CS 341L Fall 2009 Lab 3

Assigned: Wednesday, 9 September 2009

Due: 10:00am, Monday 28 September 2009 by attachment as an email to labturnin.cs341l@gmail.com

You should attach your program with your last name and the lab number in the name of the file, e.g., "knockel-lab3.S".

No late assignments will be accepted unless circumstances warrant an extension for the whole class (e.g., server is down all weekend, etc.). All code must be implemented as hand-written assembly, no copying and pasting or using a compiler is acceptable. You are expected to do your own work as an individual effort, copying the MIPS code of others is considered cheating.

The purpose of this assignment is to make you familiar with MIPS loops, function calls, stack usage, data manipulation, and register usage.

You are to implement a variety of C functions in handwritten MIPS assembly. In other words, your turned in code cannot include any library functions or link in any outside C code. You should prefix all of your own handwritten assembly functions with "my_". You'll need to write *my_strcpy*, *my_strcat*, *my_puts*, *my_strcmp*, *my_strlen*, *my_itoa*, *my_reverse*, *my_bubblesort*, and *my_swap*.

Everything should be in your own, hand-written assembly. Your versions of *strcpy*, *strcat*, *puts*, *strcmp*, and *strlen* should conform to the the Linux man pages that describe these functions and their behavior/return value. Your *puts* function can just be a wrapper of your Lab 1 code with a variable string argument and appropriate return value. Your *itoa*, *reverse*, *bubblesort*, and *swap* should conform to the C code below, where *itoa* and *reverse* are from K&R and *bubblesort* is obvious.

Additionally, starter MIPS code for this lab will be provided, as well as a testing suite written in C and a Makefile to automate compiling, assembling, and linking all of the necessary files. The starter MIPS code for this lab just wraps its respective library code, e.g. *my_strcmp* just calls *strcmp*, so you will find that the testing suite will initially report that all tests have passed. Remove the WRAP_CALL() line in each provided MIPS stub and replace it with your own, hand-written MIPS assembly, in the order that you choose, and then watch for output from the testing suite to help you debug, until all wrapper code is replaced with your own, hand-written assembly that passes all of the tests. Note that you can modify the testing suite according to your desiring, since you will only be turning in the *.S file.

Before you begin, rename yourname-lab3.S to the appropriate name, and then change its reference in the Makefile.

Compile your program with the following command

```
make
```

Then run the test suite:

```
./lab3
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

void swap(int *a, int base1, int base2)
{
     int temp;

     temp = a[base1];
     a[base1] = a[base2];
     a[base2] = temp;
}

void bubblesort(int *a, int n)
{
     int outerloop = n - 1, innerloop, swapped;

     do
     {
          swapped = 0;
          for (innerloop = 0; innerloop < outerloop; innerloop++)
          {
               if (a[innerloop] > a[innerloop + 1])
               {
                    swap(a, innerloop, innerloop + 1);
                    swapped = 1;
               }
          }
          outerloop--;
     }
     while (swapped);
}
```

```
/* reverse:  reverse string s in place */
void reverse(char s[])
{
    int c, i, j;

    for (i = 0, j = strlen(s)-1; i<j; i++, j--) {
        c = s[i];
        s[i] = s[j];
        s[j] = c;
    }
}


/* itoa:  convert n to characters in s */
void itoa(int n, char s[])
{
    int i, sign;

    if ((sign = n) < 0)  /* record sign */
        n = -n;          /* make n positive */
    i = 0;
    do {       /* generate digits in reverse order */
        s[i++] = n % 10 + '0';   /* get next digit */
    } while ((n /= 10) > 0);     /* delete it */
    if (sign < 0)
        s[i++] = '-';
    s[i] = '\0';
    reverse(s);
}
```

```
// This code isn't the stub MIPS code that you will be starting your lab with
// (that will be posted to my blog).
// This code is just for demonstration of the MIPS calling convention.
//
// Stub for function that takes a single argument and needs to use s0
// println is basically the same thing as puts

    .glob   println
    .ent    println
println:
    addiu   sp, sp, -32
    sw      ra, 28(sp)
    sw      s0, 24(sp)      // store callee-saved temporary
    sw      a0, 32(sp)      // store a0 into argument slot provided by caller

 ............

    lw      ra, 28(sp)
    lw      s0, 24(sp)      // restore callee-saved temporary
    addiu   sp, sp, 32
    jr      ra
    .end    println
```

```
// Stub for function that takes three arguments and needs to use s0 to make room
// for an array with 12 integers and return a value
// Assume this was declared as follows:
// int dosomething(int a, int b, int c)
// {
//       int A[12];
//       ............
//       return 3;
// }

    .globl  dosomething
    .ent    dosomething
dosomething:
    addiu   sp, sp, -80     // more room for a 12-int array
    sw      ra, 28(sp)
    sw      s0, 24(sp)      // store callee-saved temporary
    sw      a0, 80(sp)      // store a0 into argument slot provided by caller
    sw      a1, 84(sp)      // store a1 into argument slot provided by caller
    sw      a2, 88(sp)      // store a2 into argument slot provided by caller
    addi    s0, sp, 44      // s0 is now at the base of local array A

 ............

    li      v0, 3
    lw      ra, 28(sp)
    lw      s0, 24(sp)      // restore callee-saved temporary
    addiu   sp, sp, 80
    jr      ra
    .end    dosomething
```