CSC 3210 Computer Organization and programming

LAB 12

Random Number Generation

ASSIGNMENT FOR LAB 12

Tasks to be done in this lab:

- Learn how to seed an RNG.
- Learn how to get a random value.
- Learn how to seed the RNG to give an unpredictable sequence of values.

INSTRUCTIONS FOR LAB 12

In this lab, there are 4 parts.

- Part 1 - Making a random function
- Part 2 - Making a seed function
- Part 3 - Making the values smaller
- Part 4 - A random-looking seed

PART-1

- Suppose that we do not have an RNG. How could we make one? Fortunately, there is a good paper about this.. You do not have to read the paper. The central idea is that the author, George Marsaglia, shows that you can create good RNGs with fairly simple operations, e.g.

  y^=y>>a; y^=y<<b; y^=y>>c;

- The first one, y^=y>>a;, says to right shift the value called y by a bits, then XOR that result with the original y. After this, we do a very similar operation only left shifting by b, and finally right shifting by c. Values for a, b, and c are given in the paper, and there are many possibilities. Also, there are other possibilities for the shift operations besides the one shown, e.g. y^=y>>c; y^=y<<b; y^=y>>a; would also work.
- In this lab, you will implement an RNG. You can use rand_pt1.S (or rand_pt1_RARS.s) to get started, or you can create your own version from previous code. This program is supposed to print 20 random values, but right now it prints the value stored in memory named "lfsr" 20 times. The name "lfsr" stands for linear-feedback shift register, which is a general name for this type of algorithm.

PART-1

- Let's take a look at the code. We put the immediate value 20 into register s0. Next comes the label "myloop", which is where the loop starts. It calls the subroutine "get_rand", then prints the value returned in a0, followed by a space. The printing commands are omitted.

```
    li   s0, 20              # s0 holds the count
myloop:
    call   get_rand          # get a random value
    # print the value, then a space
    # ...
    # Update s0 to count down
    addi   s0, s0, -1
    bgt    s0, x0, myloop       # if greater or equal, jump to myloop
```

- After printing the value and a space, the code adds -1 to the loop index, stored in register s0. Next, it compares register s0 to 0, and branches to the start of the loop if s0's value is greater than 0. If you run the program now, you should get the following output.

```
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

PART-1

- For part 1 of this lab, implement a RNG in the subroutine "get_rand". The example code includes comments indicating what to do after it loads the value from memory at label "lfsr". Note that it should store the result back to memory before the subroutine returns.
- Verify that you get a series of random-seeming values.

Questions

1. **Does your code work with shifts of 2, 7, 9 as well? Note that this is from the Marsaglia paper.**
2. **Does your code work with shifts of 1, 1, 1 as well? Note that this is not one of the triplets from the Marsaglia paper. Why do you think the triplet 1,1,1 is not as good as 2,7,9?**

PART-2

- First, let's examine how we would use the seed function and the random function in a typical program. In the main part of the program, we will call the srandom function with the seed value. The following code will "seed" the random number generator with the value 3. Note that there is nothing special about using as the seed value. Choosing a seed value is beyond the scope of this lab, but if you are interested you can find academic papers on the subject. We use a seed value here to get a repeatable series of random values. The code below is from a C program, but it should make sense no matter what languages you know.

```
// Seed the random number generator
srandom(3);
```

- Next, we call the random function to get a pseudo-random number.

```
a = random();
```

- We can call the random() function as many times as we want. We only need to call the srandom() function once. How would this look in assembly? The code below shows an example of this, with "set_seed" and "get_rand" in place of "srandom" and "random", respectively.

```
li    a0, 3
call  set_seed
call  get_rand
; a0 should now hold a random value
```

PART-2

- After the call to "get_rand", we store the value held in the a0 register. Edit the "rand_pt1.S" program to add a "set_seed" function that sets the seed value to whatever the calling function put into a0. Basically, all that it needs to do is to store it in memory at the "lfsr" label. Do not call the seed function a second time; the point is to see that it does generate a sequence of values. Put all of this together, along with anything else that you need to make a fully functioning program.
- Show the program, and that it works. Run this program three times. You should observe the same three values are output each time.
- Copy the program to a new file. In the copy, change the seed value. It's OK to hard-code it like in the previous example. Run this program three times. You should observe the same three values are output each time, but that these values are different from those generated by the first version.

Questions

1. **Suppose that we set the seed to the initial data value in part 1 (lfsr: .word 1), but set it with "set_seed". Do you get the same values as you did in part 1? Why or why not?**

PART-3

- You may have noticed that the values have a large magnitude. What if we want the values to have a smaller range, such as under 32? There is an easy way to do this, and that is to AND the random result with a value. For values under 32, we can AND it with the constant 0x1f.
- Implement this. That is, get the random value by calling the "get_rand" function, then (in the main loop) AND the value with 0x1f before printing it. Show that it works.

Questions

1. **Does your program generate any negative values now? Why or why not?**
2. **Suppose that we AND the value with 0x03. When you run it, you will see a lot of repeated values. Is this still a random sequence? Explain.**
3. **Why do we limit the random values to under a power of 2? What would we need to do to limit it to the range 0 to 99?**

PART-4

- (This part is informational only -- you do not need to implement it. This is because it relies on an ecall that VSCode/Venus does not implement.) Now let's see how we can give the seed function an unpredictable value. This way, the seed itself will seem to be random, so the generated values are also going to be unpredictable. This is good for simulations and games, where we want the user's experience to be different each time.
- One common way to do this is to use a variation on the time for the seed. For example, there is a "time()" function in C that reports the current time as an unsigned integer, and this value changes every second. Therefore, using that will allow a program to make a random-seeming seed value. We will use this idea to seed the RNG.

```
# Code for RARS
li   a7, 30
ecall
# Result is time in a0,a1
# We will ignore a0
# Use a1 for the seed
```

- This is done primarily so that we can print it out. If you were to implement it, you should observe that it prints a random value each time, and that these random values are different every run. You should also observe that the seed value would therefore be different every time you run it.

IMPORTANT NOTE:

Remember that we will grade your lab report so it is vital to turn that in. The other files (your code, a text version of any log file, etc.) are to document your work in case we need more information.