PHYS 319
Labs 1 and 2 Notes

Jonathan Chan (15354146)

January 16, 2018

1 Lab 1

2 Lab 2

Some minor reminders:

- Remember to connect +5V and ground to 4-digit 7-segment display, and ground (**not** VCC) to microprocessor
- `mspdebug` needs to be exited (with CTRL-D) for the program to run

2.1 Student Number

There needs to be a move to `P1OUT` for setting each digit. Since the strobe also needs to go from low to high to actually set the digit, there are actually two moves for each digit. Below is the full program for setting the display to `4146`.

```
.include "msp430g2553.inc"

    org 0xc000
START:
    ; setup
    mov     #0x0400,        SP
    mov.w   #WDTPW|WDTHOLD, &WDTCTL
    mov.b   #11110111b,     &P1DIR

    ; set digits
    mov.b   #01100000b,     &P1OUT  ; xxx6
    mov.b   #01100001b,     &P1OUT  ; xxx6

    mov.b   #01000010b,     &P1OUT  ; xx46
    mov.b   #01000011b,     &P1OUT  ; xx46

    mov.b   #00010100b,     &P1OUT  ; x146
    mov.b   #00010101b,     &P1OUT  ; x146
```

```
    mov.b   #01000110b,     &P1OUT  ; 4146
    mov.b   #01000111b,     &P1OUT  ; 4146

    ; disable
    bis.w   #CPUOFF,        SR

    org 0xfffe
    dw      START
```

2.2 Program 1

Below is the full program annotated with comments. Making the lights blink twice as fast is simply halving the initial value set in R9, but making them blink twice as slow involves decrementing another register, since the doubled value is 80000 and will not fit in a two-byte word whose maximum value is 65536.

```
.include "msp430g2553.inc"

    org 0xC000
START:
    mov.w   #WDTPW|WDTHOLD, &WDTCTL
    mov.b   #0x41,          &P1DIR  ; #01000001b (P1.6 == LED2, P1.0 == LED1)
    mov.w   #0x01,          R8      ; #00000001b (start on LED1)
REPEAT:
    mov.b   R8,             &P1OUT
    xor.b   #0x41,          R8      ; #00000001b -> #01000000b -> ... (LED0 -> LED1 -> ...)
    mov.w   #40000,         R9      ; counts to decrement before blink
    mov.w   #40000,         R10     ; counts to decrement (2nd dec, since max val is 65536)
WAITER1:
    dec     R9
    jnz     WAITER1         ; R9 not yet 0
WAITER2:
    dec     R10
    jnz     waiter2         ; R10 not yet 0
    jmp     repeat          ; R9, R10 == 0; blink other LED

    org 0xfffe
    dw      START           ; set reset vector to 'init' label
```

2.3 Program 2

To make the LEDs cycle in the order
`none -> red -> green -> both -> none`,
the output to P1OUT needs to be
`0000 0000 -> 0000 0001 -> 0100 0000 -> 0100 0001 -> 0000 0000`.
Notice that

`0000 0000 -> 0000 0001` and `0100 0000 -> 0100 0001`
can be done with an `xor` on `0000 0001`, and
`0000 0001 -> 0100 0000` and `0100 0001 -> 0000 0000`
can be done with an `xor` on `0100 0001`. Rather than using two registers to save the two different values to `xor` on, notice that in turn
`0000 0001 -> 0100 0001 -> 0000 0001`
can be done with an `xor` on `0100 0000`. Then we initialize a register (`R8` here) to `0000 0001`, and after we have `xor`ed it with the output, we `xor 0100 0000` on `R8` to get the next value that should be `xor`ed with the output. Below is the full program annotated with comments.

```
#include "msp430g2553.inc"

org 0x0C000
RESET:
    mov.w   #0x400,          SP
    mov.w   #WDTPW|WDTHOLD, &WDTCTL
    mov.b   #11110111b,      &P1DIR      ; all pins outputs except P1.3
    mov.b   #00001000b,      &P1REN      ; enable resistor for P1.3
    mov.b   #00001000b,      &P1IE       ; P1.3 set as an interrupt
    mov.w   #0x0049,         R7          ; R7 = 0000 0000 0100 1001
    mov.b   R7,              &P1OUT      ; LED1, LED2 on
    mov.b   #0x0041,        R8           ; value to xor with R7
    EINT                                 ; enable interrupts
    bis.w   #CPUOFF,         SR
PUSH:
    xor.w   R8,              R7          ; next LED state
    xor.w   #0x0040,         R8          ; 0x0041 -> 0x0001 -> 0x0041
    mov.b   R7,              &P1OUT      ; set LEDs to new state
    bic.b   #00001000b,      &P1IFG      ; interrupt flag P1.3 set to 0
    reti                                 ; return from interrupt

    org 0xffe4
    dw PUSH                              ; interrupt from button goes here

    org 0xfffe
    dw RESET                             ; interrupt from reset button goes here
```