PHYS 319
Labs 1 and 2 Notes

Jonathan Chan (15354146)

January 18, 2018

1 Lab 1

The goal of this lab is to display the last four digits of my student number (4146) on the 4-digit 7-segment display.

The breadboard's wiring layout resembles this (there are two):

```
------------------  -------------------
------------------  -------------------
| | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | |
------------------  -------------------
------------------  -------------------
```

In $V_{OH}, V_{IH}, V_{OL}, V_{IL}$,

- The O/I means it's the voltage output/input
- The H/L means it's a voltage HI/LO (or 1/0)

Max and min are the maximum and minimum acceptable voltage for that input/output for HI/LO.

For example, a gate's acceptable voltages may look like the following:

```
+5V
|||         max
|||     V_IH
|||         min
 |
 |
 |
|||         max
|||     V_IL
|||         min
=0V
```

The 4-digit 7-segment multiplexed display has seven inputs:

- D3 D2 D1 D0: the input for a single digit, from `0x0` to `0xF`
- A1 A0: the input for selecting a digit, where `0b11` is leftmost and `0b00` is rightmost
- STR: when this voltage goes from LO to HI, the value given by Dx is loaded into the digit selected by Ax

Normally, since a single 7-segment display requires four inputs to display the $2^4 = 16$ different hex digits, 16 inputs would be required to display four digits. However, by using two inputs to select one of the $2^2 = 4$ digits to change and one input to indicate when the digit should be updated, we reduce the total number of inputs to just seven. The width of the latch switch pulse is 30 ns and there is a propagation delay from input to output of 50 ns.

Below is a rough circuit diagramme for wiring up the switches to the Dx inputs, the button to the strobe, and Ax:

```
        Ohmm...
+5V ____VVV_____________________________________________________________
            |      __       |      __       |      __       |      __       |
            |_____|  |      |_____|  |      |_____|  |      |_____|  |      |       set manually
            ______|SW|      ______|SW|      ______|SW|      ______|SW|      |__     w/ 5V|GND to
            | GRD_|__|      | GRD_|__|      | GRD_|__|      | GRD_|__|      | O|    select digit
            |               |               |               |               |__|    |   |
            |               |               |               |               |       |   |
            D3              D2              D1              D0              STR     A1  A0

         _____________________________________________________
        |   __             __             __             __   |
        |  |__|           |__|           |__|           |__|  |
        |  |__|           |__|           |__|           |__|  |
        |_____________________________________________________|

            11              10              01              00                  == A1  A0
```

`SW` indicates a switch, where the up position corresponds to HI and the down position to LO. The `O` encased in a rectangle connected to the strobe input is the button used to strobe from low to high to allow the display to accept the current inputs. To set the second-left digit to 6, for example, the switches must be in positions [down up up down] and A1 must be connected to power while A0 remains connected to ground. Then press the strobe button and the digit will appear.

2 Lab 2

Some minor reminders:

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

2.1 Student Number

The goal of this activity is to display the last four digits of my student number (4146) using the microprocessor. The pins P1.7 to P1.0 excluding P1.3 have been connected to D3 through D0, then A1 and A0, then STR. Therefore, 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 to alternate the strobe. 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

The goal of this activity is to understand the given program in assembly and to modify the blinking speed. The key features to note is that `xor 0100 0001` is used to alternate the lights from `0100 0000 -> 0000 0001`, and that the pauses between blinks is achieved by decrementing a register set to some value and waiting until that value becomes zero. Below is the full program for half-speed blinking 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 -> ... (LED1 -> LED2 -> ...)
    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           ; reset interrupt goes to START
```

2.3 Program 2

The goal of this activity is to understand the given program in assembly and to modify the behaviour from turning the LEDs on and off to turning alternating LEDs on, then both, then off. To make the LEDs cycle in the order

`none -> red -> green -> both -> none`,

the output to P1OUT needs to cycle through

`0000 0000 -> 0000 0001 -> 0100 0000 -> 0100 0001 -> 0000 0000`.

Notice that the first and third transitions

`0000 0000 -> 0000 0001` *and* `0100 0000 -> 0100 0001`

can be done by applying `xor 0000 0001`, while the second and fourth transitions

`0000 0001 -> 0100 0000` *and* `0100 0001 -> 0000 0000`

can be done by applying `xor 0100 0001`. Rather than using two registers to save these two constants, notice that in turn the transitions

`0000 0001 -> 0100 0001 -> 0000 0001`

can be done by applying `xor 0100 0000`. Therefore we initialize a register, chosen here to be `R8`, to `0100 0001` (since the LEDs begin in the both-on state), and after we have applied `xor R8` on the output to obtain the next output, `0000 0000`, we apply `xor 0100 0000` on `R8` to get the next value of `R8`, `0000 0001`, that should be `xor`ed with the next output, and so forth. Below is the full program annotated with comments. Note that although registers are a word long, we only need the last byte, so all of the `mov, xor` operations can be for just the byte.

```
.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 pull for P1.3
    mov.b   #00001000b,      &P1IE       ; P1.3 set as an interrupt
    mov.b   #00001000b,      R7          ; set LEDs off and P1.3 pullup
    mov.b   R7,              &P1OUT      ; LED1, LED2 on
    mov.b   #00000001b,      R8          ; initial value to xor with R7
    EINT                                 ; enable interrupts
    bis.w   #CPUOFF,         SR
PUSH:
    xor.b   R8,              R7          ; next LED state
    xor.b   #01000000b,      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
```

A problem I was encountering was that my P1.3 button seemed to be unpredictably sending multiple signals sometimes, which gave me difficulty in checking if the LED changing behaviour I had programmed was doing what I expected it to do. Therefore, I wrote a loop at the end of `PUSH` to keep on executing the LED changes (with a delay), so that I wouldn't have to press the faulty button to change the lights. Below is the full program for this modification.

```
.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 pull for P1.3
    mov.b   #00001000b,     &P1IE       ; P1.3 set as an interrupt
    mov.b   #00001000b,     R7          ; set LEDs off and P1.3 pullup
    mov.b   R7,             &P1OUT      ; LED1, LED2 on
    mov.b   #00000001b,     R8          ; initial value to xor with R7
    EINT                                ; enable interrupts
    bis.w   #CPUOFF,        SR
PUSH:
    xor.b   R8,             R7          ; next LED state
    xor.b   #01000000b,     R8          ; 0x0041 -> 0x0001 -> 0x0041
    mov.b   R7,             &P1OUT      ; set LEDs to new state
    mov.w   #0xFFFF,        R9          ; decrementing delay in R9
LOOP:
    dec     R9
    nop                                 ; the more nops, the longer the delay
    nop
    nop
    nop
    jnz     LOOP
    jmp     PUSH

    org 0xffe4
    dw PUSH                             ; interrupt from button goes here

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