Arithmetic Programming with a Pseudo-Random Number Generator

Matthew Beckler
beck0778@umn.edu
EE2361 Lab Section 007

February 14, 2006

Abstract

In this lab, a pseudo-random number generator is created, requiring custom addition and multiplication functions. The algorithm is implemented both in assembly language and the C programming language.

1 Introduction

This lab is intended to illustrate the limitations of the PIC18F family's arithmetic operators. The pseudo-random number generating algorithm we have implemented requires a 32-bit add and a 16x16-bit multiply, but the PIC18F family only has support for an 8x8-bit multiply and 8-bit add. To illustrate the flow of the pseudo-random number generator, a flowchart has been created, and included in the appendices. The basic premise of the projects operation is to generated pseudo-random numbers, meaning in a predefined sequence. The output number will be displayed on eight LED's attached to PORTC. The LED's are connected in an active low configuration, whereas when the pin on the microcontroller is pulled low, current will flow through the LED, causing it to light. This means that the number being output will not actually be displayed, rather the compliment of that number will be displayed. This is not a big problem, as humans can easily reverse it in their heads. When the project is initially powered-on, none of the LED's are to be lit. When the button is pressed, which is connected between PORTB pin 7 and ground, the first of the numbers is calculated and output to the LED's. Since tactile pushbutton switches are mechanical devices, they do not produce a perfect electronic signal. When pushed, the buttons generate a random sequence of 1's and 0's, which is called signal noise. The easiest way to eliminate this potential problem is to introduce a delay immediately after the first detected button press or release. This will allow time for the button to settle down. A circuit schematic, which has been provided by the instructor, is included the appendices. This lab has been programmed in both assembly and the C language. Both program sources have been included in the appendices.

2 Observations

The 16x16-bit multiplication subroutine requires 33 instruction clock cycles including the calling and returning instructions. The pseudo-random number generating subroutine, which includes two calls to the multiply subroutine and three calls to the addition subroutine, requires 222 instruction clock cycles in total, again, including the calling and returning instructions.

When initially designing this program, it was helpful to create both a flowchart and a pseudo-code representation of the program to be. The flowchart is included in the appendices, and the pseudo-code is reproduced here:

```
main
{
        wait until the button is pushed

        wait a specified time for the button to debounce

        call the new random number function

        send the new number to the output

        wait until the button is released

        wait a specified time for the button to debounce

        repeat from the top
}
```

When the experimental software and hardware are properly setup, each successive push of the button produces a new byte of output on the LED's. The first few values produces are: **0xF7, 0x48, 0xBC**. Our circuit produced the correct values for both the assembly language version as well as the C language version.

3 Conclusion

The desired outcome of this lab was successful, in that both implementations of the circuit operate properly. I feel that it was a valuable exercise in assembly language programming, as it introduced many important concepts, including modularity of subroutines, passing values into and returning values from a subroutine, translation of generic pseudo-code into assembly, project planing, and sourcecode optimization.

4 Appendices

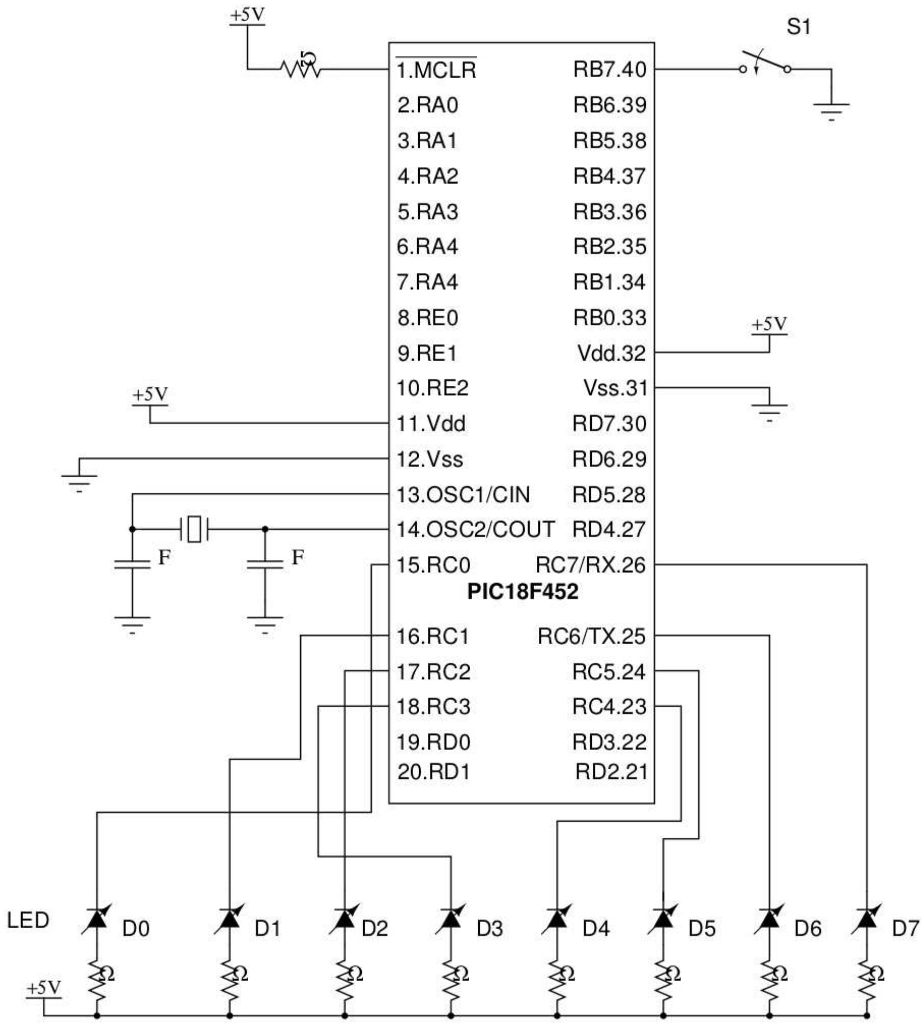

Figure 1: Circuit Schematic

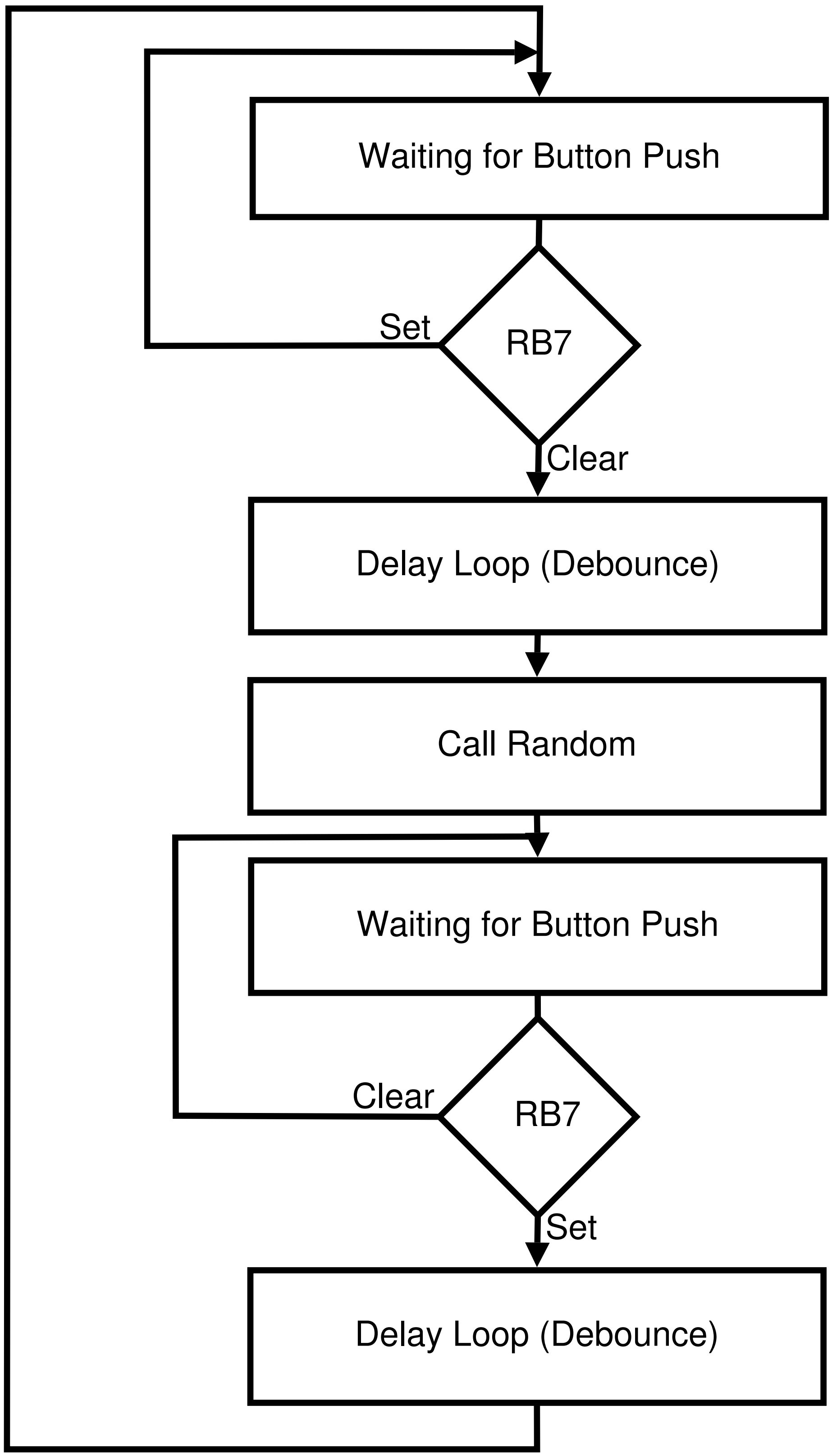

Figure 2: Program Flowchart

5 Assembly Language Program

```
        list    p=18F452
        include p18f452.inc
        radix   decimal

        cblock  0
        A0, A1, A2, A3
        B0, B1, B2, B3
        OUT0, OUT1, OUT2, OUT3
        COUNT0, COUNT1
        SEEDX0, SEEDX1, SEEDX2, SEEDX3
        SEEDY0, SEEDY1, SEEDY2, SEEDY3
        RANDA0, RANDA1, RANDB0, RANDB1
        endc

        org     0
        goto    main

        org     8
        retfie

        org     0x18
        retfie

main:
        clrf    A0
        clrf    A1
        clrf    A2
        clrf    A3
        clrf    B0
        clrf    B1
        clrf    B2
        clrf    B3
        clrf    OUT0
        clrf    OUT1
        clrf    OUT2
        clrf    OUT3
        clrf    SEEDX0
        clrf    SEEDX1
        clrf    SEEDX2
        clrf    SEEDX3
        clrf    SEEDY0
        clrf    SEEDY1
        clrf    SEEDY2
        clrf    SEEDY3
        clrf    RANDA0
        clrf    RANDA1
        clrf    RANDB0
        clrf    RANDB1
        ;initialize SEEDX
        movlw   0xB5
        movwf   SEEDX0
        movlw   0x3B
        movwf   SEEDX1
        movlw   0x12
        movwf   SEEDX2
        movlw   0x1F
        movwf   SEEDX3

        ;initialize SEEDY
        movlw   0xE5
        movwf   SEEDY0
        movlw   0x55
        movwf   SEEDY1
        movlw   0x9A
        movwf   SEEDY2
        movlw   0x15
        movwf   SEEDY3

        clrf    A0
        clrf    A1
        clrf    A2
        clrf    A3
        clrf    B0
        clrf    B1
        clrf    B2
        clrf    B3

        clrf    TRISC ;set output for PORTC
        setf    PORTC

        bcf     INTCON2, 7 ;enable internal pullups on PORTB

loop:
waitforswitchdown:
        btfsc   PORTB, 7
        goto    waitforswitchdown
        call    delay

        call    random
        movff   OUT0, PORTC

waitforswitchup:
        btfss   PORTB, 7
        goto    waitforswitchup
        call    delay

        goto    loop
```

```
random:
        ;-----------------------------------------------------------
        ; Pseud-Random Number Generator
        ; Inputs:   None
        ; Outputs:  From LSB to MSB: OUT0, OUT1, OUT2, OUT3

        clrf    A0
        clrf    A1
        clrf    A2
        clrf    A3
        clrf    B0
        clrf    B1
        clrf    B2
        clrf    B3
        clrf    OUT0
        clrf    OUT1
        clrf    OUT2
        clrf    OUT3

        movlw   0x50
        movwf   RANDA0
        movlw   0x46
        movwf   RANDA1
        movlw   0xB7
        movwf   RANDB0
        movlw   0x78
        movwf   RANDB1

        ;SEED_X = a*(SEED_X&65535) + (SEED_X>>16);
        movff   RANDA0, A0
        movff   RANDA1, A1
        clrf    A2
        clrf    A3
        movff   SEEDX0, B0
        movff   SEEDX1, B1
        clrf    B2
        clrf    B3
        call    multiply

        movff   OUT0, A0
        movff   OUT1, A1
        movff   OUT2, A2
        movff   OUT3, A3
        movff   SEEDX2, B0
        movff   SEEDX3, B1
        clrf    B2
        clrf    B3
        call    add
```

```
movff   OUT0, SEEDX0
movff   OUT1, SEEDX1
movff   OUT2, SEEDX2
movff   OUT3, SEEDX3

;SEED_Y = b*(SEED_Y&65535) + (SEED_Y>>16);
movff   RANDB0, A0
movff   RANDB1, A1
clrf    A2
clrf    A3
movff   SEEDY0, B0
movff   SEEDY1, B1
clrf    B2
clrf    B3
call    multiply

movff   OUT0, A0
movff   OUT1, A1
movff   OUT2, A2
movff   OUT3, A3
movff   SEEDY2, B0
movff   SEEDY3, B1
clrf    B2
clrf    B3
call    add

movff   OUT0, SEEDY0
movff   OUT1, SEEDY1
movff   OUT2, SEEDY2
movff   OUT3, SEEDY3

;put ((SEED_X&65535) + (SEED_Y&65535))/2; into OUT0->OUT3
movff   SEEDX0, A0
movff   SEEDX1, A1
clrf    A2
clrf    A3

movff   SEEDY0, B0
movff   SEEDY1, B1
clrf    B2
clrf    B3

call    add

rrcf    OUT3, F
rrcf    OUT2, F
rrcf    OUT1, F
rrcf    OUT0, F

return
```

```
delay:
        clrf    COUNT0
        clrf    COUNT1
delayloop:
        incf    COUNT0,f
        bnz     delayloop
        incf    COUNT1,f
        bnz     delayloop
        return

multiply:
        ;-----------------------------------------------------------
        ; 16x16 bit multiply
        ; Inputs:   From LSB to MSB: A0, A1, B0, B1
        ; Outputs:  From LSB to MSB: OUT0, OUT1, OUT2, OUT3
        clrf    OUT0
        clrf    OUT1
        clrf    OUT2
        clrf    OUT3

        movf    B0, W
        mulwf   A0
        movff   PRODL, OUT0
        movff   PRODH, OUT1

        mulwf   A1
        movf    PRODL, W
        addwf   OUT1, F    ;this might produce a carry into col2
        movf    PRODH, W
        addwfc  OUT2, F    ;this puts the new value (from PRODH)
                           ; into out2, with a possible carry
        movf    A0, W
        mulwf   B1
        movf    PRODL, W
        addwf   OUT1, F    ;this might produce a carry into col2
        movf    PRODH, W
        addwfc  OUT2, F    ;this might produce a carry into col3
        clrf    WREG       ;clear W to add to the OUT3
        addwfc  OUT3, F    ;this will take care of the carry

        movf    A1, W
        mulwf   B1
        movf    PRODL, W
        addwf   OUT2, F    ;this might produce a carry into col3
        movf    PRODH, W
        addwfc  OUT3, F

        return
```

```
add:
        ;-----------------------------------------------------------
        ; 32+32 bit addition
        ; Inputs:   From LSB to MSB: A0, A1, A2, A3; B0, B1, B2, B3
        ; Outputs:  From LSB to MSB: OUT0, OUT1, OUT2, OUT3
        ; Note:     Sets the Carry Status Flag when necessary
        movf    A0, W
        addwf   B0, W
        movwf   OUT0

        movf    A1, W
        addwfc  B1, W
        movwf   OUT1

        movf    A2, W
        addwfc  B2, W
        movwf   OUT2

        movf    A3, W
        addwfc  B3, W
        movwf   OUT3

        return

done:
        goto    done
        end
```

6 C Language Program

```
#include <p18f452.h>
static unsigned long int SEED_X = 521288629L;
static unsigned long int SEED_Y = 362436069L;

void delay(void)
{
        unsigned int count = 0;
        while (count < 0x7FFF)
        {
                count++;
        }
}

unsigned int random(void)
{
/* This function was given to us by the instructor
   in the laboratory guidelines */

        static unsigned int a = 18000, b = 30903;

        SEED_X = a*(SEED_X&65535) + (SEED_X>>16);

        SEED_Y = b*(SEED_Y&65535) + (SEED_Y>>16);

        return ((SEED_X&65535) + (SEED_Y&65535))/2;
}

void main(void)
{
        TRISC = 0;
        PORTC = 0xFF;
        INTCON2bits.RBPU = 0;

        while (1)
        {
                while (PORTBbits.RB7);
                delay();

                PORTC = random();

                while (!PORTBbits.RB7);
                delay();
        }
}
```