Lab Exercise #9 -- ARM Data Movement and Organization

A. Consider the following SPARC assembly language program.

1. In the spaces provided, give the hexadecimal contents (each byte) of the requested memory locations and registers after the instructions are executed.

```
.global main
.data
.balign 4
nums:
.byte  0x93, 0x2e
.short 0x3a67, 0xba98, 0x7654
store:
.word  0xffffffff, 0xeefeeddd, 0xd5432109
.text
.balign 4
main:
push   {lr}
ldr      r0, =store
ldr      r1, =0x12345678
str      r1, [r0, #0]  
strh     r1, [r0, #4]  
strb     r1, [r0, #8]  

@  store+0:  ______  ______  ______  ______
@  store+4:  ______  ______  ______  ______
@  store+8:  ______  ______  ______  ______

ldr     r0, =nums
ldr      r1, [r0, #4]
ldrsh   r2, [r0, #4]
ldrh    r3, [r0, #4]
ldrsb   r4, [r0, #4]
ldrb    r5, [r0, #4]

@  r1:  ______  ______  ______  ______
@  r2:  ______  ______  ______  ______
@  r3:  ______  ______  ______  ______
@  r4:  ______  ______  ______  ______
@  r5:  ______  ______  ______  ______

halt:
pop      {lr}
bx       lr
```

==========================================================================================================
2. Copy the source code into your account, then execute the program to verify your answers.

Use the following commands to translate, link and execute the program:

```
<prompt> gcc lab09.partA.s
<prompt> gdb a.out
```

If necessary, review the "gdb" commands which could be used to run the program and examine the contents of the registers. For example:

```
(gdb) break halt       -- set breakpoint at "label" halt
(gdb) p/x $r0          -- print the contents of R0 in hex
```

The contents of memory locations can also be examined using "gdb". For example:

```
(gdb) p/x &store       -- print the address of label "store" in hex
(gdb) p/x *0x20500      -- print the contents of address 0x20500 in hex
```

However, the fact that the ARM processor is a little-endian machine means that it is sometimes difficult to visualize multi-byte values in memory.

Insert the following lines immediately after the label "halt":

```
ldr     r0, =nums
mov     r1, #20
bl      display1

ldr     r0, =nums
mov     r1, #10
bl      display2

ldr     r0, =nums
mov     r1, #5
bl      display4

ldr     r0, =nums
mov     r1, #20
mov     r2, #20
bl      display
```

Then, translate and execute the program using:

```
<prompt> gcc lab09.partA.s /user/cse320/lib/memlib.o
<prompt> a.out
```

The four calls to the "memlib" functions display the 20 bytes which begin at label "nums" in four different formats. In particular, compare the output produced from function "display4" and function "display".

If any of your answers are incorrect, re-do your work to make sure that you understand the operations.
B. Working with arrays of characters.

The source file named "cse320/Labs/lab09.partB.s" contains a program to read a sequence of characters from standard input and store the characters in an array. The program uses the C library functions "getchar" and "printf" to handle communication with the user.

1. Copy the source file into your account and examine the program. Assemble and execute the program, then answer each of the following questions.

a) Why is space for the formatting strings ("fmt1" and "fmt2") reserved in the "text" section, while space for the character sequences ("original" and "copy") is reserved in the "data" section?

b) The loop to read characters from standard input may terminate under two different circumstances. What are those two circumstances, and why are both comparisons necessary?

2. Modify the program as per the comments in the source code.