Project 7: Random Number Generator

CS 200 • 20 Points Total
Due Friday, April 7, 2017

Objectives

- Create a pseudo-random number generator in assembly.
- Practice I/O and basic math in assembly.

Overview

There are several techniques for generating "random" numbers on computers, but the numbers are actually only **pseudo-random** - they give the appearance of being random but in reality are based on deterministic math. The reason these numbers appear random is because we're not aware of the state of the variables behind the scenes; we only see the "tip of the iceberg" as the next number is calculated and returned.

Linear Congruence

One popular technique for generating random numbers is called "Linear Congruence". This is a very simple formula that takes any arbitrary integer as a "seed" and plugs it into an equation with some carefully-selected constants. The resulting value is the base value for the output and also becomes the seed for the next random number.

In our case, we can take advantage of the way multiplication works in assembly to do this. Basically, we take a very large prime integer (you can find primes in Wikipedia); ideally near 16 or 32 bits in binary (depending on the integer size of our processor). This will be our multiplier. We also select another large number as our ‘seed’. Primes work best but there are some synergies that will cause the series to devolve and start repeating numbers so it is best to test values and see what works. Usually the seed is entered at runtime either by using a system clock value or input from the user – if the same seed is used every time then the same series of numbers will be generated every time. Remember, it is only **pseudo-**random.

To generate the next number in the series, multiply the seed by the multiplier. Since they are both large, the result will be too big to fit in the original registers and will be divided into two registers. If the numbers are big enough, the high order register will be nearly full. However, we don’t count on it, so we’ll use that one for our random number output. But the low order register should always be full, so we’ll use that for the seed the next time we calculate a random. The output of the linear congruence formula is some integer that falls in some arbitrary range (often $0..2^{17}-1$ or $0..2^{31}-1$) depending on integer size. More steps must be taken if you want that number to fall within a certain integer range. Here is LC in psuedocode:

```
multiplier = some big prime
seed = some big number

LC_loop:
    multiply multiplier and seed      ; result is split into high and low halves
    seed = low half result
    pseudorandom = high half result
    **range fit** pseudorandom for display or use    ; see below
    go to LC_loop to generate another
    otherwise we are done
```

Range Fitting

How do you scale a random number to always fall between two arbitrary values? One way is to mod the random number with the size of the desired range (r % size gives 0..(size-1)) and add the low value of the range. For example, in pseudocode:

```
r = get_random_number
num_low_to_high = (r % (hi - low + 1)) + low
```

This equation will give an integer value between "low" and "high" for an original value "r" of any positive number. Note: This is **not** the Linear Congruence algorithm! This is the second part of the process where you take some number (probably huge) and fit it into an arbitrary range. Before you can use this, you first have to use LC to get a pseudorandom.

For more information on generating random numbers, use the web to read up on Linear Congruence. Also, try looking up "drand48" online. Understanding how random numbers are created using linear congruence will help you debug when you get odd results (like non-random repeating numbers).

Preparation

Set up your computer for programming in MIPS assembly by downloading and installing the QtSpim emulator appropriate to your operating system. It would probably be a good idea to test things by assembling and running a simple program; a helloworld.s program should be included with the emulator or you can make your own. You may also use the MARS IDE but beware that it has some differences (mostly in the system calls) from QtSpim, so code that works on one may not work on the other. I like the MARS IDE but I prefer the QtSpim debugger, which is where I test your MIPS code if you bring it to me with problems, unless you tell me otherwise.

Requirements

Write a MIPS assembly language program that reads in a low number, a high number, (for range) and a count of how many random numbers the user wishes to generate. The program should then display the requested number of values. Each should be randomly generated and should be between "low" and "high", inclusive.

Here are some tips and requirements:

- Use the MIPS syscalls to print out prompts and read in integers. The process is detailed in 2.11 and the different commands are detailed at the end of appendix A on page 112.
- Do **NOT** use a routine you found online to generate random numbers. You must implement the Linear Congruence formula yourself.
- You will have to prompt for an initial seed. Be sure that the number is big enough (and follows the parameters for a linear congruence seed) or you may not get random results. In other words, you may have to prompt the user to try again if they give you a seed too small or too large. You'll have to decide for yourself what your min and max is. Similarly, you will want to check the other user inputs for correctness.
- Build and test your program in incremental stages to avoid being overwhelmed by details. For example, you might start off writing a loop that prints out the number "5" 20 times. You might then change it to hard-code a first and last number and then print out every number in between... and so on.
- Generating a random number is a two-step process. 1) Generate the huge random and next seed numbers using linear congruence. 2) Use the fitting formula on the first page of this assignment to scale the random you got into the range you need. Do these two steps each time you loop.

Project Report

The final step of this assignment is to create a report consisting of a cover page, an overview of the project, sample output, and the source code. See Assignment Policies on either the class website or Bb Learn.