# ECEN 4330

## Semester Project

Written by: Justin Pachl

5-4-18

University of Nebraska Lincoln
Department of Electrical and Computer Engineering (ECE)

# Abstract

The objective of this project was to design and build a computer system from scratch including the software that runs on the system. This report covers the tools used to create the project, a discussion on the hardware used, a discussion on the software composed, and a discussion on issues encountered while completing the project. Included in the appendix is supplementary technical documents such as schematic drawing, code listings, and various images of the different development stages of the project that is referenced throughout the report.

FIGURE 1 - CREATOR AND DEVELOPER

## Table of Contents

# Introduction

The Objective for this project can be separated into two parts. The first part of the objective was to design and construct a computer system with the following requirements,

- Use an 8051 Microprocessor
- Implement 64k Code memory
- Implement 64k Data memory
- Interface with an LCD
- Interface with a 7 Segment display
- Interface with a keypad
- Interface with an Analog to Digital Converter(ADC)
- Interface with a Real Time Clock(RTC)

A block diagram of the system can be seen on the following page.

The second part of the project was to develop software that would run on the computer system that could perform the following functions,

- Display the contents of RAM(Dump)
- Search for a specific byte value in RAM(Search)
- Edit the contents of RAM(Edit)
- Fill RAM with blocks of data(Fill)
- Move blocks of RAM from one location to another(Move)
- Display the temperature read from the ADC
- Display the time from the RTC

The project described by this report requires understanding in different areas of computer engineering mostly from, software design, circuit analysis, circuit design, and PCB design. This report will cover the following subjects as they pertain to this project. First it will discuss the different resources and tools used to design, construct, and debug the project. Second the report will describe the hardware implementation of the project. Following the hardware discussion, the report will cover the software implementation of the project. And finally, the report will wrap up with issues encountered during the completion of the project as well as personal opinions of the project.

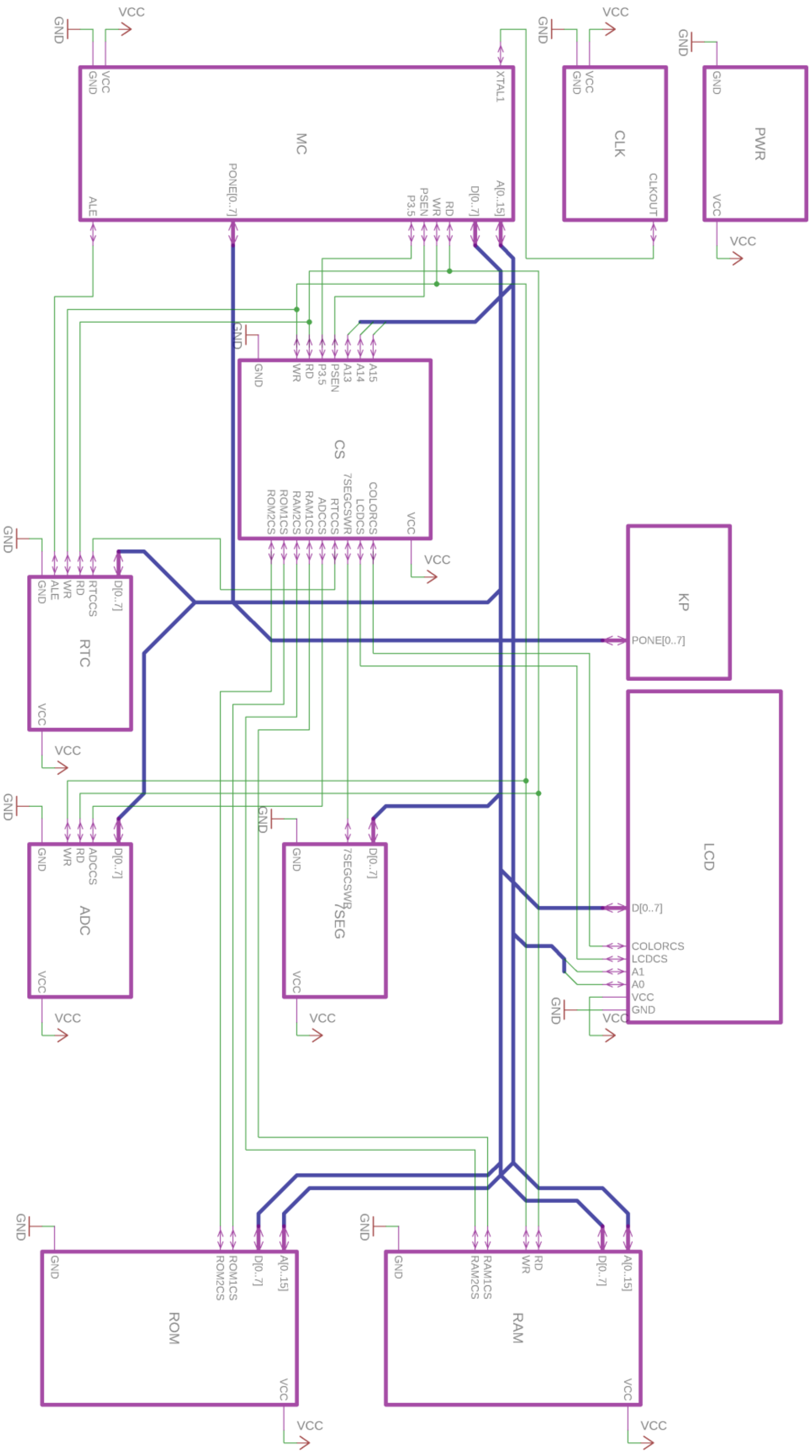


Figure 2 - System Block Diagram

# Background

This section of the report will discuss different resources and tools used to complete this project. The universities IEEE chapter supplied all the components for this project. All provided parts used in the project along with the budget for the project are listed in the BOM which can be found in the appendix section of this report. JLCPCB was chosen to manufacture the PCB for this project. They were chosen because their cost was the lowest amongst all considered manufacturers and they had good reviews amongst peers. A Comparison table of different PCB manufacturers considered for this project can be seen below ordered by price.

| MANUFACTURER | PRICE |
|---|---|
| OSH PARK | **$377.35** |
| PCBWAY | **$79** |
| JLCPCB | **$37.12** |

TABLE 1 - PCB PRICE COMPARISON

To design, construct, and debug the system, multiple software and hardware tools were utilized. The following software tools each played a part in the production of this project. Autodesk Eagle was used for schematic and PCB design. Visual Studio Code was used as the IDE for C, Assembly, and Verilog code composition. Mini Programmer Pro Software was used to transfer hex files onto the ROM chips in the system and the Small Device C Compiler(SDCC) was used to generate the hex files from C code. IspLever was used to generate the jedec file that was used to program the PAL device in the system. And finally, the Saleae Digital Analyzer Software was used to debug the circuit during the early stages of development.

Along with the software tools multiple hardware tools facilitated the completion of this project. A Mini Programmer Pro was used to write software to the physical ROM chips. A Dataman Programmer was used to program the PAL device. A Solder Iron was used to populate the PCB with the different components. A Saleae digital analyzer was used to debug circuit logic during the early stages of development. And lastly a Digital Multimeter was used to do voltage level testing, current checks, and Continuity testing on the PCB.

# Hardware Discussion

As stated in the introduction, the first part of this project was to design and construct a computer system with the following requirements met,

- Use an 8051 Microprocessor
- Implement 64k Code memory
- Implement 64k Data memory
- Interface with an LCD
- Interface with a 7 Segment display
- Interface with a keypad
- Interface with an Analog to Digital Converter(ADC)
- Interface with a Real Time Clock(RTC)

This section will first describe the design process conducted when creating this project along with the PCB design. Following the design, each component listed above will be discussed individually including what specific chip or module was used, the functionality the part provides the system, and how each part interfaces with the microprocessor.

## Design

The first step conducted when designing this system was deciding how each part would interface with the microprocessor. This included creating a memory map of the system with all the parts included. A graphic of the final memory map for the system can be seen below. The equations below each block correspond to the state of the control lines that activate each map.

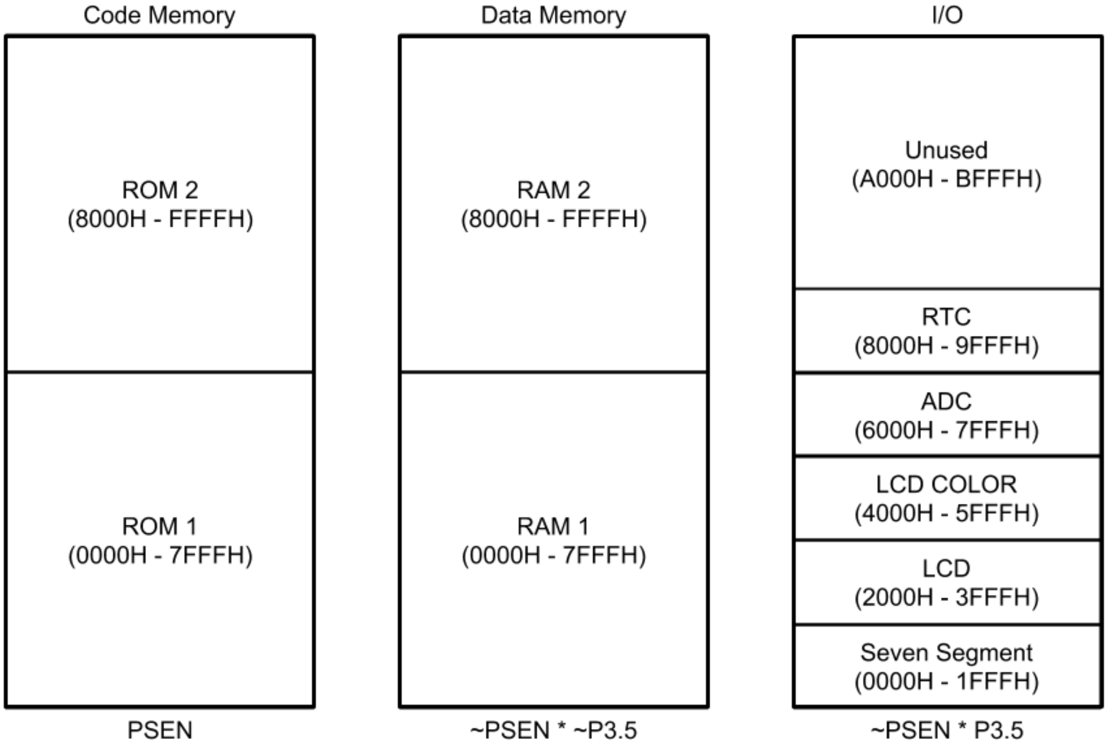


**FIGURE 3 - SYSTEM MEMORY MAP**

It should be noted that the 8051 microprocessor used in this project does not actually have three separate maps for Code memory, Data memory and I/O. By default, the 8051 only has space for Code and Data memory with PSEN controlling which is accessed however, a third map was created artificially by using one of the togglable external pins on the microprocessor as an IO/M- pin. While it does make the software implementation for the system more complicated, it allows all components to be fit into the system.

After the memory map was complete the next step was to design the chip select that would be used in the system. This was done using an address map to decide which address pins of the microprocessor would be used for selecting different components of the device. The address map for each memory map block can be seen below as well as the chip select equations generated from these maps used for each device.

| DEVICE | A15 | A14 | A13 | A12 | A11 – A0 (HEX) |
|---|---|---|---|---|---|
| **ROM 1** | 0 | 0 | 0 | 0 | 000 |
| | 0 | 1 | 1 | 1 | FFF |
| **ROM 2** | 1 | 0 | 0 | 0 | 000 |
| | 1 | 1 | 1 | 1 | FFF |

Table 2 - System code memory address map

| DEVICE | A15 | A14 | A13 | A12 | A11 – A0 (HEX) |
|---|---|---|---|---|---|
| **RAM 1** | 0 | 0 | 0 | 0 | 000 |
| | 0 | 1 | 1 | 1 | FFF |
| **RAM 2** | 1 | 0 | 0 | 0 | 000 |
| | 1 | 1 | 1 | 1 | FFF |

Table 3 - System data memory address map

| DEVICE | A15 | A14 | A13 | A12 | A11 – A0 (HEX) |
|---|---|---|---|---|---|
| **SEVEN SEGMENT DISPLAY** | 0 | 0 | 0 | 0 | 000 |
| | 0 | 0 | 0 | 1 | FFF |
| **LCD** | 0 | 0 | 1 | 0 | 000 |
| | 0 | 0 | 1 | 1 | FFF |
| **LCD COLOR** | 0 | 1 | 0 | 0 | 000 |
| | 0 | 1 | 0 | 1 | FFF |
| **RTC** | 0 | 1 | 1 | 0 | 000 |
| | 0 | 1 | 1 | 1 | FFF |
| **ADC** | 1 | 0 | 0 | 0 | 000 |
| | 1 | 0 | 0 | 1 | FFF |

Table 4 - System I/O address map

$$\begin{aligned}
ROM1CS &= PSEN + A15 \\
ROM2CS &= PSEN + \sim A15 \\
RAM1CS &= \sim PSEN + P35 + A15 \\
RAM2CS &= \sim PSEN + P35 + \sim A15 \\
SEGCSWR &= \sim WR \,\&\, P35 \,\&\, PSEN \,\&\, \sim A15 \,\&\, \sim A14 \,\&\, \sim A13 \\
LCDCS &= P35 \,\&\, PSEN \,\&\, \sim A15 \,\&\, \sim A14 \,\&\, A13 \,\&\, (\sim RD + \sim WR) \\
LCDCOLORCS &= \sim WR \,\&\, P35 \,\&\, PSEN \,\&\, \sim A15 \,\&\, A14 \,\&\, \sim A13 \\
ADCCS &= \sim P35 + \sim PSEN + \sim A15 + \ A14 + \sim A13 \\
RTCCS &= \sim P35 + \sim PSEN + \sim A15 + \ A14 + A
\end{aligned}$$

Equation 1 - Chip select equations

Once the chip select for the system was designed the final step of the system design was to draw the schematic for the overall system. Once the schematic is drawn, implementation of the system can be done. Drawing of the schematic involved reading the datasheets for each of the components purchased for this system. This was to make sure each component was correctly interfaced with regarding, logic levels, current requirements, pin layout, and timing diagrams. The datasheets were also used to generate schematic blocks and PCB footprints if none were available in the pre-built Eagle libraries. The full final schematic for the system can be found in the appendix section of this report.

## PCB Design

The Printed Circuit Board (PCB) for this project was designed in Eagle. The board is a two-layer board that was manufactured by JLCPCB. The board was configured with the restrictions provided by Sparkfun Electronics listed in the table below.

| *RESTRICTION* | *VALUE* |
|---|---|
| *Trace Width* | 10 Mil |
| *Trace Spacing* | 8 Mil |
| *Isolate* | 12 Mil |
| *Restring* | 12 Mil |

**TABLE 5 - SPARKFUN DRC SETTINGS**

The board was designed with looks in mind which is why it is so large 251mmx194mm despite the number of components populating the board. This gave plenty of space between components, allowed for larger traces, and allowed for larger trace spacing. Traces were kept as one line where possible but, in cases where traces must overlap vertical traces were placed on the top layer and horizontal traces were placed on the bottom layer. This also helped make the board neat and easy to debug. A picture of the final board design file (with no solder mask) as well as a picture of the unpopulated PCB can be seen in the appendix section of this report.

## Microprocessor

The microprocessor is a unique hardware piece to the system as it is used to control everything for it is the sole computational device used in the project. The microprocessor required for this project is one from the 8051 family of microprocessors. While these devices are traditionally used as low power microcontrollers complete with internal RAM, internal ROM, and other various microcontroller components it can be used as a microprocessor as proven by this project. Knowing this all further references to the chip will be made assuming it is using external memory.

The actual chip used for the microprocessor in this project is the *AT89C55HWD* manufactured by Atmel. An image of the chip as well as the pinout from the datasheet can be seen below.

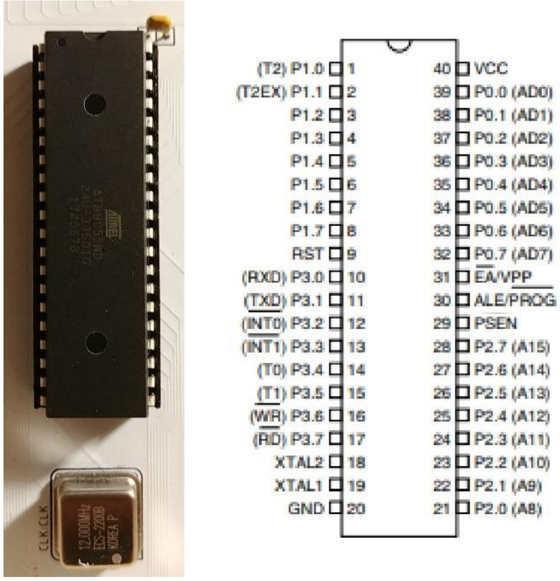


**FIGURE 4 - MICROPROCESSOR CHIP AND PINOUT**

The chip used comes in a 40-dip package. The microprocessor is driven by an ECS-2200 Crystal oscillator at 12 MHz which according to the datasheet for the processor is the ideal clock frequency. The processor has 16 pins used for addressing 8 of which are also used for data lines and must be demultiplexed using a *74HCT573* chip. The processor has many control lines including the ALE which is used for demultiplexing the address and data lines. The PSEN is another control line which is used to separate Code and Data memory maps. Along with the PSEN line, pin 3.5 which is traditionally used for the external T1 interrupt of the processor was used to create an additional memory map section and is used as an IO/M- pin. The final control lines for the processor are the RD- and WR- lines are used to indicate the intentions of the microprocessor to external devices.

Other pins of note on the device include the EA pin. This pin decides whether the processor should use internal or external memory and is grounded to use external memory for this project. The reset pin which can be used to invoke a hard reset of the processor and is attached to the specified reset circuitry provided by the datasheet. An image of the reset circuitry can be seen below.

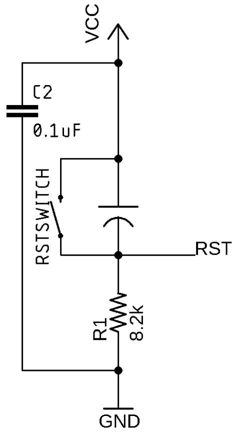


**FIGURE 5 - RESET CIRCUIT**

Lastly, are the interrupt and serial pins of the device. These pins on the microprocessor are not utilized by this system but, were exposed for use in future projects. The schematic describing the implementation of the microprocessor can be found in the appendix section of the report labeled Microprocessor.

## Code Memory

The Code memory as specified by the requirements listed at beginning of the hardware discussion had to have a capacity of 64k. To accomplish this two 32k EEPROM chips were used. The specific chip used was the *AT28C256* chip manufactured by Microchip Technology. These chips are used to hold the software for the system as they do not erase on power down. An image of the chip as well as the pinout from the datasheet can be seen below.

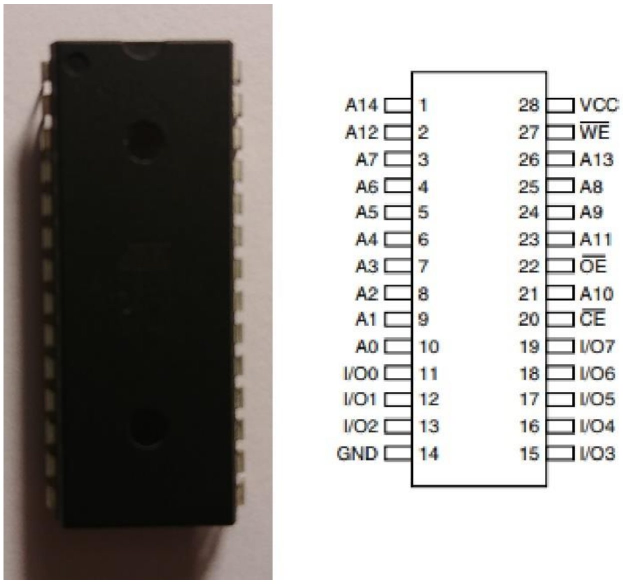


**FIGURE 6 - EEPROM CHIP AND PINOUT**

This chip was provided in a 28-dip package and has an access time of 150 nanoseconds which far exceeds the requirements for this system. To interface properly with the microprocessor the chips were connected in the following way. Both EEPROMS had their address lines (A0 - A14) connected to the corresponding address lines of the microprocessor (A0 - A14). The I/O lines (I/O0 - I/O7) were connected to the data lines of the microprocessor (D0 - D7). The CE- and OE- of the EEPROMS were attached to their corresponding chip select lines discussed in the design section of this report labeled ROM1CS and ROM2CS. Finally, the WE- was tied to logic high to prevent unpredicted behavior as it is never used by the microprocessor. The schematic describing the implementation of the Code memory for the system can be found in the appendix section of this report labeled ROM.

## Data Memory

The Data memory like the code memory needed to have a capacity of 64k. This was done using two 32K Static RAM chips. The chip used was the *HM62256P-8* manufactured by Hitachi. These chips are used as temporary storage used by the microprocessor and are erased upon power down. An image of the chip as well as the pinout from the datasheet can be seen below.

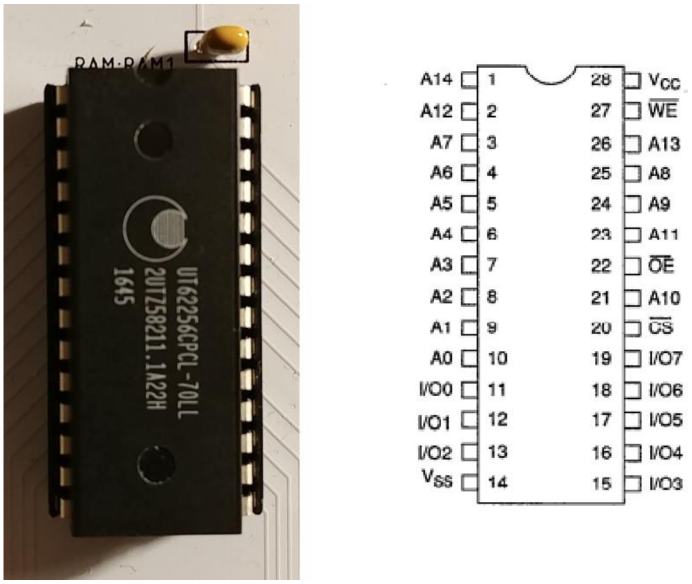
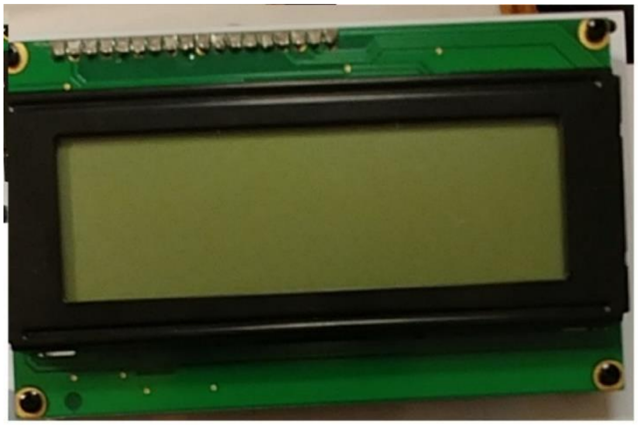


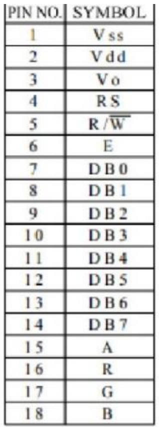
**FIGURE 7 - RAM CHIP AND PINOUT**

This chip was provided in a 28-dip package and has an access time 85 nanoseconds which far exceeds the requirements for this system. To interface properly with the microcontroller the chips were implemented in the system in the following way. The address lines of the RAM (A0 - A14) were attached to the corresponding address lines of the microprocessor (A0 - A14). The I/O lines (I/O0 - I/O7) of the RAM were connected to the data lines of the microprocessor (D0 - D7). The CS- of each RAM chip was connected to the corresponding chip select lines shown in the design section of this report labeled RAM1CS and RAM2CS. Finally, the OE- was connected to the RD- control line of the microprocessor while the WE- was connected to the WR- control line. The schematic describing the implementation of these chips can be found in the appendix section of this report labeled RAM.

## LCD

One of the requirements of this project was to interface with an LCD. The specific LCD used was the *WH2004A-CFH-JT* manufactured by Adafruit Industries LLC. The LCD is used by the microprocessor to display information to the user. An image of the LCD as well as the pinout from the datasheet can be seen below.





| PIN NO. | SYMBOL |
|---|---|
| 1 | Vss |
| 2 | Vdd |
| 3 | Vo |
| 4 | RS |
| 5 | R/$\overline{W}$ |
| 6 | E |
| 7 | DB0 |
| 8 | DB1 |
| 9 | DB2 |
| 10 | DB3 |
| 11 | DB4 |
| 12 | DB5 |
| 13 | DB6 |
| 14 | DB7 |
| 15 | A |
| 16 | R |
| 17 | G |
| 18 | B |

**FIGURE 8 - LCD MODULE AND PINOUT**

This LCD is an 18 pin device with a display capacity of 80 characters. This LCD unlike many others also has an RGB backlight which can be controlled by pins 14 - 17 on the device otherwise it follows standard LCD setup and can be interfaced using a *HD48077U* chip. For this project no LCD control chip was used and instead the LCD was interfaced directly with the microprocessor in the following way. The Data lines of the LCD (DB0 - DB7) were connected to the data lines of the microprocessor (D0 - D7). The RS and R/W- lines of the LCD were connected to the address lines A0 and A1 respectively. This was done to meet a data hold requirement specified by the timing diagrams in the LCD datasheet. Finally, the enable pins is attached to the chip select pin for the device described in the design section of this report labeled LCDCS.

Additionally, the Vo pin of the LCD which controls the contrast of the LCD was attached to a 10K potentiometer so that the user could freely adjust the contrast of the display. The contrast circuitry can be seen below.

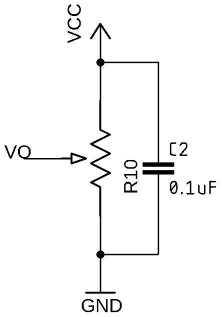


**FIGURE 9 - CONTRAST CIRCUIT**

The backlight for the LCD is also controlled by the microprocessor. This was done by making an output port using a *74HCT573* chip. This port connects the RGB lines of the LCD to the Data lines (D0 - D2) of the microprocessor. This allows the LCD backlight to be set to seven different colors or off depending on what is sent to the port. The port is selected using the chip select line labeled COLORCS described in the design section of this report. The schematic describing the implementation of the LCD module and the color control port can be found in the appendix section of this report labeled LCD.

## Seven Segment Display

Another requirement of this project was to interface with a seven segment display. The seven segment used in this project was a *SA56-11SRWA* manufactured by Kingbright. The display was mainly used to debug the system and serves no real purpose in the final product. An image of the seven segment display and pinout from the datasheet can be seen below

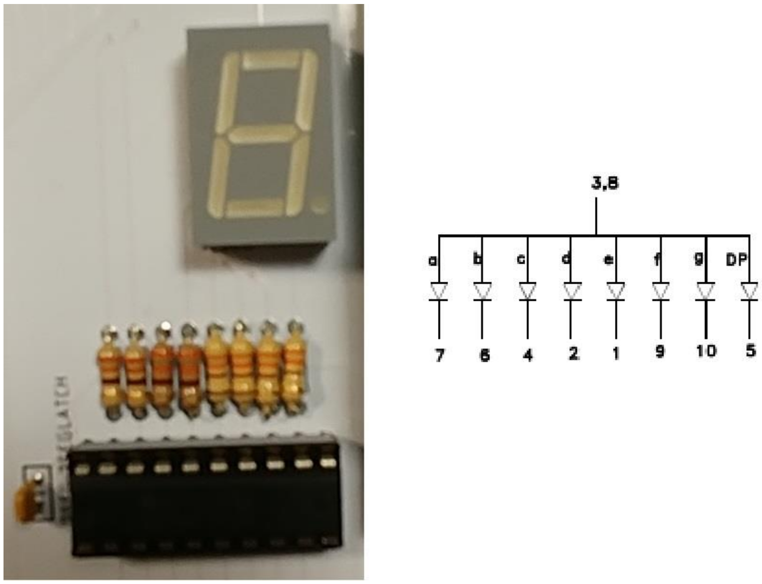


**Figure 10 - Seven segment display and pinout**

Since the seven segment is a device with no built-in port for interfacing one was created using a *74HCT573* chip. The pinout for the *74HCT573* is provided below for reference.

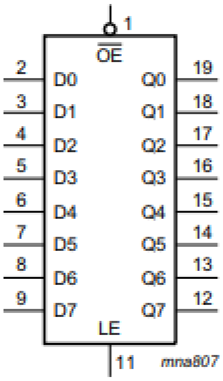


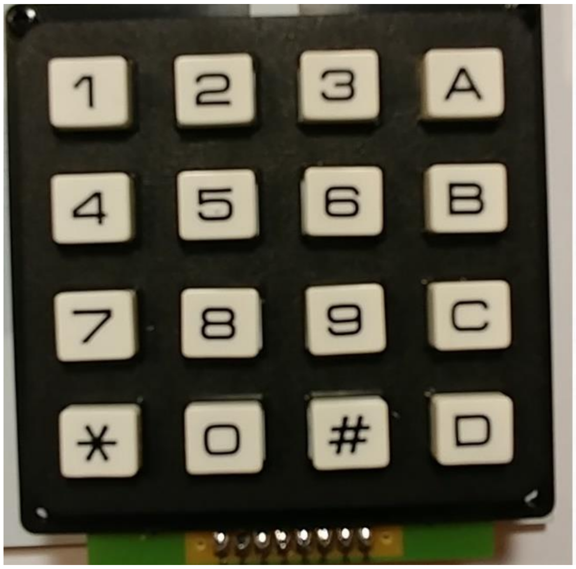
**Figure 11 - 74HCT573 pinout**

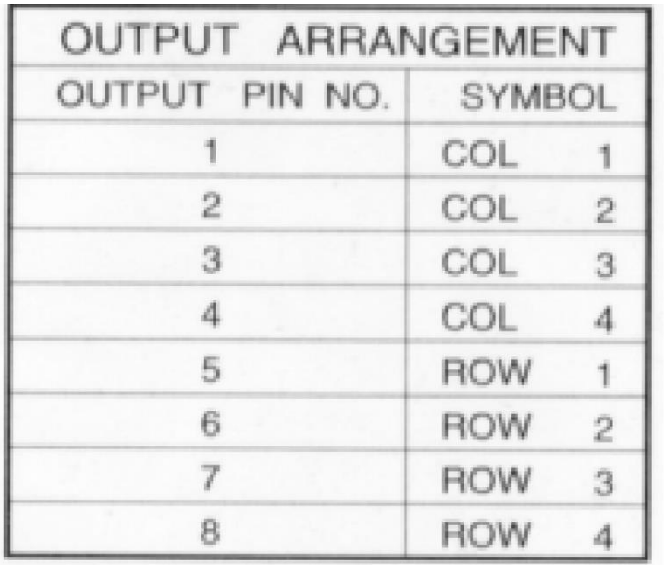
This provided a way of both the microprocessor selecting the device and to latch in the data provided by the microprocessor. To interface with the microprocessor the input data lines of the *74HCT573* (D0 - D7) are connected to the data lines of the microprocessor

(D0 - D7). The OE- of the *74HCT573* chip is tied to logic zero to always provide defined logic levels for the seven segment display. The LE of the *74HCT573* is attached to the chip select line for the seven segment display port mentioned in the design section of this report labeled 7SEGCSWR. The output pins (Q0 - Q7) of the *74HCT573* chip are connected to the different segment control lines (a - g, dp) and drive the display. The schematic describing the implementation of the seven segment display can be found in the appendix section of this report labeled 7SEG.

## Keypad

In order for a user to interface with the system an input device had to be implemented. A 4x4 keypad was used to satisfy this requirement. The keypad provides the only way of interfacing with the system. An image of the keypad as well as the pinout provided by the datasheet can be seen below.





| OUTPUT ARRANGEMENT | |
|---|---|
| OUTPUT PIN NO. | SYMBOL |
| 1 | COL 1 |
| 2 | COL 2 |
| 3 | COL 3 |
| 4 | COL 4 |
| 5 | ROW 1 |
| 6 | ROW 2 |
| 7 | ROW 3 |
| 8 | ROW 4 |

**FIGURE 12- KEYPAD MODULE AND PINOUT**

This keypad is interfaced with the microprocessor by attaching the keypad to the pins labeled port on the microprocessor. This design choice had many benefits. One it allowed interfacing with the keypad without having to use another chip select line. Two, it reduced the overall chip count of the system as no *74HCT573* chips had to be used to interface with the keypad. Finally, it increased the speed in which the keypad could be polled as no external write or read cycles were necessary to obtain information from the keypad. However; one downside to interfacing in this way is that no interrupt interaction can be done with the keypad and all inputs must be obtained by polling the keypad. The schematic describing the implementation of the keypad can be found in the appendix section of this report labeled Keypad.

## Real Time Clock

This project contains a real time clock device. The real time clock used for this system is the *RTC-72421A* chip manufactured by EPSON. This device provides a way to keep track of the amount of time that the system has been powered. An image of the RTC as well as the pinout provided by the datasheet can be seen below.

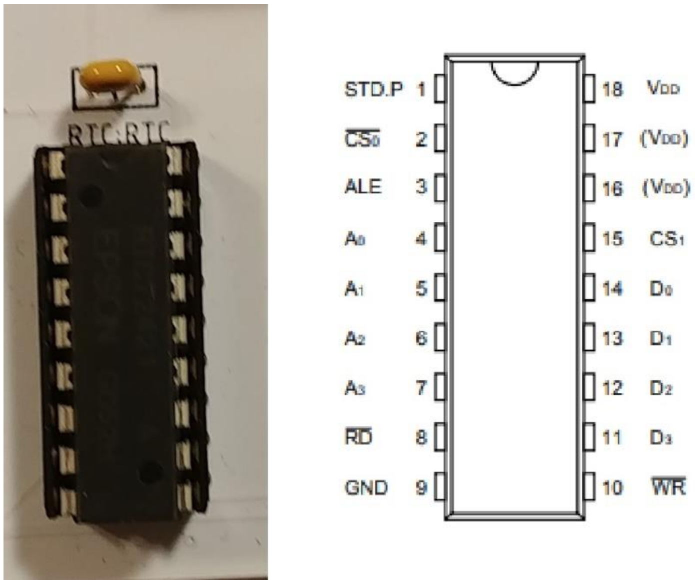


**FIGURE 13 - RTC CHIP AND PINOUT**

This RTC chip is provided as an 18-dip package. It has 16 internal registers that control the different parts of the time such as hours, months, seconds, etc. As well as control registers for setting the RTC. Each register has a data width of 4 bits. The RTC is interfaced with the microprocessor in the following way. Both the Address lines (A0 - A3) and the data lines

(D0 - D3) are connected to the corresponding data lines of the microprocessor (D0 - D3). This is possible since the RTC has an ALE pin to demultiplex the Address and data lines of the microprocessor which is connected to the ALE control line. The RD- and WR- control lines of the RTC are also connected to the RD- and WR- lines of the processor. The CS0 line of the RTC is attached to the chip select line labeled RTCCS described in the design section of this report. CS1 is tied to logic high. This pin is usually used as a power down indicator for the RTC but, is not used in this system. The STD.P pin of the RTC is an interrupt pin used updating the microprocessor at specific intervals but, is also unused in this system. For this reason, STD.P is left as a no connect as specified by the datasheet. The schematic sheet describing the implementation of the RTC can be found in the appendix section of this report labeled RTC.

## Analog to Digital Converter

Another requirement for this project was to interface with an Analog to Digital Converter (ADC). To satisfy this requirement a *TL0820ACN* chip manufactured by Texas Instruments was used. The ADC in this system is used to read the output of a *TMP36GT9Z* temperature sensor manufactured by Analog Device Inc. Images of both the ADC and temperature sensor as well as the pinout provided by the datasheets for each device can be seen below.

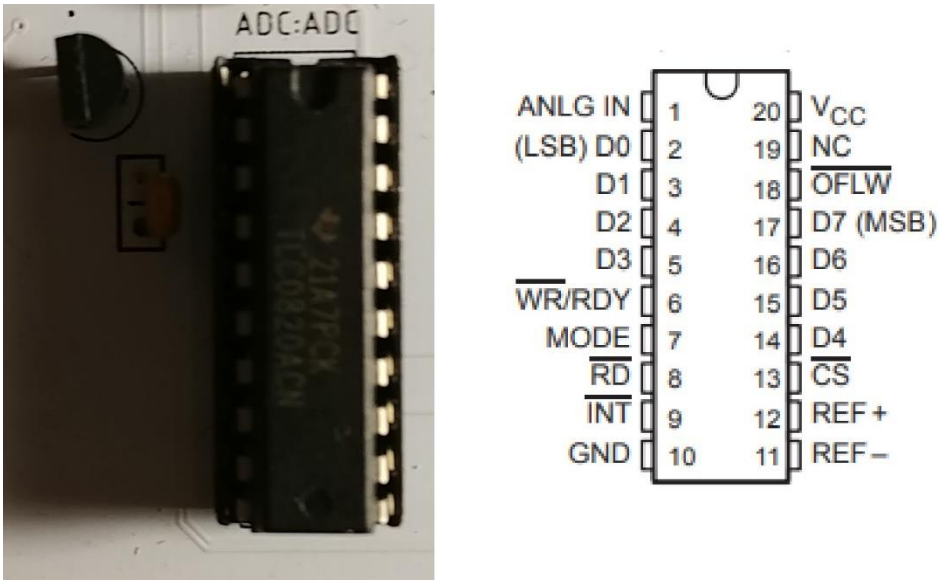


FIGURE 14 - ADC CHIP AND PINOUT

This ADC is provided as a 20-dip package. It has a resolution of 8 bits but can be configured to work with multiple ADCs to increase resolution. This system only uses one chip to read temperatures from the sensor. To interface with the processor the ADC is connected in the following way. The data lines of the ADC (D0 - D7) are connected to the data lines of the microprocessor (D0 - D7). A 5-volt reference was supplied to the voltage reference pins of the ADC. The mode pin of the ADC was pulled high to configure it to read write mode which allows the processor to control when new conversion values are latched into the ADC. The RD- and WR- pins of the ADC were both attached to the RD- and WR- control lines of the microprocessor. Finally, the CS- line of the ADC was attached to the chip select line described in the design section of the report labeled ADCCS. All other pins of the ADC were left no connect as specified by the datasheet. The schematic sheet describing the implementation of the ADC can be found in the appendix section of this report labeled ADC.

## Software Discussion

The second part of this project was to compose software that would run on the system. This software would have to be capable of performing the following programs,

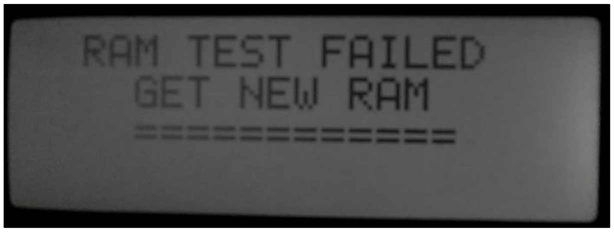
- Display the contents of RAM(Dump)
- Search for a specific byte value in RAM(Search)
- Edit the contents of RAM(Edit)
- Fill RAM with blocks of data(Fill)
- Move blocks of RAM from one location to another(Move)
- Display the temperature read from the ADC
- Display the time from the RTC

The code for this project was written completely in C and compiled into hex code using the Small Device C Compiler(SDCC). This section will discuss software startup sequence of the system and structure of the software. Afterwards, each required program will be discussed in further detail including what the program supposed to do and what the user sees when using the program. For a more detailed explanation of *how* each program was coded the source code for the software is provided in the appendix section of the report under code listing.

## Startup & Menu

Upon powering the system or reset, the software starts off with initialization of the various hardware components. This process first starts with the LCD. The LCD is initialized first so that it can be used to provide feedback to the user explaining what was currently happening in the system.

Following initialization of the LCD the RAM chips are tested to make sure that they are functioning properly. This is done by writing 0x55 to the entire 64k of RAM space and reading back the value. The processor proceeds to read back the RAM and if any value read is something other than 0x55 the system will lock up and ask to be restarted. This process is repeated for the value 0xAA. The resulting screen upon RAM test failure can be seen below.

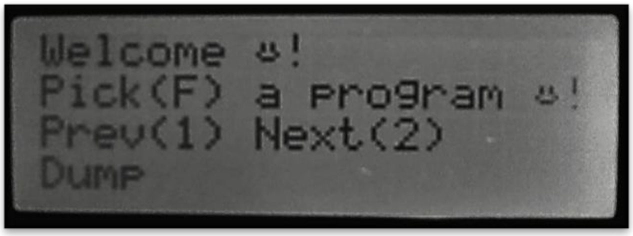


```
RAM TEST FAILED
  GET NEW RAM
  ============
```

**FIGURE 15 - RAM TEST FAILED SCREEN**

Following the RAM test the RTC is the initialized. This is done following the procedure described in the datasheet for the RTC. The RTC in this system is used specifically for an uptime counter of the system so all timer registers of the chip are initialized to zero on startup.

If the processor successfully executes the startup sequence described above the user is brought to the main menu. An image of the main menu can be seen below.



```
Welcome ☺!
Pick(F) a program ☺!
Prev(1) Next(2)
Dump
```

**FIGURE 16 - MAIN MENU**

The main menu is where the user can select a program to execute. The user can cycle through the different program options using the “1” and “2” keys on the keypad. The different program options and structure of software are shown in the tree below.

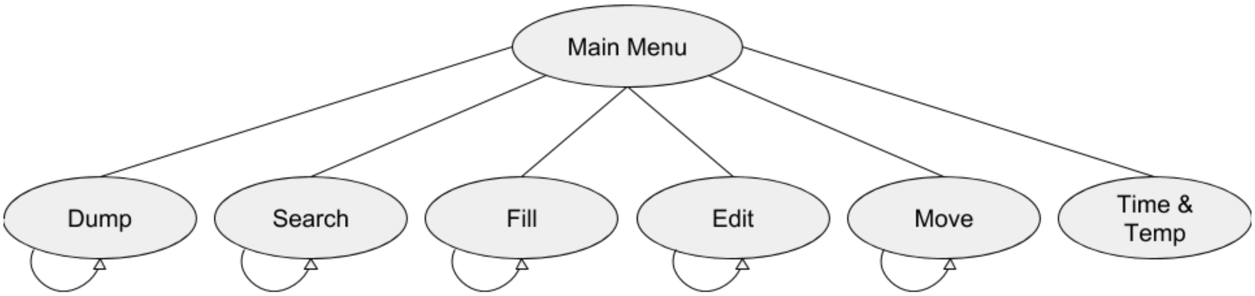
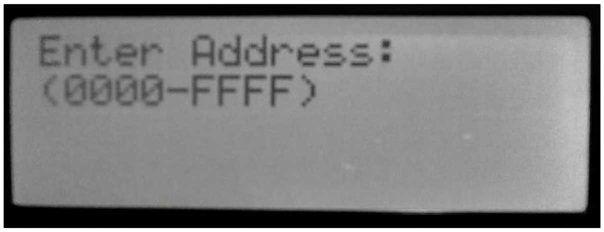


**FIGURE 17 - PROGRAM STRUCTURE DIAGRAM**

The user can select which program they would like to execute using the "*" key on the keypad. Once a program is selected the execution of that program is started and the user must complete at least one cycle of the program before returning to the main menu. The functionality of each of these programs is discussed in the following sections.

## Dump -

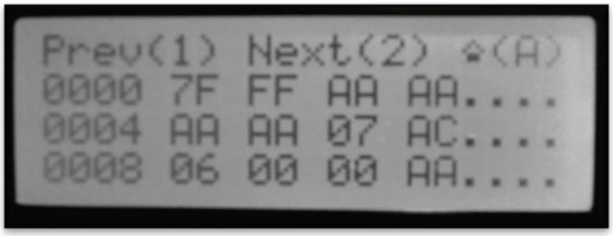
The dump program is a minified version of a debug hex dump. It is there so that the user can view the contents of the system RAM in both hex and ascii formats. When the user selects this program from the main menu they are immediately prompted for an address which can be seen in the image below.



```
Enter Address:
(0000-FFFF)
```

**FIGURE 18 - ADDRESS PROMPT SCREEN (DUMP)**

The entered value is used as the starting location to start displaying the contents of RAM. Each page show 12 bytes of RAM data and the user can view the previous 12 bytes or next 12 bytes using the "1" and "2" keys on the keypad respectively. An example of this page can be seen below.



```
Prev(1) Next(2) ⌂(A)
0000 7F FF AA AA....
0004 AA AA 07 AC....
0008 06 00 00 AA....
```

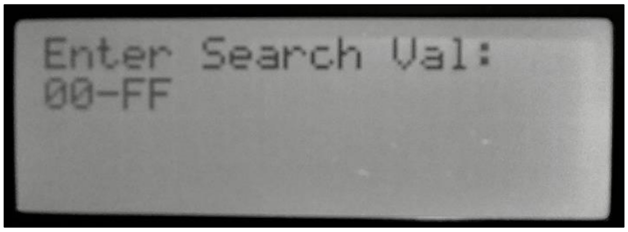
**FIGURE 19 - EXAMPLE DUMP PAGE**

If the user cycles past the end of RAM in either direction (0x0000 or 0xFFFF) the program wraps around displaying values at the other end.

After the user is done viewing the contents of data memory they can press the "A" button on the keypad to return to the main menu as prompted by the icon in the upper right corner of the LCD.

## Search

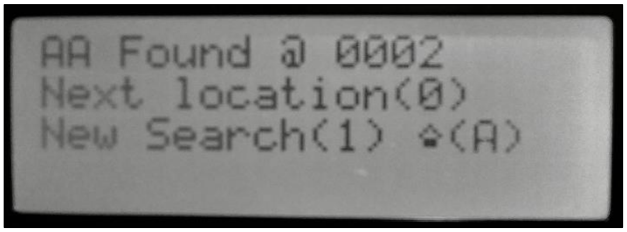
The search program is used to find a specific value in RAM. This program searches the entire contents of RAM memory for a specified value provided by the user. Upon selection of this program from the main menu the user is prompted for a search value as seen below.



```
Enter Search Val:
00-FF
```

FIGURE 20 - SEARCH VALUE PROMPT SCREEN

After the value is accepted the program begins searching RAM starting at address 0x0000. The first address with the value found, if one is found, is displayed to the user. An example of this screen can be seen below.



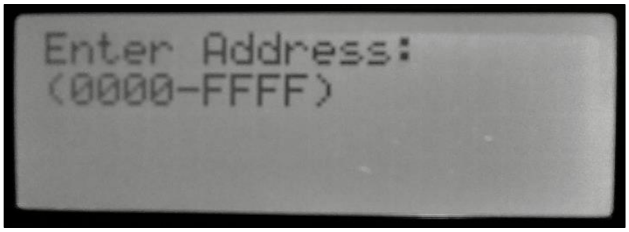
```
AA Found @ 0002
Next location(0)
New Search(1) ⌂(A)
```

FIGURE 21 - SEARCH VALUE RESULT SCREEN

If no value is found "Not found" is simply displayed instead. Regardless if a value is found or not the user is then prompted to either continue by pressing "0" which starts the same search again starting past where the previous search left off, conduct a new search by pressing "1" which simply restarts the search program from the beginning, or return to the main menu by pressing "A" so that the user can select a new program.

## Fill

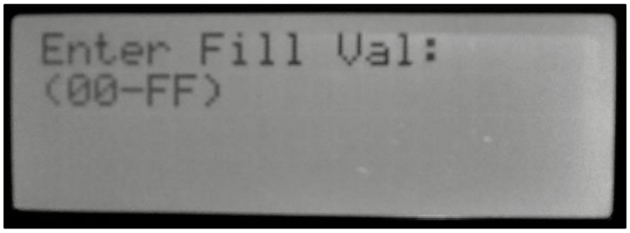
The fill program is a way of populating large blocks of ram with repeated values. This program can fill blocks up to 0xFF in size with any byte value. When the user selects the fill program from the main menu they are first prompted for the address they would like to start the fill at which can be seen below.



```
Enter Address:
(0000-FFFF)
```

FIGURE 22 - ADDRESS PROMPT SCREEN (FILL)

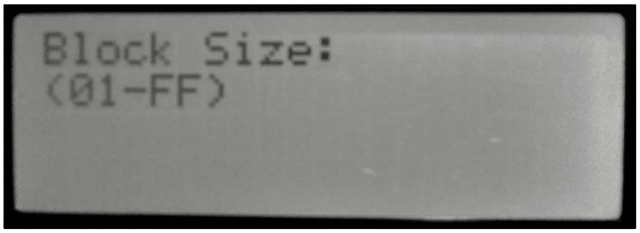
After selecting the starting address, the user is then prompted for a value ranging from 0x00 - 0xFF to use as the fill. An example of this prompt can be seen below.



```
Enter Fill Val:
(00-FF)
```

**FIGURE 23 - FILL VALUE PROMPT SCREEN**

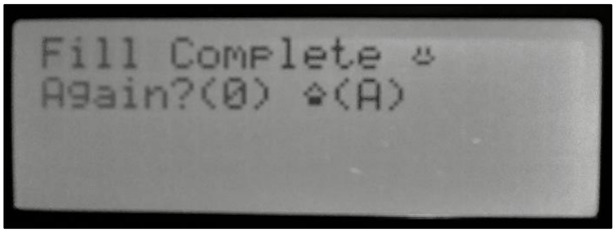
Finally, after entering a value the user is prompted for a block size to fill ranging from 0x01 to 0xFF. If the user enters 0x00 the program will reject the outcome and ask for the block size again. This prompt can be seen below.



```
Block Size:
(01-FF)
```

**FIGURE 24 - BLOCK SIZE PROMPT SCREEN (FILL)**

The program then takes these values and fills the block of RAM with the specified value. It should be noted that if the block chosen goes past the last address of RAM (0xFFFF) then the block will loop around and start writing at 0x0000. Upon completion the user is then prompted to either do another fill operation by pressing "0" which restarts the program or to return to the main menu by pressing "A" to select another program. This prompt can be seen below.

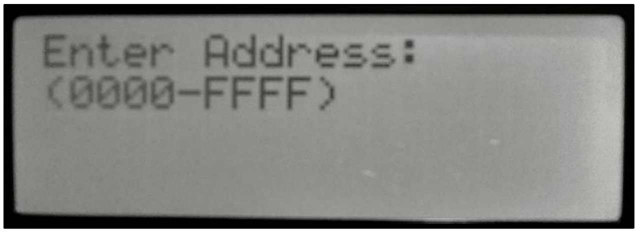


```
Fill Complete ☺
Again?(0) ⌂(A)
```

**FIGURE 25 - FILL RESULT SCREEN**

## Edit

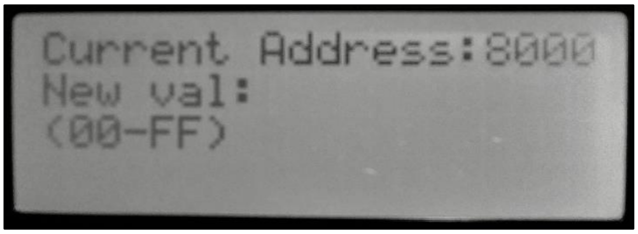
The edit program is very similar to the fill program. While both are used to edit RAM, edit does it one byte at a time. Upon selection of the program the user is requested to enter a starting address which can be seen below.



```
Enter Address:
(0000-FFFF)
```

FIGURE 26 - ADDRESS PROMPT SCREEN (EDIT)

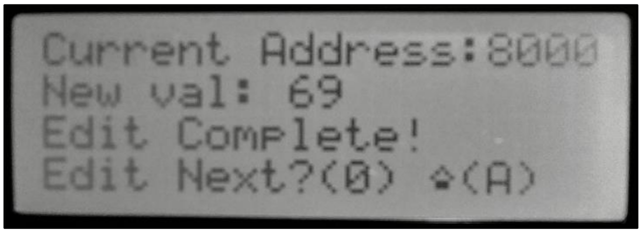
After selecting a starting address, the user is brought to the edit screen where they are prompted to enter a byte value to put at the starting address this screen can be seen below.



```
Current Address:8000
New val:
(00-FF)
```

FIGURE 27 - EDIT VALUE SCREEN

After selecting a value to place at the specified address the user is given the choice to either edit the next sequential byte in memory by pressing “0” or return to the main menu by pressing “A”. Like other programs available on the system, edit does memory wrapping so the next value edited after 0xFFFF will be 0x0000. The continue prompt mentioned above can be seen below.

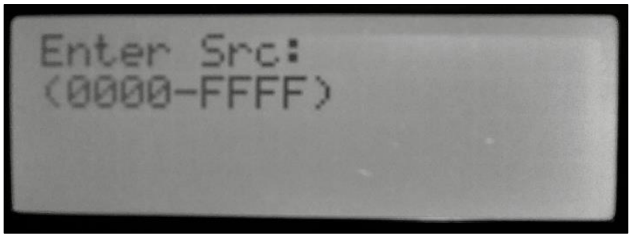


```
Current Address:8000
New val: 69
Edit Complete!
Edit Next?(0) ⌂(A)
```

FIGURE 28 - EDIT RESULT SCREEN

## Move

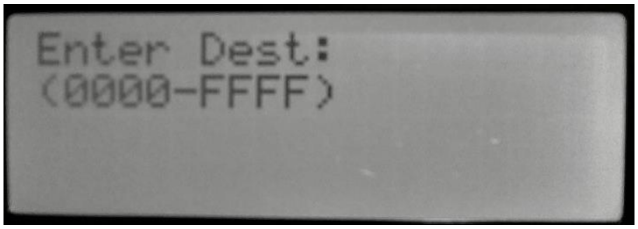
Move is another RAM edit program available on the system. Move can copy a block of memory from one point in RAM and place it in another section of RAM. Upon selection of the move program the user is prompted for a source address which can be seen below.



```
Enter Src:
(0000-FFFF)
```

FIGURE 29 - MOVE SOURCE PROMPT SCREEN

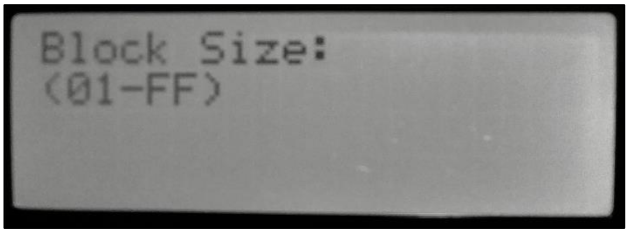
After entering a source address, the user is immediately requested to enter a destination address. The prompt for this screen can be seen below.



Enter Dest:
(0000-FFFF)

**FIGURE 30 - MOVE DESTINATION PROMPT SCREEN**

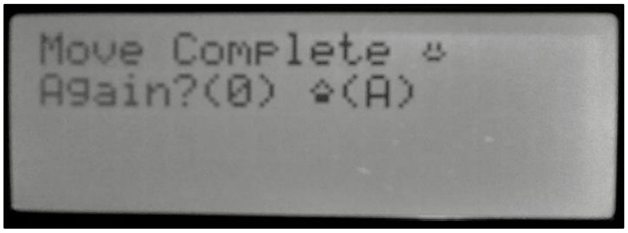
Finally, the user is asked to enter a block size in which to move ranging from 0x01 to 0xFF. If a block size of 0x00 is entered the program will ask for the block size again until the user enters a valid value. An example of this prompt can be seen below.



Block Size:
(01-FF)

**FIGURE 31 - BLOCK SIZE PROMPT SCREEN (MOVE)**

After entering the required information for the move the program proceeds to copy the bytes from the source location to the destination for the block size specified. If the block of memory goes past 0xFFFF the program will simply loop around to 0x0000. The user is notified when the move is complete and is then prompted to either move again by pressing “0” which starts the move program from the beginning or to return to the main menu by pressing “A” to select another program. This prompt can be seen below.



Move Complete ☺
Again?(0) ⌂(A)

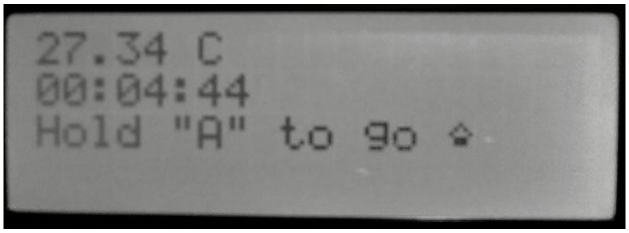
**FIGURE 32 - MOVE RESULT SCREEN**

## Time & Temp

The time & temp program displays the current state of the ADC and the RTC to the user. The program refreshes these states at a rate of once a second. The temperature read from the ADC is displayed in Celsius and is calculated from the following equation where A is the value read from the ADC and T is the temperature in Celsius.

$$T = \frac{5 \cdot 10}{256} A$$

Below the temperature the hours, minutes, and seconds from the RTC are read and displayed on the LCD. This is the amount of time that the system has been running since last power on or reset. If the user would like to return home, they simply must hold the “A” key on the keypad as displayed on the screen. The screen for this program can be seen below.



```
27.34 C
00:04:44
Hold "A" to go ⇧
```

FIGURE 33 - TIME & TEMP SCREEN

# Issue Discussion

With any project comes unforeseen problems. Some problems are harder to tackle than others but, all impede the progress of the project. Some problems can even completely change the approach you are taking when completing the project. This project had its fair share of issues both when debugging the circuit and composing the software that runs on the system. This section will cover both hardware and software problems encountered when doing the project, how they were identified, and finally how they were resolved.

## Hardware Problems

Hardware problems are problems that were encountered during the circuit debug process and are independent of what software is running on the system at the time. The first hardware problem encountered during this project was during the bread boarding phase of the project. The LCD was incorrectly attached to the power lines of the circuit. This problem was identified when the LCD proceeded to glow a faint red and was hot to the touch. The problem was easily resolved by purchasing a new LCD and correcting both the breadboard circuit as well as the schematic for the LCD to make sure this issue did not happen again.

Another issue encountered when completing this project was the timing requirements for the LCD write and read cycles were not being met. This problem was identified after attaching a Saleae digital analyzer to the inputs of the LCD to test what data was being sent to the module. Fortunately, the digital analyzer was able to track the protocol that was being used to communicate with the LCD to see if it met the standards of the LCD module. Unfortunately, both the R/W- lines and RS lines of the LCD were not being held for long enough after the negative edge of the E pin. This was because they were all being generated using the RD- and WR- lines of the microcontroller to be generated. To fix this problem address lines A0 and A1 were connected to the R/W- and RS lines of the LCD instead as they would be held long enough after the negative edge of the E signal. This did increase the complexity of interfacing with the LCD but, resolved the issue at hand.

One final issue that was encountered while doing this project was the lack of voltage regulation while bread boarding the circuit. Due to the sheer amount of parts involved when creating a prototype of the system multiple breadboards were used. An image of the prototype on the breadboard can be seen in the appendix section of this report. As the number of parts increased the voltage became less and less reliable with the more surface area that had to be covered. This was identified because a multimeter was kept close at hand to check the voltage system periodically while prototyping it. To alleviate this problem a central power rail was added to the breadboard. This reduced the amount of power rails that had to be brought to the power voltage increasing the stability of the input voltage.

## Software Problems

Software problems are problems that were encountered after the entire circuit was verified to be working as intended. While many small software issues were encountered while developing the software for this project only a few will be mentioned here otherwise this section could go on forever.

One of the first software problems encountered while developing the software for this project was when the structure of the program was being arranged. The system uses function pointers to decide where to go next after a program terminates. When this was first implemented despite which function pointer was loaded into the next function variable the entire program would restart back at the beginning before the initialization phase. This was caused because prototypes for the program functions were not made at the beginning of the code causing the compile to incorrectly set where these programs were in memory. However; this issue was not caught by the compiler because the functions were never explicitly called only indirectly through a function pointer variable.

One final major problem that was encountered during the software development phase of this project was the lack of a previous keypad state. Since the keypad used in this project was just a dumb device there was no way to pull for the last keypress or any previous data which made programming user inputs difficult. To alleviate this issue a simple keypad state variable was placed in memory. This variable is updated every time the keypad is read. This allows one to keep track of previous key presses and made designing the menu easier and more user friendly.

# Conclusion

To review, the goal of this project was to build a computer system that met the following hardware requirements,

- Use an 8051 Microprocessor
- Implement 64k Code memory
- Implement 64k Data memory
- Interface with an LCD
- Interface with a 7 Segment display
- Interface with a keypad
- Interface with an Analog to Digital Converter(ADC)
- Interface with a Real Time Clock(RTC)

Additionally, this system should have software that can perform the following functionalities,

- Display the contents of RAM(Dump)
- Search for a specific byte value in RAM(Search)
- Edit the contents of RAM(Edit)
- Fill RAM with blocks of data(Fill)
- Move blocks of RAM from one location to another(Move)
- Display the temperature read from the ADC
- Display the time from the RTC

This project was to be done over the course of the semester to help one develop the necessary skills to tackle project of this nature as well as help prepare for the capstone required for students pursuing a degree in electrical and computer engineering.

When starting this project, it first seemed very daunting. The amount of knowledge and skills required to build it was immense and the amount of time given to complete it seemed short. However; after breaking down the project into separate problems like first getting the processor to work, then getting the seven segment display to work, then the external memory, etc. the project became more manageable.

Now that it is all complete it is hard to imagine that the project posed any threats at all. This is because of all the new skills obtained when doing the project. Learning to efficiently read datasheets, break up large problem into small problems, and the willingness to test ideas before dismissing them are all major things I took away from this project but, not everything. These newly obtained skills can be applied to any project if done correctly and will be huge boons to any work pursued in the future.

# Appendix

# BOM

| ***Part Name*** | ***Part Specification*** | ***Vendor*** | ***Available Part Number*** | ***Quantity*** |
|---|---|---|---|---|
| **Socket - 0.6"** | 40-pin processor | DigiKey | ED3048-5-ND | 1 |
| **Socket - 0.6"** | 28-pin RAM | DigiKey | 3M5469-ND | 2 |
| **Socket - 0.3"** | 24-pin GAL | DigiKey | 3M5466-ND | 2 |
| **Socket - 0.3"** | 20-pin 573 & ADC | DigiKey | 3M5465-ND | 6 |
| **Socket - 0.3"** | 18-pin RTC | DigiKey | AE9995-ND | 1 |
| **ZIF Socket** | 28-pin ROM | DigiKey | A302-ND | 2 |
| **Microprocessor** | AT89C55HWD | DigiKey | AT89C55WD-24PU-ND | 1 |
| **ADC** | TLC0820ACN | DigiKey | 296-2849-5-ND | 1 |
| **Temperature Sensor** | TMP36GT9Z | DigiKey | TMP36GT9Z-ND | 1 |
| **8-bit Latch/Demultiplexer** | 74HCT573 | DigiKey | 296-1621-5-ND | 5 |
| **LCD Module** | 20x4 LCD | DigiKey | 1528-1508-ND | 1 |
| **7-Segment LED (Common A)** | SA56-11SRWA | DigiKey | 754-1460-5-ND | 1 |
| **RTC** | RTC-72421 | DigiKey | SER3231-ND | 1 |
| **Crystal (Oscillator)** | 12 MHz | DigiKey | XC269-ND | 1 |
| **Toggle switch** | SPDT | DigiKey | EG2355-ND | 2 |
| **SPST Push-Button Switch** | Reset Signal | DigiKey | 450-1650-ND | 1 |
| **Screw potentiometer** | 10 K | DigiKey | 3362P-103LF-ND | 1 |
| **Fuse** | 2.5 A | DigiKey | F2400-ND | 1 |
| **Fuse holder** | Single 2-Pin Clip | DigiKey | F4189-ND | 2 |
| **8-pin Female header** | Keypad mount | DigiKey | S7041-ND | 1 |
| **40-pin Male header** | LCD & Keypad | DigiKey | S1011EC-40-ND | 1 |
| **16-pin Female header** | LCD mount | DigiKey | S7014-ND | 1 |
| **24-pin Female header** | Sensors | DigiKey | S7022-ND | 1 |
| **Capacitor** | Electrolytic 10uF | DigiKey | 493-10487-1-ND | 1 |
| **Capacitor** | Ceramic 0.1 uF | DigiKey | BC1160CT-ND | 13 |
| **Resistor** | 330 ohm | DigiKey | CF14JT330RCT-ND | 8 |
| **Resistor** | 1K ohm | DigiKey | CF14JT1K00CT-ND | 2 |
| **Resistor** | 10K ohm | DigiKey | CF14JT10K0CT-ND | 1 |
| **RAM 32kx8** | 62256-lp70 | Jameco | 82472 | 2 |
| **EEPROM 32kx8** | AT28C256-15PU | DigiKey | AT28C256-15PU-ND | 2 |
| **GAL** | GAL22V10D | Jameco | 39167 | 1 |
| **Keypad 4x4** | 27899 | Jameco | 2131055 | 1 |
| **Green LED** | Power indicator | DigiKey | C566C-RFS-CT0W0BB2CT-ND | 1 |
| **DC power jack** | 2.1mm | Jameco | 101179 | 1 |
| **Wall power adapter** | 5V DC | Jameco | 252736 | 1 |

# Project Progress Images

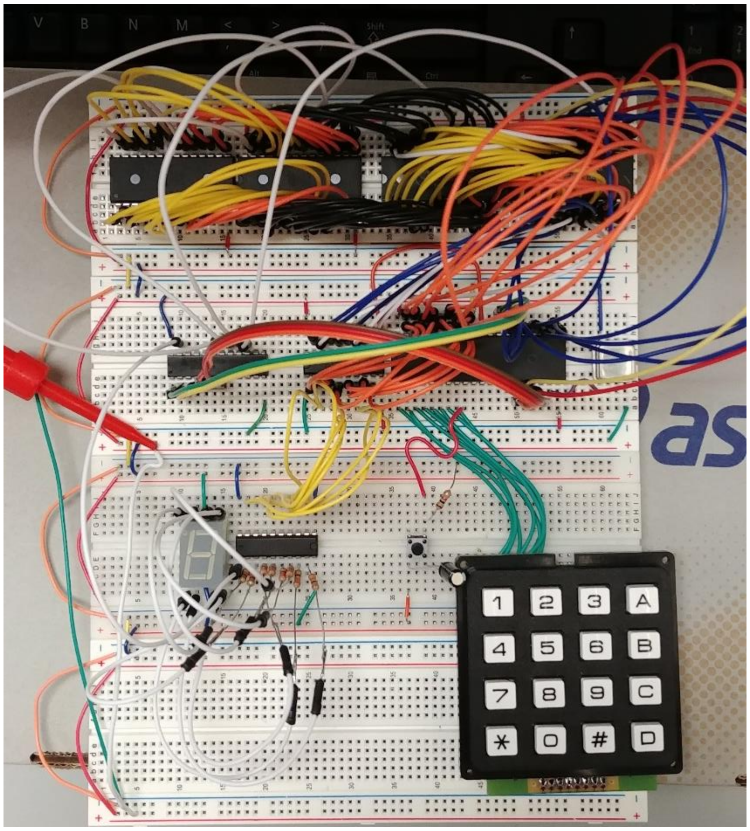


FIGURE 34 - FIRST PROGRSS PICTURE

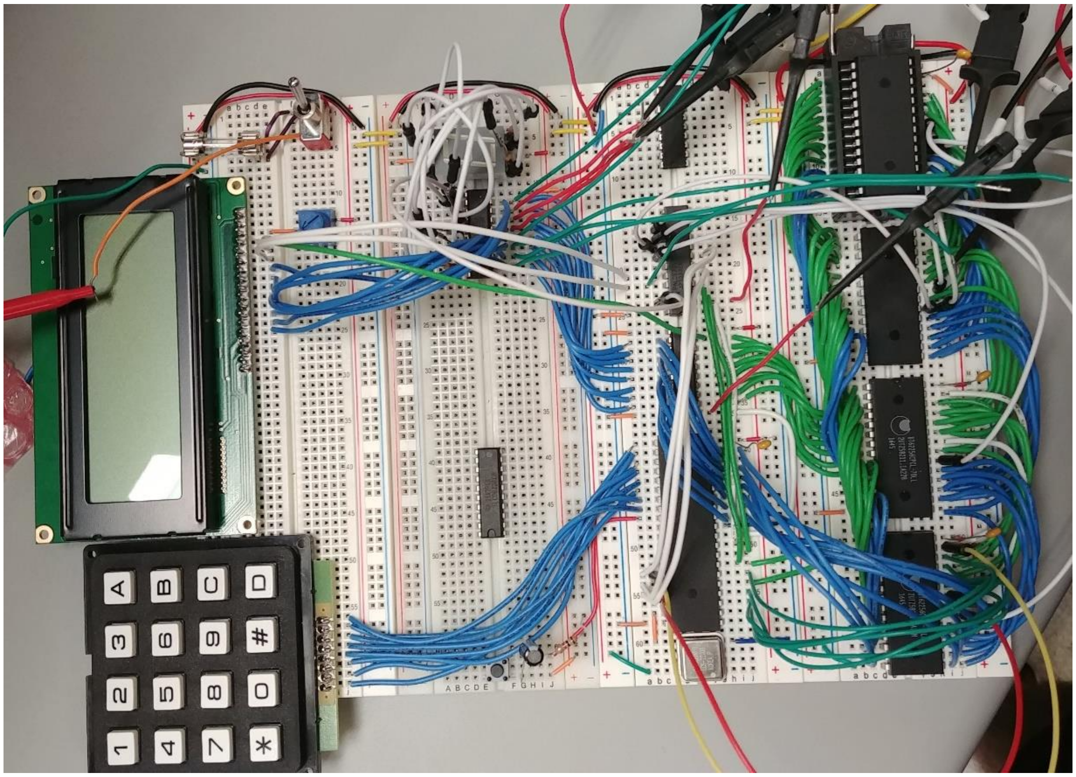


FIGURE 35 - SECOND PROGRESS PICTURE

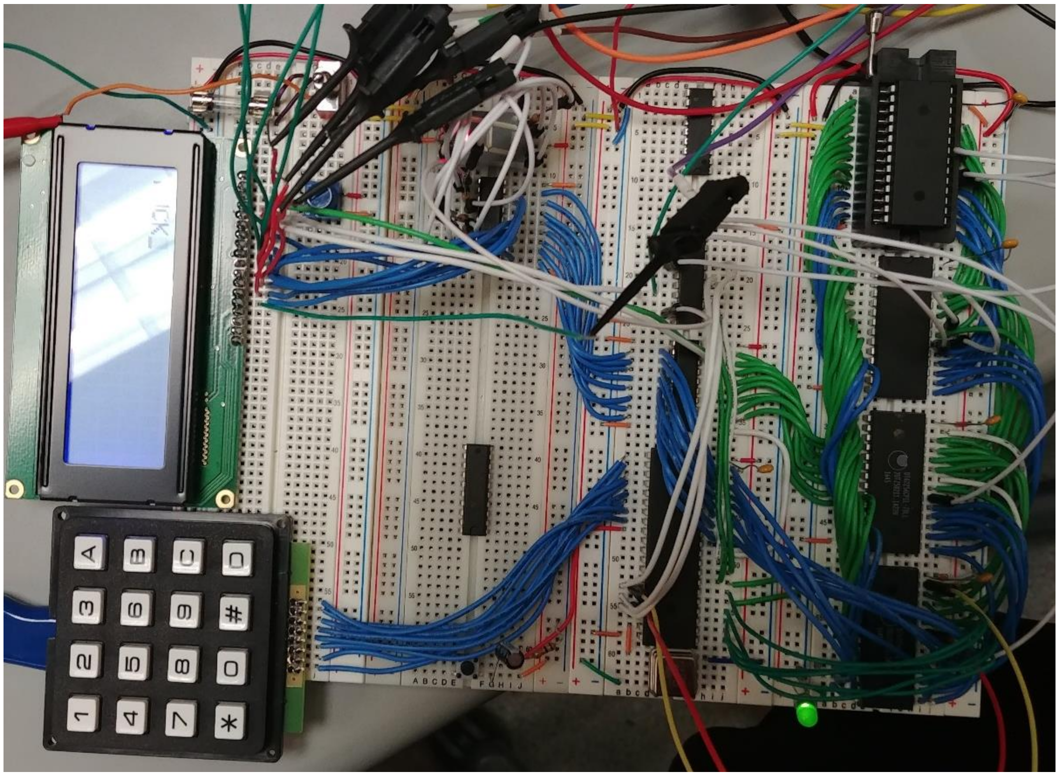


Figure 36 - Third progress picture

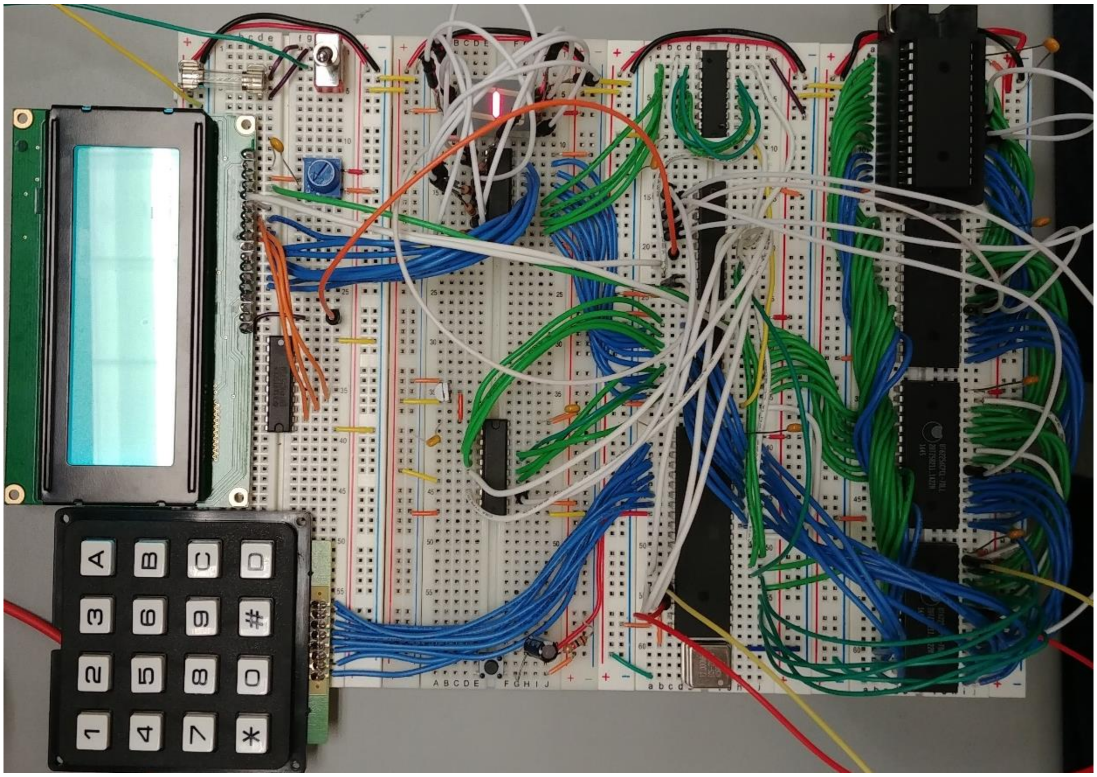


Figure 37 - Fourth progress picture

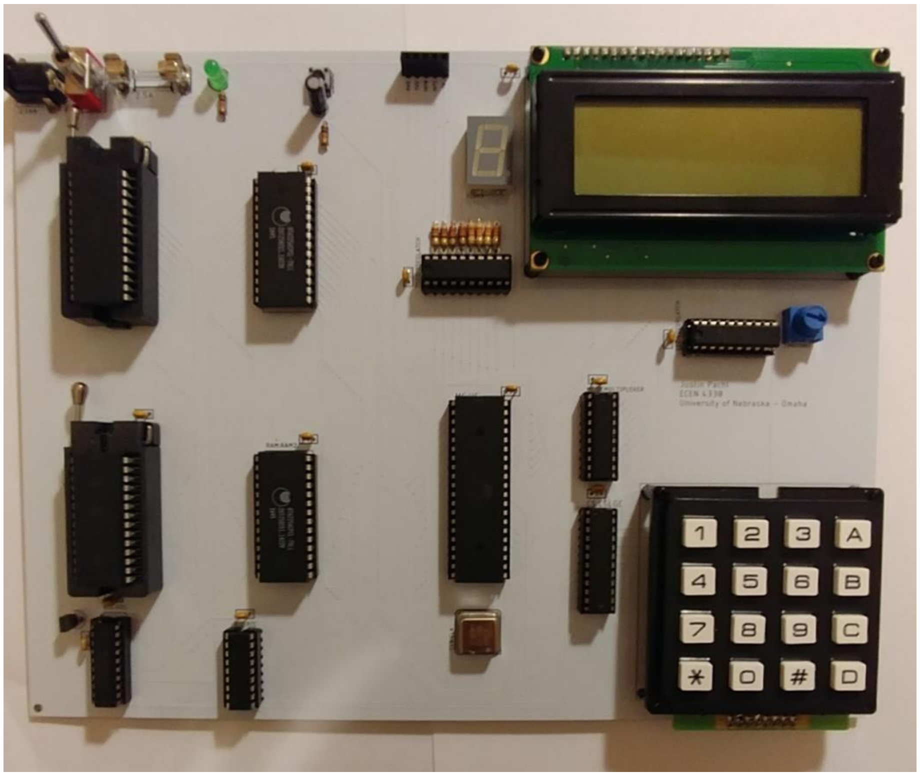


FIGURE 38 - FINAL PRODUCT

# PCB Images

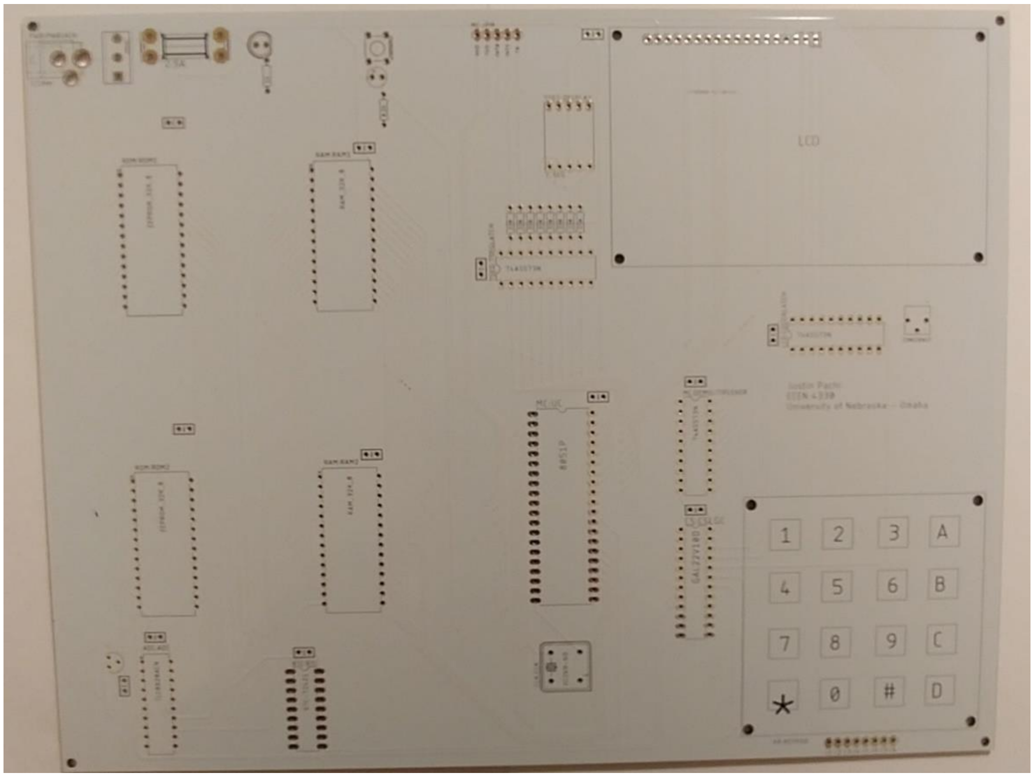


**FIGURE 39 - UNPOPULATED PCB (FRONT)**

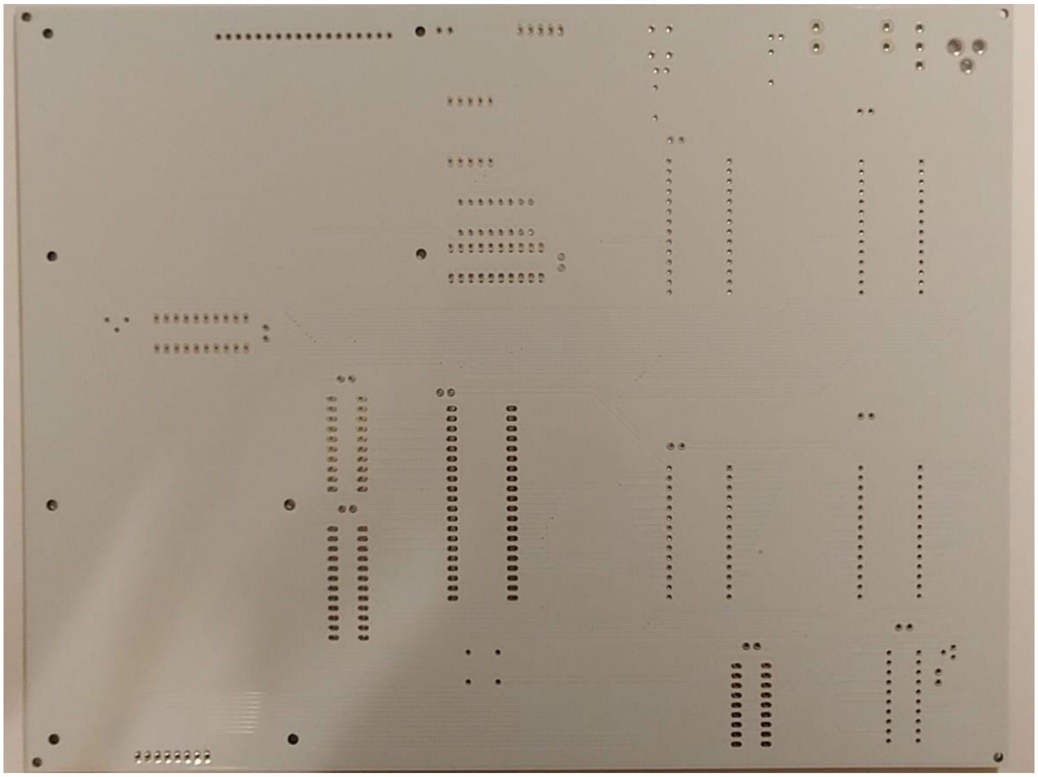
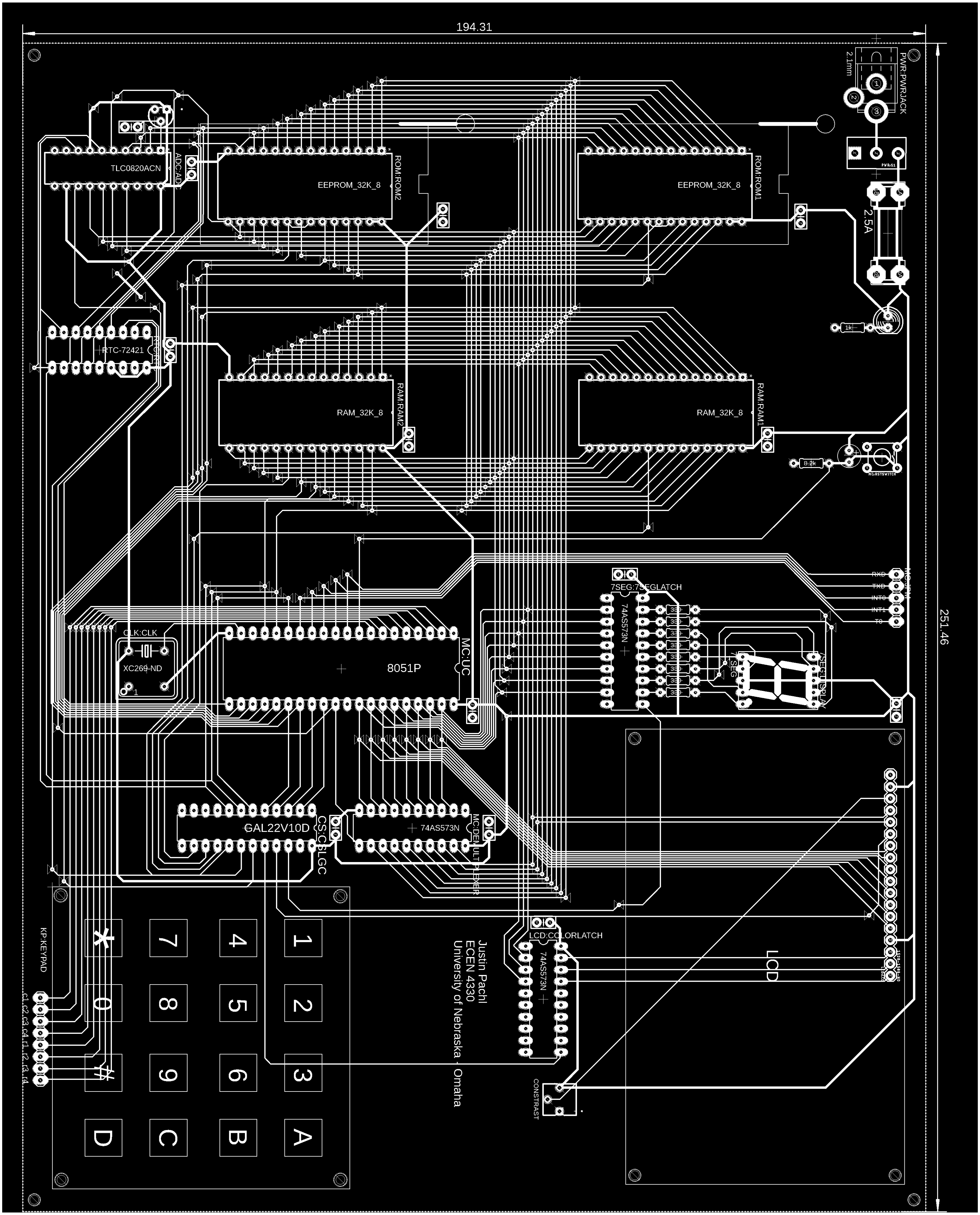
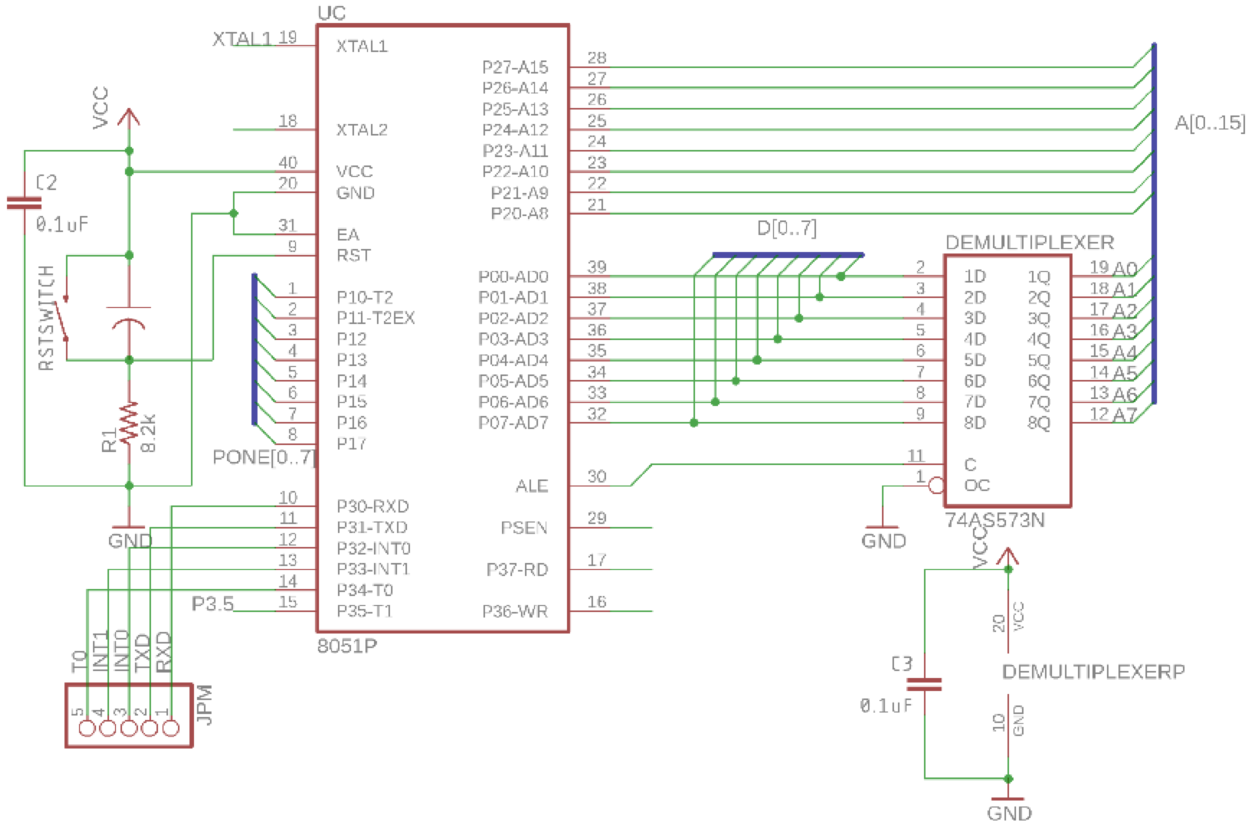


**FIGURE 40 - UNPOPULATED PCB (BACK)**


194.31
251.46
PWR:PWRJACK
2.1mm
PWR:S1
2.5A
1k
8.2k
MC:RSTSWITCH
TLC0820ACN
ADC:ADC
EEPROM_32K_8
ROM:ROM2
EEPROM_32K_8
ROM:ROM1
RTC-72421
RAM_32K_8
RAM:RAM2
RAM_32K_8
RAM:RAM1
CLK:CLK
XC269-ND
8051P
MC:UC
7SEG:7SEGLATCH
74AS573N
330
7SEG:DISPLAY
RXD
TXD
INT0
INT1
T0
GAL22V10D
CS:CSLGC
74AS573N
MC:DEMULTIPLEXER
LCD:COLORLATCH
74AS573N
CONTRAST
LCD
KP:KEYPAD
c1 c2 c3 c4 r1 r2 r3 r4
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D
Justin Pachl
ECEN 4330
University of Nebraska - Omaha

# Schematic Drawings



Microprocessor

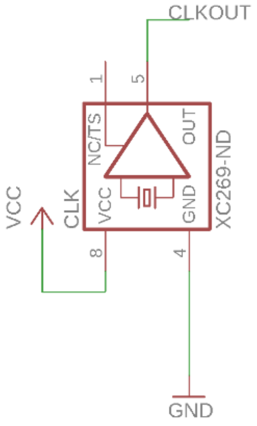
TITLE: PROJECT_V2

Document Number:

REV:

Date: 5/2/2018 10:51 PM

Sheet: 1/11



Clock Circuit

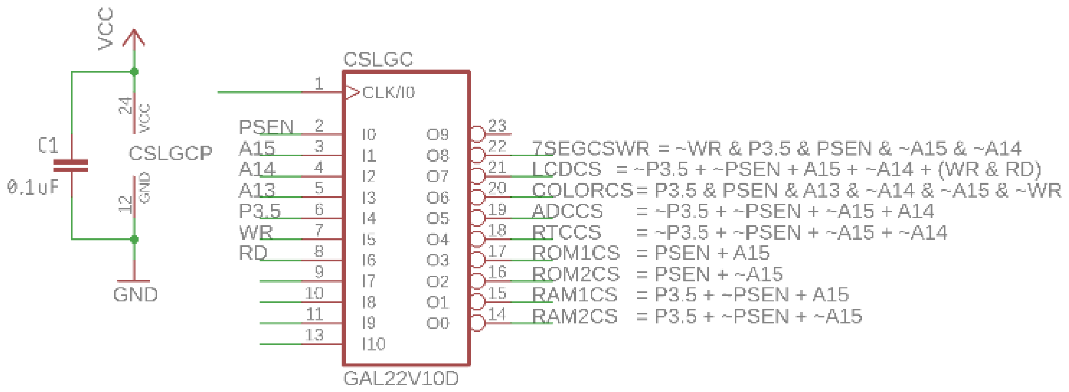
TITLE: PROJECT_V2

Document Number:

REV:

Date: 5/2/2018 10:51 PM

Sheet: 2/11


VCC
C1
0.1uF
24
VCC
12
GND
CSLGCP
GND
CSLGC
1
CLK/I0
PSEN 2
A15 3
A14 4
A13 5
P3.5 6
WR 7
RD 8
9
10
11
13
I0
I1
I2
I3
I4
I5
I6
I7
I8
I9
I10
O9
O8
O7
O6
O5
O4
O3
O2
O1
O0
23
22 7SEGCSWR = ~WR & P3.5 & PSEN & ~A15 & ~A14
21 LCDCS = ~P3.5 + ~PSEN + A15 + ~A14 + (WR & RD)
20 COLORCS= P3.5 & PSEN & A13 & ~A14 & ~A15 & ~WR
19 ADCCS = ~P3.5 + ~PSEN + ~A15 + A14
18 RTCCS = ~P3.5 + ~PSEN + ~A15 + ~A14
17 ROM1CS = PSEN + A15
16 ROM2CS = PSEN + ~A15
15 RAM1CS = P3.5 + ~PSEN + A15
14 RAM2CS = P3.5 + ~PSEN + ~A15
GAL22V10D

PSEN Must be high for this to work during external memory access

Chip select

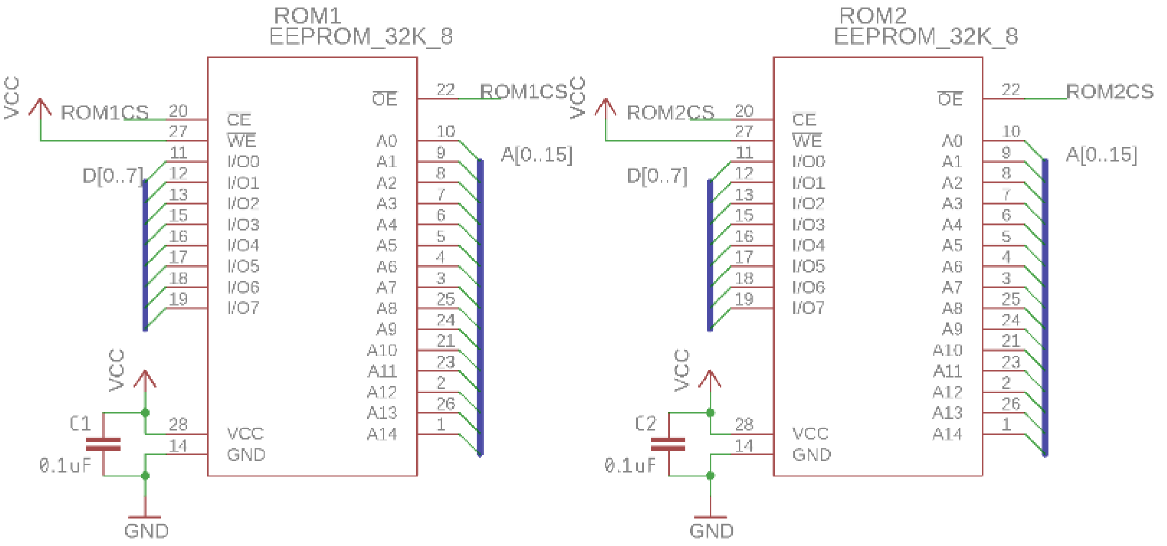
TITLE: PROJECT_V2

Document Number:

REV:

Date: 5/2/2018 10:51 PM

Sheet: 3/11



ROM

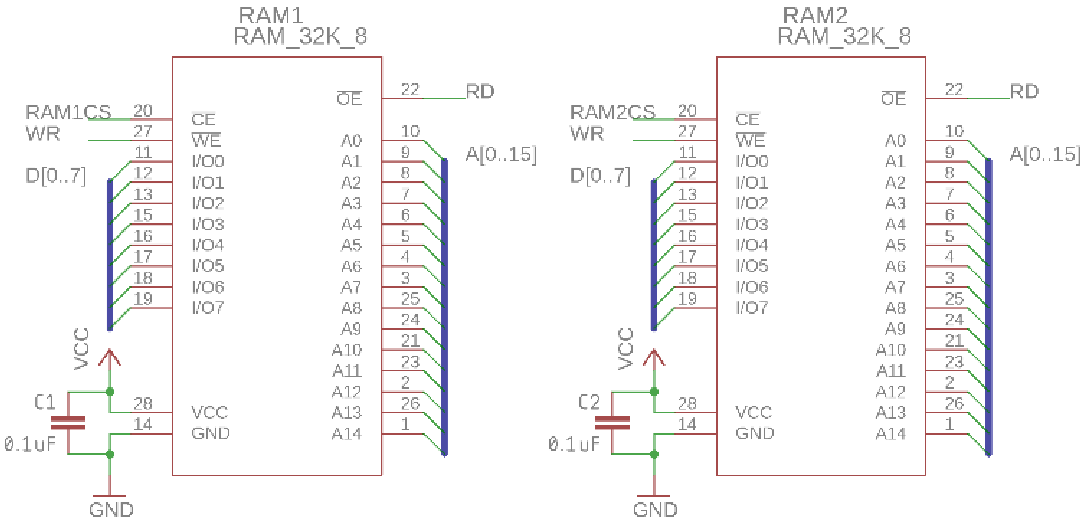
TITLE: PROJECT_V2

Document Number:

REV:

Date: 5/2/2018 10:51 PM

Sheet: 4/11

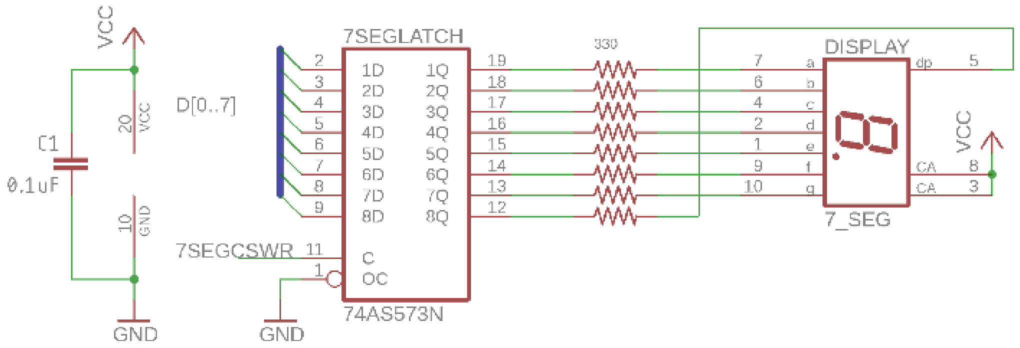


RAM

TITLE: PROJECT_V2

Document Number: REV:

Date: 5/2/2018 10:51 PM Sheet: 5/11

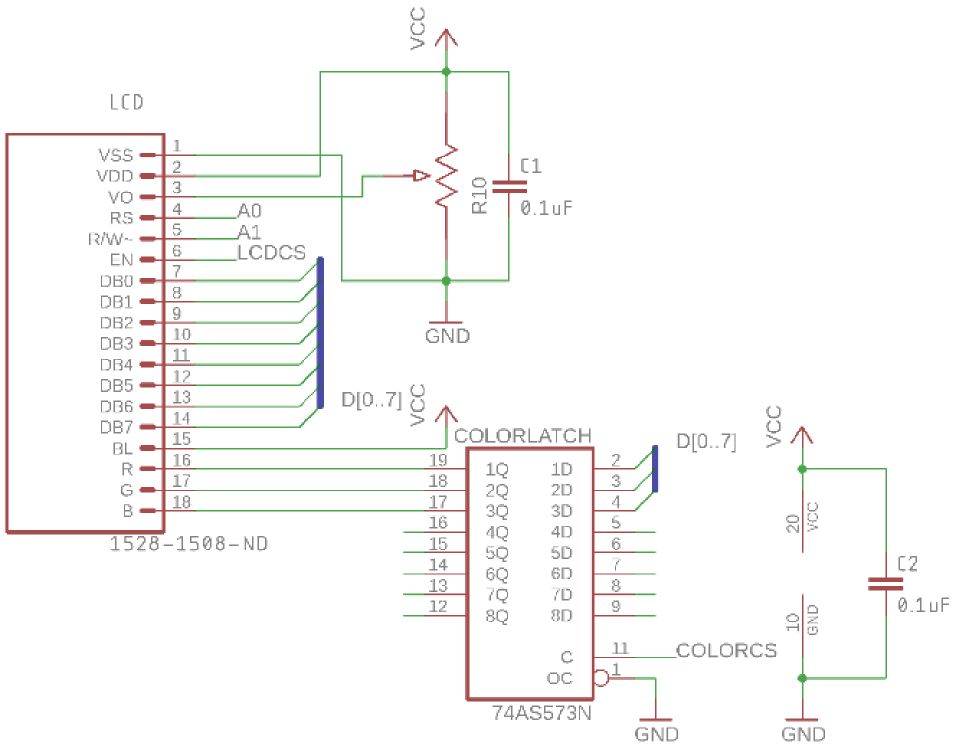


7 Segment Port

TITLE: PROJECT_V2

Document Number: REV:

Date: 5/2/2018 10:51 PM Sheet: 6/11



LCD

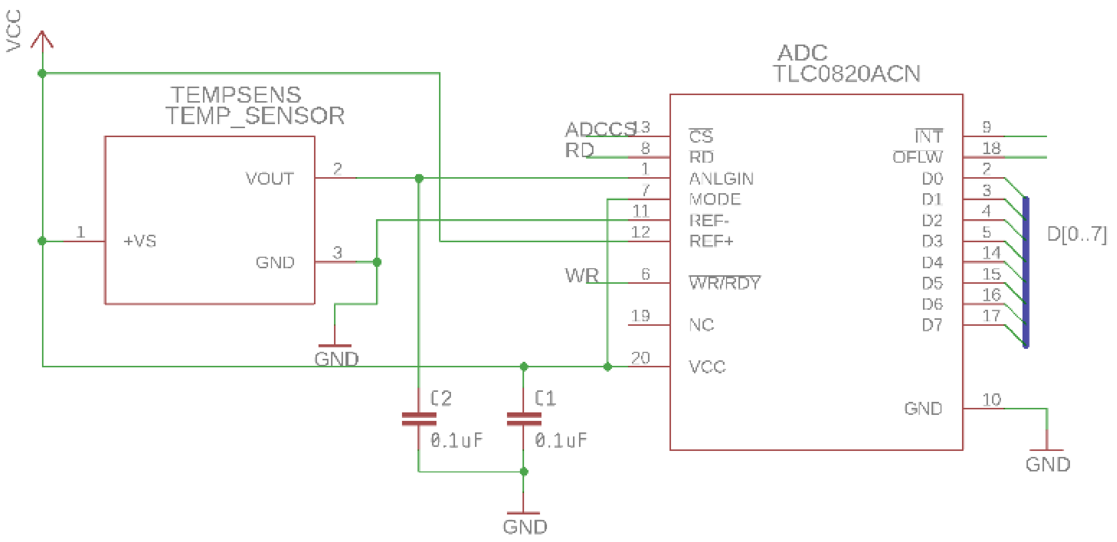
TITLE: PROJECT_V2

Document Number:

REV:

Date: 5/2/2018 10:51 PM

Sheet: 7/11

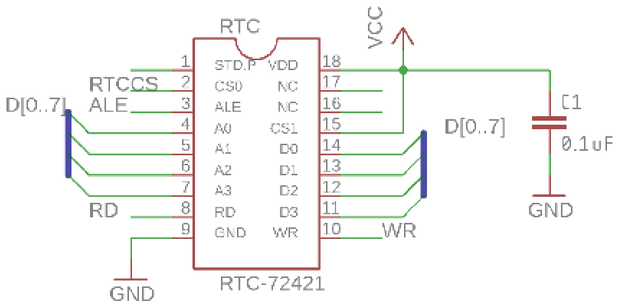


ADC

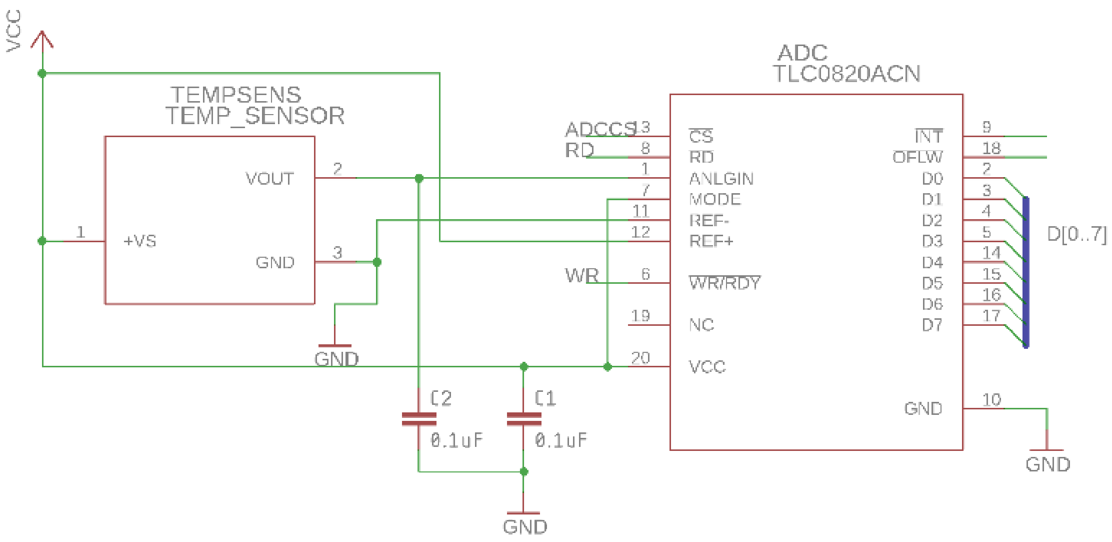
TITLE: PROJECT_V2

Document Number: REV:

Date: 5/2/2018 10:51 PM Sheet: 8/11



| | |
|---|---|
| RTC | |
| TITLE: PROJECT_V2 | |
| Document Number: | REV: |
| Date: 5/2/2018 10:51 PM | Sheet: 9/11 |



ADC

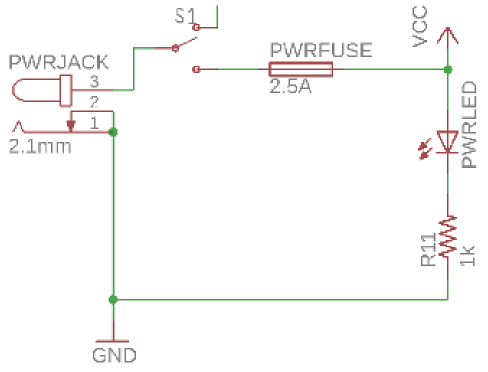
TITLE: PROJECT_V2

Document Number:

REV:

Date: 5/2/2018 10:51 PM

Sheet: 8/11



Power input

TITLE: PROJECT_V2

Document Number: REV:

Date: 5/2/2018 10:51 PM Sheet: 11/11

# Code listing

## Programs

```
/**
 * Main menu
 * Home screen of the system all programs are selected from this screen
 **/
void main_menu(void) {
    unsigned char index = 0;
    clear_LCD();
    set_LCD_line(0);
    printf_tiny("Welcome %c!", 0x01);
    set_LCD_line(1);
    printf_tiny("Pick(F) a program %c!", 0x01);
    set_LCD_line(2);
    printf_tiny("Prev(1) Next(2)");
    while(!is_pressed(KEY_F)) {
        clear_line(3);
        printf_tiny(*(program_strings + index));
        set_keypad_state_b();
        if(is_pressed(KEY_1)) {
            index = index == 0 ? num_programs - 1 : index - 1;
        } else if(is_pressed(KEY_2)) {
            index++;
        }
        index = index % num_programs;
    }
    IO_M = 0;
    state.next = programs[index];
    return;
}
```

```
/**
 * Dump Program
 * Program used to display contents of RAM
 **/
void dump_program(void) {
    // local vars
    __xdata char * start = 0;
    __idata input = 0;
    clear_LCD();
    get_address("Enter Address: ","(0000-FFFF)", &start, 0);
    clear_line(0);
    printf_tiny("Prev(1) Next(2) %c(A)", 0x00);
    do {
        // Display
        memory_dump_line(start, 4, 1);
        start += 4;
        memory_dump_line(start, 4, 2);
        start += 4;
        memory_dump_line(start, 4, 3);
        start += 4;
        do {
            // Request next step
            set_keypad_state_b();
            if(is_pressed(KEY_1)) {
                start -= 24; // Move back
                input = 1;
            } else if(is_pressed(KEY_A)) {
                input = 0xFF;
                state.next = main_menu;
            } else if(is_pressed(KEY_2)) {
                input = 1; // Move forward
            }
        } while(input == 0);
    } while(input != 0xFF);
    return;
}
```

```
/**
 * Search Program
 * Program used to search contents of RAM
 **/
void search_program(void) {
    __xdata unsigned char * start = 0;
    unsigned int count = 0;
    unsigned char search = 0;
    char found = 0;
    char input = 0;
    clear_LCD();
    get_byte("Enter Search Val: ","00-FF", &search, 0);
    do {
        count = 0;
        clear_line(1);
        printf_tiny("Searching...");
        do {
            found = (*(start) == search);
            start++;
            count--; // fixes not found bug when using continue feature
        } while( found == 0 && count > 0x0000); // Search whole memory
        clear_line(0);
        print_byte(search);
        if(found) {
            start--; // Move back to where it was last found
            printf_tiny(" Found @ ");
            print_word((unsigned int)start);
            start++; // So that we can search again
        } else {
            printf_tiny(" Not Found");
        }
        clear_line(1);
        printf_tiny("Next location(0)");
        clear_line(2);
        printf_tiny("New Search(1) %c(A)", 0x00);
        do {
            found = 2;
            set_keypad_state_b();
            // Using found as state value
            if(is_pressed(KEY_1)) {
                state.next = search_program;
                found = 1;
            } else if(is_pressed(KEY_0)) {
                state.next = search_program;
                found = 0;
            } else if (is_pressed(KEY_A)) {
                state.next = main_menu;
                found = 1;
            }
        }while(found == 2);
    } while(found == 0);
    return;
}
```

```
/**
 * Fill Program
 * Program used to edit blocks of RAM
 **/
void fill_program(void) {
    __xdata char * start;
    char val;
    unsigned char block_size;
    char index = 0;
    clear_LCD();
    get_address("Enter Address: ","(0000-FFFF)", &start, 0);
    get_byte("Enter Fill Val: ","(00-FF)", &val, 0);
    do {
        get_byte("Block Size: ","(01-FF)", &block_size, 0);
    } while(block_size == 0);
    while(block_size --> 0) {
        *(start + block_size) = val;
    }
    clear_LCD();
    set_LCD_line(0);
    printf_tiny("Fill Complete %c", 0x01);
    set_LCD_line(1);
    printf_tiny("Again?(0) %c(A)", 0x00);
    do {
      set_keypad_state_b();
      if(is_pressed(KEY_0)) {
          state.next = fill_program;
          index = 1;
      } else if(is_pressed(KEY_A)){
          state.next = main_menu;
          index = 1;
      }
    } while(index == 0);
    return;

}
```

```
/**
 * Edit Program
 * Program used to edit contents of RAM byte by byte
 **/
void edit_program(void) {
    __xdata char * start = 0;
    char index = 0;
    char newVal = 0;
    clear_LCD();
    get_address("Enter Address: ","(0000-FFFF)", &start, 0);
    do {
        index = 0;
        clear_line(0);
        printf_tiny("Current Address:");
        print_word((unsigned int)start);
        get_byte("New val: ","(00-FF)", &newVal, 1);
        *start = newVal;
        clear_line(2);
        printf_tiny("Edit Complete!");
        clear_line(3);
        printf_tiny("Edit Next?(0) %c(A)", 0x00);
        do {
            set_keypad_state_b();
            if(is_pressed(KEY_0)) {
                start++;
                index = 1;
            } else if(is_pressed(KEY_A)) {
                state.next = main_menu;
                index = -1;
            }
        } while(index == 0);
        clear_line(3);
    } while(index != -1);
    return;
}
```

```
/**
 * Move Program
 * Program used to duplicate blocks of RAM
 **/
void move_program(void) {
    __xdata char * start = 0;
    __xdata char * dest = 0;
    char index = 0;
    unsigned char block_size = 0;
    clear_LCD();
    get_address("Enter Src: ","(0000-FFFF)", &start, 0);
    get_address("Enter Dest: ","(0000-FFFF)", &dest, 0);
    do {
        get_byte("Block Size: ","(01-FF)", &block_size, 0);
    } while(block_size == 0);
    while(block_size --> 0) {
        *(dest + block_size) = *(start + block_size);
    }
    clear_LCD();
    set_LCD_line(0);
    printf_tiny("Move Complete %c", 0x01);
    set_LCD_line(1);
    printf_tiny("Again?(0) %c(A)", 0x00);
    do {
      set_keypad_state_b();
      if(is_pressed(KEY_0)) {
          state.next = move_program;
          index = 1;
      } else if(is_pressed(KEY_A)){
          state.next = main_menu;
          index = 1;
      }
    } while(index == 0);
    return;
}
```

Declarations

```
// RTC Addresses
#define S1 0x00
#define S10 0x01
#define MI1 0x02
#define MI10 0x03
#define H1 0x04
#define H10 0x05
#define D1 0x06
#define D10 0x07
#define MO1 0x08
#define MO10 0x09
#define Y1 0x0A
#define Y10 0x0B
#define W 0x0C
#define CD 0x0D
#define CE 0x0E
#define CF 0x0F
// Keys for KEYPADSTATE
#define KEY_1 (1 << 0)
#define KEY_4 (1 << 1)
#define KEY_7 (1 << 2)
#define KEY_F (1 << 3)
#define KEY_2 (1 << 4)
#define KEY_5 (1 << 5)
#define KEY_8 (1 << 6)
#define KEY_0 (1 << 7)
#define KEY_3 (1 << 8)
#define KEY_6 (1 << 9)
#define KEY_9 (1 << 10)
#define KEY_E (1 << 11)
```

```
#define KEY_A (1 << 12)
#define KEY_B (1 << 13)
#define KEY_C (1 << 14)
#define KEY_D (1 << 15)
// type state
struct state;
typedef void state_fn(void);
struct state {
    state_fn * next;
    int i;
};
// External Hardware addresses
__xdata unsigned char * __code LCD_BUSY = 0x2002;
__xdata unsigned char * __code LCD_CMD = 0X2000;
__xdata unsigned char * __code LCD_DATA = 0x2001;
__xdata unsigned char * __code LCD_COLOR = 0x4000;
__xdata unsigned char * __code SEG_DISPLAY = 0x0000;
__xdata unsigned char * __code ADC = 0x6000;
__xdata unsigned char * __code RTC = 0x8000;
// Constants and Tables
__code unsigned char LCD_LINES[] = {0x00, 0x40, 0x14, 0x54};
__code unsigned char KEYPAD_CHARS[] = { '1', '4', '7', 'F',
                                        '2', '5', '8', '0',
                                        '3', '6', '9', 'E',
                                        'A', 'B', 'C', 'D', 'X' };
__code unsigned char KEYPAD_HEX[] = {   0x01, 0x04, 0x07, 0x0F,
                                        0x02, 0x05, 0x08, 0x00,
                                        0x03, 0x06, 0x09, 0x0E,
                                        0x0A, 0x0B, 0x0C, 0x0D, 0xFF};
__code unsigned char HOME[] =   {0x00, 0x04, 0x0A, 0x11, 0x0E, 0x0E, 0x00, 0x00};
__code unsigned char SMILE[] =  {0x00, 0x00, 0x0A, 0x0A, 0x11, 0x0E, 0x00, 0x00};
__code unsigned char WORK[] =   {0x1F, 0x11, 0x15, 0x15, 0x11, 0x15, 0x11, 0x1F};
```

```c
// Special function register locations
__sbit __at (0xB5) IO_M;
__sfr __at (0x90) P1;
__sfr __at (0xF0) BREG;
// Global variables
unsigned int __xdata KEYPAD_STATE; // State of keypad since last scan
int __xdata whole_temp; // whole part of temperature
int __xdata frac_temp; // Fraction part of temperature
char __xdata last_key; // Last Keypad index since last scan
```

## Common Functions

```
void delay1ms() {
    unsigned char x = 10;
    unsigned char y = 128;
    while(x > 0){
        x--;
        y = 128;
        while(y > 0) {
            y--;
        }
    }
    return;
}
void delay(int x) {
    for(x; x > 0; x--) {
        delay1ms();
    }
    return;
char ram_test() {
    __xdata unsigned char * i = 0x0000;
    IO_M = 0;
    set_LCD_line(1);
    printf_tiny("    TESTING RAM!   ");
    set_LCD_line(2);
    // Check with 55
    BREG = 0x55;
    do{
        *i = BREG;
        i++;
    }while(i > 0x0000);
    printf_tiny("    ===");
    do {
        BREG = *i;
        if(BREG != 0x55) {
            return 0xFF;
        }
        i++;
    } while(i > 0x0000);
    printf_tiny("===");
    // Check with AA
    BREG = 0xAA;
    do{
        *i = BREG;
        i++;
    }while(i > 0x0000);
    printf_tiny("===");
    do {
        BREG = *i;
        if(BREG != 0xAA) {
            return 0xFF;
        }
        i++;
    } while(i > 0x0000);
    printf_tiny("===");
    return 0;
}
```

## Seven Segment Functions

```
void change_display(char c){
    IO_M = 1;
    *SEG_DISPLAY = c;
    IO_M = 0;
    return;
}
```

LCD Functions

```
void clear_LCD() {
    IO_M = 1;
    while(*LCD_BUSY & 0x80);
    *LCD_CMD = 0x01;
    IO_M = 0;
    return;
}
void init_LCD() {
    IO_M = 1;
    *LCD_CMD = 0b00111100;   // Function Set
    delay1ms();
    *LCD_CMD = 0b00111100;   // Function set
    delay1ms();
    *LCD_CMD = 0b00001100;   // Display On
    delay1ms();
    *LCD_CMD = 0b00000001;   // Clear Display
    delay1ms();
    *LCD_CMD = 0b00000110;   // Entry Mode set
    delay1ms();
    *LCD_CMD = 0b01000000;   // Set CG Ram
    delay1ms();
    *LCD_CMD = 0b10000000;   // Set DD Ram
    delay1ms();
    *LCD_CMD = 0b00000010;   // Set cursor home
    delay1ms();
    IO_M = 0;
    return;
}

void set_LCD_line(char line) {
    IO_M = 1;
    while(*LCD_BUSY & 0x80);
    *LCD_CMD = 0x80 | (*(LCD_LINES + line));
    IO_M = 0;
    return;
}
void clear_line(char line) {
    set_LCD_line(line);
    printf_tiny("                    ");
    set_LCD_line(line);
    return;
}
void set_LCD_cursor(char loc) {
    IO_M = 1;
    while(*LCD_BUSY & 0x80);
    *LCD_CMD = 0x80 | (loc);
    IO_M = 0;
    return;
}
void set_CG_char(char c, __code char * map) {
    unsigned char i = 0;
    c = c * 8; // Starting point for CGRAM
    IO_M = 1;
    for( i = 0; i < 8; i++) {
        while(*LCD_BUSY & 0x80);
        *LCD_CMD = 0x40 | (c + i); // Set CGRAM address
        while(*LCD_BUSY & 0x80);
        *LCD_DATA = *(map + i); // Set character code
    }
    while(*LCD_BUSY & 0x80);
    *LCD_CMD = 0x80; // Back to DDRAM
    IO_M = 0;
    return;
}
void putchar(char c) {
    IO_M = 1;
    while(*LCD_BUSY & 0x80); // Waits until the LCD is not busy
    *LCD_DATA = c;
    IO_M = 0;
    return;
}
void print_word(unsigned int word) {
    printf_tiny("%x", (word >> 12) & 0x000F ); // print address
    printf_tiny("%x", (word >> 8) & 0x000F ); // print address
    printf_tiny("%x", (word >> 4) & 0x000F ); // print address
    printf_tiny("%x", word & 0x000F ); // print address
}
void print_byte(unsigned char byte) {
    printf_tiny("%x", (byte >> 4) & 0x0F);
    printf_tiny("%x", (byte) & 0x0F);
    return;
}
```

## ADC Functions

```
void do_conversion(int * whole, int * frac) {
    unsigned int temp = 0;
    IO_M = 1; // Set IO mode
    *ADC = 0x00; // Start conversion
    delay1ms(); // Wait for conversion
    BREG = *ADC;
    IO_M = 0;   // Done with IO mode
    temp = BREG * (195.3);
    *whole = temp / 100;
    *frac = temp % 100;
    return;

}
```

RTC Functions

```
void init_RTC() {
    // Procedure for init_RTC
    unsigned char i = 0;
    IO_M = 1;   // Set to IO mode
    *(RTC + CF) = 0x04;
    *(RTC + CD) = 0x04;
    while_rtc_busy();
    *(RTC + CF) = 0x07; // Stop timer
    while(i < 0x0D) {
        *(RTC + i) = 0x00; // Load regs with 0s
        i++;
    }
    *(RTC + CF) = 0x04; // Start timer

}
char read_rtc(char reg) {
    IO_M = 1;
    while_rtc_busy();
    return (*(RTC + reg) & 0x0F);
}
void while_rtc_busy() {
    IO_M = 1;
    do {
    *(RTC + CD) = 0X00;
    *(RTC + CD) = 0X01;
    } while(*(RTC + CD) & 0x02);
    *(RTC + CD) = 0x00;
    return;
}
```

## Keypad Functions

```c
char is_pressed(int key) {
    return (KEYPAD_STATE & key) == 0;
}
char set_keypad_state_b() {
    unsigned char index = 0;
    KEYPAD_STATE = 0x0000;
    while(KEYPAD_STATE != 0xFFFF) { // Wait for blank state
        scan_keypad();
        scan_keypad();
        scan_keypad(); // Allow for bouncing
    }
    KEYPAD_STATE = 0xFFFF;
    while(KEYPAD_STATE == 0xFFFF) { // now record next keypress
        scan_keypad();
        scan_keypad();
        scan_keypad(); // Allow for bouncing
    }
    index = 0;
    while(1) {
        if((KEYPAD_STATE & (0x0001 << index)) == 0) {
            last_key = index;
            return last_key;
        }
        index++;
    }
}
char set_keypad_state_nb() {
    unsigned char index = 0;
    scan_keypad();
    scan_keypad();
    scan_keypad(); // Allow for bouncing
    while(1) {
        if((KEYPAD_STATE & (0x0001 << index)) == 0) {
            last_key = index;
            return last_key;
        }
        index++;
    }
}
void get_address(char * msg, char * rng, __xdata char ** put, char line) {
    char index = 0;
    do {
        *(put) = 0;
        clear_line(line + 2);
        clear_line(line + 1);
        printf_tiny("%s", rng);
        clear_line(line);
        printf_tiny("%s", msg);
        for(index = 3; index >= 0; index--) {
            set_keypad_state_b();
            *(put) += (*(last_key + KEYPAD_HEX) << (index * 4));
            putchar(*(last_key + KEYPAD_CHARS));
        }
        clear_line(line + 2);
        printf_tiny("Redo(F) Confirm(any)");
        set_keypad_state_b();
    } while(is_pressed(KEY_F)); // Submit confirmation

    return;
}
void get_byte(char * msg, char * rng, char * put, char line) {
    char index = 0;
    do {
        *(put) = 0;
        clear_line(line + 2);
        clear_line(line + 1);
        printf_tiny("%s", rng);
        clear_line(line);
        printf_tiny("%s", msg);
        for(index = 1; index >= 0; index--) {
            set_keypad_state_b();
            *(put) +=(*(last_key + KEYPAD_HEX) << (index * 4));
            putchar(*(last_key+KEYPAD_CHARS));
        }
        clear_line(line + 2);
        printf_tiny("Redo(F) Confirm(any)");
        set_keypad_state_b();
    } while(is_pressed(KEY_F)); // Submit confirmation
}
void scan_keypad(){
    char i = 0;
    KEYPAD_STATE = 0;
    // Column 1
    P1 = 0XFE;
    KEYPAD_STATE |= (((P1 & 0xF0) >> 4));
    // Column 2
    P1 = 0xFD;
    KEYPAD_STATE |= (((P1 & 0xF0) >> 4) << 4);
    // Column 3
    P1 = 0xFB;
    KEYPAD_STATE |= (((P1 & 0xF0) >> 4) << 8);
    // Column 4
    P1 = 0xF7;
    KEYPAD_STATE |= (((P1 & 0xF0) >> 4) << 12);
    return;
}
```

# References

Texas Instruments. TLC0820A (ACTIVE). (n.d.). Retrieved from http://www.ti.com/product/TLC0820A/technicaldocuments?HQS=TI-null-null-digikeymode-df-pf-null-wwe&DCM=yes

Atmel. AT89C55WD-24PU. (n.d.). Retrieved from https://www.digikey.com/product-detail/en/microchip-technology/AT89C55WD-24PU/AT89C55WD-24PU-ND/1118888

Epson. RTC-72421A:ROHS. (n.d.). Retrieved from https://www.digikey.com/products/en?keywords=SER3231-ND

Adafruit Industries LLC. 1528-1508-ND. (n.d.). Retrieved from https://www.digikey.com/products/en?keywords=1528-1508-ND

Atmel. AT28C256-15PU. (n.d.). Retrieved from https://www.digikey.com/products/en?keywords=AT28C256-15PU-ND

Kingbright. SA56-11SRWA. (n.d.). Retrieved from https://www.digikey.com/products/en?keywords=754-1460-5-ND

"SDCC Compiler User Guide." *Sourceforge.net*, 28 Feb. 2018, sdcc.sourceforge.net/doc/sdccman.pdf.

https://www.win.tue.nl/~aeb/comp/8051/set8051.html