CS447 Lab #5 (Week 6)

Introduction

For this lab, you will write a recursive MIPS function, with appropriate calling conventions and stack usage.

Note: This Lab must be submitted online by 11:59 pm tonight. Please see the instructions on the final page.

Objective: Write a recursive MIPS function to compute the value of the Fibonacci series at a requested index.

The **Fibonacci sequence** refers to a series of numbers, defined over the range from 0 to infinity, that begins as follows[1]:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55….

Let each element of the sequence be denoted as ***F*(*n*)**, where ***n*** indicates the index of the number's position within the sequence (with enumeration starting at 0). ***F*(*n*)** can be determined according to the following recursive formulation[2]:

F(0) = 0, F(1) = 1

For all *n* > 1:

$$F(n) = F(n-1) + F(n-2)$$

The correctness of this recurrence relation can be confirmed by examining the example numbers above. For example, *F*(6) = 8, which is equal to the sum of *F*(5) and *F*(4): 5+3.

The Fibonacci sequence is popular across disciplines, in part due to its close relationship to patterns found in nature[3]. In Computer Science, the Fibonacci sequence offers a good opportunity to practice writing recursive functions in a new language. The box below provides a pseudocode description[4] of a basic recursive function for computing *F*(*n*) for any nonnegative integer *n*:

[1] Reference: http://oeis.org/A000045
[2] Reference: http://mathworld.wolfram.com/FibonacciNumber.html
[3] See http://www.geom.uiuc.edu/~demo5337/s97b/spiral.html and http://www.bbc.co.uk/programmes/b008ct2j
[4] See http://video.mit.edu/watch/introduction-to-algorithms-lecture-19-dynamic-programming-i-fibonacci-shortest-paths-14225/. The description of the recursive algorithm starts at approximately 7:00. The remainder of the video describes dynamic programming, which is a more efficient approach to computing Fibonacci numbers.

```
fib(n):
   if n = 0: return 0
   else if n ≤ 2: return 1
   else: return fib(n-1) + fib(n-2)
```

This pseudocode can serve as helpful guidance in setting up a similar Fibonacci function in MIPS. Specific program requirements are listed below, followed by additional information and tips:

Program requirements:

1. Prompt the user to enter a nonnegative integer *n*.
   - You can assume that the user will enter an integer between 0 (since anything < 0 is not valid) and 20 (so the program won't take too long to finish).
2. Pass *n* as an argument to a function named **fib** (or any other descriptive name you might prefer). You should use register **$a0** for this purpose.
3. **Fib** should compute *F*(n) using the recursive method described above. This implies that two recursive calls to **fib** should be present within the body of **fib.**
4. **Fib** should return *F*(n) by placing it in register **$v0**.
5. Print a message to the screen telling the user what the resulting Fibonacci number is.
6. Practice appropriate calling conventions and stack usage, including:
   a. The body of the function should include statements both to push register contents onto the stack, and to pop these values before the function exits.
   b. Part (a) should be achieved by adjusting the stack pointer (**$sp**) and making use of store/load instructions to manipulate memory.
   c. **If** fib uses any **s*** registers, it should first back up any pre-existing contents of these registers, to avoid the risk of overwriting any values used by the calling instructions.
   d. The program should not rely on the assumption that the contents of any **t*** registers will be preserved across any function calls.

Sample output:

```
Please enter a non-negative integer:15
The Fibonacci number at this position is 610
-- program is finished running --
```

To test your program, you might want to make use of the table provided here:
http://oeis.org/A000045/list

Although the final program will not need to be very long, some aspects of the **fib** function may require careful thought. If you are encountering any difficulty, here are some tips:

1. The instructor provided an example recursive function, **factorial**, in the file **func_kinds.asm**. The asm file is available on the website, in the zip file of Materials for 9/27. This can be a good reference for constructing the **fib** function.
    - Keep in mind that **fib** is somewhat different, though, since you will need to call **fib** twice within its own body, and combine the results.
2. Since a bug in a recursive function can cause the program to run indefinitely (or at least until it crashes), you might find it helpful to use debugging tools, such as the run speed slider bar or the 1-step buttons.
3. It might help to start by first writing the instructions to cover the base cases (*n* = 0, 1, or 2), and confirming that they work, before attempting the recursive part.
4. Except when *n* matches one of the base cases, **fib** will be primarily operating as a non-leaf function. Remember that this has implications for how you manage the contents of register **$ra**.
5. Since all calls to **fib** use **$a0** to hold the argument and **$v0** to hold the return value, you will want to think carefully about whether and when you should backup or restore the values of these registers.
6. As is generally the case with recursion, you may find it helpful to make a drawing of the various states of the program as you progress down and emerge out of the call stack.

Submission:

If you finish during recitation, please notify the TA, so that your credit for this assignment can be confirmed right away.

Regardless of whether you finish during recitation or not, you must submit your work online. Save the file as **YourPittUserName_lab5.asm** and submit the file via the appropriate link in Courseweb (see Course Documents/Week 6 Lab). The lab must be submitted **today, October 7th**, by **11:59 pm**.