DESIGN AND IMPLEMENTATION OF MOTOROLA 68000 BASED μCOMPUTER

Arthur J. Miller

Aug. 22, 2016

Cal Poly Pomona

Department of Electrical and Computer Engineering

CAL POLY POMONA

DESIGN AND IMPLEMENTATION OF MOTOROLA 68000 BASED μCOMPUTER

Arthur J. Miller

Cal Poly Pomona
College of Engineering
3801 W Temple Ave
Pomona, CA 91768

Aug. 22, 2016

This report outlines the project for Dr Rafi's ECE 343L completed by Arthur J. Miller Summer of 2016.

Abstract

During this project a MC68000 based μComputer was designed and implemented. The CPU is connected to two 2732 EPROM chips as well as a MC6821 Input/Output chip. The MC68000 does not have an internal reset circuit so an external reset circuit was designed and implemented as well. The MC68000 is clocked by a 4Mhz TTL crystal oscillator. The prototype circuit is wire wrapped on a protoboard. Cadsoft Eagle was used to design the permanent PCB. Several assembly language programs were burned on to the EPROM chips to test the completed circuit.

Contents

List of Figures

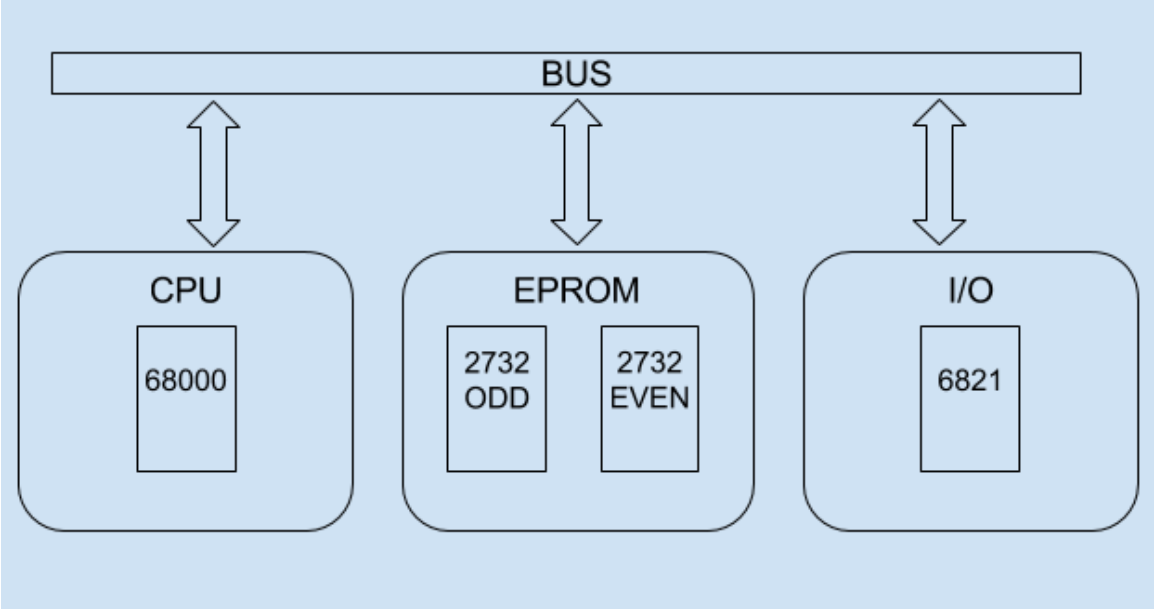
List of Tables

1 Introduction

At its core a μComputer must have a CPU, ROM, and I/O. The CPU is the brain of the computer and the heart of the design. Careful consideration must be taken in the design process to choosing a CPU. A μComputer must be able to run a user defined program - this is why ROM is needed. The designer must take into consideration whether the user of the computer will need to reprogram the device. The final component that is needed for a bear minimum μComputer is Input/Output capabilities. The CPU must be able to communicate with the outside environment. Which may be a keyboard, a display, a sensor, a led, or any other components as needed by the end user. The following sections will describe the components used in the design as well as the reasons for choosing these specific components.

2 Project Overview

2.1 System Block Diagram



2.2 Project Timeline

TABLE 1 Timeline

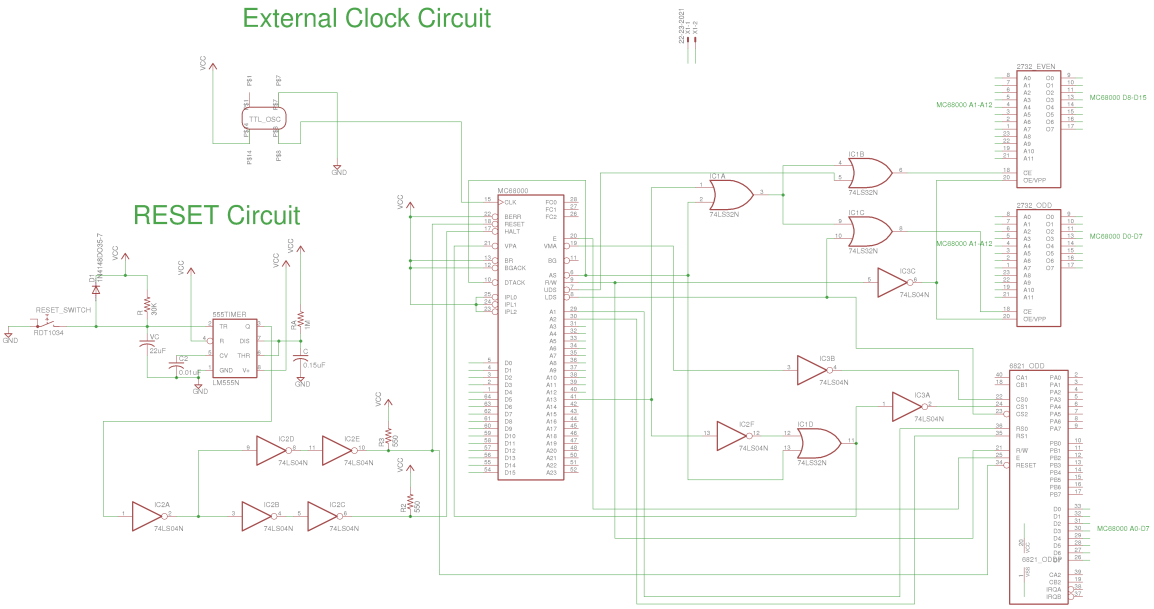
Week 1	N/A
Week 2	Download and install 68K IDE Assmbler
Week 3	Purchase parts, test 68K assembly programs
Week 4	Start reset circuit, parts received, more test 68K assembly programs
Week 5	Demo reset circuit, start wire-wrapping
Week 6	Wire-wrapping
Week 7	Finish wire-wrapping board, Test Continutity, Received PCB
Week 8	Demo Lab1, begin Lab2
Week 9	Demo Lab2, begin writing project report
Week 10	Finish writing project report

3 Hardware

The schematics for this class were typed using Cadsoft Eagle. This allowed for the schematic to be used for prototyping as well as for designing the PCB.

3.1 Full Schematic

Figure 1: Full MC68000 Based μController

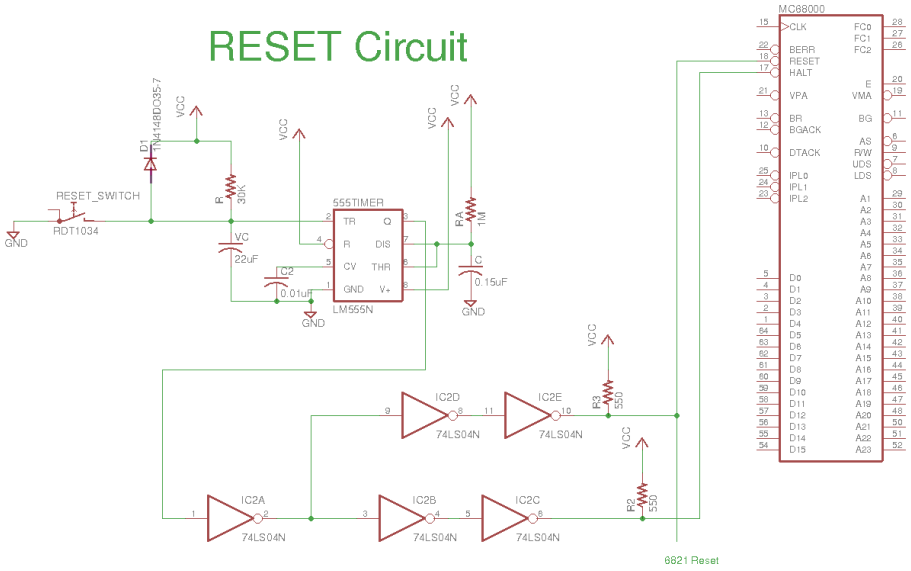


3.2 External Reset Circuit

The 68000 μProcessor does not have an internal reset circuit. The figure below shows the external reset circuits schematic connected with the 68000 and 6821 chips. The bullet points below highlight design features and implementation notes.

- We chose to design for the 100ms requirement for the start-up reset. Because the manual reset has a much smaller time requirement- thus the start-up reset fulfills both requirements.
- The implemented reset low period was 410ms. This value was increased from 350ms by increasing the pull up resistor to 10K Ω.

Figure 2: 68K to External Reset Schematic



3.3 2732 EPROM

Two 2732 EPROMS's were used for the program and data memory for the μController. One chip was used for the even bytes of memory and the other was used for the odd bytes. Several design choices are laid out in the following bullet points.

- For our design when the A_{13} pin was low that enabled the EPROM's.
- The even/odd chips were selected by the sign bit A_0. A_0 was encoded by the $\overline{UDS}$ pin for even bytes and $\overline{LDS}$ for odd bytes.

3.3.1 Memory Map

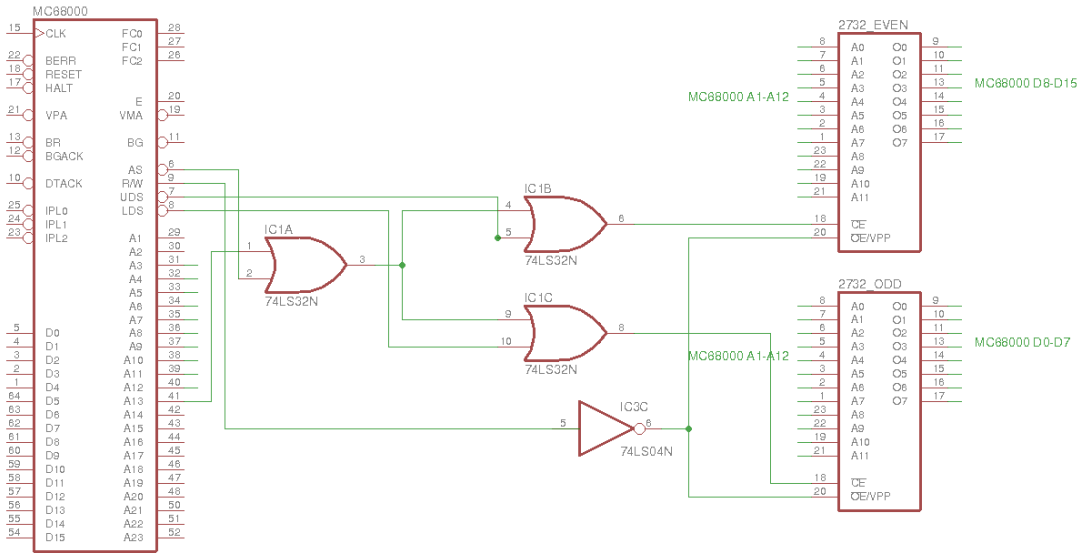
These design choices created the following memory map:

A23-A14	A13	A12-A1	A0
Don't Cares (0's)	0	$000 - $FFF	0:Even 1:ODD

Table 2: 68000 to 2732 Memory Map

- Even: $000000,$000002,$000004,..., $001FFE
- Odd: $000001,$000003,$000005, ..., $001FFF

Figure 3: 68K to 2732 Schematic



3.4 MC6821 I/O

The MC6821 is a slow and cheap input/output device compatible with the 68000 μProcessor. Several design notes were taken into consideration.

- An even and odd 6821 can be used which is encoded by the $\overline{UDS}$ and $\overline{LDS}$ pins. Our design only needed one chip so we chose it to be the odd chip.
- The 6821 is approximately 10 times slower than the 68000. The 68000 uses several dedicated input/output pins to account for the slower clock speed of the 6821. The input pin $\overline{VPA}$, output pin $\overline{VMA}$, and input pin $\overline{DTACK}$
- The 6821 I/O Ports (PORTA,PORTB) are controled via the Control Register (CRA,CRB) and Data Direction Register (DDRA,DDRB) for each port.

3.4.1 I/O Memory Map

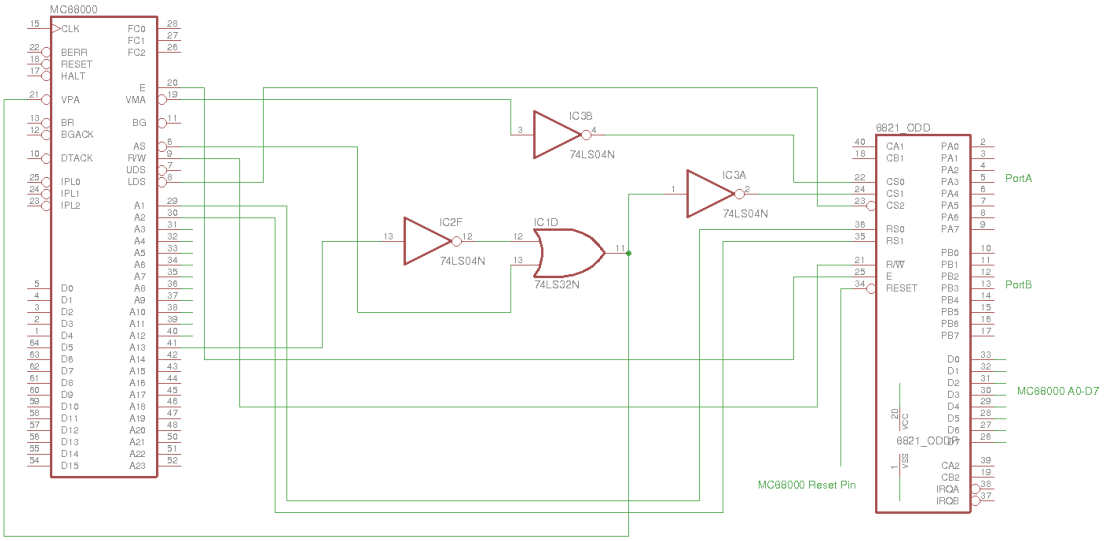
A23-A14	A13	A12-A3	A2A1	A0
Don't Cares (0's)	1	Don't Cares (0's)	00_2-11_2	0

Table 3: 68000 to 6821 Memory Map

- PortA or DDRA: $002000
- PortB or DDRA: $002004

- CRA: $002002
- CRB: $002006

Figure 4: 68K to 6821 Schematic



4 Software

4.1 Test Program 1

The purpose of Lab 1 was to test our wire wrapped circuit. It was a simple LED toggle program.

```
            ORG         $0
            DC.L        $506080         ;A7' init addr. Arbitrary (We aren't using)
            DC.L        $000100         ;PC init addr. MC68000=$0001000,2732=$080

CRA         EQU         $002003         ; A13=1(I/O),A2A1=01(CRA),A0=1(ODD)
DDRA        EQU         $002001         ; A13=1(I/O),A2A1=00(PORTA/DDRA),A0=1(ODD)
PORTA       EQU         DDRA
            ORG         $000100         ; As defined in PC init
BACK        BCLR.B      #2,CRA          ; CRA(bit2)=0: Access DDRA Register
            MOVE.B      #2,DDRA         ; bit1 = Output, bit0 = input
            BSET.B      #2,CRA          ; CRA(bit2)=1: Access PORTA Register
            MOVE.B      PORTA,D0        ; [D0]=input switch
            LSL.B       #1,D0           ; Allign for LED
            MOVE.B      D0,PORTA        ; Change output of LED
            BRA         BACK
FINISH      JMP         FINISH
```

2732 Addr	Even Byte	Odd Byte
000	00	50
002	60	80
004	00	00
006	01	00
100	08	B8
102	00	02
104	20	03
106	11	FC
108	00	02
10A	08	F8
10C	00	02
10E	20	03
110	10	38
112	20	01
114	E3	08
116	11	C0
118	20	01
11A	60	E2
11C	4E	F8
11E	01	1E

Table 4: Program 1 Byte Code

4.2 Test Program 2

For Lab2 a 4-bit binary input was used to control an external seven segment display. PORTA was connected to four external switches that are used as the 4-bit binary input. PORTB was used as an output connected to a common cathode seven segment display. This means the seven segment is grounded and a high input to a-g will turn on a segment. A single 74LS04 inverter was used to amplify the current and add the needed resistance before the seven segment display. Because of this the lookup table for the seven segment was for a common anode. After being inverted it will be the correct HIGH/LOW logic for a common cathode.

```
            ORG         $0              ; Reset Vector
            DC.L        $506080         ; A7' init addr. Arbitrary (We aren't using)
            DC.L        $000200         ; PC init addr. MC68000=$000200,2732=$100
CRA         EQU         $002003         ; A13=1(I/O),A2A1=01(CRA),A0=1(ODD)
CRB             EQU         $002007
DDRA        EQU         $002001         ; A13=1(I/O),A2A1=00(PORTA/DDRA),A0=1(ODD)
DDRB        EQU         $002005
PORTA       EQU         DDRA
PORTB       EQU         DDRB
            ORG         $000100         ;addr = 2732 addr
SEG         DC.B        $80,$F8,$48,$60,$32,$24,$04,$F0,$00,$20,
$00,$00,$00,$00,$00,$00 ;Common Anode Lookup -> Inverter -> Common Cathod Hardware
```

```
        ORG     $000200
LOOP    BCLR.B  #2,CRA          ; CRA(bit2)=0: Access DDRA Register
        MOVE.B  #$F0,DDRA       ; INPUTS: PORTA bits 0-3
        BCLR.B  #2,CRB          ; CRA(bit2)=0: Access DDRB Register
        MOVE.B  #$FF,DDRB       ; OUTPUTS: PORTAB bits 1-7
        BSET.B  #2,CRA          ; CRA(bit2)=1: Access PORTA Register
        MOVE.B  PORTA,D0        ; [D0]=input switch 0-15
        ANDI.W  #$000F,D0       ; mask
        BSET.B  #2,CRB          ; CRA(bit2)=1: Access PORTB Register
        LEA.L   $100,A0         ; Lookup table offset
        MOVE.B  (A0,D0.W),D1    ; [OFFESET+SEG]=seg outputs
        MOVE.B  D1,PORTB        ; Outputs
        BRA     LOOP
```

2732 Addr	Even Byte	Odd Byte
000	00	50
002	60	80
004	00	00
006	02	00
100	80	F8
102	48	60
104	32	24
106	04	F0
108	00	20
10A	00	00
10C	00	00
10E	00	00
200	08	B8
202	00	02
204	20	03
206	11	FC
208	00	F0
20A	20	01
20C	08	B8
20E	00	02
210	20	07
212	11	FC
214	00	FF
216	20	05
218	08	F8
21A	00	02
21C	20	03
21E	10	38
220	20	01
222	02	40
224	00	0F
226	08	F8
228	00	02
22A	20	07
22C	41	F8
22E	01	00
230	12	30
232	00	00
234	11	C1
236	20	05
238	60	C6

Table 5: Program 2 Byte Code

5 Side Projects

Several side projects were attempted along side the lab's stated project goals.

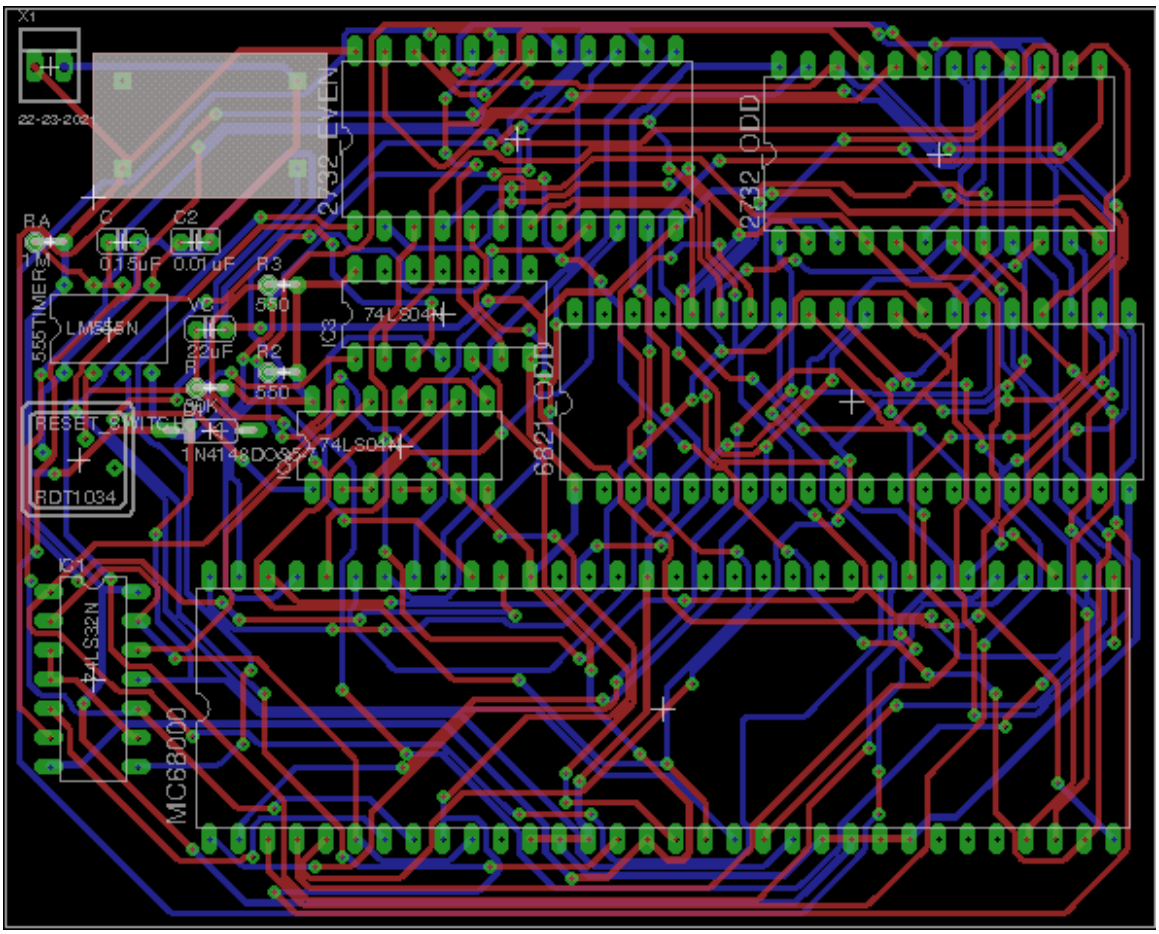
5.1 PCB Design

Wire wrapping is a good prototyping option, but creating a PCB for the circuit is a much more scalable option. CadSoft Eagle was used to design a PCB for the complete circuit. This was the first PCB I have designed but following several online tutorials a PCB for the circuit was successfully built. The design process for the PCB using Eagle was as follows:

1. Design Schematic
2. Design Board Layout
   (a) Place and organize components
   (b) Create custom part if needed (I had to do that for the TTL Oscilator)
   (c) Route or Auto Route the layout
3. Run ERC/DRC and fix any errors
4. Export Gerber files for manufacture
5. Send files to manufacture
   (a) I used Bay Area Circuits ($50 for 3 boards and recieved within 5 business Days)

The PCB design was completed and sent to the manufacture. There was not enough time to solder in the parts and verify the boards yet.

Figure 5: PCB Layout



5.2 HEX File Converter for 68K to 2732

Our design splits the memory map into even and odd bytes. This means that after the hex code is assembled ,starting at the first bit, every other eight bits needs to be grouped into a separate file. One file being all even bytes and the other being all odd bytes. It appears that the assembler being used did not have the option to output an even or odd hex file. So I decided to write my own Python program to convert the file into even and odd bytes. The main reason for this is because manually typing in the bytes into the EPROM burner is not scalable for larger programs. It also is an error prone method. In the end it was a wasted effort because the EPROM reader actually has an input option to read in the even or odd bytes. Non the less this showed me how a HEX file is encoded - specifically the Motorola S-Files.

```
# Function to convert Motorola HEX file into two Even/Odd Byte HEX files
def main():
        fname = "foo.txt" # Input Hex file name
        OutputEven = "Even.bin" # Output Even Bytes filename
        OutputOdd = "Odd.bin" # Output Odd Bytes Filename
```

```
        counter = 0 # even odd counter
        Even = [] # List stores Even Bytes
        Odd = [] # List stores Odd Bytes
        n = 8 # 1 Byte = 8 bits

        # Open Hex file
        # Read Line of Hex data and convert to Even, Odd Bytes
        with open(fname, "rb+") as f:
                for line in f:
                        print line
                        # Convert Hex string to Binary String
                        scale = 16 ## equals to hexadecimal
                        num_of_bits = 8
                        lineBin = bin(int(line, scale))[2:].zfill(num_of_bits)
                        # Convert into List of Bytes in Binary
                        Bytes = [lineBin[i:i+n] for i in range(0, len(lineBin), n)]
                        BytesStr = ' '.join(Bytes)
                        # Create Even and Odd list of binary bytes
                        for Byte in BytesStr.split():
                                if(counter % 2 == 0):
                                        Even.append(Byte)
                                else:
                                        Odd.append(Byte)
                                counter += 1
        # Close inputed hex file
        f.close()

        # Delete Even.hex && Odd.hex if exists
        try:
                os.remove(OutputEven)
        except OSError:
                pass
        try:
                os.remove(OutputOdd)
        except OSError:
                pass

        # Convert lists to strings
        EvenStr = ''.join(Even)
        OddStr = ''.join(Odd)

    # Write Even && Odd output files
        with open(OutputEven, 'w+') as f:
                f.write(EvenStr)
        f.close()
        with open(OutputOdd, 'w+') as f:
                f.write(OddStr)
        f.close()

        # Print to console Original string and new Even/Odd
        print BytesStr
        print EvenStr
        print OddStr

# Driver function
if __name__ == '__main__':
    main()
```

6 Problems

Two major problems were confronted and solved in this project. Both were in the implementation stage of the project.

1. After wire wrapping and programming the EPROMS for LAB1, the led was not working as expected. After using the multimeter to trouble shoot the problem it was found that problem was in the external circuitry. After rewiring that circuitry the LED worked as expected.

2. After programming the EPROMS for LAB2 there was no current coming from the output of PORTB (which should power the Seven Segment Display). Dr. Rafi helped point out that the look-up table data was before the program data. This effectively had the program counter point to the data rather than an instruction. After separating the look-up table from the program and pointing the program counter to the program the seven segment display worked as expected.

7 Conclusion

Both labs were completed and the implemented design was verified. After this lab I have a much better understanding of the intricacies of μComputer and μController design. The key take aways for me from this project were:

- The care needed in verifying schematic designs and in making sure the prototype is wire wrapped correctly
- How a μProcessor communicates over the bus to memory and I/O devices
- How a μProcessor communicates with slower devices
- How byte memory that is split even and odd works
- How control registers and data direction registers are used in I/O devices
- PCB Design
- How Motorola HEX (S) files are encoded
- An overall better understanding of the system design of a μComputer