Lab 3: Loops; Hex

Due Tuesday, 26 September, in class

1. **(Hexadecimal.)** The hexadecimal, or base-16, number system is a positional system in which each position represents a power of 16. There are 16 different symbols used: 0–9 (with their usual meanings) and the letters A–F (where $A = 10$, $B = 20$, \ldots, $F = 15$). Thus, the number $2e7_{16}$ is $2 \times 16^2 + 14 \times 16^1 + 7 \times 16^0 = 512 + 224 + 7 = 743_{10}$.

Each hexadecimal symbol represents four binary digits, or bits. Thus, $0_{16} = 0000_2$, $1_{16} = 0001_2$, \ldots, $F_{16} = 1111_2$. The contents of a 32-bit word in memory can thus be represented by a hexadecimal string of length 8. Addresses and data values in the MARS simulation environment are, by default, displayed using this representation. With a little bit of practice, you can easily read off the bit pattern of a word in memory from its hexadecimal representation.

2. **(A Few More MIPS Instructions.)** In this assignment you will be using various branch statements to create a loop for converting hexadecimal strings into decimal values. One of the things you will need to do is to read in a hexadecimal string of length 8.

As with all input/output in the simulator, inputting a string is done using a **syscall**. The code for **read_string** is 8. The first argument register, $a0$, must be loaded with the address of the first character in the string. The second argument register, $a1$, must be loaded with an integer containing the length of the string to be read in plus one (for the null character), i.e., the number of characters to be read in plus 1. (So to read in eight characters, you would reserve 9 bytes and use the value 9 in the $a0$ register.) Upon return from the **syscall**, the string will have been saved starting in the address specified, terminated with a null character.

You can reserve space in memory for any specific number of bytes using the **.space** directive, e.g., “.space 9”.

We have seen a few branch instructions in class. The unconditional branch instruction looks like this:

```
...
label:
...
    b label # go to the instruction at "label"
```

In addition, there are instructions for comparing registers and branching, for instance,

```
...
    bge $t3, $t4, label # go to "label" if $t3 >= $t4
...
label: ...
```
Due Tues., 26 Sept.  

You may need to load a byte into a register; the command for doing this is `lb`:

```
.data
...
hex: .space 9  # reserve space
...
.text
...
    la  $t5,hex  # load address "hex" into register
...
    lb  $t4,3($t5) # loads the fourth byte from label "hex"
```

NOTE: I am assuming that you will be entering EXACTLY 8 characters; use leading zeros if you want to test your program with small values. See sample output below.

Sample runs:

```
$ java -jar Mars.jar hex.asm
MARS 3.0 Copyright 2003-2006 Pete Sanderson and Kenneth Vollmar
00000010
The hex integer 00000010, in decimal, is 16

accs@ALDVENV27 ~/courses/cs210
$ java -jar Mars.jar hex.asm
MARS 3.0 Copyright 2003-2006 Pete Sanderson and Kenneth Vollmar
7fffffff
The hex integer 7fffffff, in decimal, is 2147483647

accs@ALDVENV27 ~/courses/cs210
$ java -jar Mars.jar hex.asm
MARS 3.0 Copyright 2003-2006 Pete Sanderson and Kenneth Vollmar
000017ae
The hex integer 000017ae, in decimal, is 6062
```

Hand in your `fully commented code`, together with the output from several sample runs. Output must be labelled as shown in the samples.