EVB Pin			Port Bit	Bit Addresses & Labels
1	2	1.		
		2.		
3	4	3.	3.3Volt	3.3Volts
		4.		
5	6	5.	P1.7	A/D input
		6.		
7	8	7.		
		8.		
9	10	9.		
		10.	P1.2	Moter
11	12	11.	P1.3	LED
		12.	P1.0	Steering
13	14	13.		
		14.	P0.6	SDA
15	16	15.	P0.7	SCL
		16.		
17	18	17.		
		18.		
19	20	19.		
		20.	P0.0	TX0 for RF
21	22	21.	P0.1	RX0 for RF
		22.		
23	24	23.		
		24.		
25	26	25.		
		26.		
27	28	27.		
		28.		
29	30	29.		
		30.		
31	32	31.		
		32.	P3.7	motor&steering control
33	34	33.		
		34.		
35	36	35.		
		36.		
37	38	37.		
		38.		
39	40	39.		
		40.		
41 ⟷ 60				

Software Initializations

A) Port I/0

```
P1MDOUT |= 0x0C
P3MDOUT &= ~0x80
P3 |= 0x80
P1MDIN &= ~0x80
P1 |= 0x80
P1MDOUT &= ~0x80
```

B) Timers

C) Interrupts

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

D) A/D

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

E) PCA

```
PCA0CN = 0x40
PCA0MD = 0x81
PCA0CPM0 = 0xC2
PCA0CPM2 = 0xC2
```

F) XBAR

```
XBR0 = 0x27
```

G) I2C

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

File needed

```
#include <c8051_SDCC.h>
#include <stdlib.h>// needed for abs function
#include <stdio.h>
#include <i2c.h>
```

Function prototype

```
void Port_Init(void);
void PCA_Init(void);
void SMB_Init(void);
void ADC_Init(void);
void Interrupt_Init(void);
void PCA_ISR(void) __interrupt 9;
int  read_compass(void);
void set_servo_PWM(void);
int  read_ranger(void);        // new feature - read value, and then start a new ping void
set_drive_PWM(void);
int  pick_heading(void);       // function which allow operator to pick desired heading


//define global variables
unsigned int PW_CENTER = _____;
unsigned int PW_RIGHT = _____;
unsigned int PW_LEFT = _____;
unsigned int SERVO_PW = _____;
unsigned int SERVO_MAX = _____;
unsigned int SERVO_MIN = _____;
unsigned char new_heading = 0;  // flag for count of compass timing
unsigned char new_range = 0;    // flag for count of ranger timing
unsigned char print_flag = 0;   // flag for count of printing
unsigned int  heading;
```

```
unsigned int  range;
unsigned int  light;
int compass_adj = 0;          // correction value from compass
int range_adj = 0;            // correction value from ranger
unsigned char r_count;        // overflow count for range
unsigned char h_count;        // overflow count for heading
unsigned char print_count;    // overflow count for printing

__sbit __at ____ RUN // a slide switch
```

Main function

Declare local variables

None

Funcion initialization

Do infinite while loop

If the run switch is stop position

Set the motor stop

Set the steer parallel to the car

Else

pick desired heading and range

If different heading

Read heading

Adjust servo PW

Flag to off

If different range

Read range (also the led data)

Flag off

If an object detected

Adjust steering PW

Adjust speed(motor PW)

End main function

Other function

(most of them use the LAB2 / LAB3 functions)

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

7

Print output for ASCII value

Character: 7 Hex: 0x37

Print output of decimal value

Decimal: 55

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

Output = Input + 65536

Result of entering 99999

159

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.