Introduction to Computer Science
HW #2
Due: 2016/04/06

Homework Rules:

Hand-written homework can be handed in **before lecture starts**. Otherwise, you may contact the TA in advance and then bring the hardcopy to the TA in BL-603 (please send e-mail in advance).

As for the programming part, you need to upload it to CEIBA before the deadline (2016/04/07 3am). The file you upload must be a **.zip** file that contains the following files:

README.txt

HW02_b04901XXX (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 any.
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.

What to submit:

base10.cpp

Your choice of 7 out of 8 problems (total 49%)

Chapter 2 Review Problems (7 pts each)

Problems 14, 17, 24*, 35.

Chapter 3 Review Problems (7 pts each)

Problems 26, 34, 38, 42.

Programming Problem (51%)

1. x86 inline assembly – Binary string to Decimal

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

Introduction to Computer Science
HW #2
Due: 2016/04/06

Show a number's **10-based** digits given its **binary representation in string**.

For example:

input("11") → since $(11)_2=(3)_{10}$, so → output($(3)_{10}$)

input("101") → since $(101)_2=(5)_{10}$, so → output($(5)_{10}$)

input("1101") → since $(1101)_2=(13)_{10}$, so → output($(13)_{10}$)

(1) The input number would always be in the range of 0~9,999,999. (unsigned)最多20bytes

(2) It is a function, and we might call it many times. Also, you need to follow the coding rules.

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

First, you need to setup the GNU g++ compiler on a machine with Intel x86 CPU. If you don't have a such machine, go to our *Computer and Information Networking Center*. As for GNU g++, you can download MinGW and setup your environment variables. Reference for you: http://shunyuan-chou.blogspot.tw/2010/02/3-mingw-msys.html

TA recommends using Codeblocks.

Second, build and run the "gcd_ref.cpp" as an example to check whether your machine is well prepared or not.

Third, learn how to write basic x86 inline assembly. Here's a perfect resources for you: http://www.cs.virginia.edu/~evans/cs216/guides/x86.html. We suggest you to learn from examples. Imitate our "gcd_ref.cpp" as starter, and refer to the website above for more details.

Fourth, have a look at our **target file: "base10.cpp"**, you need to fulfill the base10_asm function. And, we've written an evaluate function for you to debug. If you pass, that means you would get the full points of this programming problem.

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

Introduction to Computer Science
HW #2
Due: 2016/04/06

Hint:

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

```
#include <cstdio>
#include <cstdlib>
#include <ctime>
#include <cstring>
int base10_asm(const chart* a){
    int result;
    asm volatile(
        ".intel_syntax noprefix;\n"// 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
    );
```

If you need to use memory for storing something, the "push" and "pop" instructions come in handy. (Or pointer, passing variables/arrays as operands both ok)

Important 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 "base10_asm" function (e.g. add some local variables), it is ok. But don't touch the input/output argument and function name of it.

```
int base10_asm(const char* a); //keep this function signature
```

(3) Maybe you would add something for debugging, but please keep other codes outside "base10_asm" function the same as you download when you submit it.

Introduction to Computer Science
HW #2
Due: 2016/04/06

2. [Bonus] 5% Write it in a faster way

Maybe you can come up with some tricks to **beat our 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:
Beat the compiler setting –O2: you can get 5% points

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 differences machine by machine, you can write down the time result measured 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 ^_^
```