```
// lab2_skel.c
// Victor Garcia Flores
// 10.18.2019

//  HARDWARE SETUP:
//  PORTA is connected to the segments of the LED display. and to the pushbutton
s.
//  PORTA.0 corresponds to segment a, PORTA.1 corresponds to segement b, etc.
//  PORTB bits 4-6 go to a,b,c inputs of the 74HC138.
//  PORTB bit 7 goes to the PWM transistor base.

#define F_CPU 16000000 // cpu speed in hertz
#define TRUE 1
#define FALSE 0
#include <avr/io.h>
#include <math.h>
#include <util/delay.h>

//holds data to be sent to the segments. logic zero turns segment on
uint8_t segment_data[5];

//decimal to 7-segment LED display encodings, logic "0" turns on segment
//Note: They are arranged so that the value of a possible integer matched with t
he position
uint8_t dec_to_7seg[12] = {0b11000000, 0b11111001,0b10100100,0b10110000,0b100110
01,0b10010010,0b10000010,0b11111000,0b10000000,0b10011000,0b01111111,0b11111111}
;


//********************************************************************************
//                            chk_buttons
//Checks the state of the button number passed to it. It shifts in ones till
//the button is pushed. Function returns a 1 only once per debounced button
//push so a debounce and toggle function can be implemented at the same time.
//Adapted to check all buttons from Ganssel's "Guide to Debouncing"
//Expects active low pushbuttons on PINA port.  Debounce time is determined by
//external loop delay times 12.
//
uint8_t chk_buttons(uint8_t button) {
        static uint16_t state[8] = {0}; //We do what we did in lab 1, but this t
ime as an array so we can address the other buttons
        state[button] = ((state[button]<<1) | (!bit_is_clear(PINA,button)) | 0xE
000);
        if(state[button] == 0xF000) return 1;
        return 0;
}
//********************************************************************************

//********************************************************************************
*****
//                                  segment_sum

//takes a 16-bit binary input value and places the appropriate equivalent 4 digi
t
//BCD segment code in the array segment_data for display.
//array is loaded at exit as:  |digit3|digit2|colon|digit1|digit0|
void segsum(uint16_t sum) {
        //Variables for values digit  positions
        uint8_t OnesVal;
        uint8_t TensVal;
        uint8_t HundredsVal;
        uint8_t ThousandsVal;

        //Decoder "Sel#" Positions for Digits. Note: The lower bytes are 0 becau
se we aren't using them in PORTB. This also keeps in mind the value we desire in
 PWN
        //determine how many digits there are
        int NumDigits = 0;
        int tempSum = sum;
```



```
        while(sum){
                tempSum /= 10;
                NumDigits++;
        }

        //break up decimal sum into 4 digit-segments
        //---ONES---
        OnesVal = sum % 10;
        segment_data[0] = OnesVal;

        //--- Tens ---
        TensVal = (sum/10) % 10;
        segment_data[1] = TensVal;

        //--- HUNDREDS ---
        HundredsVal = (sum/100) % 10;
        segment_data[3] = HundredsVal;

        //--- THOUSANDS ---
        ThousandsVal = (sum/1000) % 10;
        segment_data[4] = ThousandsVal;

        //DDRA = 0xFF; //Make PORT A an OUTPUT
        if(sum<10){ //if there is only one digit
                //1st Set
                PORTB = 0x00;
                PORTA = dec_to_7seg[OnesVal];
                _delay_ms(2);
                PORTB = 0x40;
                PORTA = 0b11111111;
        }
        else if((sum >= 10) && (sum < 100)){ //if there are two digits
                //1st Set
                PORTB = 0x00;
                PORTA = dec_to_7seg[OnesVal];
                _delay_ms(2);

                //2nd Set
                PORTB = 0x10;
                PORTA = dec_to_7seg[TensVal];

        }
        else if((sum>=100)&&(sum<1000)){ //if there are three digits
                //1st Set
                PORTB = 0x00;
                PORTA = dec_to_7seg[OnesVal];
                _delay_ms(2);

                //2nd Set
                PORTB = 0x10;
                PORTA = dec_to_7seg[TensVal];
                _delay_ms(2);

                //3rd Set
                PORTB = 0x30;
                PORTA = dec_to_7seg[HundredsVal];
        }
        else if(sum>= 1000){ //if there are four digits
                //1st Set
                PORTB = 0x00;
                PORTA = dec_to_7seg[OnesVal];
                _delay_ms(2);

                //2nd Set
                PORTB = 0x10;
                PORTA = dec_to_7seg[TensVal];
                _delay_ms(2);

                //3rd Set
```

```
                PORTB = 0x30;
                PORTA = dec_to_7seg[HundredsVal];
                _delay_ms(2);

                //4th Set. Note: No segments need clearing.
                PORTB = 0x40;
                PORTA = dec_to_7seg[ThousandsVal];
        }
        /*
        //blank out leading zero digits
        if(NumDigits == 1){
                //blank three digit locations
                segment_data[1] = 0xFF; //remember, this will be what turns it o
ff
                segment_data[3] = 0xFF;
                segment_data[4] = 0xFF;
        }
        else if(NumDigits == 2){
                //blank two digit locations
                segment_data[3] = 0xFF;
                segment_data[4] = 0xFF;
        }
        else if(NumDigits == 3){
                //blank one digit locations
                segment_data[4] = 0xFF; //remember, this will be what turns it o
ff
        }*/

        //now move data to right place for misplaced colon position

}//segment_sum
//*********************************************************************************
****


//*********************************************************************************
*****
// Function Name:void AllSegments_BitClearer
// This function is put to clear previous digit values on the seven segment disp
lay.
// Goal: The goal is to avoid ghosting and help set un-used segments to zero.
void AllSegments_BitClearer(){
        DDRA = 0xFF;
        //Ones
        PORTB = 0x00;
        PORTA = 0b11111111;
        _delay_ms(2);

        //Tens
        PORTB = 0x10;
        PORTA = 0b11111111;
        _delay_ms(2);

        //Hundreds
        PORTB = 0x30;
        PORTA = 0b11111111;
        _delay_ms(2);

        //Thousands
        PORTB = 0x40;
        PORTA = 0b11111111;
        _delay_ms(2);
}
//*********************************************************************************
*****

//*********************************************************************************
*****
int main()
```



```
{
        //variable for current value
        uint16_t CurrCountVal = 0;

        //set port bits 4-7 B as outputs
        DDRB = 0xF0; //This addresses the upper bits like we want
        while(1){
                //Clear 4 segments
                AllSegments_BitClearer();

                //insert loop delay for debounce
                _delay_ms(2);

                //make PORTA an input port with pullups
                DDRA = 0x00; //sets as input
                PORTA = 0xFF; //pulls up the resistors

                //enable tristate buffer for pushbutton switches
                PORTB |= ((1<<PB6)|(1<<PB5)|(1<<PB4)); //Note: This was selected
 so that other outputs would not be effected (value = 0b101)

                //now check each button and increment the count as needed
                for(int BttnNum = 0;BttnNum <= 7;BttnNum++){
                        //If a certain button at position x is pressed
                        if(chk_buttons(BttnNum)){
                                // Fing out which button was pressed and increme
nt accordingly
                                if(BttnNum == 0){
                                        CurrCountVal += 1;
                                }
                                else if(BttnNum == 1){
                                        CurrCountVal += 2;
                                }
                                else if(BttnNum == 2){
                                        CurrCountVal += 4;
                                }
                                else if(BttnNum == 3){
                                        CurrCountVal += 8;
                                }
                                else if(BttnNum == 4){
                                        CurrCountVal += 16;
                                }
                                else if(BttnNum == 5){
                                        CurrCountVal += 32;
                                }
                                else if(BttnNum == 6){
                                        CurrCountVal += 64;
                                }
                                else if(BttnNum == 7){
                                        CurrCountVal += 128;
                                }
                        }
                }

                //disable tristate buffer for pushbutton switches
                PORTB = 0x00;

                //bound the count to 0 - 1023
                if(CurrCountVal > 1023){
                        CurrCountVal = 1; //if the count goes over what is asked
, re-initiate count at 1 (part of instructions in lab).
                }

                //Set PORTA as an OUTPUT
                DDRA = 0xFF;

                //Break up the disp_value to 4, BCD digits in the array: call (s
egsum)
                segsum(CurrCountVal);
```

```
                _delay_ms(2);

        }//while
}
```