

Homework 2

Decompiling Intel Assembly Language

In this homework, you will examine assembler output from gcc in order to determine what the original C code was. Please note that it is easy to spend a lot of time to go down the rabbit hole in this homework. Keep in mind that it is only a tiny part of your grade---the labs are much more important!

Log into our class server and copy `~cs213/HANDOUT/hw2.tar` to a working directory. Untar the file (`tar xvf hw2.tar`). You will find the following files:

1. `code-unopt.s` (produced by `gcc -Wall -m64 -S code.c -o code-unopt.s`)
2. `code-unopt.o` (produced by `gcc -Wall -m64 -c code.c -o code-unopt.o`)
3. `code-opt.s` (produced by `gcc -Wall -m64 -O -S code.c -o code-opt.s`)
4. `code-opt.o` (produced by `gcc -Wall -m64 -O -c code.c -o code-opt.o`)
5. `code.h`
6. `test.c`
7. `code-handin.c`
8. `Makefile`
9. `hw2.pdf` (this document)

Your goal is to figure out what C code is in `code.c` and to replicate it in `code-handin.c`. The function definitions in `code-handin.c` are currently empty. You will write them. It will probably be easier to do so by studying the contents of `code-unopt.s` and `code.h` and playing with the compiled code using `test.c`. The purpose of giving you `code-opt.s` and `code-opt.o` is give you an idea of what a compiler will do differently when optimizing. In some cases, the optimized code will be easier to understand. Here is what the various gcc options mean:

- Wall means to warn about known C issues with the code
- m64 means to produce 64 bit code
- S means to produce only the assembly output
- O means to optimize using the default options
- c means to produce the object file, but not link it
- o is the desired output filename

Note that unlike the Bomb Lab, in this homework you are given the *assembly* intermediate file (the `.s` file), in addition to the object code (the `.o` file). Furthermore, we give you both the optimized and the unoptimized outputs from the compiler.

When you run `make`, you will generate `code-handin.s`, `code-handin.o`, `test-with-handin`, and `test-with-handout`. `Code-handin.s` and `code-handin.o` are the assembly and object code for `code-handin.c` – ie, the code that you’ve written. `Test-with-handin` is an executable of `test.c` that’s linked with your `code-handin.o`. `Test-with-handout` is an executable of `test.c` that’s linked with my `code.o`. You might also find it useful to compare your `code-handin.s` with my `code-unopt.s`. If you use the test code, please note that *there is no guarantee that any of these functions will terminate or even run successfully*. However, single-stepping them with `gdb` may be enlightening.

The actual `code.c` will be distributed later so that you can check your answers. The assembly code generated by GCC is in AT&T x86_64 syntax. x86_64 is a two-operand assembly-language. An example instruction is shown below, with a comment describing its behavior.

```
addl %edx, %eax ; Behavior: %edx + %eax → %eax
```