This activity will get you familiar with performance measurement, instruction set architecture, and compiler. It should give you a general idea on how a compiler optimizes software. In particular, you will learn how architects measure performance of a system.

Prologue

We will use the free c compiler from the GNU free compiler collection (GCC) project. If you uses Linux or Mac OS X, be sure to install appropriate packets. (For Mac OS X, you may install X-code command-line tool.)

(If you do not use Windows, you may skip this part.)

For Windows, Cygwin, a linux-like environment for Windows, can be used. If you want to use Cygwin, please go to http://www.cygwin.com/ and download the installation software (setup.exe). Note that you need an internet connection for the installation.

To install Cygwin, run "setup.exe". The program will ask for a mirror site for installation, and pick an appropriate choice (e.g. a mirror in Japan or Asia). There are only two main components required for this exercise: **gcc** and **bin-utils**. Other choices are optional. I recommend that you install pico or vi for use as your text editor. The installation may take hours (depending on your internet connection).

After the installation, you will find a program group "cygwin" in your Windows start menu. Run the "cygwin".

Now, let's test that gcc is working by compiling a simple program. This is "hello.c". Use your favorite editor (e.g. notepad) to write this program. Note that all cygwin paths are relative to c:\cygwin\ (i.e. /home in cygwin will be c:\cygwin\home in Windows).

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    printf ("Hello, World\n");
}
```

run **gcc -o hello hello.c**
The program will give hello (or hello.exe on Windows). Type **./hello** (or ./hello.exe on Cygwin)to run your program. To disassembler, try **objdump -d**

hello.exe. (For Mac OS X, try **otool -tV hello**) I know that assembly may be geek to you, but that is not an issue (at least for now).

To compile a program with optimization level (between 0 and 3), use -O[level] as a parameter. (e.g. **gcc -O3 -o hello hello.c**.) Please note that clang-based compiler (e.g. X-code on Mac OS X) only supports level 1 (-O1) optimization.

To compile a program with immediate assembly result, use -S as a parameter. (e.g. **gcc -S hello.c**) The assembly will have **.s** as extension (e.g. **hello.s.**)

If you get it all working, you are ready for the exercise.

Exercises

1. (Instruction Analysis) This exercise will familiarize you with several aspects of the instruction set and the fundamentals of the compiler. Given ***max.c*** (below), please use “gcc -S max.c” to compile the code into assembly code. (The result will be in ***max.s***.) From the result, answer the following questions.

```
int max1(int a, int b) {       return (a>b)?a:b;
}

int max2(int a, int b) {
      int isaGTb=a>b;
      int max;
      if (isaGTb)
            max=a;
      else
            max=b;
      return max;
}
```

   What does the code hint about the kind of instruction set? (e.g. Accumulator, Register Memory, Memory Memory, Register Register) Please justify your answer.

Can you tell whether the architecture is either **Restricted Alignment** or **Unrestricted Alignment**? Please explain how you come up with your answer.

Create a new function (e.g. testMax) to call *max1*. Generate new assembly code. What does the result suggest regarding the register saving (caller save vs. callee save)? Please provide your analysis.

How do the arguments be passed and the return value returned from a function? Please explain the code.

Find the part of code (snippet) that does comparison and conditional branch. Explain how it works.

If max.c is compiled with optimization turned on (using “gcc -O2 -S max.c”), what are the differences that you may observe from the result (as compared to that without optimization). Please provide your analysis

Please estimate the CPU Time required by the *max1* function (using the equation CPI=ICxCPIxTc). If possible, create a main function to call max1 and use the **time** command to measure the performance. Compare the measure to your estimation. What do you think are the factors that cause the difference? Please provide your analysis.
(You may find references online regarding the CPI of each instruction.)

2. (Optimization) We will use simple fibonacci calculation as a benchmark. Please measure the execution time (using the

time command) of this given program when compiling with optimization level 0 (no optimization), level 1, level 2 and level 3. (Note that some compilers do similar optimization for all level 1, level 2 and level 3. If that is the case, you will see no difference after level 1.) You may want to run each program a few times and use the average value as a result.

```
#include <stdio.h>

long fibo(long a) {
	if (a<=0L) {
		return 0L;
	}
	if (a==1L) {
		return 1L;
	}
	return fibo(a-1L)+fibo(a-2L);
}

int main (int argc,char *argv[]) {
	for (long i=1L;i<45L;i++) {
		long f=fibo(i);
		printf("fibo of %ld is %ld\n",i,f);
	}
}
```

3. (Analysis) As suggested by the results in Exercise 2, what kinds of optimization are used by the compiler in each level in order to make the program faster? To answer this question, use “gcc -S” to generate the assembly code for each level (e.g. “gcc -S -O2 fibo.c”) and use this result as a basis for your analysis.
   (Depending on your version of the compiler, the result may vary.)