Lab 2: Morse Code Decoder

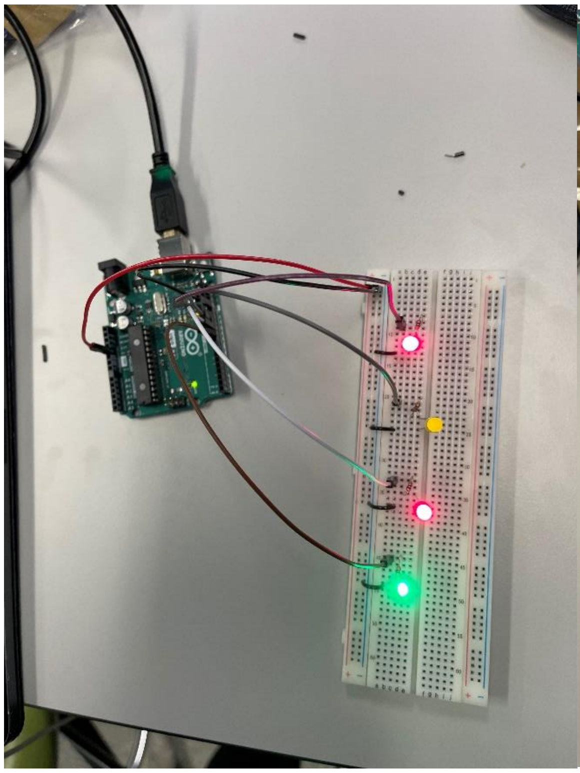
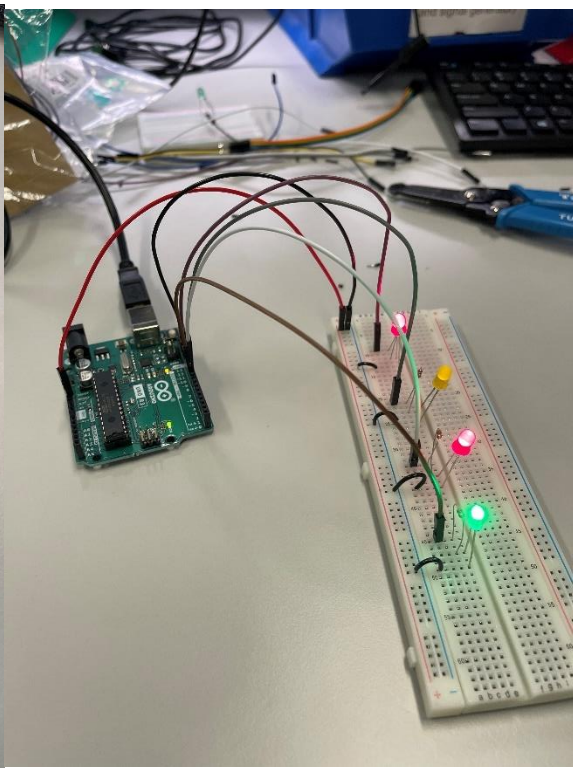
ESE519/IPD519: Introduction to Embedded Systems
University of Pennsylvania

In this document, you'll fill out your responses to the questions listed in the Lab 2 Manual. Please fill out your name and link your Github repository below to begin. Be sure that your code on the repo is up-to-date before submission!

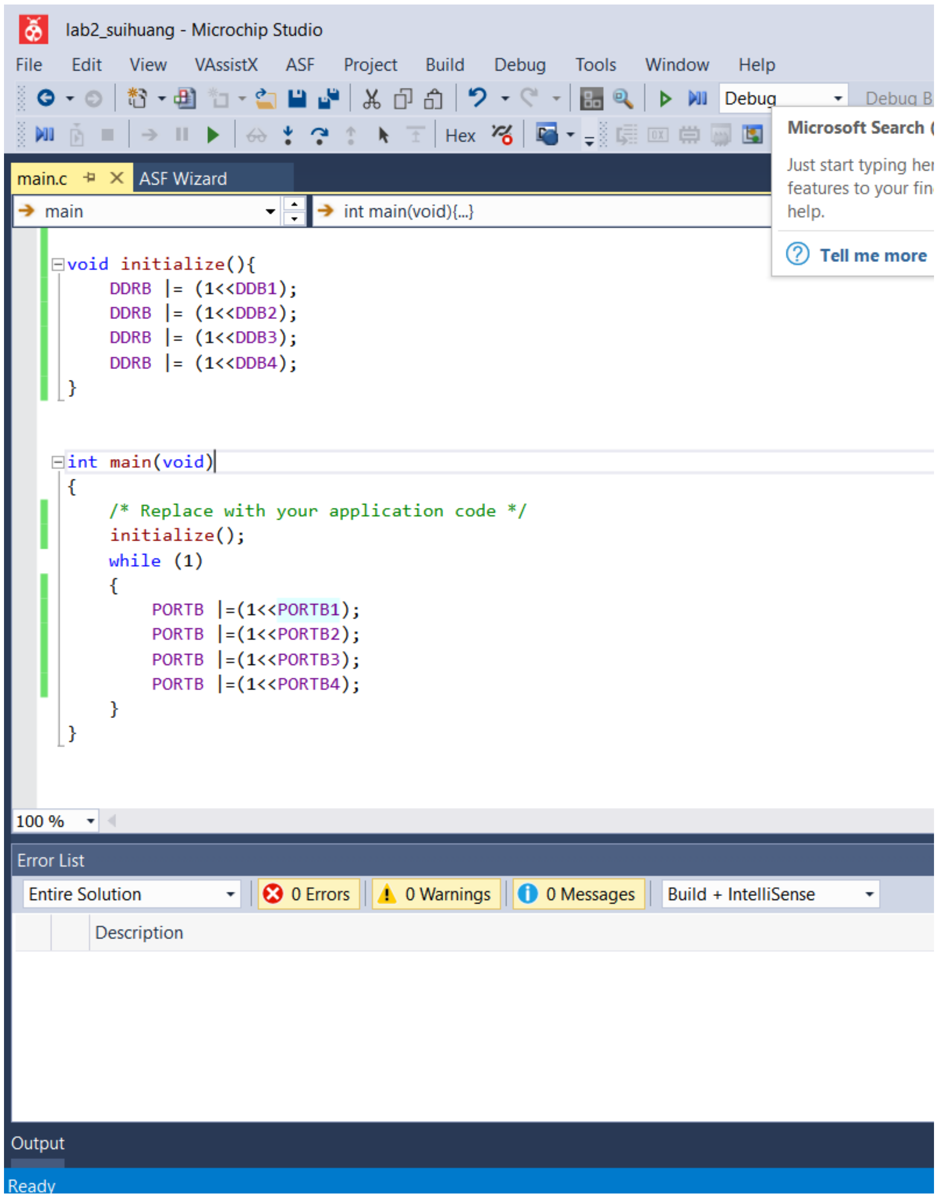
For all the questions that require a video, you can attach the video/image directly to the relevant question number or provide a link to the video (e.g. youtube, google drive, etc.).

Student Name: Sui Huang
Pennkey: suihuang
GitHub Repository:
https://github.com/GloriaSuiHuang/lab2-suihuang

1.

2.

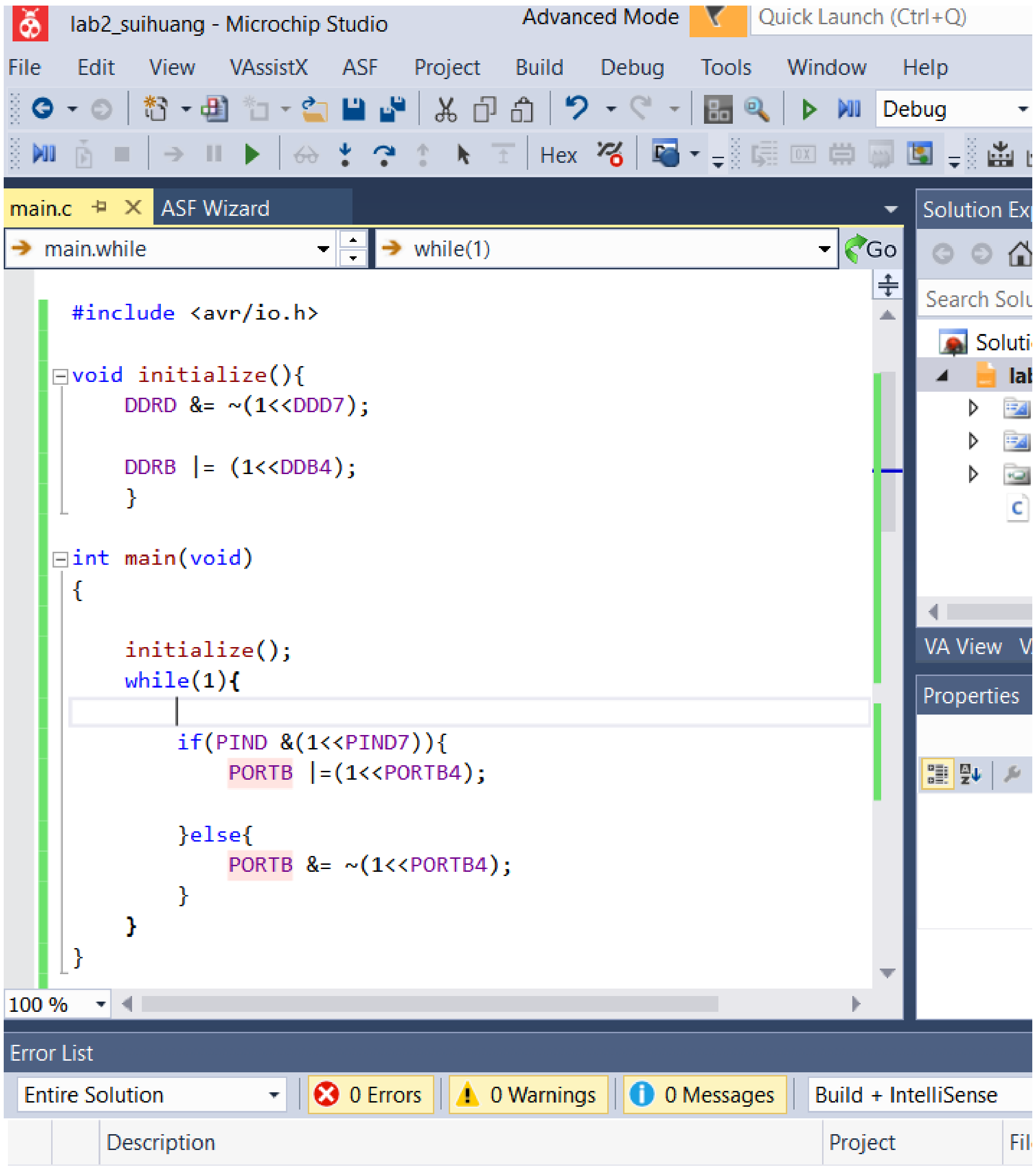


```c
void initialize(){
    DDRB |= (1<<DDB1);
    DDRB |= (1<<DDB2);
    DDRB |= (1<<DDB3);
    DDRB |= (1<<DDB4);
}

int main(void)
{
    /* Replace with your application code */
    initialize();
    while (1)
    {
        PORTB |=(1<<PORTB1);
        PORTB |=(1<<PORTB2);
        PORTB |=(1<<PORTB3);
        PORTB |=(1<<PORTB4);
    }
}
```

3.

For this question, I set PD7 as input and PB4 as output. The LED remain on as the button is pressed and turn off when the button is released. Hence, the LED turn on and off as expected.

4.



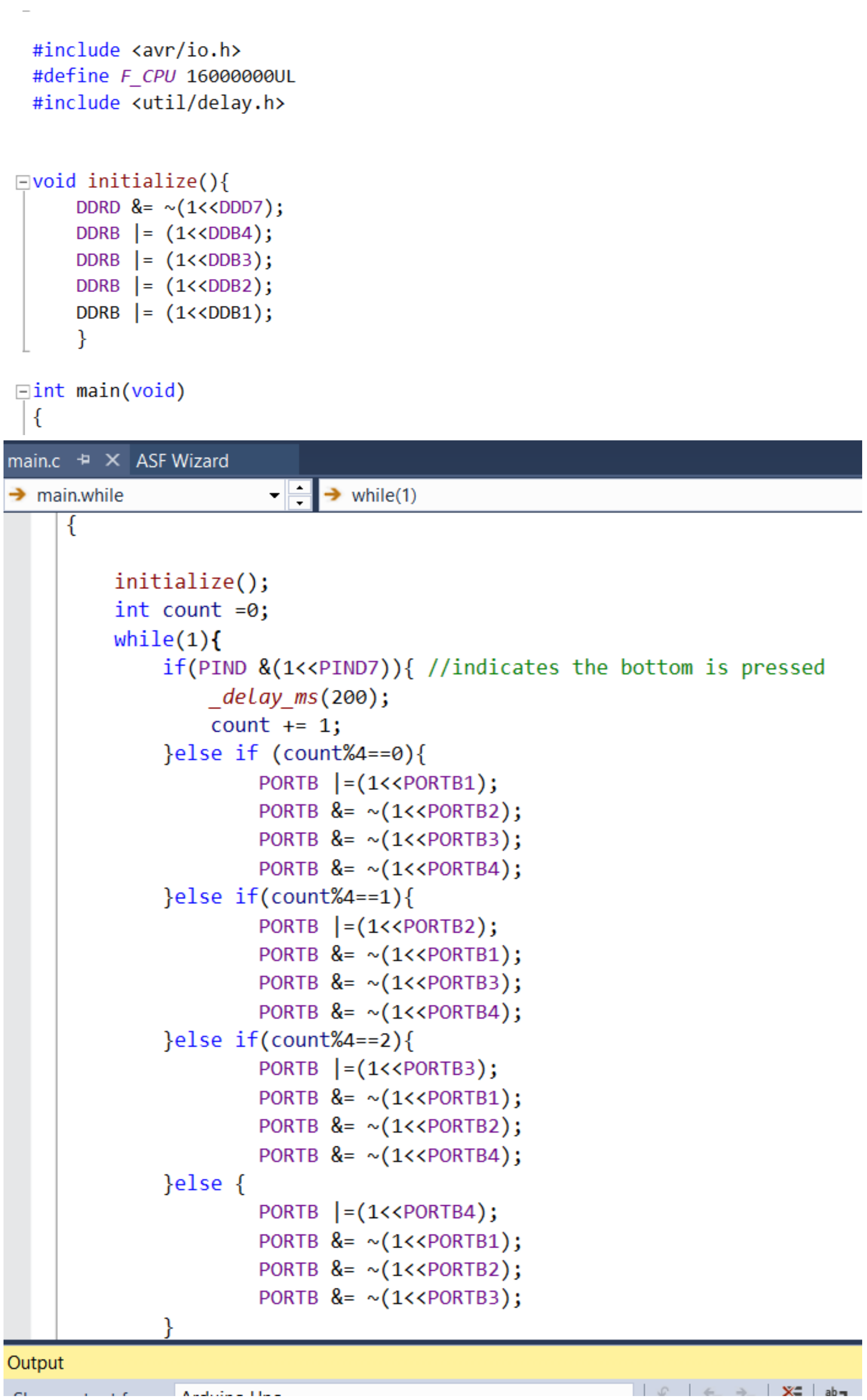
```c
#include <avr/io.h>

void initialize(){
    DDRD &= ~(1<<DDD7);

    DDRB |= (1<<DDB4);
    }

int main(void)
{

    initialize();
    while(1){
        
        if(PIND &(1<<PIND7)){
            PORTB |=(1<<PORTB4);

        }else{
            PORTB &= ~(1<<PORTB4);
        }
    }
}
```

5.

6.

Yes, the LED turn on and off as expected.

7.



```
#include <avr/io.h>
#define F_CPU 16000000UL
#include <util/delay.h>


void initialize(){
    DDRD &= ~(1<<DDD7);
    DDRB |= (1<<DDB4);
    DDRB |= (1<<DDB3);
    DDRB |= (1<<DDB2);
    DDRB |= (1<<DDB1);
    }

int main(void)
{
```

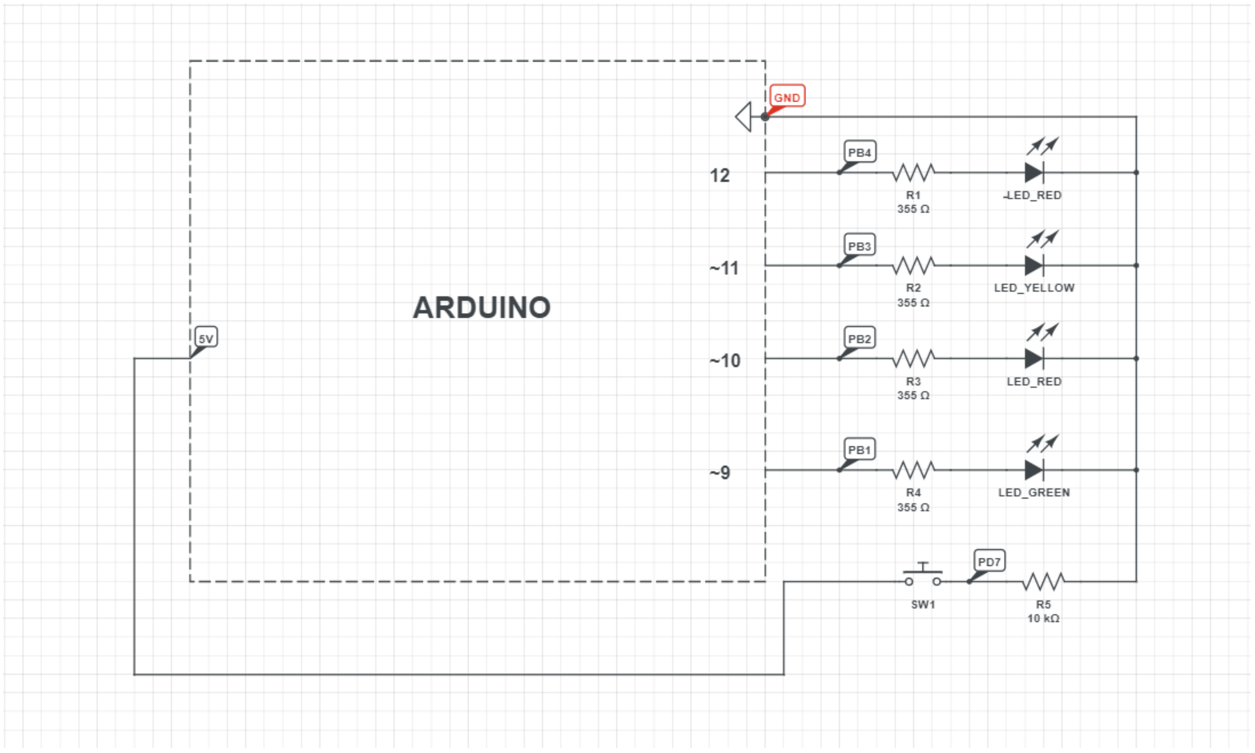
main.c   ASF Wizard

main.while   while(1)

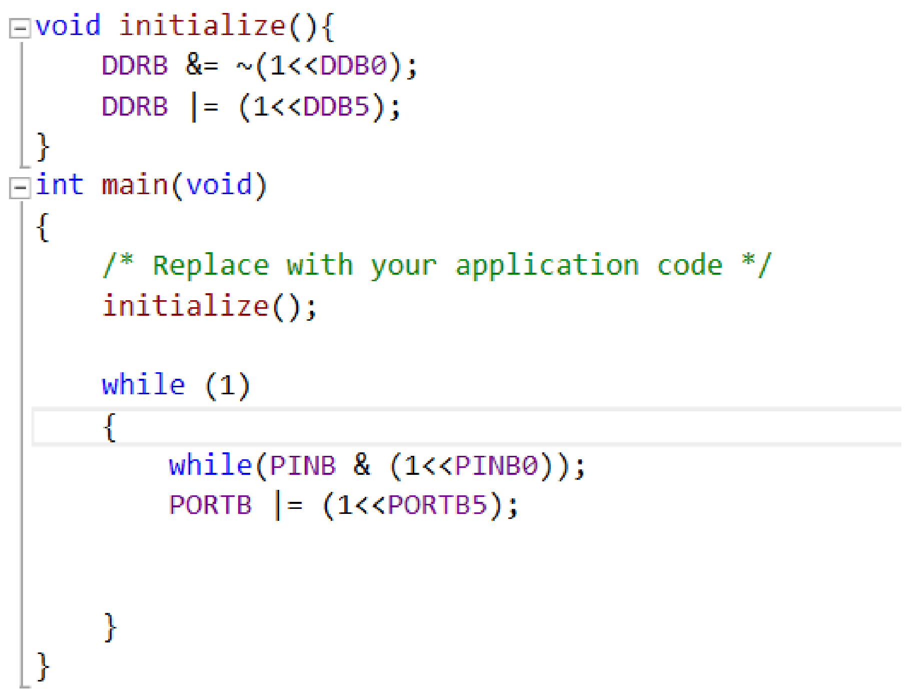
```
{

    initialize();
    int count =0;
    while(1){
        if(PIND &(1<<PIND7)){ //indicates the bottom is pressed
            _delay_ms(200);
            count += 1;
        }else if (count%4==0){
                PORTB |=(1<<PORTB1);
                PORTB &= ~(1<<PORTB2);
                PORTB &= ~(1<<PORTB3);
                PORTB &= ~(1<<PORTB4);
        }else if(count%4==1){
                PORTB |=(1<<PORTB2);
                PORTB &= ~(1<<PORTB1);
                PORTB &= ~(1<<PORTB3);
                PORTB &= ~(1<<PORTB4);
        }else if(count%4==2){
                PORTB |=(1<<PORTB3);
                PORTB &= ~(1<<PORTB1);
                PORTB &= ~(1<<PORTB2);
                PORTB &= ~(1<<PORTB4);
        }else {
                PORTB |=(1<<PORTB4);
                PORTB &= ~(1<<PORTB1);
                PORTB &= ~(1<<PORTB2);
                PORTB &= ~(1<<PORTB3);
        }
```

Output

8.


ARDUINO
GND
5V
12
~11
~10
~9
PB4
PB3
PB2
PB1
PD7
R1
355 Ω
R2
355 Ω
R3
355 Ω
R4
355 Ω
R5
10 kΩ
LED_RED
LED_YELLOW
LED_RED
LED_GREEN
SW1

9.



```
void initialize(){
    DDRB &= ~(1<<DDB0);
    DDRB |= (1<<DDB5);
}
int main(void)
{
    /* Replace with your application code */
    initialize();

    while (1)
    {
        while(PINB & (1<<PINB0));
        PORTB |= (1<<PORTB5);


    }
}
```

So initially I set PB0 to be input, and PB5 to be my output. I am using a while loop to approach my polling function. Because I want to keep checking if PIN8(PB0) has been pressed. If PIN8 is pressed, then my PINB & (1<<PINB0) would turn into zero, which is active low. (That means PIN8 is connected to GND.) Here is when my event happens. Then break the while loop, and PB5, which connects to the magical onboard LED should lit. My observation of my circuit board acts the same as I implemented.
When nothing happens, the PIN8 is polled high all the time; when you press the button, event occur, break the while loop, the LED is on.
This is different from the previous part, where I have active high. The PIN initially connect to GND, and when you pressed, it turns into high(Vdd).

10.

Advantage of interrupt over polling: in the polling method, a microcontroller continuously checks the status of a register continuously and do the task according to the status of the register. Normally, the hardware is very slow in terms of processing speed with microcontroller. Therefore, if we implement Polling method, it is going to slow up the speed of the microcontroller and hence, performance. For interrupt method, instead, the device or the register send the signal to the microcontroller that it is ready. The microcontroller stops whatever it is executing and starts to execute the program you have written for interrupts(Interrupt Service Routine-ISR). After executing the ISR, the microcontroller resumes the program where it had left. So the performance of microcontroller is far better in Interrupt method than Polling Method.

Disadvantage of interrupt over polling: interrupts require more complex hardware/software and loss of time until the CPU establishes which units request for interruption.

11.

16 MHZ = $\frac{16\ M\ tics}{S}$
When t = 30ms, tics = $\frac{16\ M\ tics}{S}$ * 30ms = 4.8*10^5 tics
When t = 200ms, tics = $\frac{16\ M\ tics}{S}$ * 200 ms = 3.2*10^6 tics
When t = 400ms, tics = $\frac{16\ M\ tics}{S}$ * 400 ms = 6.4*10^6 tics

12.

A prescaler is an electronic counting circuit used to reduce a high frequency electrical signal to a lower frequency by integer division. A prescaler divides down the clock signals used for the timer, giving reduced overflow rates. With a prescaler, you can control how fast you want your system clock to count or how fast you want your timer to count. That would reduce the time interval overflow occurs.

13. This is demoed to Professor KIM.

14.

Jason in the comic on the first page tap out:

“someday I will rule you all”

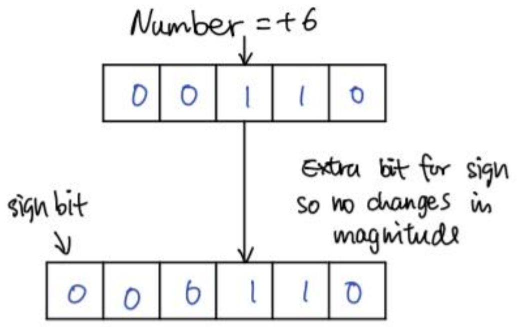
15.

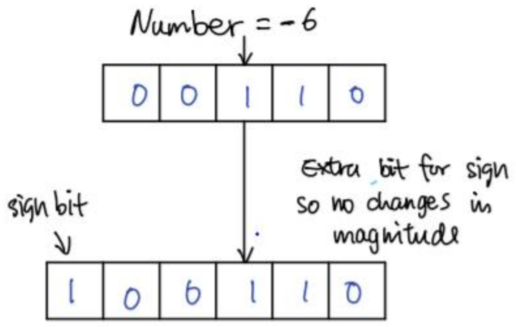
In mathematics, positive numbers(including zero)are represented as unsigned numbers. That is we do not put the +ve sign in front of them to show that they are positive numbers. If we want to represent a negative number, we would put a negative sign before them in order to show that the number is negative in value and different from a positive unsigned value, and the same is true with signed binary numbers. However, in digital circuits there is no provision made to put a plus or even a minus sign to a number, since digital systems operate with binary numbers that are represented in terms of “0’s” and “1’s”. But Digital Systems and computers must also be able to use and to manipulate negative numbers as well as positive numbers. But how do we represent signed binary numbers if all we have is a bunch of one’s and zero’s. We know that binary digits,

or bits only have two values, either a “1” or a “0” and conveniently for us, a sign also has only two values, being a “+” or a “–“.
We represent negative binary numbers using a minus symbol in front of them. In computer number representation, these numbers can be distinguishable with the help of an extra bit or flag called sign bit or sign flag in the Binary number representation system for signed numbers. This extra bit is called sign bit or sign flag which has a value of sign bit is 0 for positive numbers and 1 for negative binary numbers.
There are three methods to represent negative numbers:

1. Signed Magnitude Method :

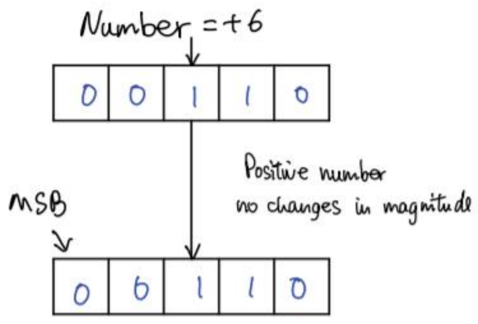
We only add an extra sign bit to recognize negative and positive numbers. Sign bit has 1 for negative number and 0 for positive number.

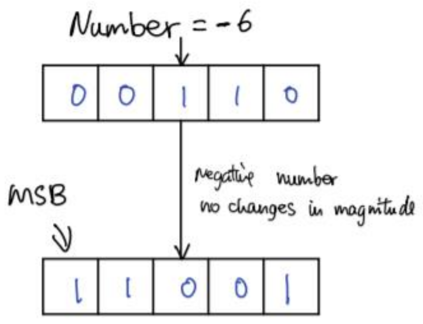




2. 1’s Complement Method :

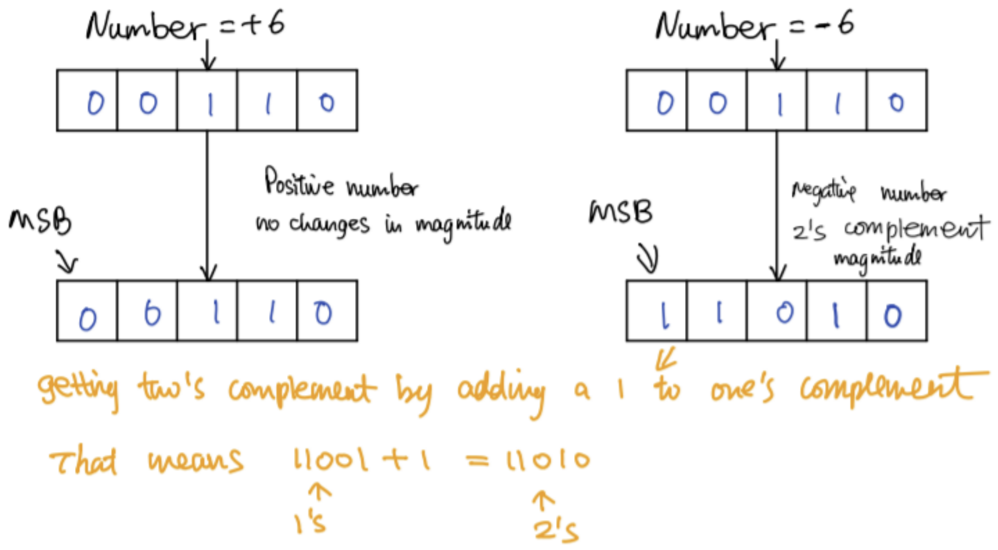
Please note that MSB is always a sign bit.





3. 2’s Complement Method :

Please note that MSB is always a sign bit.



16.

For integers we have 16 bits to store the number in a range of [0, 65535];

For example:

$$(0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 1)_2$$

$$= 0+0+0+0+0+0+0+0+0+0+0+1\times2^4+0\times2^3+0\times2^2+0\times2^1+1\times2^0$$

$$=(17)_{10}$$

Recall the scientific notation:

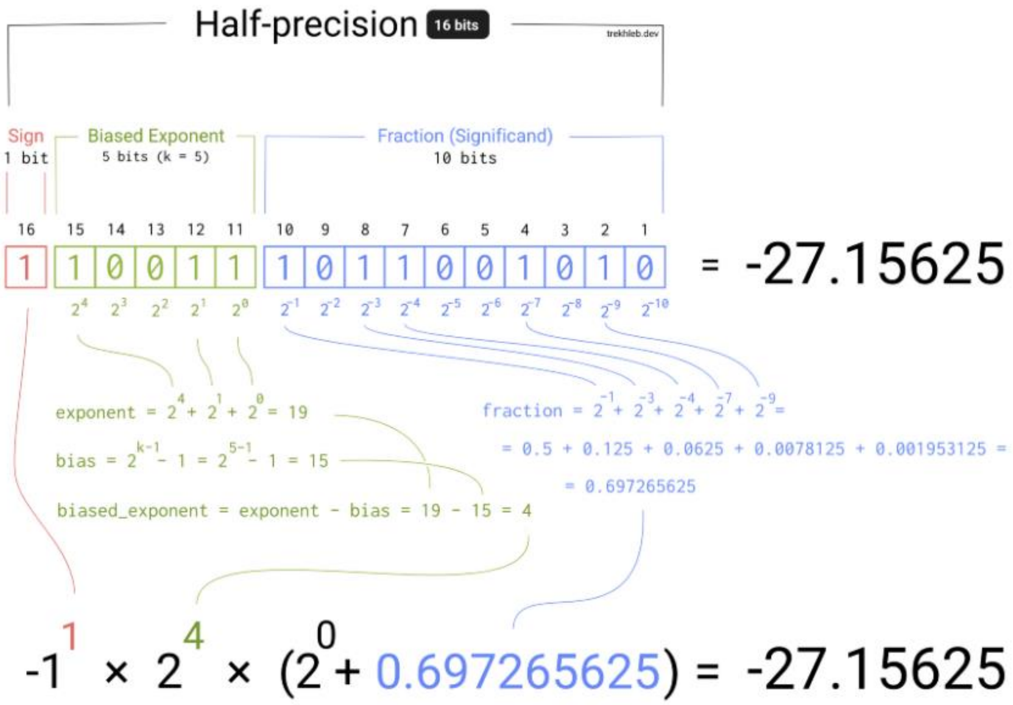
$$420000 \rightarrow 1 \times 4.2 \times 10^{5}$$

$$-0.0042 \rightarrow \underset{\text{sign}}{-1} \times \underset{\text{fraction}}{4.2} \times \underset{\text{base}}{10}^{\overset{\text{exponent}}{-3}}$$

For floating numbers, we have sign, fraction, and exponent.

Biased exponent--- subtract the bias from the exponent value to make it negative. For example, if the exponent has 5 bits, it might take the values from the range of [0, 31] (all values are positive here). But if we subtract the value of 15 from it, the range will be [-15, 16]. The number 15 is called bias, and it is being calculated by the following formula:

```
exponent_bias = 2 ^ (k−1) − 1

k – number of exponent bits
```


Half-precision
16 bits
trekhleb.dev
Sign
1 bit
Biased Exponent
5 bits (k = 5)
Fraction (Significand)
10 bits
16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
1 1 0 0 1 1 1 0 1 1 0 0 1 0 1 0
= -27.15625
2^4 2^3 2^2 2^1 2^0 2^{-1} 2^{-2} 2^{-3} 2^{-4} 2^{-5} 2^{-6} 2^{-7} 2^{-8} 2^{-9} 2^{-10}
exponent = $2^4 + 2^1 + 2^0$ = 19
bias = $2^{k-1} - 1 = 2^{5-1} - 1$ = 15
biased_exponent = exponent - bias = 19 - 15 = 4
fraction = $2^{-1} + 2^{-3} + 2^{-4} + 2^{-7} + 2^{-9}$ =
= 0.5 + 0.125 + 0.0625 + 0.0078125 + 0.001953125 =
= 0.697265625
$-1^1 \times 2^4 \times (2^0 + 0.697265625) = -27.15625$