Introduction to Computer Science
HW #2
Due: 2014/03/26

Homework Rules:

Hand-written homework can be handed in in class. Otherwise, you may contact the TA in advance and then bring the hardcopy to the TA in BL421.

As for the programming part, you need to upload it to CEIBA before the deadline (2014/03/13 3am). The file you upload must be a **.zip** file that contains the following files:

README.txt

HW01_b02901XXX (a folder that contains all .cpp & .h as required),

1. Do not submit executable files (.exe) or objective files (.o, .obj). Files with names in wrong format will not be graded. You must **remove any system calls**, such as system ("pause"), in your code if you use it.
2. In README.txt, you need to describe which compiler you used in this homework and how to compile it (if it is in a "project" form).
3. In your .cpp files, we suggest you write comments as detailed as you can. If your code does not work properly, code with comments earns you more partial credits.

Chapter 2 Review Problem (30%)

Problems 15, 24, 35

Chapter 3 Review Problem (30%)

Problems 18, 35, 40

Programming Problem (40%)

x86 inline assembly - Reverse order

In class, we have learned how CPUs work with simple assembly as example. Here we are going to try writing "real" assembly running on Intel x86 CPUs. Note that the syntax for **assembly is not standard of C/C++**, many compilers have their own syntaxes dealing inline assembly. (So it is not portable for different compilers!!) In this problem, we are going to write a function with **inline assembly** in GNU g++ that does:

Reverse the digital order of a **10-based** number.

For example: input(1234) -> output(4321)
input(1200) -> output(21) //the leading 0s are redundant
input(0xD) -> output(31) //since 0xD=13, so output 31(0x1F)

(1) The input number would always be in the range of 0~999999.
(2) It is a function, we might call it for many times. Also, you need to follow the **coding rules**.

Introduction to Computer Science
HW #2
Due: 2014/03/26

1. How to start? (GNU g++ inline assembly guide)

First, you need to setup the GNU g++ compiler on a machine with Intel CPU. If you don't have such machine, go to our *Computer and Information Networking Center*. As for GNU g++, you can download MinGW and setup your environment variables (Dev-C++ or Code::Blocks work too). Reference for you: http://shunyuan-chou.blogspot.tw/2010/02/3-mingw-msys.html

Second, build and run the "gcd_ref.cpp" as an example to check if your machine is ready.

Third, learn how to write basic x86 inline assembly. Here's a perfect resource for you: http://www.cs.virginia.edu/~evans/cs216/guides/x86.html. We suggest you to learn from examples. Check out our "gcd_ref.cpp" as a starter, and refer to the above website for more details. If you still have doubts, TA will write some guides later.

Fourth, have a look at our **target file: "reverse_order.cpp"**, you need to implement the reverse_asm function. Also, we've written an evaluate function for you, which may come in handy for debugging. If you pass, that means you would get full credits for this programming problem.

Last but not least, enjoy writing assembly and remember to submit it on time. ^^

2. Hint:

GNU g++ inline assembly use the keyword: asm. Here's a little example:

```
#include <cstdio>
#include <ctime>
int reverse_asm(int num){
    int result;
    asm volatile(
        ".intel_syntax;\n"//tell compiler using Intel syntax
    // ===== add your codes here =====
        " mov   eax,    %1  ;\n"//register eax = num
        " mov    %0,   eax  ;\n"//result = register eax
        //each instruction end with ";\n"
    // ==============================
        ".att_syntax;\n"//using AT&T syntax to pass variables
        : "=r"(result) //output operands
        : "r"(num) //input operands
        : "%eax","%ebx","%ecx","%edx"//register used
    );
    return result;
}
```

Maybe you need to use memory for storing something, the "push" and "pop" instructions can serve it. (Or pointer, passing variables/arrays as operands both ok)

3. Coding rules:

You **MUST follow these coding rules**:

(1) Here are many kinds of assembly, but you can **only** using **Intel x86 inline assembly**, while compiling and checking on **GNU g++**.

(2) You can modify "reverse_asm" function, it is OK. But don't touch the input/output argument and function name of it.

```
int reverse_asm(int num); //keep this function signature
```

(3) Maybe you would add something for debugging, but please keep other codes outside "reverse_asm" function the same as you download when you submit it.

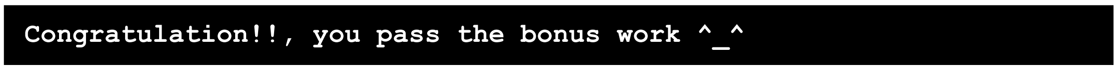
Bonus (5%)

Write it in a faster way

Maybe you can come up with some tricks to **beat GNU compiler**!! Here's bonus 5% for those who can run the asm function faster than the cpp one. We'll **measuring this by the evaluate function** (for all the cases in average).

Notes: Just beat the basic compiler setting is enough (-O0). You don't need to spend lots of time fighting with -O1, -O2. (But if you can, TA will worship you…)

If you meet the bonus requirements, write "I finish the bonus part." in the readme file to let TA know. We know that there are some difference machine by machine, you can write down the time measured by yourself. In case that it runs a little bit slower on TA's machine, we still can give you the bonus points.



```
Congratulation!!, you pass the bonus work ^_^
```