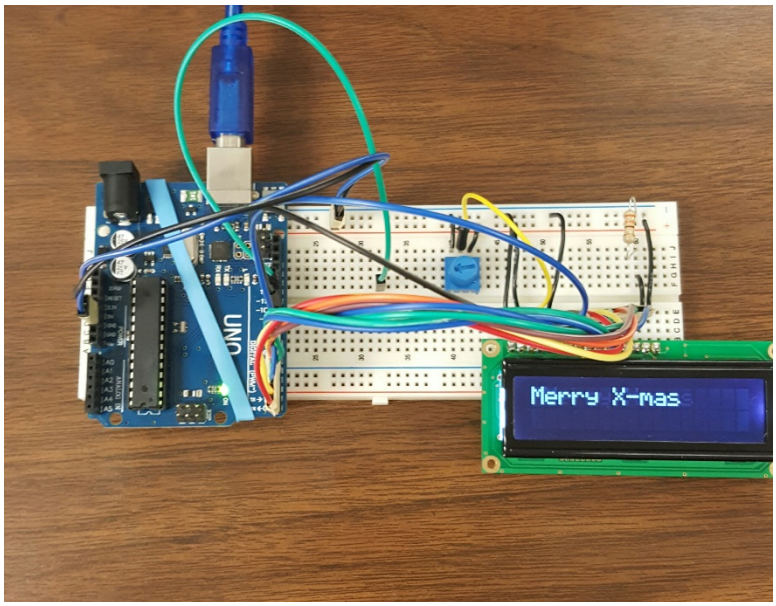

Group Project 2
LCD Display for Arduino Uno

Casey Sanchez
Evan Coffey
Bryan Newsome
Amanda Hooge

Table of Contents

Introduction

For our project, we chose to use the 16x2 LCD display and an Arduino microcontroller. To get it working all that was needed was around twelve jumper wires, a 10K potentiometer, 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.

The Microcontroller Platform

The Arduino Uno board is based on the ATmega328P processor. These boards are widely available, and can be purchased directly from the Arduino website or many other stores. The Uno features 14 I/O pins, 6 analog inputs, a 16MHz crystal, a USB port, a reset button, and is capable of outputting 3V and 5V signals. The onboard RISC-based ATmega328P processor contains an 8-bit AVR CPU, 32 kB flash memory, and is capable of internal/external interrupts.

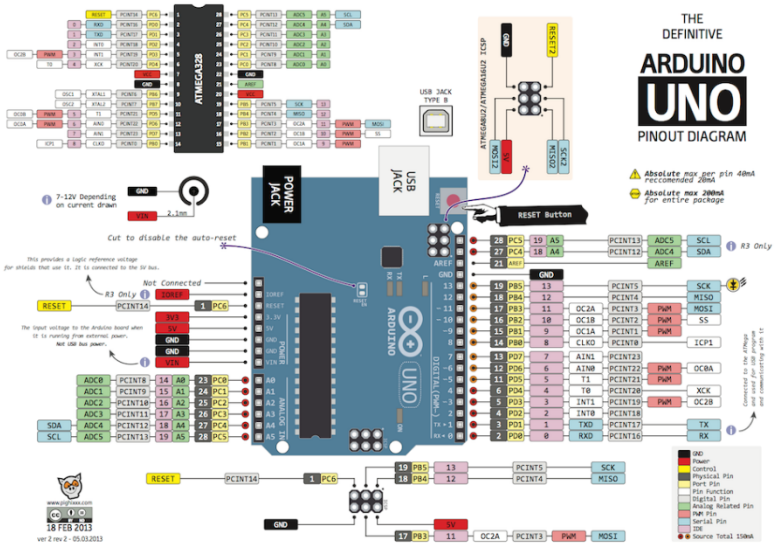

FIGURE 2: ARDUINO DIAGRAM

The Test Device

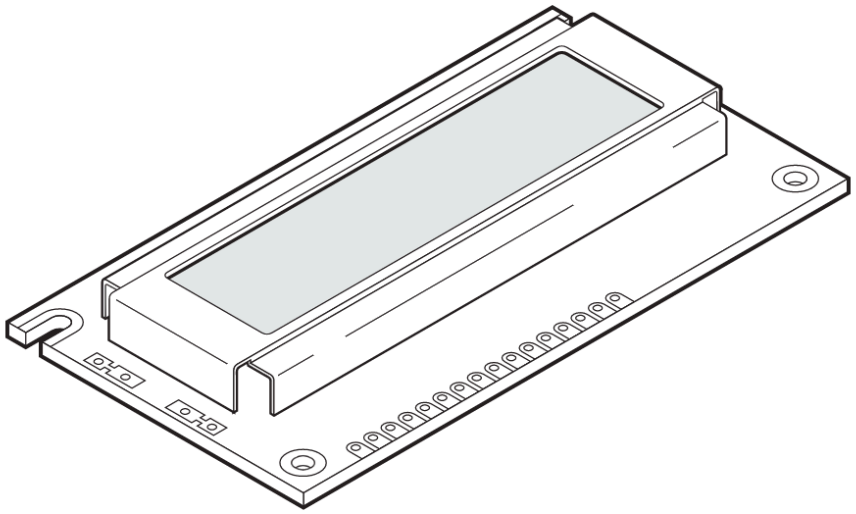

FIGURE 2: LCD DISPLAY

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.

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53	54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

FIGURE 3: 40X2 CONTROLLER MEMORY

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

FIGURE 4: 16X2 DISPLAY MEMORY

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

Vo 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

Information sent with ASCII codes can be written to the screen via the data pins. 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:

Instruction	Code										Description	Executed Time(max.)
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears all display and returns the cursor to the home position (Address 0)	1.64mS
Cursor At Home	0	0	0	0	0	0	0	0	1	*	Returns the cursor to the home position (Address 0). Also returns the display being shifted to the original position. DD RAM contents remain unchanged.	1.64mS
Entry Mode Set	0	0	0	0	0	0	0	1	1/D	S	Sets the cursor move direction and specifies or not to shift the display. These operations are performed during data write and read.	40µS
Display On / Off Control	0	0	0	0	0	0	1	D	C	B	Sets ON / OFF of all display (D), cursor NO / OFF (C), and blink of cursor position character (B).	40µS
Cursor / Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves the cursor and shifts the display without changing DD RAM contents.	40µS
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL) number of display lines (L) and character font (F)	40µS
CG RAM Address Set	0	0	0	1	ACG						Sets the CG RAM address. CG RAM data is sent and received after this setting.	40µS
DD RAM Address Set	0	0	1	ADD							Sets the DD RAM address. DD RAM data is sent and received after this setting.	40µS
Busy Flag / Address Read	0	1	BF	AC							Reads Busy flag (FB) indicating internal operation is being performed and reads address counter counts.	0µS
CG RAM / DD RAM Data Write	1	0	WRITE DATA								Writes data into DD RAM or CG RAM.	40µS
CG RAM / DD RAM Data Read	1	1	READ DATA								Reads data from DD RAM or CG RAM.	40µS

Code		Descripion	Executed Time (max)
I/D = 1: Increment I/D = 0: Decrement S = 1: With display shift S/C = 0: cursor movement R/L = 1: Shift to the right R/L = 0: Shift to the left DL = 1: 8-bit	DL = 0: 4-bit N = 1: 2lines N = 0: 1line F = 1: 5×10dots F = 0. 5×7dots BF = 1: Internal operation is being performed BF = 0: Instruction acceptable	DD RAM: Display Data RAM CG RAM: Character Generator RAM ACG: CG RAM Address ADD: DD RAM Address Corresponds to cursor address. AC: Address Counter, used for both DD RAM and CG RAM *: Invalid	fcp or fosc = 250KHz However. when frequency changes. eecution time also changes Example if fcp or fosc is 270KHz, 70µS x 250 / 270 = 37µS

FIGURE 5: LCD COMMANDS

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.

Documentation for the LCD can be found at the link below:

http://www.picaxe.com/docs/led008.pdf

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.

To disassemble our C code, we used the command "avr-objdump". This command was accessible as soon as we installed it via the command line. We also 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 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!

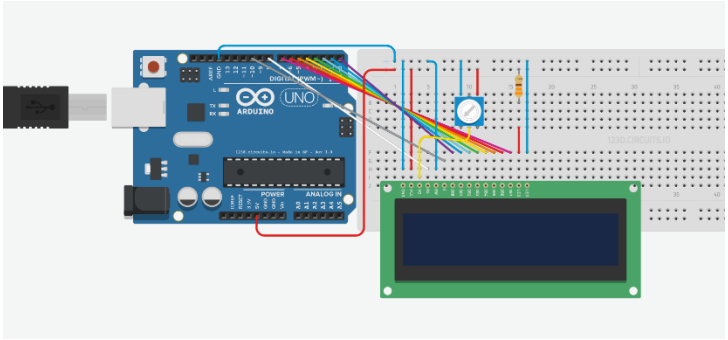

FIGURE 6: CIRCUITS LAYOUT

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.

Contributions

Casey:

- Final formatting of project report
- Report sections Title page, Table of contents, Introduction, Contributions
- Wiring LCD to Arduino
- Project code
- Research

Bryan:

- The microcontroller platform portion of report
- Wiring of LCD to Arduino
- Project code
- Research

Amanda:

- The test device portion of project report
- Wiring of LCD to Arduino
- Disassembly of C code
- Reworking of C code to assembly
- Majority of project code
- Research

Evan:

- Development tools portion of project report
- Experiment portion of project report
- Conclusion portion of project report
- Research
- Wiring of LCD to Arduino
- Project code

```
; configuration for LCD
#include <avr/io.h>
#include "../include/AVRSpecialRegs.inc"


#define LCD_DPRT        _PORTD
#define LCD_DDDR        _DDRD

#define LCD_CPRT        _PORTB
#define LCD_CDDR        _DDRB

; bit for register select
#define LCD_RS          0

; bit for enable
#define LCD_EN          2
```

```
; LCD for AVR
#include "config.inc"

        .section .text
        .global     main
        .org        0x0000

        ; set up the stack
            ldi     r28, (RAMEND & 0x00ff)
            ldi     r29, (RAMEND >> 8)
            out     _SPH, r29
            out     _SPL, r28
            eor     r1, r1
            out     _SREG, r1
            call    main
            jmp     exit

delay:
            sbiw    r24, 0x00
            breq    3f
            ldi     r18, 0x05
        2:
            dec     r18
            brne    2b
            nop
            sbiw    r24, 0x01
            rjmp    delay
        3:
            ret

commandwrite:
            out     LCD_DPRT, r24           ; load the command
            cbi     LCD_CPRT, LCD_RS        ; turn on commanding

            sbi     LCD_CPRT, LCD_EN        ; enable = 1
            ldi     r24, 0x01
            ldi     r25, 0x00
            call    delay
            cbi     LCD_CPRT, LCD_EN        ; enable = 0 for H->L pulse

            ldi     r24, 0x64               ; 100
            ldi     r25, 0x00
            jmp     delay

datawrite:
            out     LCD_DPRT, r24           ; load the data
            sbi     LCD_CPRT, LCD_RS        ; turn on data sending

            sbi     LCD_CPRT, LCD_EN        ; enable = 1
            ldi     r24, 0x01
            ldi     r25, 0x00
            call    delay
            cbi     LCD_CPRT, LCD_EN        ; enable = 0 for H->L pulse

            ldi     r24, 0x64               ; 100
            ldi     r25, 0x00
            jmp     delay

message:
            ldi     R24, 0x4d               ;'M'
            call    datawrite
            ldi     R24, 0x65               ;'e'
            call    datawrite
            ldi     R24, 0x72               ;'r'
            call    datawrite
            ldi     R24, 0x72               ;'r'
            call    datawrite
            ldi     R24, 0x79               ;'y'
            call    datawrite
            ldi     R24, 0x20               ;' '
            call    datawrite
            ldi     R24, 0x58               ;'X'
            call    datawrite
```

```
            ldi     R24, 0x2d               ;'-'
            call    datawrite
            ldi     R24, 0x6d               ;'m'
            call    datawrite
            ldi     R24, 0x61               ;'a'
            call    datawrite
            ldi     R24, 0x73               ;'s'
            call    datawrite
            ret

main:
            ldi     R24,0xFF
            out     LCD_DDDR, R24           ; turning on output mode
            out     LCD_CDDR, R24           ; turning on output mode
            cbi     LCD_CPRT,LCD_EN         ; enable = 0

            ldi     r24, 0xD0               ; 208
            ldi     r25, 0x07               ; 7
            call    delay
            ldi     r24,0x38                ; command to init LCD to 5x7 matrix

            CALL    commandwrite

            ldi     r24, 0xD0               ; 208
            ldi     r25, 0x07               ; 7
            call    delay
            ldi     r24,0x0E                ; command to turn display on, & cursor b
link
            CALL    commandwrite

start:
            ldi     R24,0x01                ; clear display
            CALL    commandwrite

            ldi     r24, 0xD0               ; 208
            ldi     r25, 0x07               ; 7
            call    delay
            ldi     r24, 0x06               ; command to shift cursor right
            call    commandwrite

            call    message

            ldi     r26, 7                  ; seven spaces before next message
spaces:
            ldi     R24, 0x20               ;' '
            call    datawrite
            dec     r26
            cpi     r26, 0
            brne    spaces

            call    message

scroll:
            ldi     r24, 0xFF
            ldi     r25, 0xFF
            call    delay
            ldi     r24, 0xFF
            ldi     r25, 0xFF
            call    delay
            ldi     r24, 0xFF
            ldi     r25, 0xFF
            call    delay
            ldi     r24, 0xFF
            ldi     r25, 0xFF
            call    delay

            ldi     R24,0x18                ; command to move display to the left
            call    commandwrite

            jmp    scroll

exit:
            rjmp exit
```