CSC 3210 Computer Organization and programming

LAB 11

String Operations

ASSIGNMENT FOR LAB 11

Tasks to be done in this lab:

- How to copy a string
- How to find the length of a string
- How to count the number of occurrences of a character in a string
- How to compare two strings

INSTRUCTIONS FOR LAB 11

In this lab, you will work with strings.

- Part 1 - Copying a given string
- Part 2 - Finding the length of a given string
- Part 3 - Counting spaces in a string
- Part 4 - Comparing two strings

PART-1

- Read and display the given string as part of this exercise. Use the string "RISC-V Assembly Programming".
- Code is provided for "string_copy.S", as shown below. Copy this to a local file. Note that the first instructions use an ecall named "sbrk" to reserve space, then store the address of the reserved space at "copy".

```
# This is written for VSCode/Venus
# Copy a string.
# -MCW
.text
main:
    # Reserve 100 bytes
    li    a0, 9           # sbrk
    li    a1,100
    ecall
    # a0 has pointer to reserved memory
    la    t0, copy
    sw    a0, 0(t0)         # store the address into "copy"
                        # We could move it to s3 and skip this,but it is a good idea.
```

PART-1

```
# Note: we use s2,s3,s4 to make sure the values do not change
      # when we do a call/ecall
      la    s2, mystring      # s2 points to "mystring"
      la    s3, copy          # s3 points to "copy"
      la    s4, mystring_end  # s4 points to memory after string
   loop:
      lb    s0, 0(s2)         # Access byte from string
      sb    s0, 0(s3)         # Copy byte to "copy"
      addi  s2, s2, 1         # Increment s2
      addi  s3, s3, 1         # Increment s3
      # Now print the array value out
      mv     a1, s0
      li     a0, 1      # print an integer
      ecall
      # print space
      li    a0, 11     # print a character
      li    a1, 32     # space (the character to print)
      ecall
      bne   s2, s4, loop     # Compare s2 with s4
                        # (address of memory after string # Jump to loop if not equal
```

PART-1

```
# We are done, so print the copy
    li    a0, 4           # print a string
    la    a1, copy        # load address of "copy" string
    ecall
    # print NL
    li    a0, 11     # print a character
    li    a1, 10     # NL (the character to print)
    ecall
    # exit the program
    li     a0, 17
    li     a1, 0              # 0 for everything is OK
    ecall
.data       # Data section, initialized variables
mystring:  .string "RISC-V Assembly Programming"
mystring_end:
#.bss      # VSCode/Venus does not support .bss
copy: .word 0
```

- Use VSCode/Venus to simulate the above code. It should print the copied string. Verify that it works.

PART-1

QUESTIONS:

1. **This code does more than just copy a string. What changes must be made to only copy a string and print the copy?**
2. **What does the bne instruction do in the loop?**
3. **What would happen if the copy buffer is not allocated enough space to hold the entire string?**
4. **How does the code ensure that the copied string (copy) is null terminated?**
5. **How do we know that this works? What modification could you make to convince someone that it does actually make a copy and is not simply printing the same thing twice?**
6. **Why do we have the line copy: .word 0 instead of copy: .word 100 if we want "copy" to point to 100 bytes?**

PART-2

- In this part, go through "mystring" character by character, and count the length. Do this in a subroutine. Then display a double-quote character, then the string, then another double-quote character, a space, the length, and a new-line.
- Here is what the code to do the count looks like. Pay attention to the comments to know how to use it.

```
find_len:
    # Caller will have code like this:
    #la    a0, mystring      # a0 points to "mystring"
    #call  find_len
    # Put the length of the string into a1
    # Set count (in register a1) to zero
    mv    a1, x0
find_len_loop:
    lb    t0, 0(a0)        # Access byte from string
    addi  a0, a0, 1        # Increment a0 (pointer to next byte)
    addi  a1, a1, 1        # Increment a1 (count)
    # Check: is the byte a zero?
    bne   t0, x0, find_len_loop
    ret
```

- Verify that the output is correct.

PART-2

Questions:

1. **Why do we use "lb" to get a character, and not "lw"?**
2. **If the string were to contain non-ASCII characters or multi-byte characters, how would that affect the length reported?**
3. **If you wanted to use different registers for the values passed to and from the subroutine, what changes would you have to make?**
4. **Suppose that the string ends with a space. Would it be obvious to the user that the number reported is correct?**

PART-3

- For this part, count the number of spaces that appear in the string. You can use a command such as li t0, ' ' to load a space into register t0 (or whatever register makes sense). Modify the code from one of the previous parts to count the spaces.
- The following algorithm should help. Lines with "#" are comments where the line under it accomplishes the task. Lines with "##" indicate places where you are meant to add code.

```
    # Load the character to look for in t0
    li    t0, ' '
    # Load the address of the string into t1
    la    t1, mystring
count_spaces:
    ## Is the character a space? If not, skip around the increment
    ## Increment the count
skip_inc:
    ## Did we find a character 0?
    ## If not, branch to count_spaces
    ## Now, write out the count and exit the program.
```

PART-3

QUESTIONS:

1. **Suppose that your friend sees your program to count spaces, and tells you that you should use blt instead of the branch command that you use at the "If not, skip around the increment" part. Would that work? Explain.**
2. **Describe what the instructions beq and bne do.**
3. **What if the string has two or more spaces in a row? Does the code still work?**
4. **What if the string begins with a space? Does the code still work?**

PART-4

- For this exercise use the string "RISC-v Assembly Programming" and "RISC-V Assembly Programming", and write a subroutine to compare the two. Print either "equal" or "not equal" depending on the result. Show the outputs, and try a few variations for the strings.

Questions:

1. **Are the original two given strings the same? Why or why not?**

IMPORTANT NOTE:

Remember that we will grade your lab report so it is vital to turn that in. The other files (your code, a text version of any log file, etc.) are to document your work in case we need more information.