ECEN -4330
Microprocessor System Design

Final Project

Tyler Zoucha

Spring 2021

Department of Electrical
&
Computer Engineering
University of Nebraska-Lincoln
(Omaha Campus)

Due Date: 5/7/2021

Table of Contents

1) Summary

Microprocessor technology was one of the most ground-breaking technological advances in the 21st century. At first glance, they may seem like useless little chips that are typically either shielded with a metallic, reflective material or an opaque, non-conductive material; looking similar in appearance to relatively cheap or non-important hardware. However, our connected and digital world relies on their useability to not only perform multitudes of tasks faster and more accurately than any human but will do the task right every time as many times as needed or wanted. The understanding and knowing how to design a microprocessor system is of the utmost importance for aspiring computer engineers as we continue to evolve with interfacing with our world using technology and data to our advantage.

2) Objective

The overall objective of this course – and thus the final lab – was to give students the opportunity to learn and design a microprocessor system by using an 8051 microcontroller as its core. Please note that the 8051 may be replaced with the reduced 8031 microcontrollers because the core of these controllers are the same as the 8051. The core will be used to expand the available on-board memory (RAM/ROM) and IO devices by extending and interfacing these devices externally attached to the microcontroller core. Completion of a working design puts theory to the test along with teaching the design theory and system troubleshooting, debugging, and deployment in a personal, yet challenging way.

3) Introduction

The purpose of this project was to teach concepts needed to design a microprocessor system; as previously stated, we were tasked with designing a microprocessor with a core 8051 microcontroller. We were tasked to use two 32K ROM and two 32K RAM chips to expand to 64K external ROM and 64K external RAM; we then expanded on this design by adding a seven-segment display, LCD screen, real-time-clock chip, matrix keypad, as well as ESP32 communication via UART.

First, the students designed a schematic after being told the requirements. I remade my schematic about 3 times throughout this project as working through problems I faced taught me specifically what I had wrong with understanding the theory going into this lab. Next, we prototyped our schematic design on a breadboard. Breadboard prototyping is tedious, and with processor architecture handling more and more bits/signals/connections as they evolve it has almost become a thing of the past. The 8051 is an 8-bit microcontroller and thus is not completely impossible to learn via breadboard prototyping. When we had a working design on the breadboard, we ported our design to a professionally routed printed circuit board. After the boards were sent/received, the components were soldered on and each end-to-end signal was

verified. The students that had misrouted signals from porting their breadboard design to a PCB had to physically altercate their PCB by cutting the wrong trace and soldering a wire to bridge the intended connection. This opened up a new world of hardware troubleshooting and diagnosing though, and from my experience with the lab, I learned how useful it is to print off a schematic and highlight/check off each and every pin/connection with verified continuity once the PCB is soldered together because my firsts run through of highlighting the signals, I discovered I misrouted two signals compared to my breadboard design and once these connections were repaired my design's hardware was appropriately working as intended. The final part of this lab was then to program a small operating system consisting of the working firmware written to drive and interface each device within the design specifications.

4) Hardware Discussion

Hardware Design

Hardware design was the bulk of the work with constructing and engineering this project; overall, I am extremely grateful for the heartaches and unforgiving issues I had along the way because they drove me to have a deeper understanding than I would have otherwise.

The 8051 was the main brain of the design; in charge of the whole show and driven by an external clock. The external access pin (~EA/VPP) was tied to ground to ensure that the microcontroller was fetching and reading from external ROM.

PORT0 (AD0-AD7) was multiplexed for both data bus and the lower address lines; the ALE pin from the 8051 was used to latch the lower address to a buffer (in this case a 74HCT573 was used as the address/data latch)

PORT1 was connected externally to the keypad for input.

PORT2 (A8-A15) held the higher address lines and was mainly connected to the memory devices allowing the microcontroller to index through each memory chip space. The two highest order bits (A15 AND A14) were also fed into the GAL chip select decoder.

PORT3 was used for either pre-defined or arbitrarily defined signals; for example, ~RD and ~WR are denoted as P3.7 and P3.6 for READ_ and WRITE_ signals, while P3.5 was used to differentiate between the I/O or memory space. P3.4 was used for the LCD C/D- signal.

Decoding is an essential for proper chip select between the ROM, RAM, and IO devices without data collision on the Data bus. PAL/GAL decoding was recommended and used to complete this task. The implemented logic and JDEC of the GAL is available in the appendix. Essentially, the ROM chips were differentiated by A15 when the PSEN- was active; A15, A14, and IO_mem were used to navigate and interface within IO space or RAM space (depending on IO_mem)

The ROM/RAM chips are almost all wired similarly to the data bus, but differentiated by their respective CS- signal along with RD- and/or WR- signal.

The matrix keypad was attached to PORT1 as an input to the 8051. This is was the only IO device not mapped onto the data bus and extended externally –the seven-segment display, real-time clock, and LCD were extended to the data bus and depending on the CS- decoding is enabled and written to/read from when needed.

The UART module is powered by a 3.3V regulator and its Rx/Tx pins respectively are tied to the TxD/RxD pins of the ESP module (also 3.3V level) and the TxD/RxD of the 8051 respectively.

Printed Circuit Board Design

To construct the printed circuit board, I ended up finding great success with Altium after given student license period. This process first started with the schematic design, mapping all the components and their signals together. Then it was followed by footprint design and board construction. Altium made creating my own footprints easy and once I saw the results of designing my own footprint and having full control over my design appearance, I routed my board in a compartmentalized way. I have the power section in one corner, I have the memory and IO interfacing taking up the majority of the board, and I intended the bottom left corner to be home to devices susceptible to a lot of noise/distortion, for example I'm isolating the ESP module to the quiet corner of my PCB.

5) Software Discussion

The software of this project was intended to show every component of our microprocessor working correctly. The LCD and seven segment displays were used to give the user immediate visual feedback. The Matrix keypad was used to input data to the microcontroller.

Upon boot, the microcontroller performs a quick power-on self-test with the LCD, RAM, and seven-segment display, but only the RAM is currently configured to throw an error to the user if one happes.

Dump displays data to the screen; the data is shown based on the block size and block type. Move is a function used to move a specified block of data from one address to another. Edit allows the user to directly edit the value at an address before being prompted to continue to the next one. Find and Count are both very similar. The difference between these two is that find finds and prints each instance of a value in memory whereas count counts and keeps track of how many instances there are in memory. The final CheckMem function is a repeat of the memory self-test that is performed on boot; first writing a byte to each address space, inverting the nibbles of each byte, and then reading it back to ensure data integrity.

6) Problem Discussion

The downside of one of my most troublesome issues was not only two, but three breadboards that had inconsistent connections while I was prototyping my design. It took me until purchasing a whole new kit of breadboards to finally believe I had learned most of the

fundamental concepts as my design began working consistently then. For what ran for an extra run of 3-4 weeks, I was troubleshooting and trying to resolve issues that I was 100% confident I should not have been experiencing and it was not until brand-new breadboards I witnessed the behavior I expected. This is what tied into making my demonstration not ready or performing fully/properly; however, I should be able to wrap up finishing the code over the next couple of days to have a fully working project.

One of the only major other major issues I had was previously mentioned: while transitioning my design from my breadboard to the PCB I misrouted at least 2 signals. I had forgotten to tie the LCD RD- signal to VCC causing data collisions and my design freezing when I first soldered on my components. Additionally, while checking each signal's continuity I had learned that during the transition from breadboard to PCB I also switched the two C/D- and IO_mem lines used for decoding and LCD data/command transfer.

The last issue that I'm currently still facing was just fighting against the clock and how far behind I managed to get; my software isn't currently performing as intended but I also haven't had the time to work through it to troubleshoot as I've only written the skeletal structure down.

7) Conclusion

This project has been the most enduring challenge I have had recently, but I do think that remote learning and not interacting with classmates held me back a bit because in previous classes we would support each other and help each other out. This time around I did the entire lab by myself and would be stuck asking myself questions I did not know the answers to for hours or days until I did. Nothing worthwhile comes easy – if it did then it wouldn't be worthwhile.

Now after this project, I am very comfortable with how to design an 8-bit microprocessor system interfacing external memory and IO devices. Being limited by a certain number of ports or internal chip limits is no longer an issue for me now because I believe I'd be able to interface most devices I'd like to.

The overall time invested into this project is uncountable because I easily have invested at least 600 hours into this project and an additional $250 for the hardware.

8) References

8051 datasheet:
https://www.keil.com/dd/docs/datashts/atmel/doc0580.pdf

ROM datasheet:
http://ww1.microchip.com/downloads/en/DeviceDoc/doc0006.pdf

RAM datasheet:
https://www.alldatasheet.com/datasheet-pdf/pdf/65363/HYNIX/HY62256A.html

RTC datasheet:
https://support.epson.biz/td/api/doc_check.php?dl=app_RTC-72423&lang=en

XTAL oscillator datasheet:
https://ecsxtal.com/store/pdf/ecs_2200.pdf

LCD datasheet:
https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf

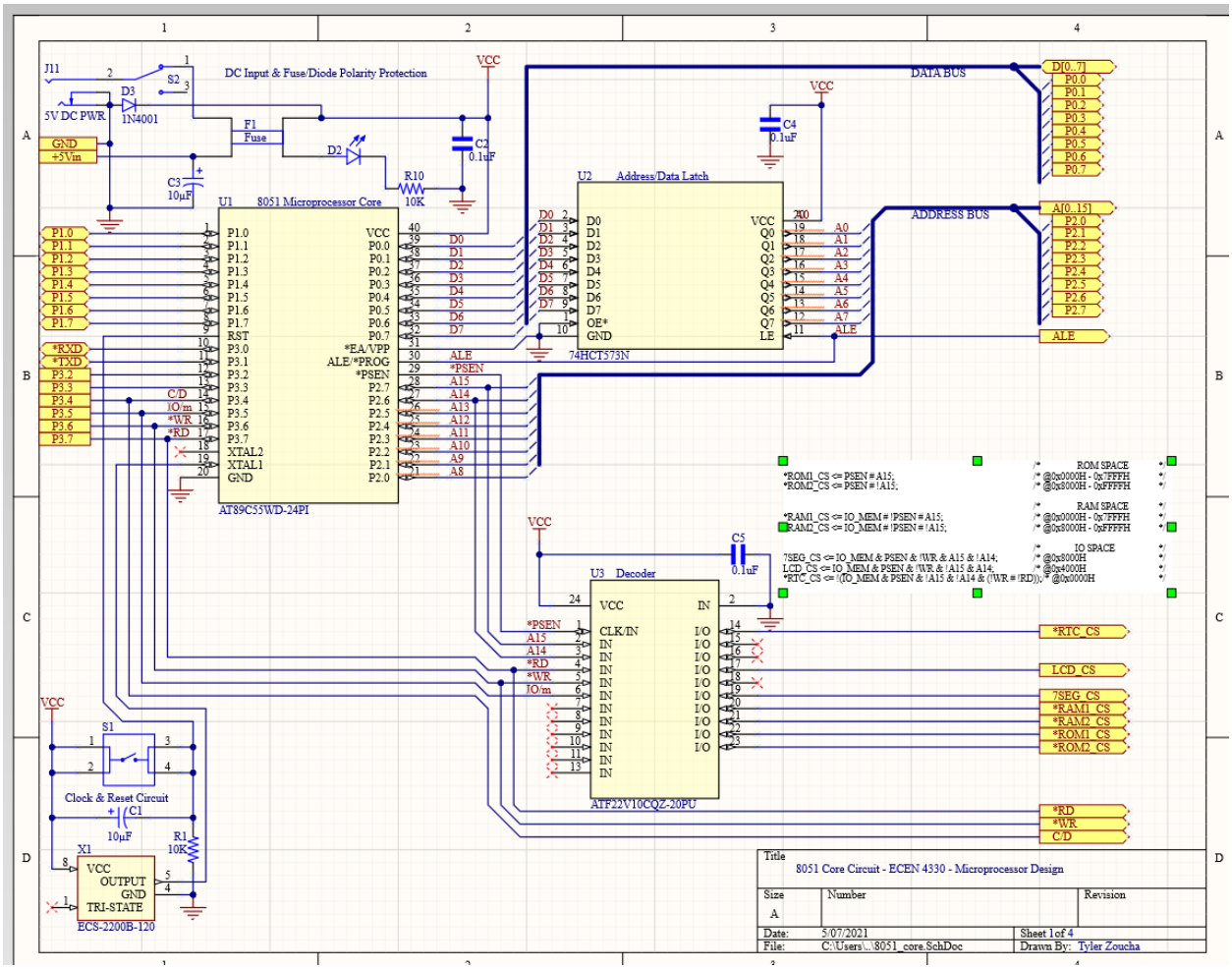
LCD additional wiki info:
http://www.lcdwiki.com/3.2inch_SPI_Module_ILI9341_SKU:MSP3218

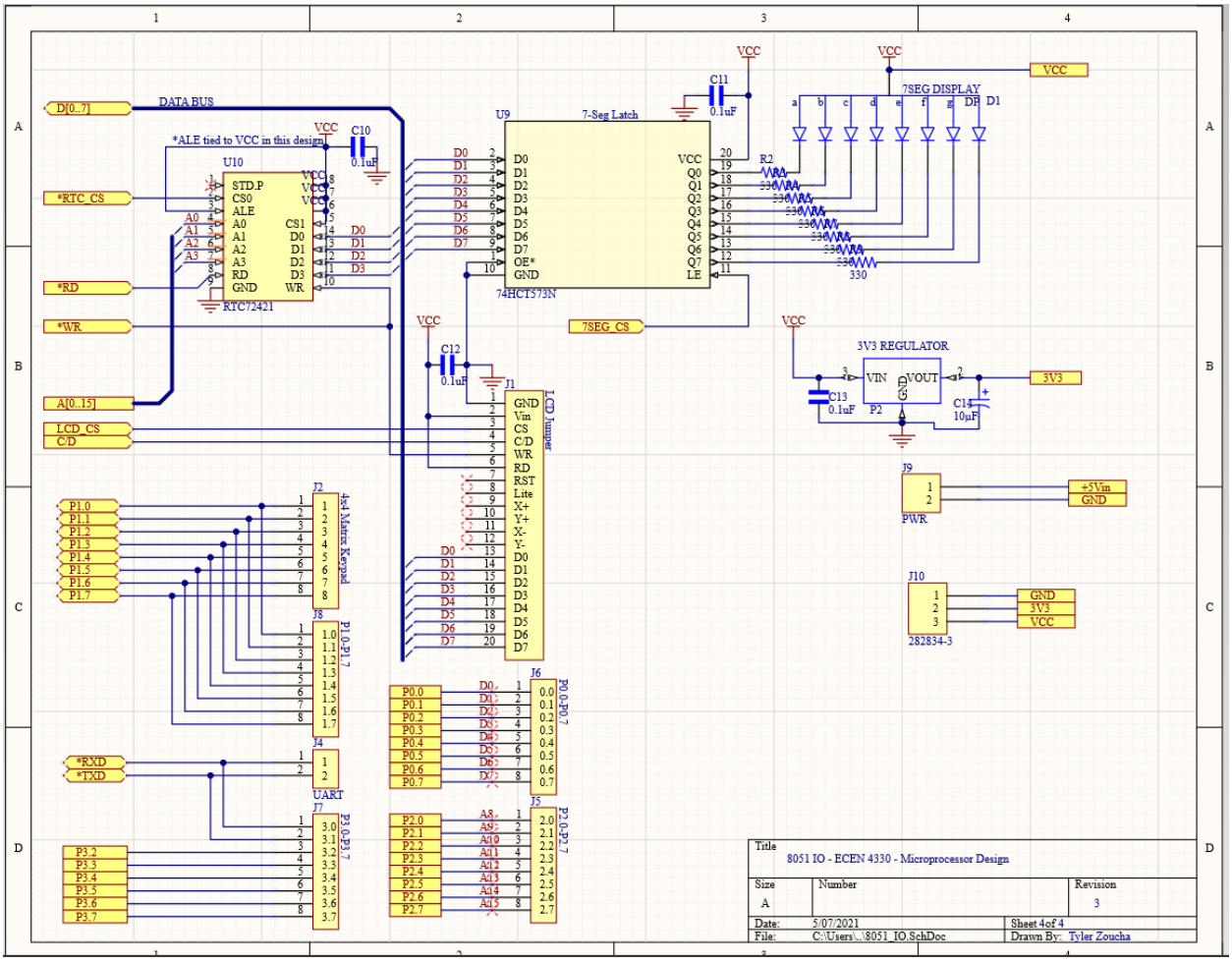
ESP8266 documentation:
https://www.microchip.ua/wireless/esp01.pdf
https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf

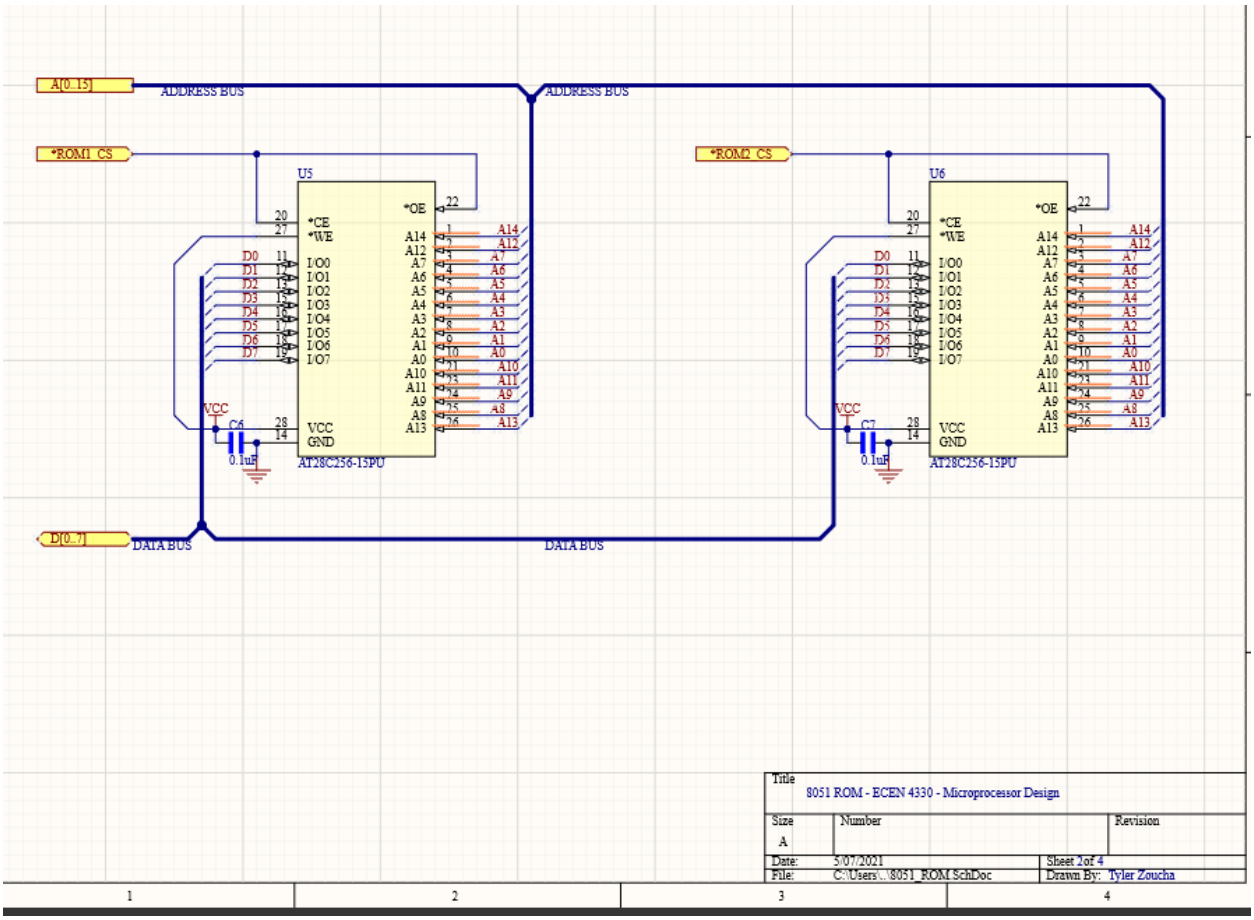
LC234x UART Controller:
https://www.ftdichip.com/Support/Documents/DataSheets/Modules/DS_LC234X.pdf

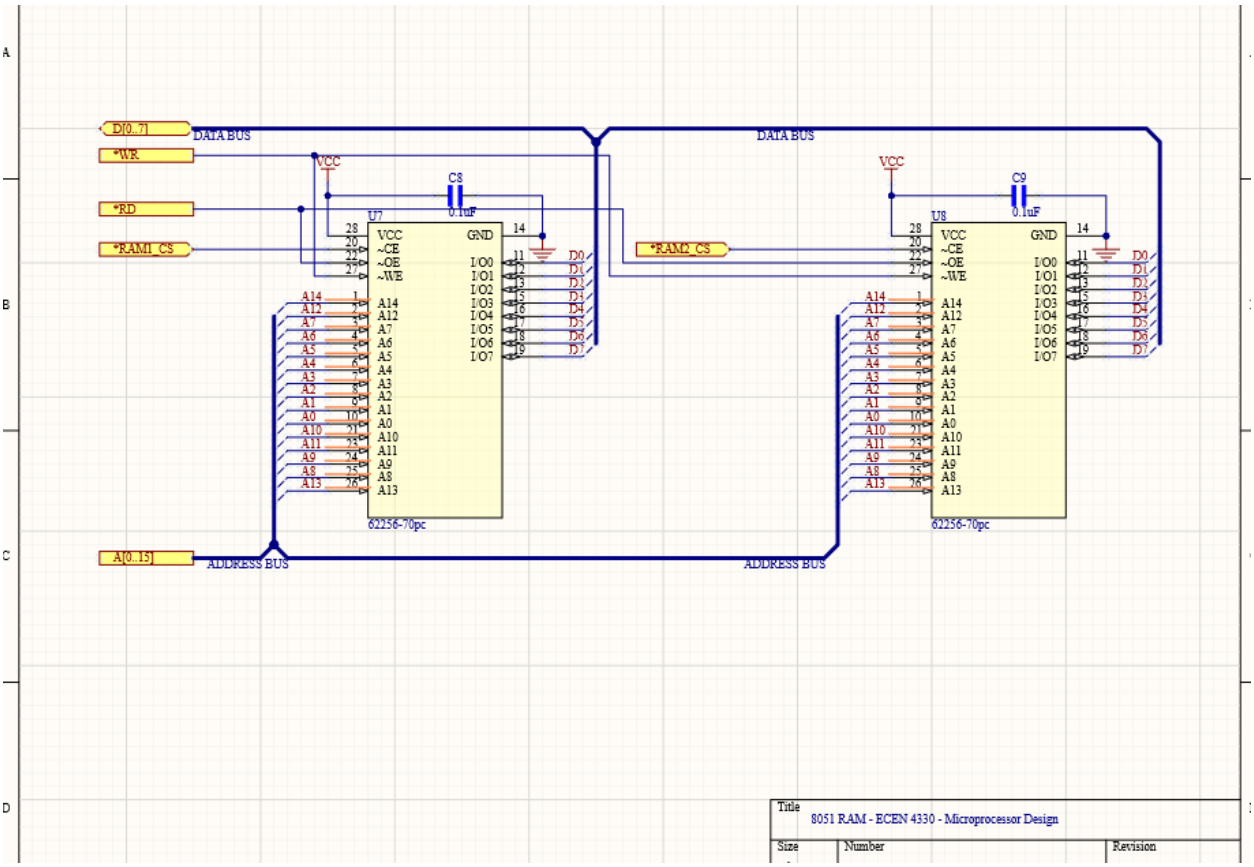
Bharat Acharya Education:
https://www.bharatacharyaeducation.com/

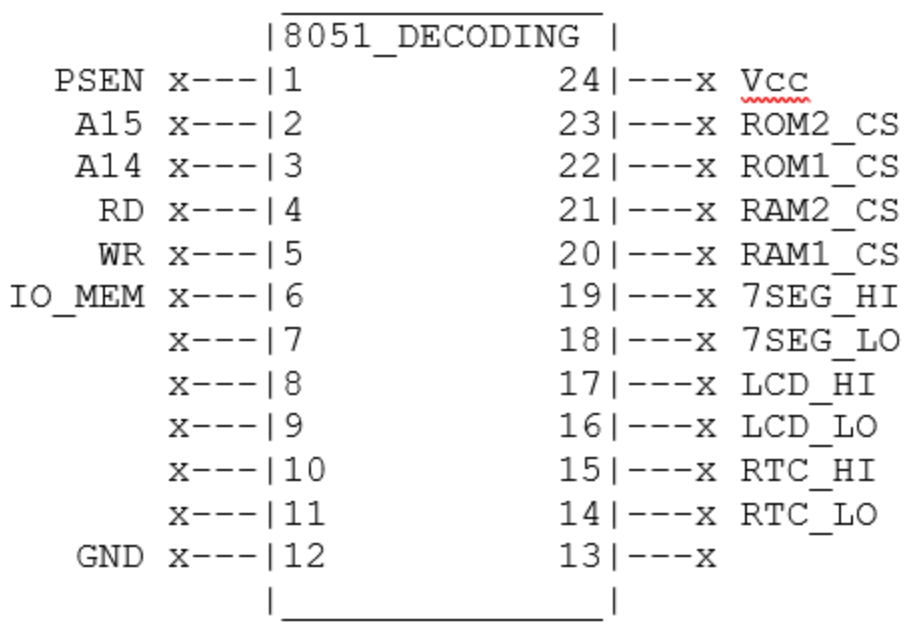
9) Appendix


DC Input & Fuse/Diode Polarity Protection
8051 Microprocessor Core
AT89C55WD-24PI
Address/Data Latch
74HCT573N
Decoder
ATF22V10CQZ-20PU
Clock & Reset Circuit
ECS-2200B-120
DATA BUS
ADDRESS BUS
D[0..7]
A[0..15]
ALE
*RTC_CS
LCD_CS
7SEG_CS
*RAM1_CS
*RAM2_CS
*ROM1_CS
*ROM2_CS
*RD
*WR
C/D
Title
8051 Core Circuit - ECEN 4330 - Microprocessor Design
Size
A
Number
Revision
Date: 5/07/2021
Sheet 1 of 4
File: C:\Users\..\8051_core.SchDoc
Drawn By: Tyler Zoucha


DATA BUS
D[0..7]
*ALE tied to VCC in this design
U10
RTC72421
*RTC_CS
*RD
*WR
A[0..15]
LCD_CS
C/D
U9
7-Seg Latch
74HCT573N
7SEG_CS
7SEG DISPLAY
3V3 REGULATOR
LCD Jumper
4x4 Matrix Keypad
UART
*RXD
*TXD
P1.0-P1.7
P0.0-P0.7
P2.0-P2.7
P3.2
P3.3
P3.4
P3.5
P3.6
P3.7
+5Vin
GND
3V3
VCC
Title
8051 IO - ECEN 4330 - Microprocessor Design
Size
A
Number
Revision
3
Date: 5/07/2021
Sheet 4 of 4
File: C:\Users\...\8051_IO.SchDoc
Drawn By: Tyler Zoucha

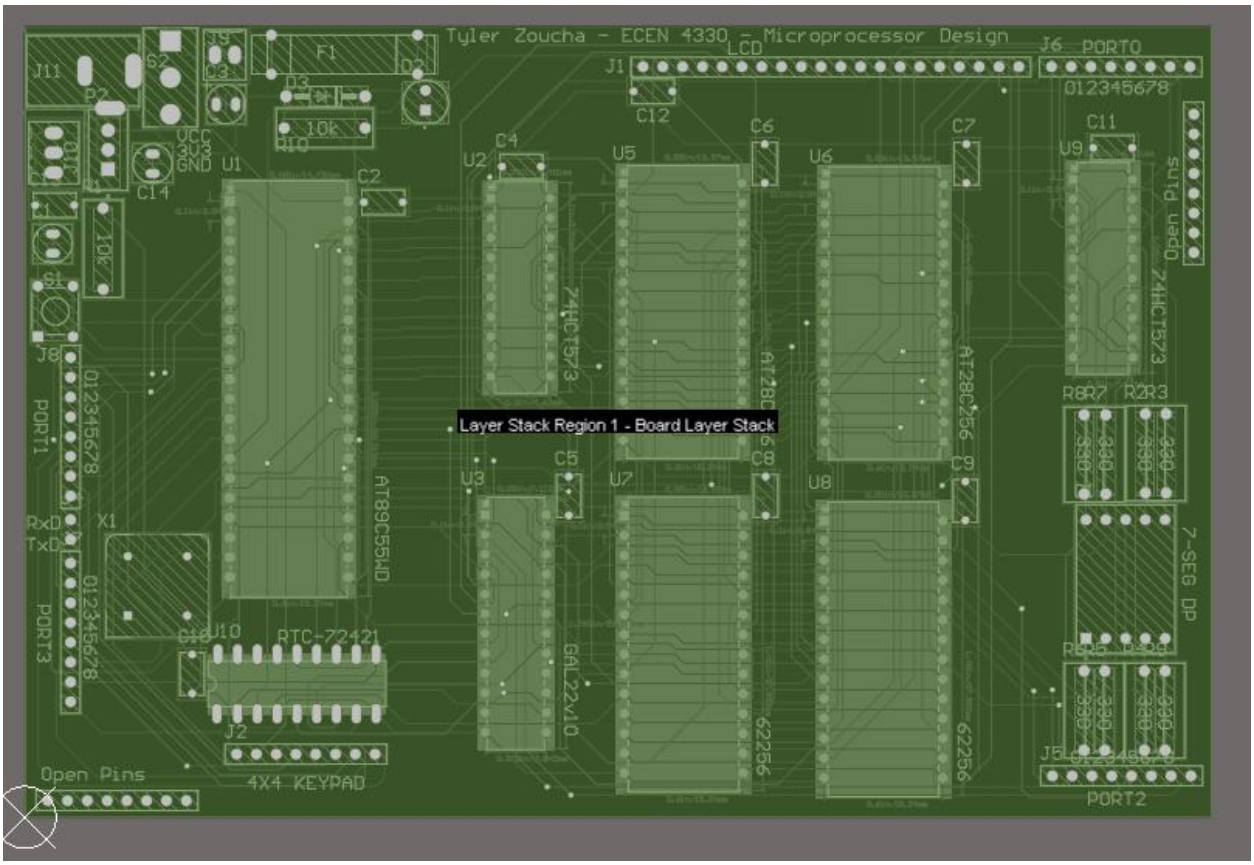

A[0..15]
ADDRESS BUS
ADDRESS BUS
*ROM1_CS
*ROM2_CS
U5
U6
*CE
*WE
*OE
I/O0
I/O1
I/O2
I/O3
I/O4
I/O5
I/O6
I/O7
VCC
GND
C6
C7
0.1uF
AT28C256-15PU
D[0..7]
DATA BUS
DATA BUS
Title
8051 ROM - ECEN 4330 - Microprocessor Design
Size
A
Number
Revision
Date: 5/07/2021
Sheet 2 of 4
File: C:\Users\...\8051_ROM.SchDoc
Drawn By: Tyler Zoucha

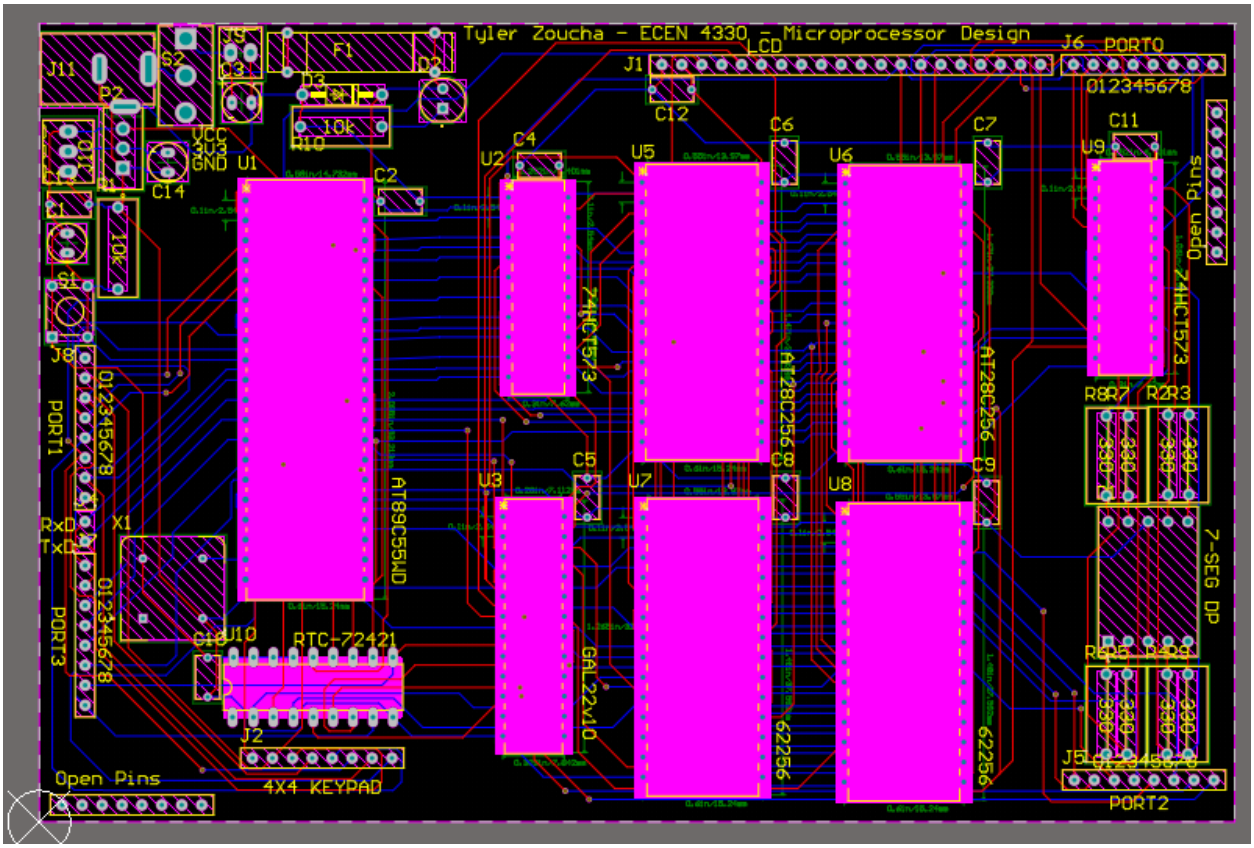

D[0..7]
DATA BUS
*WR
*RD
*RAM1_CS
*RAM2_CS
A[0..15]
ADDRESS BUS
VCC
C8
C9
0.1uF
U7
U8
62256-70pc
Title
8051 RAM - ECEN 4330 - Microprocessor Design
Size
Number
Revision

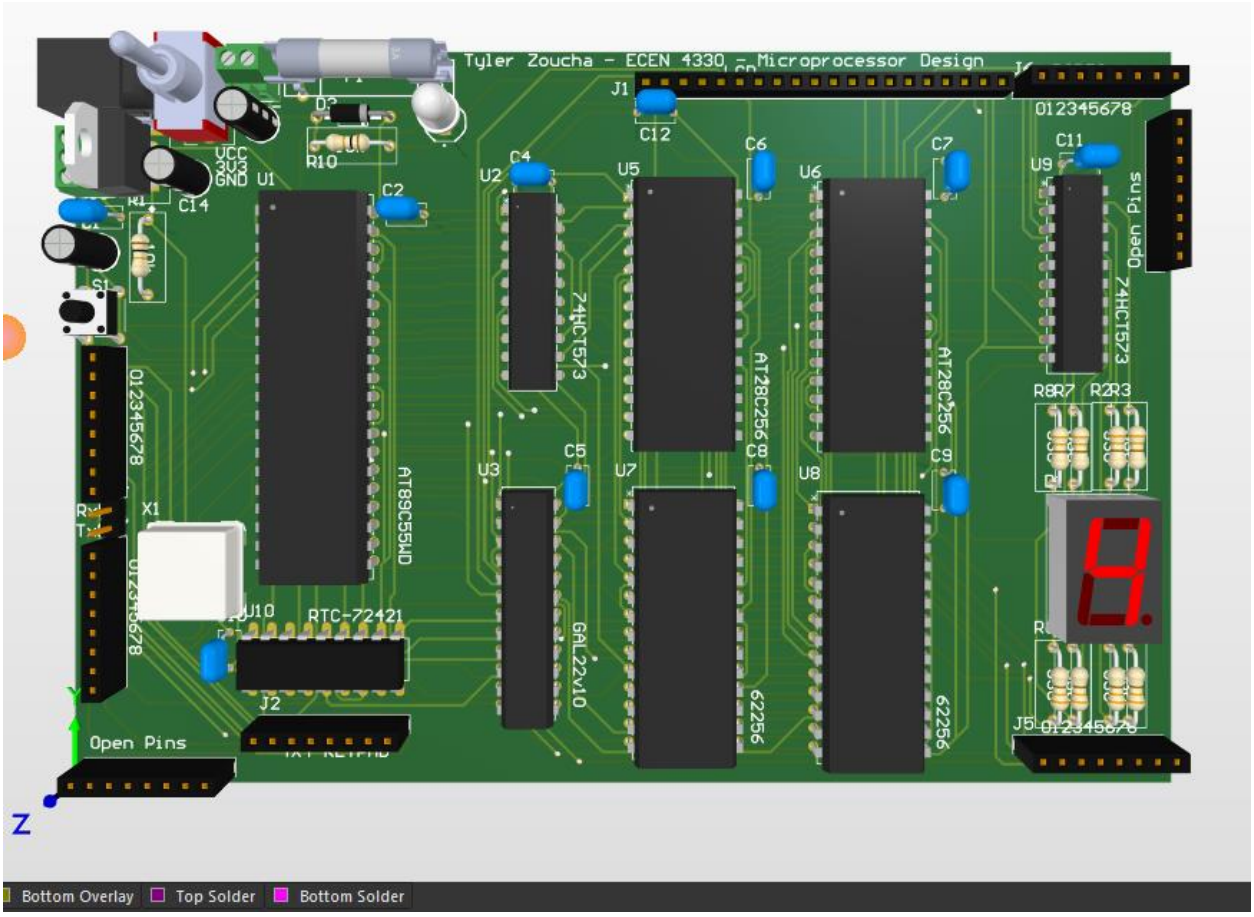


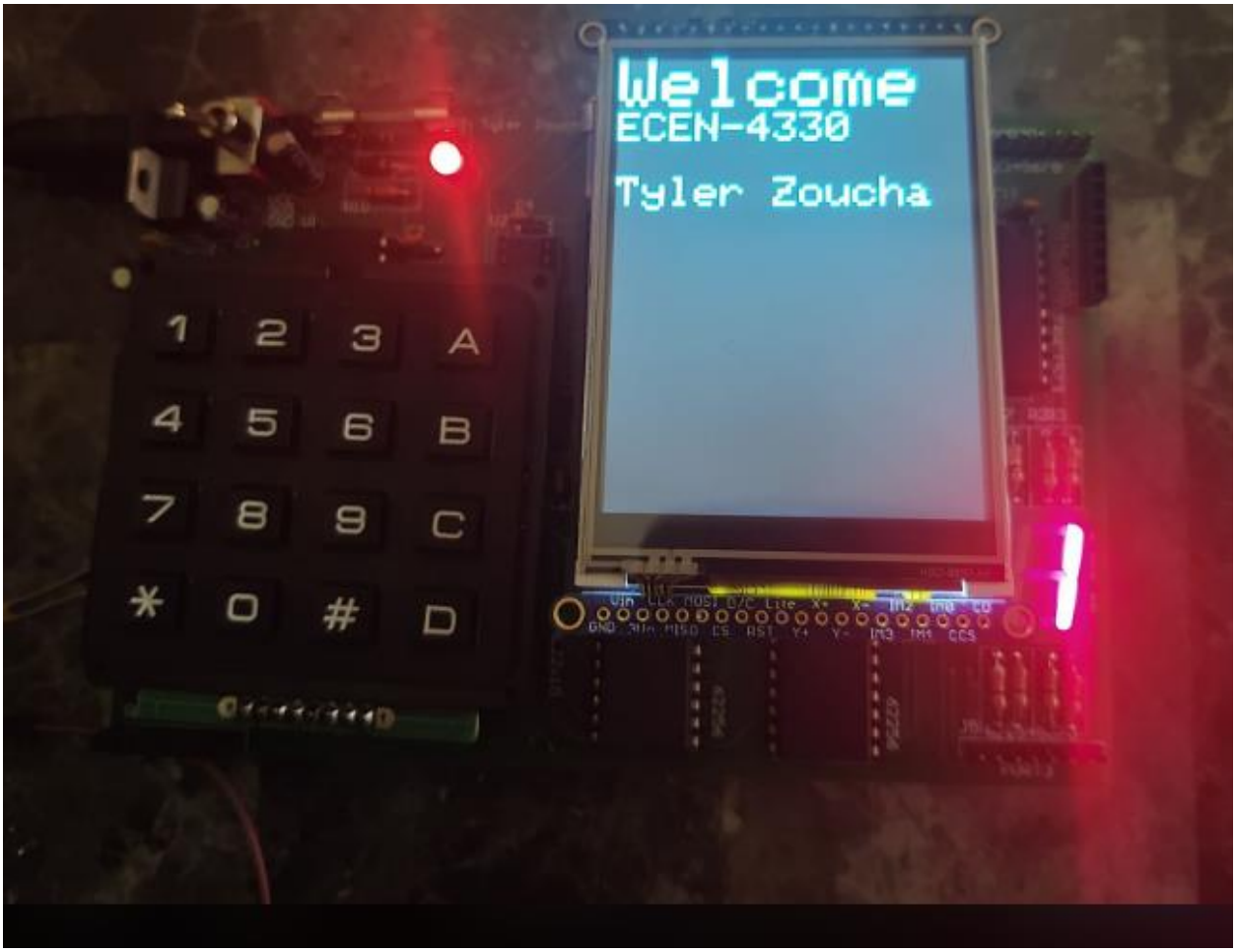

```
                                                        /*        ROM SPACE       */
*ROM1_CS <= PSEN # A15;                                 /* @0x0000H - 0x7FFFH      */
*ROM2_CS <= PSEN # !A15;                                /* @0x8000H - 0xFFFFH      */

                                                        /*        RAM SPACE       */
*RAM1_CS <= IO_MEM # !PSEN # A15;                       /* @0x0000H - 0x7FFFH      */
*RAM2_CS <= IO_MEM # !PSEN # !A15;                      /* @0x8000H - 0xFFFFH      */

                                                        /*        IO SPACE        */
7SEG_CS <= IO_MEM & PSEN & !WR & A15 & !A14;            /* @0x8000H                */
LCD_CS <= IO_MEM & PSEN & !WR & !A15 & A14;             /* @0x4000H                */
*RTC_CS <= !(IO_MEM & PSEN & !A15 & !A14 & (!WR # !RD));/* @0x0000H                */
```

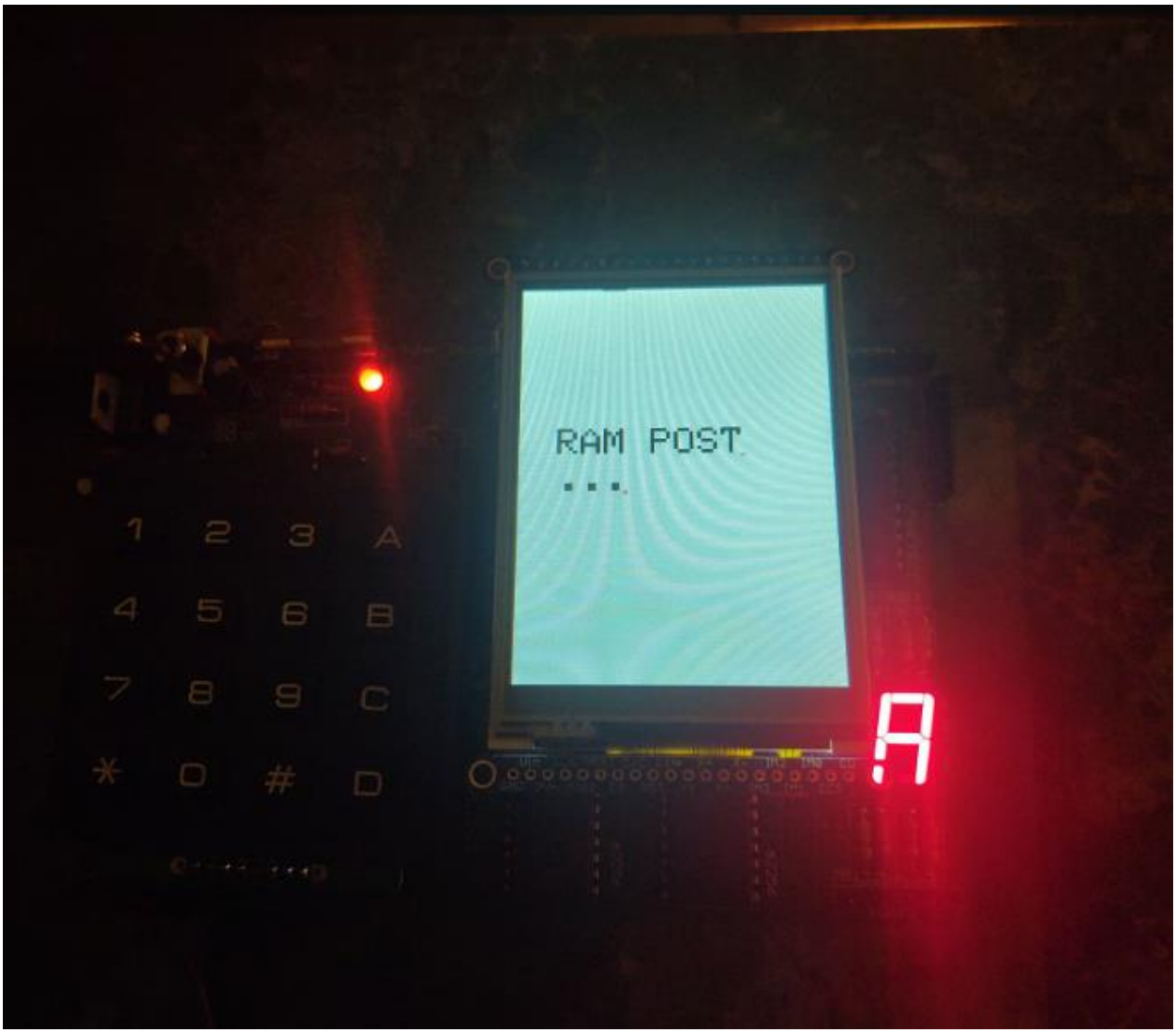

0
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D


Tyler Zoucha - ECEN 4330 - Microprocessor Design
Layer Stack Region 1 - Board Layer Stack
LCD
J1
J6 PORT0
012345678
C12
C6
C7
C11
U1
U2
U5
U6
U9
C2
C4
74HCT573
AT28C256
Open pins
J8
PORT1
012345678
R8R7
R2R3
330
RxD
TxD
X1
AT89C55WD
U3
U7
U8
C5
C8
C9
7-SEG DP
PORT3
012345678
U10
RTC-72421
C10
GAL22v10
62256
J2
4X4 KEYPAD
Open Pins
J5
PORT2


Tyler Zoucha - ECEN 4330 - Microprocessor Design
LCD
PORT0
012345678
Open Pins
74HCT573
AT28C256
AT89C55WD
RTC-72421
GAL22v10
62256
7-SEG DP
4X4 KEYPAD
PORT1
PORT2
PORT3
RxD
TxD
VCC
GND
10k
330


Tyler Zoucha - ECEN 4330 - Microprocessor Design
VCC
3V3
GND
U1
AT89C55WD
RTC-72421
74HCT573
AT28C256
62256
GAL22v10
Open Pins
Bottom Overlay
Top Solder
Bottom Solder


Welcome
ECEN-4330
Tyler Zoucha
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D


RAM POST
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D
A


[Menu]
D: DUMP
B: MOVE
E: EDIT
F: FIND
C: COUNT
A: MEM CHECK
1 2 3 A
4 5 6 B
7 8 9 C
* 0 # D
A

Code

```
/// all the required header for project here

# include <AT89C55.h>
# include "registers.h"
# include "bmp_image.h"

// define any address for lcd for address decoding
// use a latch
# define __LCD_ADDRESS__ 0x4000
# define __SEG_7_ADDRESS__ 0x8000
# define __RTC_ADDRESS__ 0x0000

// RTC address registers

# define __S1_REG__          __RTC_ADDRESS__ + 0x00
# define __S10_REG__         __RTC_ADDRESS__ + 0x01
# define __MI1_REG__         __RTC_ADDRESS__ + 0x02
# define __MI10_REG__        __RTC_ADDRESS__ + 0x03
# define __H1_REG__          __RTC_ADDRESS__ + 0x04
# define __H10_REG__         __RTC_ADDRESS__ + 0x05
# define __D1_REG__          __RTC_ADDRESS__ + 0x06
# define __D10_REG__         __RTC_ADDRESS__ + 0x07
# define __M1_REG__          __RTC_ADDRESS__ + 0x08
# define __M10_REG__         __RTC_ADDRESS__ + 0x09
# define __Y1_REG__          __RTC_ADDRESS__ + 0x0A
# define __Y10_REG__         __RTC_ADDRESS__ + 0x0B
# define __W_REG__           __RTC_ADDRESS__ + 0x0C
# define __REG_D__           __RTC_ADDRESS__ + 0x0D
# define __REG_E__           __RTC_ADDRESS__ + 0x0E
# define __REG_F__           __RTC_ADDRESS__ + 0x0F

// F register values
# define __HR_24__          0x04
# define __STOP__           0x02
# define __RESET__          0x01

# define __START_RAM__ 0x0000
# define __END_RAM__ 0xFFFE
/// LCD specific variables
// width and height of lcd in pixels
# define TFTWIDTH 240
# define TFTHEIGHT 320
```

```
// if needed to remeber command/data lines and active/idle signals

# define __ACTIVE__ 0
# define __IDLE__ 1
# define __CMD__ 0
# define __DATA__ 1

# define __KEYPAD_PORT__ P1


// defining important pins for LCD interfacing
// This is how it is defined for
# define IOM P3_5
# define CD P3_4


// definition of colors in 2-bytes
#define BLACK 0x0000
#define GRAY 0xD6BA
#define BLUE 0x001F
#define RED 0xF800
#define GREEN 0x07E0
#define CYAN 0x07FF
#define MAGENTA 0xF81F
#define YELLOW 0xFFE0
#define WHITE 0xFFFF

#define colorSelect CYAN
#define colorBackground WHITE
#define colorText BLACK

// variable definitions

#define u8 unsigned char
#define u16 unsigned int
#define u32 unsigned long


//
/// function declaration
```

```
void TFT_LCD_INIT(void);  // init function
void delay(int d);  // delay function for d ms

//void write8(u8 d);
//void write8Data(u8 d);
//void writeProbe(void);


void TFT_LCD_BEGIN(void);  // begin LCD
void writeRegister8(u8 a, u8 d);
void writeRegister16(u16 a, u16 d);
void fillScreen(unsigned int color);  // fill screen with the color defined
// set address to bound your operational area
void setAddress(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y);
// reset your LCD
void reset(void);



// draw PIXEL at one pixel
void drawPixel(u16 x3,u16 y3,u16 colour1);
// fill your LCD in operating region
void lcdfill(u16 xsta,u16 ysta,u16 xend,u16 yend,u16 color);


void fillRect(u16 x,u16 y,u16 w,u16 h,u16 color);

// draw a character
void drawchar(int x, int y, unsigned char c,u16 color, u16 bg, u8 size);
//void write(u8 c);
// set cursor in certain pixel
void setCursor(u16 x, u16 y);
// set textcolor
void setTextColor(u16 x, u16 y);
// set textsize
void setTextSize(u8 s);

// set string write
void LCD_string_write(char *str);
// dont really need this function
void drawGrayscaleBitmap(int x, int y, const u16 bitmap[], int w, int h, u16 colo
r) ;

// draw circles
void drawCircle(int x0, int y0, int r, u16 color);
```

```
// test circles
void testCircles(u8 radius, u16 color);
//void DrawRectangle(u16 x1, u16 y1, u16 x2, u16 y2);
u32 myPow(u8 m, u8 n);
void showNumberLCD(u16 x, u16 y, u32 num, u8 len);
void showNumber2LCD(u16 x, u16 y, u32 num, u8 len);

unsigned char keyDetect();
void testRAM(unsigned char d);
void freeType();
unsigned int reverse(unsigned char d);
unsigned int reverse16(unsigned int d);
void asciiToDec(unsigned char d);
void asciiToHex(unsigned char d);
void rtcInit(void);
void rtcBusy(void);
inline void rtcCmd(unsigned int addr, unsigned char d);
inline void rtcWrite(unsigned int addr, unsigned char d);
inline unsigned char rtcRead(unsigned int addr);
void rtcPrint(void);

inline void iowrite8(unsigned char __xdata* map_address, unsigned char d);
inline unsigned char ioread8(unsigned char __xdata* map_address);
inline void ramWrite8(unsigned char __xdata* map_address, unsigned char d);
inline unsigned char ramRead8(unsigned char __xdata* map_address);


void printMenu(void);                           // display main menu

void dump(void);                                // memory dump
void move(void);                                // memory move
void edit(void);                                // memory edit
void find(void);                                // memory find

unsigned int charToASCII(unsigned char key);      // convert char to ascii
unsigned int charToInt(unsigned char key);      // convert char to int

void print4Hex(unsigned char num);              // print 4 bit hex value
void print8Hex(unsigned char num);              // print 8 bit hex value
void print8ASCII(unsigned char num);            // print 8 bit ASCII value
void print16Hex(unsigned int num);              // print 16 bit hex value
void print16ASCII(unsigned int num);            // print 16 bit ASCII value
void print16Dec(unsigned int num);              // print 16 bit decimal value
```

```
// for default font
static const unsigned char font[] PROGMEM = {
    0x00, 0x00, 0x00, 0x00, 0x00,
    0x3E, 0x5B, 0x4F, 0x5B, 0x3E,
    0x3E, 0x6B, 0x4F, 0x6B, 0x3E,
    0x1C, 0x3E, 0x7C, 0x3E, 0x1C,
    0x18, 0x3C, 0x7E, 0x3C, 0x18,
    0x1C, 0x57, 0x7D, 0x57, 0x1C,
    0x1C, 0x5E, 0x7F, 0x5E, 0x1C,
    0x00, 0x18, 0x3C, 0x18, 0x00,
    0xFF, 0xE7, 0xC3, 0xE7, 0xFF,
    0x00, 0x18, 0x24, 0x18, 0x00,
    0xFF, 0xE7, 0xDB, 0xE7, 0xFF,
    0x30, 0x48, 0x3A, 0x06, 0x0E,
    0x26, 0x29, 0x79, 0x29, 0x26,
    0x40, 0x7F, 0x05, 0x05, 0x07,
    0x40, 0x7F, 0x05, 0x25, 0x3F,
    0x5A, 0x3C, 0xE7, 0x3C, 0x5A,
    0x7F, 0x3E, 0x1C, 0x1C, 0x08,
    0x08, 0x1C, 0x1C, 0x3E, 0x7F,
    0x14, 0x22, 0x7F, 0x22, 0x14,
    0x5F, 0x5F, 0x00, 0x5F, 0x5F,
    0x06, 0x09, 0x7F, 0x01, 0x7F,
    0x00, 0x66, 0x89, 0x95, 0x6A,
    0x60, 0x60, 0x60, 0x60, 0x60,
    0x94, 0xA2, 0xFF, 0xA2, 0x94,
    0x08, 0x04, 0x7E, 0x04, 0x08,
    0x10, 0x20, 0x7E, 0x20, 0x10,
    0x08, 0x08, 0x2A, 0x1C, 0x08,
    0x08, 0x1C, 0x2A, 0x08, 0x08,
    0x1E, 0x10, 0x10, 0x10, 0x10,
    0x0C, 0x1E, 0x0C, 0x1E, 0x0C,
    0x30, 0x38, 0x3E, 0x38, 0x30,
    0x06, 0x0E, 0x3E, 0x0E, 0x06,
    0x00, 0x00, 0x00, 0x00, 0x00,
    0x00, 0x00, 0x5F, 0x00, 0x00,
    0x00, 0x07, 0x00, 0x07, 0x00,
    0x14, 0x7F, 0x14, 0x7F, 0x14,
    0x24, 0x2A, 0x7F, 0x2A, 0x12,
    0x23, 0x13, 0x08, 0x64, 0x62,
    0x36, 0x49, 0x56, 0x20, 0x50,
    0x00, 0x08, 0x07, 0x03, 0x00,
    0x00, 0x1C, 0x22, 0x41, 0x00,
    0x00, 0x41, 0x22, 0x1C, 0x00,
```

```
    0x2A, 0x1C, 0x7F, 0x1C, 0x2A,
    0x08, 0x08, 0x3E, 0x08, 0x08,
    0x00, 0x80, 0x70, 0x30, 0x00,
    0x08, 0x08, 0x08, 0x08, 0x08,
    0x00, 0x00, 0x60, 0x60, 0x00,
    0x20, 0x10, 0x08, 0x04, 0x02,
    0x3E, 0x51, 0x49, 0x45, 0x3E,
    0x00, 0x42, 0x7F, 0x40, 0x00,
    0x72, 0x49, 0x49, 0x49, 0x46,
    0x21, 0x41, 0x49, 0x4D, 0x33,
    0x18, 0x14, 0x12, 0x7F, 0x10,
    0x27, 0x45, 0x45, 0x45, 0x39,
    0x3C, 0x4A, 0x49, 0x49, 0x31,
    0x41, 0x21, 0x11, 0x09, 0x07,
    0x36, 0x49, 0x49, 0x49, 0x36,
    0x46, 0x49, 0x49, 0x29, 0x1E,
    0x00, 0x00, 0x14, 0x00, 0x00,
    0x00, 0x40, 0x34, 0x00, 0x00,
    0x00, 0x08, 0x14, 0x22, 0x41,
    0x14, 0x14, 0x14, 0x14, 0x14,
    0x00, 0x41, 0x22, 0x14, 0x08,
    0x02, 0x01, 0x59, 0x09, 0x06,
    0x3E, 0x41, 0x5D, 0x59, 0x4E,
    0x7C, 0x12, 0x11, 0x12, 0x7C,
    0x7F, 0x49, 0x49, 0x49, 0x36,
    0x3E, 0x41, 0x41, 0x41, 0x22,
    0x7F, 0x41, 0x41, 0x41, 0x3E,
    0x7F, 0x49, 0x49, 0x49, 0x41,
    0x7F, 0x09, 0x09, 0x09, 0x01,
    0x3E, 0x41, 0x41, 0x51, 0x73,
    0x7F, 0x08, 0x08, 0x08, 0x7F,
    0x00, 0x41, 0x7F, 0x41, 0x00,
    0x20, 0x40, 0x41, 0x3F, 0x01,
    0x7F, 0x08, 0x14, 0x22, 0x41,
    0x7F, 0x40, 0x40, 0x40, 0x40,
    0x7F, 0x02, 0x1C, 0x02, 0x7F,
    0x7F, 0x04, 0x08, 0x10, 0x7F,
    0x3E, 0x41, 0x41, 0x41, 0x3E,
    0x7F, 0x09, 0x09, 0x09, 0x06,
    0x3E, 0x41, 0x51, 0x21, 0x5E,
    0x7F, 0x09, 0x19, 0x29, 0x46,
    0x26, 0x49, 0x49, 0x49, 0x32,
    0x03, 0x01, 0x7F, 0x01, 0x03,
    0x3F, 0x40, 0x40, 0x40, 0x3F,
```

```
    0x1F, 0x20, 0x40, 0x20, 0x1F,
    0x3F, 0x40, 0x38, 0x40, 0x3F,
    0x63, 0x14, 0x08, 0x14, 0x63,
    0x03, 0x04, 0x78, 0x04, 0x03,
    0x61, 0x59, 0x49, 0x4D, 0x43,
    0x00, 0x7F, 0x41, 0x41, 0x41,
    0x02, 0x04, 0x08, 0x10, 0x20,
    0x00, 0x41, 0x41, 0x41, 0x7F,
    0x04, 0x02, 0x01, 0x02, 0x04,
    0x40, 0x40, 0x40, 0x40, 0x40,
    0x00, 0x03, 0x07, 0x08, 0x00,
    0x20, 0x54, 0x54, 0x78, 0x40,
    0x7F, 0x28, 0x44, 0x44, 0x38,
    0x38, 0x44, 0x44, 0x44, 0x28,
    0x38, 0x44, 0x44, 0x28, 0x7F,
    0x38, 0x54, 0x54, 0x54, 0x18,
    0x00, 0x08, 0x7E, 0x09, 0x02,
    0x18, 0xA4, 0xA4, 0x9C, 0x78,
    0x7F, 0x08, 0x04, 0x04, 0x78,
    0x00, 0x44, 0x7D, 0x40, 0x00,
    0x20, 0x40, 0x40, 0x3D, 0x00,
    0x7F, 0x10, 0x28, 0x44, 0x00,
    0x00, 0x41, 0x7F, 0x40, 0x00,
    0x7C, 0x04, 0x78, 0x04, 0x78,
    0x7C, 0x08, 0x04, 0x04, 0x78,
    0x38, 0x44, 0x44, 0x44, 0x38,
    0xFC, 0x18, 0x24, 0x24, 0x18,
    0x18, 0x24, 0x24, 0x18, 0xFC,
    0x7C, 0x08, 0x04, 0x04, 0x08,
    0x48, 0x54, 0x54, 0x54, 0x24,
    0x04, 0x04, 0x3F, 0x44, 0x24,
    0x3C, 0x40, 0x40, 0x20, 0x7C,
    0x1C, 0x20, 0x40, 0x20, 0x1C,
    0x3C, 0x40, 0x30, 0x40, 0x3C,
    0x44, 0x28, 0x10, 0x28, 0x44,
    0x4C, 0x90, 0x90, 0x90, 0x7C,
    0x44, 0x64, 0x54, 0x4C, 0x44,
    0x00, 0x08, 0x36, 0x41, 0x00,
    0x00, 0x00, 0x77, 0x00, 0x00,
    0x00, 0x41, 0x36, 0x08, 0x00,
    0x02, 0x01, 0x02, 0x04, 0x02,
    0x3C, 0x26, 0x23, 0x26, 0x3C,
    0x1E, 0xA1, 0xA1, 0x61, 0x12,
    0x3A, 0x40, 0x40, 0x20, 0x7A,
```

```
    0x38, 0x54, 0x54, 0x55, 0x59,
    0x21, 0x55, 0x55, 0x79, 0x41,
    0x22, 0x54, 0x54, 0x78, 0x42, // a-umlaut
    0x21, 0x55, 0x54, 0x78, 0x40,
    0x20, 0x54, 0x55, 0x79, 0x40,
    0x0C, 0x1E, 0x52, 0x72, 0x12,
    0x39, 0x55, 0x55, 0x55, 0x59,
    0x39, 0x54, 0x54, 0x54, 0x59,
    0x39, 0x55, 0x54, 0x54, 0x58,
    0x00, 0x00, 0x45, 0x7C, 0x41,
    0x00, 0x02, 0x45, 0x7D, 0x42,
    0x00, 0x01, 0x45, 0x7C, 0x40,
    0x7D, 0x12, 0x11, 0x12, 0x7D, // A-umlaut
    0xF0, 0x28, 0x25, 0x28, 0xF0,
    0x7C, 0x54, 0x55, 0x45, 0x00,
    0x20, 0x54, 0x54, 0x7C, 0x54,
    0x7C, 0x0A, 0x09, 0x7F, 0x49,
    0x32, 0x49, 0x49, 0x49, 0x32,
    0x3A, 0x44, 0x44, 0x44, 0x3A, // o-umlaut
    0x32, 0x4A, 0x48, 0x48, 0x30,
    0x3A, 0x41, 0x41, 0x21, 0x7A,
    0x3A, 0x42, 0x40, 0x20, 0x78,
    0x00, 0x9D, 0xA0, 0xA0, 0x7D,
    0x3D, 0x42, 0x42, 0x42, 0x3D, // O-umlaut
    0x3D, 0x40, 0x40, 0x40, 0x3D,
    0x3C, 0x24, 0xFF, 0x24, 0x24,
    0x48, 0x7E, 0x49, 0x43, 0x66,
    0x2B, 0x2F, 0xFC, 0x2F, 0x2B,
    0xFF, 0x09, 0x29, 0xF6, 0x20,
    0xC0, 0x88, 0x7E, 0x09, 0x03,
    0x20, 0x54, 0x54, 0x79, 0x41,
    0x00, 0x00, 0x44, 0x7D, 0x41,
    0x30, 0x48, 0x48, 0x4A, 0x32,
    0x38, 0x40, 0x40, 0x22, 0x7A,
    0x00, 0x7A, 0x0A, 0x0A, 0x72,
    0x7D, 0x0D, 0x19, 0x31, 0x7D,
    0x26, 0x29, 0x29, 0x2F, 0x28,
    0x26, 0x29, 0x29, 0x29, 0x26,
    0x30, 0x48, 0x4D, 0x40, 0x20,
    0x38, 0x08, 0x08, 0x08, 0x08,
    0x08, 0x08, 0x08, 0x08, 0x38,
    0x2F, 0x10, 0xC8, 0xAC, 0xBA,
    0x2F, 0x10, 0x28, 0x34, 0xFA,
    0x00, 0x00, 0x7B, 0x00, 0x00,
```

```
    0x08, 0x14, 0x2A, 0x14, 0x22,
    0x22, 0x14, 0x2A, 0x14, 0x08,
    0xAA, 0x00, 0x55, 0x00, 0xAA,
    0xAA, 0x55, 0xAA, 0x55, 0xAA,
    0x00, 0x00, 0x00, 0xFF, 0x00,
    0x10, 0x10, 0x10, 0xFF, 0x00,
    0x14, 0x14, 0x14, 0xFF, 0x00,
    0x10, 0x10, 0xFF, 0x00, 0xFF,
    0x10, 0x10, 0xF0, 0x10, 0xF0,
    0x14, 0x14, 0x14, 0xFC, 0x00,
    0x14, 0x14, 0xF7, 0x00, 0xFF,
    0x00, 0x00, 0xFF, 0x00, 0xFF,
    0x14, 0x14, 0xF4, 0x04, 0xFC,
    0x14, 0x14, 0x17, 0x10, 0x1F,
    0x10, 0x10, 0x1F, 0x10, 0x1F,
    0x14, 0x14, 0x14, 0x1F, 0x00,
    0x10, 0x10, 0x10, 0xF0, 0x00,
    0x00, 0x00, 0x00, 0x1F, 0x10,
    0x10, 0x10, 0x10, 0x1F, 0x10,
    0x10, 0x10, 0x10, 0xF0, 0x10,
    0x00, 0x00, 0x00, 0xFF, 0x10,
    0x10, 0x10, 0x10, 0x10, 0x10,
    0x10, 0x10, 0x10, 0xFF, 0x10,
    0x00, 0x00, 0x00, 0xFF, 0x14,
    0x00, 0x00, 0xFF, 0x00, 0xFF,
    0x00, 0x00, 0x1F, 0x10, 0x17,
    0x00, 0x00, 0xFC, 0x04, 0xF4,
    0x14, 0x14, 0x17, 0x10, 0x17,
    0x14, 0x14, 0xF4, 0x04, 0xF4,
    0x00, 0x00, 0xFF, 0x00, 0xF7,
    0x14, 0x14, 0x14, 0x14, 0x14,
    0x14, 0x14, 0xF7, 0x00, 0xF7,
    0x14, 0x14, 0x14, 0x17, 0x14,
    0x10, 0x10, 0x1F, 0x10, 0x1F,
    0x14, 0x14, 0x14, 0xF4, 0x14,
    0x10, 0x10, 0xF0, 0x10, 0xF0,
    0x00, 0x00, 0x1F, 0x10, 0x1F,
    0x00, 0x00, 0x00, 0x1F, 0x14,
    0x00, 0x00, 0x00, 0xFC, 0x14,
    0x00, 0x00, 0xF0, 0x10, 0xF0,
    0x10, 0x10, 0xFF, 0x10, 0xFF,
    0x14, 0x14, 0x14, 0xFF, 0x14,
    0x10, 0x10, 0x10, 0x1F, 0x00,
    0x00, 0x00, 0x00, 0xF0, 0x10,
```

```
    0xFF, 0xFF, 0xFF, 0xFF, 0xFF,
    0xF0, 0xF0, 0xF0, 0xF0, 0xF0,
    0xFF, 0xFF, 0xFF, 0x00, 0x00,
    0x00, 0x00, 0x00, 0xFF, 0xFF,
    0x0F, 0x0F, 0x0F, 0x0F, 0x0F,
    0x38, 0x44, 0x44, 0x38, 0x44,
    0xFC, 0x4A, 0x4A, 0x4A, 0x34, // sharp-s or beta
    0x7E, 0x02, 0x02, 0x06, 0x06,
    0x02, 0x7E, 0x02, 0x7E, 0x02,
    0x63, 0x55, 0x49, 0x41, 0x63,
    0x38, 0x44, 0x44, 0x3C, 0x04,
    0x40, 0x7E, 0x20, 0x1E, 0x20,
    0x06, 0x02, 0x7E, 0x02, 0x02,
    0x99, 0xA5, 0xE7, 0xA5, 0x99,
    0x1C, 0x2A, 0x49, 0x2A, 0x1C,
    0x4C, 0x72, 0x01, 0x72, 0x4C,
    0x30, 0x4A, 0x4D, 0x4D, 0x30,
    0x30, 0x48, 0x78, 0x48, 0x30,
    0xBC, 0x62, 0x5A, 0x46, 0x3D,
    0x3E, 0x49, 0x49, 0x49, 0x00,
    0x7E, 0x01, 0x01, 0x01, 0x7E,
    0x2A, 0x2A, 0x2A, 0x2A, 0x2A,
    0x44, 0x44, 0x5F, 0x44, 0x44,
    0x40, 0x51, 0x4A, 0x44, 0x40,
    0x40, 0x44, 0x4A, 0x51, 0x40,
    0x00, 0x00, 0xFF, 0x01, 0x03,
    0xE0, 0x80, 0xFF, 0x00, 0x00,
    0x08, 0x08, 0x6B, 0x6B, 0x08,
    0x36, 0x12, 0x36, 0x24, 0x36,
    0x06, 0x0F, 0x09, 0x0F, 0x06,
    0x00, 0x00, 0x18, 0x18, 0x00,
    0x00, 0x00, 0x10, 0x10, 0x00,
    0x30, 0x40, 0xFF, 0x01, 0x01,
    0x00, 0x1F, 0x01, 0x01, 0x1E,
    0x00, 0x19, 0x1D, 0x17, 0x12,
    0x00, 0x3C, 0x3C, 0x3C, 0x3C,
    0x00, 0x00, 0x00, 0x00, 0x00
};
```

```
// Register names from Peter Barrett's Microtouch code
#define ILI932X_START_OSC          0x00
#define ILI932X_DRIV_OUT_CTRL      0x01
#define ILI932X_DRIV_WAV_CTRL      0x02
#define ILI932X_ENTRY_MOD          0x03
#define ILI932X_RESIZE_CTRL        0x04
#define ILI932X_DISP_CTRL1         0x07
#define ILI932X_DISP_CTRL2         0x08
#define ILI932X_DISP_CTRL3         0x09
#define ILI932X_DISP_CTRL4         0x0A
#define ILI932X_RGB_DISP_IF_CTRL1  0x0C
#define ILI932X_FRM_MARKER_POS     0x0D
#define ILI932X_RGB_DISP_IF_CTRL2  0x0F
#define ILI932X_POW_CTRL1          0x10
#define ILI932X_POW_CTRL2          0x11
#define ILI932X_POW_CTRL3          0x12
#define ILI932X_POW_CTRL4          0x13
#define ILI932X_GRAM_HOR_AD        0x20
#define ILI932X_GRAM_VER_AD        0x21
#define ILI932X_RW_GRAM            0x22
#define ILI932X_POW_CTRL7          0x29
#define ILI932X_FRM_RATE_COL_CTRL  0x2B
#define ILI932X_GAMMA_CTRL1        0x30
#define ILI932X_GAMMA_CTRL2        0x31
#define ILI932X_GAMMA_CTRL3        0x32
#define ILI932X_GAMMA_CTRL4        0x35
#define ILI932X_GAMMA_CTRL5        0x36
#define ILI932X_GAMMA_CTRL6        0x37
#define ILI932X_GAMMA_CTRL7        0x38
#define ILI932X_GAMMA_CTRL8        0x39
#define ILI932X_GAMMA_CTRL9        0x3C
#define ILI932X_GAMMA_CTRL10       0x3D
#define ILI932X_HOR_START_AD       0x50
#define ILI932X_HOR_END_AD         0x51
#define ILI932X_VER_START_AD       0x52
#define ILI932X_VER_END_AD         0x53
#define ILI932X_GATE_SCAN_CTRL1    0x60
#define ILI932X_GATE_SCAN_CTRL2    0x61
#define ILI932X_GATE_SCAN_CTRL3    0x6A
#define ILI932X_PART_IMG1_DISP_POS 0x80
#define ILI932X_PART_IMG1_START_AD 0x81
#define ILI932X_PART_IMG1_END_AD   0x82
#define ILI932X_PART_IMG2_DISP_POS 0x83
#define ILI932X_PART_IMG2_START_AD 0x84
```

```
#define ILI932X_PART_IMG2_END_AD   0x85
#define ILI932X_PANEL_IF_CTRL1     0x90
#define ILI932X_PANEL_IF_CTRL2     0x92
#define ILI932X_PANEL_IF_CTRL3     0x93
#define ILI932X_PANEL_IF_CTRL4     0x95
#define ILI932X_PANEL_IF_CTRL5     0x97
#define ILI932X_PANEL_IF_CTRL6     0x98

#define HX8347G_COLADDRSTART_HI    0x02
#define HX8347G_COLADDRSTART_LO    0x03
#define HX8347G_COLADDREND_HI      0x04
#define HX8347G_COLADDREND_LO      0x05
#define HX8347G_ROWADDRSTART_HI    0x06
#define HX8347G_ROWADDRSTART_LO    0x07
#define HX8347G_ROWADDREND_HI      0x08
#define HX8347G_ROWADDREND_LO      0x09
#define HX8347G_MEMACCESS          0x16



#define ILI9341_SOFTRESET          0x01
#define ILI9341_SLEEPIN            0x10
#define ILI9341_SLEEPOUT           0x11
#define ILI9341_NORMALDISP         0x13
#define ILI9341_INVERTOFF          0x20
#define ILI9341_INVERTON           0x21
#define ILI9341_GAMMASET           0x26
#define ILI9341_DISPLAYOFF         0x28
#define ILI9341_DISPLAYON          0x29
#define ILI9341_COLADDRSET         0x2A
#define ILI9341_PAGEADDRSET        0x2B
#define ILI9341_MEMORYWRITE        0x2C
#define ILI9341_PIXELFORMAT        0x3A
#define ILI9341_FRAMECONTROL       0xB1
#define ILI9341_DISPLAYFUNC        0xB6
#define ILI9341_ENTRYMODE          0xB7
#define ILI9341_POWERCONTROL1      0xC0
#define ILI9341_POWERCONTROL2      0xC1
#define ILI9341_VCOMCONTROL1      0xC5
#define ILI9341_VCOMCONTROL2      0xC7
#define ILI9341_MEMCONTROL      0x36
#define ILI9341_MADCTL  0x36

#define ILI9341_MADCTL_MY  0x80
#define ILI9341_MADCTL_MX  0x40
```

```
#define ILI9341_MADCTL_MV  0x20
#define ILI9341_MADCTL_ML  0x10
#define ILI9341_MADCTL_RGB 0x00
#define ILI9341_MADCTL_BGR 0x08
#define ILI9341_MADCTL_MH  0x04



#define HX8357_NOP     0x00
#define HX8357_SWRESET 0x01
#define HX8357_RDDID   0x04
#define HX8357_RDDST   0x09

#define HX8357B_RDPOWMODE  0x0A
#define HX8357B_RDMADCTL  0x0B
#define HX8357B_RDCOLMOD  0x0C
#define HX8357B_RDDIM  0x0D
#define HX8357B_RDDSDR  0x0F

#define HX8357_SLPIN   0x10
#define HX8357_SLPOUT  0x11
#define HX8357B_PTLON   0x12
#define HX8357B_NORON   0x13

#define HX8357_INVOFF  0x20
#define HX8357_INVON   0x21
#define HX8357_DISPOFF 0x28
#define HX8357_DISPON  0x29

#define HX8357_CASET   0x2A
#define HX8357_PASET   0x2B
#define HX8357_RAMWR   0x2C
#define HX8357_RAMRD   0x2E

#define HX8357B_PTLAR   0x30
#define HX8357_TEON  0x35
#define HX8357_TEARLINE  0x44
#define HX8357_MADCTL  0x36
#define HX8357_COLMOD  0x3A

#define HX8357_SETOSC 0xB0
#define HX8357_SETPWR1 0xB1
#define HX8357B_SETDISPLAY 0xB2
#define HX8357_SETRGB 0xB3
#define HX8357D_SETCOM  0xB6
```

```
#define HX8357B_SETDISPMODE  0xB4
#define HX8357D_SETCYC  0xB4
#define HX8357B_SETOTP 0xB7
#define HX8357D_SETC 0xB9

#define HX8357B_SET_PANEL_DRIVING 0xC0
#define HX8357D_SETSTBA 0xC0
#define HX8357B_SETDGC  0xC1
#define HX8357B_SETID  0xC3
#define HX8357B_SETDDB  0xC4
#define HX8357B_SETDISPLAYFRAME 0xC5
#define HX8357B_GAMMASET 0xC8
#define HX8357B_SETCABC  0xC9
#define HX8357_SETPANEL  0xCC


#define HX8357B_SETPOWER 0xD0
#define HX8357B_SETVCOM 0xD1
#define HX8357B_SETPWRNORMAL 0xD2

#define HX8357B_RDID1   0xDA
#define HX8357B_RDID2   0xDB
#define HX8357B_RDID3   0xDC
#define HX8357B_RDID4   0xDD

#define HX8357D_SETGAMMA 0xE0

#define HX8357B_SETGAMMA 0xC8
#define HX8357B_SETPANELRELATED  0xE9

#define HX8357B_MADCTL_MY  0x80
#define HX8357B_MADCTL_MX  0x40
#define HX8357B_MADCTL_MV  0x20
#define HX8357B_MADCTL_ML  0x10
#define HX8357B_MADCTL_RGB 0x00
#define HX8357B_MADCTL_BGR 0x08
#define HX8357B_MADCTL_MH  0x04
```

```
/*
    author: Tyler Zoucha
    version: v3.3
  Adapted from: Matthew Boeding, lab/class TA after being adapted from Subharthi
Banerjee, Ph.D.

    README

/// The sole reason to provide this code is to make your TFTLCD (ILI9341)
/// up and running

/// Note: Most of the code is input one place. This is not ideal and I plan to ch
ange
///   it input the future

/// Use C or inline assembly program as you please.

/// ** the code uses P0 for 8-bit interface
/// ** IOM --> P3^5
/// ** CD --> P3^4
////  I recommend leaving these definitions for UART implementation later.
////
/// ** RD --> P3^3
/// ** WR --> P3^2

/// Refer to the header file to change decoding addresses for your specific desig
n.

/// Please do not post any of the code from this course to GITHUB.

*/

///  ************** IMPORTANT ********************

// It may need redfinition of pins like
// #include <reg51.h> // change microcontroller header when using AT89C55WD

#include "ecen4330lcdh.h"
#include "font.h"

// keypad configuration
unsigned char keypad[4][4] = {{'1', '4', '7', 'E'},
                              {'2', '5', '8', '0'},
```

```
                            {'3', '6', '9', 'F'},
                            {'A', 'B', 'C', 'D'}};
unsigned char colloc, rowloc;
// store it input a variable the lcd address
__xdata unsigned char *lcd_address = (unsigned char __xdata *)__LCD_ADDRESS__;
__xdata unsigned char *seg7_address = (unsigned char __xdata *)__SEG_7_ADDRESS__;

__xdata unsigned char *read_ram_address;

unsigned char selection;

#define write8inline(d)   \
    {                     \
        IOM = 1;          \
        *lcd_address = d; \
        IOM = 0;          \
    }

#define write8 write8inline
// data write
#define write8DataInline(d) \
    {                       \
        CD = 1;             \
        write8(d);          \
    }
// command or register write
#define write8RegInline(d) \
    {                      \
        CD = 0;            \
        write8(d);         \
    }

// inline definitions
#define write8Reg write8RegInline
#define write8Data write8DataInline

u16 cursor_x, cursor_y; /// cursor_y and cursor_x globals
u8 textsize, rotation;  /// textsize and rotation
u16
    textcolor,   ////< 16-bit background color for print()
    textbgcolor; ////< 16-bit text color for print()
u16
    _width,  ////< Display width as modified by current rotation
    _height; ////< Display height as modified by current rotation
```

```
inline void iowrite8(unsigned char __xdata *map_address, unsigned char d)
{
    IOM = 1;
    *map_address = d;
    IOM = 0;
}

inline unsigned char ioread8(unsigned char __xdata *map_address)
{
    unsigned char d = 0;
    IOM = 1;
    d = *map_address;
    IOM = 0;
    return d;
}

inline void ramWrite8(unsigned char __xdata *map_address, unsigned char d)
{
    IOM = 0;
    *map_address = d;
    IOM = 1;
}

inline unsigned char ramRead8(unsigned char __xdata *map_address)
{
    unsigned char d = 0;
    IOM = 1;
    d = *map_address;
    IOM = 0;
    return d;
}
void delay(int d) /// x 1ms
{
    int i, j;
    for (i = 0; i < d; i++) /// this is For(); loop delay used to define delay va
lue input Embedded C
    {
        for (j = 0; j < 1000; j++)
            ;
    }
}

void writeRegister8(u8 a, u8 d)
```

```
{
    //IOM = 0;
    CD = __CMD__;
    write8(a);
    CD = __DATA__;
    write8(d);
    //IOM = 1;
}

void writeRegister16(u16 a, u16 d)
{
    unsigned short int hi, lo;
    hi = (a) >> 8;
    lo = (a);
    //IOM = 0;
    //  CD = 0;
    write8Reg(hi);
    write8Reg(lo);
    hi = (d) >> 8;
    lo = (d);
    CD = 1;
    write8Data(hi);
    write8Data(lo);
    //  IOM =1;
}

/// +========================RTC functions================================
void rtcInit(void)
{
    //rtcCmd(__REG_F__, __HR_24__);
    unsigned int i;
    rtcCmd(__REG_F__, __HR_24__ | __STOP__ | __RESET__); // stop and reset

    // clear the registers
    for (i = __S1_REG__; i < __REG_D__; i++)
    {
        rtcWrite(i, 0x00);
    }

    rtcCmd(__REG_F__, __HR_24__);
}

void rtcBusy(void)
{
```

```
    __xdata unsigned char *map_address = (unsigned char __xdata *)(__REG_D__);
    while ((ioread8(map_address) & 0x02))       ;
}

inline void rtcCmd(unsigned int addr, unsigned char d)
{
    __xdata unsigned char *map_address = (unsigned char __xdata *)addr;
    iowrite8(map_address, d);
}

inline void rtcWrite(unsigned int addr, unsigned char d)
{
    __xdata unsigned char *map_address = (unsigned char __xdata *)addr;
    rtcCmd(__REG_D__, 0x01);
    rtcBusy();
    iowrite8(map_address, 0x00);
    rtcCmd(__REG_D__, d);
}

inline unsigned char rtcRead(unsigned int addr)
{
    unsigned char d;
    __xdata unsigned char *map_address = (unsigned char __xdata *)addr;
    rtcCmd(__REG_D__, 0x01); // hold on
    rtcBusy();
    d = ioread8(map_address);

    d = (d & 0x0f) | 0x30;   // ascii the lower word
    rtcCmd(__REG_D__, 0x00); // hold off
    return d;
}

void rtcPrint(void)
{
    unsigned char mi1, mi10, s1, s10, h1, h10;
    unsigned char printval[9];
    printval[8] = '\0'; // end with a null character for string
    printval[2] = ':';
    printval[5] = ':';

    mi1 = 0x30;
    mi10 = 0x30;
    s1 = 0x30;
    s10 = 0x30;
```

```
    h1 = 0x30;
    h10 = 0x30; // char zero
    mi1 = rtcRead(__MI1_REG__);
    mi10 = rtcRead(__MI10_REG__);
    h1 = rtcRead(__H1_REG__);
    h10 = rtcRead(__H10_REG__);
    s1 = rtcRead(__S1_REG__);
    s10 = rtcRead(__S10_REG__);
    printval[0] = h10;
    printval[1] = h1;
    printval[3] = mi10;
    printval[4] = mi1;
    printval[6] = s10;
    printval[7] = s1;
    setCursor(30, 120);
    LCD_string_write(printval);
}

void setCursor(u16 x, u16 y)
{
    cursor_x = x;
    cursor_y = y;
}
// set text color
void setTextColor(u16 x, u16 y)
{
    textcolor = x;
    textbgcolor = y;
}

// set text size
void setTextSize(u8 s)
{
    if (s > 8)
        return;
    textsize = (s > 0) ? s : 1;
}

void setRotation(u8 flag)
{
    switch (flag)
    {
    case 0:
        flag = (ILI9341_MADCTL_MX | ILI9341_MADCTL_BGR);
```

```
        _width = TFTWIDTH;
        _height = TFTHEIGHT;
        break;
    case 1:
        flag = (ILI9341_MADCTL_MV | ILI9341_MADCTL_BGR);
        _width = TFTHEIGHT;
        _height = TFTWIDTH;
        break;
    case 2:
        flag = (ILI9341_MADCTL_MY | ILI9341_MADCTL_BGR);
        _width = TFTWIDTH;
        _height = TFTHEIGHT;
        break;
    case 3:
        flag = (ILI9341_MADCTL_MX | ILI9341_MADCTL_MY | ILI9341_MADCTL_MV | ILI93
41_MADCTL_BGR);
        _width = TFTHEIGHT;
        _height = TFTWIDTH;
        break;
    default:
        flag = (ILI9341_MADCTL_MX | ILI9341_MADCTL_BGR);
        _width = TFTWIDTH;
        _height = TFTHEIGHT;
        break;
    }
    writeRegister8(ILI9341_MEMCONTROL, flag);
}

// set address definition
void setAddress(unsigned int x1, unsigned int y1, unsigned int x2, unsigned int y
2)
{
    //IOM =0;
    write8Reg(0x2A);
    write8Data(x1 >> 8);
    write8Data(x1);
    write8Data(x2 >> 8);
    write8Data(x2);

    write8Reg(0x2B);
    write8Data(y1 >> 8);
    write8Data(y1);
    write8Data(y2 >> 8);
    write8Data(y2);
```

```
    //write8Reg(0x2C);
    //IOM =1;
}

void TFT_LCD_INIT(void)
{
    //char ID[5];
    ///int id;
    _width = TFTWIDTH;
    _height = TFTHEIGHT;

    // all low
    IOM = 0;
    //RDN = 1;
    CD = 1;

    write8Reg(0x00);
    write8Data(0x00);
    write8Data(0x00);
    write8Data(0x00);
    //IOM = 1;
    delay(200);

    //IOM = 0;

    writeRegister8(ILI9341_SOFTRESET, 0);
    delay(50);
    writeRegister8(ILI9341_DISPLAYOFF, 0);
    delay(10);

    writeRegister8(ILI9341_POWERCONTROL1, 0x23);
    writeRegister8(ILI9341_POWERCONTROL2, 0x11);
    write8Reg(ILI9341_VCOMCONTROL1);
    write8Data(0x3d);
    write8Data(0x30);
    writeRegister8(ILI9341_VCOMCONTROL2, 0xaa);
    writeRegister8(ILI9341_MEMCONTROL, ILI9341_MADCTL_MY | ILI9341_MADCTL_BGR);
    write8Reg(ILI9341_PIXELFORMAT);
    write8Data(0x55);
    write8Data(0x00);
    writeRegister16(ILI9341_FRAMECONTROL, 0x001B);

    writeRegister8(ILI9341_ENTRYMODE, 0x07);
    /* writeRegister32(ILI9341_DISPLAYFUNC, 0x0A822700);*/
```

```
    writeRegister8(ILI9341_SLEEPOUT, 0);
    delay(150);
    writeRegister8(ILI9341_DISPLAYON, 0);
    delay(500);
    setAddress(0, 0, _width - 1, _height - 1);
    ///************* Start Initial Sequence ILI9341 controller **********///

    // IOM = 1;
}
void drawPixel(u16 x3, u16 y3, u16 color1)
{

    // not using to speed up
    //if ((x3 < 0) ||(x3 >= TFTWIDTH) || (y3 < 0) || (y3 >= TFTHEIGHT))
    //{
    //  return;
    //}
    setAddress(x3, y3, x3 + 1, y3 + 1);

    //IOM = 0;

    CD = 0;
    write8(0x2C);

    CD = 1;
    write8(color1 >> 8);
    write8(color1);
    //IOM = 1;
}

// draw a circle with this function

void drawCircle(int x0, int y0, int r, u16 color)
{
    int f = 1 - r;
    int ddF_x = 1;
    int ddF_y = -2 * r;
    int x = 0;
    int y = r;

    drawPixel(x0, y0 + r, color);
    drawPixel(x0, y0 - r, color);
    drawPixel(x0 + r, y0, color);
```

```
    drawPixel(x0 - r, y0, color);

    while (x < y)
    {
        if (f >= 0)
        {
            y--;
            ddF_y += 2;
            f += ddF_y;
        }
        x++;
        ddF_x += 2;
        f += ddF_x;

        drawPixel(x0 + x, y0 + y, color);
        drawPixel(x0 - x, y0 + y, color);
        drawPixel(x0 + x, y0 - y, color);
        drawPixel(x0 - x, y0 - y, color);
        drawPixel(x0 + y, y0 + x, color);
        drawPixel(x0 - y, y0 + x, color);
        drawPixel(x0 + y, y0 - x, color);
        drawPixel(x0 - y, y0 - x, color);
    }
}

void testCircles(u8 radius, u16 color)
{

    int x, y, r2 = radius * 2, w = _width + radius, h = _height + radius;

    for (x = 0; x < w; x += r2)
    {
        for (y = 0; y < h; y += r2)
        {
            drawCircle(x, y, radius, color);
        }
    }
}
void fillRect(u16 x, u16 y, u16 w, u16 h, u16 color)
{
    if ((x >= TFTWIDTH) || (y >= TFTHEIGHT))
    {
        return;
    }
```

```
    if ((x + w - 1) >= TFTWIDTH)
    {
        w = TFTWIDTH - x;
    }

    if ((y + h - 1) >= TFTHEIGHT)
    {
        h = TFTHEIGHT - y;
    }

    setAddress(x, y, x + w - 1, y + h - 1);
    //IOM = 0;

    write8Reg(0x2C);
    //IOM = 1; IOM = 0;
    CD = 1;
    for (y = h; y > 0; y--)
    {
        for (x = w; x > 0; x--)
        {

            write8(color >> 8);
            write8(color);
        }
    }
    //IOM = 1;
}

void fillScreen(unsigned int Color)
{
    //unsigned char VH,VL;
    long len = (long)TFTWIDTH * (long)TFTHEIGHT;

    int blocks;

    unsigned char i, hi = Color >> 8,
                     lo = Color;

    blocks = (u16)(len / 64); // 64 pixels/block
    setAddress(0, 0, TFTWIDTH - 1, TFTHEIGHT - 1);

    //IOM = 0;
```

```
    write8Reg(0x2C);
    //IOM = 1; IOM = 0;
    CD = 1;
    write8(hi);
    write8(lo);

    len--;
    while (blocks--)
    {
        i = 16; // 64 pixels/block / 4 pixels/pass
        do
        {

            write8(hi);
            write8(lo);
            write8(hi);
            write8(lo);
            write8(hi);
            write8(lo);
            write8(hi);
            write8(lo);
        } while (--i);
    }
    for (i = (char)len & 63; i--;)
    {

        write8(hi);
        write8(lo);
    }

    //IOM = 1;
}
void drawChar(int x, int y, unsigned char c, u16 color, u16 bg, u8 size)
{
    if ((x >= TFTWIDTH) ||          // Clip right
        (y >= TFTHEIGHT) ||         // Clip bottom
        ((x + 6 * size - 1) < 0) || // Clip left
        ((y + 8 * size - 1) < 0))   // Clip top
    {
        return;
    }

    for (char i = 0; i < 6; i++)
    {
```

```
        u8 line;

        if (i == 5)
        {
            line = 0x0;
        }
        else
        {
            line = pgm_read_byte(font + (c * 5) + i);
        }

        for (char j = 0; j < 8; j++)
        {
            if (line & 0x1)
            {
                if (size == 1) // default size
                {
                    drawPixel(x + i, y + j, color);
                }
                else
                { // big size
                    fillRect(x + (i * size), y + (j * size), size, size, color);
                }
            }
            else if (bg != color)
            {
                if (size == 1) // default size
                {
                    drawPixel(x + i, y + j, bg);
                }
                else
                { // big size
                    fillRect(x + i * size, y + j * size, size, size, bg);
                }
            }

            line >>= 1;
        }
    }
}

void write(u8 c) //write a character at setted coordinates after setting location
 and colour
{
```

```
    if (c == '\n')
    {
        cursor_y += textsize * 8;
        cursor_x = 0;
    }
    else if (c == '\r')
    {
        // skip em
    }
    else
    {
        drawChar(cursor_x, cursor_y, c, textcolor, textbgcolor, textsize);
        cursor_x += textsize * 6;
    }
}
void LCD_string_write(char *str)
{
    int i;
    for (i = 0; str[i] != 0; i++) /* Send each char of string till the NULL */
    {
        write(str[i]); /* Call transmit data function */
    }
}

// test RAM function

void testRAM(unsigned char d)
{
    unsigned int i;
    __xdata unsigned char *ram_address;

    for (i = __START_RAM__; i < __END_RAM__; i++)
    {
        IOM = 0;
        ram_address = (unsigned char __xdata *)(i);
        *ram_address = d;
        IOM = 1;
    }
}

void freeType()
{
    unsigned char count = 0;
    unsigned char d;
```

```
    while (1)
    {

        if (count == 8)
        {
            d = '\n';
            count = 0;
            write(d);
        }
        else
        {
            d = keyDetect();
            write(d);
        }

        count++;
    }
}

unsigned char keyDetect()
{
    __KEYPAD_PORT__ = 0xF0; /*set port direction as input-output*/
    do
    {
        __KEYPAD_PORT__ = 0xF0;
        colloc = __KEYPAD_PORT__;
        colloc &= 0xF0;       /* mask port for column read only */
    } while (colloc != 0xF0); /* read status of column */

    do
    {
        do
        {
            delay(20);                          /* 20ms key debounce time */
            colloc = (__KEYPAD_PORT__ & 0xF0); /* read status of column */
        } while (colloc == 0xF0);               /* check for any key press */

        delay(1);
        colloc = (__KEYPAD_PORT__ & 0xF0);
    } while (colloc == 0xF0);

    while (1)
    {
        /* now check for rows */
```

```
        __KEYPAD_PORT__ = 0xFE; /* check for pressed key input 1st row */
        colloc = (__KEYPAD_PORT__ & 0xF0);
        if (colloc != 0xF0)
        {
            rowloc = 0;
            break;
        }

        __KEYPAD_PORT__ = 0xFD; /* check for pressed key input 2nd row */
        colloc = (__KEYPAD_PORT__ & 0xF0);
        if (colloc != 0xF0)
        {
            rowloc = 1;
            break;
        }

        __KEYPAD_PORT__ = 0xFB; /* check for pressed key input 3rd row */
        colloc = (__KEYPAD_PORT__ & 0xF0);
        if (colloc != 0xF0)
        {
            rowloc = 2;
            break;
        }

        __KEYPAD_PORT__ = 0xF7; /* check for pressed key input 4th row */
        colloc = (__KEYPAD_PORT__ & 0xF0);
        if (colloc != 0xF0)
        {
            rowloc = 3;
            break;
        }
    }

    if (colloc == 0xE0)
    {
        return (keypad[rowloc][0]);
    }
    else if (colloc == 0xD0)
    {
        return (keypad[rowloc][1]);
    }
    else if (colloc == 0xB0)
    {
        return (keypad[rowloc][2]);
```

```
    }
    else
    {
        return (keypad[rowloc][3]);
    }
}

unsigned int reverse(unsigned char d)
{
    unsigned int rev = 0;
    unsigned int val = 0;
    while (d >= 1)
    {

        val = d % 10;
        d = d / 10;
        rev = rev * 10 + val;
    }
    return rev;
}

unsigned int reverse16(unsigned int d)
{
    unsigned int rev = 0;
    unsigned int val = 0;
    while (d >= 1)
    {

        val = d % 10;
        d = d / 10;
        rev = rev * 10 + val;
    }
    return rev;
}

unsigned int charToASCII(unsigned char key) {
    if(key == 0x0) return '0';
    if(key == 0x1) return '1';
    if(key == 0x2) return '2';
    if(key == 0x3) return '3';
    if(key == 0x4) return '4';
    if(key == 0x5) return '5';
    if(key == 0x6) return '6';
    if(key == 0x7) return '7';
```

```
    if(key == 0x8) return '8';
    if(key == 0x9) return '9';
    if(key == 0xa) return 'A';
    if(key == 0xb) return 'B';
    if(key == 0xc) return 'C';
    if(key == 0xd) return 'D';
    if(key == 0xe) return 'E';
    if(key == 0xf) return 'F';
    else
        return 0xff;
}

unsigned int charToInt(unsigned char key) {
    if(key == 0x0) return 0;
    if(key == 0x1) return 1;
    if(key == 0x2) return 2;
    if(key == 0x3) return 3;
    if(key == 0x4) return 4;
    if(key == 0x5) return 5;
    if(key == 0x6) return 6;
    if(key == 0x7) return 7;
    if(key == 0x8) return 8;
    if(key == 0x9) return 9;
    if(key == 0xa) return A;
    if(key == 0xb) return B;
    else
        return 0xff;
}

void asciiToDec(unsigned char d)
{
    unsigned char val;
    unsigned int id;
    id = reverse(d);
    while (id >= 1)
    {

        val = id % 10;
        id = id / 10;
        write(val + '0');
    }
    write('\n');
}
```

```
void asciiToHex(unsigned char d)
{
    unsigned char val;
    unsigned char store[2];
    unsigned char i = 0;
    store[0] = 0;
    store[1] = 0;
    while (d >= 1)
    {

        val = d % 16;
        d = d / 16;
        if (val <= 9)
        {

            store[i] = val + '0';
        }
        else
        {
            store[i] = (val % 10) + 'A';
        }
        i++;
    }
    write(store[1]);
    write(store[0]);
    //write('\n');
}

void print4Hex(unsigned char num) {
    write((u8) charToASCII(num));
}

void print8Hex(unsigned char num) {
    print4Hex(num >> 4);
    print4Hex(num & 0x0F);
}

void print16Hex(unsigned int num) {
    print8Hex((unsigned char)(num >> 8));
    print8Hex((unsigned char)num);
}

void print16Dec(unsigned int num) {
    unsigned int val;
```

```
    unsigned int id;
    id = reverse16(num);
    while (id >= 1) {
        val = id % 10;
        id = id/10;
        write(val + '0');
    }
}

void print8ASCII(unsigned char num) {
    write((u8)num);
}

void print16ASCII(unsigned int num) {
    print8ASCII((unsigned char)(num >> 8));
    print8ASCII((unsigned char)num);
}

// LCD Power On Self-Test and Welcome message
void writeSomeLines()
{
    setRotation(0);     //rotation 0 is for flat/flush LCD
    //setRotation(2);       //rotation 2 is for tiled outward LCD
    fillScreen(GREEN);
    delay(20);
    fillScreen(BLACK);
    setTextSize(5);
    setTextColor(CYAN, BLACK);
    LCD_string_write("Welcome\n");
    setTextSize(3);
    LCD_string_write("ECEN-4330\n");
    LCD_string_write("\nTyler Zoucha\n");

    delay(125);
}

// Main Menu
void printMenu()
{
    fillScreen(colorBackground);
    setTextSize(5);
    setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[Menu]\n");
```

```
    setTextSize(2);
    setCursor(0, 60);
    LCD_string_write(" D: DUMP\n");
    setCursor(120, 60);
    LCD_string_write(" B: MOVE\n");
    setCursor(0, 120);
    LCD_string_write(" E: EDIT\n");
    setCursor(120, 120);
    LCD_string_write(" F: FIND\n");
    setCursor(0, 180);
    LCD_string_write(" C: COUNT\n");
    setCursor(0, 240);
    LCD_string_write(" A: MEM CHECK\n");
    setTextColor(colorSelect, colorBackground);
    setCursor(0, 300);
    //LCD_string_write("TEST\n");
}

void dump()
{
    unsigned char __xdata* d;
    __idata unsigned int startAdd = 0;
    __idata unsigned int blockSize = 0;
    __idata unsigned int blockType = 0;
    __idata unsigned int page = 0;
    __idata unsigned char input;
    __idata unsigned char exit = 1;
    __idata unsigned char invalidType = 1;

    //Dump Menu
    fillScreen(colorBackground);
    setTextSize(5);
    setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[DUMP]\n");
    setTextSize(2);
    setCursor(0, 60);
    LCD_string_write(" Enter Address:");
    setCursor(10, 90);
    LCD_string_write("____");
    setCursor(0, 150);
    LCD_string_write(" Input Block Type");
    setCursor(10, 180);
    setTextSize(1.7);
```

```
    LCD_string_write(" 1=BYTE, 2=WORD, 4=DWORD");
    setTextSize(2);
    setCursor(0, 220);
    LCD_string_write(" Input Size:");
    setCursor(10, 250);
    LCD_string_write("____");

    // Prompt Address input
    setTextColor(colorSelect, colorBackground);
    setCursor(0, 60);
    LCD_string_write(" Enter Address:");
    setCursor(10, 90);

    // Input user address and shifts appropriately
    input = keyDetect();
    write(input);
    startAdd += input * 16 * 16 * 16;
    input = keyDetect();
    write(input);
    startAdd += input * 16 * 16;
    input = keyDetect();
    write(input);
    startAdd += input * 16;
    input = keyDetect();
    write(input);
    startAdd += input;

    // Remove selection color
    setTextColor(colorText, colorBackground);
    setCursor(0, 60);
    LCD_string_write(" Enter Address:");

    // Prompt block type
    setTextColor(colorSelect, colorBackground);
    setCursor(0, 150);
    LCD_string_write(" Input Block Type");

    // Sanitize input until correct block type chosen
    while(invalidType) {
        input = keyDetect();

        // Show selection
        if (input == '1')
        {
```

```
            invalidType = 0;
            blockType = 1;
            setTextSize(1.7);
            setCursor(10, 180);
            setTextColor(colorSelect, colorBackground);
            LCD_string_write(" 1=BYTE");
            setTextColor(colorText, colorBackground);
            LCD_string_write(" 2=WORD 4=DWORD");
            delay(40);
            setTextSize(2);
        }
        if (input == '2')
        {
            invalidType = 0;
            blockType = 2;
            setTextSize(1.7);
            setCursor(10, 180);
            setTextColor(colorText, colorBackground);
            LCD_string_write(" 1=BYTE ");
            setTextColor(colorSelect, colorBackground);
            LCD_string_write("2=WORD");
            setTextColor(colorText, colorBackground);
            LCD_string_write(" 4=DWORD");
            delay(40);
            setTextSize(2);
        }
        if (input == '4')
        {
            invalidType = 0;
            blockType == 4;
            setTextSize(1.7);
            setCursor(10, 180);
            setTextColor(colorText, colorBackground);
            LCD_string_write(" 1=BYTE 2=WORD ");
            setTextColor(colorSelect, colorBackground);
            LCD_string_write("4=DWORD");
            delay(40);
            setTextSize(2);
        }
    }

    // Remove selection color
    setTextColor(colorText, colorBackground);
    setCursor(0, 150);
```

```
    LCD_string_write(" Input Block Type");

    // Prompt Block Size
    setCursor(0, 220);
    setTextColor(colorSelect, colorBackground);
    LCD_string_write(" Input Size");
    setCursor(10, 250);

    // Input user block size and shift appropriately.
    input = keyDetect();
    write(input);
    blockSize += input * 16 * 16 * 16;
    input = keyDetect();
    write(input);
    blockSize += input * 16 * 16;
    input = keyDetect();
    write(input);
    blockSize += input * 16;
    input = keyDetect();
    write(input);
    blockSize += input;

    LCD_string_write("\n");
    delay(25);

    fillScreen(colorBackground);
    setTextSize(5);
    setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[DUMP]\n");

    while (exit) {
        setTextSize(2);
        setTextColor(colorText, colorBackground);
        setCursor(0, 300);
        LCD_string_write(" Page:");
        setCursor(120, 300);
        print16Hex(page);
        setCursor(0, 60);
        LCD_string_write(" Address:");
        setCursor(10, 90);
        print16Hex(startAdd + page * blockType);
        d = (unsigned char __xdata *)(startAdd + page * blockType);
        if (blockType == 1)
```

```
            {
                setCursor(0, 120);
                LCD_string_write(" Hex Data:");
                setCursor(10, 150);
                print8Hex(ramRead8(d));

                setCursor(0, 180);
                LCD_string_write(" ASCII Data:");
                setCursor(10, 210);
                print8ASCII(ramRead8(d));
            }
        if (blockType == 2)
            {
                setCursor(0, 120);
                LCD_string_write(" Hex Data:");
                setCursor(10, 150);
                print8Hex(ramRead8(d));
                d++;
                print8Hex(ramRead8(d));
                d--;

                setCursor(0, 180);
                LCD_string_write(" ASCII Data:");
                setCursor(10, 220);
                print8ASCII(ramRead8(d));
                d++;
                print8ASCII(ramRead8(d));
            }
        if (blockType == 4)
            {
                setCursor(0, 120);
                LCD_string_write(" Hex Data:");
                setCursor(10, 150);
                print8Hex(ramRead8(d));
                d++;
                print8Hex(ramRead8(d));
                d++;
                print8Hex(ramRead8(d));
                d++;
                print8Hex(ramRead8(d));
                d--;
                d--;
                d--;
```

```
            setCursor(0, 180);
            LCD_string_write(" ASCII Data:");
            setCursor(10, 220);
            print8ASCII(ramRead8(d));
            d++;
            print8ASCII(ramRead8(d));
            d++;
            print8ASCII(ramRead8(d));
            d++;
            print8ASCII(ramRead8(d));
        }
    setCursor(0, 250);
    setTextSize(1.7);
    LCD_string_write(" 0=Next 1=Prev 2=Exit");
    setTextSize(2);

    input = keyDetect();

    if (input == '0')
        {
            if (page < blockSize - 1)
                page++;
            setCursor(0, 250);
            setTextColor(colorSelect, colorBackground);
            LCD_string_write(" 0=Next");
            setTextColor(colorText, colorBackground);
            LCD_string_write(" 1=Prev 2=Exit");
            delay(40);
        }
    if (input == '1')
        {
            if (page > 0)
                page--;
            setCursor(0, 250);
            setTextColor(colorText, colorBackground);
            LCD_string_write(" 0=Next ");
            setTextColor(colorSelect, colorBackground);
            LCD_string_write("1=Prev");
            setTextColor(colorText, colorBackground);
            LCD_string_write(" 2=Exit");
            delay(40);
        }
    if (input == '2')
        {
```

```
                exit = 0;
                setCursor(0, 250);
                setTextColor(colorText, colorBackground);
                LCD_string_write(" 0=Next 1=Prev ");
                setTextColor(colorSelect, colorBackground);
                LCD_string_write("2=Exit");
                delay(40);
            }
    }
}

void move() {
    unsigned char __xdata* d;
    __idata unsigned int sourceAdd= 0;
    __idata unsigned int destAdd = 0;
    __idata unsigned int blockSize = 0;
    __idata unsigned char blockType = 0;
    __idata unsigned char input;

    fillScreen(colorBackground);
    setTextSize(4);
    setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[MOVE]\n");

    setTextSize(2);
    setCursor(0, 60);
    LCD_string_write(" Input Source Address:");
    setCursor(10, 90);LCD_string_write("____");
    setCursor(0, 120);
    LCD_string_write(" Input Dest Address:");
    setCursor(10, 150);
    LCD_string_write("____");setCursor(0, 120);
    LCD_string_write(" Input Block Type");
    setCursor(0, 180);
    LCD_string_write(" 1=BYTE 2=WORD 4=DWORD");
    setCursor(0, 210);
    LCD_string_write(" Input Block Size:");
    setCursor(10, 240);
    LCD_string_write("____");
    setTextColor(colorSelect,colorBackground);
    setCursor(0, 270);
    LCD_string_write(" Input Source Address:");
    setCursor(10, 300);
```

```
    input = keyDetect();
    print4Hex(input);
    sourceAdd += input * 16 * 16 * 16;


    // Input user block size and shift appropriately.
    input = keyDetect();
    print4Hex(input);
    sourceAdd += input * 16 * 16;
    input = keyDetect();
    print4Hex(input);
    sourceAdd += input * 16;
    input = keyDetect();
    print4Hex(input);
    sourceAdd += input ;
    setTextColor(colorText,colorBackground);
    setCursor(0, 60);
    LCD_string_write(" Input Source Address:");
    setTextColor(colorSelect,colorBackground);
    setCursor(0, 90);
    LCD_string_write(" Input Dest Address:");
    setCursor(260, 90);

    // Input user block size and shift appropriately.
    input = keyDetect();
    print4Hex(input);
    destAdd += input * 16 * 16 * 16;
    input = keyDetect();
    print4Hex(input);
    destAdd += input * 16 * 16;
    input = keyDetect();
    print4Hex(input);
    destAdd += input * 16;
    input = keyDetect();
    print4Hex(input);
    destAdd += input ;
    setTextColor(colorText,colorBackground);
    setCursor(0, 90);
    LCD_string_write(" Input Dest Address:");
    setTextColor(colorSelect,colorBackground);

    setCursor(0, 120);
    LCD_string_write(" Input Block Type");
```

```
    while(blockType != 1 && blockType != 2 && blockType != 4) {
        blockType = keyDetect();
        if(blockType == 1){
            setCursor(0, 150);
            setTextColor(colorSelect, colorBackground);
            LCD_string_write(" 1=BYTE ");
            setTextColor(colorText, colorBackground);
            LCD_string_write("2=WORD 4=DWORD");
            delay(20);
            } if(blockType == 2) {
                setCursor(0, 150);
                setTextColor(colorText, colorBackground);
                LCD_string_write(" 1=BYTE ");
                setTextColor(colorSelect, colorBackground);
                LCD_string_write("2=WORD ");
                setTextColor(colorText, colorBackground);
                LCD_string_write("4=DWORD");
                delay(20);
                } if(blockType == 4) {
                    setCursor(0, 150);
                    setTextColor(colorText, colorBackground);
                    LCD_string_write(" 1=BYTE 2=WORD ");
                    setTextColor(colorSelect, colorBackground);
                    LCD_string_write("4=DWORD");
                    delay(20);
                }
    }

    setTextColor(colorText,colorBackground);
    setCursor(0, 120);
    LCD_string_write(" Input Block Type");
    setTextColor(colorSelect,colorBackground);
    setCursor(0, 180);
    LCD_string_write(" Input Block Size:");
    setCursor(260, 180);
    input = keyDetect();
    print4Hex(input);
    blockSize += input * 16 * 16 * 16; //put input 4th hexidecimal place
    input = keyDetect();print4Hex(input);blockSize += input * 16 * 16; //put inpu
t 3rd hexidecimal place
    input = keyDetect();print4Hex(input);blockSize += input * 16; //put input 2nd
 hexidecimal place
    input = keyDetect();print4Hex(input);blockSize += input ; //put input 1st hex
idecimal place
```

```
    for(unsigned int i = sourceAdd;i< (sourceAdd + blockSize * blockType);i++) {
        d = (unsigned char __xdata*)destAdd;
        ramWrite8(d,ramRead8((unsigned char __xdata*)i));
        destAdd++;
    }

    setTextColor(colorText,colorBackground);
    setCursor(0, 210);
    LCD_string_write("Done!!!");
    delay(60);
}

void edit(){
    unsigned char __xdata* d;
    __idata unsigned int address = 0;
    __idata unsigned char input;
    __idata unsigned char value = 0;
    __idata unsigned char exit =1;
    fillScreen(colorBackground);
    setTextSize(4);
    setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[EDIT]\n");
    setTextSize(2);setCursor(0, 120);
    LCD_string_write(" Input Address:");
    setCursor(260, 120);
    LCD_string_write("____");
    setTextColor(colorSelect, colorBackground);
    setCursor(260, 120);
    input = keyDetect();
    print4Hex(input);
    address += input * 16 * 16 * 16; //put input 4th hexidecimal place
    input = keyDetect();
    print4Hex(input);
    address += input * 16 * 16; //put input 3rd hexidecimal place
    input = keyDetect();
    print4Hex(input);
    address += input * 16; //put input 2nd hexidecimal place
    input = keyDetect();
    print4Hex(input);
    address += input ; //put input 1st hexidecimal place

    fillScreen(colorBackground);
```

```
    setTextSize(4);
    setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[EDIT]\n");
    while(exit){
        d = (unsigned char __xdata*)address;
        setTextSize(2);
        setCursor(0, 60);
        LCD_string_write("Address:");
        setCursor(260, 60);
        print16Hex(address);
        setCursor(0, 90);
        LCD_string_write("Content:");
        setCursor(260, 90);
        print8Hex(ramRead8(d));
        setCursor(0, 120);
        LCD_string_write("New Value:");
        setCursor(260, 120);
        LCD_string_write("__");
        setCursor(0, 150);
        LCD_string_write("Choose Next Action");
        setCursor(0, 180);
        LCD_string_write(" 0=NEXT 1=EXIT");
        setTextColor(colorSelect,colorBackground);
        setCursor(0, 120);
        LCD_string_write("New Value:");
        setCursor(260, 120);
        input = keyDetect();
        print4Hex(input);
        value += input * 16; //put input 2nd hexidecimal place
        input = keyDetect();
        print4Hex(input);
        value += input ; //put input 1st hexidecimal place
        ramWrite8(d,value);//write new value to memory

        value = 0;
        setTextColor(colorText,colorBackground);
        setCursor(0, 120);
        LCD_string_write("New Value:");
        setTextColor(colorSelect,colorBackground);
        setCursor(0, 150);
        LCD_string_write("Choose Next Action");
        input = keyDetect();
        if(input==0) {
```

```
            address++;
            setTextColor(colorSelect,colorBackground);
            setCursor(0, 180);
            LCD_string_write(" 0=NEXT ");
            setTextColor(colorText,colorBackground);
            LCD_string_write("1=EXIT");
            delay(20);
        } if (input==1) {
            exit=0;
            setTextColor(colorText,colorBackground);
            setCursor(0, 180);
            LCD_string_write(" 0=NEXT ");
            setTextColor(colorSelect,colorBackground);
            LCD_string_write("1=EXIT");
            delay(20);
        }

        setTextColor(colorText,colorBackground);
        setCursor(0, 150);
        LCD_string_write("Choose Next Action");
        setCursor(0, 180);
        LCD_string_write(" 0=NEXT 1=EXIT");
    }
}

void find() {
    unsigned char __xdata* d;
    __idata unsigned int startAdd = 0;
    __idata unsigned int blockSize = 0;
    __idata unsigned char input;
    __idata unsigned char value = 0;
    __idata unsigned char noneFound = 1;
    fillScreen(colorBackground);
    setTextSize(4);setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[FIND]\n");
    setTextSize(2);
    setCursor(0, 60);
    LCD_string_write(" Input Value:");
    setCursor(10, 90);
    LCD_string_write("__");
    setCursor(0, 120);
    LCD_string_write(" Input Start Address:");
    setCursor(10, 150 );
```

```
    LCD_string_write("____");
    setCursor(0, 180);
    LCD_string_write(" Input Block Size:");
    setCursor(10, 210);
    LCD_string_write("____");
    setTextColor(colorSelect, colorBackground);
    setCursor(0, 60);
    LCD_string_write(" Input Value:");
    setCursor(260, 60);

    input = keyDetect();
    print4Hex(input);
    value += input * 16; //put input 2nd hexidecimal place
    input = keyDetect();
    print4Hex(input);
    value += input ; //put input 1st hexidecimal place
    setTextColor(colorText, colorBackground);
    setCursor(0, 60);
    LCD_string_write(" Input Value:");
    setTextColor(colorSelect, colorBackground);
    setCursor(0, 90);
    LCD_string_write(" Input Start Address:");
    setCursor(260, 90);
    input = keyDetect();
    print4Hex(input);
    startAdd += input * 16 * 16 * 16; //put input 4th hexidecimal place
    input = keyDetect();
    print4Hex(input);
    startAdd += input * 16 * 16; //put input 3rd hexidecimal place
    input = keyDetect();
    print4Hex(input);
    startAdd += input * 16; //put input 2nd hexidecimal place
    input = keyDetect();
    print4Hex(input);
    startAdd += input ; //put input 1st hexidecimal place
    setTextColor(colorText, colorBackground);
    setCursor(0, 90);
    LCD_string_write(" Input Start Address:");

    setTextColor(colorSelect, colorBackground);
    setCursor(0, 120);
    LCD_string_write(" Input Block Size:");
    setCursor(260, 120);
    input = keyDetect();
```

```
    print4Hex(input);
    blockSize += input * 16 * 16 * 16; //put input 4th hexidecimal place
    input = keyDetect();print4Hex(input);
    blockSize += input * 16 * 16; //put input 3rd hexidecimal place
    input = keyDetect();print4Hex(input);
    blockSize += input * 16; //put input 2nd hexidecimal place
    input = keyDetect();print4Hex(input);
    blockSize += input ; //put input 1st hexidecimal place
    fillScreen(colorBackground);
    setTextSize(4);
    setTextColor(colorText, colorBackground);
    setCursor(30, 0);
    LCD_string_write("[FIND]\n");
    setTextSize(2);
    setCursor(0, 70);
    LCD_string_write(" _______________");
    setCursor(0, 60);
    LCD_string_write(" Value Found At");

    for(unsigned int i = 0; i < blockSize; i++){
        d = (unsigned char __xdata*)(i+startAdd);
        if(value == ramRead8(d)){
            setCursor(60, 120);
            print16Hex(i+startAdd);
            noneFound = 0;
            setTextColor(colorText, colorBackground);
            setCursor(0, 180);
            LCD_string_write(" 0=Next 1=Exit");
            input = keyDetect();
            if(input == 0){
                setTextColor(colorSelect, colorBackground);
                setCursor(0, 180);
                LCD_string_write(" 0=Next ");
                setTextColor(colorText, colorBackground);
                LCD_string_write("1=Exit");
                delay(20);
            } if(input == 1) {
                setTextColor(colorText, colorBackground);
                setCursor(0, 180);
                LCD_string_write(" 0=Next ");
                setTextColor(colorSelect, colorBackground);
                LCD_string_write("1=Exit");
                delay(20);
                break;
```

```
            }
            delay(50);
        }
    } if(noneFound) {
            setTextColor(colorText, colorBackground);
            setCursor(0, 180);
            LCD_string_write(" Found None.\n Exiting...");
            delay(55);
        }
}

void main(void) {
    CD = 0;
    IOM = 0;

    iowrite8(seg7_address, 0xC0);       // 0 shows up 7-segment
    IOM = 0;
    CD = 1;

    TFT_LCD_INIT();
    iowrite8(seg7_address, 0xF9);       // 1 shows up on 7-segment
    writeSomeLines();
    fillScreen(GRAY);
    setTextColor(BLACK, GRAY);
    setCursor(30, 120);
    rtcInit();
    LCD_string_write("RAM POST\n");
    setCursor(30, 150);
    testRAM(0xAA);
    setCursor(30, 150);
    LCD_string_write("...\n");
    setCursor(0,0);
    for (unsigned int i = __START_RAM__; i<__END_RAM__; i++) {
        IOM = 0;

        if(0xAA != *(unsigned char __xdata*)(i)) {
            iowrite8(seg7_address, 0x8E);   // Write F to 7-
segment; RAM test fail
            setCursor(0,0);
            LCD_string_write("ERROR FOUND At: ");
            print16Hex(i);
            delay(50);
        }
        iowrite8(seg7_address, 0x88);       // Write A to 7-segment
```

```
        IOM = 1;
    }
    while(1) {
        fillScreen(WHITE);
        rtcPrint();
        delay(20);
        delay(20);
        printMenu();
        selection = keyDetect();

        if (selection == 'D') {
            setCursor(0,60);
            LCD_string_write(" D: DUMP\n");
            delay(40);
            dump();
        } if(selection == 'B') {
            setCursor(170, 60);
            LCD_string_write(" B: MOVE\n");
            delay(40);
            move();
        } if(selection == 'E') {
            setCursor(0, 120);
            LCD_string_write(" E: EDIT\n");
            delay(40);
            edit();
        } if(selection == 'F') {
            setCursor(170, 120);
            LCD_string_write(" F: FIND\n");
            delay(40);
            find();
        } if(selection == 'A') {
            setCursor(0, 180);
            LCD_string_write(" A: COUNT\n");
            delay(40);
        }
//      freeType();
    }
}
```