CS 213, Fall 2000
Homework Assignment H1, Part B
Assigned: Sept. 19, Due: Tues., Sept. 26, 11:59PM

Randy Bryant (`Randy.Bryant@cs.cmu.edu`) is the lead person for this assignment.

The purpose of this assignment is to learn IA32/Linux assembly language and to become familiar with how C code is translated into machine code. You will do this by looking at the disassembled versions of a series of functions and decompiling them to find the C source code that produced them. Reverse engineering this code will improve your understanding of both the C constructs and machine-level code.

Introduction

In this assignment you are given a binary executable that includes a set of functions. Your task is to derive C source code that compiles into code that is functionally equivalent. You should strive to make your code produce assembly code that is identical to the supplied assembly code. GCC is very erratic, however, producing different register allocations and instruction orderings depending on minor variations in the source code. So, don't waste a lot of time trying to make your code compile to exactly the same object code.

When you are trying to figure out what a given functions does, try creating a small example to see what code the compiler emits. If you can create a series of small functions that produce part of the answer, you can then piece them together to create a solution.

Logistics

You must work alone on this assignment. The only "hand-in" will be electronic. Any clarifications and revisions to the assignment will be posted on Web page `assigns.html` in the class WWW directory.

All files for this assignment are in the directory:

```
/afs/cs.cmu.edu/academic/class/15213-f00/H1b
```

You will want to do your work on one of the class "fish" machines to be sure that you are using the correct version of the GCC compiler. See the class WWW pages for more information on these machines.

First create a (protected) directory to work in, and copy our template code using the command:

```
tar -xvf /afs/cs.cmu.edu/academic/class/15213-f00/H1b/H1b.tar
```

This will create a variety of files, including one named `probs.c`. In this assignment you will only modify this file. In addition there is a file `probs-solve.bin` which is an executable binary.

Your first step should be to fill in your name and Andrew ID in the structure at the beginning of this file.

Your task is to fill in the bodies of four functions in `probs.c`, named `switchcode`, `proc`, `recursive`, and `recursive2`. In addition you will modify the definitions of compile-time constants `ROW` and `COL`. Your goal is to make the modified functions and the array referencing code have the same functionality as the corresponding functions in the file `probs-solve.bin`.

The original C code for these functions has the following characteristics:

`switchcode`: A switch statement

`proc`: A procedure that calls another procedure

`recursive`: A simple recursive function

`recursive2`: A more convoluted recursive function

In addition, the compile time constants `ROW` and `COL` are used in a set of functions that operate on 2-dimension matrices having `ROW` rows and `COL` columns. By inspecting the functions `get_ele` and `allocate_matrix`, you can determine the values of these two constants.

You cannot view the object code directly. Instead, you need to use a disassembler. Run the command `make probs.bdis` to generate a compiled and diassembled version of `probs.c`. Similarly, you can generate a disassembled version of the target code with the command `make probs-solve.bdis`. Note that there is a lot of other stuff in the disassembled code. You will need to identify the relevant sections of the two files. You can run GDB on the file `probs-solve.bin` to extract the contents of the jump table for the switch statement problem.

Although you could brute force your solution by writing C code that uses goto's and things like that, try to write the cleanest and most abstract C code you can.

Evaluation

You will only get credit for problems in which you are able to exactly match the functionality of the assembly code versions. The five problems count one point apiece. Use the command `make atest` to generate a program that will run your functions against the original functions on some test data. See the file `README` for documentation on this program.

Hand In

Hand in your solution using the command

```
make handin NAME=yourname VERSION=XX
```

where `yourname` is your Andrew ID, and `XX` is the version number, i.e., 2, 3, ... Only your highest numbered version submitted before the deadline will be graded.