Numerical Representations and Object Files

Assigned Monday, July 14, 2014
Suggested Completion Date *Monday, July 21, 2014*

Introduction

The purpose of written homework assignments is to get you thinking about the topics being covered in lecture and in readings in the textbook which are not represented in the hands-on, programming lab assignments. These written assignments also better prepare you for course examinations. It is worth noting that the book contains many practice problems similar to the problems we ask on these written assignments. The solutions for those practice problems are located at the end of each chapter and should give you a feel for the kind of answers we expect you to turn in for these kind of assignments.

Logistics

These written homeworks will not turned in for credit like the programming lab assignments. There also won't be solutions provided, but the forums are open to all discussion about the homework. We encourage you to complete the homework to the best of your ability and then discuss your findings/questions with your peers on the forums.

Questions

Answer the following problems:

1. Homework Problem 2.76 (pg 126)
   Suppose we are given the task of generating code to multiply integer variable `x` by various different constant factors *K*. To be efficient, we want to use **only** the operations `+`, `-`, `<<`. For the following values of *K*, write C expressions to perform the multiplication using at most three operations per expression.

   A. *K* = 17:
   B. *K* = -7:
   C. *K* = 60:
   D. *K* = -112:
2. What, if anything, is wrong with the code below?

```
int test(int i, long int l) {
  return i == ((int)l);
```

```
}
```

3. For this problem you will use the jack-of-all-trades tool `objdump` to examine executable programs on Linux. Be sure to work on this problem on the provided VM. You will need a copy of the code for Lab 1 (your code does not have to be entirely correct or complete but at least some of your puzzle functions should be filled in).
   A. Run `make` in your Lab 1 directory to build your code. Run the command `objdump -t btest | grep text` and examine the output. What do the strings in the rightmost column of the output represent?
   B. Use `objdump` to disassemble your btest code (review the "Machine Programming" lecture in section 3 or check the `man` page to find the right flag). Find the labels in the assembly code corresponding to your bit puzzle functions. What x86 assembly instructions appear to perform right-shift and left-shift operations?
   C. Most Linux distributions include several other useful programs for examining binary files containing machine code. Try out commands such as `ldd`, `file`, `nm`, `strings` and `readelf`, and try other flags with `objdump`. Report something neat, unusual, or unexpected that you find. Don't forget to use the `man` pages to learn more about all of these programs.
   D. Run the `objdump -t` command on a different program on the system - choose one in the */usr/bin/* directory. You will likely see the output "*SYMBOL TABLE: no symbols*". Why does `objdump -t` appear to work on your program, but not on others?
4. **Optional Material**: What, if anything, is wrong with the code below?

```
int test(float f, double d) {
  return ((double)f) == d;
}
```

5. **Optional Material**: Recall that a floating point number is of the form: (-1)S x M x 2E . M is the mantissa and 2E is the exponent. Also recall that M ranges from 1.0 to (almost) 2.0. Tell us the values of M and E for the number 15.0. Note we do not expect you to encode this in IEEE 754 representation, we want you to just tell us in decimal the values of M and E, and the binary value of the frac (recall frac = M - 1.0).
6. **Optional Material**: Homework Problem 2.88 (pg 129)
   We are running programs on a chine where values of type `int` have a 32-bit two's-complement representation. Values of type `float` use the 32-bit IEEE format, and values of type `double` use the 64-bit IEEE format.

   We generate arbitrary integer values `x`, `y`, and `z`, and convert them to values of type `double` as follows:

```
/* Create some arbitrary values */
int x = random();
int y = random();
int z = random();
/* Convert to double */
double   dx = (double) x;
double   dy = (double) y;
double   dz = (double) z;
```

   For each of the following C expressions, you are to indicate whether or not the expression always

yields 1. If it always yields 1, describe the underlying mathematical principles. Otherwise, give an example of arguments that make it yield 0. Note that you cannot use an IA32 machine running GCC to test your answers, since it would use the 80-bit extended-precision representation for both `float` and `double`.

A. `(float) x == (float) dx`
B. `dx - dy == (double) (x-y)`
C. `(dx + dy) + dz == dx + (dy + dz)`
D. `(dx * dy) * dz == dx * (dy * dz)`
E. `dx / dx == dz / dz`

Created Sat 20 Apr 2013 4:14 PM PDT

Last Modified Mon 14 Jul 2014 12:28 PM PDT