CSE 378 11wi Homework 1: Getting Started with MIPS

Due: Monday, January 24, 2011 at 11 pm

The objective of this first assignment is to write two fairly short programs in MIPS assembly language and execute them using SPIM to verify that they work. For this purpose, you will write assembly versions of **(a) the strcmp() function** and **(b) the atoi() function**. These functions have basically the same specifications as the ones in the standard C library.

Writing Assembly language programs

There are several things that you should keep in mind when writing an assembly program. The first is to write good comments as you write your code, rather than inserting them when you are done with the program. This will make it much easier for you to track down problems as you are writing, and will save you a lot of time when debugging. Leave the comments in the program when you turn it in, as this will make it much easier for the TAs to figure out what you were trying to do with your code, and will be essential for getting partial credit if necessary.

Secondly, you should follow proper conventions in your code. For this assignment, we will focus on register usage conventions. Each register has a name associated with it and a purpose, and it is good style to follow these conventions. For example, the register `a0` is meant to be used to pass arguments to a function, while the register `t0` is used for temporary storage of data and could be erased by a called function. A list of register names and uses can be found on the green card included with the textbook, among other places.

Last but not least, simplicity is key when writing assembly programs. There will often be ways to write programs that are more compact or faster than a straightforward solution, but you should focus on getting a working, easy to understand program rather than one that is excessively clever or micro-optimized. Always make sure that you have a clean, working version to start from if you do decide to optimize your program. Remember, an optimized program that fails to perform the required tasks is not worth nearly as much as a straightforward program that succeeds. If you do choose to work on optimizing your programs, be sure to clearly comment and explain your code where necessary, as the purpose of some optimizations may not be immediately apparent.

a. A string comparison function

The purpose of `strcmp()` is to compare two strings in memory, character-by-character, to see which comes first in the standard lexicographic order.

String representation

Strings are represented by contiguous bytes in memory (each byte is an ASCII character) followed by the NUL character (`0x0`).

Interface

The arguments to your function are the addresses of the two strings to be compared. The memory address of the beginning character of the first string is in register `a0`, and the memory address of the beginning character of the second string is in register `a1`. If the first string is greater than the second, return a positive number. If the second string is greater, return a negative number. If the strings are equal, return 0.

Lexicographic ordering

The normal `strcmp()` compares two strings character-by-character, according to the integer ASCII values of the characters. This ordering is fairly intuitive; here are some examples:

“a” < *“b”*
“abc” < *“abcd”*
“A” < *“a”*

b. A string to integer conversion function

The `atoi()` function parses a string *str*, interprets its contents as a signed integer number, and returns the result as a 32-bit 2's complement binary integer. The string may begin with an optional initial *plus* or *minus* sign. Following the optional sign, if present, or beginning the string, if there is no leading sign, are one or more decimal digit characters in the range 0 to 9. The string may contain additional non-digit characters after those that form the number. Conversion should stop at either the end of the string or when a non-digit character is reached, whichever comes first. You can ignore overflow if there are too many digits in the integer part of the string. If the string does not contain any digit characters, no conversion is performed.

Interface

Your function will be provided with the memory address of the beginning character of the string to be converted, *str*, in register `a0`. Return the 32-bit two's complement integer value of *str*. Return `0xdeadbeef` if *str* does not represent a valid integer.

Testing

Here are a few possible test cases you could consider:
1234
−423
34.77
*hu*231*n*
5*lbs*.
You should come up with more test cases so that you fully test the function.

What you should do:

1. We have provided two starter files for you to use linked on the course web page. You should use `hw1-strcmp.s` for the first part and `hw1-atoi.s` for the second. In both cases, modify the files by adding code needed to complete the assignment.

2. Use the standard calling conventions and specified interfaces for the functions.

3. You may find it helpful to write the code in C first to understand and debug the algorithm before working on the assembly code.

4. Load your programs into SPIM and execute them. Each of the starter files contains a `main` function that includes most, but not all, of the code needed to call your function once. You should complete that code to call the function using the supplied test case, then add additional tests to verify that your functions work properly on a variety of input values. Some credit will be awarded for the quality of your tests. Quantity is not necessarily a virtue — be thorough but avoid excessive, redundant tests.

5. When you are satisfied with your solution, you should turn in the two assembler (`.s`) files. Turn in the complete files with your functions, the `main` functions, and the test cases you used to verify your code. Please include your name at the top of the `.s` file in a comment.

6. Submit your assignment via Catalyst using the dropbox on CSE 378 web page