Machine Organization and Assembly Language Programming

Exercise: Familiarization with SPIM

The purpose of this homework is to familiarize you with the R3000 Simulator, \textit{xspim}. You will learn how to use \textit{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 \textit{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:
\texttt{/cse/courses/cse378/CurrentQtr/ProblemSets/prob0/prob0.s}
on orcas/sanjuan. Simply \textbf{copy this file} in your own directory and use it.

I. Run the simulator in single steps.

What is printed on the \textit{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.

<table>
<thead>
<tr>
<th>Location</th>
<th>Line 16</th>
<th>Line 23 first iteration</th>
<th>Line 23 second iteration</th>
<th>Line 39</th>
</tr>
</thead>
<tbody>
<tr>
<td>$25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$24</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$8</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$8($sp)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4($sp)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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.
1. .data
2. str: .asciiz "The sum of the first 3 powers of 10 is: 
3. newline: .asciiz "\n"
4. 
5. .text
6. .globl main
7. 
8. main:
9. subu $sp, 16
10. sw $31, 0($sp)
11. sw $4, 12($sp)
12. li $14, 1
13. sw $14, 4($sp)
14. sw $14, 8($sp)
15. li $16, 10
16. li $8, 1
17. loop:
18. lw $24, 8($sp)
19. lw $25, 4($sp)
20. mul $24, $24, $16
21. sw $24, 8($sp)
22. addu $25, $25, $24
23. sw $25, 4($sp)
24. addu $8, $8, 1
25. ble $8, 2, loop
26. 
27. li $2, 4
28. la $4, str
29. syscall
30. 
31. li $2, 1
32. lw $4, 4($sp)
33. syscall
34. 
35. li $2, 4
36. la $4, newline
37. syscall
38. 
39. lw $31, 0($sp)
40. addu $sp, 16
41. j $31
42. 
43. .end main
44. 
# tells assembler that this is data
# Tells assembler that the following is code
# Entry into our code must be global
# Allocate space for storage on stack (4 words)
# Save return address (to caller of main)
# Save R4 (not necessary.)
# Put constant 1 in temporary register 14
# Place 1’s in our two words of temporary storage (sum & power)
# Put constant 10 in register 16
# Iteration counter
# Load value of power (1st time it’s 1)
# Load value of sum (1st time it’s 1)
# Compute next power of 10
# Store the power
# Add it to the sum
# Store the sum
# Increment the counter
# If counter != 2, repeat
# Sum now contains the sum of powers of 10
# Use syscall 4 (print, string)
# to display string str
# Use syscall 1 (print, int)
# to print the sum
# Get return address
# Restore stack
# Return to caller