EE 2310 Homework #7 Solutions – Complex SPIM Programs

For the problems given, develop MIPS programs that satisfy the specifications in the problem statement. Remember: On the homework due, email your NotePad files of the programs as attachments to the TA. Note: these are challenging programming problems, and deserve careful consideration and focus.

1. **Develop a program that arranges the series of decimal numbers shown in ascending order from smallest to largest. Use the space at the right for your program. Note the use of the ".word" directive to define multiple pieces of data with one data declaration.**

   When the numbers are ordered correctly, print them out as a single string from memory. Hint: You will need counters to determine when the compares are complete, and when you have printed out all the numbers. But note: The compare counter should only count eight (8), as you only do 8 compares. The print counter, however, should count nine (9), as you are printing 9 numbers!

Comments: This program is a good example of the use of recursion to "keep one's place" in the string of numbers, when a given number must be "backed up" and inserted into its rightful place. Using recursive code, the "jr" section, or "procedure uncall," gets the sort back to the next number in the sequence to be compared and placed. This program is very similar in concept to the alphabetization program from Lecture 17.

```
#                          Order Program
#  Arranges numbers in ascending numerical order.

          .text
main:     li $t8,8          # set up counter
          la $t0,nums       # load starting address
          move $t7,$t0      # keep add. for later compare
next:     beqz $t8,end      # have we done ten compare sequences?
          lw $t1,0($t0)     # \
          lw $t2,4($t0)     # / get words
          ble $t1,$t2,cnt   # are current numbers in correct order?

comp:     jal rev
          j on
rev:      sub $sp,$sp,4     # \
          sw $ra,0($sp)     # / store $ra on stack
          lw $t1,0($t0)     # \
          lw $t2,4($t0)     # / get words
          ble $t1,$t2,back   #  no's now in order, move back down list
          sw $t1,4($t0)     # \
          sw $t2,0($t0)     # no's are in wrong order; reverse them
go:       beq $t0,$t7,back  # are we at the top of the list?
          sub $t0,$t0,4     # no, therefore get next two words to
                            # compare on "backwards move"
          jal rev           # going back
back:     lw $ra,0($sp)     # compare is at position 0, so we
                            # are done with one pass; go back to last
                            # position and start new compare seq.
          addi $sp,$sp,4    # getting next $ra
          addi $t0,$t0,4    # moving back for next compare sequence
          jr $ra            # are we back to pos. for next compare?
cnt:      addi $t0,$t0,4    # go to start of next compare sequence
on:       sub $t8,$t8,1     # finished one compare sequence, so we
          j next            # start next one

end:      la $t0,nums       # Print out results and end program
          li $t8,9
          li $v0,1
nxwd:     lw $a0,0($t0)
          syscall
          sub $t8,$t8,1
          beqz $t8,stop
          addi $t0,$t0,4
          j nxwd

stop:     li $v0,10
          syscall           # we are done; QUIT!

          .data
nums:     .word 5,3,8,9,2,4,1,7,6
```

2. **A factorial function can be defined as:**

$$n! = n \cdot (n-1)!, \text{ where } 0!=1.$$

Then n! can be calculated as follows: A recursive procedure call (via jal) counts down from n to 1 to determine the number of loops necessary to calculate the n! result. Then, the jr ("procedure uncall") is used to produce a number of loops equal to the number of loops in the countdown to do the actual math. Using the definitions listed above for n! and 0!, construct a program that uses recursion to calculate n! in the manner suggested above.

The number will be input from the console. Note that the number must be 12 or less, since we are not doing floating-point calculations.

Comments: The factorial number "n" is loaded into a register, then a series of jal's (storing $ra on the stack each time), calls a loop that does a "countdown," subtracting 1 from n each loop. When the register = 0, 1 is loaded into the register (since 0!=1), and then the factorial loop starts. This is the "jr" loop. Each loop, the contents of $ra are popped from the stack, and the factorial of the current number is calculated. The program will do as many jr's as it did jal's. In each jr loop, $t1 is multiplied by $t0 ($t0 has 1 in it to start and simply acts as the "running factorial total"). In each jr loop, $t1 is increased by 1 (it will equal n on the last loop). After n multiplies, the loop ends (since there are n jr loops, as there were n jal's) and the program returns to the instruction after the initial jal, after which the value of n! is printed out and the program stops. Note that "winding up" the factorial loop simply consists of doing n jal's, so that n jr loops will calculate n! = n*(n-1)*(n-2)*…*0! Remember that 0!=1.

```
#                N! Program
# Calculates n! for any integer of 12 or less
# (due to use of only fixed-point calculations)

          .text
main:     la $a0,input
          li $v0,4
          syscall
          li $v0,5
          syscall
          li $t0,1
          move $t6,$v0
          move $t1,$v0
          jal cntdn

          move $a0,$t6
          li $v0,1
          syscall
          la $a0,ans
          li $v0,4
          syscall
          move $a0,$t0
          li $v0,1
          syscall
          li $v0,10
          syscall

cntdn:    sub $sp, $sp,4
          sw $ra,0($sp)
          sub $t1,$t1,1
          bgtz $t1, ret
          li $t1,1
          j fact

ret:      jal cntdn
fact:     mul $t0,$t0,$t1
          addi $t1,$t1,1
          lw $ra,0($sp)
          addi $sp,$sp,4
          jr $ra

          .data
input:    .asciiz "Input integer (0-12): "
ans:      .asciiz " factorial is "
```

3. **In the space provided, write a program to count the number of consonants in the phrase "hello, world!\n" (as shown in the data statement).**

   Note that the "h" in the phrase is purposely left as a small letter in order to make your loop program a little easier to write (you don't have to worry about capital letters).

   When the program has calculated the number of consonants, use the answer header provided and then output the correct number. Hint: The total number of letters minus the vowels equals the number of consonants.

Comments: This program is much simpler than the first two recursive loop programs. In this case, only the numbers of vowels and letters must be counted as the analysis loop repeats. Then letters minus vowels = consonants. The exit condition is also easy, as the loop simply tests for 0, as the phrase is null-terminated.

```
        .text
main:   la $t0,str
look:   lb $t1,($t0)
        beqz $t1,done
        blt $t1,0x61,next
        bgt $t1,0x7a,next
        addi $t4,$t4,1
        beq $t1,0x61,count
        beq $t1,0x65,count
        beq $t1,0x69,count
        beq $t1,0x6f,count
        beq $t1,0x75,count

next:   addi $t0,$t0,1
        j look
count:  addi $t2,$t2,1
        j next

done:   sub $t5,$t4,$t2
        li $v0,4
        la $a0,ans
        syscall
        move $a0,$t5
        li $v0,1
        syscall
        li $v0,10
        syscall

        .data
str:    .asciiz "hello, world!\n"
ans:    .asciiz "The number of consonants is "
```