Today:

- Announcement –lab notes marking
- Activity 2
- Programing in C continuation – interrupts
- ADC introduction if we have time

Announcement:

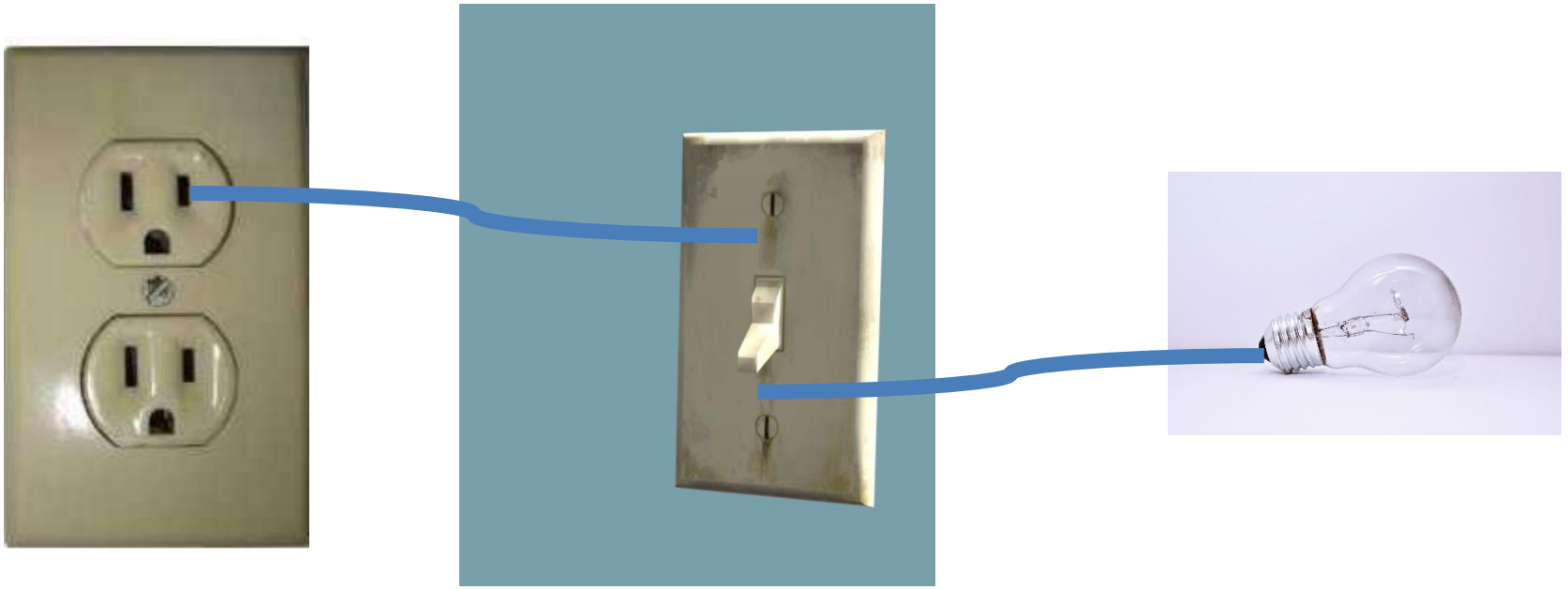
Please submit on Canvas lab notes and annotated programs as a pdf file before the beginning of lab 3.

Lab Notes Marking scheme:

Completeness and quality of procedure/circuits/code etc. used to perform all the completed activities. The full thread of communication with AI must be included	5
General Structure: objectives defined, clear enough statements of what is being done and why, what the results were.	
General clarity of the notes	2
Above and beyond. Evidence of exploration above and beyond the specific tasks requested in the manual.	1

Labs 1&2 will be worth 4 points, labs 3&4 and 5&6 8 points each pair.

Ground



What will happen when the switch is on?

C compared to assembler

- Why do we program in C?
- Why you might to have to use assembler in the future

Assembler program compiled from C

```
.include "msp430g2553.inc"
MOV.W   #0x5a80,&Watchdog_Timer_WDTCTL
MOV.B   #0x0041,&Port_1_2_P1DIR
MOV.B   #0x0040,&Port_1_2_P1OUT
PUSH    R13
PUSH    R14
MOV.W   #0x2844,R13
MOV.W   #1,R14
DEC.W   R13
SBC.W   R14
JNE     ($1_$2)
TST.W   R13
JNE     ($1_$2)
POP.W   R14
POP.W   R13
JMP     (0xc02c)
NOP
XOR.B   #0x0041,&Port_1_2_P1OUT
JMP     ($C$L1)
MOV.W   #0x0400,SP
CALL    #_system_pre_init
CLR.W   R12
CALL    #main
MOV.W   #1,R12
CALL    #C$$EXIT
 BIS.W   #0x0010,SR
JMP __TI_ISR_TRAP
JMP     (__TI_ISR_TRAP)
NOP
NOP
NOP
JMP     ($C$L1)
NOP
MOV.W   #1,R12
RET
AND.B   @R15+,0xffff(R15)
AND.B   @R15+,0xffff(R15)
```

The same program just written in Assembler

```
.include "msp430g2553.inc"
start:
        mov.w #WDTPW|WDTHOLD, &WDTCTL
        mov.b #0x41, &P1DIR
        mov.w #0x01, r8
repeat:
        mov.b r8, &P1OUT
        xor.b #0x41, r8
        mov.w #60000, r9
waiter:
        dec r9
        jnz waiter
        jmp repeat
```

Data types:

char, unsigned char – 8 bit integer (0 to 255 or -128 to 127) It can also be interpreted as character depending on the context.

int, unsigned int – usually an integer of the native word size: 16 bits (-32768 to 32767 or 0 to 65536)

long, unsigned long – 32 bit integer (~ -2×10^9 to ~2×10^9 or 0 to ~4×10^9)

long long , unsigned long long – 64 bit integer (~ -9×10^{18} to ~ 9×10^{18} or 0 to ~2×10^{19})

float – floating point number (32 bits) (floating point operations are very expensive on a processor like the msp430 that lacks a dedicated fpu - avoid if possible).

Example program Blink: while

```
#include <msp430.h>
void main(void) {
WDTCTL = WDTPW + WDTHOLD; //Stop WDT
P1DIR =0b00000001; //Set P1.0 to output;
P4DIR =0b10000000; //Set P4.7 to output;
P1OUT = 0b00000000; //Set the output Pin P1.0 to low
P4OUT = 0b10000000; //Set the output Pin P4.7 to high
while (1) { // Loop forever
_delay_cycles (500000); //This function introduces 0.5 s
delay
P1OUT = P1OUT ^ 0b00000001; //bitwise xor the output with
00000001   // can be also P1OUT^= 0b00000001;
P4OUT = P4OUT ^ 0b10000000; //bitwise xor the output with
10000000
}
}
```

Example program Blink: intrinsic

```
#include <msp430.h>
void main(void) {
WDTCTL = WDTPW + WDTHOLD; //Stop WDT
P1DIR =0b00000001; //Set P1.0 to output;
P4DIR =0b10000000; //Set P4.7 to output;
P1OUT = 0b00000000; //Set the output Pin P1.0 to low
P4OUT = 0b10000000; //Set the output Pin P4.7 to high
while (1) { // Loop forever
__delay_cycles (500000); //This function introduces 0.5 s delay
P1OUT = P1OUT ^ 0b00000001; //bitwise xor the output with 00000001
P4OUT = P4OUT ^ 0b10000000; //bitwise xor the output with 10000000
}
}
```

Setting the bits (changing from low to high)

We want to set bit 3 in a P2REN register which contains a number 0b11000000.

If we do:

P2REN = 0b00001000;

We clear the other 2 bits

Instead we do:

P2REN |= 0b00001000;

Which is the same as:

P2REN = P2REN | 0b00001000;

11000000 |00001000=11001000

Clearing the bits (changing from high to low)

P1DIR = 0b00000010; //Set pin P1.1 to output

P1OUT = 0b00000011; // set Pin P1.0 to high and P1.1 to pullup
Now I want to set P1.0 to low

If I do

P1OUT = 0b00000000; // I also cancelled the pullup!

Instead we do:

P1OUT &= ~0b00000001;

Which is the same as

P1OUT &= P1OUT &(~0b00000001);

0b00000011&11111110=000000010

Interrupts

When interrupt occurs the current microprocessor’s activity stops and the interrupt service routine (ISR) is started. It is like a function with a microcontroller specific format.
The event setting an interrupt is in fact setting a bit in a specific register. This bit is called an interrupt flag The ISR must clear this flag (some commands like the ones accessing an output communication buffer clear the specific flags automatically)!

Program with interrupt: enable

```
#include <msp430.h>
void main(void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
P1DIR = 0b00000001; //set P1.0 pin for output rest including P1.1 are inputs
P4DIR = 0b10000000; //set P4.7 pin for output
P1OUT = 0b00000011; // set Pin P1.0 to high and P1.1 to pullup
P4OUT = 0b10000000; // set Pin P4.7 to high
P1REN = 0b00000010; //enable pull up/down resistor on P1.1
P1IE = 0b00000010; //Enable input at P1.1 as an interrupt
P1IES= 0b00000010; //Interrupt occurs when input voltage goes from High to Low
_BIS_SR (LPM4_bits + GIE); //Turn on interrupts and go into the lowest
//power mode (the program stops here)
//Notice the strange format of the function, it is an "intrinsic"
//ie. not part of C; it is specific to this microprocessor
}
//Port 1 interrupt service routine starts below
void __attribute__ ((interrupt(PORT1_VECTOR))) PORT1_ISR(void) {
//code of the interrupt routine goes here
P1OUT ^= 0b00000001;
P4OUT ^= 0b10000000; // toggle the LEDs
P1IFG &= ~0b00000010; // Clear P1.3 IFG. If you don't, it just happens again.
}
```

Program with interrupt: general enable

```
#include <msp430.h>
void main(void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
P1DIR = 0b00000001; //set P1.0 pin for output rest including P1.1 are outputs
P4DIR = 0b10000000; //set P4.7 pin for output
P1OUT = 0b00000011; // set Pin P1.0 to high and P1.1 to pullup
P4OUT = 0b10000000; // set Pin P4.7 to high
P1REN = 0b00000010; //enable pull up/down resistor on P1.1
P1IE = 0b00000010; //Enable input at P1.1 as an interrupt
P1IES= 0b00000010; //Interrupt occures when input voltage goes from High to Low
_BIS_SR (LPM4_bits + GIE); //Turn on interrupts and go into the lowest
      //power mode (the program stops here)
      //Notice the strange format of the function, it is an "intrinsic"
       //ie. not part of C; it is specific to this microprocessor
}
//Port 1 interrupt service routine starts below
void __attribute__ ((interrupt(PORT1_VECTOR))) PORT1_ISR(void) {
//code of the interrupt routine goes here
P1OUT ^= 0b00000001;
P4OUT ^= 0b10000000; // toggle the LEDS is
P1IFG &= ~0b00000010; // Clear P1.3 IFG. If you don't, it just happens again.
}
```

Program with interrupt: interrupt routine

```
#include <msp430.h>
void main(void)
{
WDTCTL = WDTPW + WDTHOLD; // Stop watchdog timer
P1DIR = 0b00000001; //set P1.0 pin for output rest including P1.1 are outputs
P4DIR = 0b10000000; //set P4.7 pin for output
P1OUT = 0b00000011; // set Pin P1.0 to high and P1.1 to pullup
P4OUT = 0b10000000; // set Pin P4.7 to high
P1REN = 0b00000010; //enable pull up/down resistor on P1.1
P1IE = 0b00000010; //Enable input at P1.1 as an interrupt
P1IES= 0b00000010; //Interrupt occures when input voltage goes from High to Low
_BIS_SR (LPM4_bits + GIE); //Turn on interrupts and go into the lowest
      //power mode (the program stops here)
      //Notice the strange format of the function, it is an "intrinsic"
       //ie. not part of C; it is specific to this microprocessor
}
//Port 1 interrupt service routine starts below
void __attribute__ ((interrupt(PORT1_VECTOR))) PORT1_ISR(void) {
//code of the interrupt routine goes here
P1OUT ^= 0b00000001;
P4OUT ^= 0b10000000; // toggle the LEDS is
P1IFG &= ~0b00000010; // Clear P1.3 IFG. If you don't, it just happens again.
}
```

Program with interrupt: clear the flag

```
#include <msp430.h>
void main(void)
{
WDTCTL = WDTPW + WDTHOLD;       // Stop watchdog timer
P1DIR = 0b11110111;             //set all P1 pins for output except P1.3
P1OUT = 0b01001001;              // set Pins P1.0 and P1.6 to high and P1.3 to pullup
P1REN = 0b00001000;              //enable pull up/down resistor on P1.3
P1IE =     0b00001000;           //Enable input at P1.3 as an interrupt
_BIS_SR (LPM4_bits + GIE);       //Turn on interrupts and go into the lowest
                                 //power mode (the program stops here)
                                  //Notice the strange format of the function, it is an "intrinsic
                                 //ie. not part of C; it is specific to this chipset
}
                                 // Port 1 interrupt service routine
void __attribute__ ((interrupt(PORT1_VECTOR))) PORT1_ISR(void)
   {                             //code of the interrupt routine goes here
      P1OUT ^=0b01000001;         // toggle the LEDS
P1IFG &= ~0b00001000;          // Clear P1.3 IFG. If you don't, it just happens again.
   }
```

Programming in C

Libraries:
there are some “standard” libraries available that extend the operations you can easily use.

eg:
the math library gives access to functions like:
sin(x), cos(x), tan(x), sqrt(x), ln(x), log(x) etc...

To use math functions, you need to:
`#include <math.h>` at the top of the file

Other libraries provide routines for string manipulations and many other things...

These libraries tend to take up a substantial amount of flash memory and consume (precious) ram. You should try to avoid these on the MSP430 if at all possible!

Programming in C: functions

```
int multiply_together(int x, int y)
{
        return x*y;
}
```

You can define other functions that can take arguments and return values.

```
...

y = multiply_together(4,8);
...
```

The function definition either needs to come in the file before you call it, or you need to supply a *function prototype* before you call it.

A prototype for this function would simply be:

```
int multiply_together(int x, int y);
```

Indentation.

Please use proper indentation of your C code to make it readable!

There are tools that can help. Many text editors can help you indent properly.

See

http://www.cprogramming.com/tutorial/style.html

For more details than you care about, see:
http://en.wikipedia.org/wiki/Indent_style