CSCB58 Lab 10: Arrays and Functions

1 Introduction

Last week, we built branches and loops in MIPS assembly using labels and branches. This week, we will write some programs with arrays and functions. You might have already attempted to work with these as part of your project, but if you haven't then this will be a good starting point.

This week's lab assumes that you completed all of the material from last week. If you did not get the programs working last week, finish them before this lab.

Note: Your will submit your solution code to Quercus before the start of your practical section, please read the "Summary of TODOs" section for more details.

2 Arrays

As we discussed in the lecture, arrays are declared in the `.data` section of the assembly code. For example, the following segment of code declares an array of 6 integers.

```
.data
array1:    .word    5, 8, 3, 4, 7, 2
```

The array data locates in the memory, so in order to access the elements of the array, we need to perform memory access instructions such as `lw` and `sw`. To access each element, you need to correctly calculate the memory address of it before accessing it. The basic way to compute the memory address of an element is "`base + offset`", where `base` is the address of the first element of the array (value of "`array1`"), and `offset` is the index of the element multiplied by the size (in bytes) of each element of the array. Read the lecture slides for more details.

For your first task, write some code that iterates through the above array (`array1`) and computes the product of all elements in the array, and then prints it out to the screen. You can re-use some of the code from previous labs if you need. Be prepared to explain your code to your TA.

3 One-Level Function Calls

Just like any high level programming language, modularization (separating code into well defined procedures/functions) is an important idea for assembly programming. Conceptually, making function call is actually simple: we need to "jump" to another portion of code (the function body) then start executing the instructions in the function body. When we reach the end of that function, another "jump" is needed to go back to the caller.

In terms of passing arguments and return values, it can be done in many different ways, therefore certain conventions need to be defined to make sure that all programmers in the same project are on the same page. In this lab, we use a very simple convention (which is different from what we discussed in the lecture): Use registers `$a0` and `$a1` for storing the function arguments, and use `$v0` for storing the return value.

Download starter code `lab10b.s` from the following link:

`https://cscb58f20.ml/labs/lab10b.s`

The starter code is trying to implement the following piece of pseudocode. Read the comments in the starter code and complete the TODO parts.

```
def main():
    A = input("Enter a value for A")
    B = input("Enter a value for B")
    print "Before function"
    print "A + B = ", doAdd(A, B)
    print "A - B = ", doSub(A, B)

def doAdd(A, B):
    return A + B

def doSub(A, B):
    return A - B
```

Note: You are NOT allowed to add any label to your code.

4 Multi-Level Function Calls: Recursion

Things get more interesting when we have multi-level function calls, especially when you are implementing a recursive function. Since the return address is automatically stored in the `$ra` register when `jal` is executed,

when a function calls another function, the content of `$ra` will be overwritten and the calling function's return address will be forgotten. To avoid this, we need to remember the return address of each level of function somewhere, namely, the stack.

We can access the stack using the stack pointer value stored in register `$sp`. To push a word onto the stack, you can do:

```
addi $sp, $sp, -4    # move the stack pointer to increase stack size
sw $r, 0($sp)        # put the value in $r on the allocated space
```

To pop a word from the stack:

```
lw $r, 0($sp)        # load the word at the top of the stack
addi $sp, $sp, 4     # decrease the size of the stack
```

Other things also need to be remembered on and passed through the stack, such as the argument passed to the function being called, the return value of a function call, and the temporary values which need to be used after returning from the recursive call. The orders of the pushes and pops need to be design carefully so that you are always correctly passing and restoring the value that you expect.

Create a new file named `lab10c.s` and implement in assembly the following piece of pseudocode, which involves a recursive function `mystery`. This task could take longer time than others to finish, so you might want to start working on it earlier than usual. You may refer to the function examples in class for explanations of how to deal with the stack and functions.

```
def main():
    n = input("Enter a number: ")
    print("The result is:", mystery(n))

def mystery(n):
    if n == 0:
        return 0
    return mystery(n-1) + 2*n - 1
```

Once your get it to work, be ready to answer questions about how your code works. In particular, try to break your code by passing a very large input value n. How large does n need to be to cause an error? What kind of error is raised?

5 Summary of TODOs

Below is a short summary of the steps to be completed for this lab:

1. Read through the handout and get familiar with the procedure, and already start working early on the code to be prepared since the tasks are more challenging than previous labs.
2. Create and complete `lab10a.s` which implements the Array code.
3. Complete `lab10b.s` for one-level function calls.
4. Create and complete `lab10c.s` which implements the pseudocode with the recursive function `mystery`.
5. Before your ;ab session, submit your `lab10a.s`, `lab10b.s` and `lab10c.s` to Quercus.
6. Show up to your lab and be prepared to answer questions on how your code works.

Evaluation (5 marks in total):

1. 1 marks for part (a)
2. 1.5 marks for part (b)
3. 2.5 marks for part (c)

Congratulations! You just finished your final lab of CSCB58! Thank your TAs for their work throughout the term!