Towers of Hanoi: Recursion and the Stack
Homework Lab #4
CEG 320: Computer Org. & Assembly

PURPOSE
Understanding how functions call each other and pass information by activation record on the stack is to understanding program performance. The use of the stack introduces the distinction for call-by-reference and call-by-value parameters, underlies the importance of understanding pointers, and is critical to understanding the efficiency of function calls. In this lab, students will construct an assembly level routine to implement function calls using a software stack and LC-3 style activation records for a simple recursive program. This is a two week assignment.

ASSIGNMENT
Using the LC-3 simulator, you will construct an assembly-level program to implement the program presented (in C) below. The “Towers of Hanoi” puzzle is described in more detail in Patt, Section 17.4.

```c
#include <stdio.h>
void MoveDisk (int diskNumber, int startPost, int endPost, int midPost) {
    if (diskNumber > 1) {
        MoveDisk (diskNumber-1,startPost, midPost, endPost);
        printf("Move disk %d from post %d to post %d.
", diskNumber, startPost, endPost);
        MoveDisk (diskNumber-1, midPost, endPost, startPost);
    } else {
        printf("Move disk 1 from post %d to post %d.
", startPost, endPost);
    }
    return;
}
void main () {
    int n;
    printf("--Towers of Hanoi--\nHow many disks? ");
    scanf("%d", &n);
    printf("Instructions to move %d disks from post 1 to post 3:\n",n);
    MoveDisk (n,1,3,2);
    return;
}
```

You must call MoveDisk from your “main” program located in memory at address x3000. You must implement MoveDisk as a function located in memory at address x3100. The bottom of the software stack will be located at address x5000 (Hint: x4FFF stores main()’s local variable n. You can produce the output directly or by using TRAPS such as PUTS – you do not have to implement printf as a function. For simplicity, you may assume that the number of disks “n” is always a single digit number from 1 to 9.
DEVELOPABLES
(1) The code: A complete printout of your well-commented Assembly code. Note: Well commented means “easy to follow the high-level flow and design decisions”. You do NOT have to comment every line of code, but you should have high-level comments for each 3-10 lines that represent a functional block of code. In addition to a hard copy printout of your code, please e-mail your code to the instructor.
(2) The execution: A printout demonstrating the successful execution of your code.

PROGRAM GRADING
Both the hard copy and the email containing the .ASM code are due by the beginning of class on the due date. Late projects will be penalized 10% for each 24 hour period (or portion thereof) late. Points will be assigned as follows:

10 points: Documentation: The documentation supporting the program should clearly and adequately explain and demonstrate the successful functionality of the project.

20 points: Compilation of C Code to Assembly: Your project should properly implement the I/O for the main and MoveDisk routines as two separate functions located at the specified locations. Your main function must be able to take input from the keyboard and pass that value (by any means) to the MoveDisk subroutine.
Example:
--Towers of Hanoi--
How many disks? 1
Instructions to move 3 disks from post 1 to post 3:
Move disk 1 from post 1 to post 3.

10 points: Correct creation of the activation record: Ideally, your project should pass all subroutine parameters through the stack by appropriate creating and destroying subroutine activation records.

10 points: Fully support recursion: A complete project should create and destroy activation records in such a way that multiple calls to the same subroutine result in unique activation records per call and thus fully support recursion. For the purposes of demonstration, it is sufficient to demonstrate that you can solve a three disk problem.
Example:
--Towers of Hanoi--
How many disks? 3
Instructions to move 3 disks from post 1 to post 3:
Move disk 1 from post 1 to post 3.
Move disk 2 from post 1 to post 2.
Move disk 1 from post 3 to post 2.
Move disk 3 from post 1 to post 3.
Move disk 1 from post 2 to post 1.
Move disk 2 from post 2 to post 3.
Move disk 1 from post 1 to post 3.