ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 351E

MICROCOMPUTER LABORATORY EXPERIMENT REPORT

EXPERIMENT NO : 6

EXPERIMENT DATE : 27.12.2024

LAB SESSION : FRIDAY - 14.30

GROUP NO : G9

GROUP MEMBERS:

150220723 : Abdulsamet Ekinci

150210906 : Emil Huseynov

150220902 : Nahid Aliyev

FALL 2024

Contents

1 INTRODUCTION [10 points]

In this experiment, we implemented a sequence generator and a random number generator using two different algorithms: Blum-Blum-Shub (BBS) and Middle Square Weyl Sequence (MSWS). Additionally, we aimed to utilize these generators to create a distribution of random numbers and analyze their uniformity. This experiment emphasized the importance of subroutine-based programming with stack-based operand transfer, interrupt handling, and efficient debugging techniques.

2 MATERIALS AND METHODS [40 points]

2.1 Hardware

- **MSP430G2553 Microcontroller:** A low-power microcontroller with 16-bit RISC architecture.
- **7-Segment Display:** Used to visualize generated numbers.
- **Button (P2.5):** Configured as an interrupt-enabled input to trigger number generation.

2.2 Software

- **Code Composer Studio (CCS):** Used for assembly programming, debugging, and flashing programs onto the MSP430.

2.3 Part 1: Blum-Blum-Shub (BBS) Sequence Generator

Objective: Generate a sequence using the BBS algorithm and display it on the 7-segment display.

Procedure:

- Initialized variables $p = 11$, $q = 13$, and $M = p \times q$.
- Implemented the algorithm as an interrupt subroutine triggered by button press (P2.5).
- Calculated $r = s^2 \mod M$ and updated s to r.
- Converted the result to 7-segment display format and visualized the sequence.

Observations:

- The BBS generator produced a deterministic sequence based on the seed value.
- The implementation worked as expected, correctly updating the 7-segment display.

2.4 Part 2: Middle Square Weyl Sequence (MSWS) Generator

Objective: Implement a random number generator using the MSWS algorithm and display results on the 7-segment display.

Procedure:

- Initialized variables $x = 0$, $w = 0$, and s as the seed.
- Implemented MSWS logic: $x = \text{square}(x) + (w = w + s)$ and $r = (x >> 4) | (x << 4)$.
- Used P2.5 as an interrupt to generate a new random number.
- Displayed the result on the 7-segment display.

Observations:

- Random numbers were generated but required additional debugging to ensure correctness.
- The shifting and combining operations introduced some inconsistency, partially due to incorrect handling of register values.

2.5 Part 3: Uniform Distribution of Random Numbers

Objective: Generate 128 random numbers between 0-8 (excluding 8) using the BBS generator and analyze their uniformity.

Procedure:

- Generated random numbers using the BBS algorithm and stored them in preallocated memory.
- Counted occurrences of each number (0-7) to assess uniformity.
- Attempted to analyze the distribution across different intervals and dataset sizes.

Observations:

- The implementation failed to store numbers reliably due to memory handling issues.
- Counting logic was implemented but could not be tested thoroughly because of storage errors.

3 RESULTS AND DISCUSSION [30 points]

3.1 Part 1 Results

The BBS generator successfully produced a sequence of numbers displayed on the 7-segment display. Interrupt-driven logic worked without issues, and stack-based operand transfer was implemented effectively.

3.2 Part 2 Results

The MSWS algorithm partially worked, generating some random numbers. However, incorrect handling of shifting operations and accumulator values caused inconsistent outputs. Further debugging is required to ensure reliable functionality.

3.3 Part 3 Results

We encountered challenges in implementing the random number storage and analysis. Memory allocation issues and stack management errors led to incomplete results. While the counting logic was theoretically sound, its effectiveness could not be verified due to storage-related bugs.

4 CONCLUSION [20 points]

This experiment provided valuable insights into subroutine-based assembly programming, interrupt handling, and debugging. Key outcomes include:

- Successful implementation of the BBS sequence generator.
- Partial success with the MSWS random number generator, highlighting areas for improvement in shifting operations and accumulator management.
- Challenges in memory handling for Part 3 emphasized the importance of robust debugging and testing techniques.

Future work should focus on:

- Refining MSWS logic for consistent results.
- Addressing memory allocation issues to ensure reliable storage and analysis of random numbers.
- Improving uniformity analysis to verify the quality of random number generators.

Overall, this experiment underscored the complexity and importance of efficient coding practices and debugging in assembly programming.