Class:	CPE301L Digital Systems Architecture and Design 1001		Semester:	Spring 2025
Points		Document author:	Narek Kalikian	
		Author's email:	kalikn1@unlv.nevada.edu	
		Document topic:	Postlab 2	
Instructor's comments:				

1. Introduction / Theory of Operation

This lab focused on getting familiar with Microchip Studio (formally Atmel Studio) by programming various projects using assembly and C on ATmega 328P boards. In the first two experiments, the code was provided in the lab manual so that we could get accustomed to implementing code on the boards. Afterward, we were tasked with writing our own code for the next two experiments to understand how different functions and variables interact together on the board.

2. Prelab Content

1. Prelab Question #3 - Bit Masking:

Bitmasking is when a number is modified or manipulated by applying a binary number to it using bitwise operations in programming. Each specific bit of the number can be manipulated using bit masking. A bitwise AND can be used to extract a subset of bits, a bitwise OR can be used to set a subset of bits, and a bitwise XOR can be used to toggle a subset of bits. For example:

```
int x = 11101010;
int mask = 00001101;
int result = x & mask;                    // result = 00001010
```

// Using a bitwise AND (&) between the value (x) and the mask maps the 4 most significant bits of mask (0000) over the 4 most significant bits of x (1110). This gives a result of 00001010

which effectively extracts the 4 least significant bits of x (1010).

2. Identifying and Converting Numbers:

- Decimal: **101**
  - Binary Representation: 0b1100101
  - Hex Representation: 0x65
- Binary: **0b101**

    - Decimal Representation: 5
    - Hex Representation: 0x5
- Hexadecimal: **0x101**
    - Binary Representation: 0b100000001
    - Decimal Representation: 257

3. Operation Solutions:

- 1011 >> 2: Right shift of 2 (Divide by 4) = **0010**
- 1011 ^ 1010: 1011 XOR 1010

  1011
  1010

  0001

- 1011 ^ (~1011): 1011 XOR NOT(1011), NOT(1011) = 0100

  1011
  0100

  1111

- 1011 & (~0 << 2): 1011 AND (NOT(0) shifted left 2 bits)
  NOT(0000) = 1111
  1111 << 2 = 1100
  1011 AND 1100

  1011
  1100

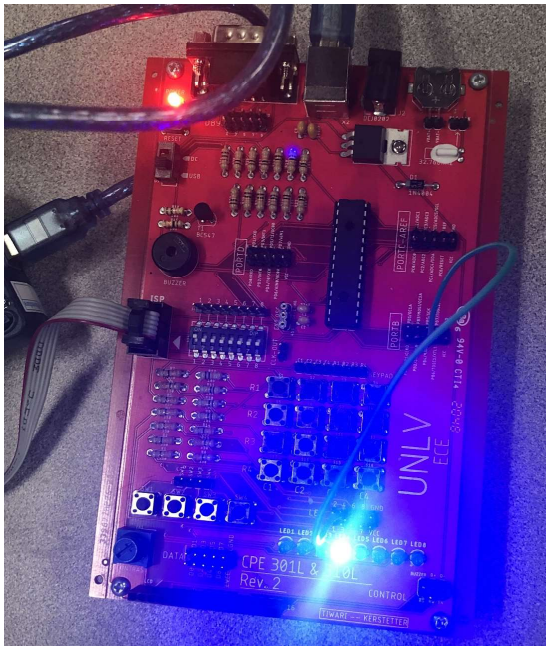
  1000

4. Status Register:

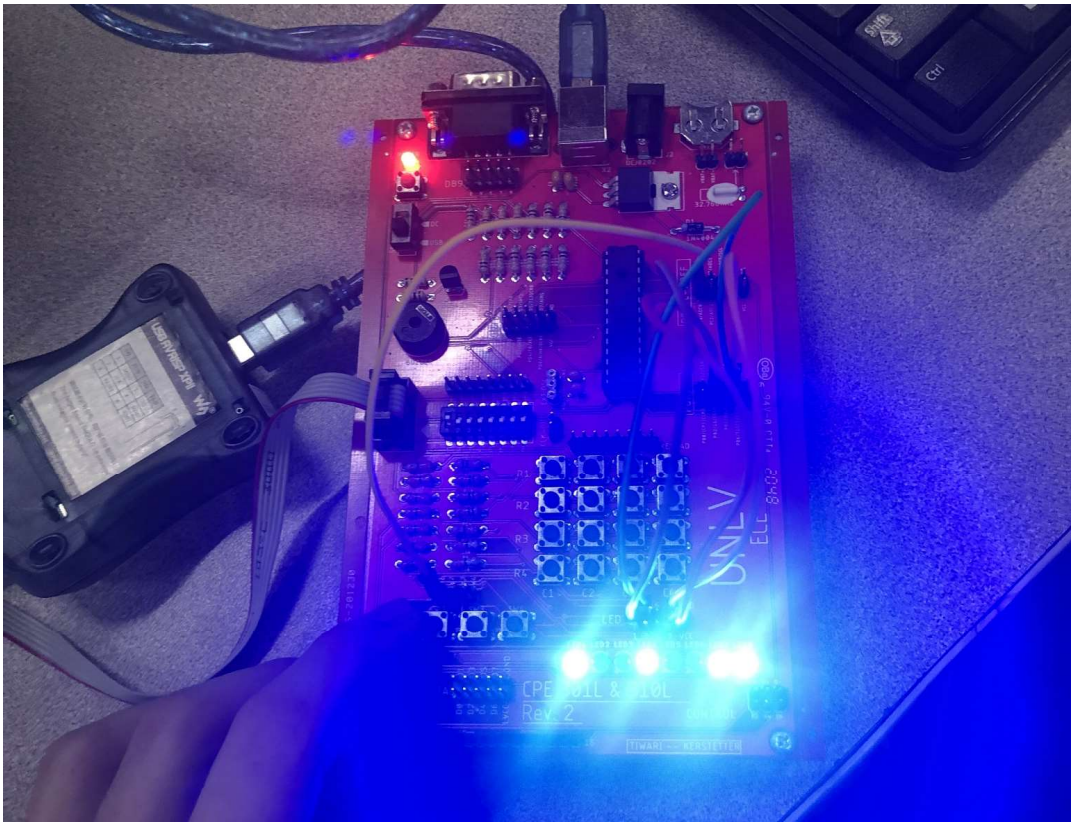
In AVR, the status register is an 8-bit register. It is used to tell the state of the processor after A/L operations occur (hence the name “status”). Essentially, the status register keeps track of the results of these operations. It is located in SRAM with special function registers and is made up of status bits, or flags. These flags are used to determine certain conditions such as carry, half carry, zero, negative, etc. In total, there are 6 conditions that set the status register. These conditions are addition/subtraction, multiplication, logical operations, comparison operations, shift/rotate, and branching/bit manipulation.

3. Description of Experiments

Experiment 1 - Board Picture:

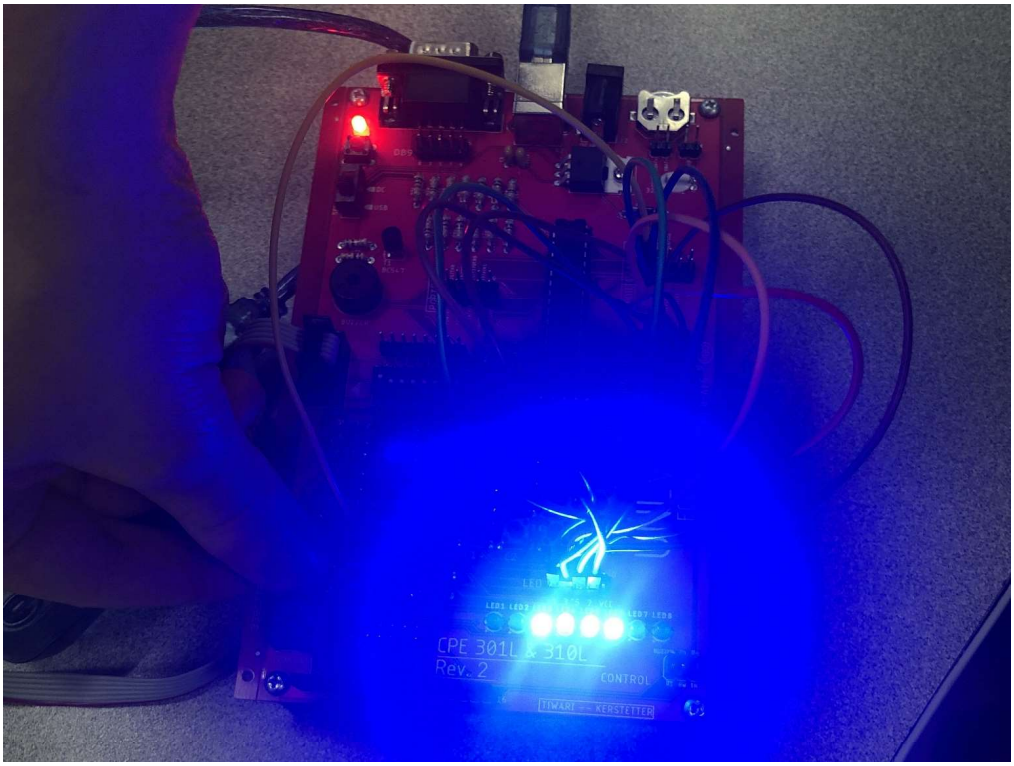


Experiment 2 - Board Picture:

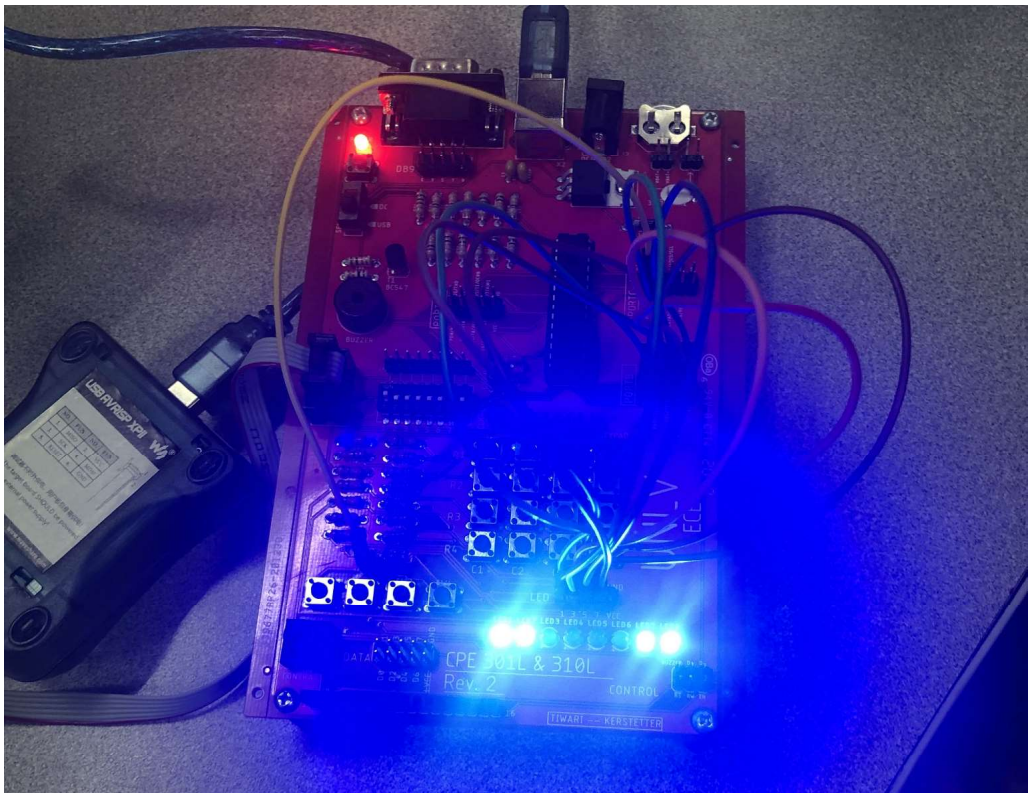


Experiment 3 - C and Assembly Code: ***Zipped and Attached in Canvas Submission***

Experiment 3 - Sequence 1 (Button Pressed) Board Picture:

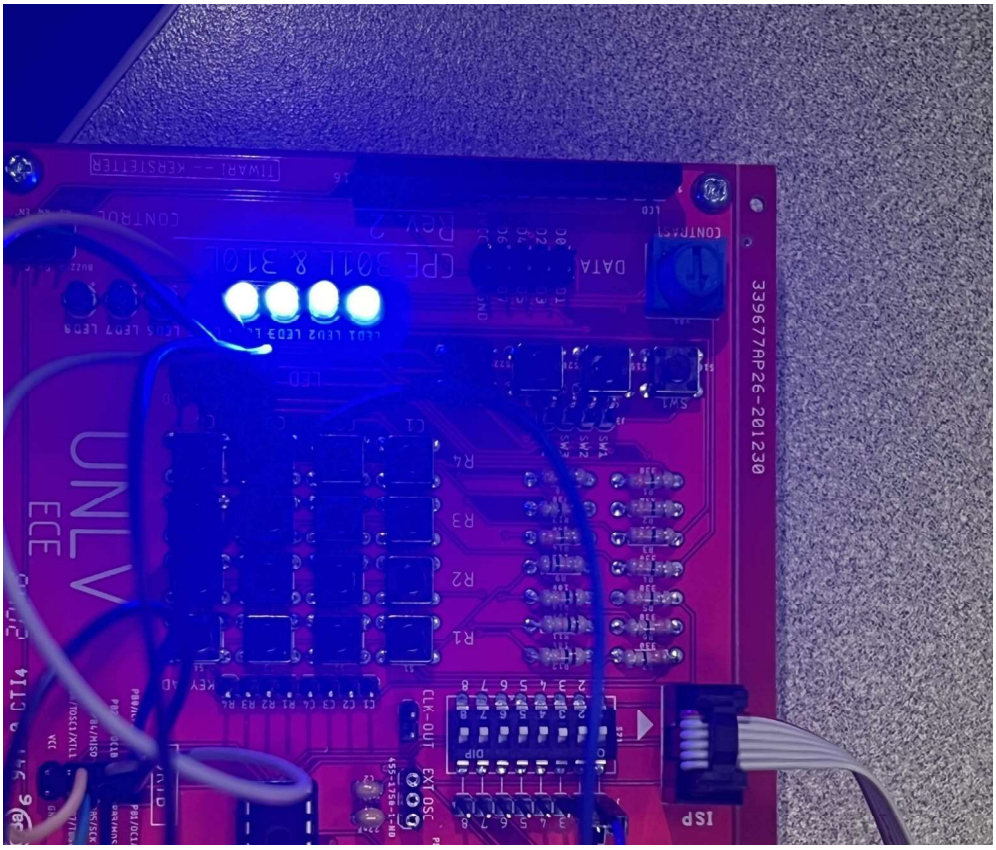


Experiment 3 - Sequence 2 (Button Not Pressed) Board Picture:

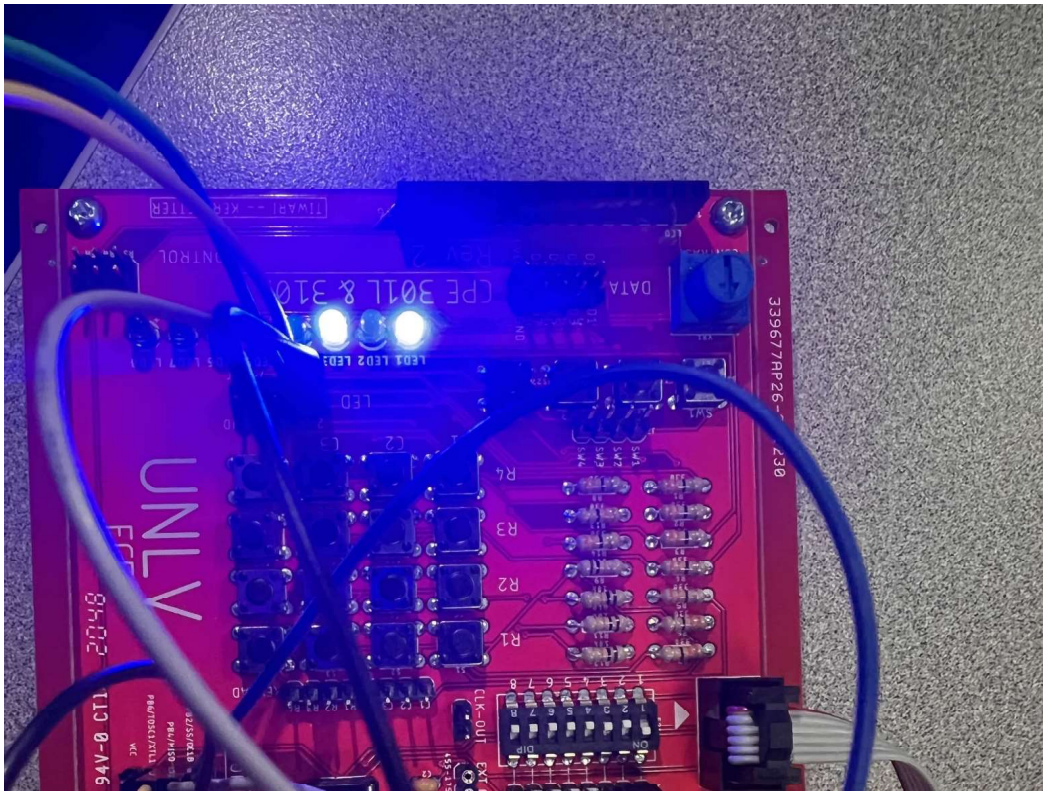


Experiment 4 - C and Assembly Code: ***Zipped and Attached in Canvas Submission***

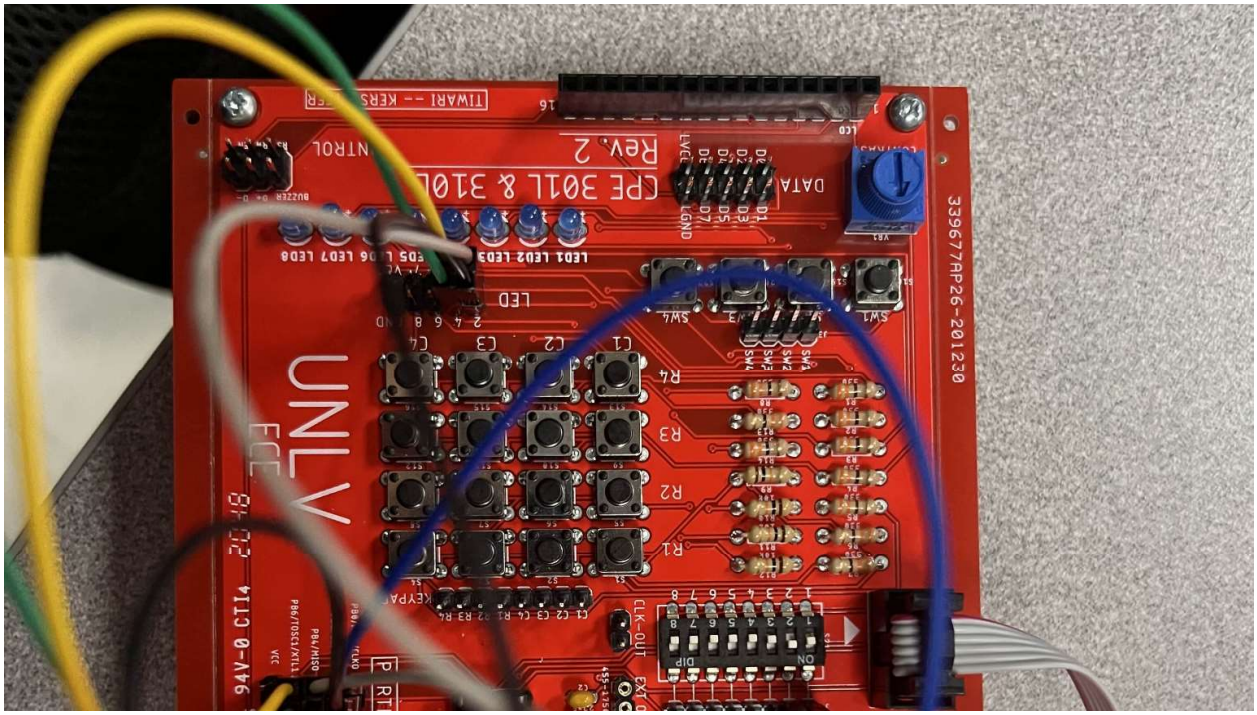
Experiment 4 - Dip Switch = 00 Board Picture:



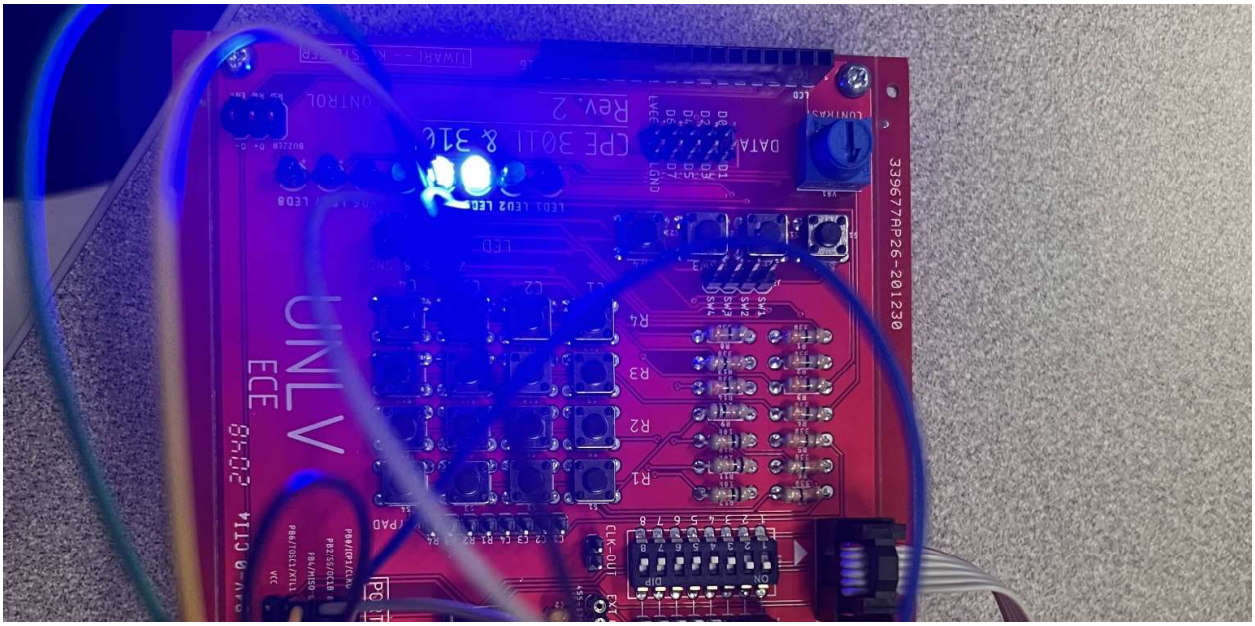
Experiment 4 - Dip Switch = 01 Board Picture:



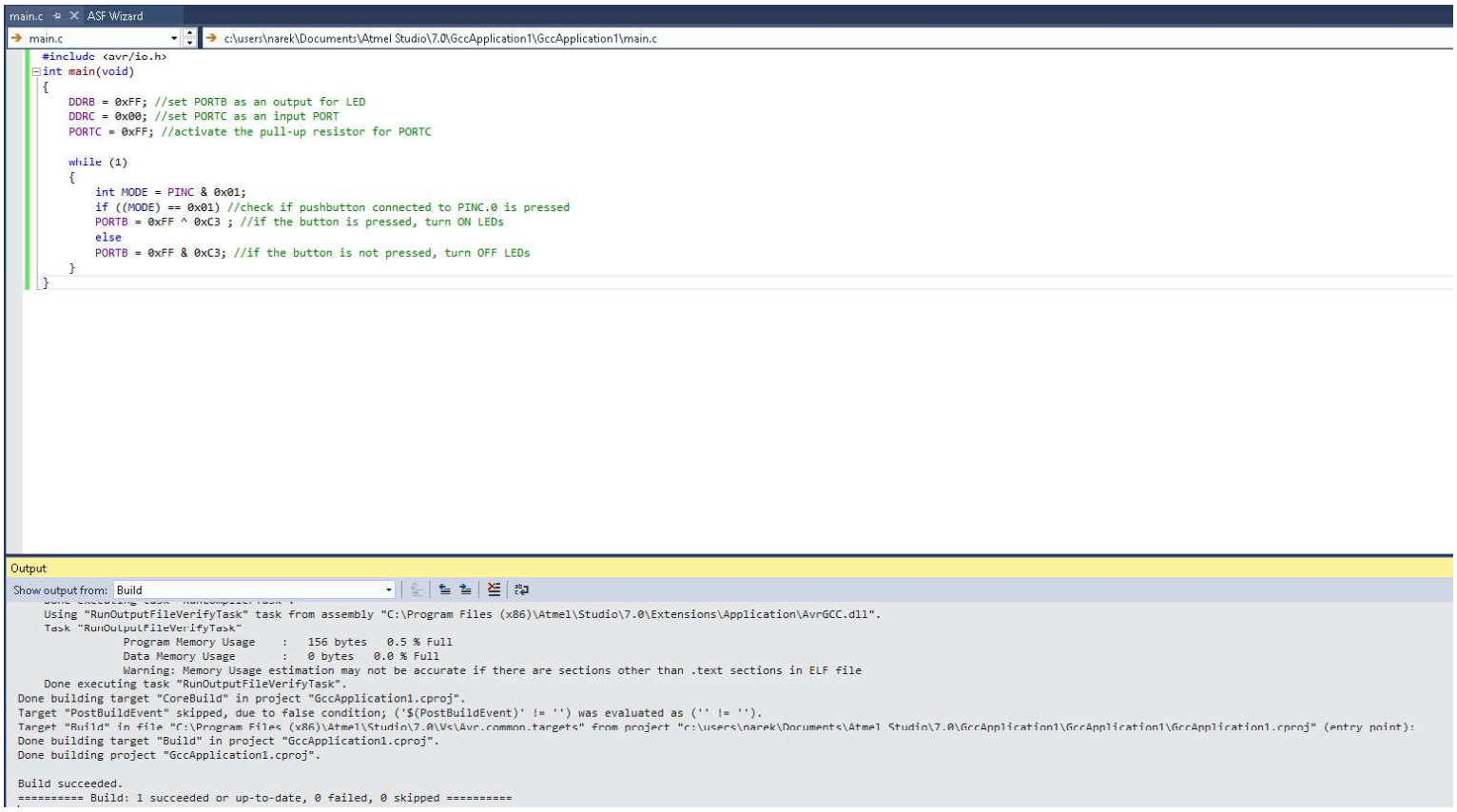
Experiment 4 - Dip Switch = 10 Board Picture:



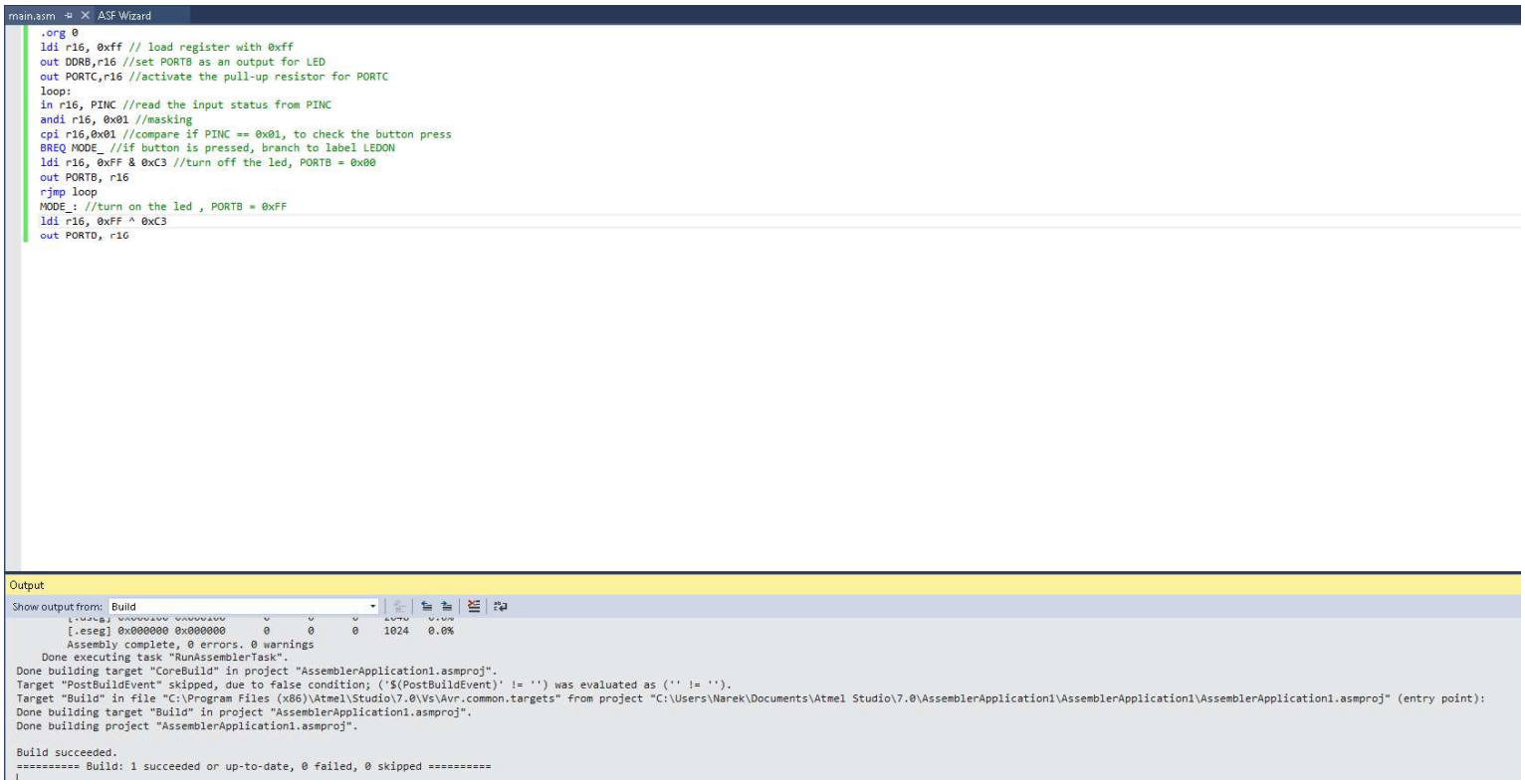
Experiment 4 - Dip Switch = 11 Board Picture:



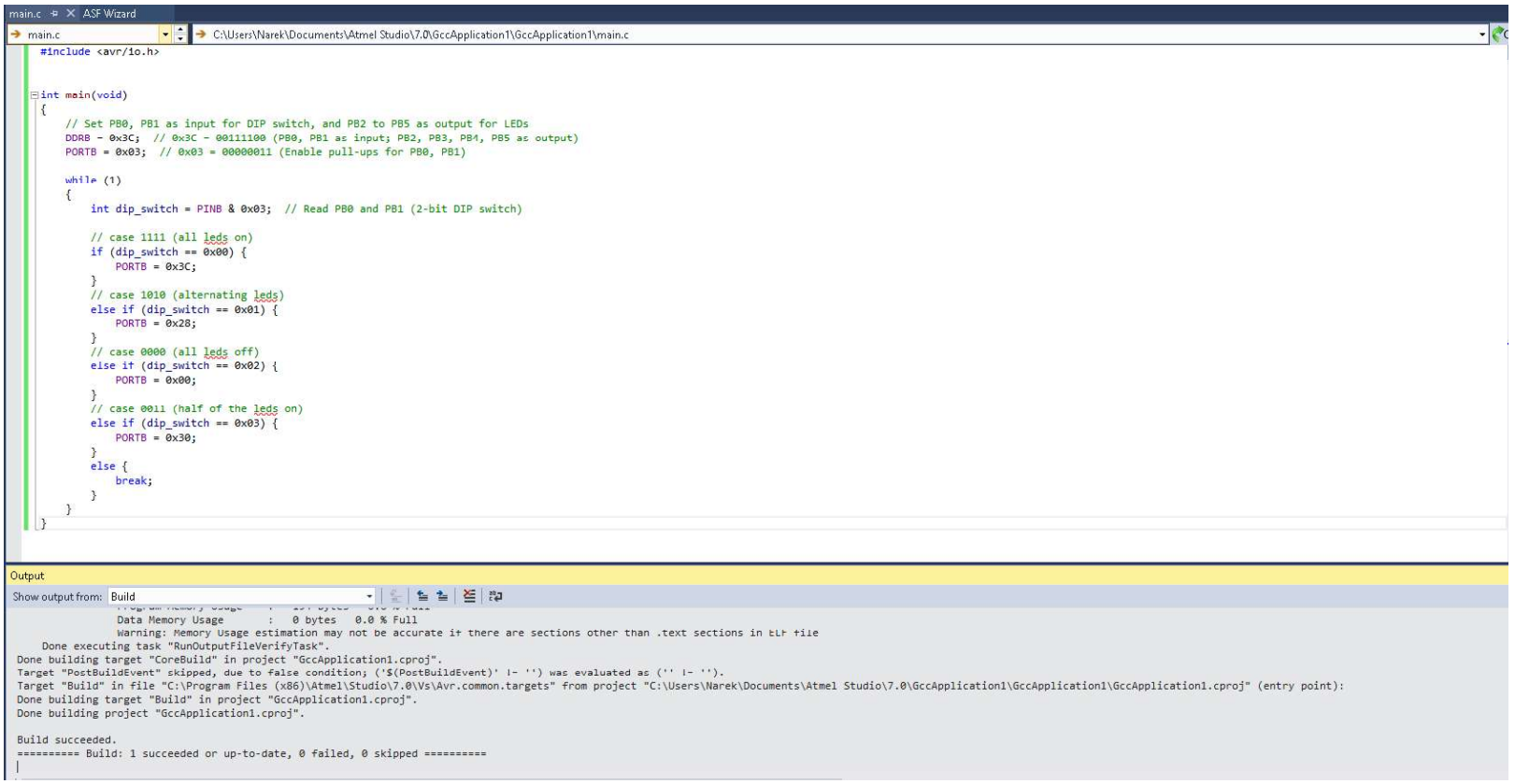
Experiment 3 C Code Screenshot:



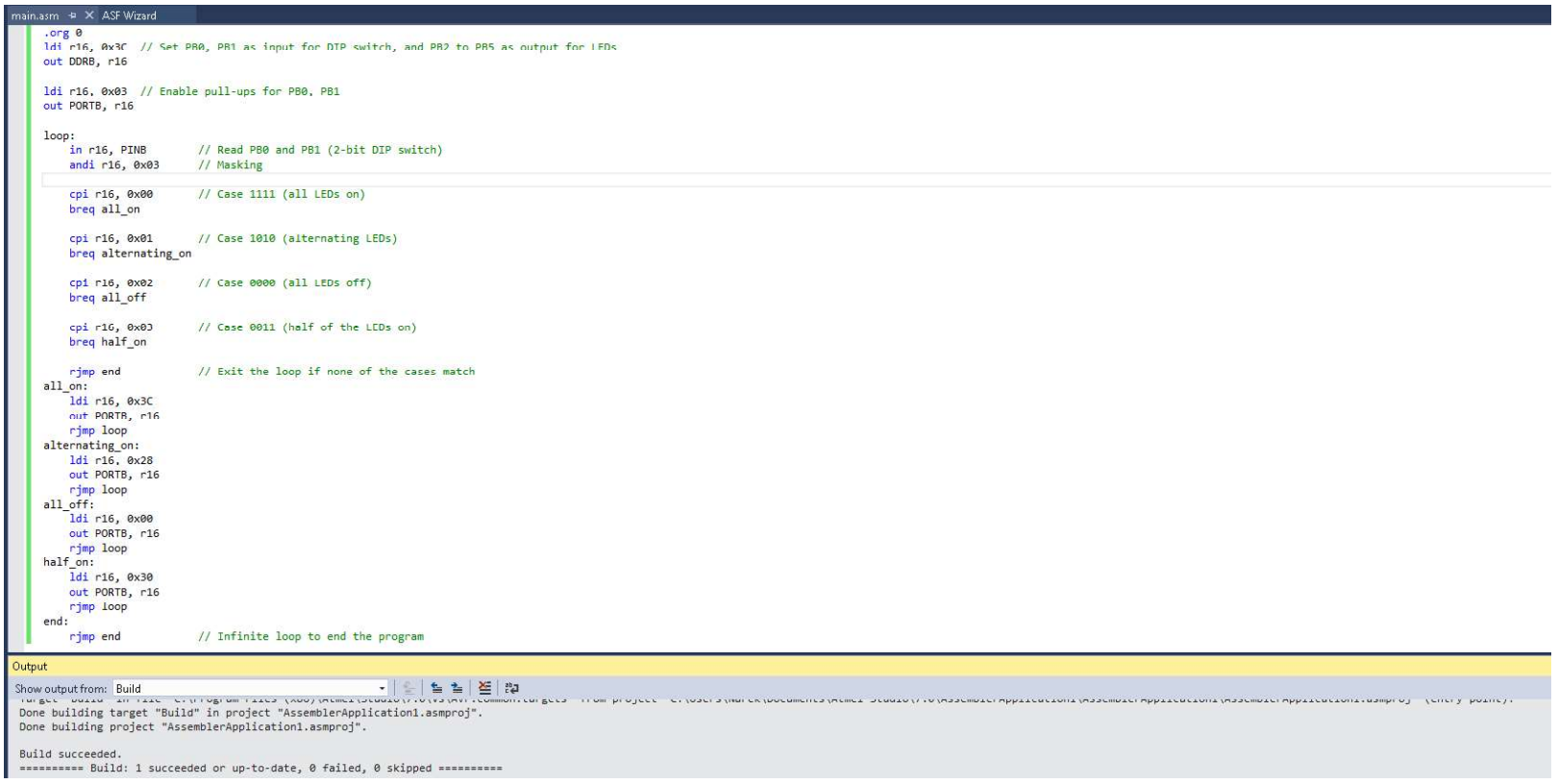
Experiment 3 Assembly Code Screenshot:



Experiment 4 C Code Screenshot:



Experiment 4 Assembly Code Screenshot:



4. Questions and Answers

1. ATmega328P Replacement: A similar microcontroller that could serve to replace the ATmega328P that we used in this lab would be the ATmega328PB. They come from the same family, share the same AVR core architecture and performance, and the PB microcontroller will be what we use for the design assignments in our lecture section. The key differences are that the 328PB has 5 additional I/O pins and is cheaper than the 328P.

2. Internal Pull Ups Not Activated: The code we implemented during this lab would likely not work very reliably if the internal pull ups were not activated. Electrical noise would cause the input pin to read high or low. In the code, the line PORTC = 0xFF sets all PORT C pins high (activates all internal pull ups) but this would cause logical errors in the conditional checks. Changing the conditional to if (!(PINC & 0x01)) checks if PINC is NOT 0x01 (not HIGH, meaning it checks if it's LOW). This will indicate a button press.

3. LED Combinations:

- Solid Red: Idle with no power
- Blinking Red: Short-circuit occurring
- Solid Green: Idle with power
- Double Green: Ready to run once plugged in
- Solid Orange: Busy (in programming state)
- Blinking Orange: Reversed target cable connection
- Blinking Red and Orange: In upgrade mode

4. Explain each line of the following code

LDI R20, 0x75 // Load Imm. hex 0x75 (117) into R20 register
LDI R21, 0x05 // Load Imm. hex 0x05 (5) into R21 register
LDI R22, 0x24 // Load Imm. hex 0x24 (36) into R22 register
ADD R20, R22 // Add contents of R22 to R20, 117 + 36 → R20 = 153
SUB R22, R21 // Subtract contents of R21 from R22, 36 - 5 → R22 = 31
ADD R20, R21 // Add contents of R21 to R20, 153 + 5 → R20 = 158
SUB R22, R20 // Subtract contents of R20 from R22, 31 - 158 → R22 = -127 (161)
ADD R20, R22 // Add contents of R22 to R20, 158 + 161 → R20 = 319 (63)
MOV R20, R21 // Move contents of R21 into R20 → R20 = 5
RJMP DONE // Relative jump to DONE label
ADD R21, R20 // SKIPPED due to RJMP, would add contents of R20 to R21
SUB R21, R22 // SKIPPED due to RJMP, would subtract contents of R22 from R21
DONE: SUB R20, R21 // Subtract contents of R21 from R20, 5 - 5 → R20 = 0
END: RJMP END // Relative jump to END label, loops

- Final decimal value of R20 register = 0

5. Conclusions

Microchip Studio is a useful piece of software used for programming AVR and SAM devices such as the ATmega328p board using both high-level C code and assembly code. Understanding the intricacies of its internal libraries, functions, and variables and how they correspond and interact with the ports and pins on the board is crucial for being able to utilize the various components of your microcontroller. We did not encounter many problems during the lab outside of some of the experiments taking longer than necessary simply because we were unfamiliar with the microcontrollers and Microchip Studio software as it was our first time using them. For example, we initially struggled to get the provided assembly code for experiment 2 to work despite successfully getting the C code to work. We realized this was an error on our part and that we needed to create a new assembly project in Microchip Studio. Moving forward, these types of problems should be minimal as we become more familiar with both the hardware and software.