Programming Activities

1. If I press and release SW2 quickly, each light flashes once. If a turtle were to press it quickly, they might actually flash multiple times. If held however, the lights will continue to flash on and off repeatedly until I let go of SW2. This makes sense since the program states that as long as “SW2 == 0” (Pressed), the lights will turn on and off, and nothing is telling the code to stop afterwards.
2. LED3 = 0 assigns the value 0 to LED3 (turns the LED off), while LED3 = 1 assigns the value 1 (turns the LED on).
3. Since when LED3 = 0, the light is off, I would assume that there’s no current flowing through the light, and therefore no voltage. With LED3 = 1 (turned on), the voltage is about 2 V across that LED.
4. “=” assigns a value to a variable, while “==” checks if a variable is equivalent to something. For example, LED3 = 1 assigns the integer ‘1’ to LED3, while if(SW2 == 0) is a boolean statement checking to see if SW2 does actually have the value ‘0’.
5. When SW3 is pressed, all the LEDs are turned off, and then turned on. One advantage of writing these statements using “LATC” is that it takes up less lines of code, since the instruction applies to everything connected to the port- since the computer only has to read 1 line of code to turn on multiple LEDs, it’s faster than having the computer run through individual lines of code for each light. The disadvantage is that it’s harder to read, since you have to be aware of the binary number that represents each LED.
6. Holding SW3 while spamming SW4 does work as expected. Switching SW3 and SW4 though, causes the SW3 switch to stop working. The way I logiced this out for myself was that “if” statements are just saying, “Hey your program can do its thing, but when this happens, x does this.” However, “while” statements are saying, “As long as this is true, your computer will do this, and only this until it’s no longer true.”
7. The order of the conditions don’t matter (works even if I switch which one’s nested into the other).
8. The result of the “&&” boolean statement and the nested “if” statements are the same, but I guess it works a little bit differently when it comes to the computer reading it. With the “if” statements, when you press one of the buttons, the computer actually reads that one of your conditions is true (assuming you press the button that applies to the encompassing “if” statement). With the “&&” statement, you meet no conditions at all unless both switches are pressed. The advantage of “&&” operators is it takes out an extra line and nest, which makes your program easier to read and understand.
9. LED4 now turns on when either one of the switches are pressed as well as when both are pressed.

Part 2:

1. It just changes the delay between the execution of each line of code, which in this case affects how long it takes between each light being turned on and off. IThe biggest I can make the delay is 4205 ms (milliseconds).
2. As the delay decreases, the pitch increases- the vibrations happen more frequently due to less delay, and higher frequencies create higher pitches.
3. After running the code once, the beeper would still be considered true/on. If you run it once again, it will switch from on to off. The disadvantage of this is that you’re probably letting go of the button after the code has already ran multiple times over, so you don’t know whether or not it’s still on.
4. if(SW2 == 0)

{

LED3 = 1;

\_\_delay\_ms(10);

LED4 = 1;

\_\_delay\_ms(10);

LED5 = 1;

\_\_delay\_ms(10);

LED6 = 1;

\_\_delay\_ms(10);

LED3 = 0;

\_\_delay\_ms(10);

LED4 = 0;

\_\_delay\_ms(10);

LED5 = 0;

\_\_delay\_ms(10);

LED6 = 0;

\_\_delay\_ms(10);

}

// Add code for your Program Analysis and Programming Activities here:

if(SW3 == 0)

{

LED3 = 1;

\_\_delay\_ms(100);

LED3 = 0;

\_\_delay\_ms(200);

LED5 = 1;

\_\_delay\_ms(100);

LED5 = 0;

\_\_delay\_ms(200);

}

if(SW4 == 0)

{

LATC = 0b00000000;

\_\_delay\_ms(1000);

LATC = 0b11110000;

\_\_delay\_ms(10);

LATC = 0b00000000;

}

if(SW5 == 0)

{

LED3 = 1;

LED5 = 1;

\_\_delay\_ms(200);

LED3 = 0;

LED5 = 0;

LED4 = 1;

LED6 = 1;

\_\_delay\_ms(200);

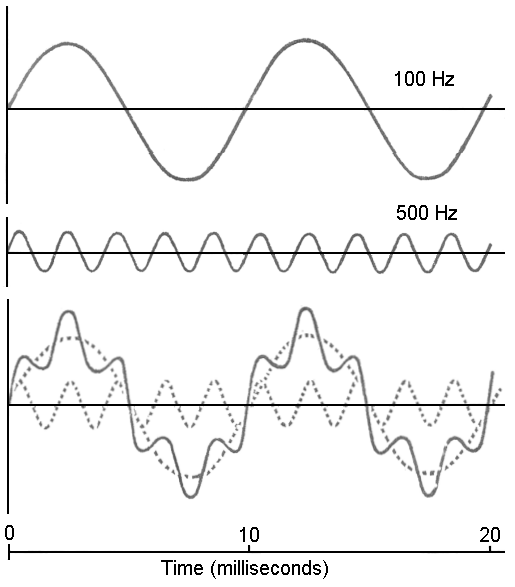
LED4 = 0;

LED6 = 0;

}

Whichever button I clicked first, the corresponding program runs. As long as I keep all the buttons pressed, the code runs in the order I initially clicked them.

1. Basically the higher frequency would interfere with the low frequency waves, so the waveform would have the general shape of the lower frequency beep, but with small crests and troughs within the line. Example:



if(SW5 == 0)

{

BEEPER = !BEEPER;

\_\_delay\_us(567);

}

if(SW4 == 0)

{

BEEPER = !BEEPER;

\_\_delay\_us(120);

}

if(SW3 == 0)

{

BEEPER = !BEEPER;

\_\_delay\_us(900);

}

if(SW2 == 0)

{

BEEPER = !BEEPER;

\_\_delay\_us(100);

}

1. if(SW3 == 0)

{

LED4 = 1;

}

if(SW4 == 0)

{

LED4 = 0;

}

1. When both are pressed, the light stays on but dimmer. Most likely the light is actually flickering on and off really fast, which is why it appears a little bit dimmer.
2. if (SW3 == 0)

{

LED4 = 1;

}

while (SW4 == 0)

{

LED4 = 0;

}

1. if(SW2 == 0)

{

LED1 = 0;

\_\_delay\_ms(75);

LED1 = 1;

\_\_delay\_ms(75);

}