```
// lab2.c
// Andrey Kornilovich
// 10.14.19

//  HARDWARE SETUP:
//  PORTA is connected to the segments of the LED display. and to the pushbuttons.
//  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 <util/delay.h>
#include <math.h>

// definitions for segment pins and port B control pins
#define SEG_A 0x01
#define SEG_B 0x02
#define SEG_C 0x04
#define SEG_D 0x08
#define SEG_E 0x10
#define SEG_F 0x20
#define SEG_G 0x40
#define SEG_DP 0x80

#define DEC_1 0x10
#define DEC_2 0x20
#define DEC_3 0x40
#define PWM 0x80

//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
uint8_t dec_to_7seg[12];

// write to dec_to_7seg all the pins to display 0-9, blank, and the decimal point
void encode_chars(){
  dec_to_7seg[0] = ~(SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F); //0
  dec_to_7seg[1] = ~(SEG_B | SEG_C); //1
  dec_to_7seg[2] = ~(SEG_A | SEG_B | SEG_G | SEG_E | SEG_D); //2
  dec_to_7seg[3] = ~(SEG_A | SEG_B | SEG_C | SEG_G | SEG_D); //3
  dec_to_7seg[4] = ~(SEG_F | SEG_G | SEG_B | SEG_C); //4
  dec_to_7seg[5] = ~(SEG_A | SEG_F | SEG_G | SEG_C | SEG_D); //5
  dec_to_7seg[6] = ~(SEG_A | SEG_F | SEG_G | SEG_C | SEG_D | SEG_E); //6
  dec_to_7seg[7] = ~(SEG_A | SEG_B | SEG_C); //7
  dec_to_7seg[8] = ~(SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G); //8
  dec_to_7seg[9] = ~(SEG_A | SEG_B | SEG_F | SEG_G | SEG_C | SEG_D); //9
  dec_to_7seg[10] = 0xFF; //display nothing
  dec_to_7seg[11] = ~(SEG_DP); //DP
}

// calling this sets PORT B to output to a specific digit
void pick_digit(int digit){
  //set the correct port B output without clobbering the rest of the register
  switch (digit){
    //first (msb) digit, Y4 on decoder
    case 0:
      PORTB &= ~(DEC_1 | DEC_2); //&= to clear decoder control pins
      PORTB |= DEC_3;            //|= set decoder control pin
      break;
    //second digit, Y3 on decoder
    case 1:
      PORTB &= ~(DEC_3);
      PORTB |= (DEC_1 | DEC_2);
      break;
    //third digit, Y1 on decoder
    case 2:
      PORTB &= ~(DEC_2 | DEC_3);
      PORTB |= DEC_1;
      break;
    //fourth (lsb) digit, Y0 on decoder
```



```
    case 3:
      PORTB &= ~(DEC_1 | DEC_2 | DEC_3);
      break;
    //colon, Y2 on decoder
    case 4:
      PORTB &= ~(DEC_1 | DEC_3);
      PORTB |= DEC_2;
      break;
    //enable button board, Y7 on decoder
    case 5:
      PORTB |= (DEC_1 | DEC_2 | DEC_3);
      break;
    //no digit or button board (off), Y6 on decoder
    case 6:
      PORTB &= ~(DEC_1);
      PORTB |= (DEC_2 | DEC_3);
      // break;
    default:
      break;
  }
}

//*************************************************************************
//                              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}; //holds present state
  state[button] = (state[button] << 1) | (! bit_is_clear(PINA, button)) | 0xE000;
  if (state[button] == 0xF000) return 1;
  return 0;
}

//*************************************************************************************
//                                      segment_sum
//takes a 16-bit binary input value and places the appropriate equivalent 4 digit
//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) {

  //break up decimal sum into 4 digit-segments
  segment_data[0] = dec_to_7seg[sum/1000]; //msb
  segment_data[1] = dec_to_7seg[(sum/100) % 10];
  segment_data[2] = dec_to_7seg[(sum/10) % 10];
  segment_data[3] = dec_to_7seg[sum % 10]; //lsb
  segment_data[4] = dec_to_7seg[10]; //assign empty value for colon

  //blank out leading zero digits
  int segs;
  for(segs = 0; segs < 3; segs++){
    //if segment is 0, blank it out
    if(segment_data[segs] == dec_to_7seg[0]) {segment_data[segs] = dec_to_7seg[10];}
    else {break;}
  }
  //now move data to right place for misplaced colon position
}//segment_sum

//*************************************************************************************
uint8_t main()
{
//set port bits 4-7 B as outputs
DDRB = 0xF0; // 1 for output, 0 for input

//define counters, and call array initializers
uint16_t num_to_display = 0;
encode_chars();
int button = 0;
int digit = 0;
```

```
while(1){
  //make PORTA an input port with pullups
  DDRA = 0x00;
  PORTA = 0xFF;

  //enable tristate buffer for pushbutton switches
  pick_digit(5);

 //now check each button and increment the count as needed
  int button;
  for(button = 0; button < 8; button++){
    if(chk_buttons(button)) {num_to_display += (1 << button);} //shift left
  }

  //bound the count to 0 - 1023
  if(num_to_display >= 1024) {num_to_display = 1;}

  //break up the disp_value to 4, BCD digits in the array: call (segsum)
  segsum(num_to_display);

  //make PORTA an output
  DDRA = 0xFF;

  //assign port A and display to a digit
  PORTA = segment_data[digit];
  pick_digit(digit);

  //increment the digit and reset
  digit++;
  if(digit >= 4) {digit = 0;}

  //delay for debounce
  _delay_ms(2);

  }//while
  return 0;
}//main
```