Fall 2022 Homework Assignment #4

(60 points)

This is individual assignment. Every student must work on it alone. Group work is not permitted. All students who work in a group on this assignment will receive an automatic 0 for this assignment. In additional, such incidents will be reported as academic integrity violation. **All solutions should be typed.** No hand-written submissions will be accepted. Please submit your solutions to question 1 and 2 as scripts that can be evaluated in QtSpim (with txt extension).

• Email to the address in the header (MS Word or PDF). Please do not submit in Canvas.

Due date: November 5th, 11:59 PM.

1. Implementing Recursion using a loop (20 points)

Implement in MIPS the following recursive (reentrant) function by using a loop (not as a non-leaf function):

```
int f(int n)
{
 If n == 0 return 0;
 If n == 1 return 1;
 return f(n-1) + f(n-2);
}
```

Suggestion: calculate the function's output for n = 0, 1, 2, 3, 4 manually, and apply the same procedure in your loop n-2 times, starting from element 0 and element 1 calculated before the loop. For example, f(2) =f(1) +f(0) =1, f(3) = f(2) +f(1) = 2 etc.

2. Writing MIPS code with loops and function calls (20 points)

Write a MIPS program that takes the ith element from an array A and calls the factorial function with n=A[i], putting that memory load and that function call in a loop going from i=0 to i=g, and placing the factorial result into msgresults[2i] for printing.
Your program must check for the array-out-of-bound conditions.

Use the following as a template for your solution:

```
.data

A: .word 0 1 2 3 4 5 6 7 8 9
i: .word 0 # start index in A
```

```
g: .word 4 # how many indices in A

msgannounce:        .asciiz  " the results for  factorials are: "
msgresults:         .asciiz  "                "       # at least 20 spaces
nLine:  .asciiz "\n"

.text
.globl main

main:

# load registers for array A
la      $s0, A                  # load address of A
la      $s1, msgresults         # load address of msgresults
lw      $s2, i                  # load word i into s2
lw      $s4, g                  # load word g into s3

# initialize address of A[i]

# initialize address of msgresults[2i]

# place your instructions here

# iteratively grab successive elements of A
Loop:
        # loop through A, check index bounds
# place your instructions here

        # load A[i]
# place your instructions here

        # now call the fact function with the argument = A[i]
# place your instructions here

        # store v0+48 into msgresults[2i]
# place your instructions here

        # update the index
# place your instructions here

        # update the address of A[i]
# place your instructions here

        # update the address of msgresults[2i]
# place your instructions here

        # loop again
        j Loop

end:
jal fact_print


ori $v0, $zero, 10              #exit
syscall

fact:
#The first time fact is called, sw saves an address in the program that called it
        addi $sp, $sp, -8 # adjust stack for 2 items
        sw $ra, 4($sp) # save the return address
        sw $a0, 0($sp) # save the argument n
        slti $t0,$a0,1 # test for n < 1
        beq $t0,$zero,L1 # if n >= 1, go to L1
# n LESS THAN 1  ------> every n will reach this point
        addi $v0,$zero,1 # return 1
        addi $sp,$sp,8 # pop 2 items off stack
        jr $ra # return to caller

L1:     addi $a0, $a0, -1 # n >= 1: argument gets (n – 1)
        jal fact # call fact with (n –1)
        lw $a0, 0($sp) # return from jal: restore argument n
        lw $ra, 4($sp) # restore the return address
        addi $sp, $sp, 8 # adjust stack pointer to pop 2 items
# We assume a multiply instruction is available, even though it is not covered until Ch 3
        mul $v0,$a0,$v0 # return n * fact (n – 1)
#fact jumps again to the return address:
        jr $ra # return to the caller


fact_print:
        li $v0, 4
        la $a0, msgannounce     # argument: string
        syscall         # print the string

        la $a0, nLine  # argument: new line string
        syscall         # print the string

        # Print the string with results
        la $a0, msgresults     # argument: string
        syscall

        la $a0, nLine          # argument: new line string
        syscall        # print the string

        jr $ra
```

Bonus: Adding v0+48 allows for the printing but it is not generalizable for g>4. Why? (5 points)

3. Tracking the stack for function calls. (20 points)
Run the script nonleaf1.txt in QtSpim (script is on Canvas) in a step-by-step fashion. As you are running, keep track of all the values that are being pushed to stack and popped off the stack in the exact order as in the script. Put it together in a report as an ordered list:

Push	**Pop**
$ra: 400030	**$a0:** 1
$a0: 3	**$ra:** 400068
…	**…**

Write the instructions corresponding to the addresses that are in $ra at the moment of each push and pop (sw/lw).