ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO : 7

EXPERIMENT DATE : 25.12.2024

LAB SESSION : WEDNESDAY - 12.30

GROUP NO : G8

GROUP MEMBERS:

150200009 : Vedat Akgöz

150200097 : Mustafa Can Çalışkan

150200016 : Yusuf Şahin

SPRING 2024

Contents

1 Introduction

In Experiment 7, we worked on implementing random number generation techniques using assembly language. The experiment had three main parts:

- Sequence generation with the Blum-Blum-Shub algorithm
- A random number generator using the Middle Square Weyl Sequence algorithm
- An analysis of the uniformity of a custom random number generator

Our main goal was to write efficient and modular code while making use of stack-based operand transfers and interrupt-driven programming.

The codes are not included in the report due to their length. The source files we uploaded to the Ninona website can be reviewed for the codes.

2 Materials and Methods

2.1 Part 1

In Part 1, we implemented the Blum-Blum-Shub (BBS) algorithm to generate a sequence. The parameters we used were:

- $p = 11$, $q = 13$
- $M = p \times q$
- Seed $s = 5$

We generated the sequence using the recurrence formula: We designed the implementation as an interrupt subroutine, triggered by pressing the button on pin P2.5. Each generated number was displayed on a 7-segment display. The code was written in assembly and included subroutines for modular arithmetic and updating the display.

2.2 Part 2

In Part 2, we implemented the Middle Square Weyl Sequence (MSWS) algorithm for random number generation. The parameters we started with were:

- Initial values: $x = 0$, $w = 0$, seed s
- Recurrence formulas:
  - $x = x^2$

- $x = x + (w = w + s)$
- $r = (x >> 4)|(x << 4)$

Like Part 1, this algorithm was also implemented as an interrupt subroutine triggered by P2.5. The output was limited to the range 0-128 and displayed on the 7-segment display. We focused on stack operations and efficient bitwise manipulations while writing the assembly code.

2.3 Part 3

For Part 3, our goal was to generate 128 random numbers in the range [0, 8) using the BBS generator from Part 1. We planned to:

- Store the generated numbers in preallocated memory
- Analyze their distribution for uniformity

However, due to time constraints, we couldn't complete this part. We had intended to count the occurrences of each number and store the results in memory for further analysis.

3 Discussions

In Part 1, we successfully implemented the BBS algorithm in an interrupt-driven framework. The choice of parameters provided a long sequence period, and our implementation of modular arithmetic was efficient.

For Part 2, the MSWS algorithm worked well for generating pseudorandom numbers with minimal computational effort. The bitwise operations we used helped ensure a uniform distribution in the desired range.

One challenge we faced was managing stack operations for operand transfers and synchronizing display updates with interrupt handling. Additionally, the limited time prevented us from fully implementing and analyzing Part 3.

4 Results

- **Part 1**: The BBS generator produced a sequence of numbers that were correctly displayed on the 7-segment display.
- **Part 2**: The MSWS generator successfully output random numbers within the range 0-128.
- **Part 3**: We couldn't obtain results for this part due to incomplete implementation.

5 Conclusion

This experiment emphasized the importance of modular and efficient coding practices in assembly language. In Parts 1 and 2, we successfully implemented two different random number generation algorithms. Although we couldn't complete Part 3, we laid the groundwork for analyzing the uniformity of random numbers.

REFERENCES