Midterm Solutions

1. For each of the following C language expressions, assumed to be issued one after another, provide the values, **in BOTH hexadecimal AND binary**, of the variable x (which is an unsigned char).

		HEXADECIMAL	BINARY
a)	x = 72; x &= ~255	0x_0______	0b_0_______________
b)	x = 37; x |= -1;	0x_FF____	0b_11111111________
c)	x = 9 >> 3;	0x_1______	0b__1_____________
d)	x = 57/6;	0x_09____	0b__00001001_____

2. Below is the assembly code produced by a compiler from a short C program for an MSP430 Launchpad. Examine the assembly code and 1) add comments to each line of code to explain the function; 2) Explain in a sentence or two what the program does (hint: 23168 = WDTPW + WDTHOLD).

```
main:
        mov     #23168, &__WDTCTL  ; disable watchdog timer
        mov.b   #1, &__P1DIR       ; set P1.0 as output, P1.1-1.7 as inputs
.L9:
        mov.b   &__P1IN, r15       ;   read P1IN, store in register R15
        and     #8, r15            ; test bit 3, check if P1.3 was high
        jne     .L9                ; if P1.3 was high, loop to .L9
.L6:
        mov.b   &__P1IN, r15       ; read P1IN, store in register R15
        and     #8, r15            ; test bit 3, check if P1.3 was high
        jeq     .L6                ; if P1.3 was low, loop to .L6
        xor.b   #1, &__P1OUT       ; toggle P1.0 output
        jmp     .L9                ; loop back to .L9
```

The program disables the watchdog timer, configures P1.0 as an output, then detects low-> high transitions on P1.3. On each low->high transition, the P1.0 output is toggled.

“Usefulness” of comments:

set WDTCTL to 23168 vs disable watchdog timer

logical and of 8 and r15 vs test if bit 3 of P1IN (P1.3) was high

These are self-evident, not worth writing

These help you understand what the code is doing.

```
mov.b &__P1IN,r15
 and     #8, r15
 jne     .L9
```

The and will set the Z and C bits in the status register:
if the result of the and is zero, Z is set and C is reset.
Otherwise Z is reset, C is set.

jne == jnz jumps if the zero bit is reset – here it jumps if the result of the and was not zero, so it jumps if P1.3 was high.

This was the code I got by compiling “Activity 4:”

```
#include <msp430.h>
int main(void){

    WDTCTL = WDTPW + WDTHOLD;
    P1DIR = 1; // P1.0 output, all others input

    while(1){
        while (P1IN & 8);
        while (! (P1IN & 8));
        P1OUT ^= 1;

    }
}
```

This was the code I got by compiling “Activity 4”

```
#include <msp430.h>
int main(void){

    WDTCTL = WDTPW + WDTHOLD;
    P1DIR = 1; // P1.0 output, all others input

    while(1){
        while (P1IN & 8);
        while (! (P1IN & 8));
        P1OUT ^= 1;

    }
}
```

loops here while P1.3 is high

loops here while P1.3 is low

proceeds from here on low->high transition

3.Write a **complete**, well annotated program in C, that will operate the MSP430G2553 as a stand-alone parallel-output ADC. Pin P1.1 is a trigger input: a high to low transition detected on this pin should cause the program to read the analog voltage value on P1.0, and then place the digital result on pins P1.2-1.7 and P2.0-P2.3. It uses P2.4 as a 'data valid' flag. P2.4 is low when the values on the output pins have been set.
The program should begin by configuring necessary registers (eg P1DIR, ADC10CTL0 etc), then it should set P2.4 to High to indicate that the output pins do not have any meaning, and put the CPU to sleep until awoken by a high to low transition on pin P1.1 triggering an interrupt.
The interrupt handler should:
a) set P2.4 High
b) use the ADC10 to read the analog voltage applied to pin P1.0,
c) place the 4 most significant bits of the ADC result on P2.0-P2.3 (P2.3 is most significant), and the lower 6 bits on pins P1.2-P1.7.
d) set P2.4 Low to indicate that the output values have been set.
e) exit until the next interrupt call the handler again.

The C programs that we examined in the first four weeks of lab sessions are in the reference package. Feel free to write on the back if necessary.

The program should begin by configuring necessary registers (eg P1DIR, ADC10CTL0 etc), then it should set P2.4 to High to indicate that the output pins do not have any meaning, and put the CPU to sleep until awoken by a high to low transition on pin P1.1 triggering an interrupt.

```
#include <msp430.h>
void main(void){
    WDTCTL=WDTPW+WDTHOLD; // disable watchdog timer
    P1DIR = 0xFC;                    // P1.0 and 1.1 inputs, rest outputs
    P2DIR= 0x1F;                     // P2.0-2.4 outputs
    ADC10CTL0 = ADC10SHT_2 + ADC10ON; // turn on ADC, set sample/hold time
    ADC10CTL1 = INCH_0;              // set ADC MUX so ADC samples from A0 = P1.0
    ADCAE0 |= 0x01;                  // enable A0 input
    P1IE=0x02;                       // enable interrupt on P1.1 H->L transition
    P2OUT |= 0x10;                   // set P2.4 high to indicate invalid data
    _BIS_SR(LPM4_bits +GIE);         // put cpu to sleep with interrupts enabled
}
```

The interrupt handler should:
a) set P2.4 High
b) use the ADC10 to read the analog voltage applied to pin P1.0,
c) place the 4 most significant bits of the ADC result on P2.0-P2.3 (P2.3 is most significant), and the lower 6 bits on pins P1.2-P1.7.
d) set P2.4 Low to indicate that the output values have been set.
e) exit until the next interrupt call the handler again.

```
#pragma vector = PORT1_VECTOR // port 1 interrupts come here
__interrupt void isr(void){
    P2OUT |= 0x10;          // set P2.4 high to indicate invalid data on outputs
    P1IFG &= ~2;                              // reset interrupt flag
    ADC10CTL0 |=ENC+ADC10SC;      // trigger ADC to make measurement
    while (ADC10CTL1 & ADC10BUSY); // loop till ADC is finished
    P1OUT = ADC10MEM<<2;              // write low 6 bits of result to P1.2-P1.7
    P2OUT = (ADC10MEM >> 6 | 0x10); // write high 4 bits of result to P2.0-P2.3
                                     // keep P2.4 high for now
    P2OUT &=~0x10;                    // reset P2.4 to indicate valid data.
}
```