1. Refer to the PIC24 system diagram given below.

a) What is the purpose of the MCLR# pin?

*The MCLR# pin on the PIC24 processor is used to reset that processor when the pin is asserted low.*

b) What is the purpose of the 910 Ω resistor labeled as R1?

*Acts as a current limiting resistor for LED L1 and as an external pullup resistor for pin RB15.*

c) What is the purpose of the .1 μF capacitor labeled as C2?

*Decoupling capacitor.*

d) Assuming the voltage drop across the L1 LED is 2 volts, what is the current flowing through resistor R1 when the LED is on.

\[ I = \frac{V}{R} = \frac{3.3V - 2V}{910 \, \Omega} = 1.43 \, \text{ma} \]
2. Give possible values for the WDTPRE and WDTPOST fields to enable a watchdog timer (WDT) reset of approximately 1 second for the PIC24 processor. Assume the WDT is clocked using a 32.768kHz oscillator. Use attached sheets describing the WDT operation as needed.

\[
\text{WDT timeout} = \frac{1}{32.768\text{kHz}} \times (\text{WDTPRE}) \times (\text{WDTPOST})
\]

Assume WDTPRE=0 => 1:32 division

\[
\text{WDTPOST} = \text{WDT timeout} \times 32768 / \text{WDTPRE}
\]

\[
= 1 \times 32768/32
\]

\[
= 1024
\]

Therefore, setting WDTPRE=0 and WDTPOST=1010 would suffice.

3. What is the difference between the PIC24 SLEEP and IDLE modes of operation?

SLEEP: Main clock stopped to CPU and all peripherals. Can be awoken by the WDT. Use the pwrsav #0 instruction to put the processor into SLEEP mode.

IDLE: Main clock stopped to CPU but not the peripherals (UART and other peripherals can still receive data). Can be awoken by the WDT. Use the pwrsav #1 instruction to put the processor into IDLE mode.

4. What are the differences between the CNx interrupts and the external interrupts (INT0, INT1 and INT2)?

CNx interrupts, when configured, generate an interrupt when a change (either 0->1 or 1->0) is seen on a given pin associated with a particular CNx capability.

INT0, INT1, and INT2 are input interrupt sources (INTx) that can be configured to be rising edge triggered or falling-edge triggered by using an associated INTxE bit (‘1’ is falling edge, ‘0’ is rising edge’. On the PIC24HJ32GP202, INT1 and INT2 must be brought out to remappable pins (RPx); INT0 is assigned a fixed pin location like the CNx pins.

5. What are the differences, and similarities, between LATx and PORTx?

When performing write operations to a pin on the PIC24, the LATx and PORTx function the same (i.e. they both write to the LATx register). When performing reads, a LATx read reads from the LATx register while a PORTx read is directly from the pin.

6. What is the purpose of weak internal pullup and open drain output capabilities on the PIC24 processor?

The weak internal pullup capability is provided so that an external pullup resistor is not needed on pins with CNy functionality. The
open drain output allows an external pullup resistor to be used to pull output pins up to logic ‘1’ where the output voltage may be something other than the typical +3.3V.

7. What is the purpose of the MCLR# pin on the PIC24 processor?

The MCLR# pin on the PIC24 processor is used to reset that processor when the pin is asserted low.

8. Assume the following circuit is used in a PIC24 system.

```
inline void CONFIG_SW1() {
    CONFIG_RB12_AS_DIG_INPUT();
    ENABLE_RB12_PULLUP();
}
```

a) Write an inline macro named CONFIG_SW1() that, when used, would correctly configure RB12 as a digital input as shown.
b) Write an inline macro named CONFIG_LED1() that, when used, would correctly configure RB13 as a digital output as shown.

```
#define CONFIG_LED1() CONFIG_RB13_AS_DIG_OUTPUT()
```

c) Write an inline macro named CONFIG_LED2() that, when used, would correctly configure RB14 as a digital output as shown.

```
#define CONFIG_LED2() CONFIG_RB14_AS_DIG_OUTPUT()
```

d) What is the purpose of the two resistors in series with LED1 and LED2?

They function as current limiting resistors so that the maximum current source/sink for the PIC24 processor will not be exceeded and that the current rating for the LED will not be exceeded.

e) Write a C program that would use the macros defined above and would alternate lighting LED1 and LED2 every time SW1 is pressed.

```
#define LED1 _LATB13
#define LED2 _LATB14
#define CONFIG_LED1() CONFIG_RB13_AS_DIG_OUTPUT()
#define CONFIG_LED2() CONFIG_RB14_AS_DIG_OUTPUT()

inline void CONFIG_SW1() {
    CONFIG_RB12_AS_DIG_INPUT();
    ENABLE_RB12_PULLUP();
}

#define SW1 _RB12
#define SW1_PRESSED() SW1==0
#define SW1_RELEASED() SW1==1

main() {
    CONFIG_SW1();
    CONFIG_LED1();
    CONFIG_LED2();
    LED1=0;
    LED2=0;
    while(1) {
        while(SW1_RELEASED());
        DELAY_MS(15);
        while(SW1_PRESSED());
        DELAY_MS(15);
        LED1=1;
        LED2=0;
        while(SW1_RELEASED());
        DELAY_MS(15);
        while(SW1_PRESSED());
        DELAY_MS(15);
    }
}
```
9. Write an interrupt service routine that would use the hardware in problem 8 and would sample SW1 every 20 ms.

```c
volatile uint8_t u8_valueSW1 = 1;
#define SW1_RAW _RB12       // raw switch value
#define SW1_VAL u8_valueSW1  // switch state

//Interrupt Service Routine for Timer3
void _ISRFAST _T3Interrupt (void) {
    SW1_VAL = SW1_RAW;      // sample the switch
    _T3IF = 0;               // clear the timer interrupt bit
}
#define ISR_PERIOD 20       // in ms
void configTimer3(void) {
    // ensure that Timer2,3 configured as separate timers.
    T2CONbits.T32 = 0;      // 32-bit mode off
    // T3CON set like this for documentation purposes.
    // could be replaced by T3CON = 0x0020
    T3CON = T3_OFF | T3_IDLE_CON | T3_GATE_OFF
    | T3_SOURCE_INT
    | T3_PS_1_64 ;           // results in T3CON= 0x0020
    PR3 = msToU16Ticks (ISR_PERIOD, getTimerPrescale(T3CONbits)) - 1;
    TMR3  = 0;               // clear timer3 value
    _T3IF = 0;               // clear interrupt flag
    _T3IP = 1;               // choose a priority
    _T3IE = 1;               // enable the interrupt
    T3CONbits.TON = 1;       // turn on the timer
}
```

10. Write a program that uses Timer2 to generate a square wave output on pin RB13 with a period of 10 ms.

```c
#include "pic24_all.h"
#define WAVEOUT _LATB13
inline void CONFIG_WAVEOUT() {
    CONFIG_RB13_AS_DIG_OUTPUT();   // use RB13 for output
}

//Interrupt Service Routine for Timer2
void _ISRFAST _T2Interrupt (void) {
    WAVEOUT = !WAVEOUT;            // toggle output
}
- \_T2IF = 0;                  // clear the timer interrupt bit
}

#define ISR_PERIOD 5    // in ms

void configTimer2(void) {
    // T2CON set like this for documentation purposes.
    // could be replaced by T2CON = 0x0020
    T2CON = T2_OFF | T2_IDLE_CON | T2_GATE_OFF
        | T2_32BIT_MODE_OFF
        | T2_SOURCE_INT
        | T2_PS_1_64 ;  // results in T2CON = 0x0020
    // subtract 1 from ticks value assigned to PR2 because period is
    // PRx + 1
    PR2 = msToU16Ticks (ISR_PERIOD, getTimerPrescale(T2CONbits)) - 1;
    TMR2  = 0;                  // clear timer2 value
    \_T2IF = 0;                  // clear interrupt flag
    \_T2IP = 1;                  // choose a priority
    \_T2IE = 1;                  // enable the interrupt
    T2CONbits.TON = 1;          // turn on the timer
}

int main (void) {
    configBasic(HELLO_MSG);
    CONFIG_WAVEOUT();     // PIO Config
    configTimer2();       // TMR2 config
    // interrupt does work of generating the square wave
    while (1) {
        doHeartbeat();       // ensure that we are alive
    }                      // end while (1)
}

11. An I²C device has a device specific address of 0b1011. Show a diagram similar to slide 8-4 from the lectures that would allow the device to have a peripheral address of 0b1011110 in a PIC24 system. Show all connections between the I²C device and the PIC24 processor.
12. Show the timing diagram corresponding to the PIC24 performing a write1I2C1() operation to the device in problem 11. The data byte to write has the value 0xAA.

```
S 1011110 0 A 10101010 A P
```

13. What is the C keyword `VOLATILE` used for in the PIC24 code discussed in class?

The `VOLATILE` keyword is used to indicate to the compiler that no optimizations should be performed on the defined variable.

14. Write a complete C main() function and corresponding ISR that would detect a falling edge transition on pin RB13. The falling edge transition should cause an INT1 interrupt. The ISR should set a semaphore (named `PIN_CHANGE`) when the transition occurs. The main() routine should monitor the semaphore and toggle an LED attached to RB15 whenever the falling edge occurs. The main() routine is also responsible for performing all necessary PIC24 configuration and for clearing the semaphore when the LED is toggled. The ISR should only set the semaphore if it has been previously cleared by the main() routine.

```c
#include "pic24_all.h"

// semaphore variable
volatile uint8 PIN_CHANGE = 0;    //initially cleared

// Interrupt Service Routine for INT1
void _ISRFAST _INT1Interrupt (void) {
    if(!PIN_CHANGE)
        PIN_CHANGE=1;
    _INT1IF = 0;    //clear the interrupt bit
}

// LED
#define CONFIG_LED() CONFIG_RB15_AS_DIG_OUTPUT()
#define LED _LATB15     // led state

// Input switch configuration, use RB13
inline void CONFIG_SW() {
    CONFIG_RB13_AS_DIG_INPUT();   // use RB13 for switch input
    ENABLE_RB13_PULLUP();         // enable the pullup
    DELAY_US(1);                  // Wait for pull-up
}

int main (void) {
    CONFIG_SW();
```
CONFIG_LED();
CONFIG_INT1_TO_RP(13);  // RP13 shares RB13 pin
/** Configure INT1 interrupt */
_INT1IF = 0;    // Clear the interrupt flag
_INT1IP = 2;    // Choose a priority
_INT1EP = 1;    // negative edge triggered
_INT1IE = 1;    // enable INT1 interrupt
while(1) {
    if(PIN_CHANGE) {
        LED=!LED;
        PIN_CHANGE=0;
    }
}

15. Assume a PIC24 system has a 10-bit successive approximation analog-to-digital converter (ADC) with Vref=3.3 volts.

   a. How many clock cycles does the PIC24 take to convert an analog voltage to a digital value?

      10 clock cycles

   b. Assuming Vin=2.50 volts, show the operation of the ADC for each bit of the digital value produced. Show your work similar to that given in lecture slide 7-10.

      i. Guess DAC input=0b1000000000=512, so Vdac=512/2^10*3.3V=1.65V. Vdac(1.65V) < Vin (2.5V), so guess of ‘1’ for bit of DAC was correct.

      ii. Guess DAC input=0b1100000000=768, so Vdac=768/2^10*3.3V=2.475V. Vdac(2.475V) < Vin (2.5V), so guess of ‘1’ for bit of DAC was correct.

      iii. Guess DAC input=0b1110000000=896, so Vdac=896/2^10*3.3V=2.888V. Vdac(2.888V) > Vin (2.5V), so guess of ‘1’ for bit of DAC was incorrect. Reset this bit to ‘0’.

      iv. Guess DAC input=0b1101000000=832, so Vdac=832/2^10*3.3V=2.681V. Vdac(2.681V) > Vin (2.5V), so guess of ‘1’ for bit of DAC was incorrect. Reset this bit to ‘0’.

      v. Guess DAC input=0b1100100000=800, so Vdac=800/2^10*3.3V=2.578V. Vdac(2.578V) > Vin (2.5V), so guess of ‘1’ for bit of DAC was incorrect. Reset this bit to ‘0’.

      vi. Guess DAC input=0b1100010000=784, so Vdac=784/2^10*3.3V=2.527V. Vdac(2.527V) > Vin (2.5V), so guess of ‘1’ for bit of DAC was incorrect. Reset this bit to ‘0’.
vii. Guess DAC input=0b1100001000=776, so Vdac=776/2^{10} \times 3.3\text{V}=2.501\text{V}. Vdac(2.501\text{V}) > Vin (2.5\text{V}), so guess of ‘1’ for bit of DAC was incorrect. Reset this bit to ‘0’.

viii. Guess DAC input=0b1100000100=772, so Vdac=772/2^{10} \times 3.3\text{V}=2.489\text{V}. Vdac(2.489\text{V}) < Vin (2.5\text{V}), so guess of ‘1’ for bit of DAC was correct.

ix. Guess DAC input=0b1100000110=774, so Vdac=774/2^{10} \times 3.3\text{V}=2.494\text{V}. Vdac(2.494\text{V}) < Vin (2.5\text{V}), so guess of ‘1’ for bit of DAC was correct.

x. Guess DAC input=0b1100000111=775, so Vdac=775/2^{10} \times 3.3\text{V}=2.498\text{V}. Vdac(2.498\text{V}) < Vin (2.5\text{V}), so guess of ‘1’ for bit of DAC was correct. Final ADC output code=0b1100000111