Experiment 8

Sevim Eftal Akşehirli, 150190028
Hacer Yeter Akıncı, 150200006
Büşra Özdemir, 150200036
Aslı Yel, 150200054

***Abstract*—This project involves using the "Blum-Blum-Shub" algorithm as an interrupt routine to create a sequence displayed on a 7-segment interface. Starting with 5, the sequence appears when triggered by pressing the button. Additionally, a timer interruption makes the display show new values every second. Three buttons allow freezing/unfreezing the number on display, adding it to a register, and displaying the total from the register while stopping the process. This experiment combines interrupts and user interaction for practical use in embedded systems.**

***Index Terms*—"Blum-Blum-Shub", 7-segment display, button, timer interrupt, BCD Conversion .**

I. Introduction

This experiment combines elements from earlier studies, focusing on gaining more hands-on experience with the MSP430 board and assembly coding. It explores the vital role of timers, a key feature in microprocessors, shedding light on their significance in functionality. Moreover, it introduces the use of a 16x2 dot matrix LCD, starting with setting up the initial configurations in Part 0 and progressing towards creating different programs to display changing text. This experiment serves as a bridge, linking previous knowledge to a deeper understanding of embedded systems and programming intricacies.

II. Materials and Methods

List of materials used and how these were used / connected (good opportunity to present block diagrams to show connections).

You can include code parts and examine line by line if you need. **PART1:**

```
Main        mov.w   &s , R8 ;R8 = R
            mov.w   &q , R4
            mov.w   &p , R5
            call    #Multiply
            mov.w   R6, R7 ;R7 = MOD
            mov.w   #0d, R11
            mov.w   #0d, R12
            mov.b   R8, R5
            jmp Display3
```

Main: The variables are initialized for the BlumBlumShub formula. Then, the Multiply function is called to calculate p×q. The result of the multiplication is stored in R6, and we perform a modulo operation on R6 with mod R7. After that, we set the R11 and R12 registers to 0 for further conversion operations. Finally, we jump to the Display loop to display the seed value on the 7-segment display.

```
BlumBlumShub
    mov.w   R8,R4
    mov.w R8,R5
    call #Multiply
    mov.w R6,R5
    mov.w R7,R4
    call #Mod
    mov.w R5,R8
```

BlumBlumShub:"In this section, a random number is generated using the Blum-Blum-Shub algorithm. To find the seed, we square the previous seed (R8) and perform a modulo operation with a determined modulus (R7). The result of the multiplication is stored in R6, and the result of the modulo operation is stored in R5.

```
Mod         cmp R4, R5
            jl Return
            sub R4, R5
            jmp Mod
```

Mod: This function subtracts the modulus until the value is less than the modulus.

```
Multiply    mov #0d, R6
L2          cmp     #0d, R5
            jeq     Return
            sub #1d, R5
            add  R4, R6
            jmp L2
```

Multiply: This function adds R4 to R6, R5 times.

```
Convertion  mov.b #0d,R11
            mov.b #0d,R12
Hundreds    cmp     #100d, R5
            jl  Tens
            sub #100,R5
            inc.b   R11
            jmp   Hundreds

Tens        cmp     #10d, R5
            jl  Loop
            sub.b   #10d, R5
            inc.b R12
            jmp   Tens
```

Convertion: Conversion is performed to find the ones, tens and hundreds digits of the generated random number.

```
Display3        clr    P1OUT
        clr    P2OUT
        mov.b  #00000010b, R6
        mov.w  #numbers, R13
        add    R11, R13
        mov.b  0(R13), P1OUT
        mov.b  R6, P2OUT

Display2        clr    P1OUT
        clr    P2OUT
        mov.b  #00000100b, R6
        mov.w  #numbers, R13
        add    R12, R13
        mov.b  0(R13), P1OUT
        mov.b  R6, P2OUT

Display1        clr    P1OUT
        clr    P2OUT
        mov.b  #00001000b, R6
        mov.w  #numbers, R13
        add    R5, R13
        mov.b  0(R13), P1OUT
        mov.b  R6, P2OUT
        jmp  Display3
```

Displays: These functions clear the current display. R6 keeps the place of the cursor in the 7-segment display. Each time we rotate left to get the next index, R13 is set with the start address of numbers. To find the corresponding number of digits, we add the value to it. After the conversion function, R11 keeps the hundreds digit, R12 keeps the tens digit, and R5 keeps the ones digit. Initially, R11 and R12 are zero, so if the number is less than a hundred or ten, it displays zero. This loop continues forever until an interrupt occurs.

```
ISR
       dint
       call #BlumBlumShub
       clr   &P2IFG
       eint
       reti
```

ISR: The interrupt calls BlumBlumShup and returns the display.

PART2:

```
BlumBlumShub
        mov.w    R8,R4
        mov.w  R8,R5
        call   #Multiply
        mov.w  R6,R5
        mov.w  R7,R4
        call   #Mod
        mov.w  R5,R8
```

BlumBlumShub: In this section, a random number is generated with the Blum-Blum-Shub algorithm, as in part 1.

```
Convertion     mov.b #0d,R11
        mov.b #0d,R12
Hundreds     cmp      #100d, R5
        jl   Tens
        sub  #100,R5
        inc.b     R11
        jmp    Hundreds

Tens       cmp      #10d, R5
                jl   Loop
                sub.b    #10d, R5
                inc.b  R12
                jmp    Tens
```

Convertion: Conversion is performed to find the ones, tens and hundreds digits of the generated random number.

```
timer_INT  mov.w #0000001000010000b, TA0CTL
       mov.w  #10480000d, TA0CCR0
       mov.w  #0000000000010000b, TA0CCTL0
       clr    &TAIFG
       eint
    .......

TISR   dint
     call  #BlumBlumShub
     clr    &TAIFG
     eint
     reti
```

Timer Interrupt: With timer interrupt, a new random number is created every 2 seconds. It updates the LCD with random numbers obtained from the algorithm.

III. Results

In the first part, we successfully implemented the ”Blum-Blum-Shub” algorithm as an interrupt subroutine. P2.5 was utilized as a button to interrupt the microcomputer, and the generated sequence was displayed on the 7-segment display. The initial seed chosen was 5, and the algorithm, based on the given parameters p=11, q=13, and an initial seed s, produced a sequence of random numbers according to the specified formula. The interrupt-driven approach allowed for the display of these numbers on the 7-segment display when the interrupt button was pressed. In the second part, we tried to add a timer interrupt to the experiment to increase functionality. We provided a continuous stream of random numbers generated by the ”Blum-Blum-Shub” algorithm. The LCD was updating and displaying the next value in the sequence, but because the timer interrupt wasn’t working properly, this was happening much faster instead of every second. Since the time given was not enough, we could not write any code for the third part and did not achieve any results.

IV. Discussion and Summary

In this experiment, we first implemented the Blum-Blum-Shub algorithm. Then, we displayed the numbers obtained as a result of this algorithm on the seven-segment display. Then we added a button to the interrupt subroutine. We created a code in which the Blum-Blum-Shub algorithm generates a new number based on the last displayed value every time the button is pressed. In the second part of the experiment, we created a code that displays the number generated by this algorithm on the LCD. The value shown by the LCD was supposed to change every 2 seconds, but we could not manage to do this within the given time. Since we did not have enough time to move on to the third part of the project, we could not do this part.