# CSCE 2303 – Computer Organization and Assembly Language Programming
# Spring 2019

## Project 1: Parser and MIPS Code Generator for *sprintf* (15%)

**Submission Deadline: March 28th, 2019, 11:59 pm**

## 1. Background

This project is intended to make the student familiar with MIPS coding. *printf* is a C function that prints any sequence of characters on the console output (like *cout* in C++). *sprintf* has basically the same behavior as *printf* except that the whole output string is put into a buffer (you specify) instead of the console output. In other words, *sprintf* composes a string with the same text that would be printed with *printf*, but instead of being printed on the screen, the text is stored as a C string.

**e.g.,**

```
x = -17, y = 43, z = 104;  printf ("x = %d, y = %x, z = %c");
```

Output on the console is: `x = -17, y = 2b, z = h`

Here x is printed as a signed decimal integer, y is printed as an unsigned hexadecimal integer, and z is printed as an ASCII code character, that is according to the format specifiers %d, %x, and %c, respectively.

*sprintf* does the same operation except that it stores the output text "`x = -17, y = 2b, z = h`" in an output string, instead of displaying it on the console output.

## 2. Requirements

You are required to develop a parser and MIPS code generator for the *sprintf* function using MARS simulator. Your program must:

I. Read a string from the user containing the full *sprintf* function statement. Example input from the user is as follows:

```
sprintf ("Calculating the area\nLength = %d and width = %d\nArea
= %d (0x%X)", a, b, c, c);
```

II. Parse the input string to extract the format string of *sprintf* that is the text in between the double quotations "". Then extract the additional arguments of the *sprintf* function, which will basically be the variable names that need to be printed in the output string with their specified formats.

For simplicity, we'll assume the program only has the variables a, b, c, d, e to deal with in the *sprintf* function, and these variables are to be loaded at the beginning of the program into registers $t5-$t9, respectively.

III. Implement a MIPS procedure to handle the *sprintf* function
`int sprintf (char *outbuf, char *format, ...)`
The first argument of the function is the address of a character array `outbuf` into which your procedure will put its results. `outbuf` should point to the allocated memory sufficient to hold the generated text. The second argument is the address of the *sprintf* format string in which each occurrence of a percent sign (%) indicates where one of the subsequent arguments is to be substituted and how it is to be formatted. The remaining arguments are values (agreed to be variables a through e as specified in II above) that are to be converted to printable character form according to the format specifiers in the format string.
*sprintf* returns at the end of the function the number of characters in its output string, not including the null at the end.

Since *printf/sprintf* can take a different number of arguments, at each use, you will need to use stack-argument passing. Your function must accept any number of arguments, passed according to MIPS conventions. The first four arguments are passed through the $a0-$a3 registers. If there are more than four arguments, the remaining arguments are passed through the stack, where each argument gets one word of stack space.

IV. The *sprintf* procedure should fill `outbuf` with the correct formatted output string. You do not have to implement all of the formatting options of the real *sprintf*. Below are the specifiers you are required to implement and deliver in your project code:

- %d: treat the argument as a signed integer and output in decimal
- %u: treat the argument as an unsigned integer and output in decimal
- %b: treat the argument as an unsigned integer and output in binary
- %x: treat the argument as an unsigned integer and output in lowercase hexadecimal
- %X: treat the argument as an unsigned integer and output in uppercase hexadecimal
- %o: treat the argument as an unsigned integer and output in octal

V. You need to have test cases in your project report containing each of the previous formatting options to show your work. Screenshots of the running program should also be present, displaying the resulting string in the console output and showing the MIPS memory holding this resulting string.

## 3. Extra Credit (1%)

You need to implement these two extra formatting options:

- %c: treat the low byte of the argument word as a character and copy it to the output
- %s: treat the argument as a pointer to a null-terminated string to be copied to the output

## 4. Project Guidelines

- Work in a group of 2 students (3 is an exception and should be approved by the Instructor).
- **All project files and the report should be submitted on Blackboard by Thur. 28-Mar-2019, 11:59 pm**. The report should outline your design and how to run your program. Also, it should outline the challenges you faced as well as any limitation your program has.
- You will have to **demo the project to the TA** on your own PCs to get the full credit. All group members must be present and they might get different grades based on the demo. Details of the demo will be posted later.