Last Name: First Name:

Computer Science
C.Sc. 342

Take Home TEST No.2

CSc or CPE

Submit by 11:59 PM, March 22, 2017

Objective:

The objective of this take home test is for students to

1. Run and debug a recursive function call on three different platforms: x86 Intel on Microsoft's Visual Studio, MIPS on MARS Simulator, and on a 64-bit Intel processor running Linux. Display and explain all frames on stack.
2. Measure and plot the time it takes to compute Factorial (N), for N= 10, 100, 1000, 10,000.

Example of a recursive procedure that calculates the factorial of a number and its code in both C and MIPS can be found in the textbook and is shown below.

Create and explain Stack Frames for the recursive function call factorial(5)

```
int factorial (int N)
{
if (N==1)
return 1;
return (N*factorial(N-1));
}
void main()
{
int N_fact=factorial(5);
}
```

1. **Compile and run this program in Debug mode in .NET environment.**
   For each call level display Frame on stack and write down the address on stack and value of
   - **Argument at current level**
   - **local variable (if any) at current level**
   - **return address at current level**
   - **EIP**
   - **EBP**
   - **ESP**

You may use arrow to point a specific location on stack frame.

At the end of calls you should display 5 frames on the stack as shown in FIGURE 1.

Last Name: First Name:

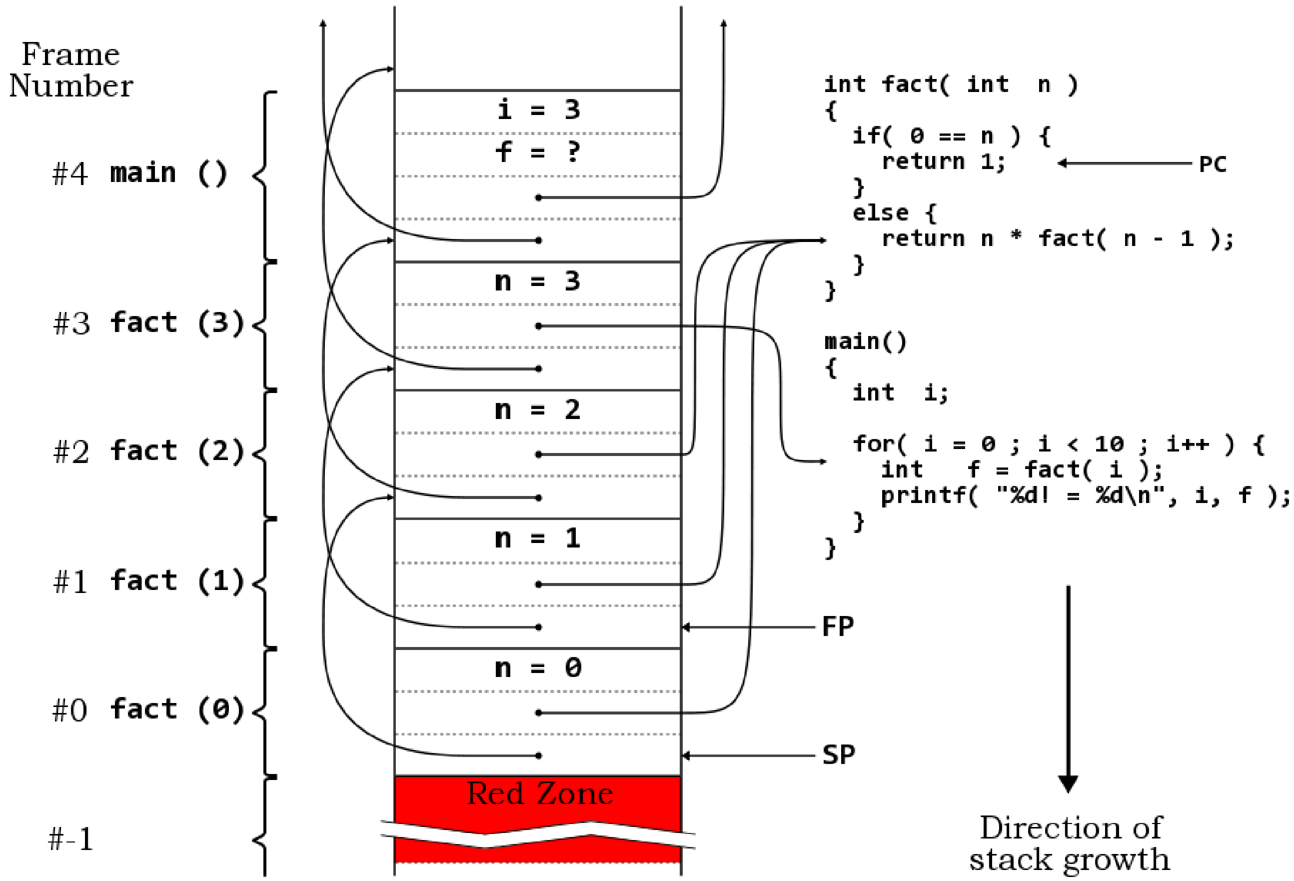


FIGURE 1. All arrows have to show labels to addresses on stack and corresponding values.

Please explain the return process – specify instructions and arguments used at each nested level when returning.

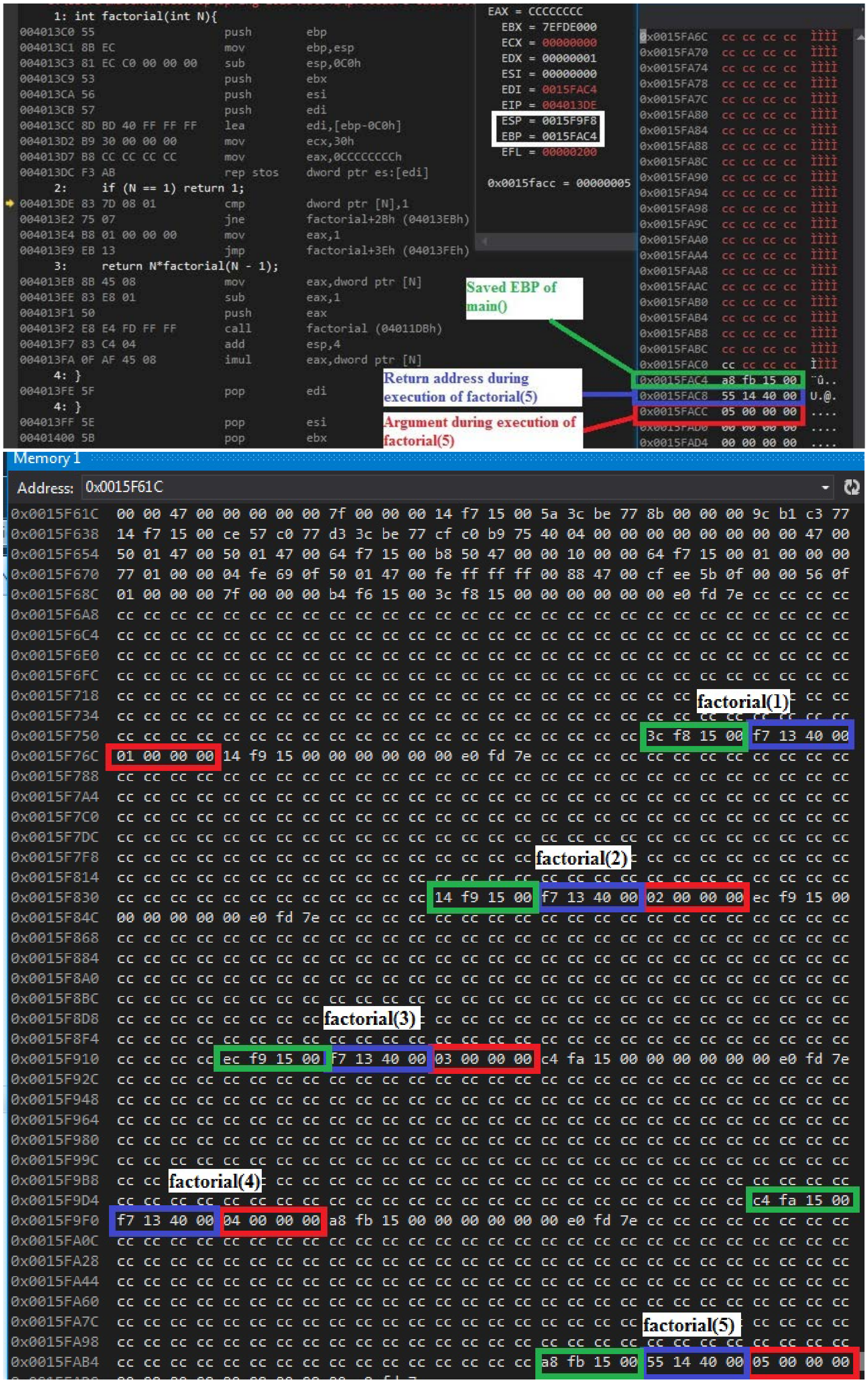
2. **(Optional) Create a lean version of the factorial() function. Instead of using CALL instruction (generated by compiler), create function call using similar to JAL instruction in MIPS - save the return address and then jump to function. Do not push and pop unnecessary information on stack (such as registers ebx, ecx, etc.) on stack.**

3. **Please repeat Section 1 using MIPS instructions and run the program on a simulator MARS. You can use example described in the section on nested procedure calls in the textbook.**

4. **Please repeat Section 1 using GCC, GDB in LINUX environment, and run the program in command mode using GDB. You can use example described in the section on nested procedure calls in the textbook.**

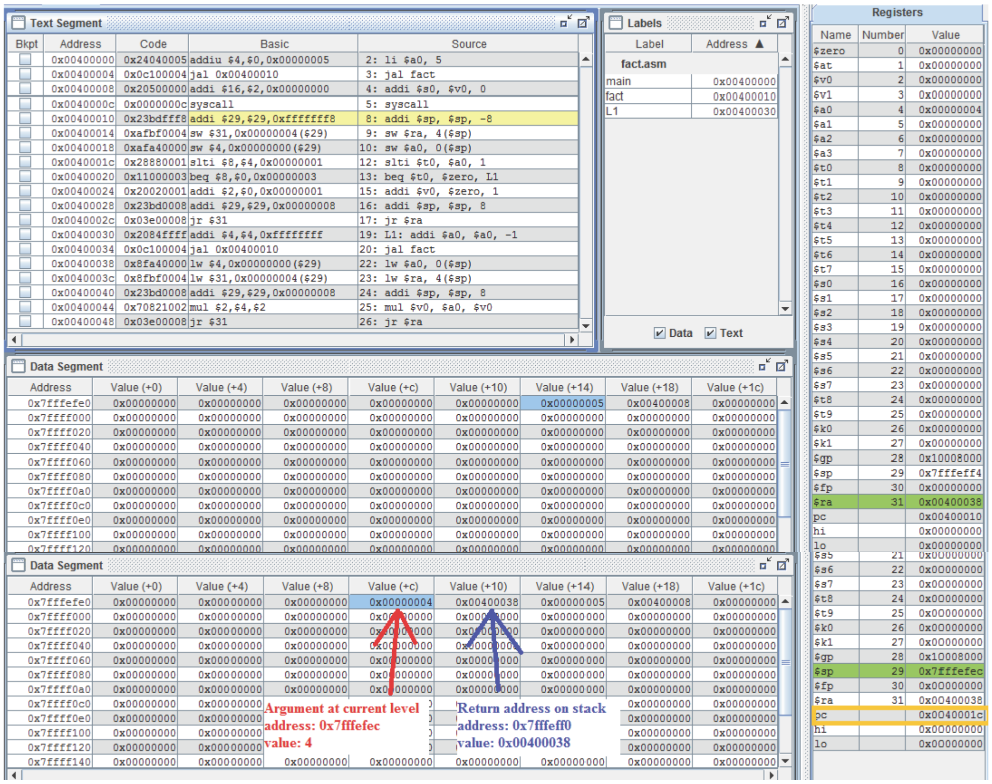
Last Name: First Name:

Sample screenshots for X86, MS Visual Studio in Debug mode



Last Name: First Name:

Sample screenshots for MIPS, Simulator MARS environment



Last Name: First Name:

Sample screenshots for 64 bit Intel processor, GDB

```
=> 0x00000000004004f6 <+0>:     push   %rbp
   0x00000000004004f7 <+1>:     mov    %rsp,%rbp
   0x00000000004004fa <+4>:     sub    $0x10,%rsp
   0x00000000004004fe <+8>:     mov    %edi,-0x4(%rbp)
   0x0000000000400501 <+11>:    cmpl   $0x1,-0x4(%rbp)
   0x0000000000400505 <+15>:    jne    0x40050e <factorial(int)+24>
   0x0000000000400507 <+17>:    mov    $0x1,%eax
   0x000000000040050c <+22>:    jmp    0x40051f <factorial(int)+41>
   0x000000000040050e <+24>:    mov    -0x4(%rbp),%eax
   0x0000000000400511 <+27>:    sub    $0x1,%eax
   0x0000000000400514 <+30>:    mov    %eax,%edi
   0x0000000000400516 <+32>:    callq  0x4004f6 <factorial(int)>
   0x000000000040051b <+37>:    imul   -0x4(%rbp),%eax
   0x000000000040051f <+41>:    leaveq
   0x0000000000400520 <+42>:    retq
End of assembler dump.
(gdb) nexti 3
0x00000000004004fe      1       int factorial(int N){
1: x/i $pc
=> 0x4004fe <factorial(int)+8>: mov    %edi,-0x4(%rbp)
(gdb) printf "rbp:%x\nrsp:%x\n",$rbp,$rsp
rbp:ffffdde0
rsp:ffffddd0
(gdb)
```