Princess Sumaya University for Technology

King Abdullah II Faculty of Engineering

Electrical Engineering Department

Princess Sumaya University for Technology جامعة الأميرة سميّة للتكنولوجيا

MICROPROCESSOR MICROPROCESSORS MIDTERM PROJECT HARDWARE

Authors:

Nedal Abu-Haltam	20210125	Computer Engineering
Abdallah Al-Omari	20191041	Electronics Engineering

Supervisor:

Dr. E. Al-Qaralleh

January 15, 2024

Abstract

Designing a hardware requires taking into consideration many aspects. Including the hardware connections, system compatibility between the different parts of the system, and of course the software that runs on the system. Thus, there are tools and environments that are available to the designers to make sure that the system works properly. In this report, we tried to design a system that reads from a 4x4 keypad and display can display it on 16x2 LCD display and a memory to store different values and to serve the transmitter and the receiver in the transmission operation. We used the 8086 microprocessor to act as the heart of the system that controls the inputs and the outputs signal between the different IO/M devices.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 OBJECTIVES

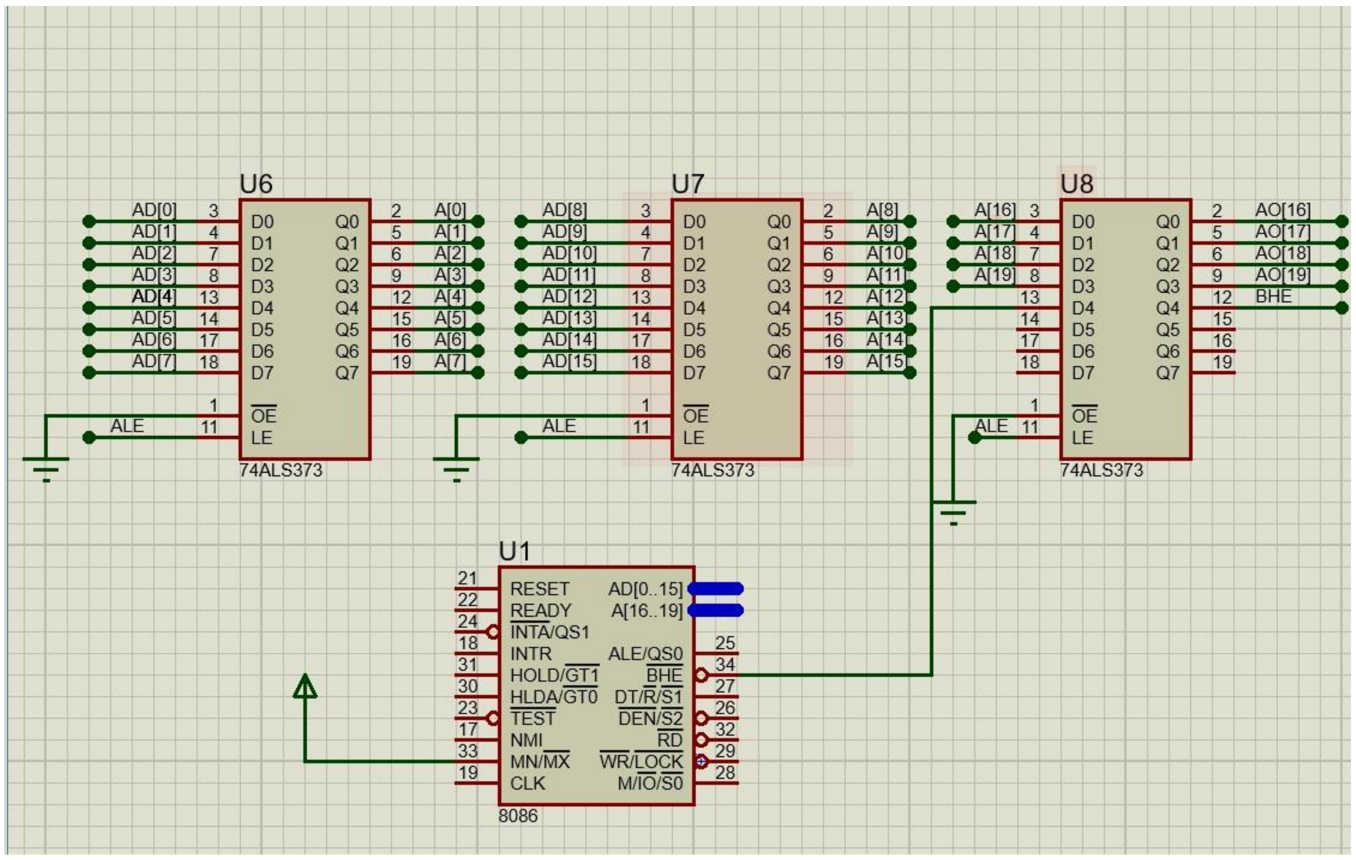
In this project we aim to design a system the uses a 4x4 keypad as an input device and to process it and display it on a 16x2 LCD display. The system should provide several RAM's based on the requirements, a ROM to save the BIOS of the 8086 microprocessor, a parallel to serial device (i.e. the 8051 UART), and the 8255 controller to organize between the inputs coming to and the outputs coming from the 8086. And finally, a software that is compatible with our design to initialize the components to be ready to operate correctly.

2 PROCEDURE AND METHODS

In this section we will go through the steps that we took during our designing process. And mentioning the details regarding that. We shall start with the hardware part of the design. And then the software part.

2.1 HARDWARE DESIGN

1- The microprocessor, it is the unit that controls the whole operation and talks to all of the different parts of the system. So, the addresses and the data buses that come out from the processor should be carefully taken from it. We took the AD[0:15] bus and connect them to the necessary laches to latch them to an address bus A[0:15] using the ALE signal and A[16:19] to another latch with BHE' to complete the 20-bit address lines from the processor. And we took the AD[0:15] and leave them as the 16-bit data bus. And of course we took the rest of the necessary signals to the system like the RD, WR, IO'/M to use them later on, etc.



U6
U7
U8
U1
74ALS373
8086
AD[0] AD[1] AD[2] AD[3] AD[4] AD[5] AD[6] AD[7]
A[0] A[1] A[2] A[3] A[4] A[5] A[6] A[7]
AD[8] AD[9] AD[10] AD[11] AD[12] AD[13] AD[14] AD[15]
A[8] A[9] A[10] A[11] A[12] A[13] A[14] A[15]
A[16] A[17] A[18] A[19]
AO[16] AO[17] AO[18] AO[19] BHE
ALE
D0 D1 D2 D3 D4 D5 D6 D7
Q0 Q1 Q2 Q3 Q4 Q5 Q6 Q7
OE
LE
RESET READY INTA/QS1 INTR HOLD/GT1 HLDA/GT0 TEST NMI MN/MX CLK
AD[0..15] A[16..19] ALE/QS0 BHE DT/R/S1 DEN/S2 RD WR/LOCK M/IO/S0

Figure 1: Latching the addresses of the microprocessor and preparing other signals.

2- The parallel to serial transceiver (8251), Is very important for external devices to communicate in different manner so we have to provide it for any later use. And this allows us to scale our design and make it compatible with different modes of operation and ecosystems. The device takes an input 8-bits parallel of data and outputs 8-bits serially to the receiver. As a port (i.e. the decoding circuit) we choose a suitable one to use it in the software for initializing it. For the CTS' and DSR', we treated them as a dummy modem so it works all the time.

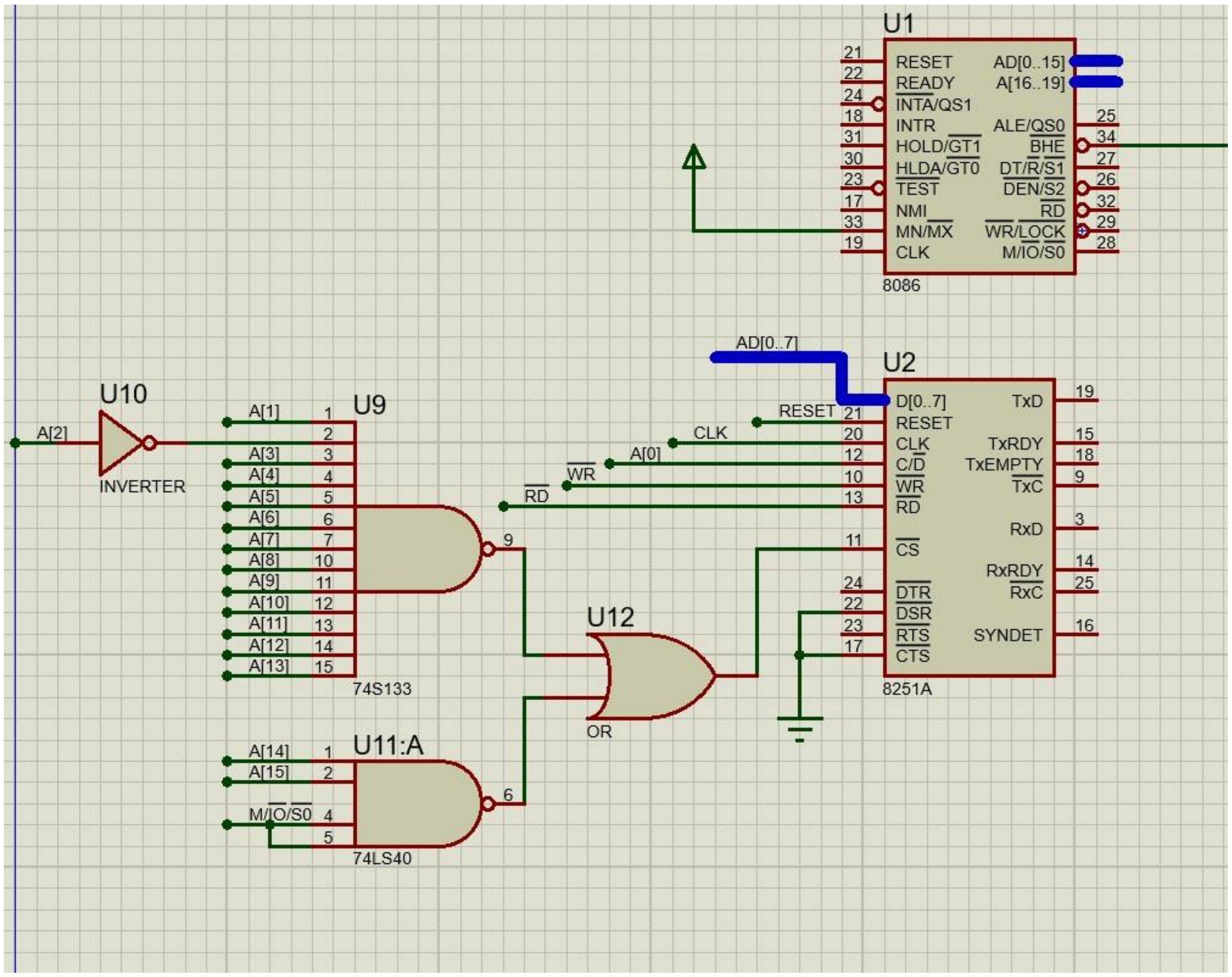


Figure 2: 8251 parallel to serial interface.

3- The 8255 controller is very essential to our system, as if it is the separator layer between the microprocessor and IO devices. We choose a decoding circuit and connect A0 and A1, to make it easy to deal with in the software when sending commands and data to its ports. We set Port A as an input we take from the keypad, Port B as an output to the LCD display, and Port C to be an output so we can use it for the walking zero in the keypad. And the 8-bits data are connected to the microprocessor for data to flow back and forth.

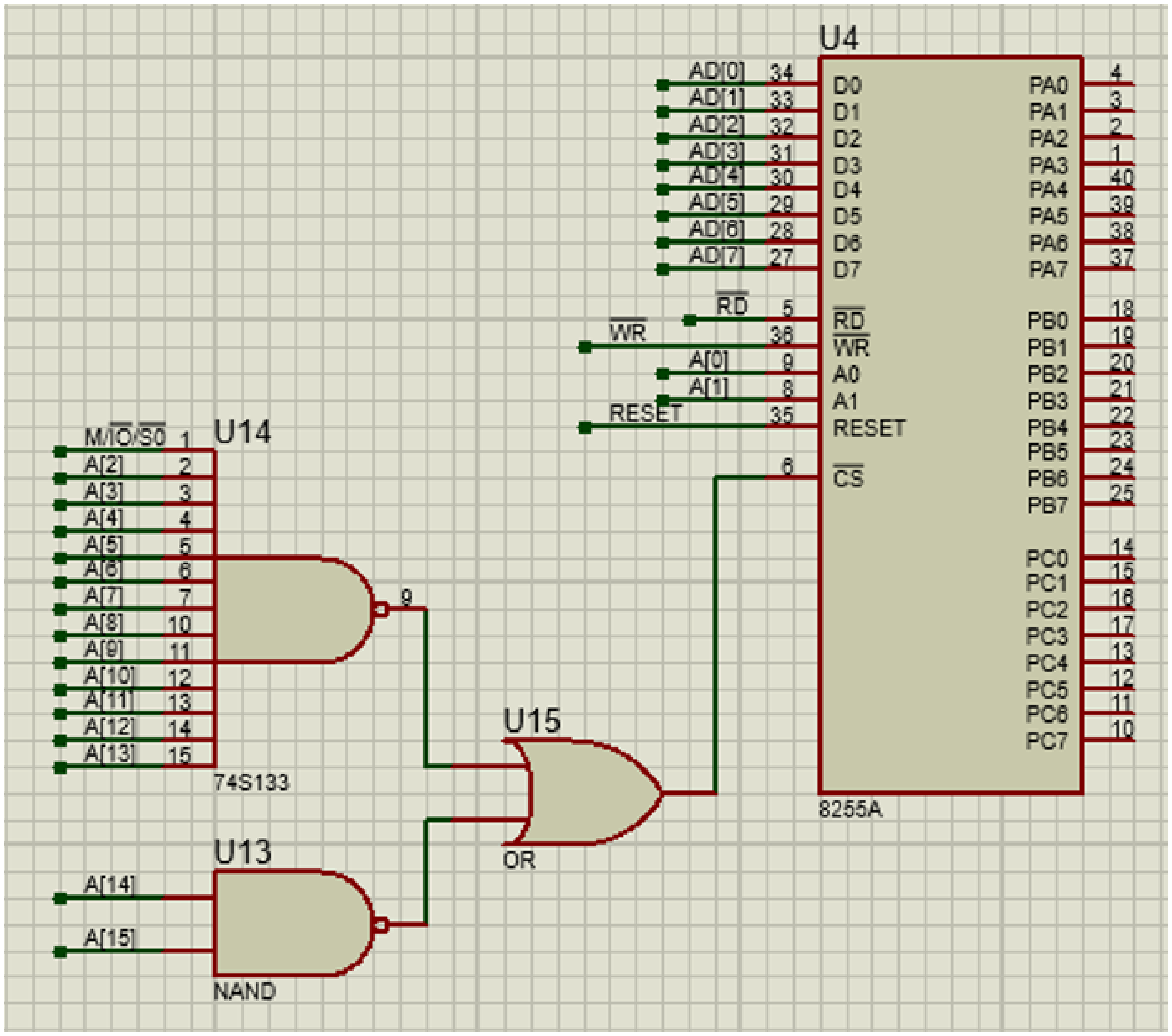


Figure 3: 8255 controller interface.

4- The LCD display is a 16x2 display that has low number of inputs, it has just 8-bits of data inputs coming from Port B of the 8255 controller, R/W' signal we decided to connect it to ground because we always want to write on the display either command or data, and the register select (RS) that separates the command bytes from the data.

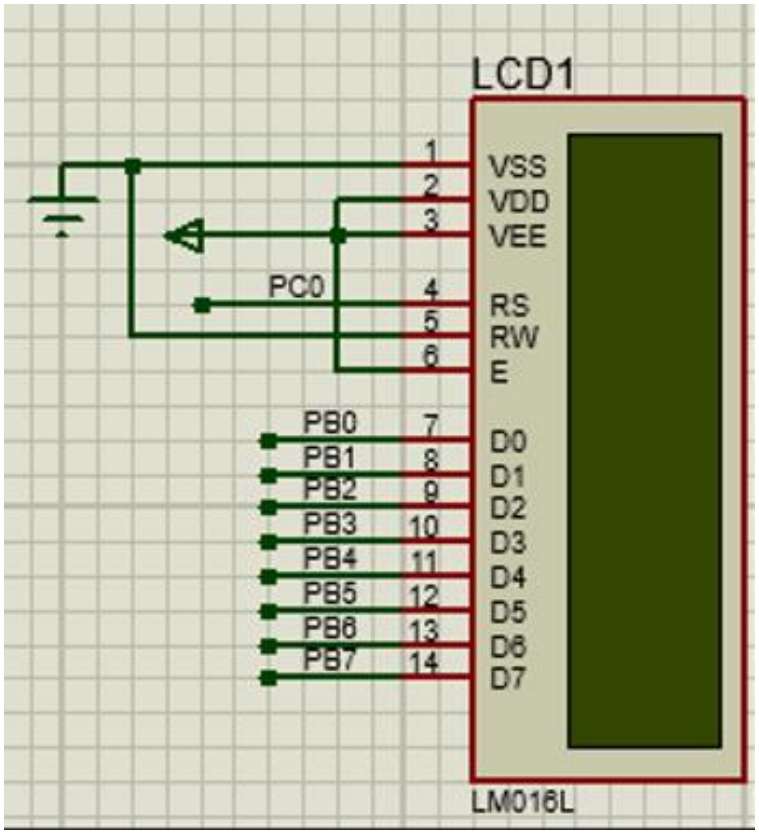


Figure 4: 16x2 LCD display connections to the 8255 IC.

5- The keypad is the input device to our system it has four pins on the bottom and these are controlled by Port C (4-7) of the 8255 controller to produce the walking zero so we can detect which column of buttons is activated. And there are four other pins on the left side and these are the inputs coming from the keypad, we connect them to Port A (0-3) se we can read it. And deal with it in the program to be able to display it on the screen.

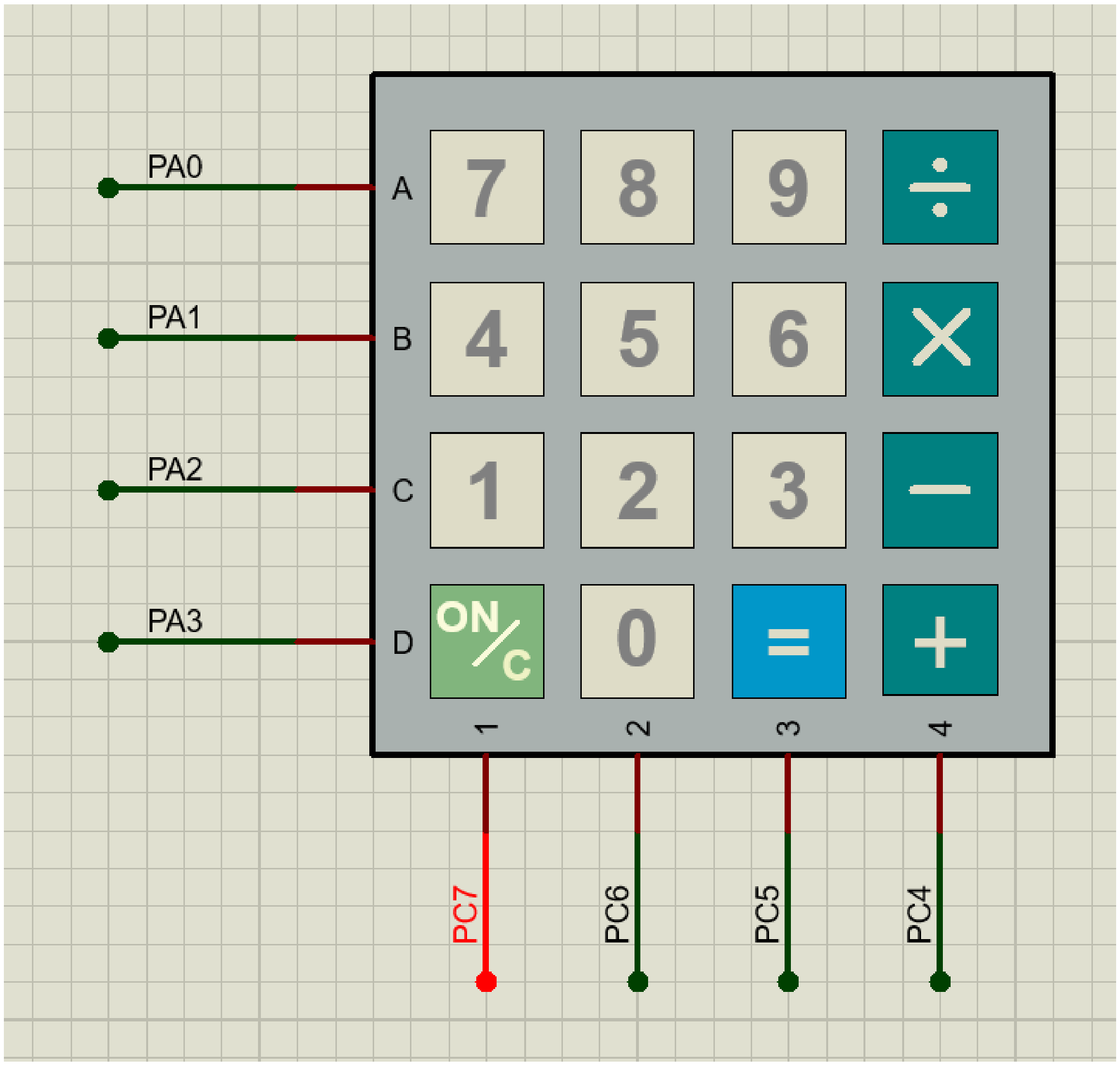


Figure 5: 4x4 Keypad interface.

6- The ROM is very simple. We know that the 8086 microprocessor saves its BIOS in a specific part of the memory which is from F000H-FFFFFH. Accordingly, we connect its decoding circuits based on this. And provide it with the rest of the necessary address lines. Taking into consideration the separation of the ROM to two 32KB, for the 8086 to read from it correctly and without any issues.

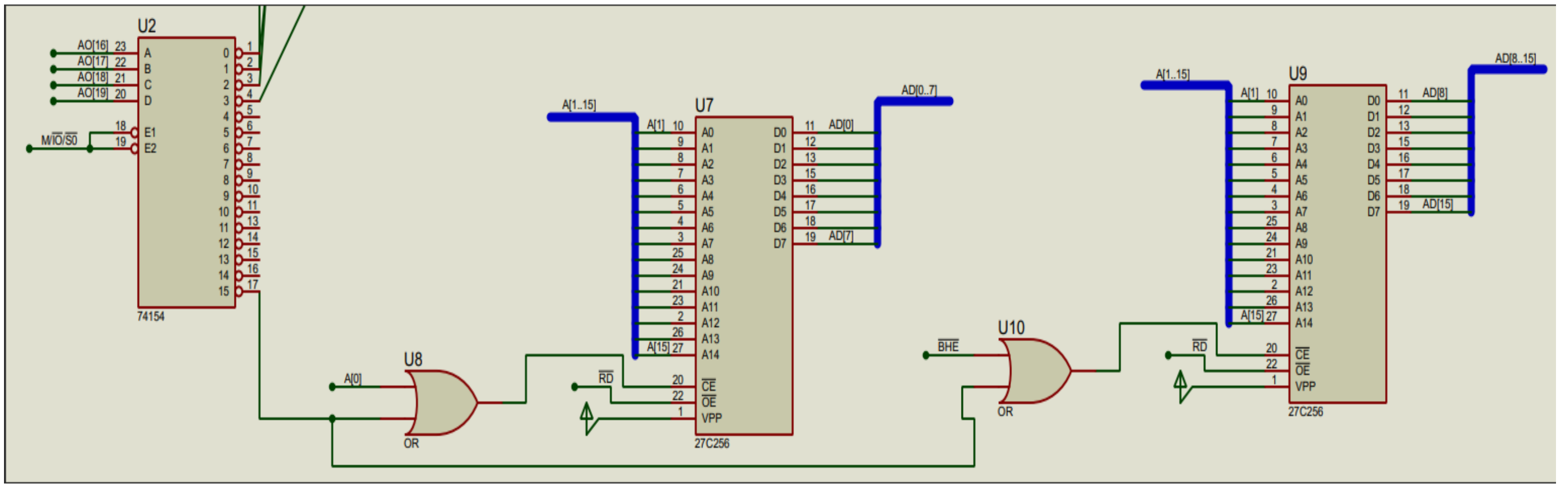


Figure 6: EPROM circuit with the decoding circuit and the data out to the microprocessor

7- Finally, the RAM part of the system and this part of the design is not very complicated. We just connect the needed address lines to the RAM, and decode the rest to the active low chip select (chip enable) of the memory based on the required addresses in the project requirements, please refer to Figure katha for the memory map of the system. In our case there are three parts of the RAM.

7.1- the first 64KB memory.

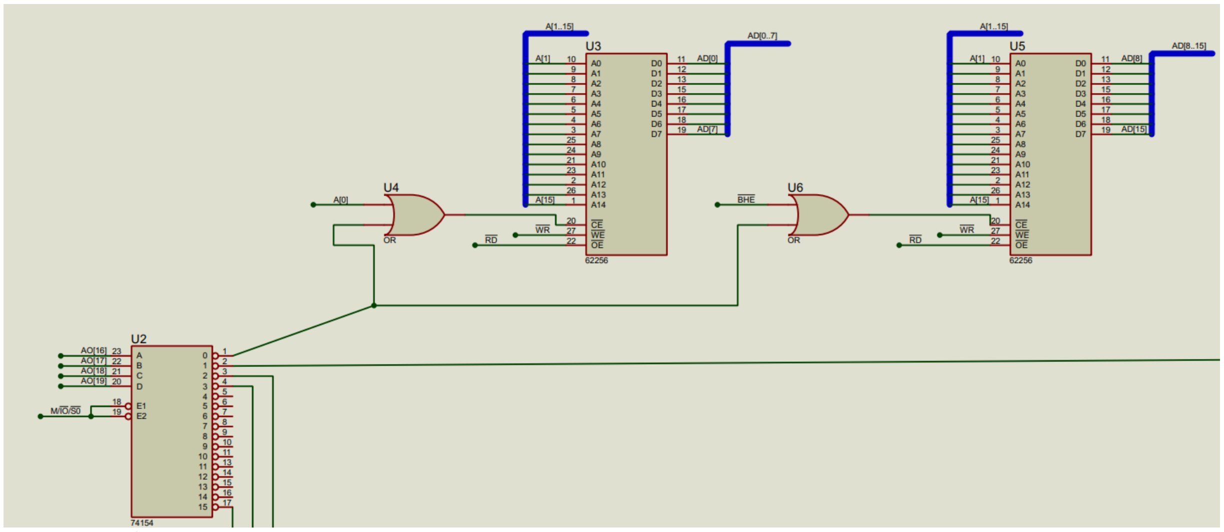


Figure 6: the 64KB

7.2- the second part is the 128KB broken into four 32KB smaller RAM's.

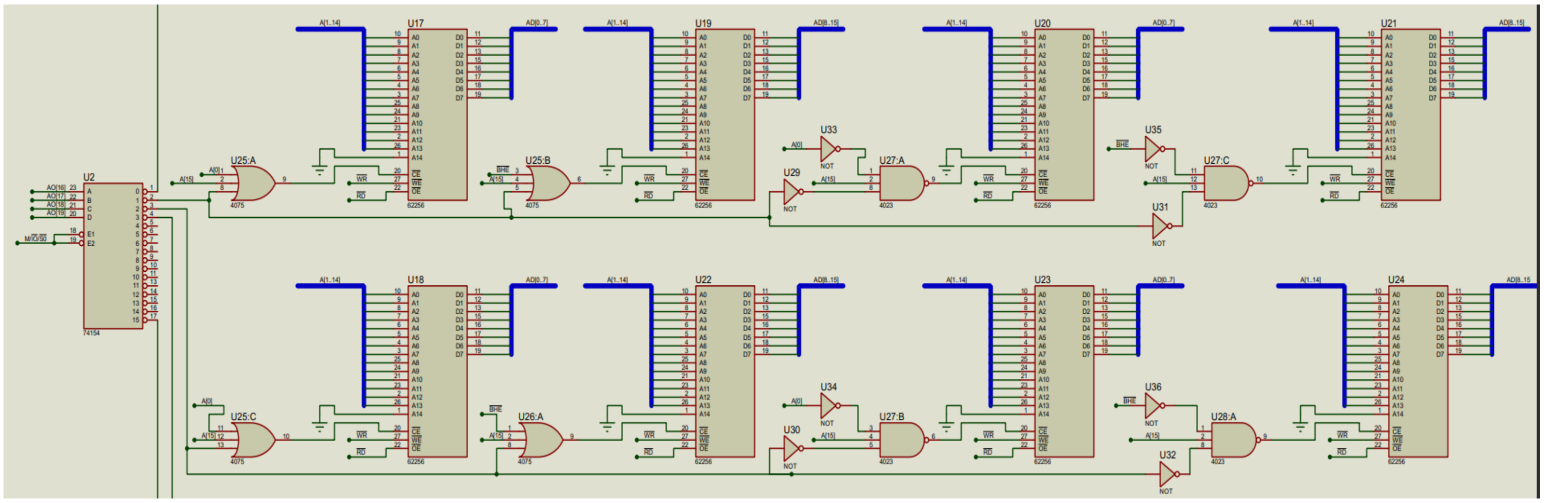


Figure 7: 128KB memory part interface.

7.3- And the last part is the 2KB memory for the transmitter and receiver feature.

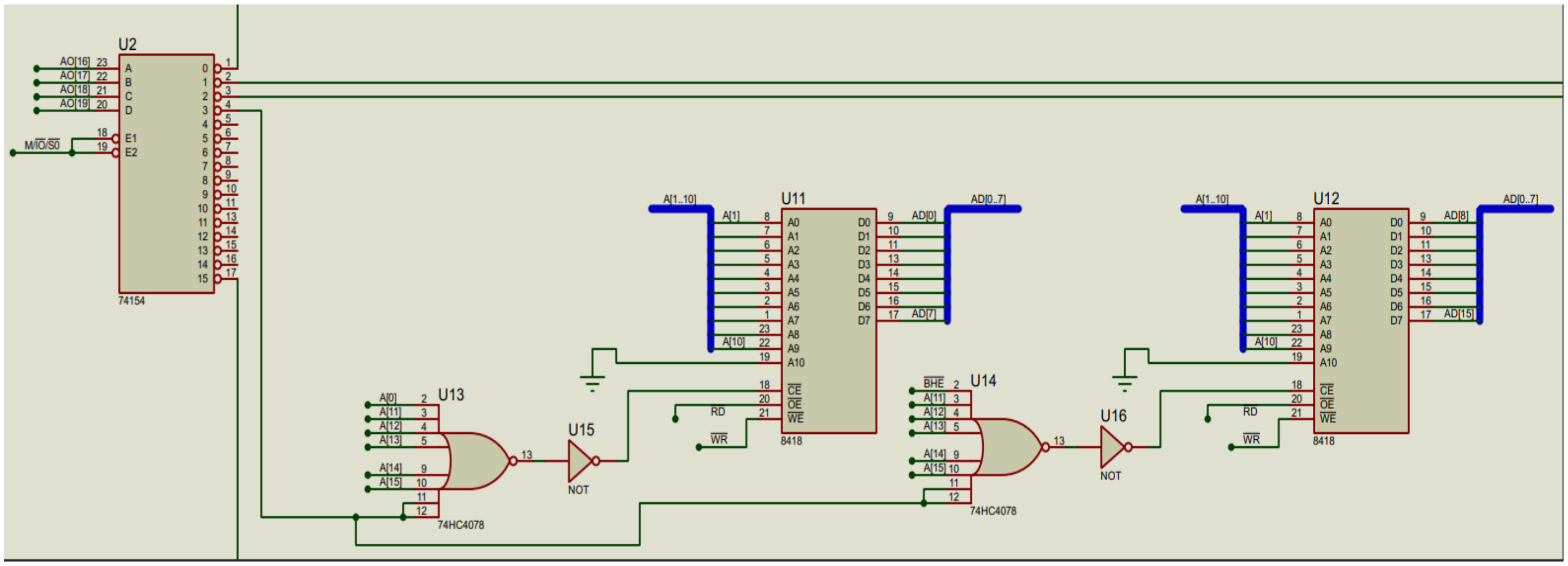
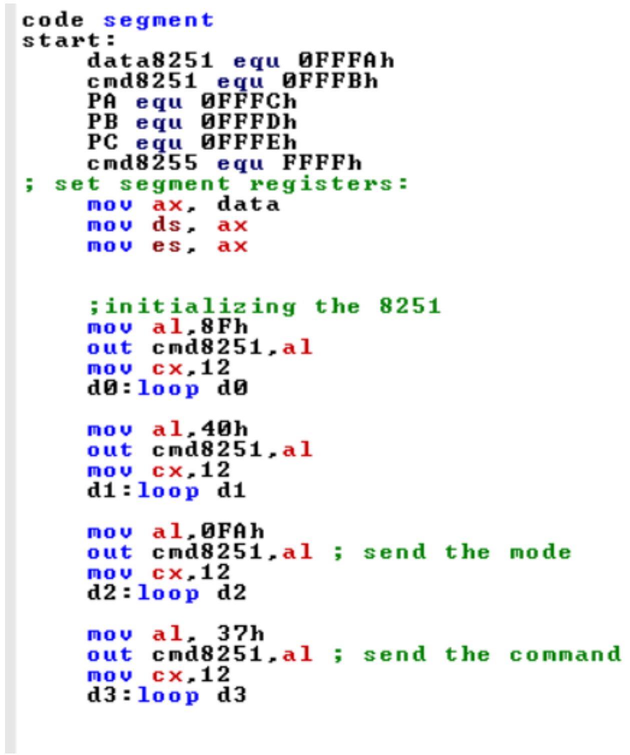


Figure 8: the 2KB RAM for transmitting and receiving data between the micro processor and the receiver.

2.2 Software Design

In the software part we wrote an assembly code to initialize the 8251 parallel to serial, the 8255 controller, and the 16x2 LCD display. So, the system can operate properly when booting up.

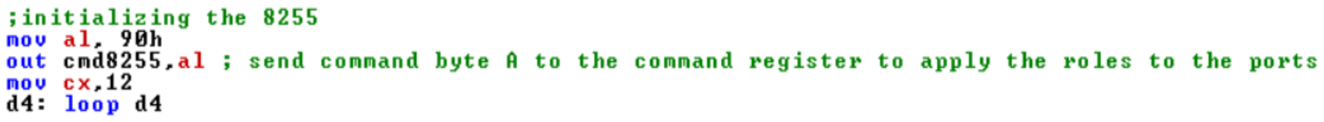
1- First of all, the 8251. Based on our hardware connections we set up the command and the data ports so we can send commands and data correctly. After that, we started by sending a command in case the 8251 was already on. And the sending a command that ensures the device to internally reset. After that, we send the desired mode of operation through the command port, and finally sending the command that contains the settings of the device to be ready to run and transmit and receive data. Notice after each out statement we follow it by a certain amount of delay so it can catch up and receive the commands one by one not to collide with each other.



```
code segment
start:
    data8251 equ 0FFFAh
    cmd8251 equ 0FFFBh
    PA equ 0FFFCh
    PB equ 0FFFDh
    PC equ 0FFFEh
    cmd8255 equ FFFFh
; set segment registers:
    mov ax, data
    mov ds, ax
    mov es, ax


    ;initializing the 8251
    mov al,8Fh
    out cmd8251,al
    mov cx,12
    d0:loop d0

    mov al,40h
    out cmd8251,al
    mov cx,12
    d1:loop d1

    mov al,0FAh
    out cmd8251,al ; send the mode
    mov cx,12
    d2:loop d2

    mov al, 37h
    out cmd8251,al ; send the command
    mov cx,12
    d3:loop d3
```

Figure 9: code that initialize the 8251 to be ready to send data.

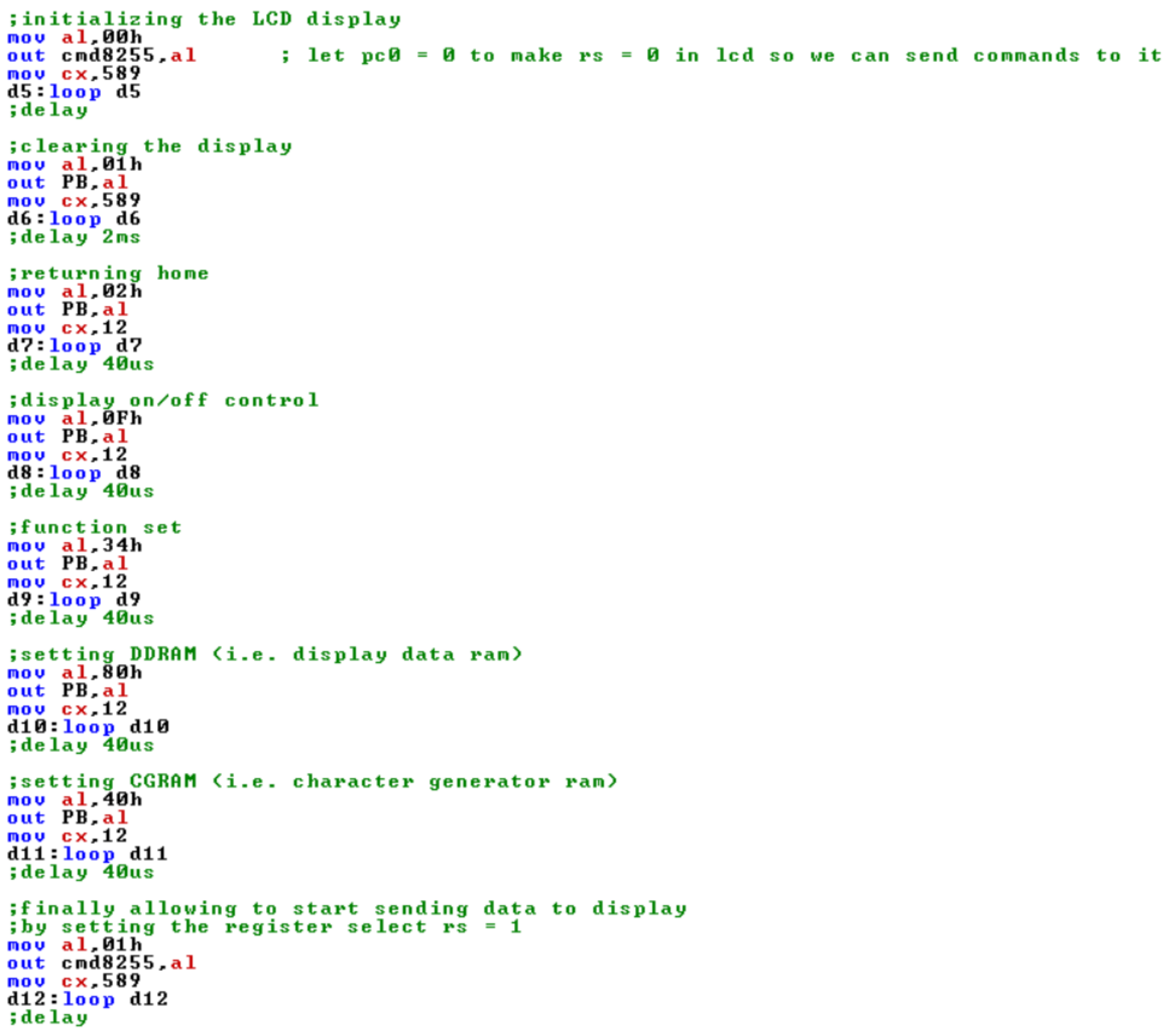
2- Initializing the 8255 controller is very simple. There is just one command byte the is called command byte A. We first program the ports using the datasheet and encode it in this byte and then send it to the command port of the controller. And follow it with a delay.



```
;initializing the 8255
mov al, 90h
out cmd8255,al ; send command byte A to the command register to apply the roles to the ports
mov cx,12
d4: loop d4
```

Figure 10: code to initialize the 8255 controller to connect processor the IO devices

3- And finally, programming the LCD display. Now based on the datasheet, there must be a certain sequence to follow when sending the command bytes to the display. We send it all of it through the 8255 controller. Firstly, we set RS to be logic zero through Port C 0 so we are allowed to send commands to it. The first command is clearing the display from any data that can hinder us through out the process. Secondly, returning home sets the DDRAM address 0 into the address counter, and returns the display to its original status if it was shifted. Controlling the display on/off, we want to turn on the display and make the cursor visible to the user. Function set, this is very critical step in setting the display because this determines data length you want to send and potentially receive in our case we want 8-bit data length. Setting the DDRAM (i.e. the display data RAM) address into the address counter, and the same thing goes for the CGRAM (character generator RAM). And finally setting the RS to be login 1 so data is allowed to be sent and displayed on the screen.



```
;initializing the LCD display
mov al,00h
out cmd8255,al       ; let pc0 = 0 to make rs = 0 in lcd so we can send commands to it
mov cx,589
d5:loop d5
;delay

;clearing the display
mov al,01h
out PB,al
mov cx,589
d6:loop d6
;delay 2ms

;returning home
mov al,02h
out PB,al
mov cx,12
d7:loop d7
;delay 40us

;display on/off control
mov al,0Fh
out PB,al
mov cx,12
d8:loop d8
;delay 40us

;function set
mov al,34h
out PB,al
mov cx,12
d9:loop d9
;delay 40us

;setting DDRAM (i.e. display data ram)
mov al,80h
out PB,al
mov cx,12
d10:loop d10
;delay 40us

;setting CGRAM (i.e. character generator ram)
mov al,40h
out PB,al
mov cx,12
d11:loop d11
;delay 40us

;finally allowing to start sending data to display
;by setting the register select rs = 1
mov al,01h
out cmd8255,al
mov cx,589
d12:loop d12
;delay
```

Figure 11: the code to reset, turn on and initialize the LCD display to receive data and display it.

3 CONCLUSIONS

In conclusion, we saw through out the designing process that there are many things should be carefully done to make sure it works and not to violate any rules when running it. And in these systems the software and the hardware should work in harmony to achieve the ultimate goal for serving the user in the best and most efficient way possible.