Joshua Pollock

CS 200 Project 7

7 April 2017

Joshua Pollock

CS 200

Project 7 – Random Number Generator

7 April 2017

**Purpose:**

The purpose of this lab is to create a pseudo-random number generator in MIPS assembly, that will take in an amount of random numbers to generate, a lower limit, an upper limit, and a seed. This will introduce us to MIPS and help us practice basic I/O and math in assembly. We will be using linear congruence to calculate the random numbers. The program will then take the numbers generated by this and fit them to a specific range using a formula given by the lab report. Another constraint is that the program must check all inputs for validness (i.e. negative amount of numbers to generate).

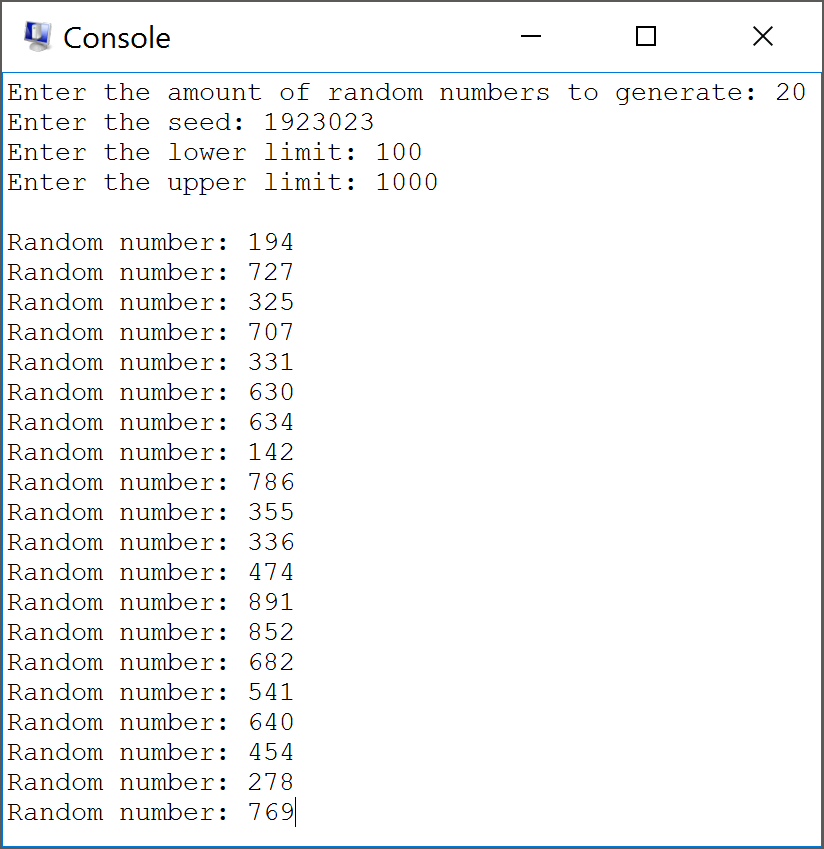
**Research:**

At first I had no idea where to start. I was not at all familiar with MIPS. I found that there were a couple of useful sources on BBLearn. The IOdemo.s file proved to be quite hepful at figuring out how MIPS operates. I used this file to set up all my variables and inputs for the user. After setting up a simple input file, I started to look at the MIPS Reference PDF that is also on BBLearn. This PDF helped to give me an understanding of how MIPS operates and I began setting up the simple math needed to complete the project.  
  
 Beginning to create the loop, I had no idea how to separate the hi and lo values. I finally found a website (<https://forum.allaboutcircuits.com/threads/multiplication-of-large-numbers-in-mips.116254/>) which had example code of how to do this. When I was testing my solution, I received weird results. It had turned out to be from this poorly commented example code. I had included the syscalls after mflo and mfhi. This cause my finally result to print out 3 times in a row. For example, 35 would display as 353535. It was only after quite some time going over my code, I found that this was the error.

The next thing I had to research was modulo division. Unlike other languages, MIPS does not have a way to simply do modulo division with a ‘%’ sign. I searched modulo division and found a Stackoverflow post (<https://stackoverflow.com/questions/2664301/how-does-modulus-divison-work>). This helped show me the steps required to do modulo division, and helped me finish implementing the last steps for the project.

**Program:**

The first version of my program was quite confusing to understand and follow. I decided to implement 6 labels that would help to check for invalid inputs. At first I just had these all contained within the main label and 3 other labels. This ended up causing some strange errors when the user entered invalid responses. Separating these into 6 labels helped me to see the progress of my code and fixed the errors I was getting as well. The next issue I had with my code is that it was displaying numbers under the lower limit. I fixed this by adding two checkers to check that the number was within the limits, and jumped back to loop if it was not. This line of code allowed me to have a seed of 1 while having the program still function. The final version of my code displays this result when ran:



**Conclusion:**

This project was quite entertaining. When I first looked at the project, I had no idea what to do or how to start. After finding the IOexample.s file on BBLearn everything began to fall in place like dominos. While completing this lab I grew quite fond of assembly. Unlike other languages, you can jump around assembly. This made coding using MIPS quite intuitive and simple. As I started to write with assembly, and referenced the reference PDF, everything became quite simple to implement. I encountered some funky errors because of syscall. Syscalls are one of the few things I still do not understand about MIPS. It took a good 30 minutes before I found the issue that caused the generated numbers to repeat 3 times.  
  
 I ended up implementing in two checkers for the loop label. These would check if the number generated was within the limits given by the user, and if they were not, the program would jump back to loop and repeat the process but with a new seed. This work around also let the seed be anything greater than zero. Another change that I made to my code was the use of labels. I created a total of 10 labels to have the program function properly. I find this makes the code a lot easier to follow and troubleshoot. Now instead of searching through a large main function and loop, the problems can be isolated to their specific labels.   
  
 Overall I greatly enjoyed this project. I have also come to adore MIPS/assembly. Troubleshooting errors may take a bit longer to do compared to other languages, but the rest of assembly was quite enjoyable. I look forward to seeing the more complex projects in the next few weeks.