CS270 Recitation 9
LC-3 Recursion Exercise using Activation records

Goals

- To understand purpose of the runtime stack.
- To be able to draw the contents of the runtime stack.
- To learn how to implement recursion in LC-3.

Background

The local variables are kept in the *run-time stack* using the *activation record* mechanism illustrated in this recitation. Each call of a subroutine has its own activation record. (The global variables are kept in the global area of the memory, and the dynamically allocated variables are kept in the *heap*; we will look at them later.)

The runtime stack is an area of memory used to keep track of a program's progress. When a function is called, a new activation record is added to the stack. Each activation record contains space for the function parameters, the return value, the return address, the frame pointer, and finally the local variables.

The return value is the value returned to the caller, for example using "return 1" in C. The return address (R7) is the address to the next instruction in the calling function. The stack pointer (R6) points to the newest value on the stack. Since the stack pointer changes often, the frame pointer (R5) is used to load and store parameters and locals. The frame pointer contains the address of the first allocated local variable.

An activation record does not automatically appear, it must be built using assembly instructions to push values onto the stack.

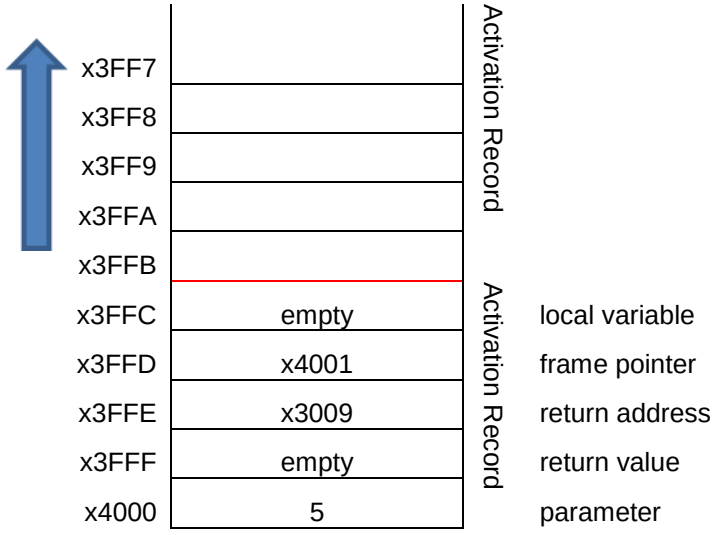

Figure 1: Stack snapshot

When a function is called, the *caller*:

- Pushes the function parameters onto the stack.
- Transfers control to the *callee* (the called function) using JSR or JSRR.

Upon entry into a function, the *callee*:

- Allocates space for the return value by decrementing the stack pointer (in R6) by the number of locals.
- Pushes the return address (in R7) onto the stack.
- Pushes the caller's frame pointer (in R5) onto the stack. R5 becomes R6-1.
- Allocates space for any local variables by decrementing the stack pointer.

```
;; setup caller portion of activation record
;; push function parameters
        ADD   R6,R6,#-1     ; Push step 1: decrement stack pointer
        STR   R2,R6,#0      ; Push step 2: copy param val=val-1
        JSR   FACTORIAL     ; Call factorial
;; tear down caller portion of activation record
;; push function parameters
        LDR   R0,R6,#0      ; Load result of call into a register
        ADD   R6,R6,#1      ; Pop return value
        ADD   R6,R6,#1      ; Pop parameter val
```

Figure 2: Code for setup and tear-down

When a function call completes, its activation record must be removed from the stack. This is achieved by popping values off of the stack into the appropriate registers.

When a function returns, the *callee*:

- Writes the return value to the allocated location, usually the frame pointer + 3.
- De-allocates local variables by adding to the stack pointer.
- Restores the caller's frame pointer by popping it off of the stack into R5.
- Restores the return address by popping it off of the stack into R7.
- Returns control to the caller by executing RET.

When a function is returns, the *caller*:

- Pops the return value into a register for use later.
- Pops the arguments passed to callee off of the stack.

Assignment

Create a new directory called R8, all files should reside in this directory. Download r9.asm and r9.c

Examine the LC-3 to see how it implements the recursive C program and how it constructs the activation record for each call.

You can set break points just before setup and just after tear-down (that will give you 4 breakpoints) to see how the stack frames are created and pushed into the stack, and then eventually removed.

Fill the provided table with values from the runtime stack of a program that computes a factorial. Use the LC-3 simulator to fill in at least three activation records in this table. Label which locations are used for the parameters, return value, return address, frame pointer, and locals. The C equivalent has been provided to help understand the program flow.

When you are finished, show the table to your TA.