COMPUTER SYSTEMS ORGANISATION – SPRING 2012
LAB ASSIGNMENT

To be submitted in the lab next week

This handout contains the problem statement of the assignment followed by a small description of the "myPrint" function (provided in the template codes sent to you earlier) and how it works. Note that you have to change this function to implement the necessary functionality needed to solve the following problem.

Problem Statement: Write a program in arm assembly language which prints a sequence of steps to be followed to solve "Towers of Hanoi"(TOH) with 'n' rings (numbered from 1 to n in increasing order of weights)

Input: A single integer 'n'.
Output: Steps to be followed to solve TOH with n rings. Each step should be printed on a single line. A step is denoted by three integers "x y z", x being the ring to be moved, y is source rod and z is the destination rod. Assume that the rods are numbered 1, 2 and 3. Initially all the rods are at rod #1 and have to be moved to rod #3.

"myPrint" function: For convenience instructions are numbered from 1 to 5.

```
myPrint:
     stmfd     sp!,{r0,r1, r2, r3, r7, lr}   @1
     add  r7, sp, #0                         @2
     adr  r0, .LC0                           @3
     bl   printf                             @4
     ldmfd     sp!,{r0,r1, r2, r3, r7, pc}   @5
```

First let us describe the printf function in C and then relate it to this function.

printf in "C": All of us are familiar with "C" language, and the function printf provided by "C" to print any formatted string on the console/terminal. The syntax of printf in case of printing a single integer (say x) is

```
printf("%d\n", x);
```

Note that in case of printing an integer printf only takes two arguments,
1. The formatted string "%d\n".
2. An integer to replace the value of %d in the string.

Now with this background let us look at the myPrint function. We'll look at lines 1 and 5 at the end. First let us look at line 4. In this line we are making a function call to "printf". When we compile this code, the linker sees that we are making a function call to "printf" and hence imports its code into the file. But where are the arguments for printf? This is done in lines 2 and 3 (some of it is done by you when you move integer to be printed in r1 before calling myPrint). "printf" by convention needs register 'r7' to contain the value of stack pointer. Hence move (sp+0) i.e., the value in stack pointer to register 'r7'(this is just a convention, don't worry if you don't understand this). Also, "printf" follows the convention that the arguments should be passed in registers r0, r1, r2 and r3. That means the first argument should be passed in r0, second in r1, third in r2 and fourth in r3. In case of more arguments we will have to make use of stack. But in our case we need only two arguments, viz. the formatted string and the integer value. So, we first declare a string at location .LC0 in the code.

```
.LC0:
     .ascii    "%d\012\000"
     .align 2
```

Don't worry about \012 and \000. You can also declare the string as

```
.LC0:
     .ascii    "%d\n\0"
     .align 2
```

Sounds familiar? Now we need to load the address of this string into r0. Note that command in line 3 does exactly this thing. And in r1 we need the second argument, i.e. the integer to be printed. But note that we decided to call myPrint function with the integer to be printed in r1 placed before the function call. Hence r1 already has the value to be printed and there is no need to do any extra work for it. So, that's it! Just call printf and its done.

Finally we just need to see the instructions 1 and 5. Recall that across function calls we need to preserve the value of the registers. So, one way is that when a function is called we need to store the value of the registers it uses on stack and restore them on return (concept of callee saving the registers). The lines 1 and 5 does exactly this thing. We are changing r0 (for providing the address of string "%d\n" to printf), r7 and link register (since we are making another function call inside this function) inside our function. But why are we storing r1 and r2. It turns out that printf changes the value of the registers r1, r2 and r3. So, we need to store these registers also and recover them on return. Note that in last line we are directly restoring the value of lr into pc and hence there is no need of the instruction "mov pc, lr" to return back to the caller function.