Assignment SPIM 3

Due Date: April 29

Purpose

In this project, you will try out SPIM subroutines, in the form of recursion. To do this properly, you must store local variables on the stack. This forces you to take care of the $sp and $fp pointers to store the variables in memory correctly. *Follow the in-class and/or text examples carefully!*

Problem

We can find the maximum value in a list of integers by using a loop. But what fun is that? Instead, we will find the maximum value in the list using recursion. An algorithm for finding this is as follows:

```
function findMax (list)
   valOne = first value in list
   valTwo = findMax (rest of the list)

   if (valOne > valTwo)
      return valOne
   else
      return valTwo
```

You should be able to trace this algorithm to convince yourself that it finds the largest value. However, note this algorithm does not include all of the details necessary to code the program. For example, what is the stopping condition (base case)? Thus, you may need additional variable(s)/parameter(s) in the implementation of the algorithm.

Input

Assume the program has a list (array) of 50 integers in global memory (thus, you can access this memory location in any subroutine). The program should prompt for n, the size of the sublist for which we wish to find the maximum value. Thus, $1 \leq n \leq 50$. The program should then return the maximum value in the sublist comprised of the first n values of the list.
You may assume the input will always be correct; no error checking is necessary.

Output

The output should be the maximum value of the first n values of the list. For example, if the list stored in memory was:
3, 12, -87, 22, 367, 90, -2, 68, 7860, ...
and $n = 6$, then the output would be:
`The maximum value of the sublist is 367`

Specifics

- You **must** do this recursively; no loops are allowed. In order for this to work properly, you must store local values in the stack before each new recursive call. Follow the sample program done in class, or the example in the book. Note that it doesn't hurt to allocate a little more storage than you need, but it's disastrous if you do not allocate enough (because you will overwrite data).

- **Do not treat any registers as global variables!** We are attempting to simulate how a good recursive function written in a high-level language gets translated to assembly.
- Your recursive solution must add some elements to the algorithm shown above, especially a parameter(s). Note that the list is global, so you don't have to pass the array itself as a parameter.
- The maximum value should be **returned** to the main procedure in $v0 and then displayed.
- Follow the argument passing and return value conventions described in class. This will make it easier for you to debug.
- For your own sanity, use good commenting style, including function and argument comments. Include an introductory comment as in your previous projects.

Notes

- Your best bet is to write a solution in a short C++ or Python program first, and then translate that to SPIM. Ultimately, the SPIM code will not be very long, but it can be confusing.
- Do not wait to start! You know how long it takes to write code in SPIM, and this is trickier than the last project.
- Add the Honor Code in a comment near the top that includes your electronic "signature." Turn in your source code via email by 11:59:59 PM on the due date. Use the naming convention as follows: your first initial followed by your last name and finally "SPIM3.s",[1] as in `mgousieSPIM3.s`. Note there are **no spaces** in the filename.

To understand recursion, one must first understand recursion.
– Stephen Hawking

[1] .a or .asm are valid filename extensions as well.