Introduction to Computer Science
HW #2
Due: 2015/04/08

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_b03901XXX (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 (18:8%, 24:16%, 35:8%)

18. Suppose the memory cells at addresses F0 through FD in the machine described in Appendix C contain the following (hexadecimal) bit patterns:

Address	Contents
F0	20
F1	00
F2	22
F3	02
F4	23
F5	04
F6	B3
F7	FC
F8	50
F9	02
FA	B0
FB	F6
FC	C0
FD	00

If we start the machine with its program counter containing F0, what is the value in register 0 when the machine finally executes the halt instruction at location FC?

24. A game that used to be popular among computer hobbyists is core wars—a variation of battleship. (The term *core* originates from an early memory technology in which 0s and 1s were represented as magnetic fields in little rings of magnetic material. The rings were called cores.) The game is played between two opposing programs, each stored in different locations of the same computer's memory. The computer is assumed to alternate between the two programs, executing an instruction from one followed by an instruction from the other. The goal of each program is to cause the other to malfunction by writing extraneous data on top of it; however, neither program knows the location of the other.

a. Write a program in the machine language of Appendix C that approaches the game in a defensive manner by being as small as possible.

b. Write a program in the language of Appendix C that tries to avoid any attacks from the opposing program by moving to different locations. More precisely, beginning at location 00, write a program that will copy itself to location 70 and then jump to location 70.

Introduction to Computer Science
HW #2
Due: 2015/04/08

*35. Identify both the mask and the logical operation needed to accomplish each of the following objectives:
a. Put 1s in the upper4 bits of an 8-bit pattern without disturbing the other bits.
b. Complement the most significant bit of an 8-bit pattern without changing the other bits.
c. Complement a pattern of 8 bits.
d. Put a 0 in the least significant bit of an 8-bit pattern without disturbing the other bits.
e. Put 1s in all but the most significant bit of an 8-bit pattern without disturbing the most significant bit.

Hint of 2.24:

(a) 可以讓 processor 不斷執行而不 halt 且程式越短越好。

(b) 比較容易的做法是：不要當作程式在搬程式，當作他是把記憶體 00~30 的內容用迴圈搬到 70~A0，之後把 30 代換成程式的長度即可。

Chapter 3 Review Problem (30:8%, 35:10%, 42:8%)

30. List in chronological order the major events that take place when a process is interrupted.

*42. The following is the "dining philosophers" problem that was originally proposed by E. W. Dijkstra and is now a part of computer science folklore.

Five philosophers are sitting at a round table. In front of each is a plate of spaghetti. There are five forks on the table, one between each plate. Each philosopher wants to alternate between thinking and eating. To eat, a philosopher requires possession of both the forks that are adjacent to the philosopher's plate.

Identify the possibilities of deadlock and starvation (see Problem 40) that are present in the dining philosophers problem.

Definition of starvation:

A process that is waiting for a time slice is said to suffer **starvation** if it is never given a time slice.

*35. Students who want to enroll in Model Railroading II at the local university are required to obtain permission from the instructor and pay a laboratory fee. The two requirements are fulfilled independently in either order and at different locations on campus. Enrollment is limited to twenty students; this limit is maintained by both the instructor, who will grant permission to only twenty students, and the financial office, which will allow only twenty students to pay the laboratory fee. Suppose that this registration system has resulted in nineteen students having successfully registered for the course, but with the final space being claimed by two students—one who has only obtained permission from the instructor and another who has only paid the fee. Which requirement for deadlock is removed by each of the following solutions to the problem?
a. Both students are allowed in the course.
b. The class size is reduced to nineteen, so neither of the two students is allowed to register for the course.
c. The competing students are both denied entry to the class and a third student is given the twentieth space.
d. It is decided that the only requirement for entry into the course is the payment of the fee. Thus the student who has paid the fee gets into the course, and entry is denied to the other student.

Introduction to Computer Science
HW #2
Due: 2015/04/08

Programming Problem (52%)

1. x86 inline assembly – Decimal to Octal

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:

Show a given number's **8-based** digits by **10-based** number.

For example:

input($(23)_{10}$) → since $(23)_{10}=(27)_{8}$, so → output($(27)_{10}$)

input($(1024)_{10}$) → since $(1024)_{10}=(2000)_{8}$, so → output($(2000)_{10}$)

input($(1234)_{10}$) → since $(1234)_{10}=(2322)_{8}$, so → output($(2322)_{10}$)

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

(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

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: "base8.cpp"**, you need to fulfill the base8_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: 2015/04/08

Hint:

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

```
#include <cstdio>
#include <ctime>
int base8_asm(int num){
    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
    );
    return result;
}
```

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 "base8_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 base8_asm(int num); //keep this function signature
```

(3) Maybe you would add something for debugging, but please keep other codes outside "base8_asm" function the same as you download when you submit it.

Introduction to Computer Science
HW #2
Due: 2015/04/08

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 -O1: you can get 3% points; the -O2: you can get 5% points.

If you meet the bonus requirements, write "I finish the bonus part(O1/O2)." in the readme file to let TA know. We know that there are some difference 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 ^_^
```