Laboratory Worksheet #10
Keypad/LCD Input Exercise

This worksheet is an activity to learn how to use the functions associated with reading from the keypad and writing to the LCD display. In completing the requested C-program you will develop a segment of code that can be integrated into your Lab 4, 5 and 6 exercises. This will permit you to input multi-digit values from the LCD keypad while ignoring multiple inputs of the same key because the user didn't release the button quickly enough. This is very similar to the pushbutton switch debouncing that you have already performed for the Lab 2 exercise.

Exercise 1: Inputting a single keypad character

1) Download the `kpdlcdtestPCA.c` code.
2) Change XBR0 to be consistent with your lab 3 code.
3) Connect the LCD/Keypad to your protoboard, using the header pins to connect to power, ground, SDA and SCL.

   When you put the LCD/Keypad away, make sure the header pin is connected to the ribbon cable, not left on your protoboard.
4) Verify the hardware and software are setup correctly by running the `kpdlcdtestPCA.c` code and checking the output on both Putty and the LCD.

   `Note: You should see multiple lines being printed.`
5) Change the code so that a single push and release of a keypad button results in a single read and print (on both Putty and the LCD).

   This implementation is very similar to reading a pushbutton press in Lab 2, with different indicators for press and release.
6) Add code that converts numeric characters to their decimal value. Add a print statement that prints the decimal value, using %u typecasting.

Report the results of your print statement.

Button that was pressed

5

Print output for ASCII value

Character: 53 Hex: 35

Print output of decimal value

Decimal: 5

Exercise 2: Inputting a multdigit number

1) Change the Exercise 1 code to read a two digit number. The first key input should be the 10s digit and the second key input should be the ones digit. Print the number, using %u typecasting to verify your code is working correctly.

   Single digit values may be entered by pressing '0' for the first digit.

2) change your program to use the function `kpd_input(char mode)` to accept a multi-digit unsigned integer using the keypad.. Your program should take the unsigned integer value returned by the function and print it on the SecureCRT terminal (not the LCD panel). Try both modes [`kpd_input(0)` and `kpd_input(1)`] and note the differences in the way they work.

3) Note what happens when a value is entered outside the range of 0 – 65535. Enter 65536 and 65537 to see what value is actually returned.

4) Find the equation that gives the actual value returned by the function when the input value is outside its allowed range.

5) Predict what will be returned when 99999 is entered.

Result of entering 65536

0

Result of entering 65537

1

Equation for actual value when any 5 digit number greater than 65535 is input

x-65536

Result of entering 99999

34463

There are some critical issues with the use of kpd_input() that may cause the 8051 to freeze. It seems if the I2C bus communication sequence is interrupted (a PCA0 interrupt) in the middle of a transaction the processor locks up waiting for something that will never occur. The best way to avoid this problem, other than repeatedly disabling interrupts just before using kpdinput(), is to keep your PCA ISR short and efficient. Also make sure you are NOT using any Timer 0 interrupts. They are no longer necessary for anything. When printing values received from kpdinput() remember to use %u rather than %d since they must be declared as unsigned int.

This program should be completed BEFORE integrating together the two parts of Lab 3 into a single program to control both the steering and speed of the car in Lab 4.

When complete, include Worksheet 10 with your Laboratory 4 Pre-lab submission.

YILU ZHOU

EVB Pin			Port Bit	Bit Addresses & Labels
1	2	1.		
		2.		
3	4	3.		
		4.		
5	6	5.		
		6.	P1.4	0x94 POT
7	8	7.		
		8.		
9	10	9.		
		10.	P1.2	0x92 CEX2
11	12	11.	P1.3	0x93 CEX3
		12.	P1.0	0x90 CEX0
13	14	13.		
		14.	P0.6	0x86 SDA
15	16	15.	P0.7	0x87 SCL
		16.		
17	18	17.		
		18.		
19	20	19.		
		20.		
21	22	21.		
		22.		
23	24	23.		
		24.		
25	26	25.		
		26.		
27	28	27.		
		28.		
29	30	29.		
		30.		
31	32	31.	P3.6	0xB6 SS
		32.		
33	34	33.		
		34.		
35	36	35.		
		36.		
37	38	37.	P3.1	0xB1 BILED1
		38.	P3.0	0xB0 BILED0
39	40	39.		
		40.		

41 ⟷ 60

Software Initializations

A) Port I/0

```
P1MDIN &= 0xEF;
P1MDOUT &= 0xEF
P1MDOUT |= 0x0D;
P1 |= ~0xEF;
P3MDOUT &= 0xBF;
P3MDOUT |= 0x03;
P3 |= ~0xBF;
```

B) Timers

C) Interrupts

```
EA = 1;
EIE1 |= 0x08;
```

D) A/D

```
REF0CN = 0x03;
ADC1CN = 0x80;
ADC1CF |= 0x01;
```

E) PCA

```
PCA0MD =0x81;
PCA0CPM2 = 0xC2;
PCA0CN = 0x40;
```

F) XBAR

```
XBR0 = 0x27;
```

G) I2C

```
SMB0CR = 0x93;
ENSMB = 1;
```

compile derivatives
  #include <studio.h>
  #include <<stdlib.h>
  #include <c8051_SDCC.h>
  #include <i2c.h>

declare global variables
  const unsigned char RANGER_ADDRESS, COMPASS_ADDRESS, POT_ADDRESS.
  unsigned char RANGER_DATA[2], COMPASS_DATA[2], SHARED_DATA[2], RANGER_READ_INDICATOR, COMPASS_READ_INDICATOR, AD_VALUE, OUTPUT_MODE.
  unsigned int DISTANCE, HEADING, DRIVE_MOTOR_PW, STEERING_SERVO_PW, PCA_COUNTER, DRIVE_MOTOT_MAX, DRIVE_MOTOR_MIN, STEERING_SERVO_LEFT, STEERING_SERVO_RIGHT, TARGET_HEADING, TARGET_DRIVE_MOTOR_GAIN, TARGET_STEERING_SERVO_GAIN, PCA_START, DRIVE_MOTOR_NEUT AND STEERING_SERVO_NEUT.
  int HEADING_ERROR.
  sbit POT, SS, CF,BILED0, BILED1.

function prototypes
  void Port_Init(void);
  void PCA_Init(void);
  void XBR0_Init(void);
  void Interrupt_Init(void);
  void SMB_Init(void);
  void ADC_Init(void);
  void PCA_ISR(void) __interrupt 9;
  void Drive_Motor(void);
  void Speed_Controller(void);
  void Ping_Ranger(void);
  void Read_Ranger(void);
  void Drive_Motor_Init(void);
  void Steering_Servo(void);
  void Direction_Controller(void);
  void Read_Compass(void);
  void Steering_Servo_Init(void);
  void Read_AD_Input(void);
  void POT_Reader(void);
  void Read_Keypad(void);
  void Flight_Recorder(void);
  unsigned int Is_SS_On(void);
  void Turn_BILED_Green(void);
  void Turn_BILED_Red(void);
  void Turn_BILED_Off(void);
  void Reset_PCA_Counter(void);
  void Wait_For_1s(void);

main function
  declare local variables
    NONE
  initialize system, ports and PCA_COUNTER
    Sys_Init();
    putchar(' ');
    Port_Init();
    PCA_Init();

```
    XBR0_Init();
    Interrupt_Init();
    SMB_Init();
    ADC_Init();
  Turn off the BILED.
  Read the data from POT using POT_Reader.
  initialize the drive motor using Drive_Motor_Init.
  initialize the steering servo using Steering_Servo_Init.
  Begin infinite loop
    If the slide switch is on
      Turn BILED to green
      Set the drive motor PW to forward
      Drive the motor
      Wait for 3s
      Begin infinite loop while the SS is on
        wait for 3 second
        call Read_Ranger
        call Speed_Controller
        call Drive_Motor
        call Direction_Controller
        call Steering_Servo
    End if
    Else if the slide switch is off
      read the input from the keypad
      set DRIVE_MOTOR_PW to DRIVE_MOTOT_MAX
      call Drive_Motor
      set STEERING_SERVO_PW to STEERING_SERVO_NEUT
      call Steering_Servo
    End if
  End infinite loop

functions
  void Drive_Motor(void)
    Set the low byte of the CEX2
    Set the high byte of CEX2
  End function

  void Speed_Controller(void)
    If the DISTANCE is larger than 20cm
      Set the PW of drive motor to forward
      Turn the BILED to green
      Set the target heading to east.
    End if
    else if the DISTANCE if smaller than 20cm
      Set the PW of the drive motot to reverse
      Turn the BILED to red
      Set the target heading to west
    End if
  End function

  void Ping_Ranger(void)
    write the register 0 of the ranger to read the DISTANCE in cm.
  End function
```

```
void Read_Ranger(void)
   Read the data from register 2 and 3 from the ranger
   Convert the read data to DISTANCE
   Use Ping_Ranger to send the signal again
End function

void Drive_Motor_Init(void)
   Set the PW of the drive motor to neutal
   Turn the drive motor on for neutral
   Wait for 1s
End function

void Steering_Servo(void)
   Update the low byte of CEX0
   update the high byte of CEX0
End function

void Steering_Servo_Init(void)
   Set the PW of the steering servo to neutral
   Turn the steering servo on for neutral
   Wait for 1s
End function

void Direction_Controller(void)
   Read the data from compass using Read_Compass
   calculate the HEADING_ERROR
   If the HEADING_ERROR is larger 1800
      HEADING_ERROR minus 3600
   End if
   If the HEADING_ERROR is smaller than -1800
      HEADING_ERROR plus 3600
   End if
   Set the corresponding PW for the steering servo
   If the PW for the steering servo is larger than the STEERING_SERVO_RIGHT
      Set the PW to the STEERING_SERVO_RIGHT.
   End if
   If the PW fro the steering servo is smaller than the STEERING_SERVO_LEFT
      Set the PW to the STEERING_SERVO_LEFT.
   End if
End function

void Read_Compass(void)
   Read the data from the compass register 2 and 3
   Convert the read data to HEADING
End function

void Read_AD_Input(void)
   Set P1.4 as the analog input for the POT
   Clear the "Conversion Completed" flag
   Initiate A/D conversion
   Wait for conversion to complete
   Set the AD_VALUE
End function
```

```
void POT_Reader(void)
   Read the AD_VALUE using Read_AD_Input().
   calculate the max of the drive motor
   calcualte the min of the drive motor.
   calculate the right PW for the steering servo.
   calculate the left PW for the steering servo.
End function

void Read_Keypad(void)
   Get the target heading from the keypad
   If the value is out of range
      Ask the user to input another value
   End if
   Get the TARGET_STEERING_SERVO_GAIN from the keypad
   Get the TARGET_DRIVE_MOTOR_GAIN from the keypad
End function

void Flight_Recorder(void)
   if the MODE is not 1
      Output to user-friendly UI.
   End if
   else
      Print DISTANCE, HEADING, AD_VALUE, DRIVE_MOTOR_PW, STEERING_SERVO_PW
   End if
End function

unsigned int Is_SS_On(void)
   if SS is on
      return 1
   End if
   else if SS is off
      return 0
   End iff
End function

void Turn_BILED_Green(void)
   turn red LED off
   turn green LED on
End function

void Turn_BILED_Red(void)
   turn red LED on
   turn green LED off
End function

void Turn_BILED_Off(void)
   turn red LED off
   turn green LED off
End function

void PCA_Init (void)
   Enable SYSCLK/12 and enable interrupt.
   Enable CCM2 16bit PWM.
   Enable PCA counter.
```

```
   For 20ms period.
End function

XBR0_Init(void)
   Configure crossbar with UART, SPI, SMBus, and CEX channels.
End function

void Interrupt_Init(void)
   Enable general interrupt.
   Enable PCA overflow interrupts.
End function

void SMB_Init(void)
   Set the clock frequency to be 100kHz.
   Enable SMB.
End function

void ADC_Init(void)
   configure ADC1 to use VREF.
   Set a gain of 1.
   Enable ADC1.
End function

void PCA_ISR(void) __interrupt 9
   increment the PCA_COUNTER
   if 3s has passed
      set RANGER_READ_INDICATOR to 1
   End if
   if 1s has passed and switch is on
      call Flight_Recorder
   End if
   if CF
      Clear overflow flag.
      start count for 20ms period
   End if
   handle other PCA0 overflows
End function

void Reset_PCA_Counter(void)
   reset PCA_COUNTER to 0
End function

void Wait_For_1s (void)
   reset PCA_COUNTER
   Begin infinite loop while PCA_COUNTER less than 51
   wait for 1s
   End infinite loop
   reset PCA_COUNTER
End function
```