PHYS 319
Labs 3 and 4 Notes

Jonathan Chan (15354146)

January 30, 2018

To compile the C programs to .asm and .elf, I've modified my Makefile as below.

```
SOURCES = $(wildcard *.c)
EXEC    = $(patsubst %.c, %.elf, $(SOURCES))
DEVICE  = msp430g2553
INSTALL_DIR=$(HOME)/ti/msp430_gcc

GCC_DIR =  $(INSTALL_DIR)/bin
SUPPORT_FILE_DIRECTORY = $(INSTALL_DIR)/include

CC      = $(GCC_DIR)/msp430-elf-gcc
GDB     = $(GCC_DIR)/msp430-elf-gdb

#O0 works, O1 works, O2 doesn't -Os works
CFLAGS = -I $(SUPPORT_FILE_DIRECTORY) -mmcu=$(DEVICE) -Os -g
LFLAGS = -L $(SUPPORT_FILE_DIRECTORY) -T $(DEVICE).ld

all: prog1 prog2 adc pwm

prog1: prog1.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o prog1.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o prog1.asm
prog2: prog2.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o prog2.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o prog2.asm
adc:   adc.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o adc.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o adc.asm
pwm:   pwm.c
    $(CC) $(CFLAGS) $(LFLAGS) $? -o pwm.elf
    $(CC) $(CFLAGS) $(LFLAGS) $? -S -o pwm.asm

debug: all
    $(GDB) ${EXEC}

clean:
    rm prog1.elf prog2.elf adc.elf pwm.elf prog1.asm prog2.asm adc.asm pwm.asm
```

1 Program 1

Compiling from C, the produced Assembly file has a *lot* of extra code, and appears to have been optimized differently. Below is the relevant section of the compiled .asm with some comments comparing lines to last lab's prog1.asm code.

```
.LCFI0:
        .loc 1 21 0
        MOV.W   #23168, &WDTCTL         ; turn off watchdog
        .loc 1 22 0
        MOV.B   #65, &P1DIR             ; set output direction (P1.6 and P1.0 for LEDs)
        .loc 1 23 0
        MOV.B   #1, &P1OUT              ; set initial state (LED1 on)
.LBB2:
        .loc 1 27 0
        MOV.W   #-5536, R12             ; amount to decrement by (60000 shown as signed word)
.L6:
.LVL0:
        MOV.W   R12, @R1                ; use R1 as working register to decrement (first loop)
.L3:
        .loc 1 28 0
        MOV.W   @R1, R13                ; use R13 as working register for this loop
        CMP.W   #0, R13 { JEQ    .L2    ; go to next countdown if this one has reached zero
        .loc 1 29 0
        ADD.W   #-1, @R1                ; decrement counter
        BR  #.L3                        ; loop
.L2:
.LVL1:
        .loc 1 27 0
        MOV.W   R12, @R1                ; reset R1 as working register to decrement (second loop)
.L5:
        .loc 1 28 0
        MOV.W   @R1, R13                ; use R13 as working register for this loop
        CMP.W   #0, R13 { JEQ    .L4    ; break if countdown has reached zero
        .loc 1 29 0
        ADD.W   #-1, @R1                ; decrement counter
        BR  #.L5                        ; loop
.L4:
.LVL2:
.LBE2:
        .loc 1 32 0 discriminator 1
        XOR.B   #65, &P1OUT             ; switch LEDs
        .loc 1 26 0 discriminator 1
        BR  #.L6                        ; loop from top of loops
```

This was generated using the following C code that doubles the count and thus halves the blinking rate.

```c
#include <msp430.h>

void main(void) {
    volatile unsigned int count;
    WDTCTL = WDTPW + WDTHOLD;       // Stop WDT
    P1DIR = 0x41;                   // Set P1 output direction
    P1OUT = 0x01;                   // Set the output

    while (1) {                     // Loop forever
        for (volatile unsigned char i = 0; i < 2; i++) {    // decrement by 60000 twice
            count = 60000;
            while (count != 0) {
                count--;            // decrement
            }
        }
        P1OUT = P1OUT ^ 0x41;       // bitwise xor the output with 0x41
    }
}
```

2 Program 2

Below is the C code used to make the LEDs blink in the red–green–both–none pattern, using the same XOR trick described in the last lab notes.

```c
#include <msp430.h>

volatile unsigned char stateChanger;

void main(void) {
    WDTCTL = WDTPW + WDTHOLD;   // Stop watchdog timer
    P1DIR  = 0xF7;              // C does not have a convenient way of
                                // representing numbers in binary; use hex instead
    P1OUT  = 0x08;              // LEDs off
    P1REN  = 0x08;              // enable resistor
    P1IE   = 0x08;              // Enable input at P1.3 as an interrupt
    stateChanger = 0x1;         // 0x01 to toggle LED0; 0x40 to toggle LED1

    _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) {
    P1OUT ^= stateChanger;      // toggle the LEDS
    stateChanger ^= 0x40;       // 0x01 -> 0x41 -> 0x01
    P1IFG &= ~0x08;             // Clear P1.3 IFG. If you don't, it just happens again.
}
```