Lab 4: Functions and Memory Allocations

Submission timestamps will be checked and enforced strictly by the CourseWeb; **late submissions will not be accepted**. Check the due date of this lab on the CourseWeb. Remember that, per the course syllabus, if you are not marked by your recitation instructor as having attended a recitation, your score will be cut in half.

In this lab, you are going to write functions (procedures) and learn how to allocate memory on the fly

Memory Allocations

In general, if we want to use the main memory to store some data, we need to allocate it in advance usually in the data segment (`.data`). For example, if we want to allocate 100 bytes:

```
.data
      buffer:    .space   100
```

In the above case, `.space` directive allocates a section of the main memory (in data segment) for 100 bytes where the location of the section of memory can be referred to by the label `buffer`. For example, we you want to store a value 12 (as a byte), this can be done by

```
      addi $s0, $zero, 12
      la   $s1, buffer
      sb   $s0, 0($s1)
```


Often time, we have no idea how many bytes we need to allocate in advance. As a result, we trend to allocate way to much space in advance which wastes a lot of memory. In case of MIPS, we can allocate a region of memory on the fly while the program is running. The memory region will be allocated in heap instead of the data segment. This can be done by using the system call number 9. Simply set the number of bytes that you want to allocate in `$a0`, set `$v0` to 9, and execute the `syscall` instruction. The address of newly allocate section of memory will be in the register `$v0`. For example, suppose you have a string of length 20 and you want to allocate memory to store this string:

```
      addi $v0, $zero, 9
      addi $a0, $zero, 21          # Need an extra byte for the null character
      syscall
      add  $s0, $zero, $v0         # $s0 is the address of the memory
      # Copy the string
      :
```

Syscall 8 (Read String)

Note that a null-terminated string in MIPS is simply a sequence of characters in a contiguous region of the main memory. Recall that a null-terminated string can be created in the data segment as follows:

```
.data
      aString:   .asciiz   "Hello World!!!"
```

MIPS also provides a system call to receive a string from the keyboard input. This is system call number 8 which requires two arguments, `$a0` — address of input buffer, and `$a1` — the maximum number of characters to read. The following is an example of a code that read string from keyboard input and print it on the console screen:

```
.data
       buffer:    .space    100
.text
       addi $v0, $zero, 8          # Syscall 8: Read string
       la   $a0, buffer            # Set the buffer
       addi $a1, $zero, 100        # Set the maximum to 100 (size of the buffer)
       syscall
       addi $v0, $zero, 4          # Syscall 4: Print string
       la   $a0, buffer            # Address of the string to print
       syscall
```

Unfortunately, the system call number 8 has a strange behavior. If a user types `Hello` and presses Enter, the data in the buffer (e.g., from the above code) will be:

```
0x48 0x65 0x6C 0x6C 0x6F 0x0A 0x00 0x00 ...
```


Note that `0x48` is the ASCII character `'H'`, `0x65` is the ASCII character `'e'`, and so on. Note that before the first null-character (`0x00`), the buffer contains the character `0x0A` which is the line feed character (Enter). In general, it should not be a part of an input string from a keyboard.

Functions

For this lab, you are going to implements four small functions related to null-terminated strings. All functions must follow the calling conventions discussed in class as follows:

1. Use `jal` instruction to call a function
2. Use `jr $ra` instruction to return to the caller
3. Arguments should be stored in registers `$a0` to `$a3`
4. Return values should be stored in register `$v0` and `$v1`
5. Caller cannot expect that values stored in registers `$t0` to `$t9` will be the same after making a function call
6. Callee must maintain values stored in registers `$s0` to `$s7`
7. Callee must use stack to backup registers
8. Stack should be adjusted according to the size of the memory required to backup registers and restore before going back to a caller

For this lab, a starter code is given. Download the file `lab04.asm` from the CourseWeb under this lab. In that file, you will see the main program on the top and four functions, including their descriptions, that you have to implement right after the main program. Simply add your implementation between a function name and the instruction `jr $ra`. You must implement the following functions:

1. `_strLength`: This function takes exactly one argument in `$a0` which is the address of a null-terminated string. It returns the length of the given string (number of characters excluding the null-character) in `$v0`.

2. `_strCopy`: This function takes two arguments; (1) an address of a destination in `$a0`, and (2) an address of a source null-terminates string. This function simply copy (character-by-character) from source to destination. **Note** that after this function is done, the destination should contain a null-terminated string identical to the given source string.

3. `_strCompare`: This function takes two arguments in `$a0` and `$a1` which are addresses of two null-terminated strings. This function compares whether two strings are identical. If they are identical, a 0 is returned in the register `$v0`. Otherwise, -1 is returned.

4. `_readString`: The purpose of this function are as follows:

   (a) read a string from the keyboard input using the system call number 8,

   (b) get rid of the linefeed character at the end,

   (c) allocate a region in heap for storing this input string,

   (d) copy string from temporary buffer to the memory allocated in previous step

   (e) return the memory address of the input string (in heap) using the register `$v0`

   Since the system call number 8 requires a buffer to store an input string temporary, your function can simply use a given buffer named `buffer` located in the data segment (provided by the starter code) as shown below


```
.data
        :
buffer: .space  100
```


   Note that if this function needs to know the length of the input string to allocate the correct amount of memory. You must use your implementation of the function `_strLength`. During the copy process from `buffer` to a new allocated memory, it must uses the function `_strCopy`. Use system call number 9 as discussed earlier to allocate memory in the heap section. Do not forget to allocate one extra byte for the null character.

Test Code

For this lab, a starter code is given. This code will perform the following:

1. Ask user for a number of strings that user wants to enter

2. Allocate memory in heap to store **addresses** of input string (not string themselves)

3. Ask user to enter strings by calling your function `_readString`. Every time your function `_readString` returns, it will store the address returned by the `_readString` function into the memory allocated earlier in step 2.

4. Once all strings have been entered, it will display all string starting from index 0.

5. Then it will ask user to enter another string

6. Once user enter a new string, it will search for the index of a given string using your `_strCompare` function. If a string is found, it will display the index of the string. Otherwise, it simply says string not found.

The following is an example of the output of the program when a string is found.

```
How many strings do you have?: 5
Please enter a string: Hello
Please enter a string: World
Please enter a string: Computer
Please enter a string: Data
Please enter a string: Assembly
The string at index 0 is "Hello"
The string at index 1 is "World"
The string at index 2 is "Computer"
The string at index 3 is "Data"
The string at index 4 is "Assembly"
Please enter a string: Computer
The index of the string "Computer" is 2.
```


The following is an example of the output of the program when a string is not found.

```
How many strings do you have?: 5
Please enter a string: Hello
Please enter a string: World
Please enter a string: Computer
Please enter a string: Data
Please enter a string: Assembly
The string at index 0 is "Hello"
The string at index 1 is "World"
The string at index 2 is "Computer"
The string at index 3 is "Data"
The string at index 4 is "Assembly"
Please enter a string: Dog
Could not find the string "Dog".
```


Note that if the program does not work properly, most likely there is something wrong with one of your functions. So, make sure your functions work correctly and follow all calling convention discussed in class.

Submission

Submit your `lab04.asm` file via CourseWeb before the due date stated on the CourseWeb.