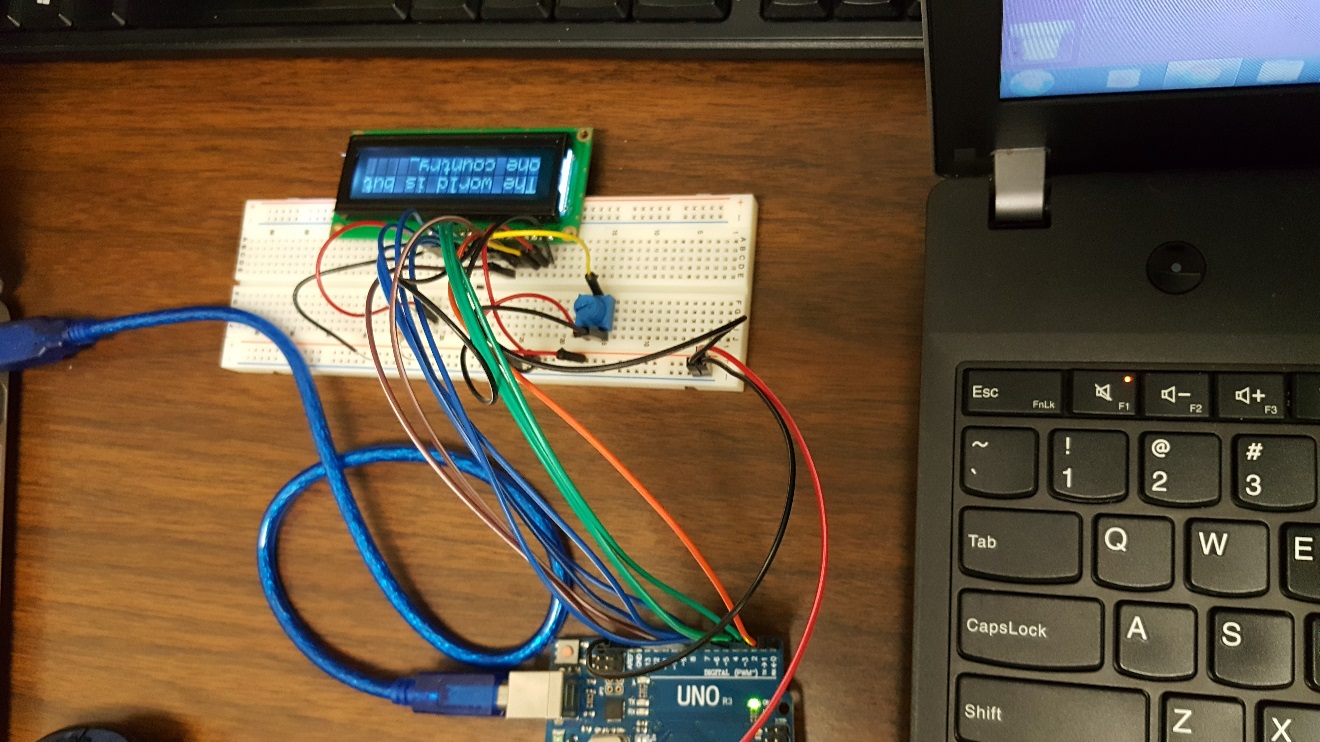
Obviously put better pic here

Group Project 2

LCD Display For Arduino

Casey Sanchez

Evan Coffey

Bryan Newsome

Amanda Hooge |

Table of Contents

[Introduction 2](#_Toc469181654)

[The Test Device 2](#_Toc469181655)

[Development Tools 2](#_Toc469181655)

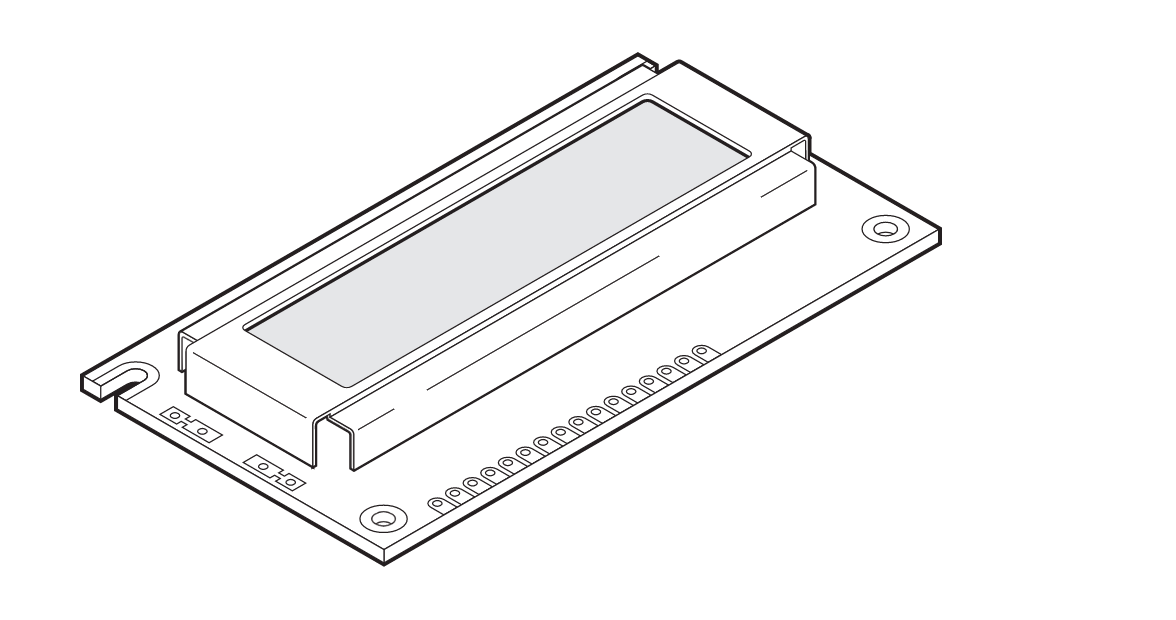
[Your experiment 5](#_Toc469181656)

[Conclusion 6](#_Toc469181657)

# Introduction

For our project, we decided to use the 16x2 LCD display and an Arduino microcontroller. To get it working all that was needed was around twelve jumper wires(what are those called?), a basic breadboard and the microcontroller. We decided to display a simple message on the LCD screen. This proved a very daunting task. After much research and meticulous reading of wiring diagrams, we were successful. (This part is if we get anything else working and word it better.) As soon as we were comfortable with the assembly language for the LCD, we were eager to add more components to out demonstration. Being in the holiday mood, we added blinking LED lights and a nice tune with the buzzer.

The Test Device



Our project utilized an LCD, or Liquid Crystal Display, to display text on a screen. LCDs are widely available and can be more advantageous than their predecessor the LED display. Both technologies debuted in the early '60s and have low costs. The LCD though uses less power than the LED display and normally offers more graphics to print. Because they are easily programmed, LCDs can be found everywhere from TVs to tablets. LCDs even made flat screen TVs possible.

The LCD used for our project has a 16x2 character grid, meaning 16 characters wide, 2 lines deep. Each character is a 5x7 pixel matrix. Other commonly used LCDs vary in size including 40x2, 20x2, 20x4, 16x1, 16x4, 40x4, and 8x1.

What makes programming the LCD so easy is it has its own chip. A lot of LCDs, ours included, are controlled by the HD44780 Hitachi Controller. These chips are all the same for whichever format display so it is up to the programmer to keep in mind the display's size. Memory is contained in 80 bytes of Display Data Random Access Memory, arranged in 2 lines of 40 addresses. Each memory location controls the corresponding location on the display even when there is no display there, as is the case for a 16x2 display. On the other hand, the larger 40x4 displays contain 2 chips to house twice the data.

40x2 Controller Memory::LCD-controller-memory.png 16x2 Display Memory:

:LCD-16x2-addressing.png

The LCDs come with either 14 or 16 interface pins and an LED backlight. The pins are ordered as follows:

Vss Ground

Vcc 5v power supply

V0 contrast adjustment pin:

Receives analog voltage input to adjust the display's contrast.

RS register select:

command register (0) or data register (1)

RW read/write:

write (0) or read (1), like check for busy flag

r/w to ground is permanently in write

E enable:

allows lcd to latch data at the pins when a high-to-pulse occurs

DB0-DB7

8-bit parallel data port

or operation in 4 bit (db4-db7)

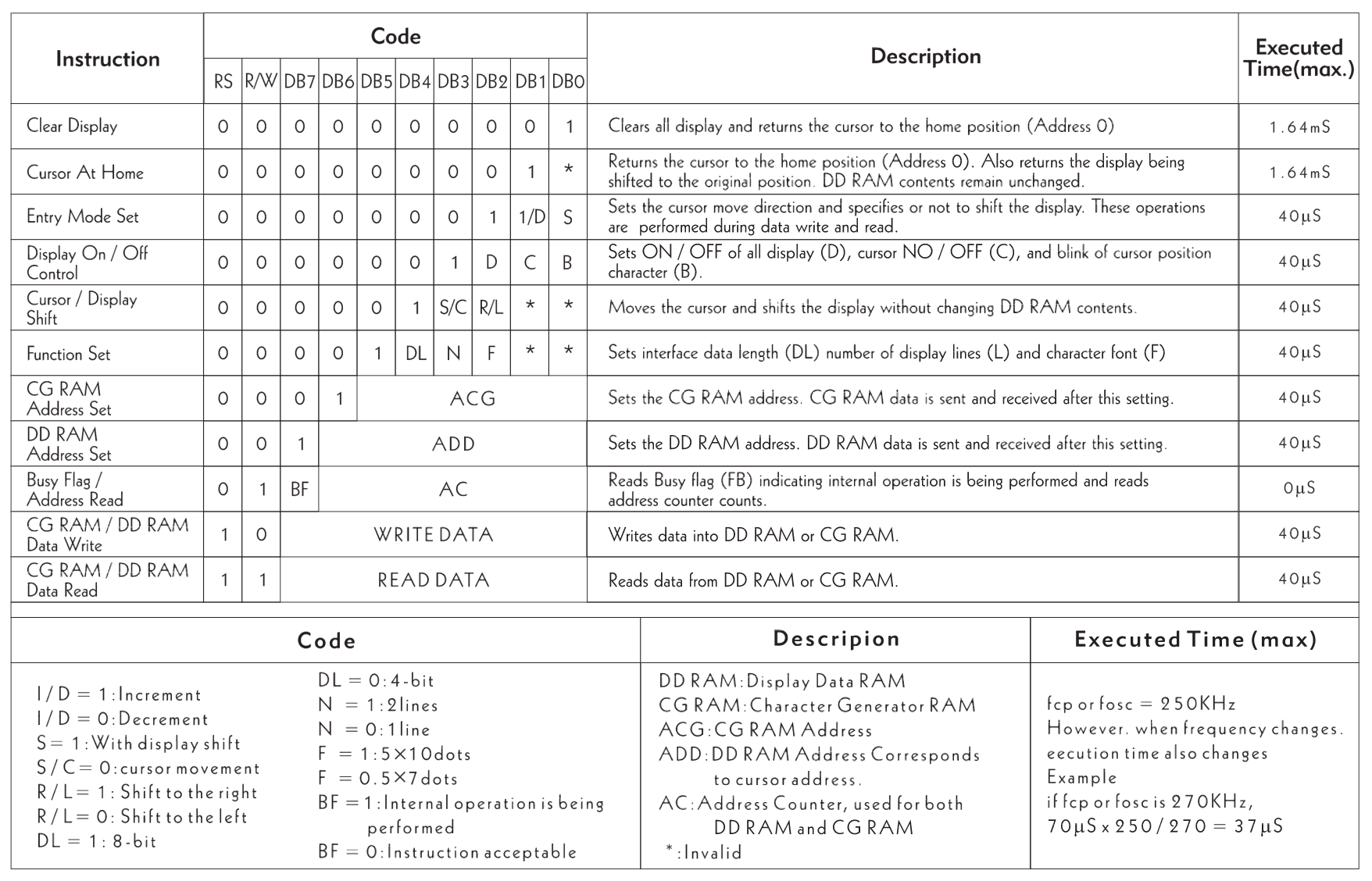
anode

cathode

for led backlight

The data pins can write to information sent with ASCII codes to the screen. To save I/O pins, the LCDs may be operated in 4-bit mode, instead of 8-bit mode. In either mode, the programmer can also choose to use time delays or utilize the busy flag to take the polling approach to delays.

The following table lists the commands available to interface with the LCD:



For our code, we initialized the 5x7 display, set the cursor and turned on the display, cleared the display, set the cursor to move right, and sent a message. In doing so, we utilized two functions one for sending commands, the other data. Our instructions were sent to the functions in R24. Each function either set the RS bit to send data or cleared the bit to send commands. Then a high to low pulse was sent through the enable pin. Of course, we also had to set up a delay function so each command would have enough time to finish before starting the next.

Here is a cheat sheet of the commands:

|  |  |
| --- | --- |
| **(Hex)** | **Command** |
| 1 | Clear display screen |
| 2 | Return home |
| 4 | Decrement cursor (shift cursor to left) |
| 6 | Increment cursor (shift cursor to right) |
| 5 | Shift display right |
| 7 | Shift display left |
| 8 | Display off, cursor off |
| A | Display off, cursor on |
| C | Display on, cursor off |
| E | Display on, cursor blinking |
| F | Display on, cursor blinking |
| 10 | Shift cursor position to left |
| 14 | Shift cursor position to right |
| 18 | Shift the entire display to the left |
| 1C | Shift the entire display to the right |
| 80 | Force cursor to beginning of 1st line |
| C0 | Force cursor to beginning of 2nd line |
| 28 | Initiate 2 lines of 5x7 matrix (D4-D7, 4-bit) |
| 38 | Initiate 2 lines of 5x7 matrix (D0-D7, 8-bit) |

Once we had working code up and running (in assembly!), we played around with scrolling the text. Given more time, we plan on making our own characters to write to the LCD.

# Development Tools

When we began our experiment, we used the Arduino IDE 1.6.13. The software is open-source, written in Java, and works on Windows, Mac OS X, and Linux platforms. It can load code in various languages onto any Arduino microcontroller. It was very easy to install and get started right away. It even came pre-packaged with some helpful example code. Eventually, we stopped using the IDE and started loading code onto the Arduino via the command line on Amanda’s Mac machine.

\*\*\*The disassembler we used to get assembly language out of our C code is [\_\_\_\_\_\_\_\_\_\_\_\_\_]. Blah blah blah blah blah.

We communicated through email and collaborated on Github. We also used Google to find various pieces of documentation which helped us complete the assignment. No other development tools were needed.

# Your experiment

Our objective from the beginning was simply to get the 16x2 LCD to display some text. Given that there was no documentation for the LCD and that none of our group members felt confident writing the assembly language from scratch, we decided to think of other options.

We ended up trying a few, but before we could, we had to properly connect the LCD and our Arduino to a mutual breadboard. A google search or two later, and we had documentation which assisted us with this. We leaned in, put the wires in their place, and the screen lit up. “Hello world,” at last!

The first code we loaded onto our microcontroller was C code that we ran through the Arduino IDE. We knew that we’d eventually be turning in assembly language, but we wanted to make sure it worked before we got our hands dirty with the tough stuff. It worked like a charm, so we packed up for the day.

The above took place a few days after the group project was assigned. A week or so later, after some communication through email, chatting during lab time, and some internet research, we reconvened to try to get the LCD to work with assembly language code.

This is where things got hairy. As I mentioned before, we wanted to find assembly language code and modify it for our specific needs, but at this point, we had a hard time finding working code anywhere on the internet. We got very close with one attempt, finding some code that got the LCD to light up. Unfortunately, however, we couldn’t successfully modify it to display text. As much as we tried, for some reason, the only working assembly code we could produce was our disassembled C code.

Eventually, we decided that our best bet was simply to dig into the disassembled C code, make sense of it, trim the fat to make it more readable, and submit that. After breaking once more to complete our assigned sections of the lab report and look over the assembly language code, we came together one last time to piece the report together and agree on the assembly language code we wanted to turn in.

# Conclusion

We learned that transmitting data from a computer to a device is as simple as buying some wires, a breadboard, and an Arduino, and sitting down to write (or find!) some code. We only worked with one device, but it’s clear to see that one could let his or her imagination run wild with all the gizmos available on the market today.

Our assignment was to submit code written in assembly language, but that’s not at all a requirement of using these gadgets. Anyone with a basic understanding of programming and some experience with any popular high-level language could get started with microcontrollers.