Machine Organization and Assembly Language Programming

Exercise : Familiarization with SPIM

The purpose of this homework is to familiarize you with the R3000 Simulator, *xspim*. You will learn how to use *xspim* to run assembly language programs and to extract information to debug your programs. You will learn assembly operations as you read Appendix A and Chapter 3. (Recall that you need to start reading the Appendix.) Don't worry if you cannot understand all of it now. The purpose of this assignment is simply for you to become familiar with using *xspim*.

For this assignment we will use the program on page 3. This program computes the sum of the first 3 powers of 10. The line numbers on the left are for reference only and are not part of the program.

You do not have to type this program in yourself. The program already resides in:
/cse/courses/cse378/CurrentQtr/ProblemSets/prob0/prob0.s
on orcas/sanjuan. Simply **copy this file** in your own directory and use it.

I. Run the simulator in single steps.
What is printed on the spim console?

Do you notice any discrepancies between the given program and the instructions executed?
What are the discrepancies

II. Complete the following table by examining various registers and memory location contents after executing the appropriate instructions. Try and use the breakpoint command.

Location	Line 16	Line 23 first iteration	Line 23 second iteration	Line 39
$25				
$24				
$8				
8($sp)				
4($sp)				

Below is the program, written in C. It is also in:
/cse/courses/cse378/CurrentQtr/ProblemSets/prob0/prob0.c on orcas/sanjuan.

```
#include <stdio.h>

main ()
{
int i;
int sum = 1;
int power = 1;

for (i = 1; i <= 2; i = i + 1){
   power = power * 10;
           sum = sum + power;
 }
printf ("The sum of the first 3 powers of 10 is %d\n", sum);
}
```

Take a look at the assembly code (prob0.s) and the C-code (prob0.c) to begin to see the relationship and differences between operations in a high level language and operations in assembly language.

```
.data                                          # tells assembler that this is data
str: .asciiz "The sum of the first 3 powers of 10 is: "
newline: .asciiz "\n"

.text                                          # Tells assembler that the following is code
.globl main                                    # Entry into our code must be global

main:
subu $sp, 16                                   # Allocate space for storage on stack (4 words)
sw $31, 0($sp)                                 # Save return address (to caller of main)
sw $4, 12($sp)                                 # Save R4 (not necessary.)
li $14, 1                                      # Put constant 1 in temporary register 14
sw $14, 4($sp)                                 # Place 1's in our two words of
sw $14, 8($sp)                                 # temporary storage (sum & power)
li $16, 10                                     # Put constant 10 in register 16
li $8, 1                                       # Iteration counter
loop:
lw $24, 8($sp)                                 # Load value of power (1st time it's 1)
lw $25, 4($sp)                                 # Load value of sum (1st time it's 1)
mul $24, $24, $16                              # Compute next power of 10
sw $24, 8($sp)                                 # Store the power
addu $25, $25, $24                             # Add it to the sum
sw $25, 4($sp)                                 # Store the sum
addu $8, $8, 1                                 # Increment the counter
ble $8, 2, loop                                # If counter != 2¸, repeat
                                               # Sum now contains the sum of powers of 10
li $2, 4                                       # Use syscall 4 (print_string)
la $4, str                                     # to display string str
syscall

li $2, 1                                       # Use syscall 1 (print_int)
lw $4, 4($sp)                                  # to print the sum
syscall

li $2, 4
la $4, newline
syscall

lw $31, 0($sp)                                 # Get return address
addu $sp, 16                                   # Restore stack
j $31                                          # Return to caller

.end main
```