ECE 3724/CS 3124 Test #3 – Spring 2005- Reese. You may use only the provided reference materials. All figures are on the last page.

Part I: (64 pts)

a. (5 pts) Write *C* code that configures PORTB for the IO shown in the figure for problem (a) on the Figure sheet. The internal weak pullup must be enabled. Do not assume any default bit values.

```
//one solution
RBPU = 0;
TRISB1 = 1;
TRISB5 = 1;
TRISB7 = 0;
```

```
//another solution
RBPU = 0;
TRISB = 0x7F;
```

b. (14 pts) Assuming the IO configuration of the previous problem, write a *while(1){}* loop that implements the LED/Switch IO state machine shown for problem (b) in the figures. Either use a *switch()* statement approach or a *if-then-else* approach. Assume you have available the *DelayMs()* function for blinking the LED; you do NOT have to include debounce delays for the switch input.

```
char state;
main(){
 //configure ports, not shown
 state = BLINK;
 while (1){
  switch(state) {
   case BLINK:
     while (RB1) {  // while not pressed
             if (LB7) LB7 = 0; else LB7 = 1;
              DelayMs(200);
              }
    state = ON_1;
    break;
   case ON_1:   RB7 = 1; // while pressed
     while (!RB1);
     if (RB5) state = BLINK; else state = OFF;
     break;
    case OFF: RB7 = 0;
          while (RB1);   // press & Release
          while (!RB1);
          state = ON_2
          break;
     case ON_2:
           RB7 = 1;
          while (RB1);   // press & Release
          while (!RB1);
           state = OFF; break;
       }
     }
```

c. (20 pts) For the LED/Switch configuration shown in Figure (c), implement the flowchart of figure (c) in an *interrupt driven manner*. Divide your solution into two code segments -- an ISR, and *main()* code that includes initialization code for the interrupt system, variables used by the ISR, and initializes the LEDs to OFF. Any changes to an LED must be done within the ISR; the *while(1)* loop in *main()* will BE EMPTY; all of the work of changing the LED state is done in the ISR. This is possible because we are not blinking the LEDs and thus do not need any delays in the ISR. There are many solutions to this problem. Your ISR must be triggered by changes on the switch inputs; it cannot wait for a switch input change to occur.

1. (13 pts) ISR code (do not worry about debouncing the switch inputs).

2. (7 pts) *main()* code – you can assume PORT inputs/outputs and the weak pullup has been initialized. However, you must show your configuration code for the interrupt system, any variables you use, and turn the LEDs initially OFF. I will require that you disable interrupt priorities to keep things simple. Any interrupt flags you use must be initially cleared. Do not assume any default bit settings.

```
// solution #1
void interrupt my_isr() {
 if (INT1IF && INT1IE) {
    INT1IF = 0;
  INT1IE = 0; INT2IE = 1; // enable SW_B
 if (LB1) LB6 = 0; else LB6 = 1; //toggle LED
 INT2IF = 0; clear just in case sw_b pressed
 }
 else if (INT2IF && INT2IE) {
    INT2IF = 0;
  INT2IE = 0; INT1IE = 1; // enable SW_A
 if (LB7) LB7 = 0; else LB7 = 1; //toggle LED
 INT1IF = 0; // clear just in case sw_a pressed
 }
 }
// this solution uses interrupt enables to
// enable/disable SW_A, SW_B in sequence
main(){
  // configure ports, not show
 INT1IF = 0;
 INT1IE = 1; // enable INT1
INT2IF = 0;
 INT2IE = 0; //disable INT2
 IPEN = 0;  // no priorities
// rising edge interrupts
INTEDG1 = 1; INTEDG2 = 1;
RB7 = 0; RB6 = 0; //both LEDs off
PEIE = 1; // not really needed
GIE = 1;    // enable interrupts
 while(1){
   //empty
 }
}
```

```
// solution #2
char state;
void interrupt my_isr() {
 if (INT1IF && state == 0) {
    INT1IF = 0;
  if (LB1) LB6 = 0; else LB6 = 1; //toggle LED
  state = 1;  // allow LED_B to toggle
 INT2IF = 0; // clear just in case
 }
else if (INT2IF && state == 1) {
    INT2IF = 0;
  if (LB7) LB7 = 0; else LB7 = 1; //toggle LED
  state = 0;
  INT1IF = 0; // clear just in case
  }
 }
// this solution uses the state variable
// to implement sequencing of LEDs
main(){
  // configure ports, not show
 INT1IF = 0;
 INT1IE = 1; // enable INT1
INT2IF = 0;
 INT2IE = 1; //enable INT2
 IPEN = 0;  // no priorities
// rising edge interrupts
INTEDG1 = 1; INTEDG2 = 1;
RB7 = 0; RB6 = 0; //both LEDs off
PEIE = 1; // not really needed
GIE = 1;    // enable interrupts
state = 0;    // state is 0 first.
 while(1){
   //empty
 }
}
```

d. (5 pts) Assume an asynchronous serial channel with a baud rate of 38400, and an asynchronous data format of 1 start bit, 8 data bits, with 6 stop bits between characters. Compute the bandwidth of this channel in BYTES per SECOND.

15 bits * 1/38400 = 390 us to transmit one byte

1/390 us = 2560 bytes transmitted per second

e. (7 pts) Write *C* code that implements the *char getch()* function (receives one character from the serial port). *No interrupts are enabled*. Upon entry to the *getch()* function, check for UART overrun – if this has occurred then execute a software reset by using inline assembly code.

```
char getch(){
  if  (OERR) {
     asm("reset");
   }
  while (!RCIF);
  return(RCREG);
}
```

f. (8 pts) Write an ISR that is triggered by the arrival of a character in the USART and that places the incoming data into a circular buffer name *buf*. Assume the buffer has a maximum of 16 characters, and pointers named *tail* and *head*. The *head* pointer is used for placing data into the buffer, while the tail is used for taking data out of the buffer.

```
void interrupt my_isr (){
  if  (RCIF) {
     head++
    if (head == 16) head = 0;  // wrap pointer
   buf[head] = RCREG;
 }
}
```

g. (5 pts) Write a *C* code fragment to go at the beginning of *main()* that detects if a power-on reset has occurred and prints out a message. You must also ensure that after a power-on reset has occurred, that it is not falsely detected again by a non-power-on reset.

```
if (!POR) {
   printf (" Power-on Reset has occurred!");
   POR = 1;   // set bit to avoid false detection
}
```

Part II: (40 pts) Answer 9 out of the next 11 questions. Cross out the 3 questions that you do not want graded. Each question is worth 4 pts.

1. Given a voltage of 5 V, a clock freq of 30 MHz, and a current consumption of 15 mA, what is the new current consumption predicted by theory if the voltage is reduced to 3.5 V and the clock frequency to 20 MHz?

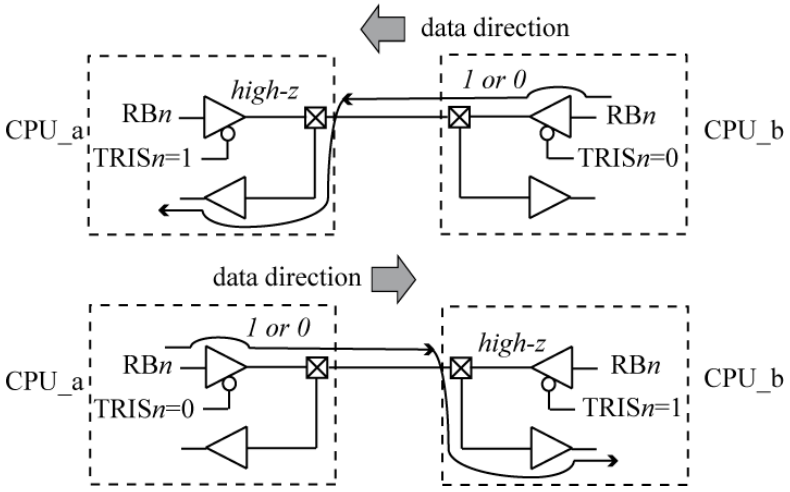
```
15 mA (3.5 * 3.5 * 20)/(5 * 5 * 30) = 4.9 mA
```

2. Write a *C* code fragment that puts the PIC18 into sleep mode. What is the principle reason for using sleep mode on a microcontroller?

```
asm("sleep");   //puts PIC into sleep mode.

Sleep mode saves power by turning off the main clock.
```

3. Draw a picture that shows how tri-state buffers are used to implement a half-duplex communication channel



4. What is the SPBRG value for a baud rate of 57,600 assuming an FOSC of 30 MHz and high speed mode?

SPBRG = 30 MHz/(16 * 57600) – 1 = 31.56, round to 32.

5. During normal PIC operation, how do you prevent the watch dog timer (WDT) from expiring?

Execute the "clrwdt" instruction periodically (clear watchdog timer)

6. What voltage levels (approximately) represent logic '0' and logic '1' in the RS232 interface? Be specific.

An RS232 "0" is between +3V to +25V.
An RS232 "1" is between –3V to –25V.

7. How is reading the LATB register different from reading the PORTB register?

A read of LATB returns the last value written to PORTB; LATB contains the value used to drive the PORTB outputs.

A read of PORTB returns the logic level of the actual pins.

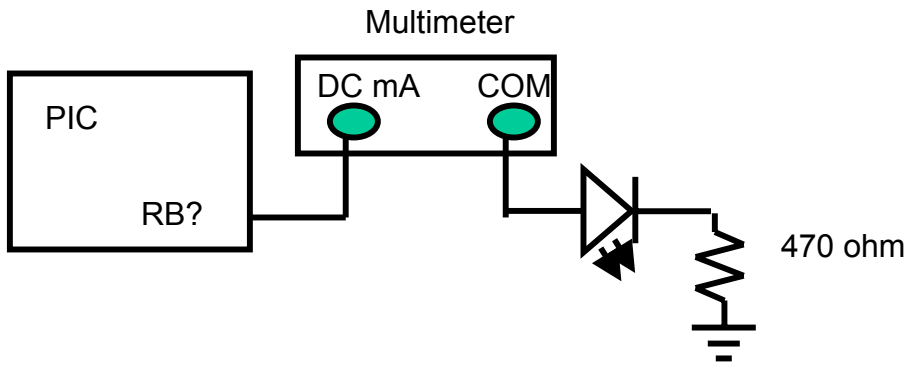
Typically, if PORTB is configured as an output, then reading LATB and PORTB returns the same value as LATB is driving the PORTB pins. However, if a PORTB pin is directly shorted to VDD or GND, then this overpowers the LATB driver, and reading PORTB can return a different value from a read of LATB.

8. Write *C* code that configures the RB1/INT1 input as an enabled, low-priority interrupt, rising edge triggered. The priority system must be enabled and the RB1/INT1 interrupt enabled. Do not assume any default bit values.

The original problems asked for RB0/INT0, which was a mistake on my part as INT0 is always a high priority interrupt. The code below configures RB1/INT1 as a low priority interrupt (problem statement corrected to specify RB1/INT1).

```
INT1IF = 0;      // clear flag for INT1
INT1IE = 1;      // enable INT1
INT1IP = 0;      // set priority of INT1 as low priority
IPEN = 1; // enable priority system
GIE = 1;  // enable all enabled interrupts
```

9. Draw a diagram that shows how to measure the current flowing through the LED connected to PORT RB7 of figure (a). Be specific.



10. Using the definition of a circular buffer in problem (f), when does OVERFLOW of the circular buffer occur?

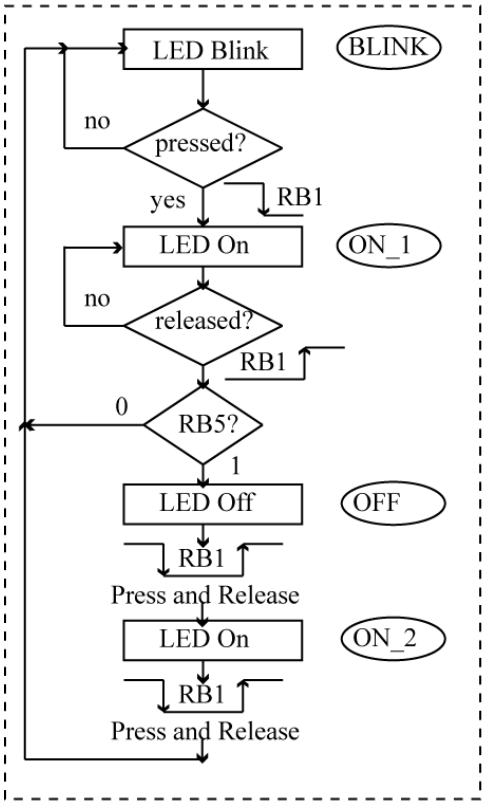
If head is equal tail after placing data into the buffer (after head is incremented and wrapped), then the buffer has overflowed.

11. What registers are AUTOMATICALLY saved when a PIC18 interrupt occurs? Give the individual names.

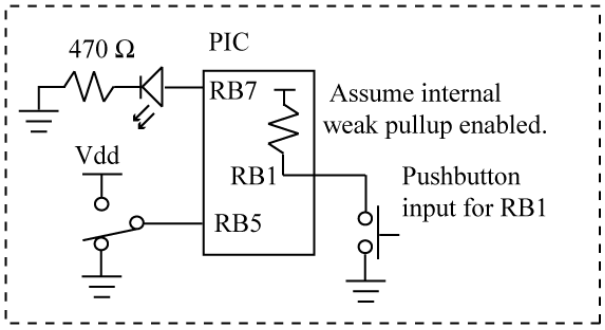
BSR, W, and STATUS are automatically saved to the shadow registers.

Figures

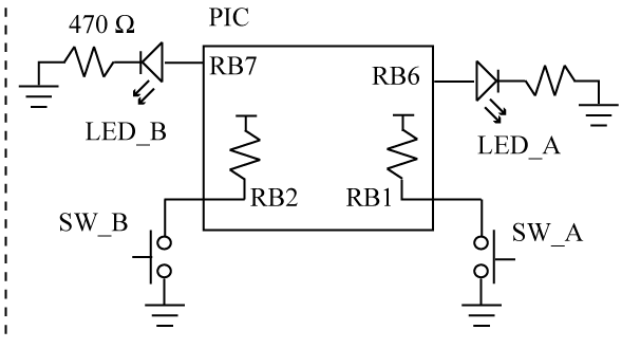
Problem (b)



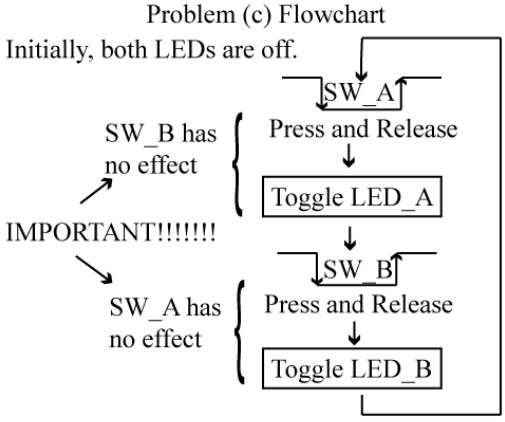
Problem (a)



Problem (c)



Problem (c) Flowchart



To toggle an LED:
If it is ON, then turn it OFF.
If it is OFF, then turn it ON.