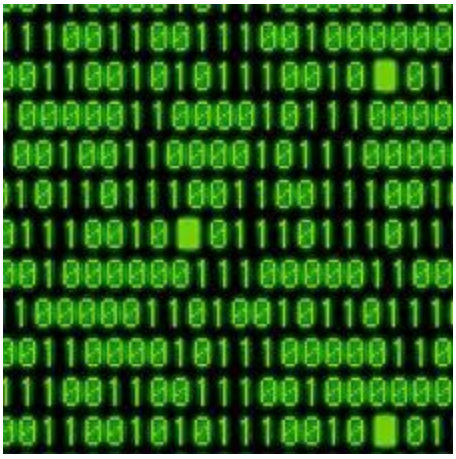
Sprintf Function

Implementing the sprintf function using assembly language



Ramy ElGendi

Sara Ahmed

30.03.2019

Computer Organization and Assembly Language

Summary of the Code

Our code is meant to implement a sprintf function. To implement the function, we must first parse the string, meaning we store what is between quotations into a character array known as “format”. We did so by implementing a function known as “ProcessInput” that takes whatever is in the input string, and looks for the possible occurrences that could occur. It takes whatever is in between quotations, and stores it in the format string. If a space is found that is not in between the quotations, the ProcessInput loop will iterate again, meaning that it will ignore its presence. If a comma is found, that means that its most likely in between the parameters, or the values that the sprintf is meant to take, meaning that those values will be stored. These stored values are placed in registers $t5-$t9, so if the character ‘a’ is found that will automatically correspond to $t5 in our code, and so forth. Once the appropriate value is located, it will be stored in the stack, in the correct order. To do that, we had to store the values normally, and then reverse the stack. All that is left now is to store the values in the character array known as the outbuf, as the format string, and parameters have been passed as arguments to the sprintf function. The sprintf function takes the format character array, and scans it for a % sign, once that's found its send to a function named findfunct, where it looks at the corresponding byte, and sends it straight to the appropriate function. The function implements the conversion needed, and stores the output in the outbuf character array. Once the format has been fully iterated through, it returns back to the main where the outbuf will be printed, and the entire program will end.

Challenges We Faced

The biggest challenge for us was implementing the reverse stack function. It, at first, took us quite a while to realize that we had to reverse the stack, and once we realized that we had to, it took us quite a while to learn how to. Another major challenge for us was starting the project. Once we read the project requirements, we were incredibly overwhelmed by the amount we had to cover, and it was very difficult to know where to begin in the first place. However, once we started implementing the functions one by one, the whole code came together quite fast.

Limitations of Our Program

Although our code can take any number of arguments its still limited to $t5-$t9, meaning that we have only 5 values that the user can choose from. This is a major limitation as compared to the regular sprintf function, it can't take any number of variables. In addition, the user cannot enter values that are not a, b, c, d, or e, and each value has its predetermined value, making it hard for the user to change it at will. Another limitation is MARS inability to alter the first signed bit of the program, so when outputting the a negative integer to decimal, I had to place the negative or dash in front of the number to get it to output correctly. This was a reoccuring theme in all our functions. The amount of bytes that the outbuf can print is predetermined, so the user is limited to a set amount of space. The same goes for the input, and format string.