Lab 4:
SRAM, LCD, Keypad and The Mortgage Calculator

Bruce Oshokoya
Jonathan Peevy
Natalie Morningstar

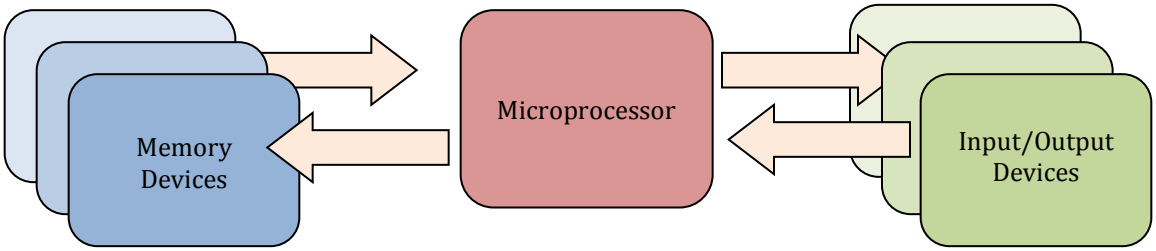
CMPE 310: System Design & Programming
University of Maryland, Baltimore County
December 17, 2013

Introduction

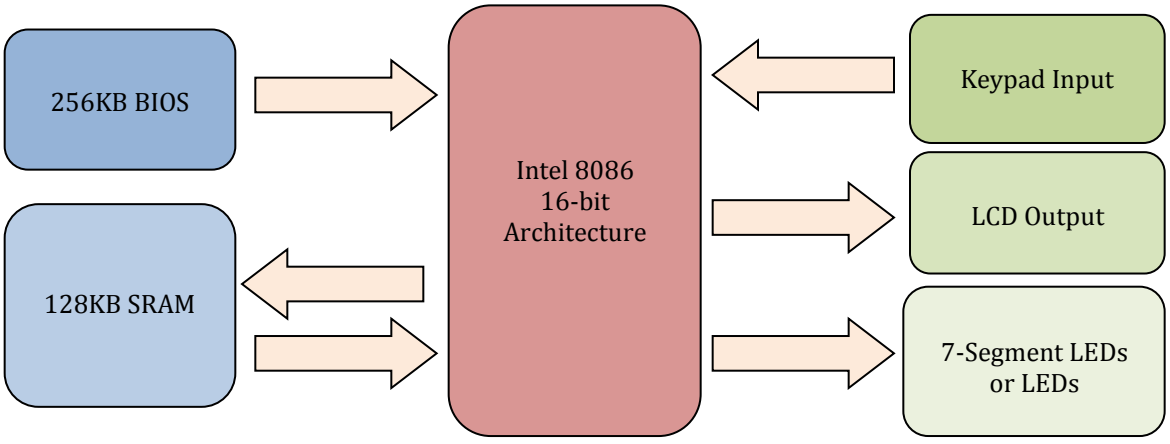
This lab consists of a Mortgage Calculator software application meant to run on the UMBC Trainer System board architecture and a full hardware implementation exclusive from the UMBC Trainer System. The exclusive hardware implementation is referred to as the 8086 Based System. Schematics for both the UMBC Trainer System and 8086 Based System are provided in this report so that the client may reference them throughout the report.

- **Background**

The 8086 Based System completed in this lab is one good example of a microprocessor-based computer system. A microprocessor-based computer system is one that includes a microprocessor, memory device and an input/output system. Likewise the Mortgage Calculator programmed on the UMBC Trainer System demonstrates the primary functions of a microprocessor: transfer of data between memory devices and input/output systems, arithmetic and logic operations and program flow. The figure below models a general microprocessor-based computer system.



The microprocessor for this computer system is the Intel 8086. In this computer system 256KB of flash memory are used for the BIOS (Basic Input/Output System) to store setup information for the system. On the 8086 Based System EEPROM is used and on the UMBC Trainer System CMOS is used. In addition, for both the UMBC Trainer System and 8086 Based System 128KB of SRAM (Static-RAM) is used to temporarily hold data while the microprocessor performs arithmetic and logic operations like those needed in our Mortgage Calculator. Both systems use a keypad as an input, but they differ slightly, one a 5X5 matrix and the other a 4X3 matrix. Again, both systems utilize an LCD Display so the user can navigate the software. Error indicators notify the user through either the 7-segment display or the LEDs. Below is a simple yet slightly more explicit model of our computer system.



- **Problem Description**

Now in phase three, the final task was to interface our system with SRAM through schematics and write a Mortgage Calculator for the UMBC 8086 Trainer System. The specifications that the client requested for the 8086 Based System are as follows:

1. Create a detailed schematic of the 8086 Based System.
2. Develop a software test plan for debugging the Mortgage Calculator on the UMBC 8086 hardware.
3. Demonstrate the proper operation.

The second assignment in this lab was to design the schematics for the UMBC 8086 Trainer system. The exact specifications are as follows:

1. 8086 in minimum mode and its associated bus buffering/demultiplexing logic. NMI should be connected to a push-button switch with appropriate timing logic. The data bus, address bus and all control signals should be connected to headers after buffering and demultiplexing for external access to design expansion boards.
2. 256KB of CMOS flash composed of 128K x 8 28F010 CMOS Flash Memory, decoded into two banks, highest address FFFFFH.
3. 128KB of SRAM composed of 32K x 8 CY7C199 Static SRAM, decoded into two banks, lowest address 00000H.
4. 8284A Clock Generator along with crystal and reset sub circuits
5. 3 8255 chips, decoded at the following addresses (all in hex), all port connections should be pulled to headers for external access.
    a. PPI 1: Port addresses FFFF (control register), FFFD, FFFB, FFF9
    b. PPI 2: Port addresses FFFE (control register), FFFC, FFFA, FFF8
    c. PPI 3: Port addresses FFF7 (control register), FFF5, FFF3, FFF1
6. 1 8259 decoded at FFF6 (command) and FFF4 (data). IR0 connected to a push button switch with necessary circuits, IR1 connected to 8254 counter 2 output, IR2 connected to the 8279 IRQ output and IR3 connected to the 16550 INTR output. All other IR lines connected to headers for external access.
7. 1 8254 decoded at FFDE (command), FFDC, FFDA, FFD8, with all counter 1 and counter 3 pins connected to headers. Counter 2 output connected to IR1 of the 8259, gate and clock connected to header.
8. 1 8279 decoded at FFF2 (command) and FFF0 (data) which will be operated in the decoded mode. Connect 20 push button switches connected as four rows and five columns. Also connect two switches to the control and shift inputs. In your layout arrange these 22 keys as well as reset, switch connected to the IR0 input on the 8259 and NMI keys to form a 5x5 keyboard matrix layout. CLK input to the 8279 should be the PCLK signal from the 8284A.
9. 1 16550 UART decoded in the high bank at odd port address from FFEF to FFE1. The clock input to the UART should be PCLK from the 8284A and the INTR output should be connected to IR3 of the 8259. Make the connections for serial port using a MAX 235 line driver/ receiver and a DSUB-9 connector. Enable all receiver inputs on the MAX 235.
10. 1 20 character x 4 line LCD display with no back-light and an integrated LCD controller decoded at addresses FFD6, FFD4, FFD2 and FFD0.
11. 2 common-anode 7-seg LEDs with a decimal point segment connected to 74374 latches

decoded at FFCE and FFCF, 8 LEDs connected to a 74374 latch decoded at FFCC.
12. 8 DIP switches connected to a 74244 decoded at FFCA.
13. A power terminal block to provide power to the board. A 100uF decoupling capacitor should be connected next to the power terminal block. Place a 0.1uF decoupling capacitor next to each of the major chips in your project.
14. Other misc. components which are required for the above circuits e.g. resistor packs, caps, etc. You can use discrete gate ICs, 3-to-8 decoder, 2-to-4 decoder or 16L8 (preferable) for decoding various addresses. Include the 16L8 programs if you decided to use them for decoding. Consider using the least number of chips required to perform IO and memory decoding.

The schematics for this UMBC 8086 Trainer system are in the Appendix B.

- **Goals and Specifications**

Given the limited scope of time available and available labor, the goal of was to finish as much as possible before submitting to the client. To achieve this goal, tasks were divided between three teammates: one implementing software, one implementing hardware onto schematics and one floater who worked both on implementing software and hardware schematics. This dynamic was an efficient use of time. The software goals were divided into 5 phases: demonstration of proper functionality of one, the display, two, the integrated circuit programming, third, the interrupt handling, forth, the application and lastly, the user interface. This evolution type model was decided early to help manage debugging and help manage time. Because there were nearly 20 schematics that need to be implemented, the hardware goal was simply design early and often. This too, was an effective strategy and proper use of time.

The System Architecture Section

The top level of the system that was constructed is focused on the 8086 microprocessor. The 8086 microprocessor is the brain of the system. It controls the 74LS373 latch and the 74LS245 buffer. The 8086 may be the brain but the 28C256 EEPROM's is the memory for the 8086. These directly affected what the 8086 processor would do. In the beginning stages of this endeavor it was unknown what would be the best approach to design the embedded system. There are of total of thirteen different components that make up the system, the 74LS373 latch, the 74LS245, the 8086 microprocessor, a clock generator, the 28C256 32kx8 EEPROM, the 74LS244 buffer, the 7400 AND, the 7432 OR, the CY7C128A SRAM, the Series 96 keypad, the 1 line x 16 characters LCD, the 7 Segment Display & LEDs. The goal of the project was to interface these devices in a configuration that supported correct functioning of the processor, verified by checking signals with an oscilloscope. The problem was how to organize these IC's efficiently on a circuit board.

- **Possible Approaches**

Since the problem as described included a design for how components should be connected, the only thing left to plan was the configuration of the IC's physically on the board that would assist us in completion of the lab successfully. One possible approach was to somehow to compact the clock generator as much as possible and put it one corner of the circuit board. This seemed logical at first but it was later determined that it would be problematic trying to understand what was going on the circuit board. For starters the clock generator would be too compact to properly function. However even if it would function correctly it would be entangled with the wires from

the other part of the board. When there are too many wires on a breadboard error debugging becomes nearly impossible.

Another problem that had to be tackled was how to organize the rest of the breadboard. Since the clock generator would be located elsewhere, how would the 8086 microprocessor, the 3 latch's and 3 buffers, logic gate IC's and the EEPROM's be organized that would be the most efficient when wiring them up. If it is not organized well then it is easy to lose track of the wires and make a mistake, making debugging time-costly. Originally the three different sections of IC's were going to be lined up vertically across the circuit board so each section would be together. However it was decided that other options would be better suited to the design, due to the fact that there would be a lot of bridging as a result. When organized in a not so efficient way then there is a possibility that more long wires will be used and the more bridging that would happen. Unfortunately this problem only got worse after phase one was finished and phase two needed to be implemented it. Phase one took up so much space on the board it meant that phase two would require a lot more thought.

For phase 3 of the project the problem was not the hardware and trying to find room on the board to interface the devices but it is trying to figure out how to work with the UMBC 8086 trainer Board. In this phase of the project a restriction was given in which our board that we were working on will not be used in programming and interfacing. Instead this entire phase will be done on the 8086 Trainer Board. The trainer board has all the components already interfaced. The only problem would be to figure out how everything works when it is already done so you can make it work for what you want it to do.

- **Selected Approach**

For phase one it was decided that in order to efficiently construct the system, two circuit boards were needed. One circuit board would contain just the clock generator, while the other would contain the rest. On the non-clock generator board the IC's were grouped together by importance. The most important IC's needed to be right underneath the 8086 microprocessor. It was agreed upon that the IC that was interfaced with the most was the 74LS373's. That was placed right underneath the 8086 while the 74LS245's was placed to the right of it. Then for phase two it was decided that the 74LS244 buffer had to be in between the two EEPROM's due to needed wiring of the two components. As a result the three components were placed at the empty space right of the 74LS245's. That left room for the only components left which were the 7432 OR and 7400 AND IC. So those components were placed right underneath the 74LS245's. This arrangement gave the most room for the clock generator and minimized the amount of bridging of wires. For phase 3 the schematics were done for both the Trainer Board and the original board. Doing it this way allows for optimal understanding the interfacing the 8086 across multiple designs. The set up is shown in the in Figure 1 below.

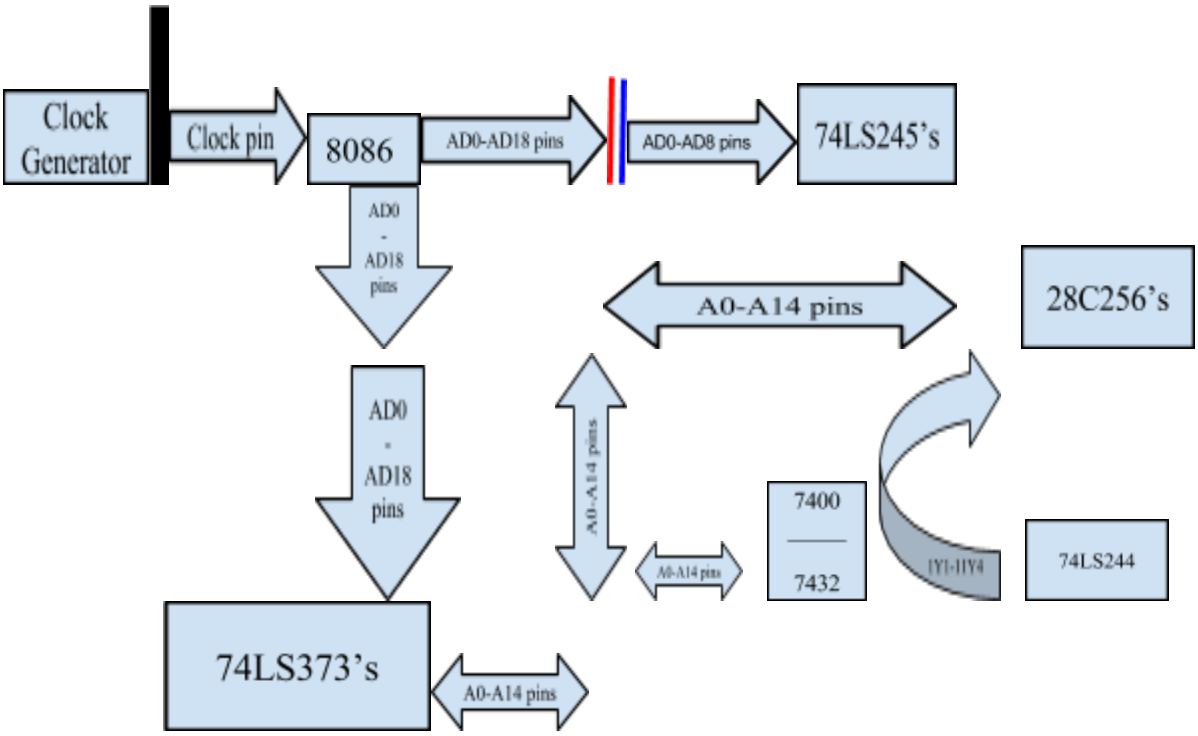


Figure 1: Phase I &II Implementation Approach

- **Test Plan**

In order to make sure the finished product runs smoothly a plan will be needed for debugging the UMBC 8086 hardware and the software.

- **Hardware**

The hardware that will be used for the finished product is the UMBC 8086 Trainer System. However it is required that a way needs to be made in which our finished product will work in the integrated system that was talked about earlier in the report. In order for this to be possible a schematic needs to be done for both the UMBC Trainer Board and the original board. The goal of the schematic will be to first wire it up there with the logic that makes sense. After both schematics are done then the next step would be to load the program and then determine if it is a software issue or is it a hardware issue. Setting the plan to test the hardware this way also determines if anything was overlooked in building the schematic. Then if it is determined that it is a hardware issue we will go back and change the schematic. It's from the schematic that we can determine how to program the code correctly.

- ***Software***

In order for the software to work correctly it has been split into 5 different versions. Splitting into 5 different versions will allow for incremental testing as well for efficient debugging.

1. **Lab4v1.asm**

Labv1.asm is the first version of the code that will be tested. It has only one function, it test displaying on the LCD. The code displays a welcome message that reads:

Welcome to CMPE 310
Fall 13
Trainer Board
8086 - Project 4

Testing the program to see if it can display to the LCD screen is important in the development of the product because it tests the boundaries of the screen that will be displayed. A restriction was given that the screen can only show 20 characters per line and a total of four lines can be shown at a time. This display is the only verification that our code works correctly. In conclusion having knowledge of the medium that will be worked with is very important.

2. **Lab4v2.asm- (Includes IC)**

This is the version is which the IC's that will be used in the project will actually be programmed. This means that all the IC's will be programmed with the appropriate command words in order for the IC's to work properly for the specific program that is needed for this project.

3. **Lab4v3.asm- (Includes interrupts)**

In this version of the code all the interrupts will be programmed so the 8086 will know when to stop what it is doing and do a requested function. This included making sure the keyboard can get send output to the LCD correctly.

4. **Lab4v4.asm- (Includes functionality)**

This is the version of the code in which we will actually be programming the Mortgage calculator. The program may or may not be set up to print to the LCD however it will be programmed to do the correct calculations.

5. **Lab4v5.asm- (Includes user interface)**

This is the final version of the program and this part definitely includes printing to the LCD in the right format restrictions. This part also includes a lot of polishing of the code to make it run faster et cetera.

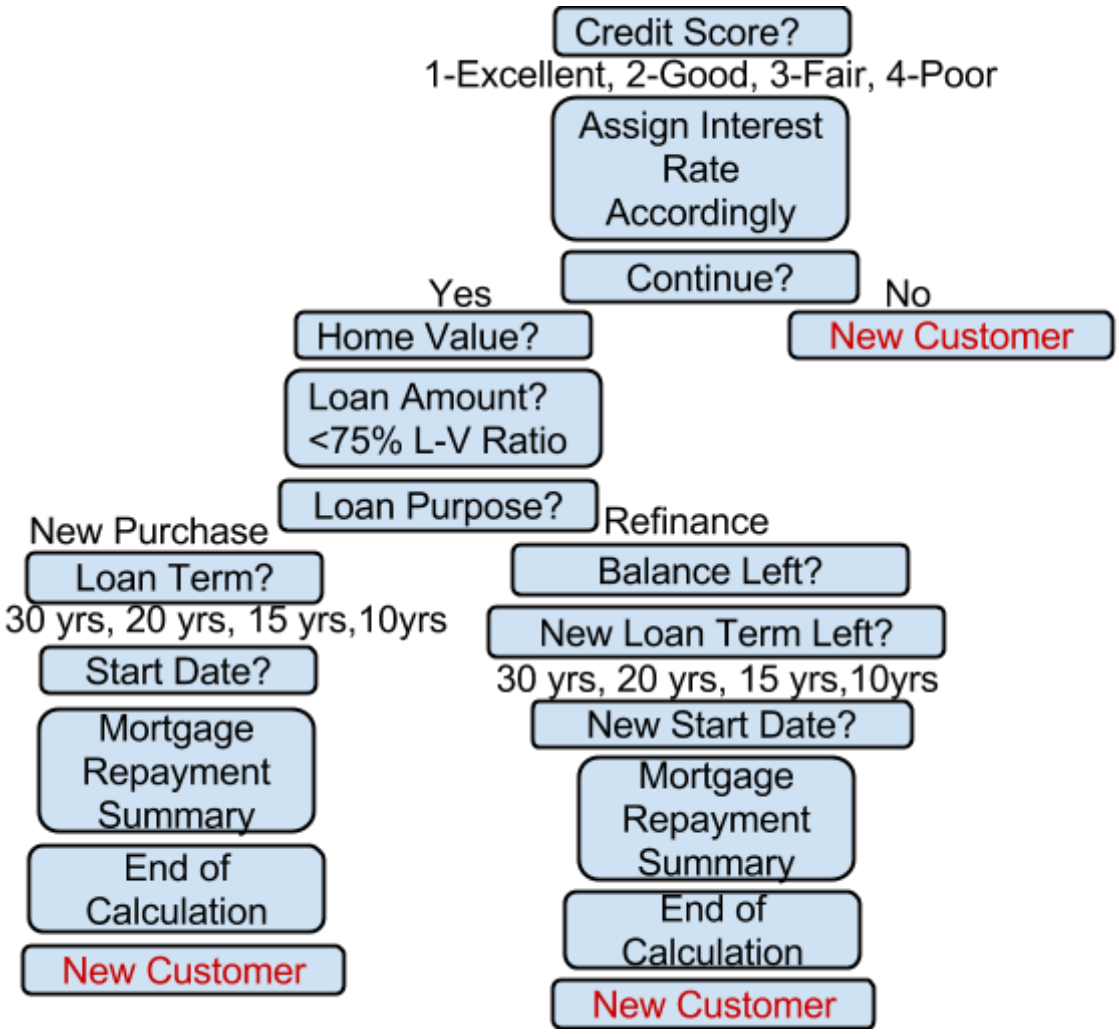


Figure 1.a: Test Flowchart of How Calculator Should Work

The Hardware System Design Section

There are eight types of IC's that this project will utilize: the 8284 clock generator, the 8086 microprocessor, the 74LS373 octal transparent latch, the 74LS245 bidirectional buffer, the 74LS244 unidirectional buffer, the 7432 OR gate, the 7400 NAND gate, and the 82C256 EEPROM. The main new component in this lab is the EEPROM, as interfacing this device is the main objective of the lab. The other new components (NAND gate, OR gate, and unidirectional buffer) are used to generate control signals for the EEPROM functionality.

- **System Clock**

The system clock of the CPU is generated by the 8284 clock generator and 4 MHz crystal. The RC circuit and switch are used to safely reset the CPU. In general, a system clock is an essential component in computing. It is best described as the heartbeat that allows all devices to be synchronous. Without a clock, it would be impossible to create synchronous signals.

The clock generator's primary functions are clock outputs, reset synchronization, ready synchronization, oscillator output and clock synchronization. Each of these primary functions has designated pins that can be seen in Figure 2 below.

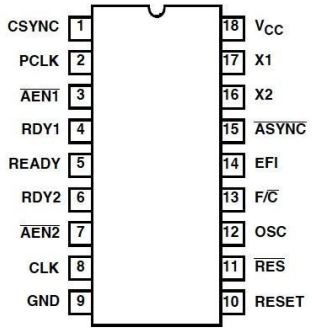


Figure 2: 82C84 Clock Generator
Pin Configuration

The clock outputs for this IC are CLK and PCLK. The CLK output drives the the CPU and in this case, the 8086 CLK input. The duty cycle of the CLK output is 33% and the duty cycle of the PCLK is 16.6%. The PCLK is a peripheral clock signal used to drive peripheral devices in the system. The clock rate at which the CPU requires to be operated will be explored further in the Experimental Results section. Proper timing constraints must be met in order to reset the CPU. This is done with the RESET synchronization, which is partially implemented externally with an RC circuit. In order for the CPU to be reset, RESET must be held for at least 50 microseconds. READY synchronization is used to select between RDY1 and RD2 or both. These ready signals control insertion and the wait states of the CPU. The oscillator or OSC as seen as pin 12 in Figure 2 is predominantly used so that other systems can derive a control, namely the buffered crystal. There are two standard clock synchronization methods on the 8284. The two options available are defined through the frequency/crystal select as seen on Figure 2 on pin 13. In this system, the frequency/crystal select is grounded so that clock synchronization can be enabled by the crystal and not an external frequency input (EFI). The schematics for the completed system clock can be seen in Appendix A: System Clock. Both functionalities of the RESET synchronization and clock synchronization can be seen in Figure 3. The top half of Figure 3 details the clock synchronization and the bottom half details the RESET synchronization.

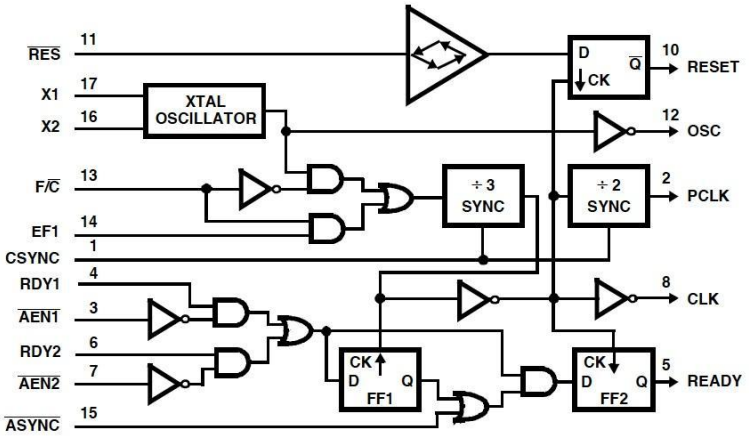


Figure 3: 82C84 Clock Generator Functional Diagram

The results of the physical demonstration validate all the above primary functions of the system clock. Given that the system clock is functioning properly, it will drive the CPU, which will then drive all other primary functions of the system.

- **Central Processing Unit**

The main components of this systems central processing unit are Intel's 16-bit 8086 microprocessor, 74LS373 latches and 74LS245 buffers. Each component is dependent on the previous functioning correctly. The 8086 and latches/buffers form the complete CPU unit and are unaltered from the successful previous lab.

- **Intel 8086 Processor**

The Intel 8086 is at the core of the system. Whereas other components make possible the functioning of the 8086, or other functions, the 8086 in itself is the brain: able to communicate with components via I/O, as well as process data, and control the system as a whole.

The 8086 has two modes of operation, each having a slightly different pin-out scheme. In MIN mode, the processor is allowed full control of the I/O bus. This is the more simple of the two operation modes, but is only for use with a single processor. MAX mode enables the ability to have multiple processors sharing the same bus. This is achieved through the use of a HOLD-type interrupt system. The processor requiring use of the bus sends a lock signal to the coprocessors (all running in MAX mode), which causes the coprocessors to effectively detach themselves from the bus, giving bus use to the original processor until it has finished its use of the bus. For the purposes of this lab, MIN mode is sufficient, due to requiring only one processing unit.

In MIN mode operation, the 8086's pins can be grouped into three interfaces: the Memory/IO interface, the Interrupt interface, and the DMA interface. Each interface suits a purpose, with IO being responsible for the communication of the processor with other components. The interrupt interface handles hardware and software interrupts to the processor, while the DMA interface handles processing holding, in which the processor "detaches" from the bus, giving control to other devices.
The IO interface consists of all the address/data bus (AD) lines connected to the processor, as well as the data control lines. The bus lines are initially multiplexed together to save pin room on

the IC, and must be demultiplexed by the latches and buffers in order to enable use of the system. Address lines specify the address of a device to interact with, while data lines transfer information between these devices and the processor. Data control lines give more information on the data operations, such as status/read-or-write/etc.

The interrupt interface controls the operation of the processor. It includes the hardware and software interrupt pins (INTR and NMI, respectively), as well as the hard reset (RESET) coming from a push-button switch. These are used to synchronize operation of the processor with operation of other components. When a device requires processing, a hardware interrupt is deployed to the processor, causing a cease in communication on the bus until the processor has completed the task. In other situations, such as the use of "listener" software, a software interrupt may be of use. Once a software interrupt has been initiated using NMI, the interrupt by design cannot be reset until the entire system has been reset.

Finally the DMA interface deals with the operation of the processor with other components attached. Using the HOLD and HLDA pins (hold and hold acknowledge), the processor is able to logically detach from the bus and process information independently while other devices use the bus. This is a major feature in bus design and allows for efficient use of the buses in the system.

Below in Figure 4 is the pin diagram of Intel 8086 16-bit microprocessor. Later in the Experimental Results section certain signals are tested with the oscilloscope.

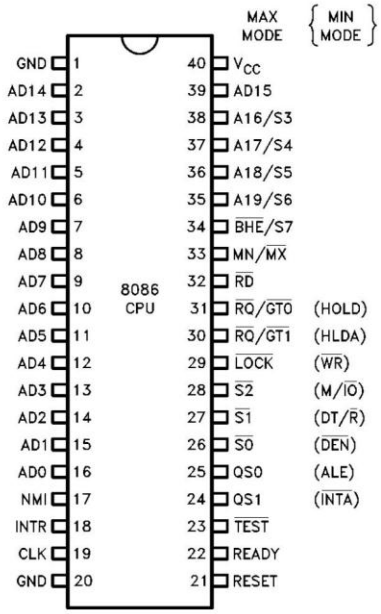


Figure 4: Intel 8086 Microprocessor Pin Configuration

- **Latches**

The 74LS374 that was used was manufactured using advanced Low Power Schottky technology. It is an Octal D-type Flip-Flop containing separate D-type inputs for each flip-flop. The 74LS373 has 3 different state outputs to be used in coordination with a bus. Since it is a latch IC it uses flip-flops which change asynchronously whenever the latch enable is high. However when it is

low the data is latched. Data only appears on the bus system when Output enable is set to low, but when set to high the bus output is in a high impedance state.
These are all of its features listed out:

1. Eight Latches in a Single Package
2. 3-State Outputs for Bus Interfacing
3. Hysteresis on Latch Enable
4. Edge-Triggered D-Type Inputs
5. Buffered Positive Edge-Triggered Clock
6. Hysteresis on Clock Input to Improve Noise Margin
7. Input Clamp Diodes Limit High Speed Termination Effects

Below in Figure 5 is a pin diagram of the 74L373 Latch. Later in the Experimental Results section the outputs of the Latch are tested to verify functionality.

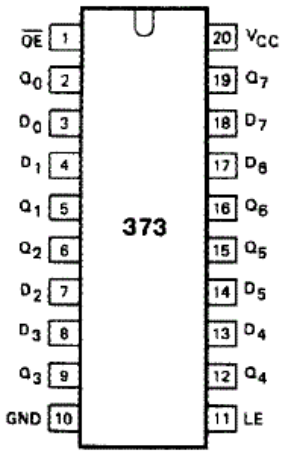


Figure 5: 74LS373 Latch
Pin Configuration

- **Buffers**

The 74LS245 octal bus transceivers are set up in way in which it allows asynchronous two-way traffic between data buses. It allows data transmission from the A bus to the B bus or from the B bus to the A bus, depending on the direction-control input. Implemented in the IC there is a control-function which minimizes external timing requirements. Its output-enable input makes it easy to disable the IC when the buses need to be isolated.
These are its features listed out:

1. 3-State Outputs Drive Bus Lines Directly
2. PNP Inputs Reduce DC Loading on Bus Lines
3. Hysteresis at Bus Inputs Improves Noise Margins
4. Typical Propagation Delay Times Port to Port, 8 ns

Below in Figure 6 the 74S245 Buffer can be seen. Later in the Experimental Results section various pins are tested for signal output.

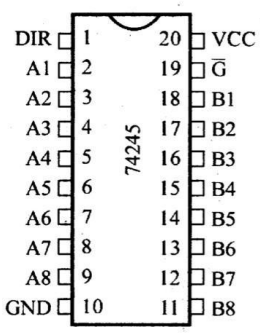


Figure 6: 74LS245 Buffer
Pin Configuration

The 74LS244, octal buffer is designed specifically to improve both the performance and density of three-state memory address drivers, clock drivers, and bus-oriented receivers and transmitters. It is made in a way in which the designer has a choice of combinations dealing with inverting and noninverting outputs, symmetrical, active-low output-control inputs, and complementary output-control inputs. It features a high fan-out, improved fan-in, and 400-mV noise margin.

These are its features listed out:

1. Hysteresis at Inputs to Improve Noise Margins
2. 3-State Outputs Drive Bus Lines or Buffer Memory Address Registers
3. Input Clamp Diodes Limit High-Speed Termination Effects

Below in Figure 6a the 74S244 Buffer can be seen.

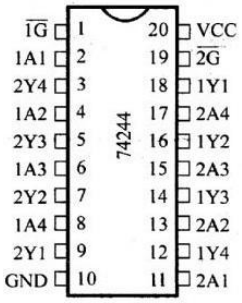


Figure 6a: 74LS244 Buffer
Pin Configuration

- **Memory**

Memory is important in interfacing 8086, it holds the actual program that will be running.

  - **28C256 32K X 8 EEPROM**

The 28C256 32K X 8 EEPROM is a high performance electrically erasable and programmable read only memory. Its 256K of memory is organized as 32,768 words by 8 bits. The 28C256 is accessed like a Static RAM for the read or write cycle without the need for external components. The device a 64 byte page register to allow writing of up to 64 bytes simultaneously. During a write cycle the addresses and 1 to 64 bytes of data are internally latched, freeing the address and

the data bus for other operations. Following the initiation of a write cycle, the device will automatically write the latched data using an internal control time. The end of a write cycle can be detected by DATA Polling of I/07. Once the end of a write cycle has been detected a new access for a read or write can begin.

These are its features listed out:

1. Fast read Access Time - 150 ns
2. Fast Write Cycle times
3. Low Power dissipation
4. Hardware and Software Data Protection
5. DATA polling for END of Write Detection
6. Single 5V+- 10% Supply
7. Automatic Page Write Options

Below in Figure 6b the 28C256 32K X 8 EEPROM can be seen.

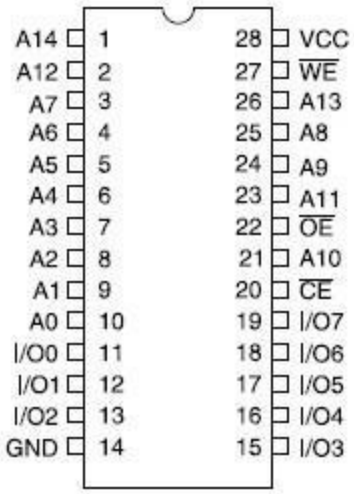


Figure 6b: 28C256 32K X 8 EEPROM
Pin Configuration

- **SRAM**

The CY7C128A chip is a CMOS static RAM organized as 2048 words by 8 bits. Easy memory expansion is provided by an active LOW Chip Enable (CE), and an active LOW Output Enable (OE) and drivers that are three-state. Writing to the device is accomplished when the Chip Enable and Write Enable (WE) inputs are both LOW. Data, which are on the pins IO0 -I/O7 are written into the memory location specified on the address pins A0 -A10. Reading the device is accomplished by taking Chip Enable and Output Enable and keeping them low while Write Enable remains high. Under these conditions, the contents of the memory location specified on the address pins will appear on the eight I/O pins. The I/O pins remain in high-impedance state when Chip Enable or Output Enable is high or Write Enable is low.

The following table lists which peripherals are addressed at each SRAM location.

Address	Peripheral
FFFFh	PP1 Control Register
FFFEh	PP2 Control Register
FFFDh	PP1 address 1
FFFCh	PP2 address 1
FFFBh	PP1 address 2
FFFAh	PP2 address 2
FFF9h	PP3 address 3
FFF8h	PP3 address 3
FFF7h	PP3 Control Register
FFF6h	Interrupt Controller Command (8259)
FFF5h	PP1 address 1
FFF4h	Interrupt Controller Data (8259)
FFF3h	PP1 address 2
FFF2h	Keyboard Command (8279)
FFF1h	PP1 address 3
FFF0h	Keyboard Data (8279)
FFEFh	UART address 8
FFEDh	UART address 7
FFEBh	UART address 6
FFE9h	UART address 5
FFE7h	UART address 4
FFE5h	UART address 3
FFE3h	UART address 2
FFE1h	UART address 1
FFDEh	Counter Command (8254)
FFDCh	Counter 2(8254)
FFDAh	Counter 1 (8254)
FFD8h	Counter 0 (8254)
FFD6h	LCD address 4
FFD4h	LCD address 3
FFD2h	LCD address 2
FFD0h	LCD address 1
FFCFh	7-Segment Display 2
FFCEh	7-Segment Display 1
FFCCh	Diodes
FFCAh	DIP Switches

The SRAM is only one of the devices that needed to be interfaced with the 8086, the I/O devices also needed to be connected.

- **Input/Output**
    - Keyboard

The Series 96 is uses conductive rubber to mate the appropriate PC board traces. It's features also includes having matrix circuitry easily customized legend, matrix circuitry, termination mates with standard connectors, tactile feedback to operator, 1,000,000 operations per button, it is compatible with high resistance logic inputs and is tested to rigid statistical process control to

insure that it is reliable. The keyboard itself contains 12 keys, which are the numbers 0 - 9, the "*" symbol and the "#" sign.

- LCD Display

The LCD display that will be used has a module 1 line x 16 characters on the original board. On the 8086 board it has a 4 line x 20 character display. The 4 line x 20 charcter display has a duty cycle of 1/16, it contains 5 x 8 dots for each character which includes the cursor. It takes in a + 5 V power supply. The LED can be driven by pin 1, pin 2, pin 15, pin 16 or A and K.

- 7 Segment Display & LEDs

A seven segment has ten pins, eight pins correspond to the eight LEDs, the remaining two pins are shorted. These segments come in two configurations, Common cathode (CC) and Common anode (CA). In CC, the negative terminals of all LEDs are connected to the common pins. The common is connected to ground and a particular LED glows when its corresponding pin is given high. In CA arrangement, the common pin is given a high logic and the LED pins are given low to display a number.

The Software System Design

To further simplify our code process, we split the code into five major versions. Each version was to implement a new functionality, and could be tested accordingly. This helped us avoid the confusion towards the end of the lab, and we were able to complete the assignment accordingly

- Initialization

Version 1.0 of the assembly program was essentially the given code. In this version, a simple greeting message was displayed on the LCD. This code was used to test the output functionality.

- Programming the 8259 and 8279

Using the UMBC trainer board essentially saved us the hassle of programming the 82C55 Programmable Peripheral Interface, as the board is already configured to have all components interfaced. At this point we only needed to program the 8259 Interrupt Controller and the 8279 Keyboard/Display Interface.

Of the eight command words for the 8279, only two are used, simplifying the project greatly. First the Mode Set command word is sent, sending information that informs the MPU of the presence of a 16-key system already decoded. The Clock command word is then sent, used to divide the internal clock of the 8279, to ensure proper function.

The 8259 Interrupt Controller is then configured. Of the four Initialization Command Words (ICW's), three are sent, ignoring ICW-3 since no slave 8259's are connected. The 8259 is programmed to be edge-triggered in a single unit configuration.ICW-2 then sets the correct interrupt vector table address, and ICW-4 is sent for further settings, such as single-mode and 8086-mode. Finally OCW-2 is sent, setting all lines except the one in question.

- Interrupts

For this version of the code, the goal was to have accurate interrupt handling, along with character recognition. We were successful, able to verify that key presses changed the screen (proving interrupt handling) and correct ASCII characters were saved and displayed on the LCD screen. To implement this functionality, we used the given vector table generation code as a

guideline to create our own. Once the table is stored in memory we focused on the interrupt routine. Through use of XLATB, we were able to use a key lookup table and correctly save ASCII characters. In order to ease programming, we decided to store the latest entered key into a location in memory.

- Functionality

This version of code (v4.0) includes mortgage calculations. The equations are located below in the report. This version can be tested as a standalone program, assisting in the debugging process as we do not require a trainer board to test.

- User Interface

The final version of our assembly code is dubbed v5.0 and seen as the final version of the software. There is not much to further explain here, as v5.0 simply integrates the standalone code of v4.0 with the interfacing/interrupt code of v3.0. In this version, User Interface is also focused on, with various user menus being displayed during operation to guide user input.

In order to simplify calculations and avoid exponential calculations, we pre-calculated all possible coefficients to the mortgage payments, attached below. The program then looks up the correct coefficient based on input arguments. This simplified our code greatly and also allowed for faster execution.

Grid of calculated pre-loan mortgage monthly payments:

	.02	**.05**	**.10**	**.25**
10	.1113	.130	.163	.280
15	.078	.096	.131	.259
20	.061	.080	.117	.252
30	.045	.065	.106	.250

Grid of rounded pre-loan mortgage monthly payments of 2^-n:

	.02	**.05**	**.10**	**.25**
10	3	3	3	2
15	4	3	3	2
20	4	4	3	2
30	5	4	3	2

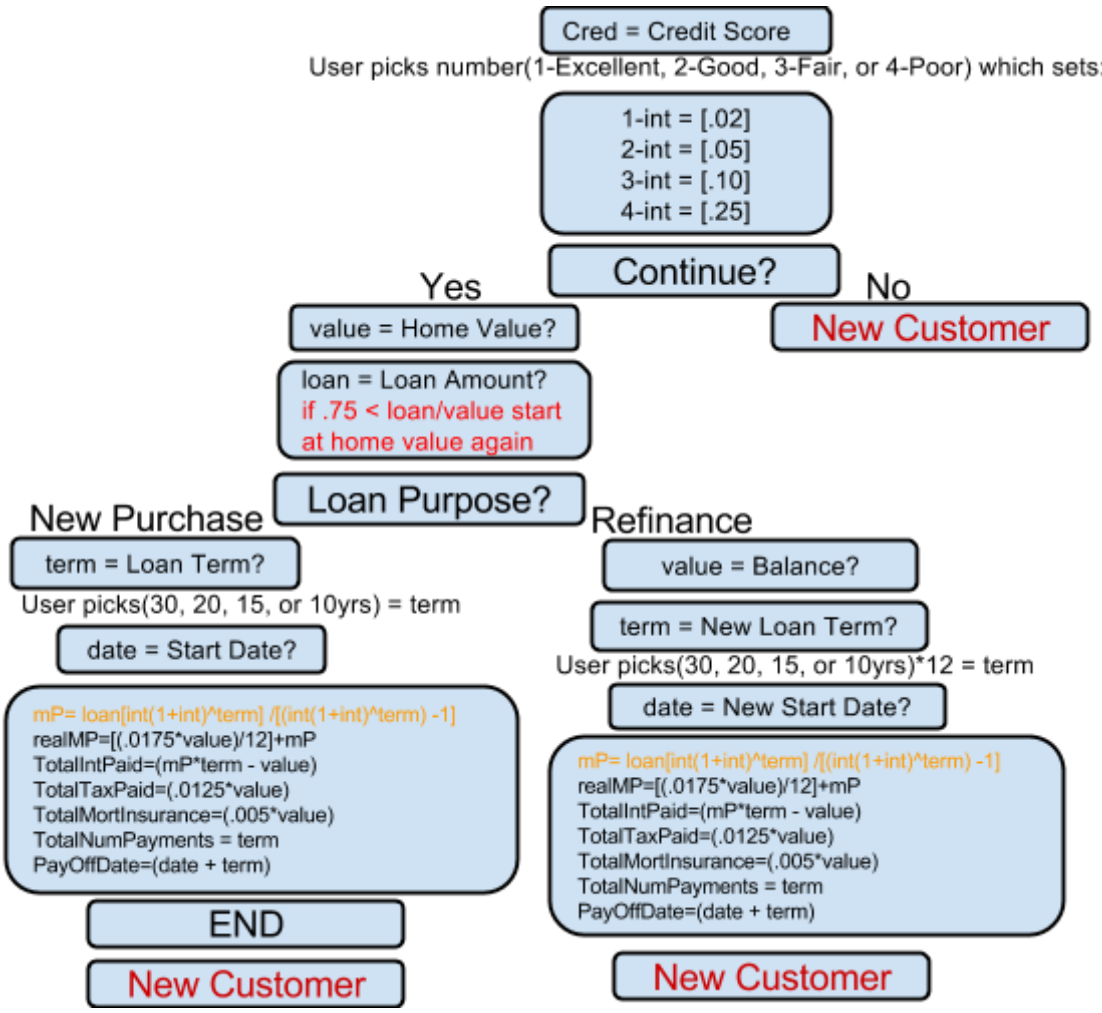


Figure 6c: Psuedocode Flowchart of Mortgage Calculator

Experimental Results

Prior to building the system, the datasheets for the IC's were reviewed, schematic were designed and expected values were predicted. Following that procedure ensured that testing upon demonstration would be successful. Testing was divided into three main efforts, one the system clock, CPU and EEPROM. The expected results of the CPU and EEPROM were dependent completely on the system clock function properly.

- **System Clock**

The system clock was constructed from the 82C84 clock generator, crystal and simple RC circuit. In order to ensure that the system clock was function properly a set tests were implemented. Knowing that the crystal would output a clock rate of 4MHz, it was expected that the CLK output would be 1.33MHz and that PCLK would output 0.666MHz (of 666kHz). In Figures 7 and 8 both expected results are met.

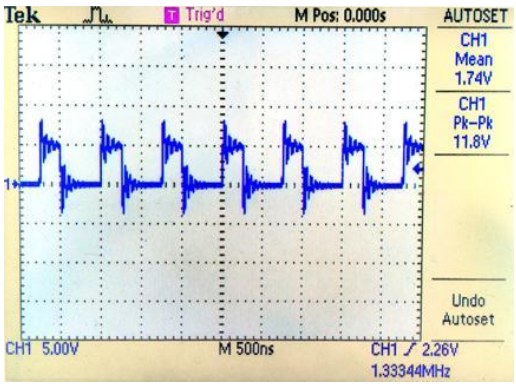


Figure 7: 82C84 Pin 8 CLK

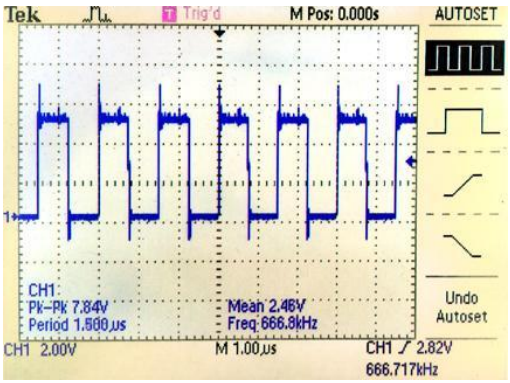


Figure 8: 82C84 Pin 2 PCLK

To confirm that the oscillator, OSC, was functioning properly, it was expected that it should look like a square wave with the same frequency of the 4MHz crystal. Samples were taken from both the crystal and OSC. Figure 9 to the left shows a clean signal sine wave with an output of 3.99MHz. The expected reading was 4MHz, but knowing that crystals are very sensitive, it was not a surprise that it was not exact. Figure 10 below is the sample OSC reading. Output rate requirement was met at 4MHz, but the square wave is not very clean. However, it is a square wave and met within the expected results.

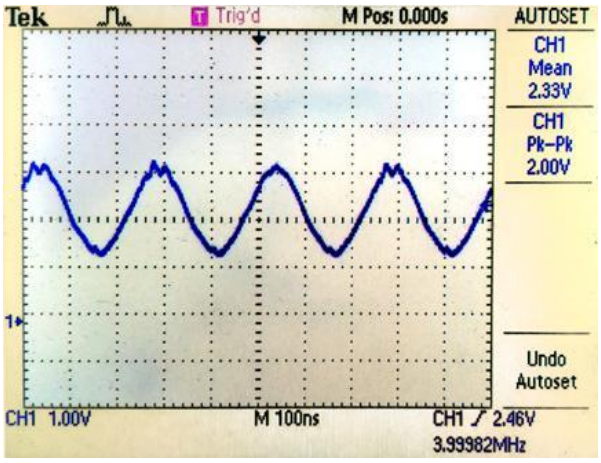


Figure 9: 4MHz Crystal

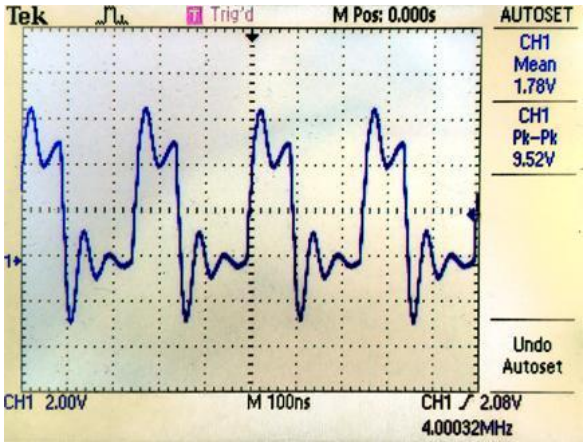


Figure 10: 82C84 Pin 12 OSC

The last two signals needed for complete verification were the RESET and READY signals. It was expected that when the one, the reset switch would switch the state of the machine and two that the READY would hold a constant wait state. In Figure 11 the RESET pulse was sample to show a change in states and in Figure 12 the constant wait state is captured.

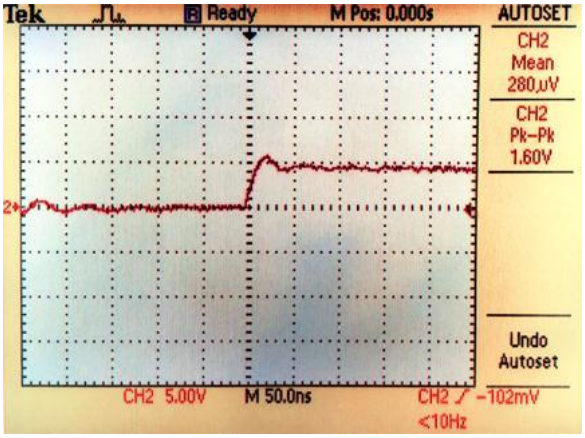


Figure 11: 8284 Pin 10 RESET

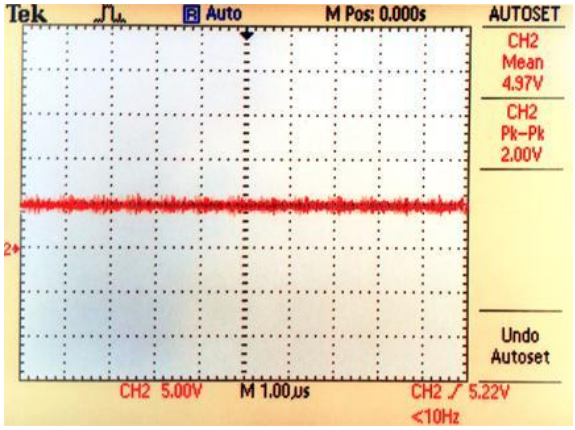


Figure 12: 8284 Pin 4 READY

Having validated CLK, PCLK, OSC, RESET, READY and crystal, further testing of the CPU and EEPROM can be diagnosed.

- **Central Processing Unit**

Figure 13 and Figure 14 shows the Address Latch Enable and Q0 of 8086 respectively. Looking at the figure below it shows that the 8086 is resetting properly. The Address Latch enable is taking in 5V and giving out 250 MHz. The peaks in the ALE signal indicate enabling of the latches used to demultiplex the buses. Q0 is taking in 2V giving out a frequency 83.33Mhz, currently showing a signal of not much interest. Once the 8086 is interfaced with memory, these lines will show much more interesting results. The 8086 is viewed during reset at this time resets correctly, with values returning to original state. If a device had been interfaced with the 8086, more evident changes would be available.

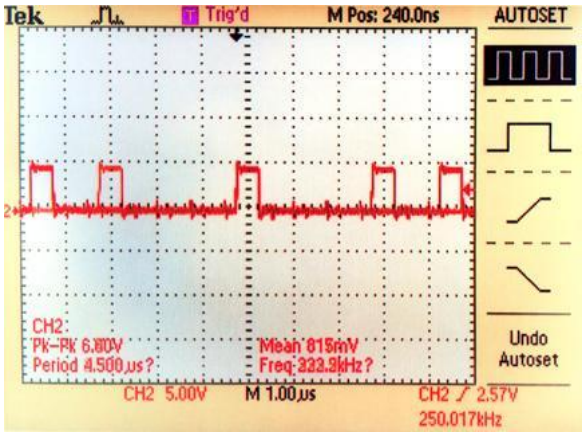


Figure 13: 8086 ALE

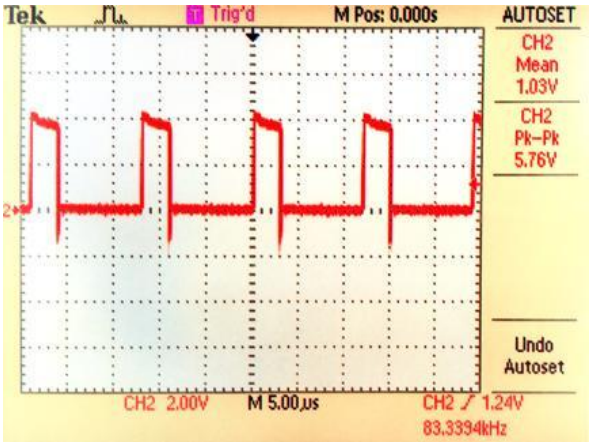


Figure 14: 373 Q0

In Figures 15-17 the signal outputs of the 74LS245 buffer are tested.

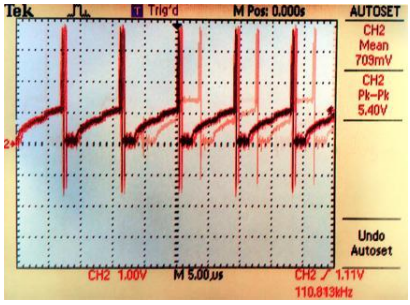


Figure 15: 245 B0

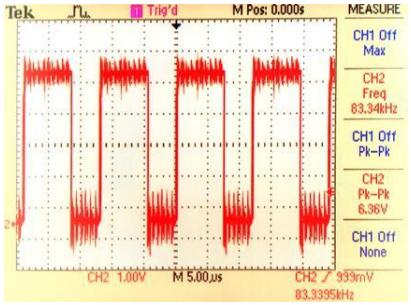


Figure 16: 245 DIR

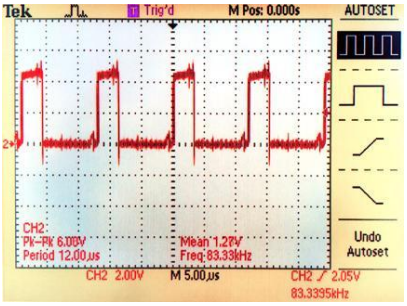
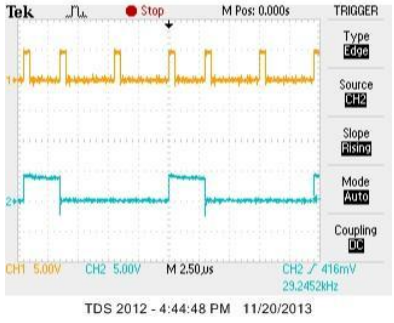


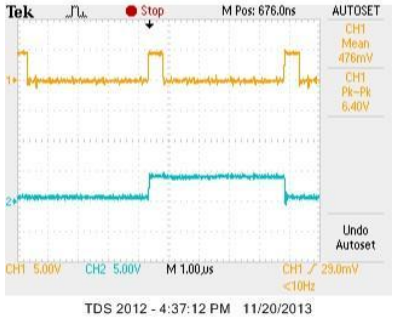
Figure 17: 245 AD0

- **EEPROM Unit**

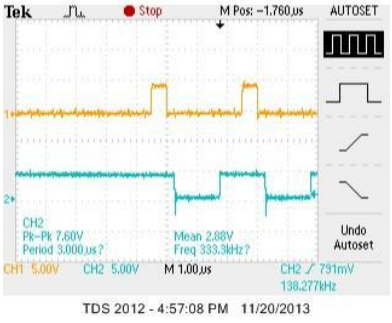
In order to check the functionality of the EEPROM several tests were recorded. It is important to recall that the 8086 has a 16-bit data bus and therefore can use what is called a write strobe to select between memory banks. The low and high banks of memory are selected using combinational logic of the Write pin (WR), Bus High Enable (BHE) and Bus Low Enable (BLE). The Low Bank is selected using BLE, in this case A0, and the High Bank is selected using BHE. For the purposes of testing, results were only compared to BLE or A0, not BHE.



Ch1: ALE, Ch2: A0



CH1: ALE, Ch2: CE

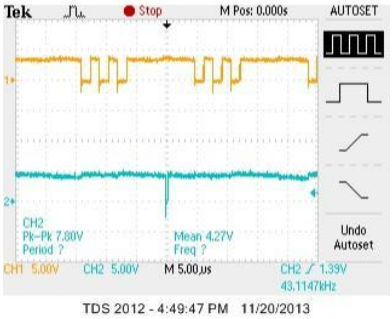


Ch1: ALE, Ch2: RD

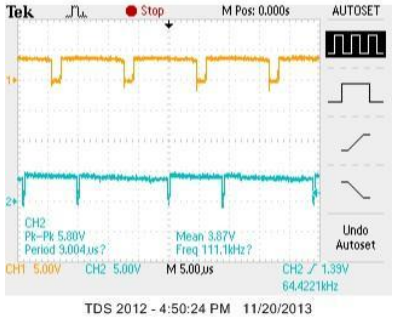
Address Latch Enable (ALE) tells the processor there is a transmission of memory address information happening. Below describes the logic required for data transmission between memory and CPU.

BHE	A0	
0	0	Word on $AD_{15} - AD_0$
0	1	Upper byte on $AD_{15} - AD_8$
1	0	Lower byte on $AD_7 - AD_0$
1	1	Upper byte on $AD_7 - AD_0$

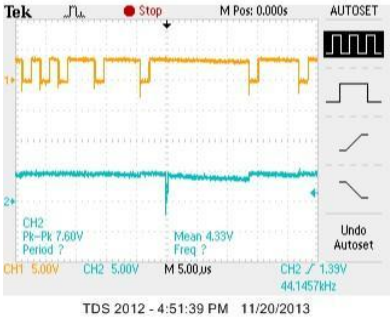
Whenever the Read Signal (RD) is a logic zero, data is taken from the memory. Below are three samples taken with RD and D1.



Ch1: RD, Ch2: D1 sample 1

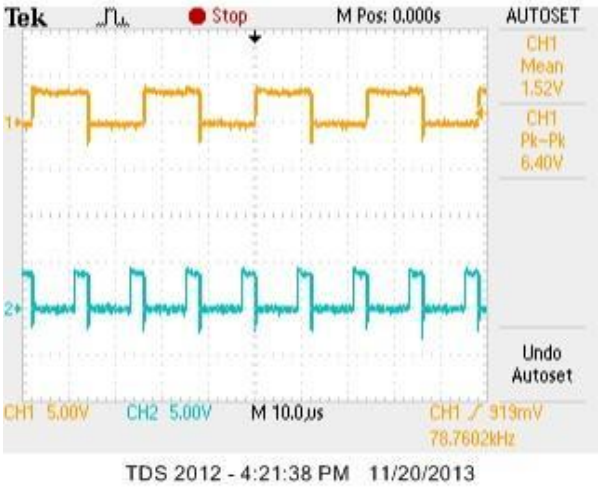


Ch1: RD, Ch2: D1 sample 2

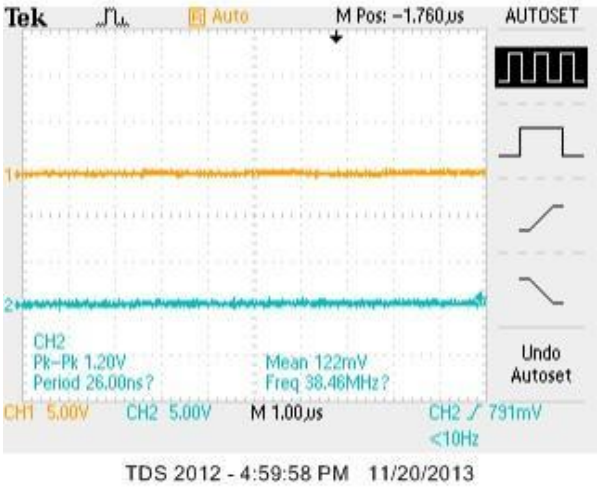


Ch1: RD, Ch2: D1 sample 3

Below is a sample that shows A0 (BLE) and A1. Here we can see that when A0 is a logic zero, A1 switches to a logic 0. Another sample is taken here to show that both the Write Signals for Low and High Banks are zero which is correct because we are not writing any data.



Ch1: A1, Ch2: A0



Ch1: WRL, Ch2: WRH

The results taken from phase two show that 8086 processor is receiving write signals generated from the infinite loop hard coded in the High and Low banks of memory. The data is matches the code that was hard coded so it stands to reason that this system is working.

- **Code Testing**
  - Case Testing

Case testing for the system was simple, yet extensive. We first tested by checking interrupt handling, as well as character recognition. These initially were not functional, however after study and tinkering, the interrupt handling worked perfectly. Once character recognition was functional, we were able to input keys and check recognition based on the character displayed on screen. Our software at this point excelled, and recognized all characters, without dashes, which was an issue for other groups. Finally, we ran through our mortgage calculations and tested the output of these formulas, which was verified as being functional, and accurate to a reasonable degree.

Conclusions

In order to complete this lab it was important to develop an understanding of how peripheral devices are interfaced with the microprocessor, as well as how the memory is virtualized within the processor, in order to program correctly. Seeing the LCD display various information as the program executed was interesting and helped solidify concepts featured in the lab description. Careful planning was a huge component in this team's success. Schematics were designed early, which allowed time for errors to be fixed, and many versions of software were developed, in order to test more easily. Each time we checked our code for our expected outcomes we were always surprised to find that our careful planning paid off, with a few changes of course. We completed the project with minor debugging and were overall very pleased with our results, see Figure 18 for our initial build, and final system implementation. We feel confident in the operation of our system.

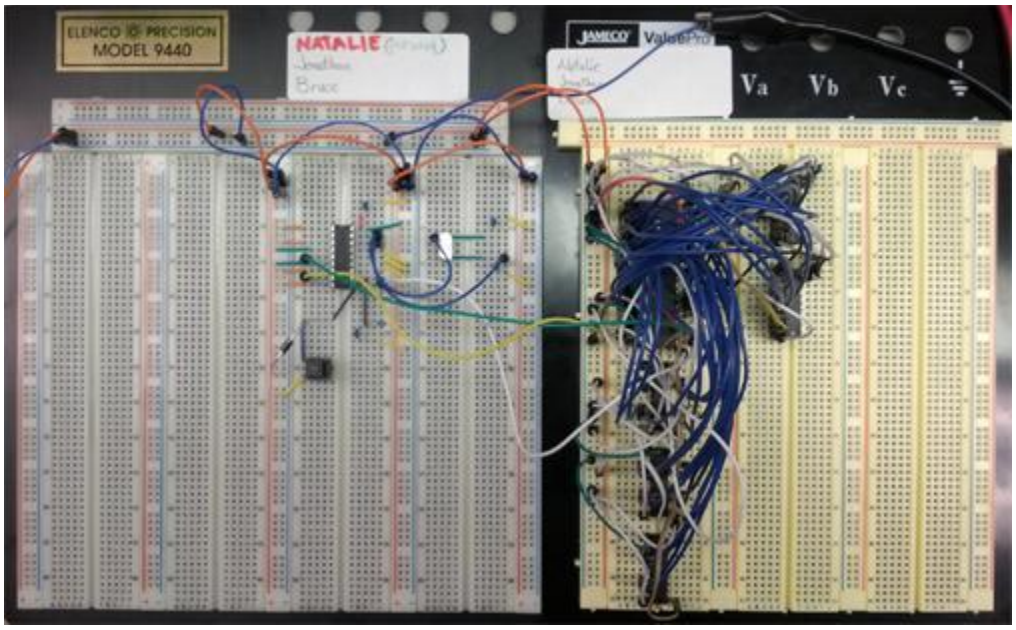
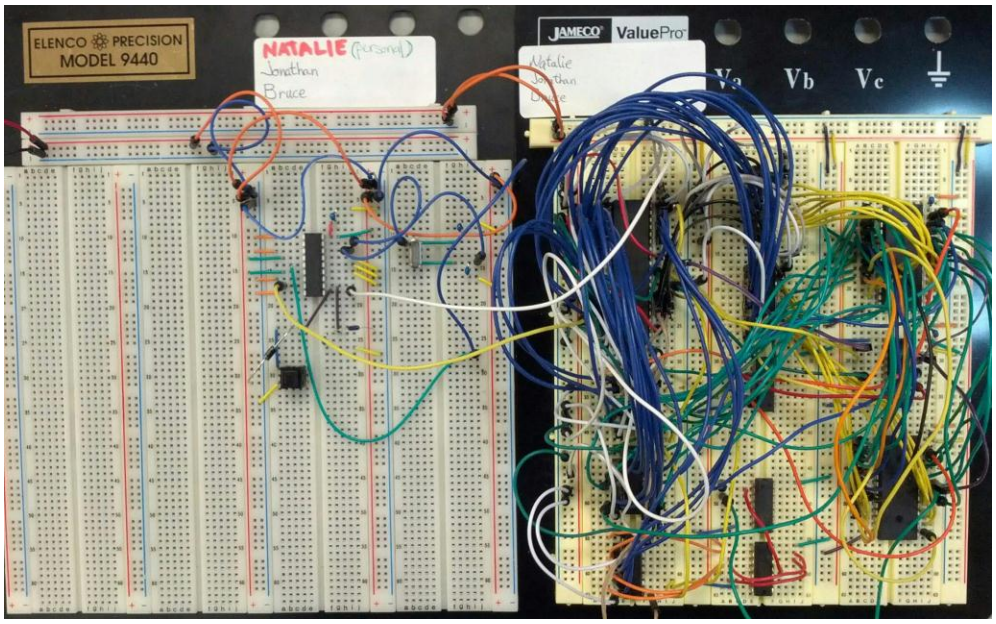


Figure 18: Phase 1 Clock Generator and CPU Implementation



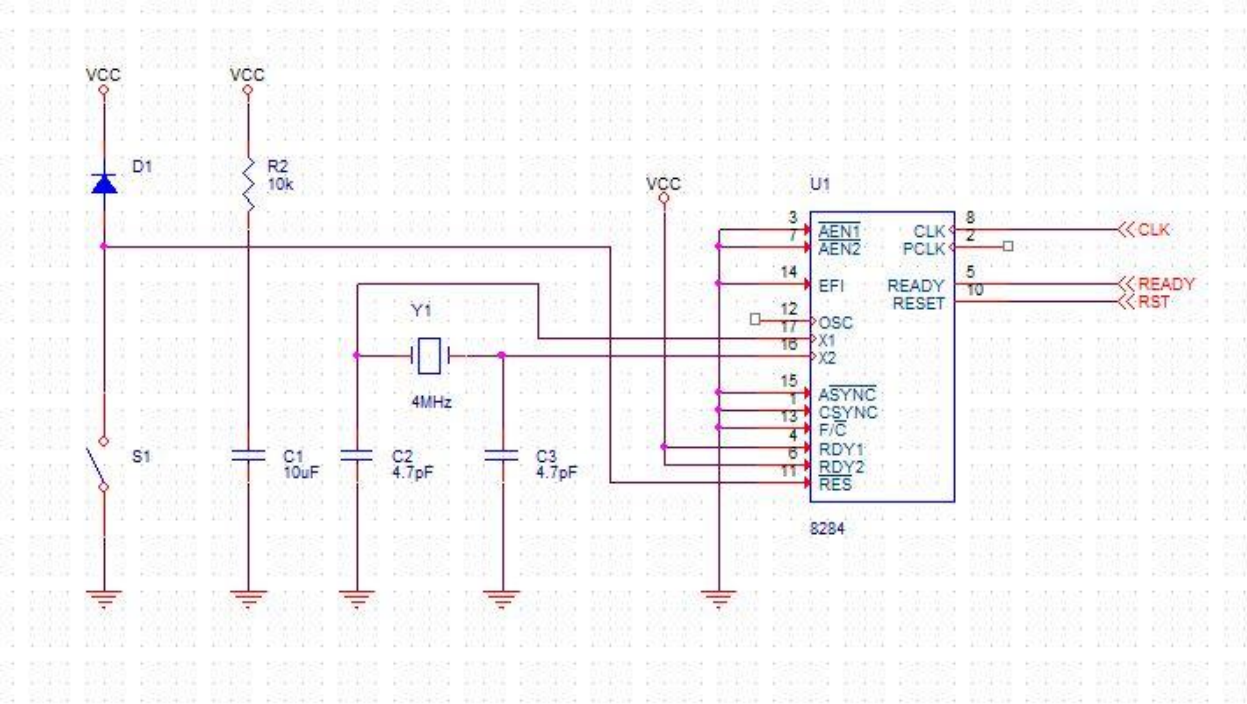
Phase 2 Clock Generator, CPU and EEPROM Implementation

References

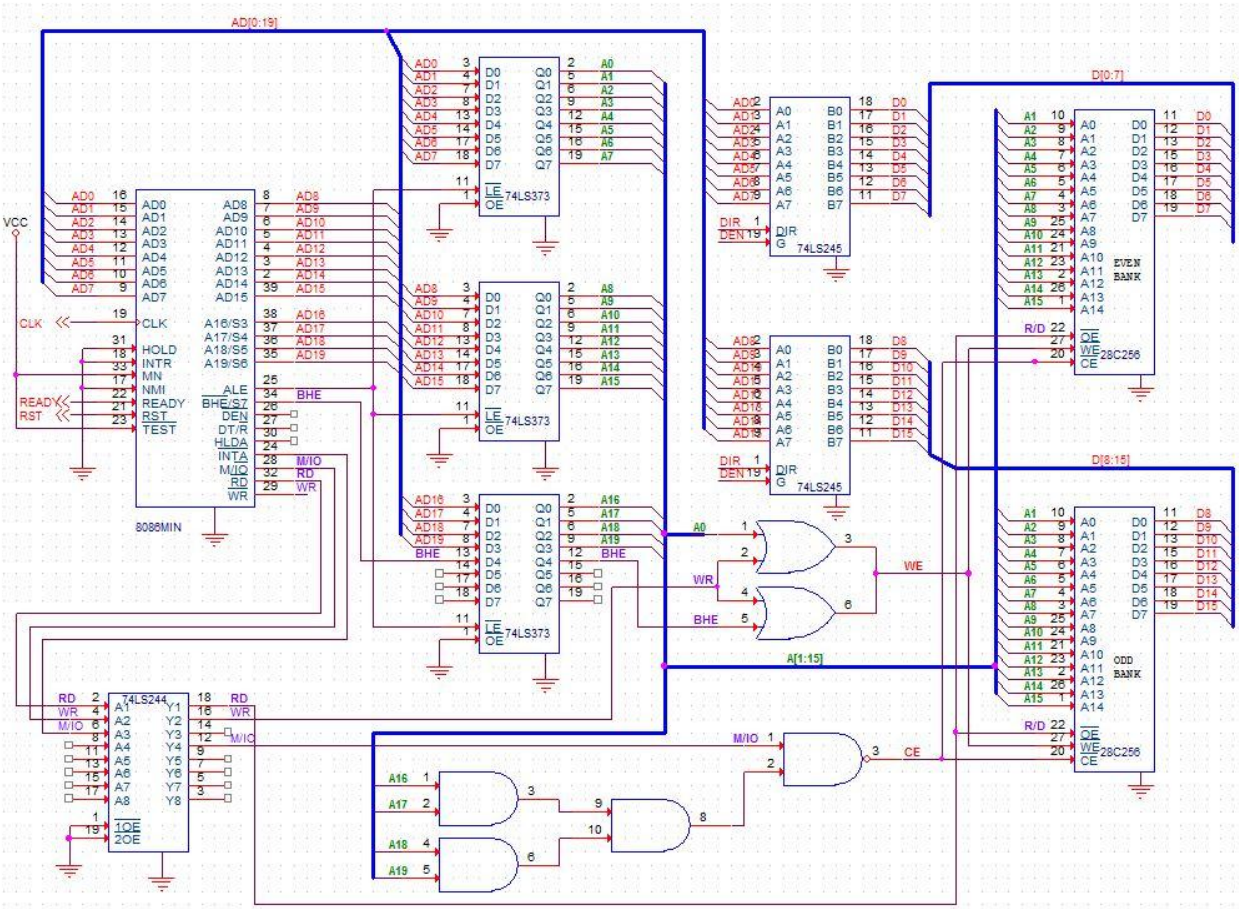
Brey, Barry. The Intel Microprocessors: Architecture, Programming, and Interfacing. Upper Saddle River: Prentice Hall PTR, 2009. Print.

Mano, Morris and Michael Ciletti. *Digital Design, 4th Edition*. India: Dorling Kindersley Pvt. Ltd., Pearson Education South Asia, 2007. Print.

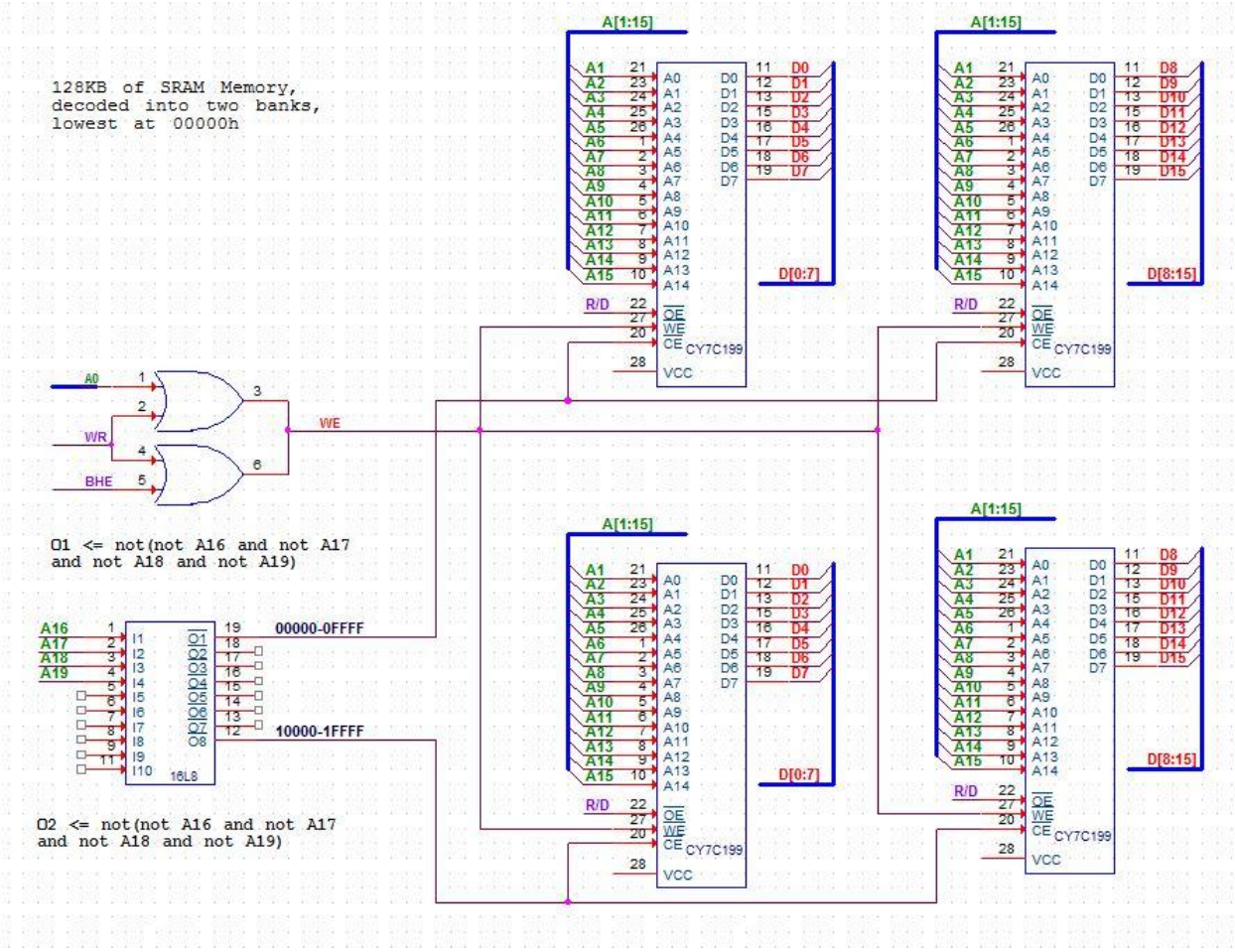
Appendix A-1: 8086 Based System - Clock Generator



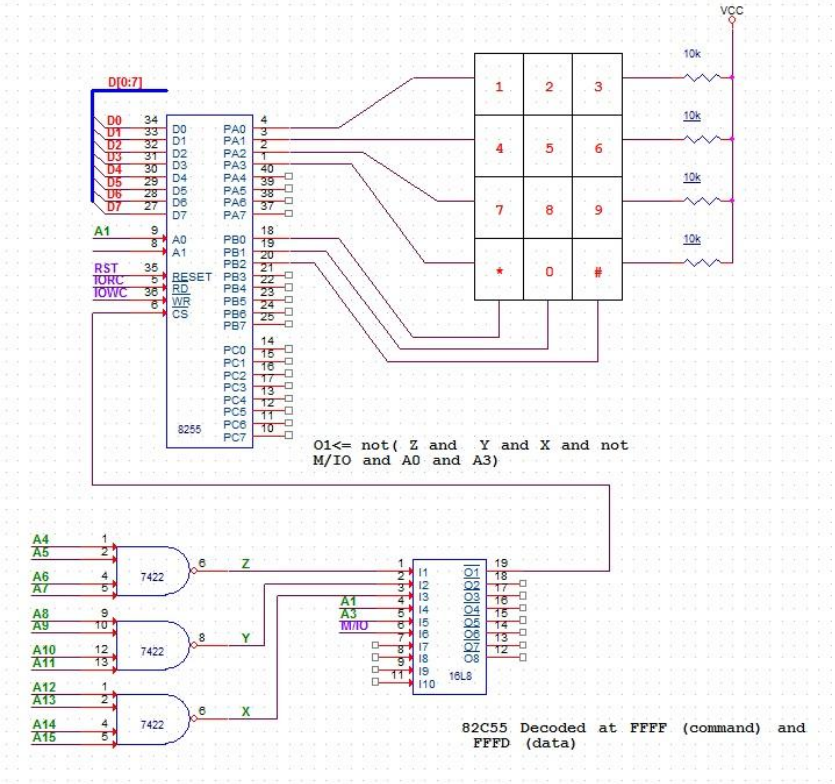
Appendix A-2: 8086 Based System – CPU & EEPROM



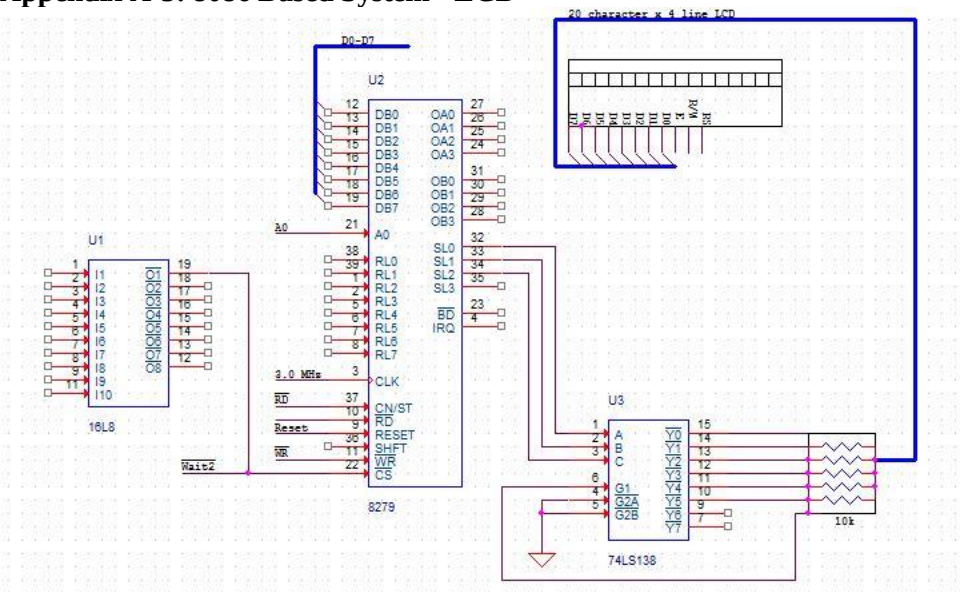
Appendix A-3: 8086 Based System - SRAM



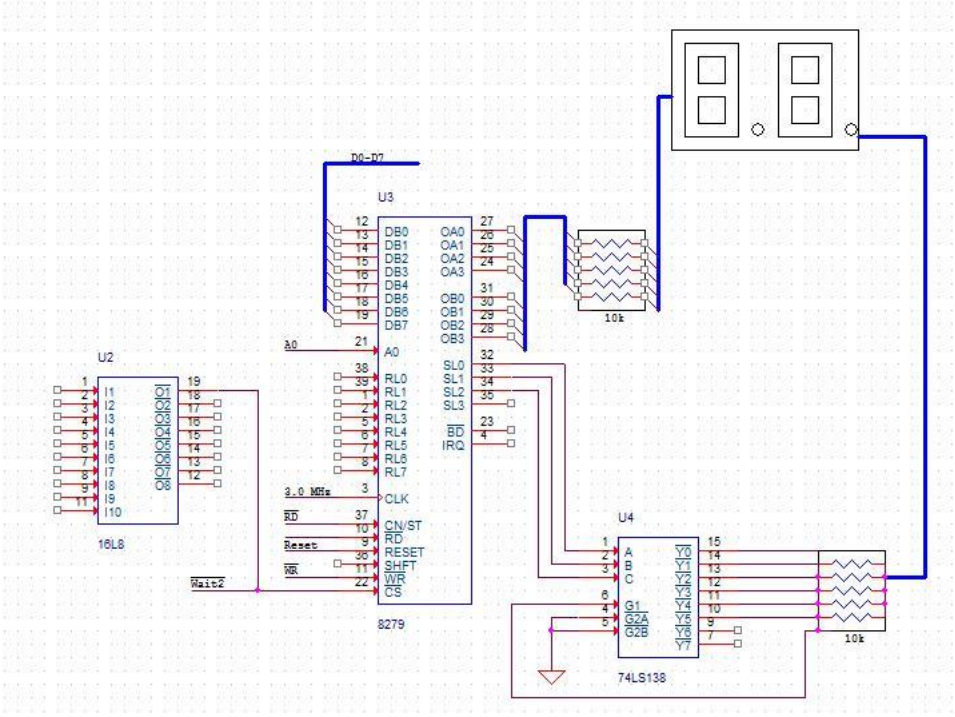
Appendix A-4: 8086 Based System – 82C55 and Keyboard



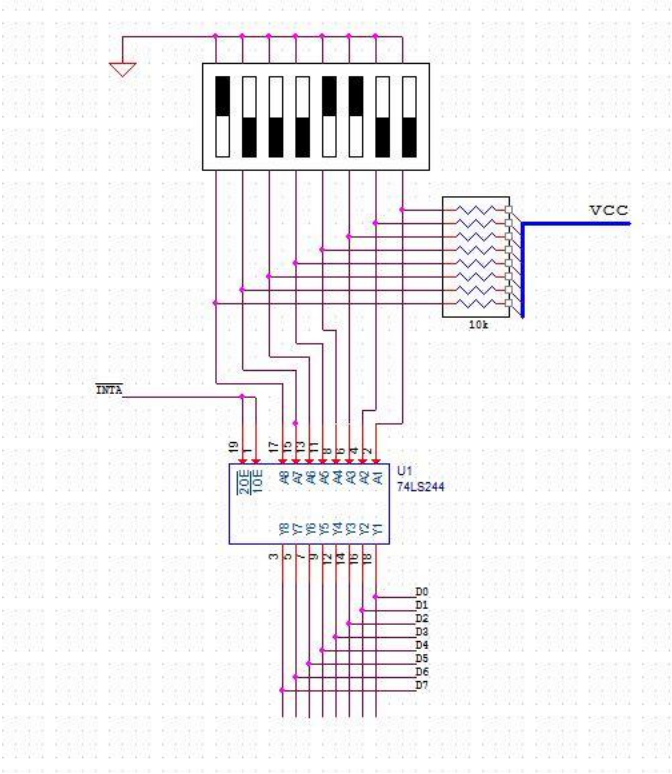
Appendix A-5: 8086 Based System - LCD



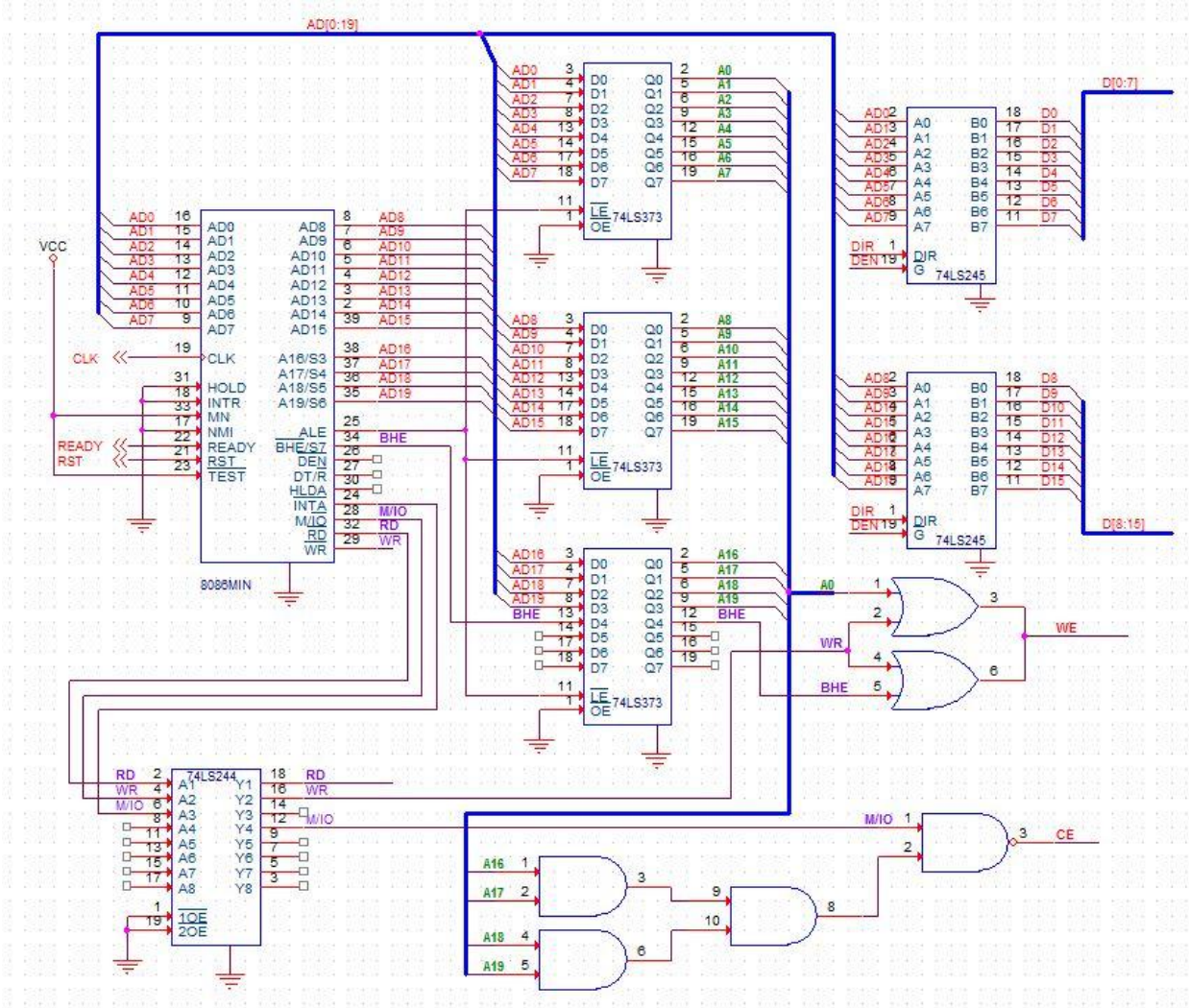
Appendix A-6: 8086 Based System - 7-Segment Display & Diodes



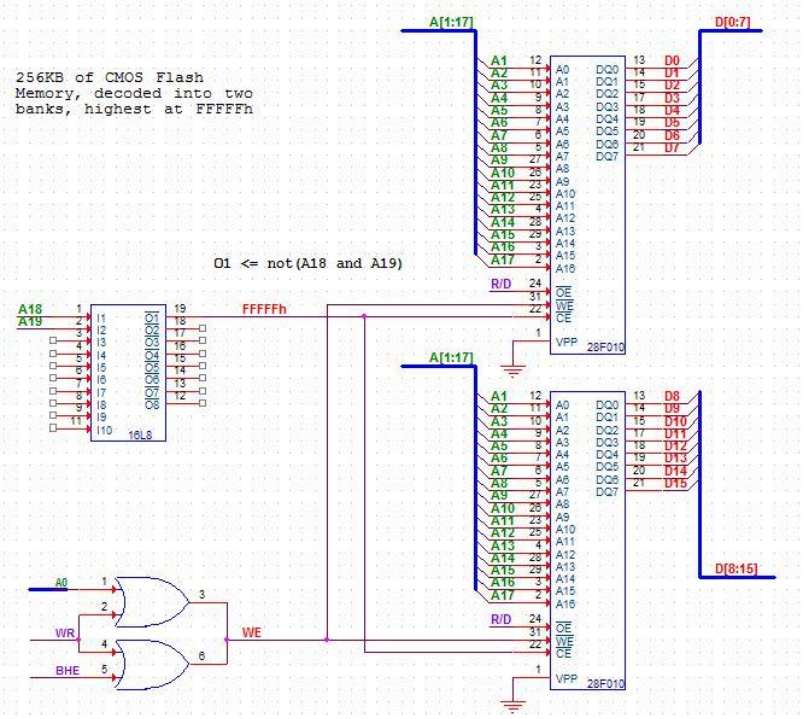
Appendix A-7: 8086 Based System - DIP



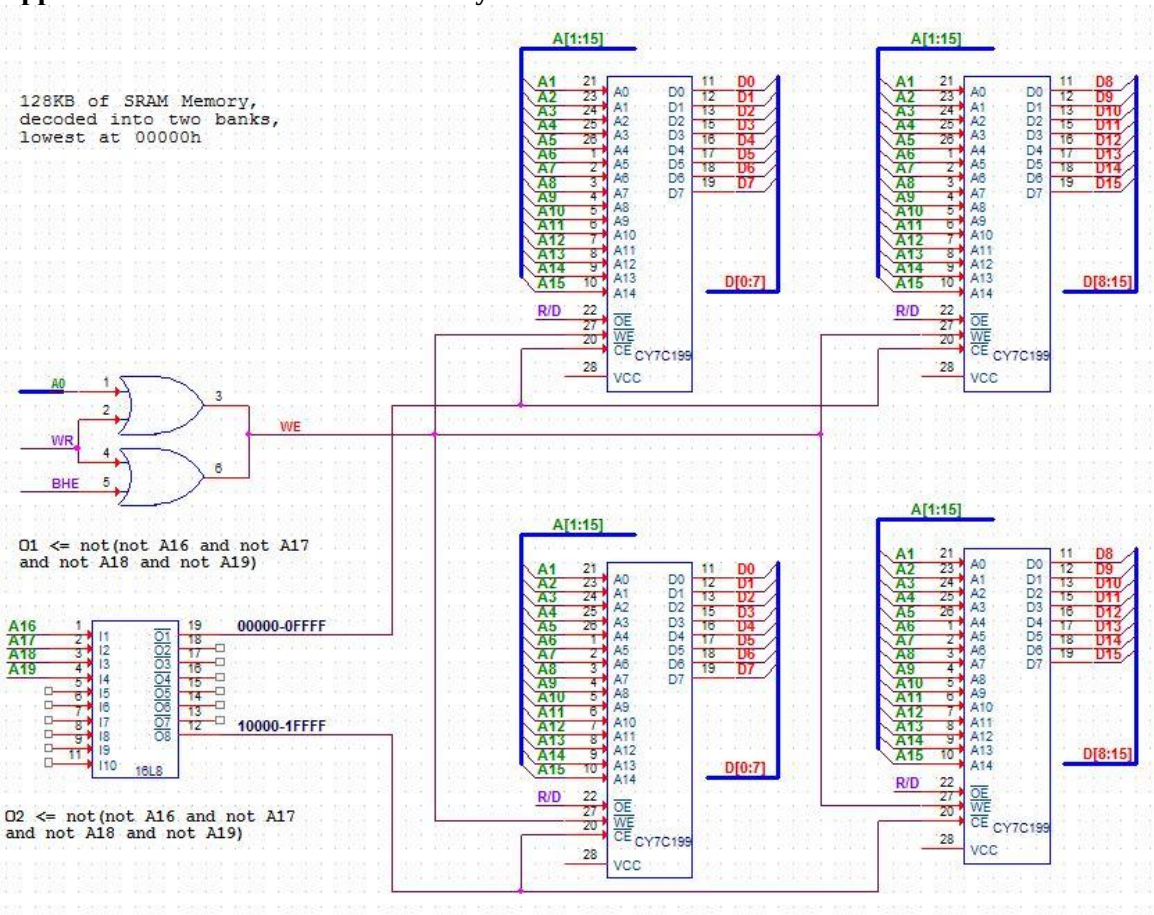
Appendix B-1: UMBC 8086 Trainer System - 8086



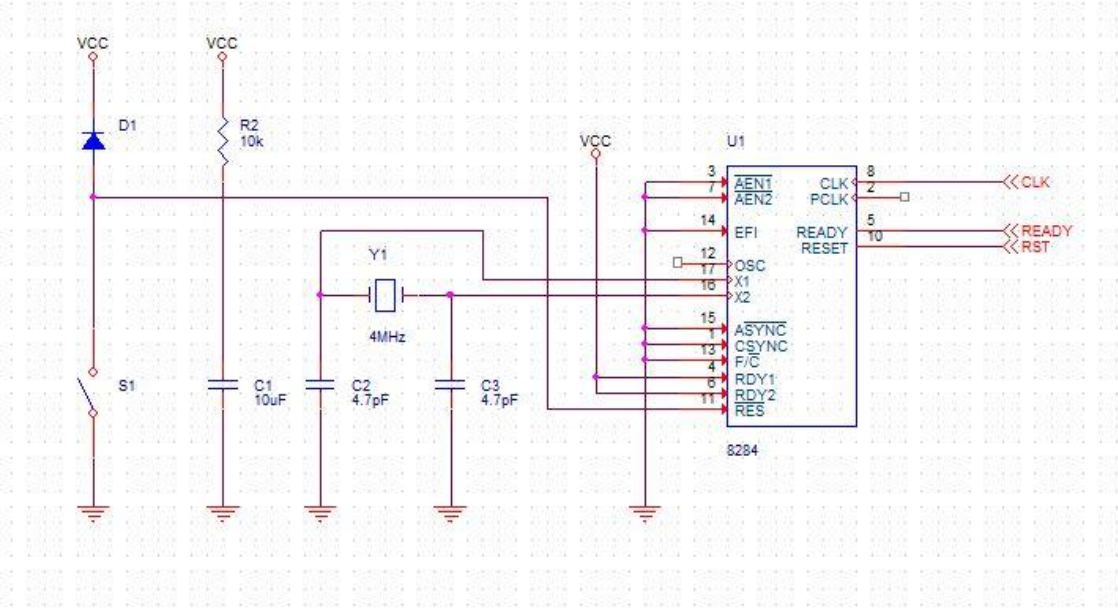
Appendix B-2: UMBC 8086 Trainer System – CMOS



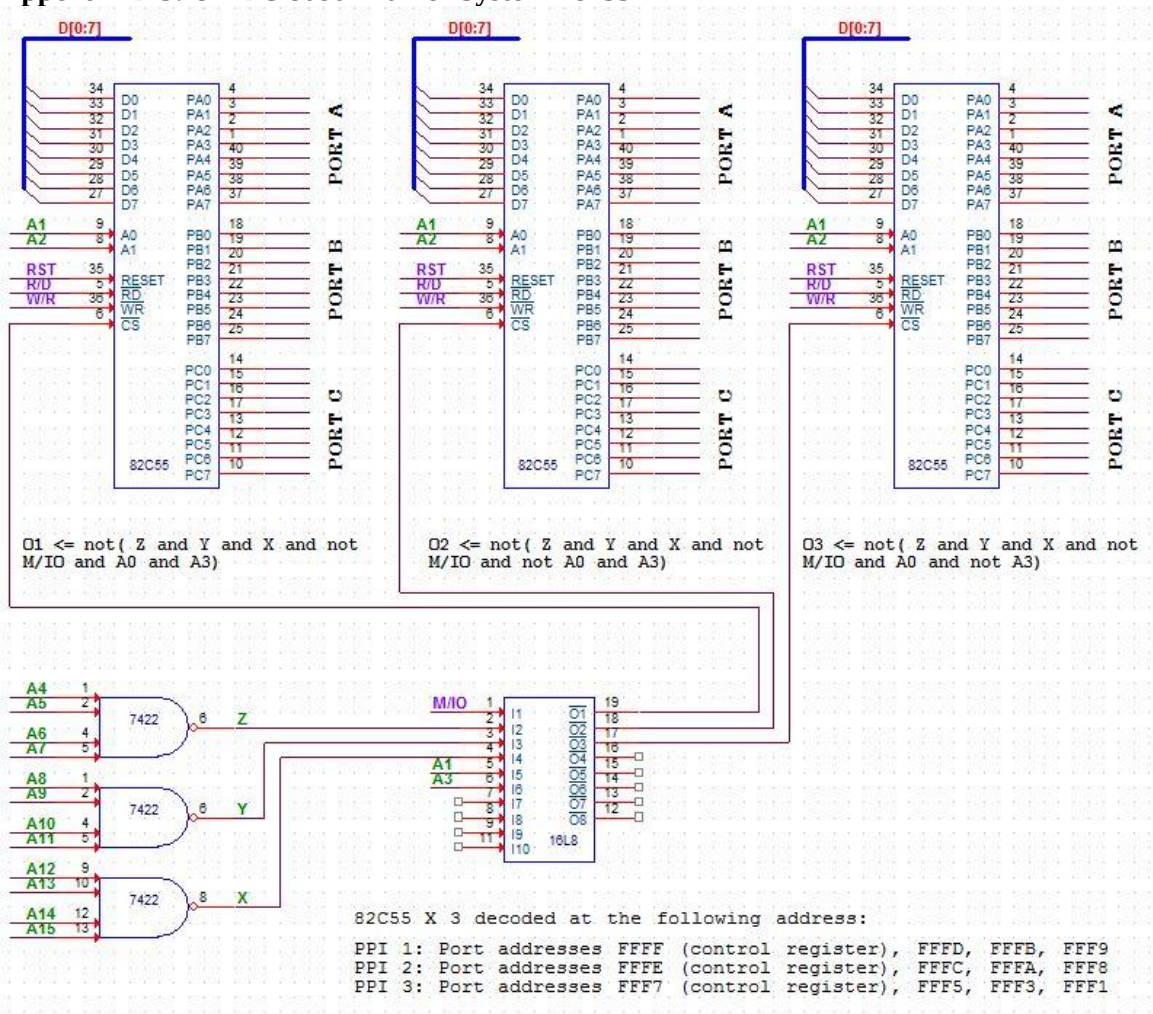
Appendix B-3: UMBC 8086 Trainer System – SRAM



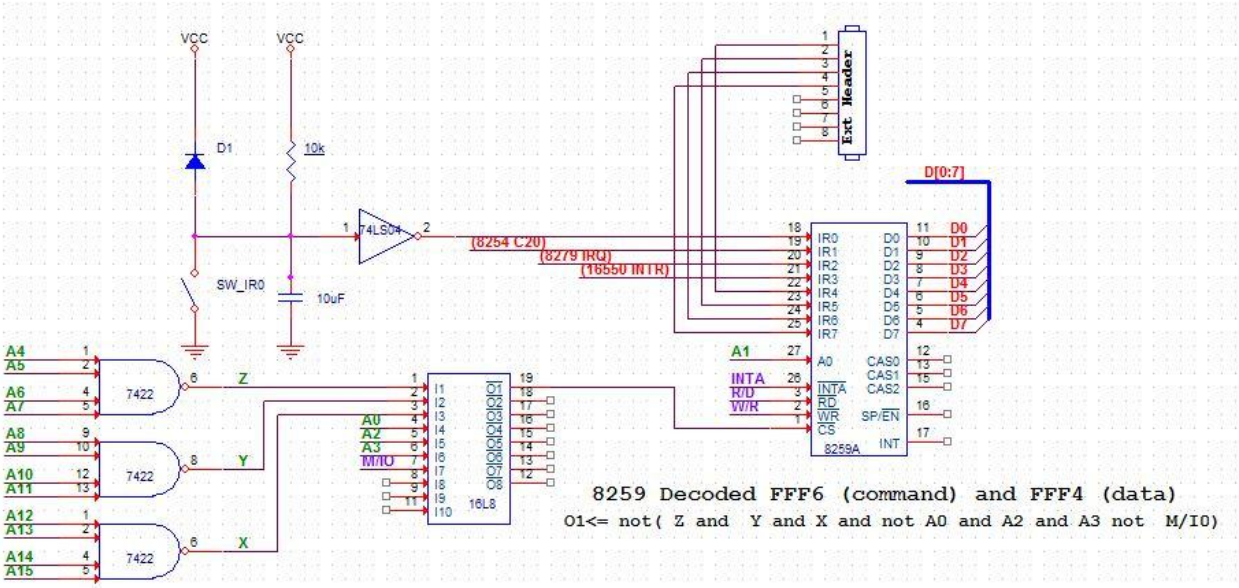
Appendix B-4: UMBC 8086 Trainer System - Clock Generator



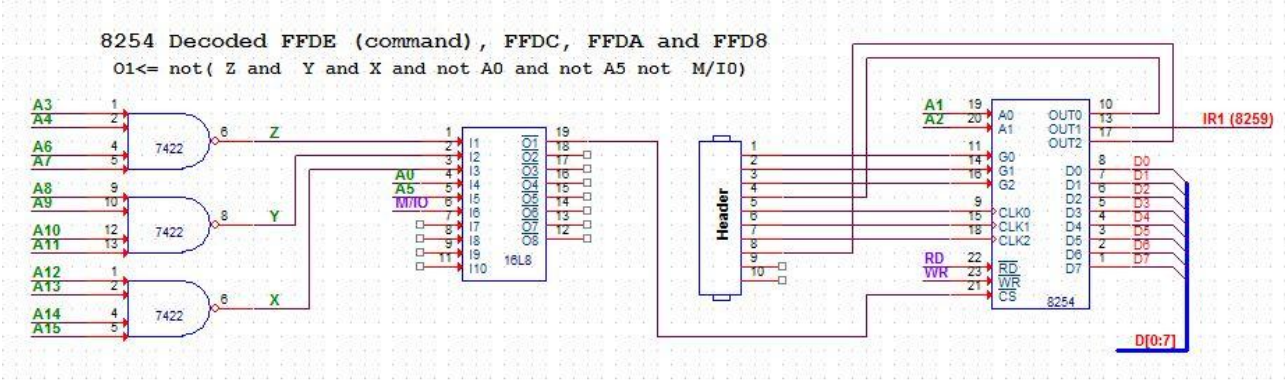
Appendix B-5: UMBC 8086 Trainer System - 8255



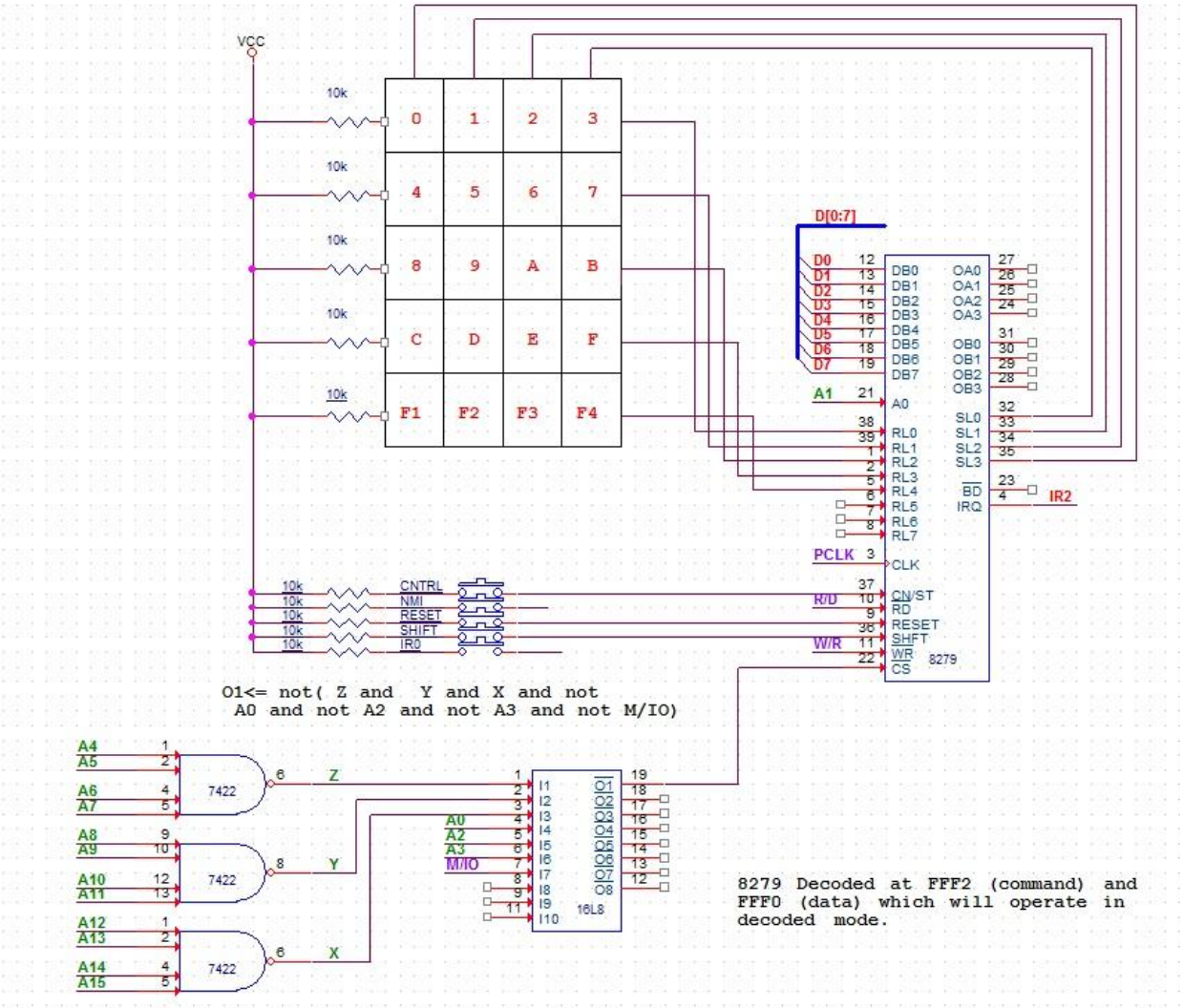
Appendix B-6: UMBC 8086 Trainer System - 8259



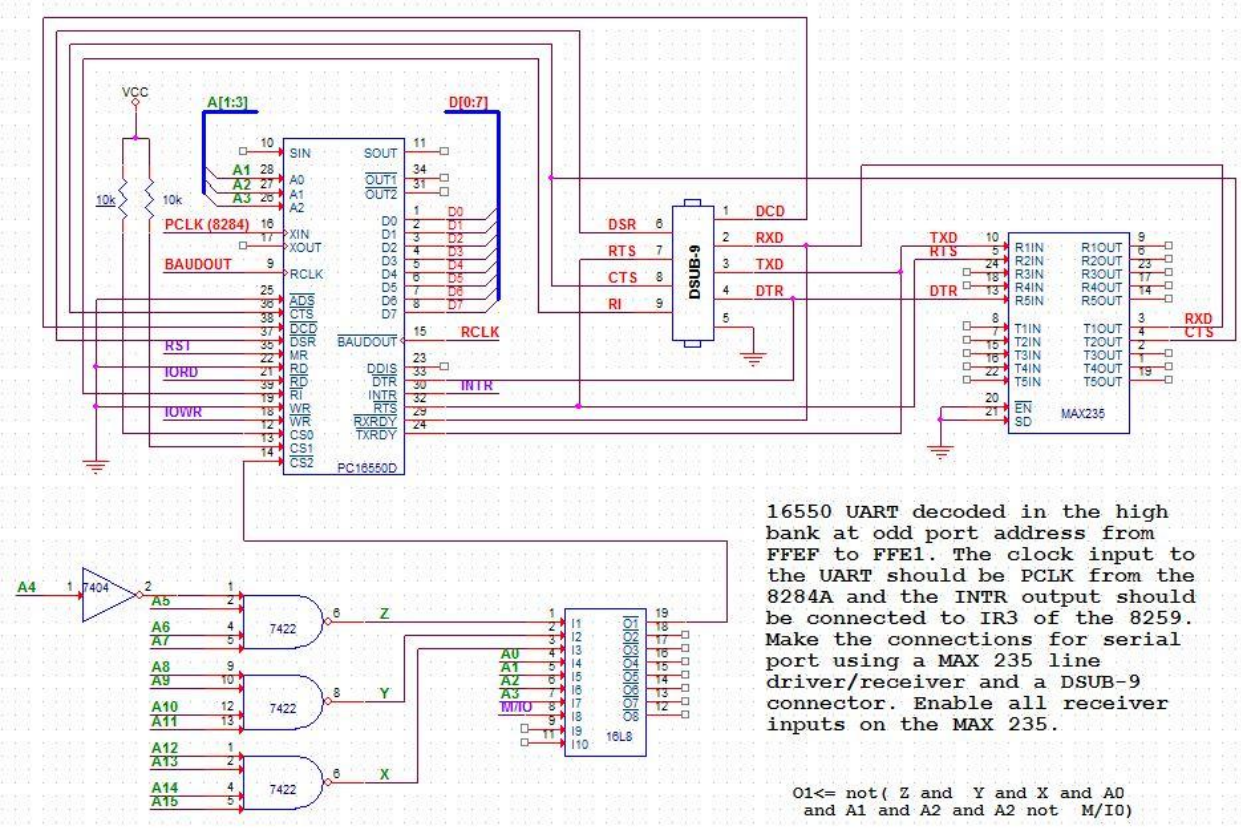
Appendix B-7: UMBC 8086 Trainer System - 8254



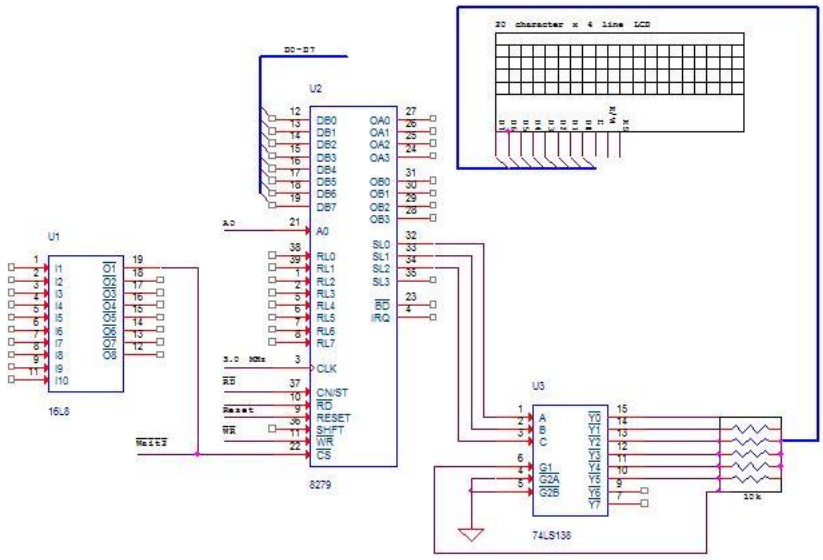
Appendix B-8: UMBC 8086 Trainer System - 8279



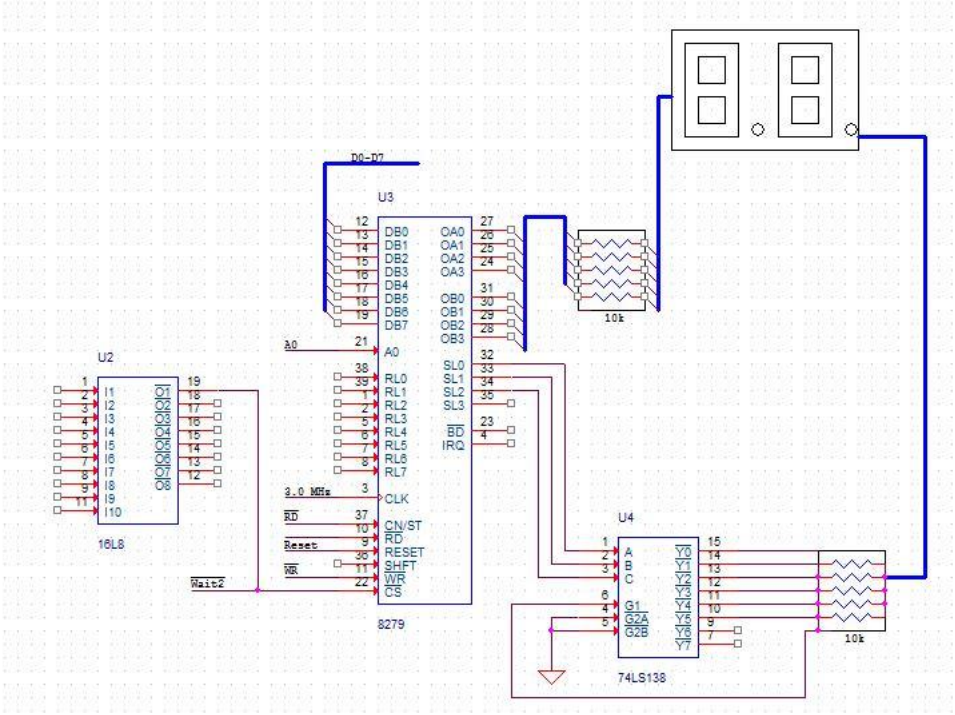
Appendix B-9: UMBC 8086 Trainer System – 16550



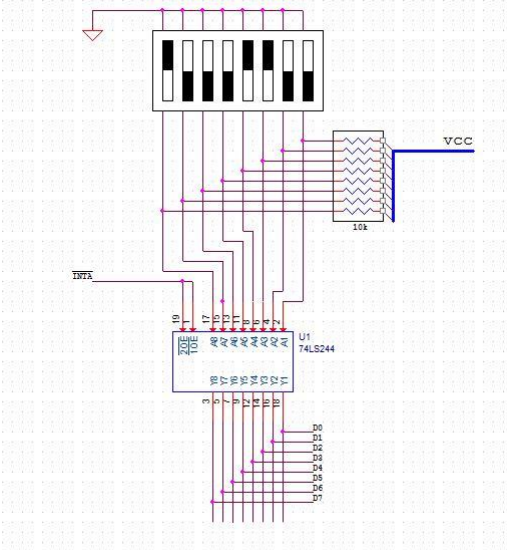
Appendix B-10: UMBC 8086 Trainer System – LCD



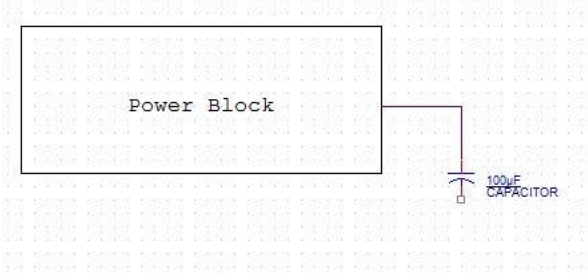
Appendix B-11: UMBC 8086 Trainer System - 7-Segment Display & Diodes



Appendix B-12: UMBC 8086 Trainer System – DIP



Appendix B-13: UMBC 8086 Trainer System - Power Block



Appendix C: Mortgage Calculator Code

```
; Names: Jonathan Peevy, Natalie Morningstar, Bruce Oshokoya
; Date: December 10, 2013
; Course: CMPE-310
; Assignment: Lab 4
; Version: 5
;
; This version of the lab 4 program builds on the functionality of the v2 program, which was able to display a message to LCD, and
; included programming of 8279 interface and 8259A interrupt controller.
; Version 3 adds the functionality of interrupts from key presses.


; Force 16-bit code and 8086 instruction set
; ----------------------------------------

BITS 16
CPU 8086

; Keyboard Display Controller Port Addresses (example)
; ----------------------------------------

%define KBD_CMD 0xFFF2 ;keyboard command register
%define KBD_DAT 0xFFF0 ;keyboard data register

; Interrupt Controller Port addresses (example)
; ----------------------------------

%define INT_A0  0xFFF4 ;8259 A0=0
%define INT_A1  0XFFF6 ;8259 A0=1

%define LED 0xFFCC

; Define States
; ----------------------------------

%define CRED_MENU 0
%define IR_ACCEPT  1
%define ENTR_VAL  2
%define ENTR_TERM 3
%define ENTR_DATE  4
%define ENTR_PURP  5
%define FINAL 6
```

```
; ------------------------------------------------------------------------------------------------------------
;

; ROM code section, align at 16 byte boundary, 16-bit code

; Code segment: PROGRAM, relocate using loc86 AD option to 0xF0000 size is variable, burn at
location 0x18000 in the Flash
; Use loc86 BS option to put a jump from 0xFFFF0 (reset location) to the code segment, burn at location
0x1FFF8 in the Flash
;

; ------------------------------------------------------------------------------------------------------------
;;THIS IS YOUR CODE SEGMENT, don't change anything till the cli instruction

section PROGRAM USE16 ALIGN=16 CLASS=CODE

..start

        cli                             ; Turn off interrupts

        ; Program 8279 - Keyboard/Display Interface
        mov dx, KBD_CMD
        mov al, 00001001b               ; Program Mode command byte. 000 prefix, 01 - 16 key system,
001 - decoded
        out dx, al
        mov al, 00111001b               ; Clock command byte. 001 prefix, 11001 to divide CLK by 2.
        out dx, al

        ; Program 8259A - Interrupt Controller
        mov dx, INT_A0
        mov al, 00010011b               ; ICW-1. 0001 prefix, 0 - edge triggered, 0 - don't care, 1 -
single, - ICW-4 not needed.
        out dx, al
        mov dx, INT_A1
        mov al, 00001000b               ; ICW-2.
        out dx, al
        mov al, 00000001b               ; ICW-4. 000 prefix, 0 - non-nested, 00 - no buffer (no slave
units), 0 - normal EOI, 1 - 8086 mode.
        out dx, al
        mov al, 11111101b               ; OCW-1. Set on all lines, except line IR1.
        out dx, al
```

```
        call KEYPAD_INIT

        ; Setup up interrupt vector table
        ; -------------------------------
        ;;; EXAMPLE, take this code out when not using interrupts. This loads two interrupt vectors at
vectors 0x09 and 0x0A

;       mov bx, 0
;       mov es, bx
 ;   mov bx, 8H * 4          ;Interrupt vector table 0x08 base address
  ;  mov cx, INTR1           ;INTR1 service routine
   ; mov [es:bx+4], cx       ;offset
    ;mov [es:bx+6], cs       ;current code segment

        mov cx, word 0
        mov es, cx
        mov [es:36], word INTR1
        mov [es:38], word cs

        mov bx, 8
W2:     mov cx, 0xFFFF
W1:     dec cx
        jnz W1
        dec bx
        jnz W2

        call DISP_CRED_MENU

        ;; When using interrupts use the following instruction (jmp $) to sit in a busy loop, turn on
interrupts before that
        sti
        jmp $

INTR1:
                mov DX, KBD_DAT
                in AL, DX
                ;; Translate raw key press byte contained in AL to ASCII character
                and AL, 00011111b ; Remove first three bits
                mov BX, key_table ; Move key table address location to BX
                XLATB ; Translate

                mov [char],al

                cmp byte[state], CRED_MENU
```

```
                jne IR_ACC
                call DISP_IR_MENU
                mov byte[state], IR_ACCEPT
                jmp EXIT
IR_ACC:         cmp byte[state], IR_ACCEPT
                jne VAL
                call DISP_VAL
                mov byte[state], ENTR_VAL
                jmp EXIT
VAL:    cmp byte[state], ENTR_VAL
                jne TERM
                call DISP_TERM
                mov byte[state], ENTR_TERM
                jmp EXIT
TERM: cmp byte[state], ENTR_TERM
                jne DATE
                call DISP_CRED_MENU
                mov byte[state], ENTR_DATE
                jmp EXIT
DATE: cmp byte[state], ENTR_DATE
                jne PURP
                call DISP_DATE
                mov byte[state], ENTR_PURP
                jmp EXIT
PURP:  cmp byte[state], ENTR_PURP
                jne FINAL_
                call DISP_PURP
                mov byte[state], FINAL
                jmp EXIT
FINAL_:         cmp byte[state], FINAL
                jne EXIT
                call DISP_FINAL
                mov byte[state], CRED_MENU
EXIT:

                mov AL, 0x20
                mov dx, INT_A0
                out dx, AL
                iret

KEYPAD_INIT:
;       pusha
        push ax
        push bx
```

```
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, welcome
        mov cx, len1
        mov dx, 0
        mov si, ds
        int 10H
        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
        ret

DISP_PURP:
;       pusha
        push ax
        push bx
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, purp_menu
        mov cx, purp_len
        mov dx, 0
        mov si, ds
        int 10H
        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
        ret

DISP_DATE:
;       pusha
```

```
        push ax
        push bx
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, date_menu
        mov cx, date_len
        mov dx, 0
        mov si, ds
        int 10H
        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
        ret

DISP_TERM:
;       pusha
        push ax
        push bx
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, term_menu
        mov cx, term_len
        mov dx, 0
        mov si, ds
        int 10H
        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
        ret
```

```
DISP_VAL:
;       pusha
        push ax
        push bx
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, val_menu
        mov cx, val_len
        mov dx, 0
        mov si, ds
        int 10H
        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
        ret

DISP_CRED_MENU:
;       pusha
        push ax
        push bx
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, crd_menu
        mov cx, crd_len
        mov dx, 0
        mov si, ds
        int 10H
        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
```

```
        ret

DISP_IR_MENU:
;       pusha
        push ax
        push bx
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, ir_menu
        mov cx, ir_len
        mov dx, 0
        mov si, ds
        int 10H
        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
        ret

CALC_STATS:
        push ax
        push bx
        push cx
        push dx
        push si
        pushf

        ;caculation
        mov al, [term]
        mov bx, [credit]
        sbb al  ;assume they give you 1-4
        mov cl, 5
        mul al, cl
        add ax, bx


        ;get index
        mov bx, array1D
```

```
add ax, bx
mov al, [ax]

;actual loan amount
mov cl, [loan]
mov bx, al
shr ax, bx
mov [mP], ax

mov al, [value]
mov bl, 175
mul bl
mov bl, 10000
div bl
mov bl, [mP]
add al, bl
mov [realMP], al

mov al, [mP]
mov bl, [term]
mul bl
mov bx, [value]
sub ax, bx
mov [intPaid], ax

mov al, [value]
mov bl, 125
mul bl
mov bl, 10000
div bl
mov [taxPaid], al

mov al, [value]
mov bl, 5
mul bl
mov bl, 10000
div bl
mov [mortIns], al

mov al, [term]
mov [numPay], al

mov al, [date]
mov bl, [term]
```

```
        add al, bl
        mov [payDate], al

        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
        ret

DISP_FINAL:
;       pusha
        push ax
        push bx
        push cx
        push dx
        push si
        pushf
        mov ax, 0
        mov bx, final_disp
        mov cx, final_len
        mov dx, 0
        mov si, ds
        int 10H

        mov ax, [realMP]
        mov [number], ax
        call TOSTRING
        mov cx, bx
        lea si, strRes
        ;more stuff

        popf
        pop si
        pop dx
        pop cx
        pop bx
        pop ax
;       popa
        ret

TOSTRING:
        mov ax, [number]
```

```
        mov cx, 10
        xor bx,bx

DIVIDE:
        xor dx, dx
        div cx
        push dx
        inc bx
        test ax, ax
        jnz DIVIDE

        ret

        ; Setup up interrupt vector table
        ; ------------------------------
        ;;; EXAMPLE, take this code out when not using interrupts. This loads two interrupt vectors at
vectors 0x09 and 0x0A

   ;mov bx, 8H * 4              ;Interrupt vector table 0x08 base address
   ;mov cx, INTR1               ;INTR1 service routine
   ;mov [es:bx+4], cx           ;offset
   ;mov [es:bx+6], cs           ;current code segment
   ;mov cx, INTR2               ;INTR2 service routine
   ;mov [es:bx+8], cx           ;offset
   ;mov [es:bx+10], cs          ;segment

; ---------------------------------------------------------------------------------------------------------------------
;

; RAM data section, align at 16 byte boundary, 16-bit
; ---------------------------------------------------------------------------------------------------------------------
;;; THIS IS YOUR DATA SEGMENT

section CONSTSEG USE16 ALIGN=16 CLASS=CONST

        welcome:        db " Welcome to CMPE 310"    ; 20 characters per line
                                db "     Fall '13       "
                                db "     Final Lab      "
                                db " Mortgage Calculator"
        len1:           equ $ - welcome

        crd_menu:       db "1 - Excellent      "
                                db "3 - Fair           "
                                db "2 - Good           "
```

```
                        db "4 - Poor          "
crd_len:        equ $ - crd_menu

ir_menu:        db "    Rate:         "
                        db "     Continue?     "
                        db "                   "
                        db "   1-Yes    2-No    "
ir_len:         equu $ - ir_menu

val_menu:       db "Enter Value:       "
                        db "                   "
                        db "                   "
                        db "                   "
val_len:equ $ - val_menu

term_menu:      db "Enter Term:        "
                        db "                   "
                        db "                   "
                        db "                   "
term_len:       equ $ - term_menu

date_menu:      db "Enter Date:        "
                        db "                   "
                        db "                   "
                        db "                   "
date_len:       equ $ - date_menu

purp_menu:      db "Purpose?          "
                        db "2-Refinance        "
                        db "1-New Purchase     "
                        db "                   "
purp_len:       equ $ - purp_menu

final_disp:     db "RealMP:           "
                        db "IntPaid:          "
                        db "TaxPaid:          "
                        db "Payments:         "
final_len:      equ $ - final_disp

state:          db 0
key_table:      db "01231XXX45672XXX89AB3XXXCDEF4" ; look up table
char:           db 0
array1D     db 3,3,3,2,4,3,3,2,4,4,3,2,5,4,3,2    ; 4 Rows by 4 Columns
date:           db 0
```

```
term:       db 0
credit:     db 0
value:      db 0
loan:       db 0
mP:             db 0
realMP:     db 0
intPaid: dw 0
taxPaid:    db 0
mortIns:    db 0
numPay:         db 0
payDate:    db 0
strRes:     db 16, dup (0)
number:         db 0
```