Lab 3: Stacks

Requirements:

1. This assignment as well as other assignments in this class must be finished on Windows operating system.
2. Zip your program(s) submit on Canvas
3. Due 11:59pm Oct. 7, 2013

Assignments:

In Homework 2, you wrote a program to display first 12 Fibonacci numbers. In fact, the definition of Fibonacci numbers is recursive and it is easy to write a recursive function in Java as follows:

```
import java.lang.*;
import java.util.*;

public class fibonacci{
        public static int fib(int n){ // assumes n>=0
            if(n==0 || n==1){
                return n;
             }else{
                return fib(n-1)+fib(n-2);//recursive calls
             }
        }

        public static void main(String args[]){
             for(int i=0;i<7;i++){
                 System.out.println(fib(i));
             }
        }
}
```

In this lab, you are going to write ***a recursive procedure*** in the assembly language to calculate Fibonacci numbers. You are supposed to use the stack to pass parameter and return result. For example, if I want to calculate fibonacci(7), the following program calls the procedure and saves the result in eax

```
push 07h
call fibonacci ; the call to the procedure
pop eax ; eax gets the result
```

In order to access the parameter and result, you need to understand how the stack works.

Table 1

Stack	
Hypothetical Address	Contents
00000080	07h
00000079	
00000078	
00000077	

00000076	return addr
00000075	
00000074	
00000073	

Table 1 shows the stack when procedure ***fibonacci*** is invoked. Now the value of ESP should be 73 and the parameter starts at 77, which is ESP+4. However, if your procedure uses the USES operator as follows:

```
fibonacci PROC USES ESI EAX EBX ECX
        ; codes
        ret
fibonacci ENDP
```

The stack should be substantially different

Table 2

Stack	
Hypothetical Address	Contents
00000080	07h
00000079	
00000078	
00000077	
00000076	return addr
00000075	
00000074	
00000073	
00000072	ESI
00000071	
00000070	
00000069	
00000068	EAX
00000067	
00000066	
00000065	
00000064	EBX
00000063	
00000062	
00000061	
00000060	ECX
00000059	
00000058	
00000057	

Now ESP is 57 and the parameter is at ESP+20. In order to access the actual value of the parameter, you need to use indirect operand, which is [ESP+20] or [ESP+4], depending on individual situation. You should use another register to save the address of the parameter, because ESP changes once you have push/pop instructions and it may cause problems. The following code shows how you can do it

```
        mov esi, esp ; esp→esi
```

```
add esi, 20 ; why 20? See Table 2
mov eax, [esi] ; eax has the value of the parameter
```

In the procedure, the result should be saved to where the parameter was stored. This is implied by these three lines

```
push 07h
call fibonacci ; the call to the procedure
pop eax ; eax gets the result
```

In your procedure, you may find the following two lines useful

```
cmp eax, 2   ; compares eax and 2
jl L1        ; if eax<2, jumps to L1
```

Please write a main procedure to test your Fibonacci procedure by displaying the first 12 Fibonacci numbers.

Grading Policies:

Correctly implement the main procedures	40%
Correctly implement the Fibonacci procedure	40%
Correct output	20%