.

   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.

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

$$t_E = \#ofinstructions \times CPI \times \frac{1}{ClockRate} = (1.27\times10^{11} + 1.01\times10^{11}) \times CPI \times \frac{1}{1\times10^{9}} = 1013 \text{seconds}$$

   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.

2) [8 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 *x* 2 *x IC* = 8 x *IC* ns
time$_B$ = cycle time of B x times IC x times CPI of B
= 2 *x* 1*:*2 *x IC* = 2*:*4 *£ IC* ns
B is faster than A.

3) [9 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 50% of 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:

$$\text{Overall Speedup} = \frac{1}{(1-f)+\frac{f}{s}}$$

$$= \frac{1}{(1-0.5)+\frac{0.5}{5}} = 1.667 = 166.7\%$$

Assuming that the original cost of the computer is x.

Original CPU cost = 1/3x
New CPU cost = 5 * *1/3x*
New cost of computer = (5 * *1/3 + ½)x = 2.67 x*
Overall increase in cost = 267%

4) [8 points] The incomplete MIPS procedure below computes the recursive function

$$foo(a,b) = \begin{cases} b, \text{ if } a = 0 \\ foo(a-1,b-a) * b, \text{ otherwise} \end{cases}$$

Finish the procedure by adding the code for the procedure entrance, the recursive procedure call, and the procedure exit. Do not modify any of the given code, and be sure to follow the MIPS convention for register usage.
Hint: $t0 and $s0 hold the local values of *a* and *b*, respectively.

You may write the answer to this question in the boxes provided and attach the sheets with the rest of your solutions.

```
foo:# Procedure entrance
       addi $sp, $sp, -4     # Since procedure may call another
       sw   $ra, 0($sp)      # procedure, save $ra

       move $t0, $a0         # $t0 = a

       addi $sp, $sp, -4     # Since $s0 is a saved register,
       sw   $s0, 0($sp)      # before we over-write it, must
                             # save its value.

       move $s0, $a1         # $s0 = b
                             # We use $s0 to hold b, because we
                             # need the value of "b" after the
                             # call to "foo".
                             # i.e. foo(a-1,b-a) * b

      # if (a == 0) return b
      bne  $t0, $zero, Else
      move $t1, $s0
      j    Exit

      # otherwise, return foo(a-1,b-a)*b
Else: sub  $t2, $t0, 1
```

```
        sub  $t3, $s0, $t0
        # Recursive procedure call

        move $a0, $t2           //Move arguments to argument
                                //registers
        move $a1, $t3

        jal foo                 //Call the procedure

        move $t1, $v0




        mul  $t1, $t1, $s0

Exit:   move $v0, $t1
        #Procedure exit
        lw $s0, 0($sp)
        lw  $ra, 4($sp)
        addi $sp, $sp, 8 //Restore $ra and $s0

        jr $ra           //Quit procedure
```

5) [10 pts] Some computer architectures do not provide instructions for multiplication. The following Java code describes one (not particularly efficient) algorithm that could be used to perform multiplication within such an instruction set:

```
public static int myMultiply( int a, int b ) {

      int rv;

      if ( a == 0 ) {
            rv = 0;
      } else if ( a > 0 ) {
            rv = myMultiply( a - 1, b ) + b;
      } else {
            rv = - myMultiply( -a, b );
      }

      return rv;
}
```

Write a MIPS program that implements this algorithm.

```
myMultiply:
            addi $sp, $sp, -8            #Save $ra to the stack
            sw   $ra, 4($sp)
            sw   $s0, 0($sp)       #Save $s0 to the stack

            move $t0, $a0                #$t0 = a
            move $s0, $a1                #$s0 = b
```

```
            #If (a==0) return rv = 0;
            bne   $t0, $0, Greater
            move  $v0, $0              # return "0"
            j     Exit         # It is not a good programming practice
                               # to have
                               # multiple exit points from a program.
    Greater:
            #if(a > 0), return myMultiply( a - 1, b ) + b
            slt      $t2, $0, $t0
            beq   $t2, $0, Less
            addi    $t2, $t0, -1          # $t2 = a-1

            move  $a0, $t2          # Move arguments to argument
                                    # registers
            move  $a1, $s0

            jal   myMultiply        # call procedure

            add   $v0, $v0, $s0     # $v0 = myMultiply (,) + b
            j     Exit

            # return   - myMultiply( -a, b )
     Less:
            sub   $t2, $0, $t0            #$t2 = -a

            move  $a0, $t2
            move  $a1, $s0

            jal myMultiply          # call procedure

            sub   $v0, $0, $v0

add   $t2, $0, $t1                  #$t2 = -a

            move  $a0, $t2
            move  $a1, $s0

            jal myMultiply          # call procedure

            sub   $v0, $0, $v0

     Exit:
            lw    $ra, 4($sp)       # Restore $s0 and $ra
            lw    $s0, 0($sp)
            addi  $sp, $sp, 8

            jr    $ra               # Exit procedure
```

6) [10 points] Convert the following decimal numbers to single precision IEEE 754 floating point format:
   (a) -21.625
   (b) 9.25

(a) (21:625)=> 10101.101

$(-1)^1 \times (1.01011010) \times 2^4$, exp = 131

1 1000 0011 0101 1010 0000 0000 0000 000

(b) 9.25 => 1001.01

$(-1)^0 \times (1.00101) \times 2^3$, exp = 130

0 1000 0010 0010 1000 0000 0000 0000 000

7) [15 points] The following bit patterns are floating point numbers in single precision IEEE 754 format. Convert them to decimal:

(a) 10011000 => exp = 25

1.011011 => 1.421875

$(-1)^1 \times 1.421875 \times 2^{25} = -47, 710, 208$

(b) 01111011 => exp = -4

1.0011 => 1.1875

$(-1)^0 \times 1.1875 \times 2^{-4} = 0.07421875$

(c) 00000000 => exp = -126

0.0111 => 0.4375

$(-1)^0 \times 0.4375 \times 2^{-126} = 5.1427877 \times 10^{-39}$