Assignment 4 Solution

2.20. a) Neither nesting nor recursion are supported.

b) Nesting is supported, because different Call instructions will save the return address at different memory locations. Recursion is not supported.

c) Both nesting and recursion are supported.

2.21. To allow nesting, the first action performed by the subroutine is to save the contents of the link register on a stack. The Return instruction pops this value into the program counter. This supports recursion, that is, when the subroutine calls itself.

N-Factorial (N!)

N! = 1*2*3*…*(N-1)*N

a) Write a program that uses a loop to compute N!. Please use pseudocode (C or Java is fine). DO NOT write a function.

```
factorial = 1;
for (i = N; i > 0; i--)
{
    factorial = factorial * i;
}
```

b) Write the program in part (a) using assembly language. Please use the simplified assembly language we have used in class. DO NOT use the PowerPC assembly language.

```
         Move       #1,R0          ; R0 = 1 (factorial result)
         Move       N,R1           ; R1 = N
         Branch=0   DONE           ; Special case: 0! = 1 (Optional)
LOOP:    Multiply   R1,R0          ; R0 = R1 * R0
         Decr       R1             ; i--
         Branch>0   LOOP           ; Branch if i > N
DONE:    Move       R0,factorial   ; Store result
```

c) Write a recursive function in pseudocode (C or Java is fine) to compute the factorial of a given number, N. The skeleton is given below.

```
int factorial(int N)
{
     if (N < 2)
          return 1;
     else
          return(N * factorial(N-1));
}
```

d) Write the recursive factorial function from part (c) in assembly language. Please follow the rules below:

- Use the simplified assembly language we have used in class. DO NOT use the PowerPC assembly language.
- Assume the required parameter(s) are passed on the stack.
- All registers need to be saved on the stack if used.

```
FACT:     Move            FP,-(SP)        ; Save old frame pointer
          Move            SP,FP           ; Assign new frame pointer
          MoveMultiple    R0-R1,-(SP)     ; Save registers
          Move            8(FP),R0        ; R0 = N

IF:       Compare         #2,R0           ; Check R0 - 2
          Branch>=0       ELSE            ; if R0 >= 2
          Move            #1,R1           ; Result = 1
          Branch          FI

ELSE:     Move            R0,R1           ; R1 = R0 = N
          Subtract        #1,R1           ; R1 = N - 1
          Move            R1,-(SP)        ; Push (N-1) on the stack
          Call            FACT            ; Factorial(N-1)
          Move            (SP)+,R1        ; Pop result into R1

          Multiply        R0,R1           ; R1 = N * Fibonacci(N-1)

FI:       Move            R1,8(FP)        ; Place answer on the stack
          MoveMultiple    (SP)+,R0-R1     ; Restore registers
          Move            (SP)+,FP        ; Restore frame pointer
          Return
```