Microprocessor System Design

8051 System Design

Michael Goberling

5/2/2017

Hamid Sharif-Kashani

TA: Sushanta Mohan Rakshit

Table of Contents

1 Objective

To learn the learn the required steps to design, prototype, and arrange a PCB for a functional 8051 microcontroller system with the following specifications:

- 64K of SRAM (2 chips)
- 64K of EEROM (2 chips)
- A 7-segment display
- A keypad with 16 keys
- A 4x20 LCD
- A Real-Time Clock
- An analog to digital converter with temperature sensor input

2 Background

Modeling an 8051 microcontroller system begins with understanding how to utilize system ports and address memory and I/O devices. When using external memory and devices in a microcontroller system, selecting components to communicate with the microcontroller on a bus system in a timely manner is essential to a functional design. Therefore, the first required skill for designing this system is possessing logical analysis abilities to choose decoding addresses for device chip selects or for the device appropriate data latches in a way that will extinguish any possible conflicts on the bus system.

A prominent portion of designing a microcontroller system is performing analysis on the required parts by parsing and mining provided or procured datasheets. Aspects such as package size, pin count, pin description, electrical characteristics, and device initialization are all things to consider when arranging a microcontroller system.

Following theoretical system design, visualization of such a system is necessary for implementation. Visual interpretation can be in the form of description, drawing, or schematic capture. For this system, a system schematic with a hierarchical block diagram was designed.

This schematic was modeled with the following in mind: description of the system, theoretical decoding for the system, and electrical requirements for the system.

Procuring a schematic for any given electronic design requires capture software. For the purposes of this project, Autodesk's schematic capture and PCB design software, EAGLE, was utilized. EAGLE features an intuitive system that allows users to create footprints, packages, and connect those devices with little clutter. The reason for choosing EAGLE was that the net naming convention allows for clean and easy to understand schematics that translate smoothly to PCB designs.

In the event packages or footprints for the design you have theorized are not available to you, knowledge of symbol creation in your chosen software and datasheet analysis abilities are essential to the visualization process.

Upon successful theoretical design and visual interpretation, prototyping such a design is necessary to ensure proper functionality.

For means of implementing decoding, PAL technology was utilized. This means that knowledge of VHDL or Verilog programming and testing, and flashing such chips are required for this system design. Quartus 9.1 was used as a development and testing environment for means of establishing decoding logic and generating VHDL files. ISPLever Project Navigator was used to translate the VHDL file into a JEDEC file. A Dataman and the accompanying software was used to flash the PAL chip with the generated JEDEC file.

Experience in electrical circuit design and analysis are required for optimal testing. Possessing knowledge and ability of the utilization of tools such as logic analyzers, oscilloscopes, and digital multimeters is fundamental to constructing, testing, and ensuring system functionality.

Knowledge of the instruction set and internal memory system for Intel's 8051 is required for implementing firmware for system functionality. For information regarding the instruction set, *The 8051 Microcontroller: A Systems Approach (Mazidi, Mazidi, Mckinlay)* was referenced.

An intuitive and informative development environment is also optimal for writing assembly language or C firmware for a microcontroller system. For means of developing software for this system, MCU 8051 IDE was utilized. The software features a simulation platform which allows

programmers to visualize register values, system flags, and port values. Additionally, the simulator provides port configuration and interaction with devices such as 2x20 LCDs, matrix keypads, LEDs and more. The MCU 8051 IDE environment also includes functionality for quick access of information regarding 8051 instructions such as valid register and flags that may be affected for any given instruction. Due to the simplicity of the design, system software was completed on the fly, with no unit tests performed on individual components. This was due to the timeline required for the project, and the desire to ensure cross-functionality of the components within the design.

Combining hardware design and software design is the quintessential requirement for designing any microcontroller system. It is important to understand that while theoretical hardware design may appear to be functional, it is only when software is applied to a system that realizations of design can truly occur. This is mentioned, because many changes were made to the original hardware design while attempting to prototype and breadboard the system. Decisions on changes were made mostly due to potential bus conflicts. These changes will be further explained in the *Problems Encountered* portion of this report, but for the bulk of the explanation, the final design will be described.

Arranging a PCB when a microcontroller design has been prototyped and approved for functionality is the last portion of design. For means of procuring PCB design files, EAGLE was used. Translating schematic captures to PCB designs in EAGLE is an easy task provided footprints and packages have been determined correctly from the required datasheets. While connecting rats, trace size, signal proximity, and size constraints were considered.

Upon successful PCB generation and ordering, ensuring correct connection of traces via DMM testing is crucial. Once all traces have been verified, soldering the discrete parts, mounting equipment, and chips to the board is the final frontier. Testing board functionality with soldered parts, and developing the remaining software is all that stands between you and a functional microcontroller system design.

3 Procedure

3.1 System Design

3.1.1 Mapping and Decoding

For theorizing system design, methods from the *Microcontroller System Design* course instructed by Hamid Sharif-Kashani were used. Given the system specifications of two chips of 32K RAM and ROM a decision had to be made with how to access 128k of total memory, given the 8051 only has 16 address lines. The decision was made to create separate data memory and program memory maps by including the PSEN line for decoding these chips. When PSEN goes low to access program memory, data memory will not be effects, and vice versa. Additionally, given that all address locations for RAM and ROM would be filled by these chips, P3.0 on the 8051 was delegated as the I/O line to create a third and final memory map for I/O. These memory maps can be found in the *Decoding* section of this report.

Using logical analysis, it was determined that for differentiating between the two memory chips, address line 15 could be used, because between 0000h - 7FFFh and 8000h - FFFFh only address line 15 does not change for either chip in the RAM or ROM memory map.

Using an I/O pin in the design allowed for the choice of any combination of non-conflicting address lines to be utilized for I/O device decoding, without need for RAM or ROM address consideration. For simplicity, the design features four address lines -- one per device. The address map featuring the relevant signals for decoding can be in the *Decoding* portion of this report.

Writing code for decoding involved developing a VHDL program with pin declarations that could be used within the system. Knowing that the PAL chip provided had 22 available pins for input or output, the pin declarations were the first order of business. The pin declarations are shown below, and can be verified by looking at the code in the *Decoding Code* section of this report. Red signifies input signal while blue signifies an output signal. The WR line was initially intended to be used in the decoding, but was eventually decided to be unnecessary. This will be

covered in the *Problems Encountered* section of this report. Further software discussion can be found in the *Software Discussion* section of this report. The pin declarations for the PAL chip can be found in the *Decoding* section of this report.

3.1.2 Reset Circuitry

Three components were required for designing the reset circuit of the 8051 chip: A switch button, capacitor, and a resistor. Small consideration was given in terms of the timing of the capacitor-resistor circuit given that the reset pin only needs to be high for two machine cycles before the microcontroller can detect a reset, and given that the supplied parts only featured a small amount of different types of discrete capacitors and resistors.

Calculations were performed to determine the minimum time required for the reset line to be driven low for a successful system reset given a 12MHz clock. First, a single period was determined.

$$T = \frac{1}{F} = \frac{1}{12MHz} = 83.33ns$$

With the value for one oscillator period, the amount of time needed for one machine cycle to complete could be calculated

$$MC = (Period)(12) = (83.33ns)(12) = 1us$$

With the value for one machine cycle calculated, take this by 2 to get the amount of time required for the reset pin to be low

$$Reset\ time = MC * 2 = (1us) * (2) = 2us$$

With the time required for the reset pin to be low, the minimum required capacitor value given a 10kΩ resistor was calculated.

$$T = R * C; C = \frac{T}{R} = \frac{2us}{10k\Omega} = 20nF$$

Given the previous calculation, a 10uF capacitor was chosen to be paired with the 10kΩ resistor for use in the reset circuitry. This gave the processor plenty of time to take in the reset pin given a switch button press.

3.2 Schematic Capture

Now, with the ability to select which device could be active on the bus system at any given time in software, and the reset circuitry for the 8051 determined, the schematic was designed using EAGLE. Using a hierarchical block and module system, a page was created for the main schematic that would house each device module. Meanwhile additional schematic pages for the modules were created for capturing each device including the oscillator and power components. Special consideration was made to ensure that the power connection was above the ground connection for clarity. These schematic pages can be found at the end of this report in the *Schematic* section.

The symbol and footprint used for capacitors in schematic capture were C-US, while the symbol and footprint used for resistors in schematic capture were R-US_.

The 8051 symbol for the schematic and footprint for the PCB was taken from the EAGLE libraries as part number AT89C51-24PC. The pins were checked as correct using the provided datasheet for the AT89C55WD. Port 1 connected to the keypad, P3.0 connected to the PAL, P3.1 connected to R/W on the LCD, P3.2 connected to RS on the LCD, and the rest of the Port 3 pins left no connect. /WR and /RD were connected to ROM, RAM, the ADC, RTC, and PAL accordingly.

The symbols and footprints used for ROM and RAM were taken from a library created by Collin MD Peterson, and are part number AT28C256. The footprints for ROM were edited to accompany ZIF sockets for easy access of ROM. The address and data bus were connected accordingly to these chips. For ROM, /WE was connected to VCC, and /OE was connected to /PSEN. For RAM, /WE was connected to /WR, and /OE was connected to /RD. XTAL1 was connected to the clock circuitry. RST was connected to the reset circuitry.

The symbol and footprint used for the reset switch were taken from the EAGLE libraries as part number SW-SPST-TACT-4.

The symbol and footprint used for the clock oscillator was taken from a library created by Collin MD Peterson, and is part number ECS-2200. OUT was connected to XTAL1 on the 8051.

The symbol and footprint used for the DC power jack was taken from the EAGLE libraries as part number JACK-PLUG.

The symbol and footprint used for all data latches was taken from the EAGLE libraries as part number 74HCT573N. The latch was connected to the data bus accordingly, and the output of the latch connected to the required device.

The symbol and footprint used for the seven-segment display was taken from the EAGLE libraries as part number 7-SEG-SA52-11.

The symbol and footprint used for the keypad was taken from a library created by Collin MD Peterson, and is part number AK-1604-N-BWB. Columns 1-4 were connected to port 1 pins 0-3, and Rows 1-4 were connected to port 1 pins 4-7.

The symbol and footprint used for the PAL device was taken from the EAGLE libraries as part number GAL22V10D-7LPN. Pin declarations have been outlined in the *Decoding* section of this report.

The symbol and footprint used for the LCD was taken from a library created by Collin MD Peterson, and is part number WH2004A-CFH-JT. It was decided that the LCD did not need a data latch due to its internal Enable line. R/W was connected to P3.1, and RS was connected to P3.2.

The symbols and footprints for the temperature sensor and ADC were taken from a library created by Collin MD Peterson, and are part numbers TMP36 and TLC0820AC-N. /RD was connected to /RD on the 8051 with the data lines of the ADC connected to the data bus.

Modules were created for the reset circuitry, clock circuitry, power circuitry, all RAM and ROM chips, the seven-segment display, the keypad, decoding circuitry, temperature sensing circuitry, and the RTC.

Pins do not physically need to be connected on the schematic in EAGLE. When a new net is created, a name is required for it. If two names match up, they are connected in the netlist. All devices excluding the keypad were connected via net declarations to the data bus of the 8051. The keypad was connected to port 1 of the 8051 for ease of programming and so that a data latch

did not need to be used. Chip selects, and other control signals were connected to devices accordingly and net names created accordingly. Finally, VCC and GND connections were confirmed for each module and device.

3.3 Prototyping

For testing connection between two chips on the board, a DMM was used in audible mode by touching one terminal to one pin, and the other terminal to the other checked pin. Sound would be made if there was a connection due to the small resistances in wires. For testing signals within the system while debugging, the Saleae Pro 16 Channel logic analyzer was used along with the accompanying software.

Prior to constructing circuitry, power and ground were applied to the 8051 chip and the ALE pin was analyzed using an oscilloscope to ensure that the microcontroller was functional. Additionally, power and ground were applied to the clock oscillator and to ensure that oscillation was occurring accordingly using an oscilloscope. As a final precaution to using the 8051, the reset circuitry was tested and the clock analyzed to ensure that a proper system reset was taking place.

Following confirmation that the microcontroller, clock component, and reset circuitry were functional, all components were placed on the board in succession of testing with writing, spatial sensitivity, and proximity in mind. Spatial sensitivity was account for while placing the temperature sensor and oscillator. Proximity was considered especially for placing the memory chips and the microcontroller so that long wires were not required as noise can cause various issues with electronic circuits.

First, the 8051 was placed in the top middle of the board, with the memory chips right above it, and the address latch just to the side. The data bus and ALE were connected to the address latch, and the output of the address latch connected to one memory chip, and each successive memory chip connected to the data bus of that memory chip. The same was accomplished for the upper eight bits of the address bus minus the address latch. The data bus of the microcontroller was connected to one chip as well, and broken out to each successive memory chip. Each bus connection was then checked using the DMM.

Following bus connection, the programmed PAL was then placed on the board, while signals were connected in the order outlined in the *Decoding* section of this report. Each address line, control signal, and chip select line was then checked using the DMM.

After the 8051, memory chips, and PAL chip were placed on the board, the seven-segment display and its data latch were connected to the data bus, and chip select for the seven-segment. Simple software was written to ensure the latch and connections were working correctly by blinking the decimal point of the seven-segment display. This code is further discussed in the *Software Discussion* portion of this report.

After confirming that interfacing with the seven-segment was functional, the LCD was connected to the circuit and its appropriate signals. The LCD was connected this early, because it would be impossible to test the RTC and ADC without it. First, the power connections were tested, and code was written to initialize the LCD using the datasheet provided. Diagnosing issues with the LCD took a large portion of time due to electrical problems. These considerations will be discussed in the *Problems Encountered* portion of this report.

With the LCD interfacing, the keypad was set up on port 1 of the 8051, and subsequently a subroutine from *The 8051 Microcontroller: A Systems Approach* (Mazidi, Mazidi, Mckinlay) was used to get a byte in ascii of the key press into the accumulator. This code can be found in the *Source Code* section under the subroutine *PROMPTKEYPAD* or the modified subroutine *POLLKEYPAD.* Reference to book in the *References* section.

Following keypad integration, the RTC and ADC with sensor were connected to the system and subsequently tested using developed code. For the RTC, all registers we set to zero upon wakeup of the system, and subsequently read in a loop. The same was applied to the ADC, only no initial registers were required to be set for the ADC.

With functionality of all components tested and debugged, the system, its decoding, and all electrical considerations proved to be sufficient for the design requirements. Software was developed for the system on the fly for testing. No unit tests were developed for specific components. Because the parts need to all work together for the final design, no benefit was seen from skipping around writing the firmware from the beginning.

3.4 PCB Design

The PCB design files were generated using Autodesk's EAGLE. With the schematic completed, and all footprints created or procured, part placement began. Pad sizes were analyzed and confirmed for each part and each datasheet. A square of 9in. x 9in. was drawn for the board initially to provide enough space for all components, to minimize the possibility of electrical issues, and because no casing or design restraints were placed on the size of the board.

Power traces were made .05 in. thick to accommodate how important those signals are. The output of the clock trace was made .032 in. thick to accommodate the importance of that signal as well. Finally, all other traces were made .012 in. thick for spatial reasons.

To begin, power components were set in the top left, so that power is applied at the top left of the board as it made organizational sense. For clarity and organization, the top copper layer of traces flow in a vertical fashion, while the bottom copper layer of traces flow in a horizontal fashion.

The 8051 is placed on the left side of the board, and all subsequent memory devices placed to the right of the controller continuing the flow of the board from top left to bottom right.

The LCD and matrix keypad were placed on the bottom left and bottom right portion of the board respectfully, so that the components were out of the way of the rest of the circuit. Outlines were added for these parts so that, again, they were out of the way of other components.

The temperature sensor was placed at the top of the board so that it was out of the way of any sort of electrical interference.

The clock oscillator was placed away from the temperature sensor and close to the microcontroller for the same interference reasons.

The drill files were generated using a CAM Processor job file acquired from SparkFun. This job generated the top copper, bottom copper, top silk, bottom silk, top soldermask, bottom soldermask, and drill files. The manufacturer used was Bay Area Circuits, because I had good experience with them. The final PCB design can be found in the *Appendix* of this report.

4 Source Code Discussion

Software design was centered around assembly language using the 8051 instruction set. As mentioned in the *Background* portion of this report, MCU 8051 IDE was used for development and testing of system source code due to its useful simulation features, which provide clarity in terms of internal memory allocation and system flag statuses.

For baseline knowledge, to get the desired values onto the address and data bus, the instruction MOVX is used. Often, the indirect addressing value is used with the DPTR register to get a 16-bit wide address onto the address bus, or with R0 to get an 8-bit address onto the address bus. This is primarily used to achieve a proper chip select for a specific device prior to loading a value onto the data bus with whatever is in A.

An iterative delay system was conceived and tested for timing using the MCU 8051 IDE simulator. A simple 1ms delay was written, called, DELAY_1MS so that other time delays could be implemented. This function simply uses two loops with two registers evaluated using the DJNZ instruction to eat up exactly 1ms. Every other delay written simply called the DELAY_1MS function an iterative amount of times relative to the desired length of the delay. This function is useful in LCD initialization and various message displaying applications for clarity on the LCD.

For sending commands to the LCD module, a function COMNWRT was written. This subroutine simply takes the command previous in A before the call, clears the RS and RW lines on the LCD to signal a command is about to be sent to the module, and then puts the address of the LCD and the accompanying data on the bus system. This subroutine is accompanied by a small 1ms delay to ensure functionality.

The LCD is first initialized using a subroutine called LCD_INIT. This subroutine was taken from the datasheet for the LCD and works by waiting for 50ms, and successively sending commands to the LCD using the previously explained subroutine to accomplish setting the function set to 8-bit, 2-line, and 5x8 dots using 38h, setting the display to on using 0Ch, setting the DDRAM address to 00h to start, and finally setting the display to normal US cursor printing using 06h.

For printing strings and characters to the LCD, two functions PRINTCHAR, and PRINTSTRING, were taken from a previous *Assembly Language Programming* course. PRINTSTRING will take the string pointed at by the DPTR before the call and print it by iteratively calling PRINTCHAR. PRINTCHAR prints the character currently in A to the LCD by loading the address for the LCD into R0, setting the I/O line high, and moving the address onto the address bus, and the character onto the data bus. PRINTSTRING iteratively calls PRINTCHAR until a zero is detected at the data pointer. Hence, all strings end with \0 to signify that a string declaration has ended.

Subroutines were written with the DDRAM addresses of the desired locations in mind on the LCD. Because overflow on the 4x20 LCD goes from line 1 to line 3, this was required. Additionally, to print the temperature in the top right, the command was calculated and pin-pointed to fit right with no character spaces left to the right. These functions work by taking the DDRAM addresses of each character location on the datasheet and setting the 8^{th} bit high to signify a set DDRAM command to the LCD.

For keypad interfacing, PROMPTKEYPAD and POLLKEYPAD were written. Taken from the book *The 8051 Microcontroller: A Systems Approach*, the two subroutines monitor port 1 until one of the row lines has gone low. The subroutine then grounds each column to check for a matched value in a LUT placed at the end of the program. The difference between PROMPT and POLL is that the prompt function will wait for a keypress, while the POLL function will continue out of the subroutine if no change on the keypad is detected.

Since acquiring a byte of information at a time is incredibly important throughout this project, a subroutine called GETBYTE was written to obtain a packed BCD byte from the user where the top nibble is the first keypress entered, and the bottom nibble is the second keypress entered. This subroutine uses ascii detection to see if the value is a number of a character, and converts it into a single hex value depending on the keypress by using bit-masking.

Another important aspect of the source code is overflow and underflow checking. Because the subroutines that use two byte inputs from the user obtain these by storing two bytes in two separate registers, whenever a decrement of increment is performed in the software, overflow or underflow detection is implemented by checking the lower byte. 00h is checked for incrementing

as all incrementing is done before checking, and if the lower byte is every 00h, that means that it WAS FFh. If this decision is found, the higher byte associated with that lower byte is incremented. The same is true for decrementing, only instead of 00h being checked, it is FFh that is checked. This is due to the fact that if a lower byte value was 00h and it was decremented, it would be FFh. A value coming into a subroutine can never be FFh and be mis-checked, because pre-decrementing takes place in all parts of the firmware. When this is detected, the program decrements the higher byte associated with that lower byte as well.

For security purposes, users are first requested to press one to login. Once a user has pressed one, they are prompted to enter a 4-digit passcode allocated to them by myself, with three correct tries allowed. The software then accepts four nibbles as keypresses in the form of ascii characters and saves the high byte into R1 and R2, and saves the low byte into R3 and R0. The system then decides if the passcode is a stored LUT at the end of the software. The way the LUT table works is that the high and low bytes are stored in succession, and checked using the zero flag on the accumulator. The passwords are stored in succession, and therefore each password has a verifiable profile, dependent on how far into the LUT the software goes. This profile is stored in R5, and this register is incremented each time a password is checked. This register is used for profile identification later. If it is, then the user is granted access. If it is not, then the user is required to enter in another password, and the number of tries is decreased, which is stored in R6. If the number of tries reaches 0, then the software jumps to LOCKOUT, which simply runs in a loop until power is taken from the system. Once the user gains access the subroutine jumps to CHECKPROFILE, where it checks the value of R5, and prints a welcome message to the accompanying profile. They can now access the main portion of the code.

This software also features a confirmation message each time a two byte value is entered into the system. This was implemented by using the CJNE instruction to check for ‘A’ to submit the value, or ‘D’ to redo the submission. If A is pressed, the program proceeds, otherwise if D is pressed, the program returns to the prior PROMPTKEY call, and restarts the submission process.

The software is built on a looping main subroutine called MONITOR or, depending on the condition of the software MONITORMENU. This subroutine calls other subroutines to get the temperature, print the temperature, and to get and print the values in the RTC. Additionally, this subroutine calls the POLLKEYPAD subroutine, which grabs a keypress from the user in the

form of the ascii byte of the key they pressed. The software then uses the CJNE instruction with A to compare the options of B for move, E for edit, F for find, D for dump, or 1 for logging the user out in their ascii forms. If no accepted key is pressed, the program loops back to keep monitoring the status of the RTC and ADC.

A simple function to test the seven-segment display was written called WAKEUP. This subroutine moves the address of the seven-segment data latch to R0, moves the value for blinking the decimal point into A, then moves that information to the address and data bus. The program completes this cycle of flashing the decimal point three times before returning to the main subroutine.

The RTC is initialized at each login through a subroutine called RTC_INIT. This subroutine was taken from the RTC datasheet provided to the class, and works by setting all internal seconds, minutes, and hours registers to 0 by sending their addresses onto the address bus, and 00h onto the data bus, and then starting the count by sending 00h to 4Fh, and sending 00h to 4Dh.

GETTEMP simply loads the address of the ADC into R0, then moves the value on the data bus into A using the MOVX instruction. GETRTC functions similarly, but instead also prints the value of each register by converting it to ascii and then calling PRINTCHAR.

HEXTOASCII converts the HEX value obtained from GETTEMP into three separate ascii values stored in R7, R6, and R5 in order of top digit to bottom digit if looking at the temperature “100”.

PRINTTEMP sets the LCD to the desired DDRAM location on the LCD, and then prints the ascii values obtained from HEXTOASCII in succession. A degree symbol is printed after by sending D8h to the LCD, and then a capital F with 45h.

PROMPTMOVE, PROMPTDUMP, PROMPTEDIT, and PROMPTFIND will print strings to prompt the user to enter the source, block size, and destination required for the relevant subroutine. All of these subroutines take advantage of the GETBYTE subroutine, and subsequently store 16-bit values in two 8-bit registers for later use in the subroutines.

MOVE will take the values entered in PROMPTMOVE, and move the amount of block size from the source address to the destination address in external memory. Because this subroutine

uses two bytes in the form of two single byte addresses, it performs overflow detection for those registers for incrementing or decrementing. The software continues to move values while incrementing the source and destination address and decrementing the block size until the block size is zero.

DUMP will take the values for source and block size form PROMPTDUMP and display the memory dump of that location for the duration of the block size. This program is the most memory intensive as it utilizes every possible general purpose register the 8051 offers. R2 and R3 represent the block size, R4 and R5 represent the current address, R0 tracks the number printed to a line, because the 4x20 LCD does no overflow lines in an appropriate manner, R6 tracks the current page number, R7 tracks the amount printed to the LCD, and R1 is a temporary register used for printing a packed BCD value to the LCD. Because the block size cannot be zero, dump works by initially printing the location entered by the user, then incrementing the low byte of the data pointer each time a print is successful. The number of bytes printed to a line is then checked, if it is not 6, then the program continues printing, otherwise it jumps to the second line. For formatting purposes, the LCD can only print 12 bytes at a time to the display. For this reason, the second conditional is to see if the LCD is full, and if it is, to see what the user wants to do. If either the LCD is full, or the block size has been reached, the program jumps to DONE, where the user is prompted with a decision to '2' Exit the program and return to main, '0' continue to the next page, or '1' to go to the previous page. To continue to the next page, the user must have block size left to print. The program checks the higher and lower bytes of the block size to see if there is anything left. If there is, then the program simply jumps back to print, because the next address was pre-incremented, and the page number is incremented. If there is not, the program does not accept the key press, and instead jumps back for another keypress. If the user decides to go to the previous page, the page number is then checked. If it is zero, the keypress is invalid, and the program jumps back to get another keypress, if it is not zero, then the amount printed to the LCD currently, and 12 are subtracted from the address with underflow detection, and the program returns to print.

EDIT takes the starting location entered by the user in the form of two bytes, and prompts the user to replace the byte at that location in external memory. Using GETBYTE, the user enters a byte to the keypad, and then the value is then moved into external memory using MOVX. After

the byte has been entered, the location is updated, the new value displayed at the location, and the user is prompted to ‘1’ exit the program, or ‘0’ continue to the next address to edit that byte.

FIND takes the starting location entered by the user in the form of two bytes and takes the block size entered by the user in the form of two bytes, and takes in a single byte to check memory against. The subroutine checks external memory for the entered byte value by loading the value at the current address into A, and subtracting is with the stored entered byte value. Using JZ, we can see if the accumulator is zero. If it is not, the values do not match, and the program continues, otherwise, the value is found, the subroutine prints that the byte was found, and then prints the location the byte was found at. If the value is not found, the subroutine continuously checks until the block size has run out from decrementing. The address checked is incremented at the end of each check as well.

5 Hardware Discussion

Note: Hardware has been explained thoroughly throughout the *Prototyping*, *Schematic*, and *Background* stages of this report.

The generated PCB is a 9 in. x 9 in. board. The system runs on a 5V DC input from the AC converter coupled with a DC jack supplied with the project. A switch determines the flow of power at the beginning of the circuit. Upwards, and power is on - downwards, and power is off. A fuse is placed shortly after the switch for surge protection at a small scale. Decoupling capacitors are placed at the ground and VCC connections of each component to ensure decoupling of AC signal. A capacitor has been placed across the GND and OUT terminals of the temperature sensor to ensure a smooth reading.

The designed system features the 8-bit 12MHz AT89C55WD Intel 8051 microcontroller, and 64k of ROM and RAM accessed externally. The decoding for these chips has been laid out in the *Decoding* section at the end of this report. The 8051 features eight general purpose 8-bit registers R0-R7, a single 16-bit register DPTR (split into two 8-bit registers DPH and DPL), and four 8-bit ports, with only Port 1 totally accessible without any other signals attached to it. An address latch was connected to the multiplexed AD0-AD7 to generate address lines in line with the pulse of the ALE signal of the 8051, which pulses high at the beginning of each machine cycle. XTAL1

is connected to the oscillator output for clock generating purposes. RST is connected to the output of the reset circuitry for reset purposes. EA/VPP is tied low, because external memory access is taking place. P3.0 is used for I/O decoding, P3.1 is connected to R/W on the LCD, P3.2 is connected to RS on the LCD, P3.3-P3.5 are not used, /RD and /WR are connected to memory, the ADC, and RTC to ensure proper writing and reading from those components.

Each ROM and RAM chip requires 15 address lines to access the full extent of memory within. Likewise, to access these locations 8-bit registers were repurposed within the software development of this system to obtain multiple 16-bit values. For RAM /OE is connected to /RD, /WE is connected to /WR. For ROM, /OE is connected to /PSEN, and /WE is connected to VCC, because writing to ROM is never a good idea while the system is on. Each chip select is connected to the chip select outputs on the PAL.

The PAL decoding chip was programmed using Quartus 9.1 to generate VHDL files, ISPLever Project Navigator to convert those VHDL files into JEDEC files, and a Dataman and its accompanying software to flash the PAL chip. Pin Declarations can be found in the *Decoding* section of this report.

The LCD module is a 4x20 80-character display module with backlight coloring and character opacity via a potentiometer. The backlight is blue by pulling the B pin to ground. The RS line determines if data or command registers are being selected, and the R/W line determines if the software is reading from or writing to the LCD module. The Enable line is connected to the chip select of the LCD on the PAL.

The 4x4 matrix keypad is connected to Port 1 as the sole device on this port. Columns 1-4 are connected to P0-P3 on Port 1, and Rows 1-4 are connected to P4-P7 on Port 1. This keypad can generate characters between 0-F by using software to pull down and check each column.

The ADC is a TLC0820AC-N, an 8-bit resolution analog to digital converter that has an ANLG_IN connection to the temperature sensor, 8 bits of data to the data bus, and a /RD line connected to the /RD line of the 8051. Chip select is connected to the chip select on the PAL.

The RTC is an Epson RTC72421 with registers for seconds, minutes, and hours. The ALE pin of the RTC is connected to the ALE pin of the microcontroller. Address lines zero through three are

connected to address lines zero through three on the microcontroller. /RD is connected to /RD on the microcontroller, /WR is connected to /WR on the microcontroller, and the 4-bit RTC output is connected to the data bus. Chip select is connected to the chip select on the PAL.

6 Problems Encountered

Problems were chiefly encountered during the prototyping stage of this design. The most prominent problem was an electrical problem regarding the first potentiometer connected to the LCD module. Software solutions were implemented time after time for initialization of the LCD, but to no avail. All connections were checked and verified, and the problem persisted. Eventually, the potentiometer was checked for resistance range, and was stuck at its highest available resistance. This was checked with a DMM. The theory is that the drop across the potentiometer was too large for the voltage range to be met for the LCD module. After the potentiometer was replaced, the LCD module displayed the test string used at the time, and the system functioned.

In the PCB design stage, following the soldering of parts to the board, there was a time where the LCD was displaying unintelligible characters. Having experience with checking the connections on each chip throughout the project, a DMM was used to check each connection. It was found that address line 12 on the first ROM chip was not connected to the microcontroller appropriately via a bad solder joint. The solder joint was quickly fixed, and the system worked just as it had before.

Mid-project, the data bus and port 1 configuration was changed to the current state. Originally, all devices were on port 1, and all devices had a data latch before them to make them into ports. After running into bus conflicts by trying to make the keypad work on a data latch, the switch was made mid-semester to put all devices beside the keypad on the data bus.

Mid-project, the PCB design and schematic capture software was switched from ORCAD Capture and PCB Design to Autodesk's EAGLE. This switch was made because I could not be in the lab for spring break and shortly thereafter, and due to monetary constraints, could not purchase ORCAD's software out of pocket. EAGLE is available to students for free, and thus I quickly downloaded it, and began transferring my work from ORCAD to EAGLE as quickly as

possible. It proved to be the better decision, because EAGLE has easier net-naming and netlist generation as netlist generation happens on the fly as soon as a new net is named, and thus the translation from schematic to PCB design was accomplished much quicker.

7 Conclusions

7.1 Summary

In conclusion, the steps required to design, prototype, and arrange a PCB design for an Intel 8051 microcontroller system given the requirements outlined in the *Objectives* section of this report have been learned and completed. All hardware is functional and integrates with the written software accordingly. All objectives have been met as all devices are interfaced with at one point or another through the operating system's run-time. Engineering practices have been used to test, prototype, and design the board by way of time planning, using logic to develop decoding for each device, and by asking questions in a timely manner to instructors and peers.

7.2 Future Work

Future PCB designs will be much smaller in size. The spacing consideration for this design were due to the time constraints of the semester, and to ensure the functionality of hardware so that software could be developed and my programming skills could be displayed.

Completing the computer and electronics related work early would provide a time window where I could focus on the mechanical aspects of the build. Currently, the build is an exposed PCB, which is not appealing. I would like to be able to focus on a device case, device carrying case, or extensive mounting equipment for the keypad and LCD.

Towards the beginning of the project, I was wary of asking questions to those around me and my instructors, which led to taking care of some of my problems taking longer than they should have. In the future, I will be more diligent about asking for help in areas that I need it, and of course returning the favor to those who are in need as well.

8 References

Mazidi, M. A., Mazidi, J. G., & McKinlay, R. D. (2013). *The 8051 microcontroller: a systems approach*. Boston: Pearson.

Datasheets supplied via BlackBoard by instructors for:

- *Intel 8051 (AT89C55WD)*
- *ECS-2200*
- *RAM and ROM*
- *Seven Segment Display*
- *74HCT573N (data latch)*
- *AK-1604-N-BWB (matrix keypad)*
- *PAL22V10 (decoder chip)*
- *WH2004A-CFH-JT(4x20 LCD)*
- *TLC0820AC-N (analog to digital converter)*
- *TMP36 (temperature sensor)*

9 Appendices

9.1 Source Code

```
;*******************************************
;*Author: Michael Goberling                *
;*Course: 4330 Microprocessor Design       *
;*Assignment: 8051 Source Code             *
;*Due date: 5/2/17                         *
;*Revision: 1.4                            *
;*******************************************


            org 0h
            sjmp    start
;=========================================================
;| Data equates                                          |
;=========================================================
io_temp     EQU     10h
io_sevenseg EQU     20h
io_rtc              EQU     40h
io_lcd              EQU     80h

lcd_clear   equ     00000001b
lcd_home    equ     00000010b
lcd_fn_set  equ     00111100b
lcd_onoff_cntl      equ     00001111b
lcd_entry_set equ   00000110b
lcd_ddram   equ     10000001b

RS          EQU     P3.2
RW                  EQU     P3.1

keypad              EQU     P1


;=============================================================================
;| Start of the program                                                      |
;=============================================================================
start:
            LCALL   LCD_INIT                    ;LCD initialization

            MOV     R0, #IO_SEVENSEG            ;clear 7 segment
            MOV     A, #11111111B
            LCALL   IOTOGGLE

            LCALL   wakeUp                      ;7 segment initialization(3 decimal place blinks)
relogin:    LCALL   login                       ;waits for a user to press 1 to continue
            LCALL   getPasscode                 ;user enters passcode that allows them access
            LCALL   RTC_INIT                    ;initialize the RTC so that login time is kept

monitormenu:        LCALL   displayMenu         ;Display menu options

monitor:
            ;LCALL  flash7seg                   ;quickly flash status of 7 segment
            LCALL   getTemp                     ;temperature in A now
            LCALL   getRTC                      ;update time by reading RTC regs
            LCALL   hexToAscii
            LCALL   printTemp                   ;print values in R6 and R7 to LCD
```

```
            ;42h = move
            ;44h = dump
            ;45h = edit
            ;46h = find
            LCALL   pollKeypad

            CJNE    A, #42H, compare1           ;check for move, or 'B'
            LCALL   CLEAR_LCD                   ;if found, clear lcd
            MOV     DPTR, #test1                ;print selection string
            LCALL   printString
            LCALL   halfseconddelay             ;leave it up for some time
            LCALL   CLEAR_LCD                   ;clear lcd for entering menu
            LCALL   promptMove
            LCALL   MOVE                        ;go to main move function
            sjmp    monitormenu                 ;jump back
compare1:
            CJNE    A, #44H, compare2           ;check for dump, or 'D'
            LCALL   CLEAR_LCD
            MOV     DPTR, #test2
            LCALL   printString
            LCALL   halfseconddelay
            LCALL   CLEAR_LCD
            LCALL   promptDump
            LCALL   DUMP
            sjmp    monitormenu
compare2:
            CJNE    A, #45H, compare3           ;check for edit, or 'E'
            LCALL   CLEAR_LCD
            MOV     DPTR, #test3
            LCALL   printString
            LCALL   halfseconddelay
            LCALL   CLEAR_LCD
            LCALL   PROMPTEDIT
            LCALL   EDIT
            SJMP    monitormenu
compare3:
            CJNE    A, #46H, compare4           ;check for find, or 'F'
            LCALL   CLEAR_LCD
            MOV     DPTR, #test4
            LCALL   printString
            LCALL   halfseconddelay
            LCALL   CLEAR_LCD
            LCALL   PROMPTFIND
            LCALL   FIND
            SJMP    monitormenu
compare4:
            CJNE    A, #31H, compare5           ;check for logout, or '1'
            LCALL   CLEAR_LCD
            MOV     DPTR, #goodbye
            LCALL   printString
            LCALL   halfseconddelay
            LCALL   CLEAR_LCD
            LJMP    relogin
compare5:
            CJNE    A, #31H, monitorLJMP        ;check for logout, or '1'
            LCALL   CLEAR_LCD                   ;implemented monitorLJMP for 8-bit
            MOV     DPTR, #sevensegmsg          ;address issues
            LCALL   printString
            LCALL   halfseconddelay
            LCALL   CLEAR_LCD
            LJMP    sevenseg
            LJMP    monitormenu
```

```
monitorLJMP:        LJMP    monitor

FOREVER:  SJMP    FOREVER
;=========================================================================
;| prompt for value between 30h and 7Fh to not mess with registers      |
;=========================================================================
promptMove:

bdata:              Lcall   clear_lcd
            mov     DPTR, #bSource          ;print menu message
            LCALL   printString

            MOV     DPTR, #DIGITMSG
            LCALL   PUT_LINE2
            LCALL   PRINTSTRING

            LCALL   PUT_LINE3_CB
            LCALL   GETBYTE                 ;2 byte block size will be in R1

            mov     A, R1
            mov     R2, A                   ;XX00H IN R2

            LCALL   GETBYTE

            MOV     A, R1
            MOV     R3, A                   ;00XXH IN R3

CONT27:             MOV     DPTR, #VERIFYINPUT
            LCALL   PUT_LINE4
            LCALL   PRINTSTRING
            LCALL   PROMPTKEYPAD

            CJNE    A, #41H, CONT26         ;IF THEY HIT 'A' AND ACCEPT
            LJMP    REDO                    ;MOVE FORWARD

CONT26:             CJNE    A, #44H, CONT27 ;IF THEY HIT 'D' AND WANT TO REDO
            LJMP    BDATA

REDO:               LCALL   clear_lcd
            mov     DPTR, #bblock
            LCALL   printString

            MOV     DPTR, #DIGITMSG
            LCALL   PUT_LINE2
            LCALL   PRINTSTRING

            LCALL   PUT_LINE3_CB
            LCALL   GETBYTE                 ;Source address will be in R1
            mov     A, R1
            mov     R4, A                   ;XX00H IN R4

            LCALL   GETBYTE
            MOV     A, R1
            MOV     R5, A                   ;00XXH IN R5

CONT29:             MOV     DPTR, #VERIFYINPUT
            LCALL   PUT_LINE4
            LCALL   PRINTSTRING
            LCALL   PROMPTKEYPAD

            CJNE    A, #41H, CONT28         ;IF THEY HIT 'A' AND ACCEPT
```

```
            LJMP    CONT32                  ;MOVE FORWARD

CONT28:     CJNE    A, #44H, CONT29         ;IF THEY HIT 'D' AND WANT TO REDO
            LJMP    REDO

CONT32:     CJNE    R5, #0, CONT6
            CJNE    R4, #0, CONT6           ;CANT HAVE 0 AS THE BLOCK SIZE
            SJMP    REDO

CONT6:      LCALL   clear_lcd
            mov     DPTR, #bDest
            LCALL   printString

            MOV     DPTR, #DIGITMSG
            LCALL   PUT_LINE2
            LCALL   PRINTSTRING

            LCALL   PUT_LINE3_CB
            LCALL   GETBYTE                 ;Destination address now will be in R1
            mov     A, R1
            mov     R6, A                   ;XX00H IN R6

            LCALL   GETBYTE
            MOV     A, R1
            MOV     R7, A                   ;00XXH IN R7

CONT31:     MOV     DPTR, #VERIFYINPUT
            LCALL   PUT_LINE4
            LCALL   PRINTSTRING
            LCALL   PROMPTKEYPAD

            CJNE    A, #41H, CONT30         ;IF THEY HIT 'A' AND ACCEPT
            LJMP    ENDPROMPTMOVE           ;MOVE FORWARD

CONT30:     CJNE    A, #44H, CONT31         ;IF THEY HIT 'D' AND WANT TO REDO
            LJMP    CONT6

ENDPROMPTMOVE:
            RET
;===============================================================================
;| Copy a block of memory to another location                                  |
;===============================================================================
;SOURCE R2R3H
;BLOCK    R4R5H
;DEST     R6R7H

MOVE:
            CLR     P3.0

back:
            mov     DPH, R2
            mov     DPL, R3                 ;DPTR NOW CONTAINS SOURCE ADDR
            movx    A, @DPTR

            mov     DPH, R6
            mov     DPL, R7                 ;DPTR NOW CONTAINS DEST ADDR
            movx    @DPTR, A

            inc     R3                      ;INC LOWER BYTES
            inc     R7

            DEC     R5                      ;DEC LOWER BYTE OF BLOCK SIZE
```

```
            CJNE    R3, #00H, CONT4                 ;IF LOWER BYTE OF SOURCE IS 00H AFTER INC
            INC     R2                              ;INC HIGH BYTE OF SOURCE
CONT4:
            CJNE    R7, #00H, CONT5                 ;IF LOWER BYTE OF DEST IS 00H AFTER INC
            INC     R6                              ;INC HIGH BYTE OF DEST
CONT5:
            CJNE    R5, #0FFH, CONT3                ;IF R7 IS FFH AFTER DEC, THEN DEC HIGH BYTE
            DEC     R4                      ;HERE
CONT3:
                                                    ;ELSE CONTINUE THE PROGRAM
            CJNE    R4, #0, BACK                    ;IF HIGH BYTE IS NOT ZERO, CONTINUE
            CJNE    R5, #0, BACK                    ;IF LOW BYTE IS NOT ZERO, CONTINUE
                                                    ;ELSE, IF BOTH ARE ZERO, THEN DONE
            LCALL   clear_lcd
            mov     DPTR, #bdone
            Lcall   printString
            LCALL   halfseconddelay

            RET
;=========================================================================
;| prompt for values to show a given block of memory                      |
;=========================================================================
promptDump:
            LCALL   clear_lcd
            mov     DPTR, #bsource
            LCALL   printString

            MOV     DPTR, #DIGITMSG
            LCALL   PUT_LINE2
            LCALL   PRINTSTRING

            LCALL   PUT_LINE3_CB
            LCALL   GETBYTE                         ;Start address will be in R1
            mov     A, R1
            mov     R4, A                           ;XX00H IN DPH (R4)

            LCALL   GETBYTE
            MOV     A, R1
            MOV     R5, A                           ;00XXH IN DPL (R5)

CONT35:             MOV     DPTR, #VERIFYINPUT
            LCALL   PUT_LINE4
            LCALL   PRINTSTRING
            LCALL   PROMPTKEYPAD

            CJNE    A, #41H, CONT34                 ;IF THEY HIT 'A' AND ACCEPT
            LJMP    BSIZEPROMPT                     ;MOVE FORWARD

CONT34:             CJNE    A, #44H, CONT35         ;IF THEY HIT 'D' AND WANT TO REDO
            LJMP    PROMPTDUMP

BSIZEPROMPT:        LCALL   clear_lcd
            mov     DPTR, #bBlock
            LCALL   printString

            MOV     DPTR, #DIGITMSG
            LCALL   PUT_LINE2
            LCALL   PRINTSTRING

            LCALL   PUT_LINE3_CB
```

```
            LCALL   GETBYTE
            mov     A, R1
            mov     R2, A                       ;XX00H WILL BE IN R2

            LCALL   GETBYTE
            MOV     A, R1
            MOV     R3, A                       ;00XXH WILL BE IN R3

CONT37:             MOV     DPTR, #VERIFYINPUT
            LCALL   PUT_LINE4
            LCALL   PRINTSTRING
            LCALL   PROMPTKEYPAD

            CJNE    A, #41H, CONT36             ;IF THEY HIT 'A' AND ACCEPT
            LJMP    CONT33                      ;MOVE FORWARD

CONT36:             CJNE    A, #44H, CONT37     ;IF THEY HIT 'D' AND WANT TO REDO
            LJMP    BSIZEPROMPT

CONT33:             CJNE    R2, #0, CONT14
            CJNE    R3, #0, CONT14
            LJMP    BSIZEPROMPT
CONT14:
            LCALL   CLEAR_LCD
            RET
;===============================================================================
;| Show the contents of a given block of memory                                |
;===============================================================================
;BLOCK SIZE:            R2R3H
;CURRENT ADDR:          R4R5H
;# Printed to Line:     R0H
;PAGE #:                R6H
;# PRINTED TO LCD:     R7H
;Temp reg for printing: R1H

Dump:
            CLR     P3.0
            MOV     R6, #0                      ;MAKE PAGE # 0 AS ORIGIN
            MOV     R7, #0                      ; # PRINTED TO LCD to 0
            MOV     R0, #0

loop:
            MOV     DPH, R4
            MOV     DPL, R5
            MOVX    A, @DPTR                    ;(R4R5h)
            MOV     B, A
            anl     A, #0f0h
            rr      A
            rr      A
            rr      A
            rr      A
            mov     R1, A                       ;To save the raw value
            CLR     C
            SUBB    A, #0Ah                     ;check if letter
            jnc     letter3
            mov     A, R1                       ;Reload A
            orl     A, #30h                     ;Should have ascii number value now(03h --> 33h)
            LCALL   printChar                   ;put character to LCD
            sjmp    next
letter3:    mov     A, R1
            orl     A, #30h                     ;ascii non-normalized
            add     A, #07h                     ;ascii normalized (3Fh --> 46h)
```

```
            LCALL   printChar
next:               mov     A, B
            anl     A, #0fh
            mov     R1, A                   ;to copy before check
            CLR     C
            subb    A, #0Ah
            jnc     letter4
            mov     A, R1
            orl     A, #30h
            LCALL   printChar
            sjmp    finish
letter4:    mov     A, R1
            orl     A, #30h
            add     A, #07h
            LCALL   printChar               ;print the normalized second character
finish:             mov     A, #20h
            LCALL   printChar               ;print space


            INC     R0                      ;INC AMOUNT PRINTED TO LINE
            INC     R5                      ;INC CURRENT ADDRESS
            INC     R7                      ;INC AMOUNT PRINTED TO LCD


            CJNE    R5, #00H, CONT13
            INC     R4                      ;INC HIGH BYTE IF LOW BYTE OV
CONT13:
            DEC     R3                      ;DEC LOW BYTE OF BLOCK SIZE
            CJNE    R3, #0FFH, CONT15
            DEC     R2                      ;DEC HIGH BYTE IF LOW BYTE UV
CONT15:

            CJNE    R2, #0, CONT11          ;If maximum block size hasnt been reached, then move
            CJNE    R3, #0, CONT11          ;forward
            LJMP    DONE                    ;IF BOTH HIGH/LOW BYTE OF BLOCK SIZE 0, JUMP
                                            ;TO DONE AND PROMPT
CONT11:
            CJNE    R0, #6, LOOP            ;IF LINE ISNT FILLED, KEEP PRINTING
            LCALL   PUT_LINE2               ;OTHERWISE, MOVE TO SECOND LINE
            MOV     R0, #0                  ;CLEAR AMOUNT PRINTED TO LINE, AND PRINT
NEXT LINE
            CJNE    R7, #12, LOOP           ;CHECK IF TOTAL AMOUNT PRINTED TO LCD IS 12

DONE:
            MOV     DPH, R4
            MOV     DPL, R5

            PUSH    DPH
            PUSH    DPL

            MOV     DPTR, #DUMPPROMPT
            LCALL   PUT_LINE3
            LCALL   PRINTSTRING

            MOV     DPTR, #DUMPPROMPT2
            LCALL   PUT_LINE4
            LCALL   PRINTSTRING

            POP     DPL
            POP     DPH

            ;LCALL  PUTDUMPADDR             ;PRINT NEXT ADDRESS
```

```
DONE3:          LCALL   PROMPTKEYPAD    ;WHEN BLOCK SIZE IS FULL, PROMPT,
                                        ;WHEN LCD IS FILLED, PROMPT
                CJNE    A, #32H, CONT16 ;PROMPT FOR EXIT, IF NOT PRESSED, CHECK '0'
                LJMP    ENDDUMP

CONT16:         CJNE    A, #30H, CONT17 ;TRY TO GO TO NEXT PAGE, IF NOT PRESSED,
CHECK '1'
                CJNE    R2, #0, NEXTPAGE ;If maximum block size has been reached, then DONT GO
                CJNE    R3, #0, NEXTPAGE ;TO NEXT PAGE
                LJMP    DONE3           ;IF BLOCK SIZE REACHED, INVALID KEY PRESS

CONT17:         CJNE    A, #31H, DONE3  ;TRY TO GO TO PREVIOUS PAGE, IF NOT PRESSED,
REPROMPT
                CJNE    R6, #0, PREVPAGE ;CHECK PAGE ZER0
                LJMP    DONE3           ;IF PAGE 0, REPROMPT

NEXTPAGE:LCALL  CLEAR_LCD               ;next page routine
                INC     R6              ;INC PAGE #
                MOV     R7, #0
                LJMP    LOOP

PREVPAGE:LCALL  CLEAR_LCD               ;previous page routine
                MOV     R0, #0          ;RESET AMOUNT PRINTED TO LINE
                DEC     R6              ;DEC PAGE #

                MOV     A, R3           ;LOW BYTE OF BLOCK SIZE
                CLR     C
                ADD     A, R7           ;REUPDATE BLOCK SIZE
                JC      INCHBYTE
                CLR     C
                ADD     A, #12          ;ADD LAST PAGE AMOUNT
                JC      INCHBYTE2
                MOV     R3, A           ;UPDATE LOW BYTE OF BLOCK
                SJMP    CONT18


INCHBYTE:       INC     R2              ;INC HIGH BYTE IF CARRY ON R7 ADDITION
                ADD     A, #12
                MOV     R3, A           ;UPDATE LOW BYTE OF BLOCK
                SJMP    CONT18

INCHBYTE2:      INC     R2              ;INC HIGH BYTE IF CARRY ON 12 ADDITION
                MOV     R3, A           ;UPDATE LOW BYTE OF BLOCK

CONT18:         MOV     A, R5           ;MOVE BACK LOW BYTE OF CURRENT ADDRESS
                CLR     C
                SUBB    A, #12
                JC      DECHBYTE        ;NO CARRY ON FIRST SUBB
                CLR     C
                SUBB    A, R7           ;SUBB CURRENT PAGE AMOUNT
                JC      DECHBYTE2       ;CHECK IS CARRY ON PAGE AMOUNT SUBB
                MOV     R5, A
                MOV     R7, #0          ;clear amount printed to page
                LJMP    LOOP

DECHBYTE:
                DEC     R4              ;CARRY ON FIRST SUBB, UPPER BYTE UPDATED
                SUBB    A, R7           ;SUBB CURRENT PAGE AMOUNT
                MOV     R5, A
                MOV     R7, #0          ;clear amount printed to page
                LJMP    LOOP            ;REPRINT AND REPROMPT WITH NEW ADDRESS
```

```
DECHBYTE2:      DEC     R4                      ;PREVIOUS ADDRESS
                MOV     R5, A
                MOV     R7, #0                  ;CLEAR AMOUNT PRINTED TO PAGE
                LJMP    LOOP
ENDDUMP:
                RET
;===============================================================================
;| PRINT ADDRESS FOR DUMP                                                      |
;===============================================================================
PUTDUMPADDR:
                LCALL   PUT_ADDR
                mov     A, #28h                 ;print '('
                LCALL   printChar

                MOV     DPH, R4                 ;PUT SAVED DPH IN DPH
                MOV     A, R4
                LCALL   PRINTADDR               ;printAddr will print HIGH BYTE

                MOV     DPL, R5                 ;PUT SAVED DPL IN DPL
                MOV     A, R5                   ;PRINTADRR WILL PRINT LOW BYTE
                LCALL   PRINTADDR

                mov     A, #68h                 ;print 'h'
                LCALL   printChar

                mov     A, #29h                 ;print ')'
                LCALL   printChar

                RET
;===============================================================================
;| Prompt for edit values                                                      |
;===============================================================================
promptEdit:

                LCALL   clear_lcd
                mov     DPTR, #eSource
                LCALL   printString

                MOV     DPTR, #DIGITMSG
                LCALL   PUT_LINE2
                LCALL   PRINTSTRING

                LCALL   PUT_LINE3_CB

bData1:         LCALL   GETBYTE                 ;Source address will be in R1
                mov     A, R1
                mov     DPH, A                  ;DPH NOW XX00H
                MOV     R3, A                   ;SAVE DPH IN R3

                LCALL   GETBYTE
                MOV     A, R1
                MOV     DPL, A                  ;DPL NOW 00XXH
                MOV     R4, A                   ;SAVE DPL IN R4

CONT40:         MOV     DPTR, #VERIFYINPUT
                LCALL   PUT_LINE4
                LCALL   PRINTSTRING
                LCALL   PROMPTKEYPAD

                CJNE    A, #41H, CONT39         ;IF THEY HIT 'A' AND ACCEPT
                LJMP    CONT38                  ;MOVE FORWARD
```

```
CONT39:         CJNE    A, #44H, CONT40         ;IF THEY HIT 'D' AND WANT TO REDO
                LJMP    PROMPTEDIT

CONT38:

    here12:     RET
;================================================================================
;| edit byte by byte starting at a location                                     |
;================================================================================
edit:
                CLR     P3.0

                LCALL   clear_lcd
                mov     A, #28h                 ;print '('
                LCALL   printChar

                MOV     DPH, R3                 ;PUT SAVED DPH IN DPH
                MOV     A, R3
                LCALL   PRINTADDR               ;printAddr will print HIGH BYTE

                MOV     DPL, R4                 ;PUT SAVED DPL IN DPL
                MOV     A, R4                   ;PRINTADRR WILL PRINT LOW BYTE
                LCALL   PRINTADDR

                mov     A, #68h                 ;print 'h'
                LCALL   printChar

                mov     A, #29h                 ;print ')'
                LCALL   printChar

                mov     A, #3Ah                 ;print ':'
                LCALL   printChar

                mov     A, #20h                 ;print space
                LCALL   printchar

                LCALL   printByte               ;print the byte

                LCALL   PUT_LINE2               ;Go to next line

                PUSH    DPH
                PUSH    DPL

                mov     DPTR, #replace          ;Point dptr to replace request string
                LCALL   PRINTSTRING

                MOV     DPTR, #DIGITMSG1
                LCALL   PUT_LINE3
                LCALL   PRINTSTRING

                LCALL   PUT_LINE4_CB

                POP     DPL
                POP     DPH

                LCALL   GETBYTE                 ;New byte should be in R1

                MOV     A, R1                   ;new byte is in A
                MOV     DPH, R3
                MOV     DPL, R4
                MOVX    @DPTR, A                ;move new byte to source address location
```

```
            LCALL   clear_lcd
            mov     A, #28h                 ;print '('
            LCALL   printChar

            MOV     DPH, R3
            MOV     A, DPH
            LCALL   PRINTADDR               ;printAddr will print HIGH BYTE

            MOV     DPL, R4
            MOV     A, DPL                  ;PRINTADRR WILL PRINT LOW BYTE
            LCALL   PRINTADDR

            mov     A, #68h                 ;print 'h'
            LCALL   printChar

            mov     A, #29h                 ;print ')'
            LCALL   printChar

            mov     A, #3Ah                 ;print ':'
            LCALL   printChar

            mov     A, #20h                 ;print space
            LCALL   printchar

            LCALL   printByte               ;print the updated byte

            mov     A, #68h
            LCALL   printchar

            LCALL   PUT_LINE2
            mov     DPTR, #user1
            LCALL   printString

            LCALL   PUT_LINE3
            MOV     DPTR, #user2
            LCALL   PRINTSTRING

    eInput: LCALL   promptKeypad            ;To get a decision from the user
            cjne    A, #31h, cont1          ;if key press is 1 exit, else continue
            mov     DPTR, #exitmsg
            LCALL   clear_lcd
            LCALL   printString
            sjmp    done2

    cont1:  cjne    A, #30h, eInput
            INC     R4
            CJNE    R4, #00H, OV1
            INC     R3
OV1:
            LJMP    Edit
done2:
            RET
;===============================================================================
;| PROMPT USED FOR FIND                                                        |
;===============================================================================
promptFind:
            LCALL   clear_lcd
            mov     DPTR, #esource
            LCALL   printString

            MOV     DPTR, #DIGITMSG
```

```
        LCALL   PUT_LINE2
        LCALL   PRINTSTRING

        LCALL   PUT_LINE3_CB
        LCALL   GETBYTE
        mov     A, R1
        mov     R2, A                       ;high byte of address now in xx00h R2

        LCALL   GETBYTE
        MOV     A, R1
        MOV     R3, A                       ;low byte of address now in 00xxh R3
                                            ;source address now in DPTR
CONT42: MOV     DPTR, #VERIFYINPUT
        LCALL   PUT_LINE4
        LCALL   PRINTSTRING
        LCALL   PROMPTKEYPAD

        CJNE    A, #41H, CONT41             ;IF THEY HIT 'A' AND ACCEPT
        LJMP    ZERO                        ;MOVE FORWARD

CONT41: CJNE    A, #44H, CONT42         ;IF THEY HIT 'D' AND WANT TO REDO
        LJMP    PROMPTFIND

ZERO:           LCALL   clear_lcd
        mov     DPTR, #fBlock
        LCALL   printString

        MOV     DPTR, #DIGITMSG
        LCALL   PUT_LINE2
        LCALL   PRINTSTRING

        LCALL   PUT_LINE3_CB

        LCALL   GETBYTE
        mov     A, R1
        mov     R4, A                       ;XX00H OF BLOCK SIZE IN R4

        LCALL   GETBYTE
        MOV     A, R1
        MOV     R5, A                       ;00XXH OF BLOCK SIZE IN R5

CONT44:         MOV     DPTR, #VERIFYINPUT
        LCALL   PUT_LINE4
        LCALL   PRINTSTRING
        LCALL   PROMPTKEYPAD

        CJNE    A, #41H, CONT43             ;IF THEY HIT 'A' AND ACCEPT
        LJMP    CONT45                      ;MOVE FORWARD

CONT43:         CJNE    A, #44H, CONT44     ;IF THEY HIT 'D' AND WANT TO REDO
        LJMP    ZERO

CONT45:         CJNE    R5, #0, CONT7
        CJNE    R4, #0, CONT7               ;CANT HAVE BLOCK SIZE OF ZERO
        LJMP    ZERO
CONT7:
        LCALL   clear_lcd
        mov     DPTR, #FindByte
        LCALL   printString

        MOV     DPTR, #DIGITMSG1
        LCALL   PUT_LINE2
```

```
            LCALL   PRINTSTRING

            LCALL   PUT_LINE3_CB
            LCALL   GETBYTE
            mov     A, R1
            mov     R6, A                   ;byte to find in R6

            LCALL   clear_lcd
            RET

;===========================================================================
;| See if a byte is in a specific location                                 |
;===========================================================================
;SOURCE     R2R3H
;BLOCK      R4R5H
;BYTE       R6H
find:
            CLR     P3.0

            MOV     DPH, R2
            MOV     DPL, R3
            movx    A, @DPTR                ;GET VALUE IN AT ADDRESS LOCATION

            CLR     C
            subb    A, R6
            jz      Found                   ;IF RESULT IS ZERO, THEN THE BYTE IS FOUND

            CJNE    R4, #0, CONT8
            CJNE    R5, #0, CONT8           ;SEE IF WE ARE OUT OF BLOCK SIZE
                                            ;IF NOT, CONTINUE, INC DPTR, DEC BLOCK SIZE
            MOV     DPTR, #nFound           ;Didn't find byte, print message
            LCALL   printSTRING
            LCALL   HALFSECONDDELAY
            LCALL   HALFSECONDDELAY

            SJMP    HERE14                  ;RETURN TO THE PROGRAM

CONT8:

            INC     R3
            CJNE    R3, #00H, CONT9
            INC     R2                      ;CHECK IF LOWER BYTE HAS BEEN OVERFLOWED
CONT9:
            DEC     R5
            CJNE    R5, #0FFH, CONT10
            DEC     R4                      ;CHECK IF LOWER BYTE HAS ROLLED OVER
CONT10:
            LJMP    FIND                    ;HAVE NEW DPTR VALUE, AND NEW BLOCK SIZE

Found:
            PUSH    DPH
            PUSH    DPL
            mov     DPTR, #FOUNDBYTE        ;Found the byte, print message
            LCALL   printSTRING

            POP     DPL
            POP     DPH

            LCALL   PUT_LINE2
            mov     A, #28h                 ;put '('
            LCALL   PRINTchar
```

```
            MOV     A, DPH                  ;PRINT DPH
            LCALL   PRINTADDR               ;print the address it was found at @DPTR

            MOV     A, DPL                  ;PRINT DPL
            LCALL   PRINTADDR

            mov     A, #68h                 ;print 'h'
            LCALL   PRINTChar

            mov     A, #29h                 ;'put ')'
            LCALL   PRINTchar

            LCALL   HALFSECONDDELAY
            LCALL   HALFSECONDDELAY
            LCALL   HALFSECONDDELAY
            LCALL   HALFSECONDDELAY

            here14: RET
;===============================================================================
;| To flash status decimal place                                               |
;===============================================================================
flash7seg:
            PUSH    0
            SETB    P3.0
            MOV     R0, #io_sevenseg

            MOV     A, #01111111b
            LCALL   ioToggle                ;what is in dptr goes to address, A to data
            LCALL   delay_50ms

            MOV     A, #11111111b
            LCALL   ioToggle

            POP     0
            CLR     P3.0
            RET

;===============================================================================
;| To update the time...                                                       |
;===============================================================================
getRTC:
            push    0
            push    acc

            LCALL   PUT_RTC                 ;print it to the correct spot

            MOV     R0, #45H                ;top hour digit
            LCALL   readReg
            ORL     A, #30H                 ;convert to ascii
            LCALL   printChar

            MOV     R0, #44H                ;bottom hour digit
            LCALL   readReg
            ORL     A, #30H
            LCALL   printChar

            MOV     A, #3Ah                 ;print ":"
            LCALL   printChar

            MOV     R0, #43H                ;get top minute digit
            LCALL   readReg
            ORL     A, #30H
```

```
                LCALL   printChar

                MOV     R0, #42H                ;get bottom minute digit
                LCALL   readReg
                ORL     A, #30H                 ;convert to ascii
                LCALL   printChar

                MOV     A, #3AH                 ;print ":"
                LCALL   printChar

                MOV     R0, #41H
                LCALL   readReg
                ORL     A, #30H
                LCALL   printChar

                MOV     R0, #40H
                LCALL   readReg
                ORL     A, #30H
                LCALL   printChar

                pop     acc
                pop     0
                RET

;=========================================================================
;| To update the temperature...                                         |
;=========================================================================
getTemp:
                PUSH    0
                MOV     R0, #10H
                SETB    P3.0                    ;Get the info from the ADC
                MOVX    A, @R0
                SUBB    A, #9
                CLR     P3.0
                POP     0

                RET
;=========================================================================
;| To print the byte at an address                                      |
;=========================================================================
printAddr:
                push    0E0h
                push    1
                MOV     B, A
                anl     A, #0f0h
                rr      A
                rr      A
                rr      A
                rr      A
                mov     R7, A                   ;To save the raw value
                CLR     C
                SUBB    A, #0Ah                 ;check if letter
                jnc     letter5
                mov     A, R7                   ;Reload A
                orl     A, #30h                 ;Should have ascii number value now(03h --> 33h)
                LCALL   printChar               ;put character to LCD
                sjmp    next2
    letter5:mov         A, R7
                orl     A, #30h                 ;ascii non-normalized
                add     A, #07h                 ;ascii normalized (3Fh --> 46h)
                LCALL   printChar
    next2:      mov     A, B
```

```
            anl     A, #0fh
            mov     R7, A                   ;to copy before check
            CLR     C
            subb    A, #0Ah
            jnc     letter6
            mov     A, R7
            orl     A, #30h
            LCALL   printChar
            sjmp    finish2
    letter6:mov     A, R7
            orl     A, #30h
            add     A, #07h
            LCALL   printChar               ;print the normalized second character
    finish2:
            pop     1
            pop     0E0h
            RET

;=========================================================================
;| To print temperature to the LCD                                       |
;=========================================================================
printTemp:

            LCALL   PUT_TEMP
            MOV     A, R6                   ;10s place of the temp
            LCALL   printChar
            MOV     A, R7                   ;1s place of the temp
            LCALL   printChar
            MOV     A, #0DFH                ;print degree symbol
            LCALL   printChar
            MOV     A, #43H
            LCALL   printChar

            RET
;=========================================================================
;| Converts byte in A from hex to ascii                                  |
;=========================================================================
hexToAscii:
            MOV     B, #10
            DIV     AB
            MOV     R7, B
            MOV     B, #10
            DIV     AB
            MOV     R6, B
            MOV     R5, A
            ORL     7, #30H                 ;first digit in R7
            ORL     6, #30H                 ;Second digit in R6
            ORL     5, #30H                 ;Third digit in R5
            RET
;=========================================================================
;| Waits for somebody to login                                           |
;=========================================================================
login:
            LCALL   CLEAR_LCD
            MOV     A, #0CH                 ;TURN CURSOR OFF
            LCALL   COMNWRT

REPRINT:
            MOV     R0, #92H                ;TOP RIGHT
            MOV     R1, #95H                ;BOTTOM LEFT
            MOV     R2, #20H                ;SPACE
            MOV     R3, #0C0H               ;LEFT BAR
```

```
                MOV     R4, #0D3H                       ;RIGHT BAR

CONTPRINT:      MOV     DPTR, #LOGINART1
                LCALL   PUT_LINE1
                LCALL   PRINTSTRING

                MOV     DPTR, #osName
                LCALL   PUT_LINE2
                LCALL   printString

                MOV     DPTR, #LOGINART2
                LCALL   PUT_LINE3
                LCALL   PRINTSTRING

                MOV     DPTR, #loginMSG
                LCALL   PUT_LINE4
                LCALL   printString

CONT22:         LCALL   PROMPTKEYPAD
                CJNE    A, #31h, CONT22                 ;IF A ONE IS NOT PRESSED, KEEP PRINTING ART
                LJMP    ENDLOGIN                        ;OTHERWISE IF IT IS EQUAL TO 1, LOGIN

                ;BORDER ART AND ANIMATION
;               MOV     A, R0                           ;PUT AT APPROPRIATE ADDRESS OF TOP BAR
;               LCALL   PUT_FLEX
;               MOV     A, R2                           ;LOAD SPACE
;               LCALL   PRINTCHAR
;               DEC     R0                              ;DECREMENT THE TOP BAR ADDRESS
;
;               MOV     A, R1
;               LCALL   PUT_FLEX                        ;PUT AT BOTTOM BAR
;               MOV     A, R2                           ;LOAD SPACE
;               LCALL   PRINTCHAR
;               CJNE    R1, #0A6H, CONT21
;               LJMP    RIGHTLEFT
;CONT21:
;               LCALL   POLLKEYPAD
;               CJNE    A, #31h, CONT19                 ;IF A ONE IS NOT PRESSED, KEEP PRINTING ART
;               SJMP    ENDLOGIN                        ;OTHERWISE IF IT IS EQUAL TO 1, LOGIN
;CONT19:
;               INC     R1                              ;INCREMENT THE BOTTOM BAR ADDRESS
;               ;LCALL  DELAY_100MS
;               LJMP    CONTPRINT
;
;RIGHTLEFT:
;               MOV     A, R3                           ;PRINT SPACE AT LEFT BAR
;               LCALL   PUT_FLEX
;               MOV     A, R2
;               LCALL   PRINTCHAR
;
;               MOV     A, R4                           ;PRINT SPACE AT RIGHT BAR
;               LCALL   PUT_FLEX
;               MOV     A, R2
;               LCALL   PRINTCHAR
;
;               LCALL   POLLKEYPAD
;               CJNE    A, #31h, CONT22                 ;IF A ONE IS NOT PRESSED, KEEP PRINTING ART
;               LJMP    ENDLOGIN                        ;OTHERWISE IF IT IS EQUAL TO 1, LOGIN
;CONT22:
;
;               ;LCALL  DELAY_100MS
;               LJMP    REPRINT
```

```
ENDLOGIN:
        RET
;================================================================================
;| Displays the passcode prompt messages                                        |
;================================================================================
displayPasscode:
        LCALL   CLEAR_LCD
        MOV     DPTR, #myPasscode
        LCALL   PUT_LINE1
        LCALL   printString

        LCALL   PUT_LINE2_CB
        ;MOV    DPTR, #myPasscode2
        ;LCALL  PUT_LINE2
        ;LCALL  printString

        RET
;================================================================================
;| Gets the key presses and decides if they are valid                           |
;================================================================================
getPasscode:
        CLR     A
        MOV     R6, #3                  ;TRIES LEFT
        MOV     R5, #0                  ;PROFILE #
retry:
        MOV     DPTR, #attempts         ;print attempts string
        LCALL   PUT_LINE4
        LCALL   printString

        MOV     A, R6                   ;print attempts left number
        ORL     A, #30H
        LCALL   printChar
        CLR     A

        LCALL   displayPasscode         ;display passcode message

        CLR     A
        LCALL   promptKeypad            ;get first digit in ascii from keypad
        ;MOV    A, #38h                 ;TEST

        PUSH    ACC
        MOV     A, #2AH
        LCALL   printChar               ;print * to the LCD
        POP     ACC

        ANL     A, #0FH
        LCALL   rotateleft
        MOV     R1, A                   ;move to R0 to save

        CLR     A
        LCALL   promptKeypad
        ;MOV    A, #37H                 ;TEST
        PUSH    ACC
        MOV     A, #2AH
        LCALL   printChar               ;print * to the LCD
        POP     ACC
        ANL     A, #0FH
        ORL     A, R1                   ;first byte of pw in R1
        MOV     R1, A                   ;new cumulative saved
        MOV     R2, A                   ;saved in R2 also

        CLR     A
```

```
            LCALL   promptKeypad
            ;MOV    A, #30H                 ;TEST
            PUSH    ACC
            MOV     A, #2AH
            LCALL   printChar               ;print * to the LCD
            POP     ACC
            ANL     A, #0FH
            LCALL   rotateleft
            MOV     R0, A                   ;new cumulative saved

            CLR     A
            LCALL   promptKeypad
            ;MOV    A, #31H                 ;TEST
            PUSH    ACC
            MOV     A, #2AH
            LCALL   printChar               ;print * to the LCD
            POP     ACC
            ANL     A, #0FH
            ORL     A, R0                   ;second byte of pw stored in r0
            MOV     R3, A                   ;saved in R3 also
            MOV     R0, A

            LCALL   delay_100ms             ;so you can see full password

;R1 and R2 contain xx
;R0 and R3 contain yy
;to make 'xxyy' the password

            MOV     DPTR, #pwList           ;LUT of valid passwords
checkPW:    CLR     A
            MOV     A, R2                   ;load saved cumulative value
            MOV     R1, A
            CLR     A
            MOVC    A, @A+DPTR              ;grab actuall password value from LUT
            JZ      doOver                  ;if end of LUT is hit, reprompt

            CLR     C
            SUBB    A, R1                   ;otherwise check xx
            JZ      secondByte              ;if they are exact, valid xx
            INC     DPTR                    ;otherwise, pw cannot be valid at all
            INC     DPTR                    ;inc dptr and jump to next xxyy

            CLR     A
            MOVC    A, @A+DPTR              ;check first byte of next xxyy
            JZ      doOver                  ;if zero, end of LUT reached
            INC     R5                      ;otherwise, increment potential profile
            sjmp    checkPW                 ;check the next pw in LUT
secondByte:
            INC     DPTR
            MOV     A, R3                   ;load yy
            MOV     R0, A
            CLR     A
            MOVC    A, @A+DPTR              ;load yy of saved LUT value

            CLR     C
            SUBB    A, R0                   ;check if equal
            JZ      success                         if exact, valid yy
            INC     DPTR                            ;otherwise jump to next xxyy
            INC     R5                              ;update potential profile
            SJMP    checkPW                         ;repeat check
doOver:
            MOV     R5, #0                          ;clear potential profile if re-entering
```

```
                LCALL   CLEAR_LCD
                MOV     DPTR, #incorrectCode                    ;print incorrect code prompt
                LCALL   PUT_LINE1
                LCALL   printString
                LCALL   halfseconddelay

                LCALL   CLEAR_LCD
                CLR     A                                       ;conditional to check if we should
                DEC     R6                                      ;retry or lock the system
                MOV     A, R6
                JZ      lockout                                 ;jump if zero to lock system

                MOV     DPTR, #tryagain                         ;prompt again if more tries
                LCALL   PUT_LINE1
                LCALL   printString
                LCALL   halfseconddelay
                LJMP    retry


                ;DJNZ    R6, retry                      ;Three tries to get pw right before
                ;SJMP    lockout                                ;entering lockout

success:        LCALL   CLEAR_LCD                               ;clear the lcd
                MOV     DPTR, #pwSuccess               ;and print success message
                LCALL   printString

                ;check profiles to display
                ;michael = 0
                ;collin = 1
                ;riley = 2

                LCALL   checkProfile                            ;uses R5 to determine what profile
                                                                ;has put their passcode in

                RET
;================================================================================
;| After 3 unsuccessful logins, lock the board                                  |
;================================================================================
lockout:
                LCALL   CLEAR_LCD
                MOV     DPTR, #lockedmsg        ;display lockout message for all-time
                LCALL   PUT_LINE1                       ;on line 1 of LCD
                LCALL   printString

locked:         SJMP    locked                          ;infinite loop

                RET
;================================================================================
;| [UNUSED]Scramble the input value in A for security                           |
;================================================================================
scrambleKey:
                ADD     A, #23H                         ;Michael Jordan

                RL      A                               ;Rotate left three times for '91-'93
                RL      A
                RL      A

                RL      A                               ;Rotate left three more times for '96-'98
                RL      A
                RL      A

                RET
```

```
;=========================================================================================
;| iterate through list of profiles to compare R5 to                                     |
;=========================================================================================
checkProfile:
            LCALL   PUT_LINE2
            CJNE    R5, #0, checkCollin         ;check for michael
            MOV     DPTR, #michael
            SJMP    printName

checkCollin: CJNE   R5, #1, checkRiley          ;check for collin
            MOV     DPTR, #collin
            SJMP    printName

checkRiley:  CJNE   R5, #2, checkSharif         ;check for riley
            MOV     DPTR, #riley                    ;if not, exit (should never happen)
            SJMP    printName

checkSharif: CJNE   R5, #3, checkJeff           ;check for prof. sharif
            MOV     DPTR, #sharif
            SJMP    printName

checkJeff:  CJNE    R5, #4, exit                    ;check for jeff
            MOV     DPTR, #jeff

printName:  LCALL   printString
            LCALL   halfseconddelay
exit:
            RET
;=========================================================================================
;| Procedure for 7-segment interaction                                                   |
;=========================================================================================
sevenseg:
            push    0
            MOV     R0, #IO_SEVENSEG

            ;will implement 7-segment interaction
            ;at a later date

            pop     0

            RET
;=========================================================================================
;| Rotates left 4 times                                                                  |
;=========================================================================================
rotateleft:
            RL      A
            RL      A
            RL      A
            RL      A
            RET
;=========================================================================================
;| Procedure to wait for an ascii byte press by the user; "1" = 31h                      |
;=========================================================================================
promptKeypad:
            MOV     keypad, #0FFh
    K1:     MOV     keypad, #0FH
            MOV     A, keypad
            ANL     A, #0Fh
            CJNE    A, #0Fh, K1                 ;check if key is still pressed on pad
    K2:     LCALL   delay_1ms
            MOV     A, keypad
```

```
            ANL     A, #0Fh
            CJNE    A, #0Fh, OVER               ;if not, then ground each row until 0 found
            SJMP    K2
    OVER:   LCALL   delay_1ms
            MOV     A, keypad
            ANL     A, #0Fh
            CJNE    A, #0Fh, OVER1
            SJMP    K2
    OVER1:  MOV     keypad, #0EFH               ;row 0 (1110)
            MOV     A, keypad
            ANL     A, #0FH
            CJNE    A, #0FH, ROW_0
            MOV     keypad, #0DFH               ;row 1 (1101)
            MOV     A, keypad
            ANL     A, #0FH
            CJNE    A, #0FH, ROW_1
            MOV     keypad, #0BFH               ;row 2 (1011)
            MOV     A, keypad
            ANL     A, #0FH
            CJNE    A, #0FH, ROW_2              ;row 3 (0111)
            MOV     keypad, #07FH
            MOV     A, keypad
            ANL     A, #0FH
            CJNE    A, #0FH, ROW_3
            LJMP    K2
    ROW_0:  MOV     DPTR, #KCODE0
            sjmp    kFIND
    ROW_1:  MOV     DPTR, #KCODE1
            sjmp    kFIND
    ROW_2:  MOV     DPTR, #KCODE2
            sjmp    kFIND
    ROW_3:  MOV     DPTR, #KCODE3
            sjmp    kFIND
    kFIND:  RRC     A
            JNC     MATCH
            INC     DPTR
            sjmp    kFIND
    MATCH:  CLR     A
            MOVC    A, @A+DPTR
            MOV     keypad, A
            RET
;=============================================================================
;| Procedure to poll for an ascii byte press by the user; "1" = 31h                                    |
;=============================================================================
pollKeypad:
            MOV     keypad, #0FFh
    K3:     MOV     keypad, #0Fh
            LCALL   delay_1ms
            MOV     A, keypad
            ANL     A, #0Fh
            CJNE    A, #0Fh, OVER3
            SJMP    exit1                       ;otherwise, exit and go back to updating
    OVER3:  MOV     keypad, #0EFH               ;row 0 (1110)
            MOV     A, keypad
            ANL     A, #0FH
            CJNE    A, #0FH, xROW_0
            MOV     keypad, #0DFH               ;row 1 (1101)
            MOV     A, keypad
            ANL     A, #0FH
            CJNE    A, #0FH, xROW_1
            MOV     keypad, #0BFH               ;row 2 (1011)
            MOV     A, keypad
```

```
            ANL     A, #0FH
            CJNE    A, #0FH, xROW_2             ;row 3 (0111)
            MOV     keypad, #07FH
            MOV     A, keypad
            ANL     A, #0FH
            CJNE    A, #0FH, xROW_3
            LJMP    exit1
    xROW_0: MOV     DPTR, #KCODE0
            sjmp    kFIND2
    xROW_1: MOV     DPTR, #KCODE1
            sjmp    kFIND2
    xROW_2: MOV     DPTR, #KCODE2
            sjmp    kFIND2
    xROW_3: MOV     DPTR, #KCODE3
            sjmp    kFIND2
    kFIND2: RRC     A
            JNC     MATCH2
            INC     DPTR
            sjmp    kFIND2
    MATCH2: CLR     A
            MOVC    A, @A+DPTR
            MOV     keypad, A
    exit1:
            RET
;=============================================================================
;| 7 Segment wakeup procedure (3 DP blinks)                                  |
;=============================================================================
wakeUp:
            PUSH    0
            SETB    P3.0
            MOV     R0, #io_sevenseg

            MOV     A, #01111111b
            LCALL   ioToggle                    ;what is in dptr goes to address, A to data
            LCALL   delay_100ms
            LCALL   delay_100ms

            MOV     A, #11111111b
            LCALL   ioToggle
            LCALL   delay_100ms
            LCALL   delay_100ms

            MOV     A, #01111111b
            LCALL   ioToggle
            LCALL   delay_100ms
            LCALL   delay_100ms

            MOV     A, #11111111b
            LCALL   ioToggle
            LCALL   delay_100ms
            LCALL   delay_100ms

            MOV     A, #01111111b
            LCALL   ioToggle
            LCALL   delay_100ms
            LCALL   delay_100ms

            MOV     A, #11111111b
            LCALL   ioToggle
            LCALL   delay_100ms
            LCALL   delay_100ms
            POP     0
```

```
                CLR     P3.0

                RET
;=========================================================================
;| Procedure to display my name on the LCD                               |
;=========================================================================
displayName:
                LCALL   PUT_LINE1
                MOV     DPTR, #myName
                LCALL   printString

                LCALL   PUT_LINE2
                MOV     DPTR, #myClass
                LCALL   printString

                LCALL   halfseconddelay
                LCALL   CLEAR_LCD
                LCALL   halfseconddelay

                RET
;=========================================================================
;| Procedure to display the menu on the LCD screen                       |
;=========================================================================
displayMenu:
                LCALL   CLEAR_LCD
                LCALL   PUT_LINE2               ;print choices 1
                MOV     DPTR, #menu1
                LCALL   printString             ;will print the string pointed @ by dptr

                LCALL   PUT_LINE3               ;print choices 2
                MOV     DPTR, #menu2
                LCALL   printString

                LCALL   PUT_LINE4               ;print choices 2
                MOV     DPTR, #logout
                LCALL   printString

                RET
;=========================================================================
;| Procedure to initialize the LCD                                       |
;=========================================================================
LCD_INIT:
                CLR     RW
                CLR     RS

                LCALL   DELAY_50MS

                MOV     A, #38H
                LCALL   COMNWRT
                LCALL   DELAY_1MS

                MOV     A, #38H
                LCALL   COMNWRT
                LCALL   DELAY_1MS

                MOV     A, #0CH
                LCALL   COMNWRT
                LCALL   DELAY_1MS

                MOV     A, #01H
                LCALL   COMNWRT
                LCALL   DELAY_5MS
```

```
                MOV     A, #06H
                LCALL   COMNWRT

                RET
;===============================================================================
;| RTC initialization                                                          |
;===============================================================================
RTC_INIT:

                MOV     R0, #4Fh                ;F REG INIT
                MOV     A, #00h
                LCALL   ioToggle                ;Send whats in R0 to Address bus
                                                ;Send whats in A to data bus
                MOV     R0, #4Dh
                MOV     A, #00h                 ;CD register init
                LCALL   ioToggle

                ;LCALL  checkBusy

                MOV     R0, #4FH
                MOV     A, #03H                 ;RESET THE COUNTER
                LCALL   ioToggle

                ;SET CURRENT TIME FOR REGS
                MOV     R0, #40H                ;FIRST SECONDS
                MOV     A, #00H
                ;LCALL  SETHOLD
                LCALL   IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #41H                ;SECOND SECONDS
                MOV     A, #00H
                ;LCALL  SETHOLD
                LCALL   IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #42H                ;ETC...
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #43H
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #44H
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #45H
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #46H
```

```
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #47H
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #48H
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #49H
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #4AH
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD

                MOV     R0, #4BH
                MOV     A, #00H
                ;LCALL  SETHOLD
                CALL    IOTOGGLE
                ;LCALL  CLEARHOLD


                ;START COUNTER AND RELEASE HOLD
                MOV     R0, #4Fh                        ;F REG INIT
                MOV     A, #00h
                LCALL   ioToggle

                MOV     R0, #4Dh
                MOV     A, #00h                         ;CD register init
                LCALL   ioToggle

                RET
;=============================================================================
;| Check if the RTC is busy                                                  |
;=============================================================================
checkBusy:
                PUSH    0
                PUSH    ACC
                MOV     R0, #4Dh                ;GET CD REG IN RTC
waitBusy:
                MOV     A, #05H
                SETB    P3.0
                MOVX    @R0, A                  ;SET HOLD
                CLR     P3.0

                SETB    P3.0
                MOVX    A, @R0                  ;READ IN THE CD REG
                CLR     P3.0
```

```
            ;JNB    ACC.1, busyReady ;CHECK IF BUSY BIT HIGH

            MOV     A, #04h
            SETB    P3.0
            MOVX    @R0, A                  ;clear hold to let busy bit update
            CLR     P3.0
            LCALL   DELAY_1MS
            SJMP    waitBusy

busyReady:
            POP     ACC
            POP     0

            RET
;=========================================================================
;| Read a register in the RTC                                           |
;=========================================================================
readReg:
            PUSH    0
            PUSH    ACC

            MOV     R0, #4DH                ;SET THE HOLD BIT
            MOV     A, #05H
            SETB    P3.0
            MOVX    @R0, A
            CLR     P3.0

            POP     ACC
            POP     0

            ;LCALL  checkBusy               ;Wait until not busy
            SETB    P3.0                    read valUE
            MOVX    A, @R0
            CLR     P3.0

            ANL     A, #0FH                 ;MASK OFF LOWER HALF

            PUSH    ACC

            MOV     R0, #4DH                ;CLR THE HOLD BIT
            MOV     A, #04H
            SETB    P3.0
            MOVX    @R0, A
            CLR     P3.0
            POP     ACC



            RET
;=========================================================================
;| Write to a register in the RTC                                       |
;=========================================================================
writeReg:
            ;LCALL  checkBusy               ;Wait until not busy
            SETB    P3.0
            MOVX    @R0, A                  ;Read in value
            CLR     P3.0

            PUSH    ACC
            MOV     R0, #4DH
            MOV     A, #04H                 ;Clear hold
```

```
            SETB    P3.0
            MOVX    @R0, A
            CLR     P3.0
            POP     ACC

            RET
;===============================================================================
;| To write a command to the LCD THAT IS IN A                                  |
;===============================================================================
COMNWRT:
            PUSH    ACC
            PUSH    0
            MOV     R0, #io_lcd
            CLR     RS                          ;RS
            CLR     RW                          ;RW
            SETB    P3.0
            MOVX    @R0, A
            CLR     P3.0
            LCALL   DELAY_1MS
            POP     ACC
            POP     0
            RET

;===============================================================================
;| To clear the LCD                                                            |
;===============================================================================
CLEAR_LCD:
            PUSH    0
            push    ACC
            MOV     A,#01H
            LCALL   COMNWRT                     ;CLEAR THE LCD
            LCALL   DELAY_5MS
            MOV     A, #0CH                     ;REMOVE THE CURSOR
            LCALL   COMNWRT
            pop     ACC
            pop     0
            RET
;===============================================================================
;| To print the temperature in the top right corner                            |
;===============================================================================
PUT_TEMP:
            push    0
            push    ACC
            CLR     RS
            MOV     R0, io_lcd
            MOV     A, #090H
            LCALL   COMNWRT
            LCALL   DELAY_5MS
            SETB    RS
            pop     ACC
            pop     0
            RET
;===============================================================================
;| Put the temperature in the top left of the LCD                              |
;===============================================================================
PUT_RTC:
            PUSH    0
            PUSH    ACC
            CLR     RS
            MOV     R0, io_lcd
            MOV     A, #080H
            LCALL   COMNWRT
```

```
                LCALL   DELAY_5MS
                SETB    RS
                POP     ACC
                POP     0
                RET
;================================================================================
;| Print the string on the first line of the LCD                                |
;================================================================================
PUT_LINE1:
                PUSH    0
                PUSH    ACC
                CLR     RS
                MOV     R0, io_lcd
                MOV     A, #080H
                LCALL   COMNWRT
                LCALL   DELAY_5MS
                SETB    RS
                POP     ACC
                POP     0
                RET
;================================================================================
;| Put string on the second line of the LCD                                     |
;================================================================================
PUT_LINE2:
                PUSH    0
                PUSH    ACC
                CLR     RS
                MOV     R0, io_lcd
                MOV     A, #0C0H
                LCALL   COMNWRT
                LCALL   DELAY_5MS
                SETB    RS
                POP     ACC
                POP     0
                RET
;================================================================================
;| Put string on the second line of the LCD w/ cursor blinking                  |
;================================================================================
PUT_LINE2_CB:
                PUSH    0
                PUSH    ACC
                CLR     RS
                MOV     R0, io_lcd
                MOV     A, #0C0H                ;DDRAM ADDRESS
                LCALL   COMNWRT
                LCALL   DELAY_5MS
                MOV     A, #0FH                 ;SET CURSOR ON AND BLINKING
                LCALL   COMNWRT
                LCALL   DELAY_1MS
                SETB    RS
                POP     ACC
                POP     0
                RET
;================================================================================
;| Put string on line 3 of the lCD                                              |
;================================================================================
PUT_LINE3:
                PUSH    0
                PUSH    ACC
                CLR     RS
                MOV     R0, io_lcd
                MOV     A, #94H
```

```
                LCALL   COMNWRT
                LCALL   DELAY_5MS
                SETB    RS
                POP     ACC
                POP     0

                RET

;===============================================================================
;| Put string on line 3 of the LCD w/ cursor blinking
    |
;===============================================================================
PUT_LINE3_CB:
                PUSH    0
                PUSH    ACC
                CLR     RS
                MOV     R0, io_lcd
                MOV     A, #94H                 ;DDRAM ADDRESS
                LCALL   COMNWRT
                LCALL   DELAY_5MS
                MOV     A, #0FH                 ;CURSOR BLINKING
                LCALL   COMNWRT
                LCALL   DELAY_1MS
                SETB    RS
                POP     ACC
                POP     0

                RET
;===============================================================================
;| Put string on line 4 of the LCD                                             |
;===============================================================================
PUT_LINE4:
                PUSH    0
                PUSH    ACC
                MOV     R0, io_lcd
                CLR     RS
                MOV     A, #0D4H
                LCALL   COMNWRT
                LCALL   DELAY_5MS
                MOV     A, #0CH
                LCALL   COMNWRT
                LCALL   DELAY_1MS
                SETB    RS
                POP     ACC
                POP     0
                RET

;===============================================================================
;| Put string on line 4 of the LCD w/ cursor blinking                          |
;===============================================================================
PUT_LINE4_CB:
                PUSH    0
                PUSH    ACC
                MOV     R0, io_lcd
                CLR     RS
                MOV     A, #0D4H
                LCALL   COMNWRT
                LCALL   DELAY_5MS
                MOV     A, #0FH
                LCALL   COMNWRT
                LCALL   DELAY_1MS
                SETB    RS
```

```
            POP     ACC
            POP     0
            RET

;===============================================================================
;| PRINTS ADDRESS OF DUMP ON LINE 3                                            |
;===============================================================================
PUT_ADDR:
            PUSH    0
            PUSH    ACC
            MOV     R0, io_lcd
            CLR     RS
            MOV     A, #0A1H
            LCALL   COMNWRT
            LCALL   DELAY_5MS
            SETB    RS
            POP     ACC
            POP     0
            RET

;===============================================================================
;| STARTINGS PRINTING AT THE DDRAM VALUE OF A BEFORE ENTERING SUBROUTINE       |
;===============================================================================
PUT_FLEX:
            PUSH    0
            PUSH    ACC
            MOV     R0, io_lcd
            CLR     RS
            LCALL   COMNWRT
            LCALL   DELAY_5MS
            SETB    RS
            POP     ACC
            POP     0
            RET

;GETBYTE grabs two key presses and combines them into a single byte value
;the byte value will be returned in R1, or is available on key_out
;===============================================================================
;| grabs two key presses and combines them into a single byte value, returns   |
;| in A                                                                        |
;===============================================================================
GETBYTE:
            push    0
            PUSH    7
            LCALL   promptKeypad        ;Get first digit of block
            LCALL   PRINTCHAR
            ;mov    A, keypad           ;move first digit to A
            MOV     R7, A               ;SAVE VALUE
            SUBB    A, #40h
            jnc     letter
            mov     A, R7               ;else, regrab the output from key
            anl     A, #0fh             ;mask to get data
            sjmp    rotate
    letter: mov     A, R7               ;if letter regrab, data
            anl     A, #0fh             ;mask off lower half
            ;add    A, #09h             ;add 09h to normalize
            ADD     A, #09H             ;it is normalize
    rotate: RL      A
            RL      A
            RL      A
            RL      A
            mov     R0, A
```

```
invalid:LCALL   promptKeypad            ;Get first digit of block
        LCALL   PRINTCHAR
        ;mov    A, KEYPAD               ;move first digit to A
        MOV     R7, A
        SUBB    A, #40h
        jnc     letter2
        mov     A, R7           ;else, regrab the output from key
        anl     A, #0fh                 ;mask to get data
        sjmp    here13
letter2:mov     A, R7           ;if letter regrab, data
        anl     A, #0fh                 ;mask off lower half
        ;add    A, #09h                 ;add 09h to normalize
        ADD     A, #09H
        anl     A, #0fh
here13: orl     A, R0                   ;Now both bits are in A
        mov     R1, A                   ;To preserve block size in R1
        POP     7
        pop     0
        RET

;=========================================================================
;| Print a string to the LCD                                            |
;=========================================================================
printString:
        CLR     A
        movc    A, @A+DPTR
        JZ      pExit
        LCALL   printChar
        INC     DPTR
        SJMP    printString
pExit:          RET
;=========================================================================
;| Print a byte in A to the LCD                                         |
;=========================================================================
printByte:
        push    0E0h
        push    1
        MOVX    A, @DPTR
        MOV     B, A
        ANL     A, #0f0h
        rr      A
        rr      A
        rr      A
        rr      A
        mov     R7, A                   ;To save the raw value
        CLR     C
        SUBB    A, #0Ah                 ;check if letter
        jnc     letter13
        mov     A, R7                   ;Reload A
        orl     A, #30h                 ;Should have ascii number value now(03h --> 33h)
        LCALL   printChar               ;put character to LCD
        sjmp    next1
letter13:       mov     A, R7
        orl     A, #30h                 ;ascii non-normalized
        add     A, #07h                 ;ascii normalized (3Fh --> 46h)
        LCALL   printChar
next1:          mov     A, B
        anl     A, #0fh
        mov     R7, A                   ;to copy before check
        CLR     C
        subb    A, #0Ah
        jnc     letter14
```

```
                mov     A, R7
                orl     A, #30h
                LCALL   printChar
                sjmp    finish1
letter14:       mov     A, R7
                orl     A, #30h
                add     A, #07h
                LCALL   printChar               ;print the normalized second character
finish1:        mov     A, #20h
                pop     1
                pop     0E0h

                RET
;=========================================================================
;| Print a character to the LCD      IN ACC                              |
;=========================================================================
printChar:
                push    0
                SETB    RS
                CLR     RW
                MOV     R0, #io_lcd
                SETB    P3.0
                MOVX    @R0, A
                LCALL   delay_1MS
                CLR     P3.0
                pop     0
                RET
;=========================================================================
;| A delay for .5s                                                       |
;=========================================================================
halfSecondDelay:
                LCALL   delay_100ms
                LCALL   delay_100ms
                LCALL   delay_100ms
                LCALL   delay_100ms
                LCALL   delay_100ms
                RET
;=========================================================================
;| Procedure that sends A to data bus and whats in DPTR to the address bus |
;=========================================================================
ioToggle:
                SETB    P3.0
                MOVX    @R0, A
                CLR     P3.0
                RET
;=========================================================================
;| clear hold bit on rtc                                                 |
;=========================================================================
setHold:
                PUSH    0
                PUSH    ACC

                MOV     R0, #4DH                        ;SET THE HOLD BIT
                MOV     A, #05H
                SETB    P3.0
                MOVX    @R0, A
                CLR     P3.0

                POP     ACC
                POP     0
                RET
```

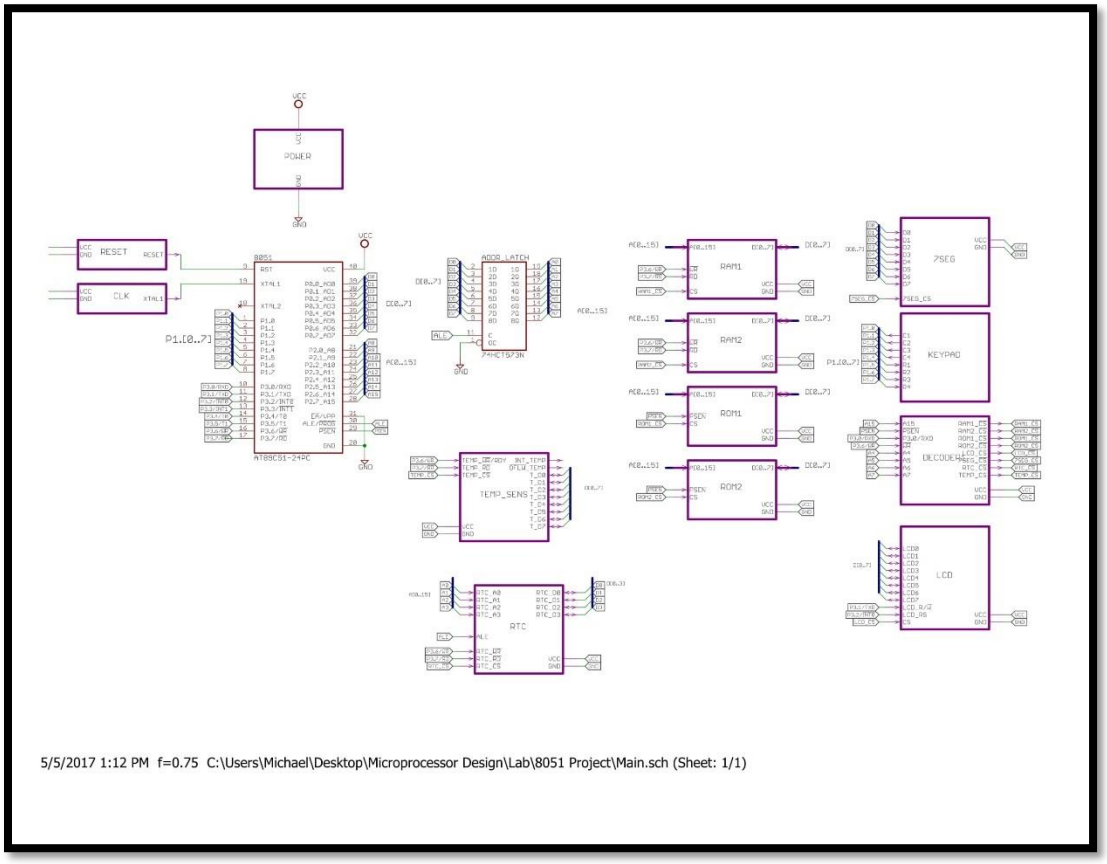
```
;=========================================================================
;| clear hold bit on rtc                                                 |
;=========================================================================
clearHold:
            PUSH    0
            PUSH    ACC

            MOV     R0, #4DH                    ;CLR THE HOLD BIT
            MOV     A, #04H
            SETB    P3.0
            MOVX    @R0, A
            CLR     P3.0

            POP     ACC
            POP     0
            RET
;=========================================================================
;| Iterative 100ms delay using delay_1ms                                 |
;=========================================================================
DELAY_100ms:

            PUSH    3
            MOV     R3,#97
    HERE7:  LCALL   DELAY_1ms
            DJNZ    R3,HERE7
            POP     3
            RET
;=========================================================================
;| Iterative 50ms delay using delay_1ms                                  |
;=========================================================================
DELAY_50ms:

            PUSH    3
            MOV     R3,#50
    HERE8:  LCALL   DELAY_1ms
            DJNZ    R3,HERE2
            POP     3
            RET
;=========================================================================
;| Iterative 10ms delay using delay_1ms                                  |
;=========================================================================
DELAY_10ms:

            PUSH    3
            MOV     R3,#10
    HERE2:  LCALL   DELAY_1ms
            DJNZ    R3,HERE2
            POP     3
            RET
;=========================================================================
;| Iterative 5ms delay using delay_1ms                                   |
;=========================================================================
DELAY_5ms:
            PUSH    3
            MOV     A, #5
            MOV     R3, A
    HERE3:  LCALL   DELAY_1MS
            DJNZ    R3, HERE3
            POP     3
            RET
;=========================================================================
```

```
;| 1ms delay                                                                                    |
;=============================================================================================
DELAY_1ms:
                PUSH    3
                PUSH    4

                MOV     R3,#33

    HERE6:      MOV     R4,#14
    HERE5:      DJNZ    R4,HERE5
                DJNZ    R3,HERE6
                POP     4
                POP     3
                RET
;=============================================================================================
;| Look up tables & Strings                                                                   |
;=============================================================================================
;login strings
loginMSG:       db      ' Press [1] to Login \0'
goodbye:        db      '    Logged Out    \0'
LOGINART1:      DB      'O------------------O\0'
osName:         db      '|  Goberling OS   |\0'
LOGINART2:      DB      'O------------------O\0'
DIGITMSG:       DB      '4 Digits (xxxxh)\0'
DIGITMSG1:      DB      '2 Digits (xxh)\0'

;program strings
bBlock:         db      'Enter Block Size\0'
bSource:        db      'Enter Source Addr.\0'
bDest:          db      'Enter Dest. Addr.\0'
bdone:          db      'Move Complete.\0'
eSource:        db      'Enter Source Addr.\0'
fBlock:         db      'Enter Block Size\0'
replace:        db      'Enter Desired value\0'
exitmsg:        db      'Program Exited\0'
user1:          db      '[0]Next Addr \0'
user2:          db      '[1]Exit \0"
FindByte:       db      'Enter value to Find\0'
FoundByte:      db      'Found value @ \0'
nFound:         db      'Byte Not Found\0'
memend:         db      'End of Memory (FFh)\0'
exitmsg2:       db      '[2] Exit\0'
DUMPPROMPT:     DB      '[0] Next \0'
DUMPPROMPT2:    DB      '[1] Prev.   [2] Exit\0'

;password strings
myPasscode:     db      'Enter 4-Digit PIN: \0'
VERIFYINPUT:    DB      '[A] Submit  [D] Redo\0'
incorrectCode:  db      'Incorrect Passcode\0'
tryAgain:       db      'Please Try Again\0'
pwSuccess:      db      'Welcome Back\0'
lockedMsg:      db      'System Locked.\0'
attempts:       db      'Tries Left: \0'

;name strings
michael:        db      'Michael!\0'
collin:         db      'Collin!\0'
riley:          db      'Riley!\0'
sharif:         db      'Prof. Sharif!\0'
jeff:           db      'Jeff!\0'

;menu strings
```

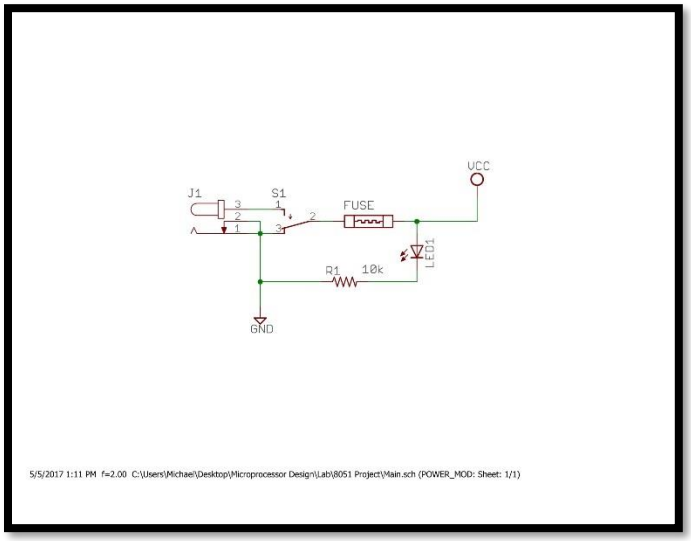
```
myName:             db      'Michael Goberling\0'
myClass:    db      'CEEN 4330 \0'
menu1:              db      '[B] Move    [D] Dump\0'
menu2:              db      '[E] Edit    [F] Find\0'
logout:             db      '[1] Logout  [7] 7Seg\0'
runtimeMenu:        db      '[Runtime] \0'
tempMenu:   db      '[Temp] \0'

;test strings
test1:              db      'Move Selected.\0'
test2:              db      'Dump Selected.\0'
test3:              db      'Edit Selected.\0'
test4:              db      'Find Selected.\0'
sevensegmsg:db      '7Seg Selected.\0'

;Profiles:
;   Michael  0
;   Collin   1
;   Riley    2
;   Sharif   3
;   Jeff     4

;profiles    ;0          ;1          ;2          ;3      ;4
pwList:     db      97h, 01h, 34H, 25H, 11H, 11H, 43H, 30H, 60H, 73H, 0
;compare valid passwords 2 bytes at a time

;matrix keypad LUT
KCODE0:     db      '1', '2', '3', 'A'
KCODE1:     db      '4', '5', '6', 'B'
KCODE2:     db      '7', '8', '9', 'C'
KCODE3:     db      'F', '0', 'E', 'D'
            END
```

9.2 Schematic

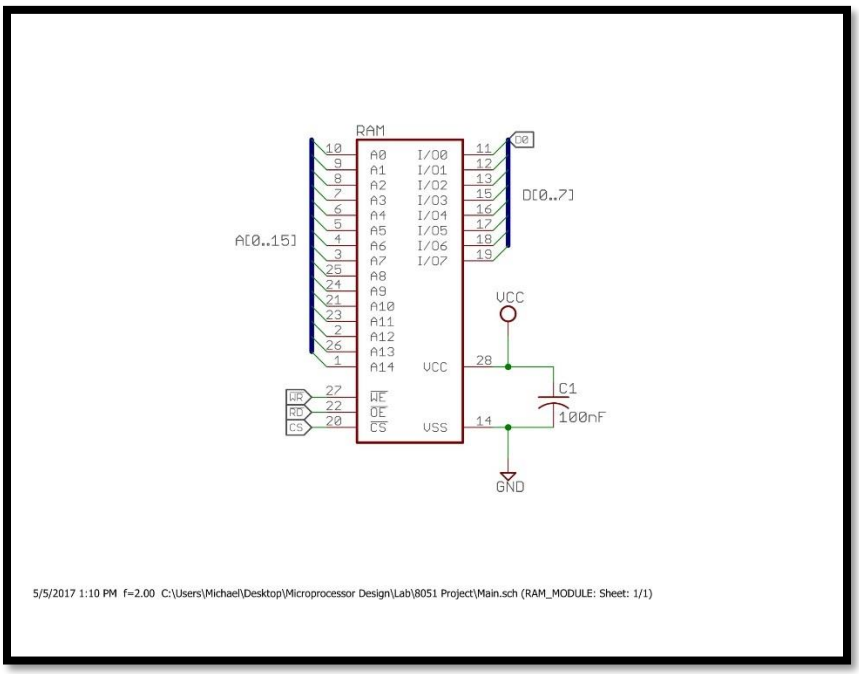
Main



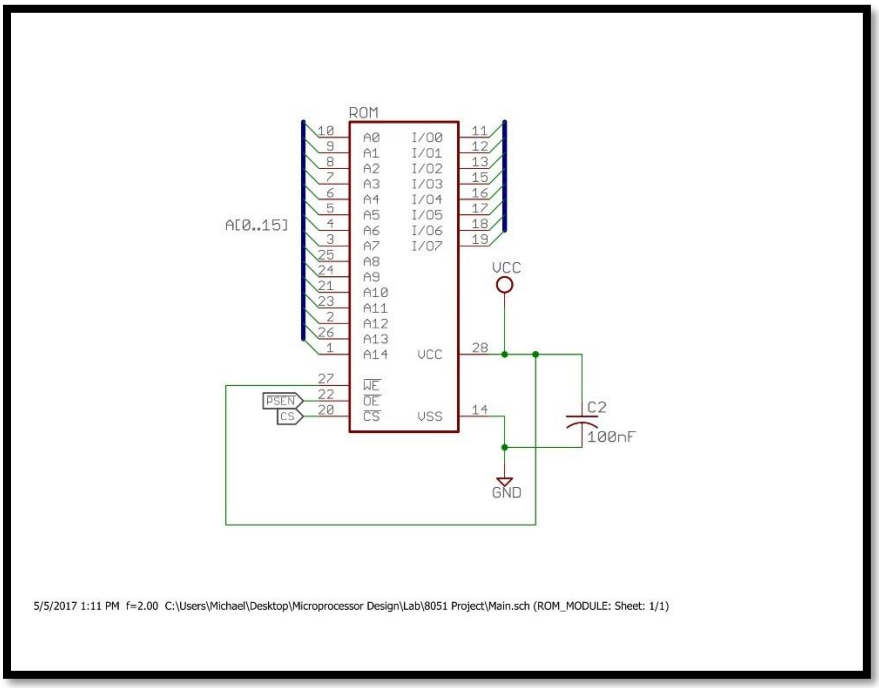
Power



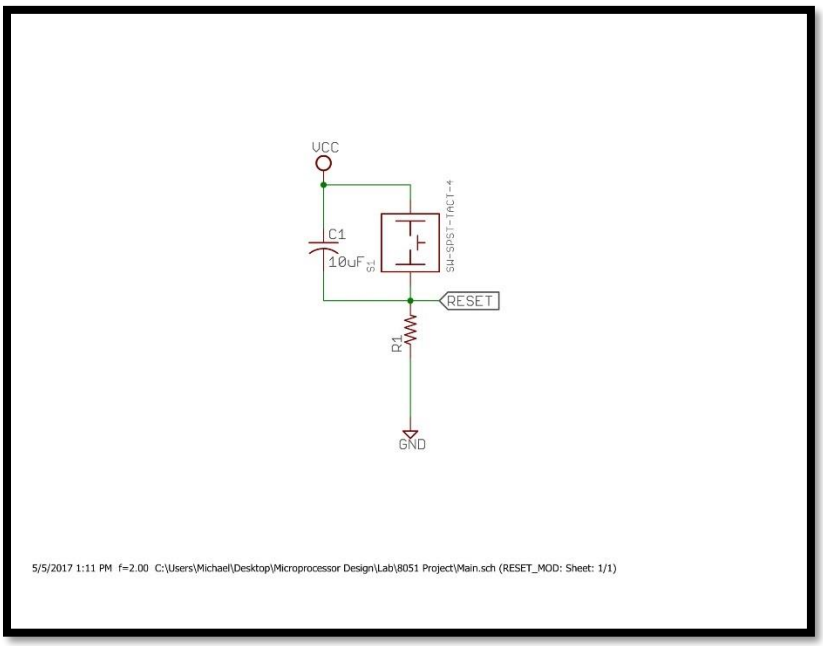
RAM


RAM
A[0..15]
D[0..7]
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
I/O0
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
VCC
VSS
WE
OE
CS
WR
RD
C1
100nF
GND
5/5/2017 1:10 PM f=2.00 C:\Users\Michael\Desktop\Microprocessor Design\Lab\8051 Project\Main.sch (RAM_MODULE: Sheet: 1/1)

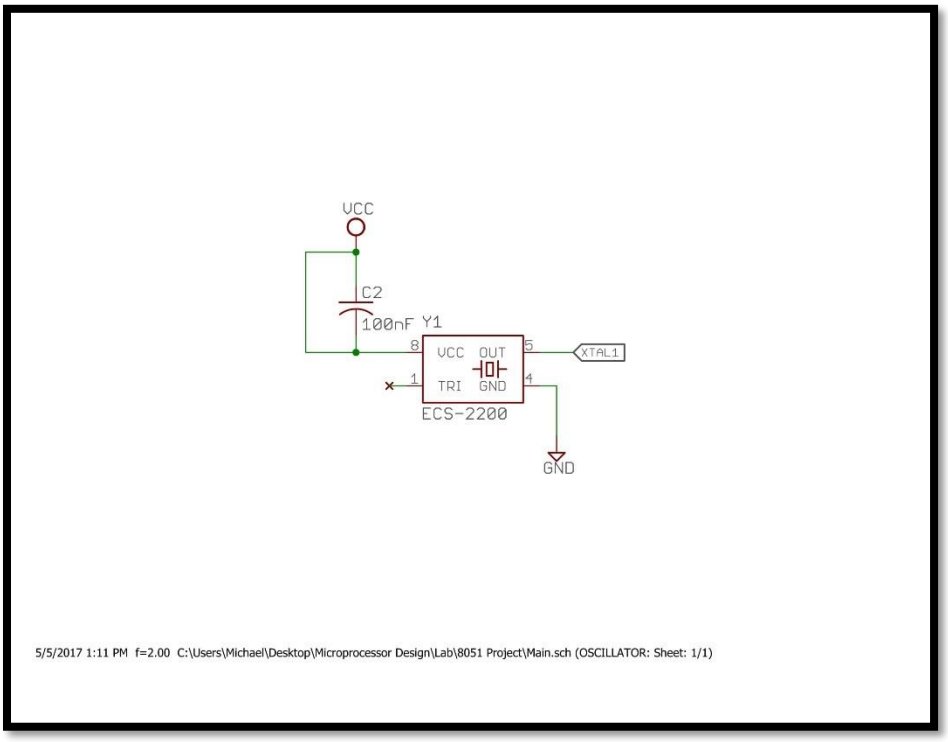
ROM


ROM
A[0..15]
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
I/O0
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
VCC
VSS
WE
OE
CS
PSEN
C2
100nF
GND
5/5/2017 1:11 PM f=2.00 C:\Users\Michael\Desktop\Microprocessor Design\Lab\8051 Project\Main.sch (ROM_MODULE: Sheet: 1/1)

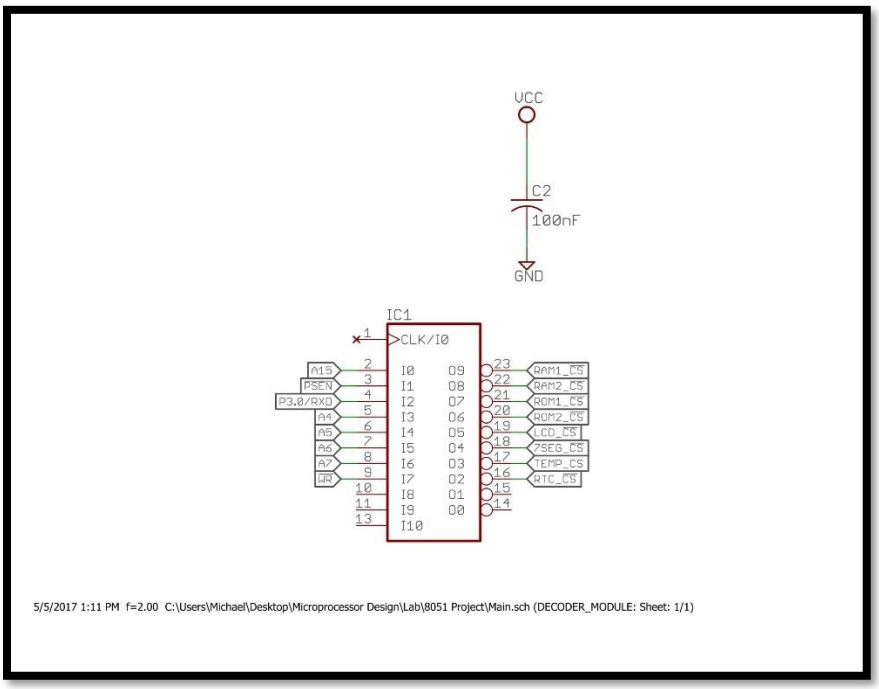
Reset


VCC
C1
10uF
S1
SW-SPST-TACT-4
RESET
R1
GND
5/5/2017 1:11 PM f=2.00 C:\Users\Michael\Desktop\Microprocessor Design\Lab\8051 Project\Main.sch (RESET_MOD: Sheet: 1/1)

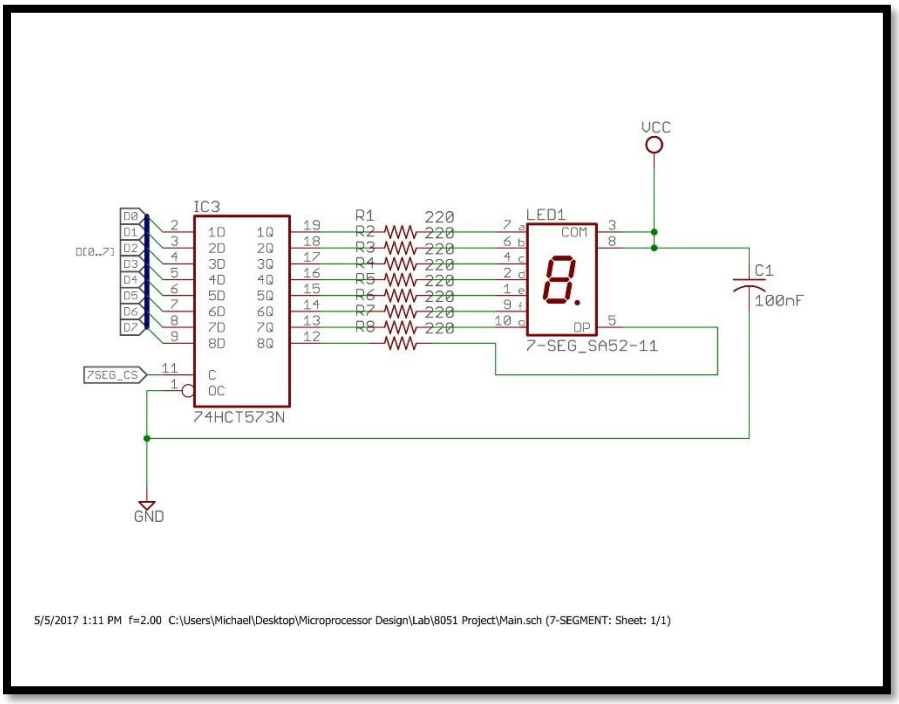
Clock


VCC
C2
100nF
Y1
8
VCC
OUT
5
XTAL1
1
TRI
GND
4
ECS-2200
GND
5/5/2017 1:11 PM f=2.00 C:\Users\Michael\Desktop\Microprocessor Design\Lab\8051 Project\Main.sch (OSCILLATOR: Sheet: 1/1)

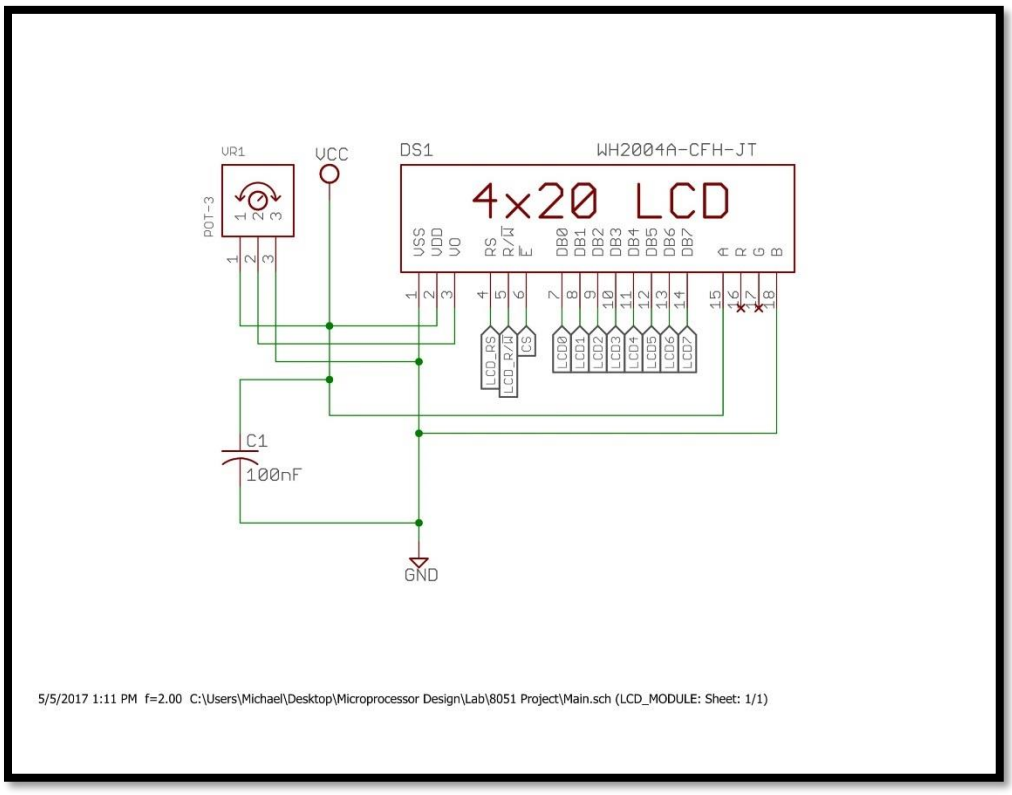
Decoder



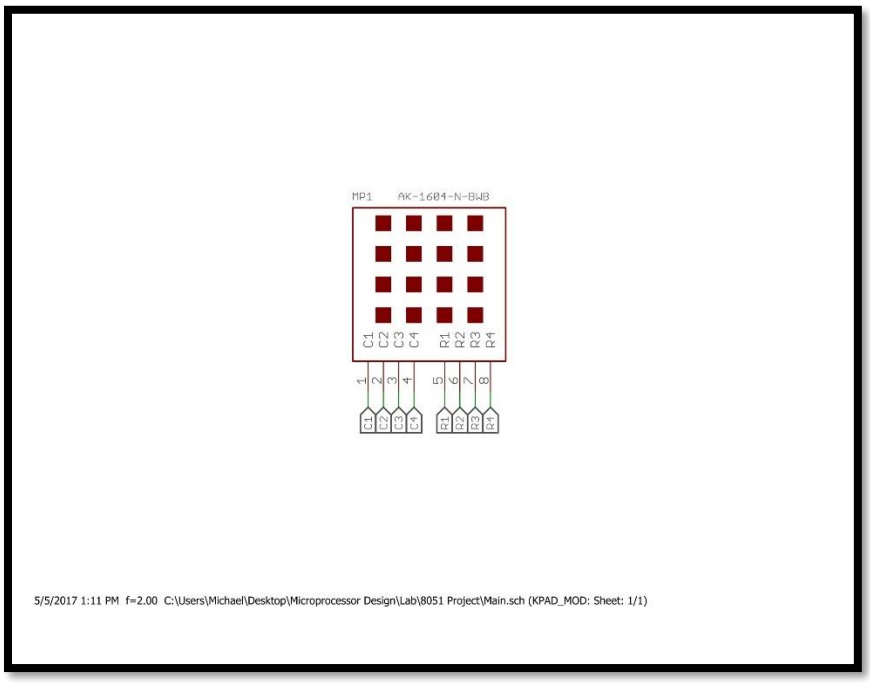
Seven Segment



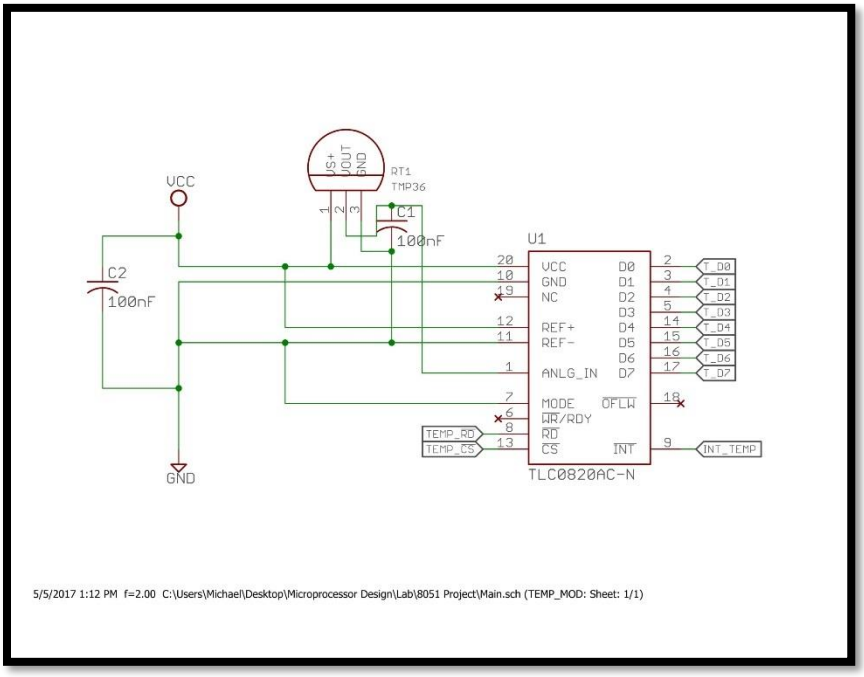
LCD


VR1
VCC
DS1
WH2004A-CFH-JT
4x20 LCD
POT-3
VSS
VDD
VO
RS
R/W
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
A
R
G
B
LCD_RS
LCD_R/W
CS
LCD0
LCD1
LCD2
LCD3
LCD4
LCD5
LCD6
LCD7
C1
100nF
GND
5/5/2017 1:11 PM f=2.00 C:\Users\Michael\Desktop\Microprocessor Design\Lab\8051 Project\Main.sch (LCD_MODULE: Sheet: 1/1)

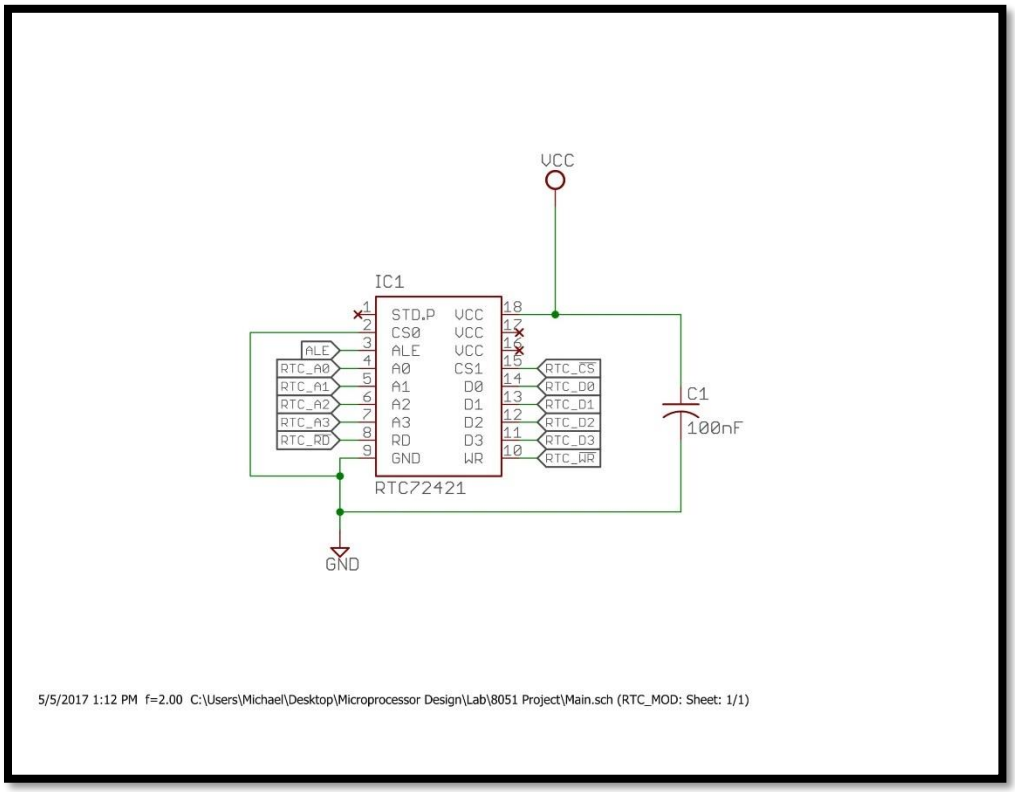
Keypad


MP1
AK-1604-N-BWB
C1
C2
C3
C4
R1
R2
R3
R4
5/5/2017 1:11 PM f=2.00 C:\Users\Michael\Desktop\Microprocessor Design\Lab\8051 Project\Main.sch (KPAD_MOD: Sheet: 1/1)

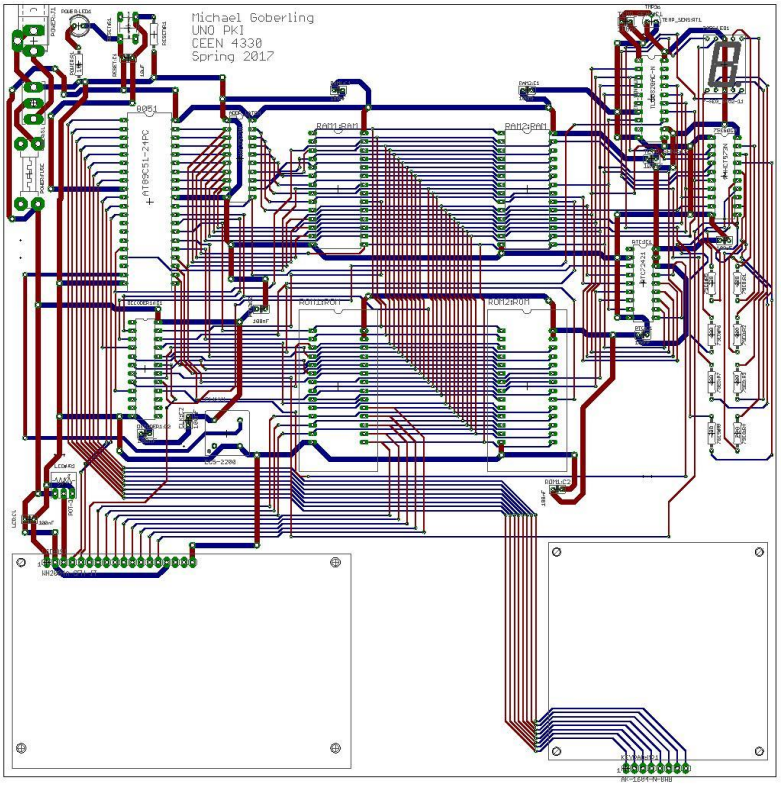
ADC and Temperature Sensor



Real Time Clock



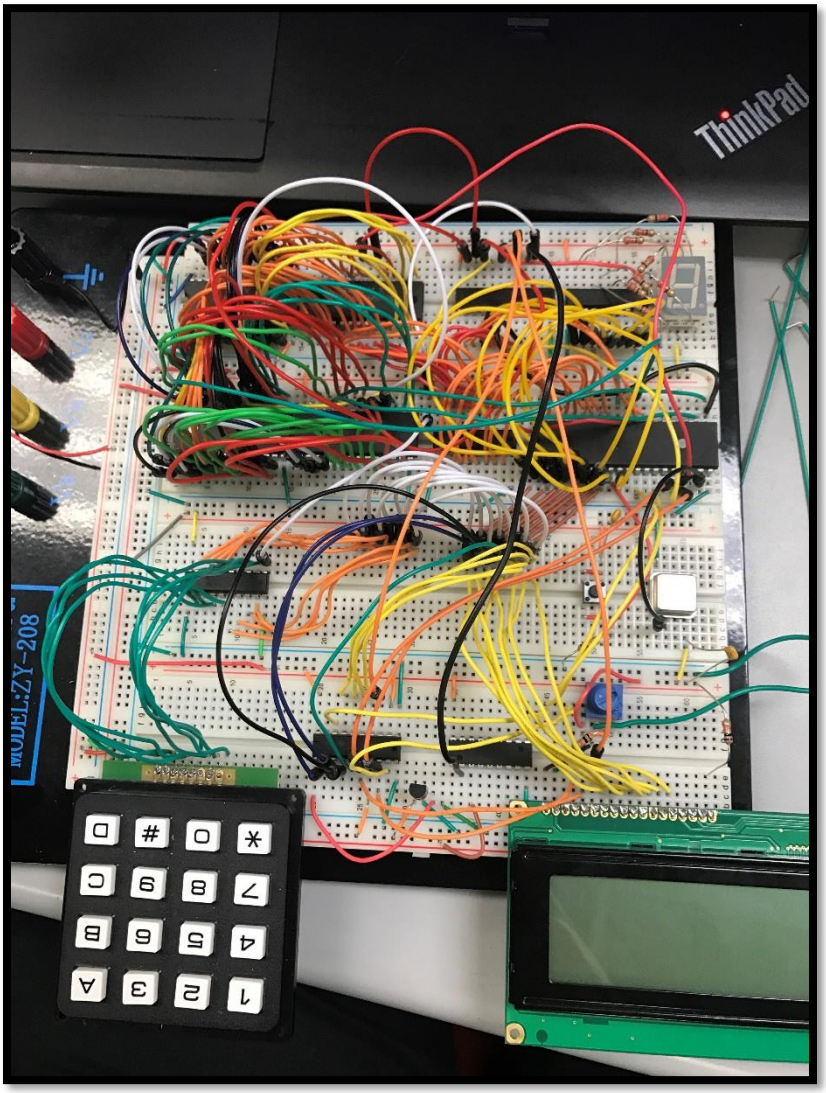
9.3 PCB Design



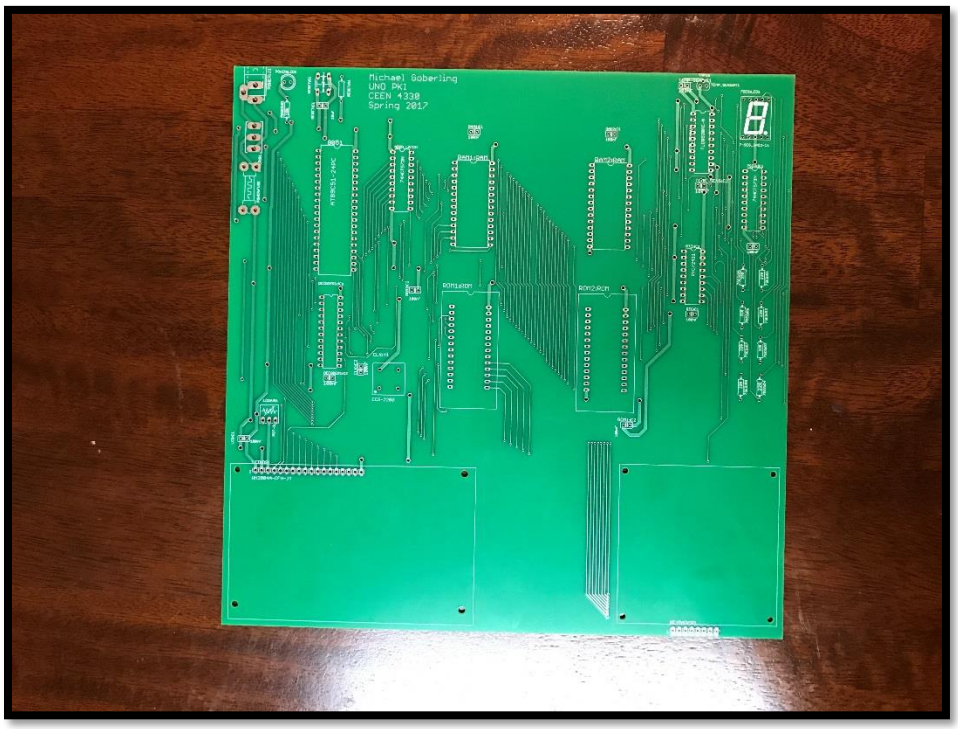
5/5/2017 1:16 PM f=0.75 C:\Users\Michael\Desktop\Microprocessor Design\Lab\8051 Project\Main.brd

9.3.1 Pictures

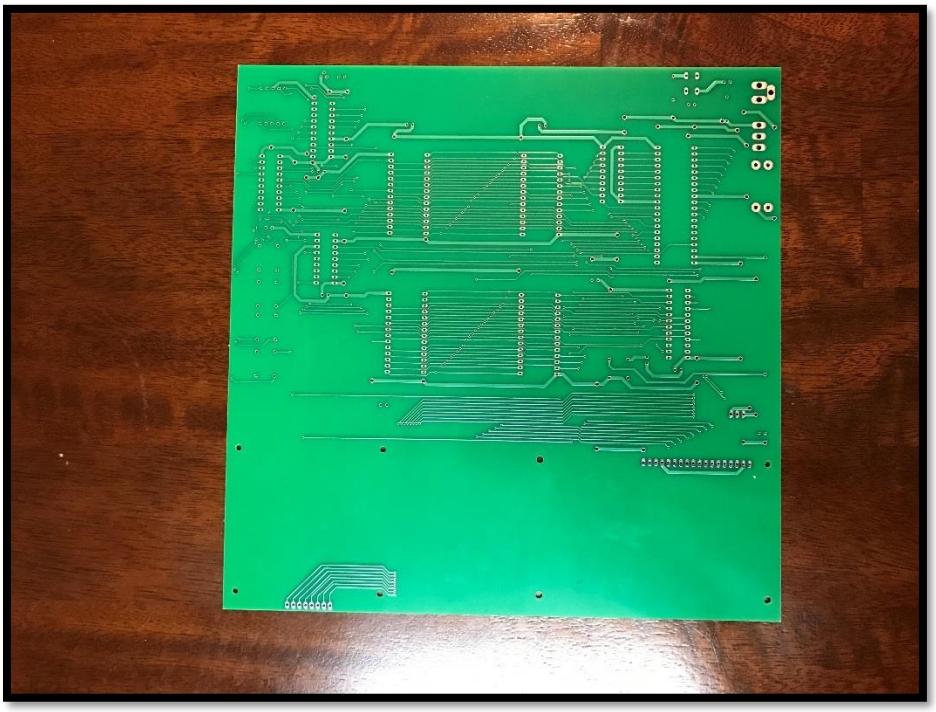
Prototyping



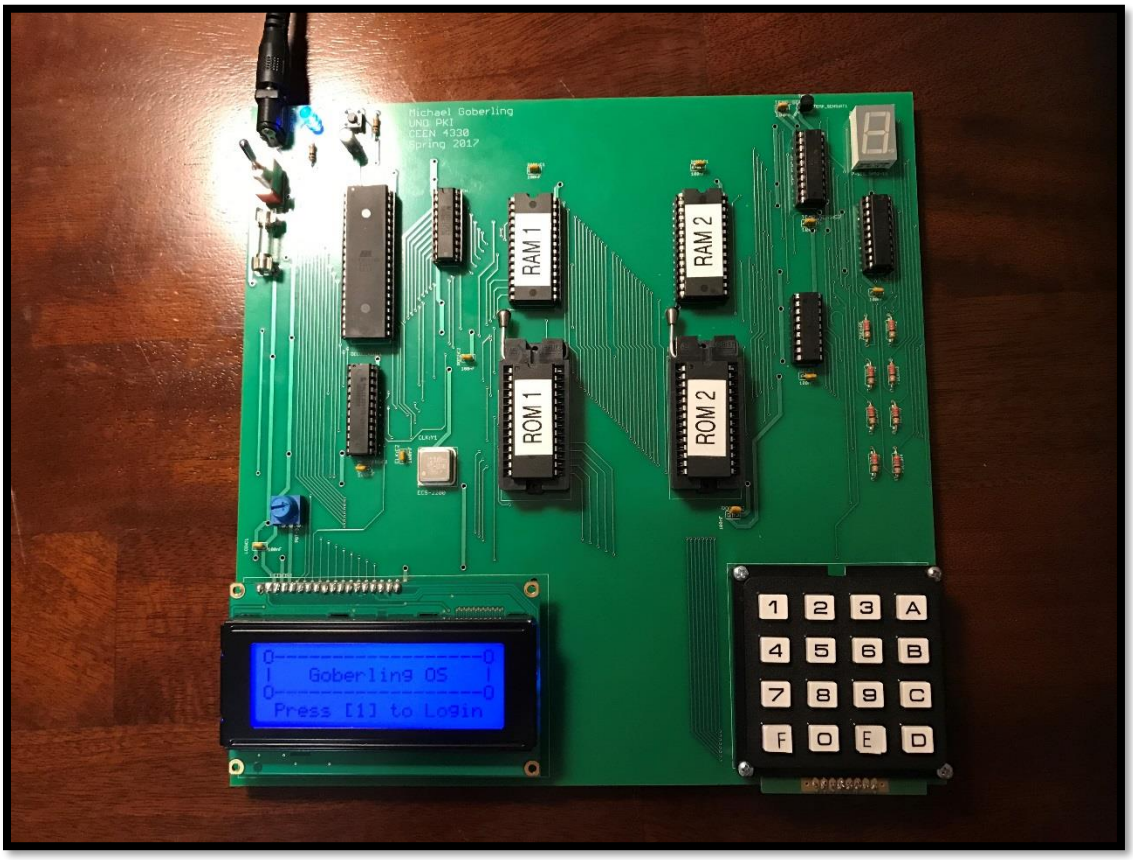
Front



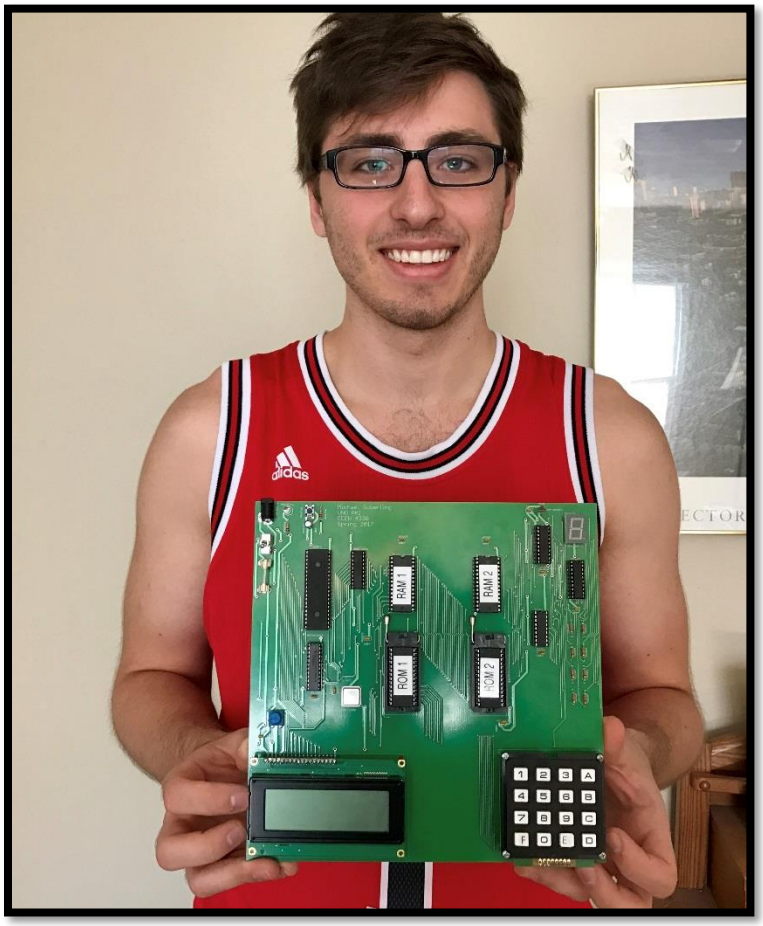
Back



Front of Populated Board



Myself and Board



9.4 Decoding

9.4.1 Decoder Logic

Memory Maps

RAM	
CHIP 1	FFFFh
	8000h
CHIP 2	7FFFh
	0000h

ROM	
CHIP 1	FFFFh
	8000h
CHIP 2	7FFFh
	0000h

I/O			
LCD			0080h
RTC			0040h
Seven Segment Latch			0020h
ADC w/ Temp Sensor			0010h
			0000h

Address Map

Y - Signals | X - Lines	A15	PSEN	P3.0	A7	A6	A5	A4
RAM 1 (0000h - 7FFFh)	0	1	0	X	X	X	X
RAM 2 (8000h - FFFFh)	1	1	0	X	X	X	X
ROM 1 (0000h - 7FFFh)	0	0	0	X	X	X	X
ROM 2 (8000h - FFFFh)	1	0	0	X	X	X	X
ADC (0010h)	X	X	1	0	0	0	1
SEVEN SEGMENT (0020h)	X	X	1	0	0	1	0
RTC (0040h)	X	X	1	0	1	0	0
LCD (0080h)	X	X	1	1	0	0	0

Pin Declarations for PAL Device

Signal	PIN	Signal	PIN
A15	2	RTC_CS	16
PSEN (input)	3	TEMP_CS	17
P3.0	4	SEVENSEG_CS	18
A4	5	LCD_CS	19
A5	6	ROM2_CS	20
A6	7	ROM1_CS	21
A7	8	RAM2_CS	22
WR	9	RAM1_CS	23

9.4.2 Decoder Code

```
-- Michael Goberling
-- 8051 Decoding
-- Spring 2017
-- CEEN 4330

library ieee;
use ieee.std_logic_1164.all;

ENTITY decoder8051 IS
		PORT(	A15, PSEN, P3_0_RXD, A7, A6, A5, A4, WR
					:		IN BIT;
				RAM1_CS, RAM2_CS, ROM1_CS, ROM2_CS, LCD_CS, SEVENSEG_CS,
TEMP_CS, RTC_CS		:		OUT BIT);

		attribute loc : string;

		attribute loc of A15			: signal is "P2";
		attribute loc of PSEN			: signal is "P3";
		attribute loc of P3_0_RXD	: signal is "P4";
		attribute loc of A4			: signal is "P5";
		attribute loc of A5			: signal is "P6";
		attribute loc of A6			: signal is "P7";
		attribute loc of A7			: signal is "P8";
		attribute loc of WR			: signal is "P9";

		attribute loc of RAM1_CS	: signal is "P23";
		attribute loc of RAM2_CS	: signal is "P22";
		attribute loc of ROM1_CS	: signal is "P21";
		attribute loc of ROM2_CS	: signal is "P20";
		attribute loc of LCD_CS		: signal is "P19";
		attribute loc of SEVENSEG_CS	: signal is "P18";
		attribute loc of TEMP_CS	: signal is "P17";
		attribute loc of RTC_CS		: signal is "P16";

END decoder8051;



ARCHITECTURE behavior OF decoder8051 IS
BEGIN

		RAM1_CS <= (A15 OR NOT PSEN OR P3_0_RXD);
		RAM2_CS <= (NOT A15 OR NOT PSEN OR P3_0_RXD);
		ROM1_CS <= (A15 OR PSEN);
		ROM2_CS <= (NOT A15 OR PSEN);

		TEMP_CS		<= (NOT P3_0_RXD OR A7 OR A6 OR A5 OR NOT A4);
		SEVENSEG_CS	<= (P3_0_RXD AND NOT A7 AND NOT A6 AND A5 AND NOT A4 AND
NOT WR);
		RTC_CS		<= (P3_0_RXD AND NOT A7 AND A6 AND NOT A5 AND NOT A4);
		LCD_CS		<= (P3_0_RXD AND A7 AND NOT A6 AND NOT A5 AND NOT A4 AND
NOT WR);


END behavior;
```