Project Report

Pseudo-Random Number Generator using Linear Congruence in MIPS Assembly

Manish Pratap Choudhary (mc4323)
Course: CS 200 **Computer organization**
Instructor: Patrick Kelley
Date: April 12, 2024

Overview of the Project:

The aim of this project was to develop a pseudo-random number generator using the Linear Congruence algorithm in MIPS assembly language. The Linear Congruence algorithm is a straightforward formula that generates a sequence of pseudo-random numbers based on a seed value and a multiplier. This project serves as an introduction to assembly language programming for students, emphasising the basics of user input, mathematical operations, and looping constructs.

Implementation Approach

1. User Input:
   - The program prompts the user to enter the minimum and maximum integer values for the range of random numbers to generate.
   - It then asks for the number of random values the user wants to produce.
   - Finally, the program requests a seed number to initialise the pseudo-random number generator.

2. Range Fitting:
   - After obtaining the user inputs, the program calculates the range of random numbers by subtracting the minimum value from the maximum value and adding 1.

3. Random Number Generation (Linear Congruence Algorithm):
   - The program iterates through a loop to generate the specified number of random values.
   - Within each iteration, it applies the Linear Congruence algorithm:
     - Multiplies the seed value by a predetermined large prime multiplier.
     - Extracts the high part of the result as the pseudo-random number.
     - Updates the seed for the next iteration.

4. Error Handling:
   - Error handling mechanisms are implemented to ensure valid user inputs:
     - Checks for input values within the specified ranges.
     - Displays appropriate error messages and prompts the user to enter valid inputs.

Results

Upon successful execution, the program generates a series of pseudo-random numbers within the specified range. The user is presented with the requested number of random values, each fitting between the provided minimum and maximum values.

Sample Output

Here is an example of the program's output:

```
Enter the smallest number acceptable (0<=n<=998): 10
Enter the biggest number acceptable ( n <= 1000): 50
```

How many randoms do you want (5 - 100)? 10
Enter a seed number (1073741824 - 2147483646): 2000000000
Here is the series of randoms:
43
13
25
20
30
40
35
15
50
10

-- program is finished running --

Enter the smallest number acceptable (0<=n<=998): 0
Enter the biggest number acceptable (n <= 1000): 10
How many randoms do you want (5 - 100)? 10
Enter a seed number (1073741824 - 2147483646): 1073741824
Here is the series of randoms:
7
5
7
4
1
10
3
1
6
1

-- program is finished running --

Enter the smallest number acceptable (0<=n<=998): 20
Enter the biggest number acceptable (n <= 1000): 50
How many randoms do you want (5 - 100)? 5
Enter a seed number (1073741824 - 2147483646): 1173741850
Here is the series of randoms:
45
47
22
21
35

-- program is finished running --

Source Code

```
#-------------------------------------------------------------------------------
# lcrand.s
#
# This program prompts the user for a min and max integer value and then for an
# integer number of results to produce. Finally, it asks for a number to use as
# a random number seed. With those four values properly entered, it loops
# generating a series of random numbers that fit between the min and max values
# previously entered, using the Linear Congruence algorithm.
#
# This is intended to be the first project in Assembly Language written by
# students in CS200 at NAU as an introduction to the language. It does not
# require the use of subroutines or other advanced techniques.
#
# Author: Manish pratap choudhary (mc4323)
# Date:4 04/12/2024
#
# Revision Log
#-------------------------------------------------------------------------------
# 10/02/2023 - Created Skeleton for student use
#-------------------------------------------------------------------------------
.data
# constants
Multiplier: .word 1073807359 # a sufficiently large prime
# You probably have enough registers that you won't need variables.
# However, if you do wish to have named variables, do it here.
# Word Variables First
# HalfWord Variables Next
# Byte Variables Last (Strings and characters are Byte data)
# common strings
minPrompt: .asciiz "Enter the smallest number acceptable (0<=n<=998): "
maxPrompt: .asciiz "Enter the biggest number acceptable ( n <= 1000): "
qtyPrompt: .asciiz "How many randoms do you want (5 - 100)? "
rsdPrompt: .asciiz "Enter a seed number (1073741824 - 2147483646): "
smErr1: .asciiz "That number is too small, try again: "
smErr2: .asciiz "The max cannot be less than the min, try again: "
bgErr: .asciiz "That number is too large, try again: "
outStr: .asciiz "Here is the series of randoms:\n"
newLine: .asciiz "\n"
.text
.globl main
#-------------------------------------------------------------------------------
# The start: entry point from the MIPS file is encompassed in the kernel code
# of the QTSPIM simulator. This version starts with main: which does everything
# in this simple program.
#-------------------------------------------------------------------------------
main: # start of the main procedure
```

```
# Prompt for the minimum result and get the value
min_input:
    li      $v0, 4              # syscall to print string
    la      $a0, minPrompt          # load address of minPrompt
    syscall

    li      $v0, 5              # syscall to read integer
    syscall                     # read integer
    move    $t0, $v0                # save the input in $t0

    # Check if the input is within the valid range
    blt     $t0, 0, min_input_error # if input < 0, go to error
    bgt     $t0, 998, min_input_error # if input > 998, go to error

# Prompt for the maximum result and get the value
max_input:
    li      $v0, 4              # syscall to print string
    la      $a0, maxPrompt          # load address of maxPrompt
    syscall

    li      $v0, 5              # syscall to read integer
    syscall                     # read integer
    move    $t1, $v0                # save the input in $t1

    # Check if the input is within the valid range and greater than minimum
    blt     $t1, $t0, max_input_error # if input < min, go to error
    bgt     $t1, 1000, max_input_error # if input > 1000, go to error

# Prompt for the number of randoms to generate
qty_input:
    li      $v0, 4              # syscall to print string
    la      $a0, qtyPrompt          # load address of qtyPrompt
    syscall

    li      $v0, 5              # syscall to read integer
    syscall                     # read integer
    move    $t2, $v0                # save the input in $t2

    # Check if the input is within the valid range
    blt     $t2, 5, qty_input_error # if input < 5, go to error
    bgt     $t2, 100, qty_input_error # if input > 100, go to error

# Prompt for the seed number
seed_input:
    li      $v0, 4              # syscall to print string
    la      $a0, rsdPrompt          # load address of rsdPrompt
    syscall
```

```
    li      $v0, 5                  # syscall to read integer
    syscall                         # read integer
    move    $t3, $v0                # save the input in $t3

    # Check if the input is within the valid range
    blt     $t3, 1073741823, seed_input_error # if input < 1073741823, go to error
    bgt     $t3, 2147483646, seed_input_error # if input > 2147483646, go to error

    # Output the series of random numbers
    li      $v0, 4                  # syscall to print string
    la      $a0, outStr             # load address of outStr
    syscall

    # Calculate the range
    sub     $t4, $t1, $t0           # $t4 = max - min
    addi    $t4, $t4, 1             # $t4 = max - min + 1

    # Initialize loop counter
    move    $t5, $t2                # $t5 = number of randoms

random_loop:
    # Generate a random
    lw      $t7, Multiplier         # Load Multiplier value into $t4
    mulu    $t3, $t3, $t7           # Multiply seed by Multiplier
    mflo    $t3
    mfhi    $t6                     # $t6 = high part of result (random)
    move    $t3, $t6                # $t3 = high part of result (new seed)

    # Range fit the random for output
    div     $t6, $t4                # $t6 = random / range
    mfhi    $t6                     # $t6 = random % range
    add     $t6, $t6, $t0           # $t6 = random % range + min

    # Print out the result
    li      $v0, 1                  # syscall to print integer
    la    $a0, 0($t6)               # load random number to print
    syscall

    # Print a newline
    li      $v0, 4                  # syscall to print string
    la      $a0, newLine            # load address of newLine
    syscall

    # Decrement loop counter
    subi    $t5, $t5, 1             # decrement loop counter

    # Repeat loop if loop counter > 0
```

```
    bgtz    $t5, random_loop        # if $t5 > 0, repeat loop

exit_program:
    # Exit program
    li      $v0, 10                 # syscall to exit program
    syscall

min_input_error:
    # Print error message for min input
    li      $v0, 4                  # syscall to print string
    la      $a0, smErr1             # load address of smErr1
    syscall
    j       min_input               # jump back to prompt for min input

max_input_error:
    # Print error message for max input
    li      $v0, 4                  # syscall to print string
    la      $a0, smErr2             # load address of smErr2
    syscall
    j       max_input               # jump back to prompt for max input

qty_input_error:
    # Print error message for quantity input
    li      $v0, 4                  # syscall to print string
    la      $a0, qtyPrompt          # load address of qtyPrompt
    syscall
    j       qty_input               # jump back to prompt for quantity input

seed_input_error:
    # Print error message for seed input
    li      $v0, 4                  # syscall to print string
    la      $a0, bgErr              # load address of bgErr
    syscall
    j       seed_input              # jump back to prompt for seed input
```

Conclusion

In conclusion, this project has provided valuable hands-on experience in MIPS assembly language programming, particularly in implementing a pseudo-random number generator using the Linear Congruence algorithm. Key learnings from this project include:
- Understanding the Linear Congruence algorithm for pseudo-random number generation.
- Handling user input validation and error checks.
- Implementing looping constructs for iterative tasks.
- Gaining familiarity with MIPS assembly language syntax and instructions.

Overall, the project successfully met its objectives of creating a functional pseudo-random number generator in assembly language. This experience has laid a solid foundation for further exploration into more complex assembly programming concepts.

End of Report
