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

Homework 6
(Stacks and Recursion)

This assignment is due Tuesday, November 11, 2003 for Sections 1, 2 and 3.

Learning Objectives

- In the process of completing this homework assignment, students will develop their abilities to
- Use stacks to implement procedure calls in assembly language.
- Implement recursive procedures in an assembly language.

General Instructions

Submit your solutions on a separate sheet of paper.

Problems

1. [7 points] You are provided with a MIPS program that uses two procedures: `power` and `multiply` to determine x^y. (Yes, this is the program from Exam 1). Rewrite the parts of the program written in *red italics* so that the program preserves registers using the stack instead of static memory locations.

```
            .text
            .globl main
            .globl multiply
            .globl product

main:                                   # Determines 2^5
      li    $s0, 2
      li    $s1, 5

      add   $a0, $s0, $zero             # Move the parameters to registers
      add   $a1, $s1, $zero             # $a0-$a1 before procedure call
      jal   power
      add   $s2, $v0, $zero             # Move returned value from $v0

      addi  $v0, $zero, 10              # Getting ready to exit program
      syscall

power :                                 # Procedure to determine x^y
      la    $t0, SAVERA                 # Save the return address to memory
      sw    $ra, 0($t0)

      add   $t0, $a0, $zero             # x
      add   $t1, $a1, $zero             # y

      addi  $t2, $zero, 1               # count
      addi  $t3, $zero, 1               # value

      la    $t4, SAVET0               # Save values in the temp registers
      sw    $t0, 0($t4)               # that will be used after the
                                      # procedure call
```

```
        la    $t4, SAVET1
        sw    $t1, 0($t4)
loop1 :
        bgt   $t2, $t1, exit1         #if (count > y) done with loop

        la    $t4, SAVET2             # Save $t2 into memory before the
        sw    $t2, 0($t4)             # procedure call

        add   $a0, $t3, $zero
        add   $a1, $t0, $zero
        jal   multiply
        add   $t3, $v0, $zero         # Move to $t3, the new "value"


        la    $t4, SAVET0   # Restore values in the temp registers
        lw    $t0, 0($t4)   # that were saved before the procedure call

        la    $t4, SAVET1
        lw    $t1, 0($t4)

        la    $t4, SAVET2
        lw    $t2, 0($t4)

        addi  $t2, $t2, 1       # count = count + 1
        j     loop1

exit1 :
        add   $v0, $t3, $zero   # Move "value" to $v0

        la    $t0, SAVERA       # Restore return address to $ra
        lw    $ra, 0($t0)
        jr    $ra


multiply :                      # Procedure to determine a x b
        add   $t0, $a0, $zero   # a
        add   $t1, $a1, $zero   # b
        li    $t2, 1            # count
        li    $t3, 0            # product
loop2  :
        bgt   $t2, $t1, exit2   #if(count > b) done with loop
        add   $t3, $t3, $t0     # product = product + a
        addi  $t2, $t2, 1       # count = count + 1
        j loop2
exit2 :
        add   $v0, $t3, $zero # Move "product" to $v0
        jr    $ra

        .data
SAVERA:           .word 0
SAVET0:           .word 0
SAVET1:           .word 0
SAVET2:           .word 0
```

2. [10 points] 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.

3. [8 points] The incomplete MIPS procedure on the following page computes the following recursive function:

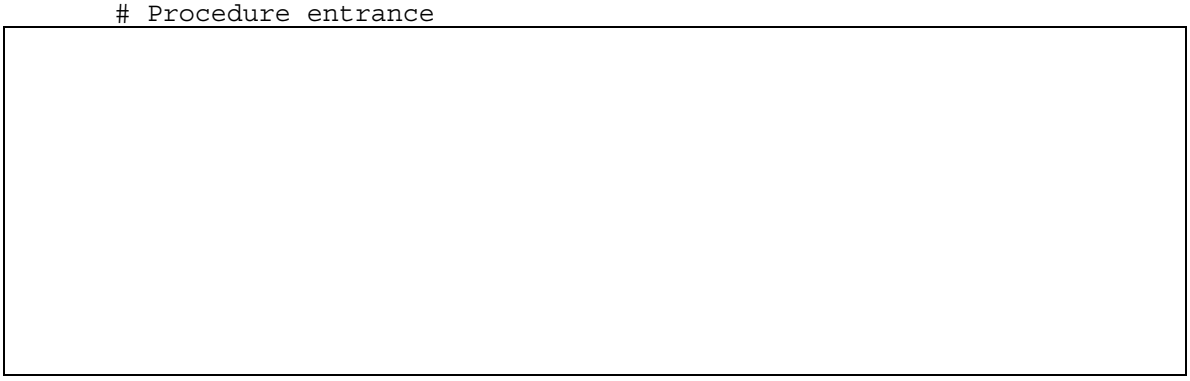
	{	*b*, if *a = 0*
foo(a,b) =		
		*foo(a-1,b-a) * b*, otherwise

   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.

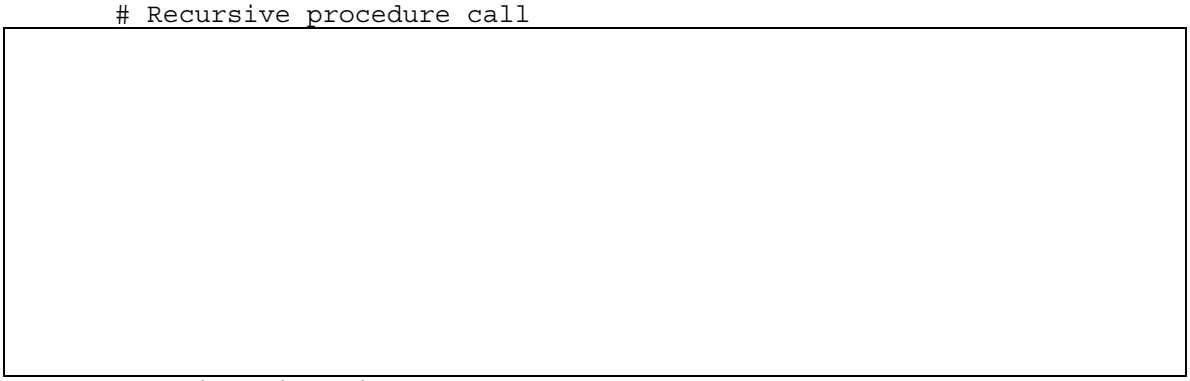
   You may write the answer to this question in the boxes provided and attach the sheet with the rest of your solutions.

```
foo:
      # Procedure entrance
```



```
      # 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
```



```
      mul  $t1, $t1, $s0

Exit: move $v0, $t1
      #Procedure exit
```