EE/CS/CPE 3760 Program 2

# Overview

**Work on this assignment individually. Nobody else should see your code. You may discuss high-level strategies as in other programming courses.**

You are to write a function in MIPS assembly for use in the SPIM simulator. Your procedure will convert a hexadecimal value in string form into an integer. A test harness, *prog2\_test\_harness.s*, has been provided to drive your function for testing purposes. The test harness asks the user to enter the hexadecimal number as a string, calls your function, and prints out the resulting integer. You may find it useful to write your own test harness to enable faster or more thorough testing. You may also find that prototyping in C is helpful. However, you will only turn in your function, not any additional development code.

# Requirements

1. Provide a subroutine *convert\_hex\_str* that converts a string representation of a hexadecimal number into its numeric (binary) form
   1. **Input Parameters**:   
      $a0 contains the address of the first character of a Null-terminated string consisting of only:
      1. Digit characters (‘0’-‘9’)
      2. Hexadecimal letter characters, either upper or lower case (‘a’-‘f’, ‘A’-‘F’)
      3. A NUL terminator (ASCII 0)
   2. **Return Values**:
      1. $v0: The converted integer value. It must contain an integer that corresponds to the value represented by the string passed to the function. The string is considered a hexadecimal representation of an unsigned integer that fits in 31 bits.
      2. $v1: An error status. 0 means no error. 1 means overflow error. If an error is returned, the integer value in $v0 is ignored and can be any value.
2. Your function must be called convert\_hex\_str
3. Your function must accept one input value, a pointer to the string, in register $a0. You must not modify the string during processing. You may assume the input string is formatted properly according to the rules in (1a). Your function will not be tested with improperly formatted strings, and there are no behavioral requirements if the string is not properly formatted.
4. Your function should not invoke any syscalls. You do not need to print output or read keyboard input. That is all done by the test harness.
5. Overflow. You are returning a normal 32b signed twos-complement integer. Even though the value will always be non-negative, $v0 uses signed form. What is the largest positive value that can be represented? If the input string represents a value larger than this, your function should report an error by returning 1 in $v1.
6. Your function must follow MIPS register conventions:
   1. Callee-saved registers ($s regs) must not be altered by your function, or they must be saved to the stack and restored before returning.
   2. You must not modify memory outside of your call stack frame. (If you don’t know what a call stack frame is, just don’t modify memory at all.)
   3. You must conclude your subroutine by returning to the return address in register $ra.
7. Your code must be well-commented to receive full credit. This means **any non-trivial line of code should have an accompanying comment**. For examples of appropriate comments, refer to any of the code examples on Canvas.

# Examples

|  |  |  |
| --- | --- | --- |
| Input String | Result | Comments |
| “70” | $v0=112 $v1=0 | 70 hex = 112 decimal |
| “aF” | $v0=175 $v1=0 |  |
| “0000000abc” | $v0=2748 $v1=0 | Any number of leading zeros are OK |
| “123456789” | $v0=anything $v1 =1 | Overflow! Too large (9 digits) |
| “80000000” | $v0=anything $v1 =1 | Overflow! Larger than max positive int |

# Algorithmic Hints

You may use any reasonable approach. The following suggestions will help you devise a complete solution. An efficient solution is not required for full credit, but it is nice. Good solutions will use fewer than 25 MIPS instructions/pseudoinstructions.

The string exists in memory from first character to last. The first character represents the most significant hex digit. You can process the hex digits one at a time – use LBU (load byte unsigned) to make reading each digit easier. Start with hex\_value=0. For each character in the string, convert the ASCII code to its hexadecimal value (0 to 15). Since a hexadecimal digit represents four bits, shift hex\_value left 4 bits, and add the digit’s value:

hex\_value = (hex\_value << 4) + digit\_value

Increment the string pointer to proceed to the next character. When you reach the NUL terminator, you are done, and hex\_value will contain the integer value.

There are two related overflow conditions. (1) More than eight digits (not counting leading zeros) and (2) fits in eight digits, but greater than maximum positive integer (this is the same as bit 31 == ‘1’). The easiest way to check for this is by looking at the most significant bits of your result as you build it.

Your function should not assume any initial values other than those specified for $a0. If your function works for the first hex conversion, but fails for subsequent conversions, you have probably made an assumption about initial register values or overwritten an $s register (callee-saved).

# Turning in your Solution

1. Place your function and any related data variables in between the comments in the test harness provided. Do not modify any code outside of these comments. The program should assemble and run in this form.
2. Save the above code as a .s file, and name it using the following convention: “last\_first\_prog2.s”. For example, Luke Skywalker would use: “Skywalker\_Luke\_prog2.s”.
3. Upload your solution to Canvas. There is no need to upload a screen shot.