Last Name: First Name:

Computer Science
C.Sc. 342

Take Home TEST
CSc or CPE

Submit by to TA by 12:00 PM, May 1, 2023

Objective:

The objective of this take-home test is to demonstrated understanding how recursive function calls allocate memory, how recursive functions are executed, how control is transferred from one level to the next, how parameters are transferred form one level to the next.

1. First, study the textbook section 2.8 “Supporting procedures in computer hardware”.
2. Second, Implement the tutorial example on factorial recursive function calls as shown below:
   2.1.1 Run and debug a recursive function calls on three different platforms:
   a. x86 Intel using Microsoft's Visual Studio,
   b. MIPS on MARS Simulator, and
   c. 64-bit I7 (I5 or I3) processor running Linux (or MAC with I7 or M1 processor).
   d. Display and explain all frames on stack.
3.
   a. Measure and plot the time it takes to compute Factorial (N), for N= 10, 100, 1000, 10,000.
   b. Repeat tutorial example 1 and 2 to compute ***GCD(a,b) using recursive version of EUCLEDEAN algorithm for two integers a>0, b>0. To refresh*** GCD(a,b) computation please refer to last 3 pages of this assignment.
4. **What to Submit:** report, working project files and how to use it.

! " ! # $%& ' () * & + , ')(**of a recursive procedure t**hat calculates the factorial of a number and its code in both C and MIPS can be found in the textbook section 2.8 Implement 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.

Last Name: First Name:

For each !"##$#%&%# 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.

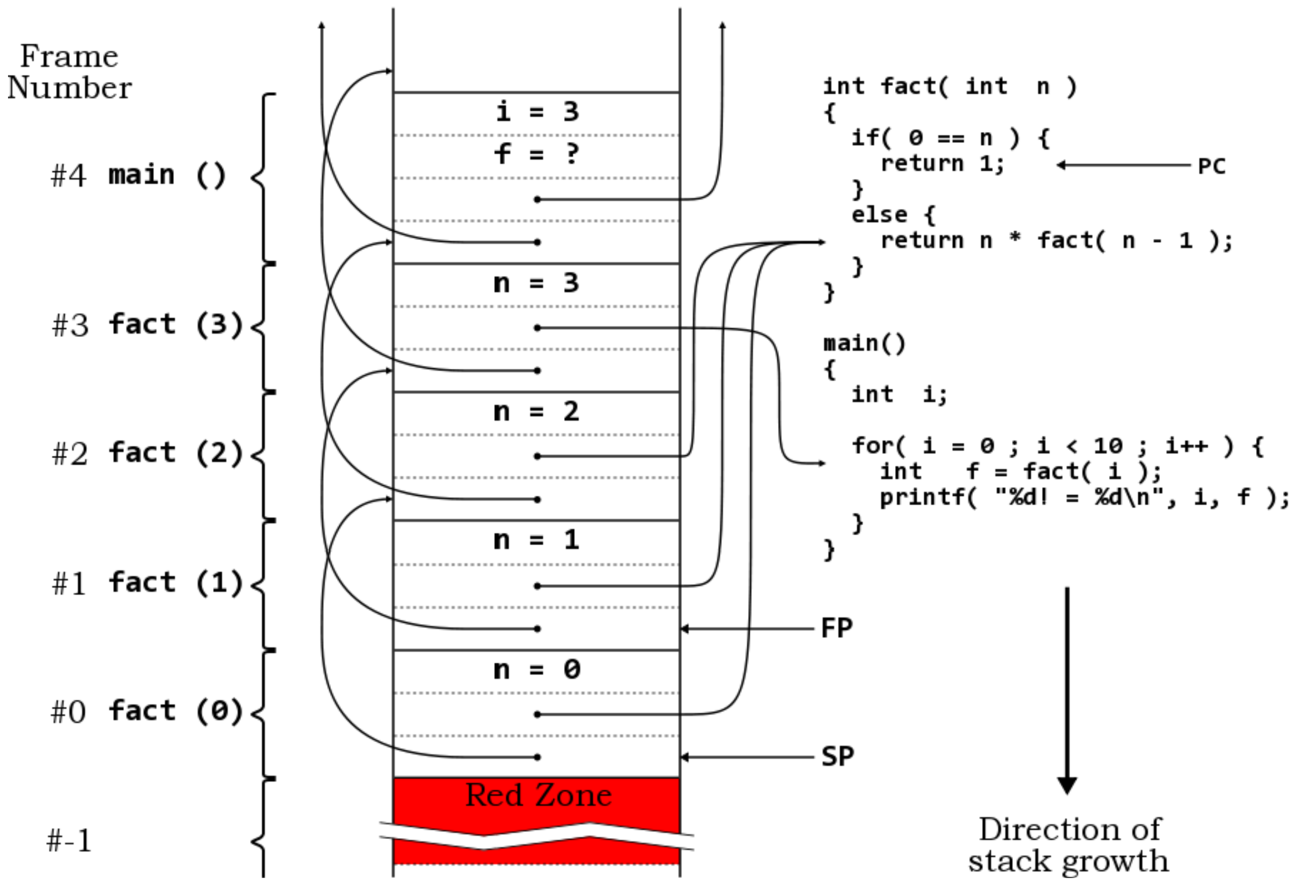


FIGURE 1. All arrows have to show labels to addresses on stack and corresponding values.

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.**

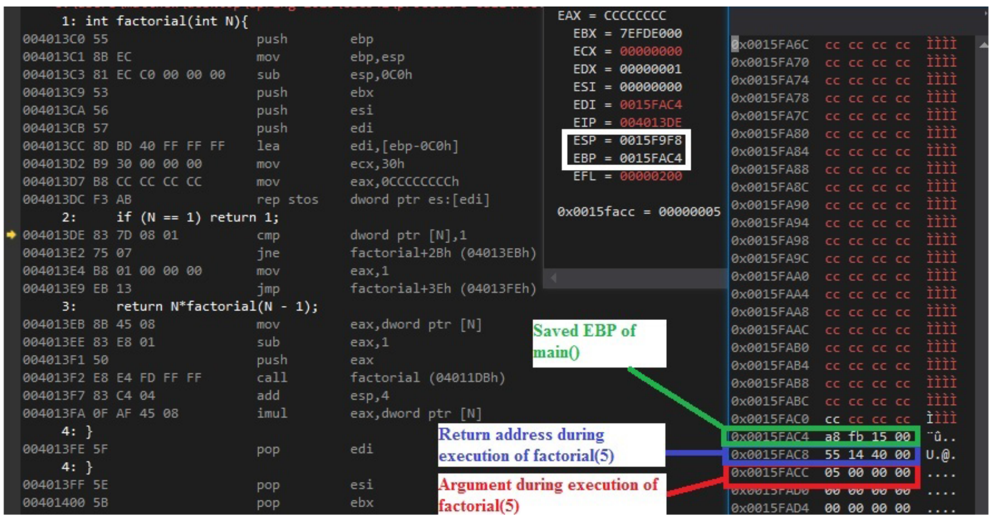
Last Name: First Name:

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.**

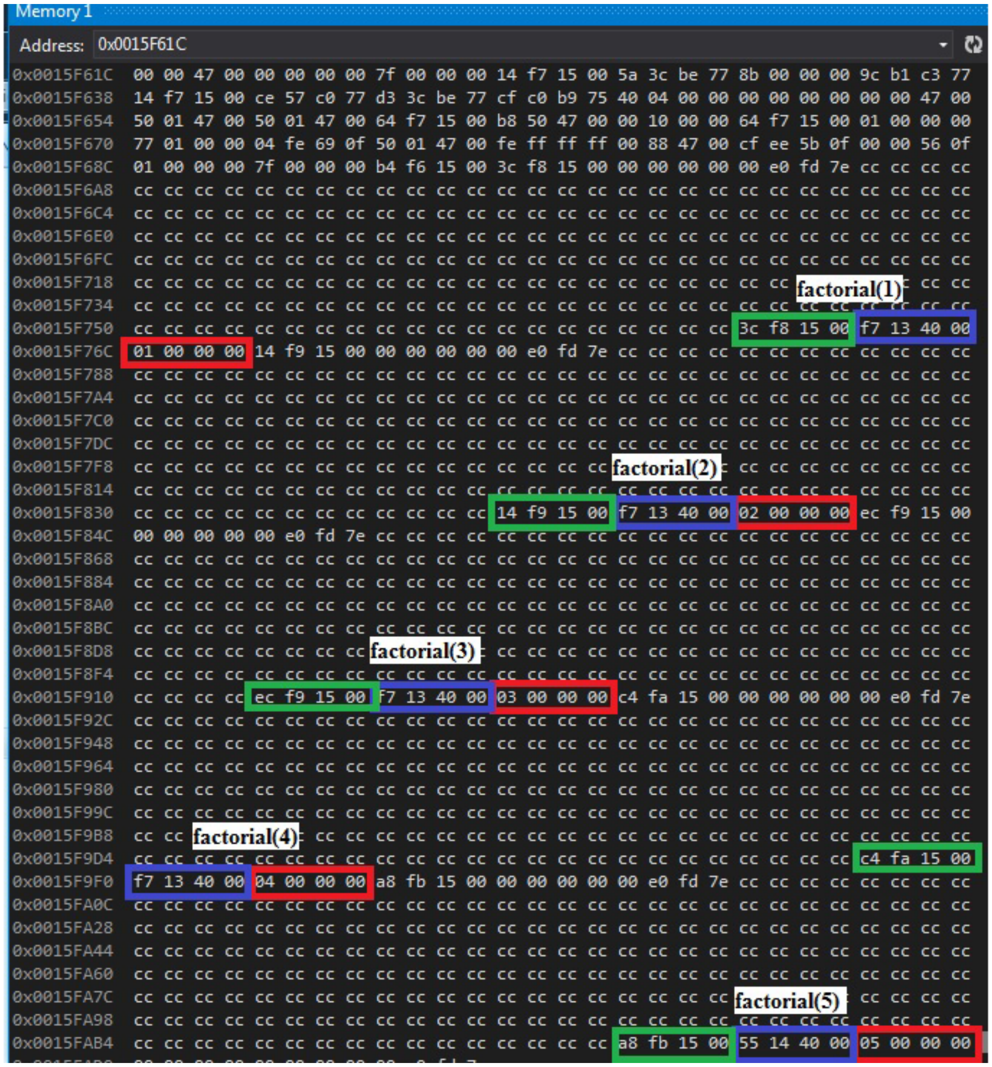
Sample screenshots for X86, MS Visual Studio in Debug mode

! !



!

Last Name: First Name:



!

!

!

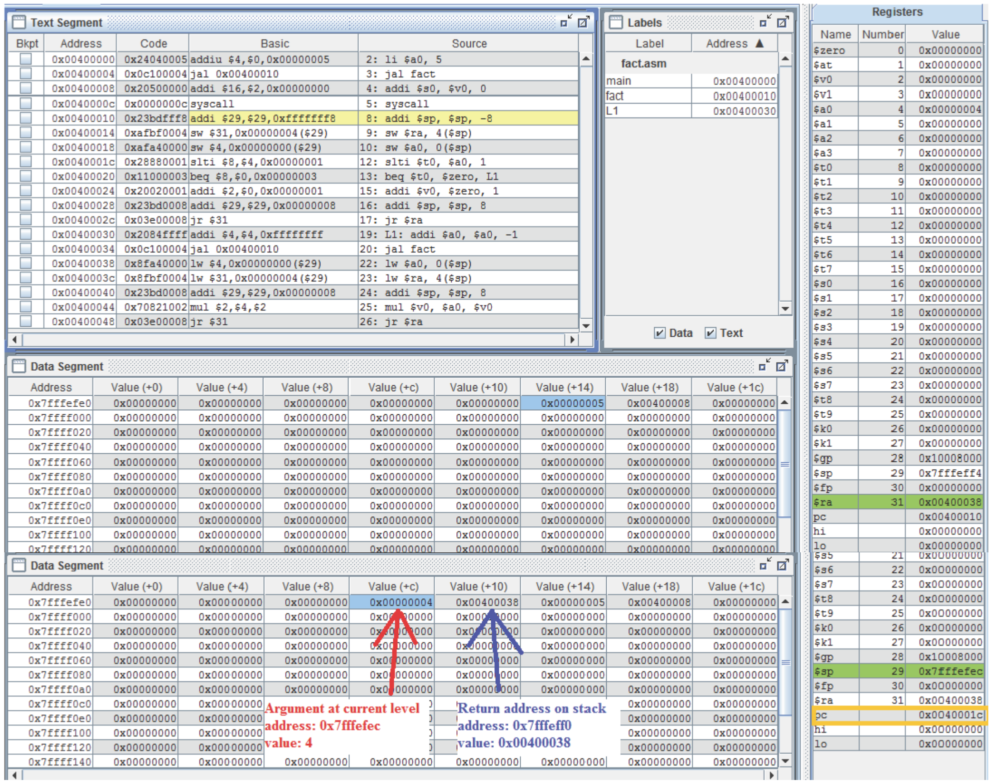
!

!

Sample screenshots for MIPS, Simulator MARS environment

!

Last Name: First Name:

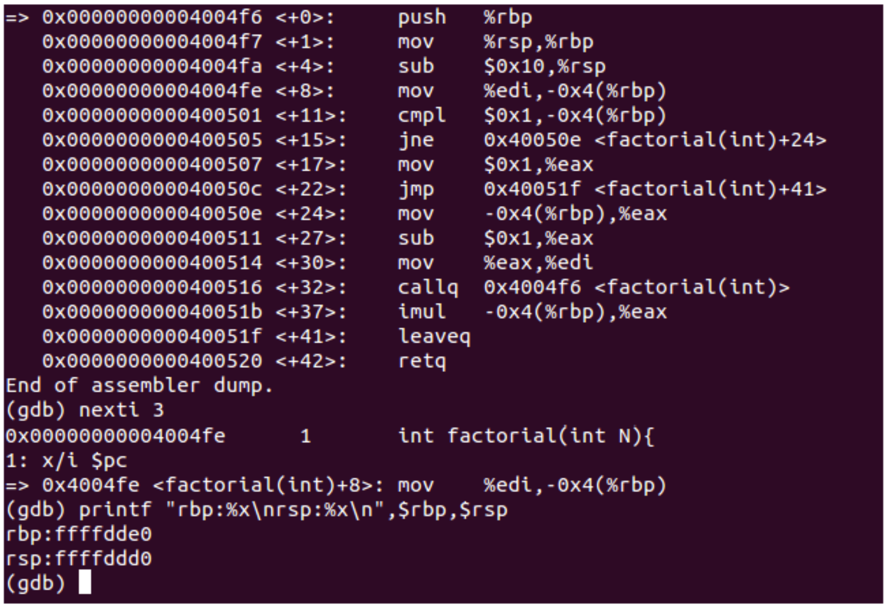


Last Name: First Name:

Sample screenshots for 64 bit Intel processor, GDB

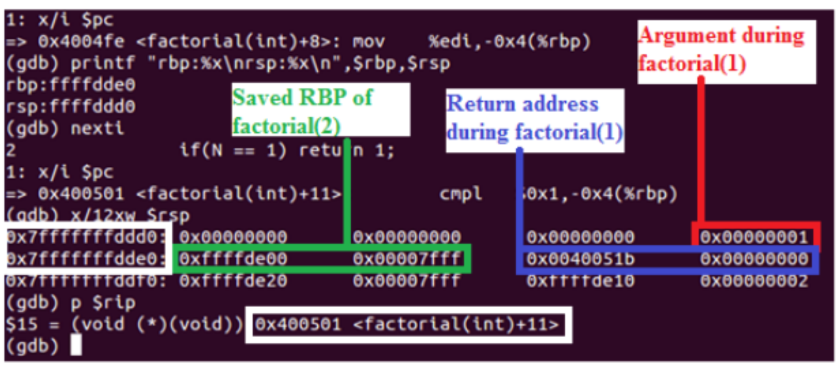
!

!



!

!



!

!"#$%&%'()*+$!,*-.+!/$

$

$

$

$

(01203$45$67#$89:4;2<=>?$

$

3.2. The Euclidean Algorithm

3.2.1. The Division Algorithm. The following result is known as *The Division Algorithm*:[1] If $a, b \in \mathbb{Z}$, $b > 0$, then there exist unique $q, r \in \mathbb{Z}$ such that $a = qb + r$, $0 \leq r < b$. Here q is called *quotient* of the *integer division* of a by b, and r is called *remainder*.

3.2.2. Divisibility. Given two integers a, b, $b \neq 0$, we say that b *divides* a, written $b|a$, if there is some integer q such that $a = bq$:

$$b|a \Leftrightarrow \exists q,\ a = bq\,.$$

We also say that b *divides* or is a *divisor of* a, or that a is a *multiple* of b.

3.2.3. Prime Numbers. A *prime* number is an integer $p \geq 2$ whose only positive divisors are 1 and p. Any integer $n \geq 2$ that is not prime is called *composite*. A non-trivial divisor of $n \geq 2$ is a divisor d of n such that $1 < d < n$, so $n \geq 2$ is composite iff it has non-trivial divisors. *Warning*: 1 is not considered either prime or composite.

Some results about prime numbers:

1. For all $n \geq 2$ there is some prime p such that $p|n$.
2. (Euclid) There are infinitely many prime numbers.
3. If $p|ab$ then $p|a$ or $p|b$. More generally, if $p|a_1 a_2 \dots a_n$ then $p|a_k$ for some $k = 1, 2, \dots, n$.

3.2.4. The Fundamental Theorem of Arithmetic. Every integer $n \geq 2$ can be written as a product of primes uniquely, up to the order of the primes.

It is customary to write the factorization in the following way:

$$n = p_1^{s_1}\, p_2^{s_2} \dots p_k^{s_k}\,,$$

where all the exponents are positive and the primes are written so that $p_1 < p_2 < \cdots < p_k$. For instance:

$$13104 = 2^4 \cdot 3^2 \cdot 7 \cdot 13\,.$$

[1] The result is not really an "algorithm", it is just a mathematical theorem. There are, however, algorithms that allow us to compute the quotient and the remainder in an integer division.

3.2.5. Greatest Common Divisor. A positive integer d is called a *common divisor* of the integers a and b, if d divides a and b. The greatest possible such d is called the *greatest common divisor* of a and b, denoted $\gcd(a, b)$. If $\gcd(a, b) = 1$ then a, b are called *relatively prime*.

Example: The set of positive divisors of 12 and 30 is $\{1, 2, 3, 6\}$. The greatest common divisor of 12 and 30 is $\gcd(12, 30) = 6$.

A few properties of divisors are the following. Let m, n, d be integers. Then:

1. If $d|m$ and $d|n$ then $d|(m + n)$.
2. If $d|m$ and $d|n$ then $d|(m - n)$.
3. If $d|m$ then $d|mn$.

Another important result is the following: Given integers a, b, c, the equation

$$ax + by = c$$

has integer solutions if and only if $\gcd(a, b)$ divides c. That is an example of a *Diophantine equation*. In general a Diophantine equation is an equation whose solutions must be integers.

Example: We have $\gcd(12, 30) = 6$, and in fact we can write $6 = 1 \cdot 30 - 2 \cdot 12$. The solution is not unique, for instance $6 = 3 \cdot 30 - 7 \cdot 12$.

3.2.6. Finding the gcd by Prime Factorization. We have that $\gcd(a, b) =$ product of the primes that occur in the prime factorizations of both a and b, raised to their lowest exponent. For instance $1440 = 2^5 \cdot 3^2 \cdot 5$, $1512 = 2^3 \cdot 3^3 \cdot 7$, hence $\gcd(1440, 1512) = 2^3 \cdot 3^2 = 72$.

Factoring numbers is not always a simple task, so finding the gcd by prime factorization might not be a most convenient way to do it, but there are other ways.

3.2.7. The Euclidean Algorithm. Now we examine an alternative method to compute the gcd of two given positive integers a, b. The method provides at the same time a solution to the Diophantine equation:

$$ax + by = \gcd(a, b)\,.$$

It is based on the following fact: given two integers $a \geq 0$ and $b > 0$, and $r = a \bmod b$, then $\gcd(a, b) = \gcd(b, r)$. Proof: Divide a by

b obtaining a quotient q and a remainder r, then

$$a = bq + r\,, \quad 0 \le r < b\,.$$

If d is a common divisor of a and b then it must be a divisor of $r = a - bq$. Conversely, if d is a common divisor of b and r then it must divide $a = bq + r$. So the set of common divisors of a and b and the set of common divisors of b and r are equal, and the greatest common divisor will be the same.

The Euclidean algorithm is a follows. First we divide a by b, obtaining a quotient q and a remainder r. Then we divide b by r, obtaining a new quotient q' and a remainder r'. Next we divide r by r', which gives a quotient q'' and another remainder r''. We continue dividing each remainder by the next one until obtaining a zero remainder, and which point we stop. The last non-zero remainder is the gcd.

Example: Assume that we wish to compute $\gcd(500, 222)$. Then we arrange the computations in the following way:

$$\begin{array}{rcll} 500 & = & 2 \cdot 222 + 56 & \rightarrow \; r = 56 \\ 222 & = & 3 \cdot 56 + 54 & \rightarrow \; r' = 54 \\ 56 & = & 1 \cdot 54 + 2 & \rightarrow \; r'' = 2 \\ 54 & = & 27 \cdot 2 + 0 & \rightarrow \; r''' = 0 \end{array}$$

The last nonzero remainder is $r'' = 2$, hence $\gcd(500, 222) = 2$. Furthermore, if we want to express 2 as a linear combination of 500 and 222, we can do it by working backward:

$$\begin{aligned} 2 &= 56 - 1 \cdot 54 = 56 - 1 \cdot (222 - 3 \cdot 56) = 4 \cdot 56 - 1 \cdot 222 \\ &= 4 \cdot (500 - 2 \cdot 222) - 1 \cdot 222 = 4 \cdot 500 - 9 \cdot 222\,. \end{aligned}$$

The algorithm to compute the gcd can be written as follows:

```
procedure gcd(a,b)
  if a<b then   {make a the largest}
    swap(a,b)
  while b ≠ 0
    begin
      r := a mod b
      a := b
      b := r
    end
  return a
end gcd
```

The next one is a recursive version of the Euclidean algorithm:

```
procedure gcd_recurs(a,b)
  if b=0 then
    return a
  else
    return gcd_recurs(b,a mod b)
end gcd_recurs
```