Lab Exercise #9 -- SPARC Control Constructs

This exercise develops familiarity with selective and repetitive control constructs in SPARC assembly language.

Recall that the SPARC has 32 general-purpose registers (%r0 - %r31) which are visible to a given function.

Registers %r14, %r15, %r30 and %r31 are used to manage the run-time stack of activation records and their contents must not be modified by a function.

Registers %r0 through %r7 are shared between all functions in a program. Since they are global, a function should not assume that their contents are the same before and after a call to another function.

Registers %r8 through %r13 are used to pass arguments to functions; %r8 is used to hold the value returned by a function.

Registers %r16 through %r23 are local to a given function. Thus, that function can assume that their contents are the same before and after a call to another function.

Those 32 registers are often viewed as four groups of 8 registers:

```
  global registers  %g0 - %g7  (same as %r0 - %r7)
  out registers     %o0 - %o7  (same as %r8 - %r15)
  local registers   %l0 - %l7  (same as %r16 - %r23)
  in registers      %i0 - %i7  (same as %r24 - %r31)
```

Assembly language functions which invoke library functions must follow the conventions outlined above.

A. Examine the SPARC assembly language program which is contained in the file named "~cse320/Labs/lab09.program.s" and answer the questions below.

1. Complete the following table to show the contents of the l ("local") registers after the program executes. Give the contents of each register as a decimal number and as a hexadecimal number. If the program does not place a value into a given local register, write "unknown" as your answer.

	Decimal Value	Hexadecimal Value
%l0:	_____	_____
%l1:	_____	_____
%l2:	_____	_____
%l3:	_____	_____
%l4:	_____	_____
%l5:	_____	_____
%l6:	_____	_____
%l7:	_____	_____

2. The output produced by the program is shown below. Circle and label the memory area corresponding to the four integer variables ("W", "X", etc).

```
MEMORY CONTENTS [00021a64 to 00021a93]
  00021a64: 0a 4c 61 62 20 45 78 65 72 63 69 73 65 20 23 39
  00021a74: 0a 00 00 00 00 00 00 37 00 00 00 48 00 00 00 34
  00021a84: 00 00 00 4e 00 00 00 00 ff ff ff ff 00 00 00 00
```

3. Based on the output produced by the program, complete the following table to show the address and contents of each of the four integer variables. Give your answers as eight-digit hexadecimal numbers (including leading zeroes).

	address of variable	contents of variable
W:	_____	_____
X:	_____	_____
Y:	_____	_____
Z:	_____	_____

4. Copy the program into your account, then replace the indicated "nop" with a call to "iu_window" to verify your answers. After revising the program, translate and execute the program using UNIX commands such as:

```
<prompt> gcc your_file.s ~cse320/lib/reglib.o
<prompt> a.out
```

Check your answers against the values displayed by the "iu_window" function.

B. Design and implement the SPARC assembly language program specified below.

1. The program will process all of the characters in the standard input stream. It will compute and display:

a) the number of times that the character ’A’ appears in the input stream.

b) the number of times that a character which is classified as a decimal digit (any character in the range {0-9}) appears in a given text file.

You may wish to copy "lab09.start.s" and edit the resulting file to complete this exercise. Please note that the statements labeled "Block #3" in the program for Part A illustrates one method for determining whether or not a value is within a specified range.

2. Test your program using any arbitrary text file. For example, if your data file (containing ASCII characters) is named "lab09.data", the following command would be used to load and execute your program:

```
<prompt> a.out < lab09.data
```

You may also input text directly from the keyboard by executing the program without input redirection:

```
<prompt> a.out
```

An end-of-file is simulated by a control-d at the beginning of a line.