Alexander Lemieux
Baxter Vintage Computing

Milestones

Oct 18th- start learning 6502 assembly and set programming goal
Jan 17th- have all the computers diagnosed and parts ordered
Feb 28th- have all computers fixed
Apr 3rd- be able to write hello world in assembly (completed 1/28)
Apr 3rd- be able to display a dot on the screen (completed 2/28)
May 3rd- be able to move dot using input from keyboard (completed 3/20/20)

##

End

5/26
Last day of class, highschool. This will be the last update. I read the journal from start to end one last time to reflect on the past year. I enjoyed the past 4 years of learning, growing, failing, succeeding, crying, and laughing. These memories and experiences will stay with me throughout my life. I have achieved so much over the past year, and I am proud of that. Even though I completed all of my goals that I set, there is much more for me to learn about assembly and programming. Because of this, I will continue to learn and program in assembly so that I can one day achieve my ultimate goal of assembly programming which is: to develop a full apple ii game written in assembly. This has been an amazing journey, I am filled with joy thinking about my senior year and this flex friday project is a big part of that. I will leave you with a parody quote.
“Assembly, the final frontier. These are the voyages of the student Alexander Lemieux. His one year mission: to explore a strange new programming language. To seek out help from documentation and communities. To boldly go where no student (to my knowledge) has gone before!”

5/15
Today I will record my presentation. Today is the last Flex Friday I will ever have.
5/8
Today I will start by planning out my presentation which I will record. I will record my presentation by recording a private google meet where I will also show my screen so I can demo the code. If I finnish planning my presentation before the end of the day, I will try to fix the script that I’ve been working on to display a sprite.

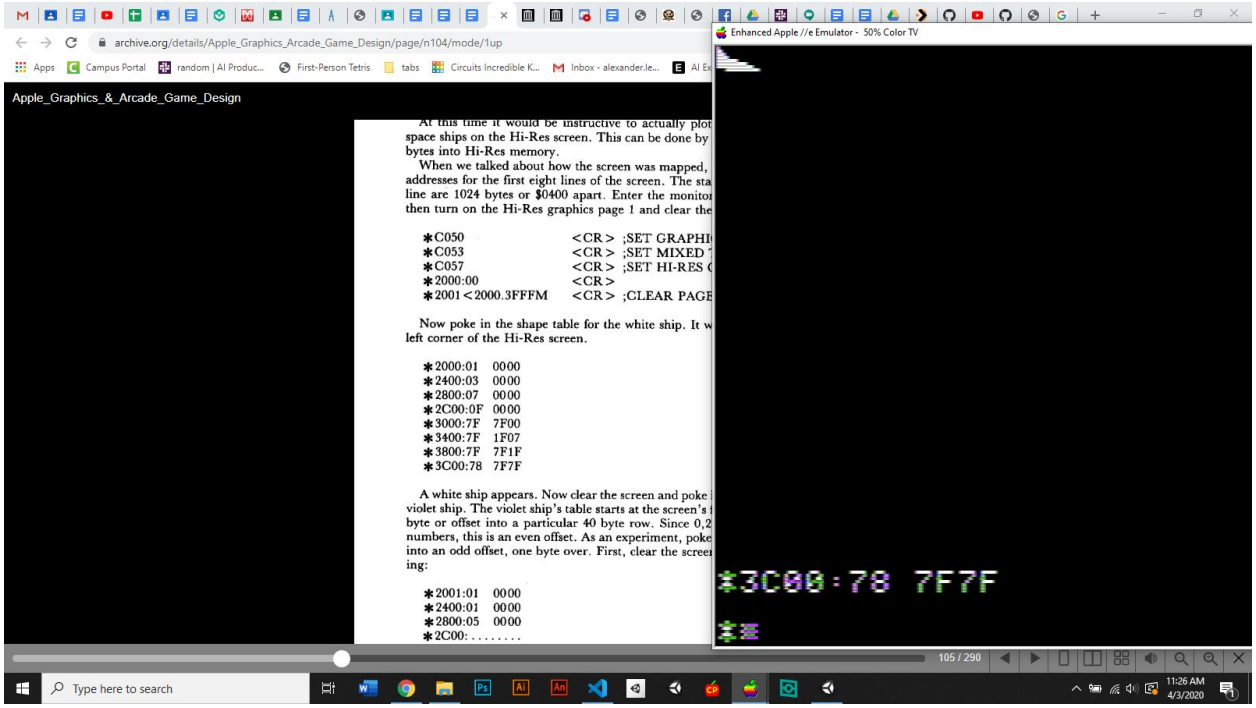
5/4
After talking with someone on the facebook group, I learned that my code should work and the reason it doesn't is probably because of an assembler error. The person said my code looks good and that the problem might be with the software I am using. Because of this, I have decided to spend the remaining two weeks working on recording a presentation. I know this seems like I'm giving up but I'm not. I have surpassed my original goal of having a pixel appear on the screen by the end of the year.
5/1
Today I will type in the full program from the book into the emulator. If that works, I will work on my method which is to draw the sprite computationally, if I can get it working, it would save a ton of space and would be faster to write. I will also look at the Xdrawing subroutines found in the book. The Xdrawing subroutines are useful for moving sprites because you don't have to waste time by clearing the entire screen everytime you want to move a sprite. The program I typed in didn't work so I made a post In the facebook group, someone there will help me troubleshoot the code

4/28
Today I looked at my post that I made on the apple ii facebook group. People explained that to increase speed the lookup table is generated by the programmer by hand (like all of the examples in the book) that seems like a lot of work to me because I just want one shape to be drawn, but for someone who is making a game with more sprites it seems like less work. Someone told me that you can actually generate the lookup tables at the start and they linked this website. This is a disassembly of the applesoft basic programming language/ rom. In it there is a subroutine that gets the screen address, the subroutine starts at $f411. I will look into this because I don't care about speed, I will also look into the method that I thought of which is keep the starting screen address stored, and then add $0400 to a temp address to get to the next line. (like the top left of the shape would be the screen address that would be stored, the address would be copied into a temp variable, the hex value of that section would be loaded and stored at the (temp location + width offset) , this would be repeated across the line until the shapes width is reached (the variable would start at 0 and count up), then the offset would be set to 0, the temp location would be incremented by $0400 and the process would be repeated until the height of the shape was reached, when you want to move the shape, you change the stored screen value (+ or - $0400 for down or up and + or - $1 for right of left) and then draw the shape). Its confusing but I think it would be easier for my purposes.
4/17
Today I will attempt to make a sprite using the method shown in the book (making a sprite table, then drawing each pixel in on the screen in the code). Today I began talking with the facebook group about why some parts of the subroutines repeat themselves. The reason for that is for the lookup table to work (the lookup table is what is used to see the screen position/ memory location for each point on the shape). I got a lot of code typed out and will test it next flex Friday
4/10
Today I will try to get in touch with some people in the apple ii facebook group who have experience with apple ii assembly game programming and sprite tables, I am doing this so I can

make sure I understand how it works and also make sure I have a clear understanding of how to implement the sprites. The original mario NES assembly code is less useful than I thought because the 6502 chip in the nes uses some different op codes (the operation codes) than the 6502 chip in the apple ii, this means that I can't copy what I see there but I can still learn from it. I figured out the order that the subroutines need to be (I think) the order is page 116,115,118 (basically put the subroutine on page 116 first and then the subroutine on page 115 and then the subroutine on page 118. Page 136 and 137 show an example of this. Hopefully next week I will try to program in a sprite drawing script. Page 191 has full code for a ship that drives like a car. Page 249 has the full code for a game with the drawing of the sprite and input etc, probably the best thing I can use to check if i'm doing stuff right.

4/3

Today I will continue reading the book about making and implementing sprites on the apple ii in assembly. I will follow along with the example in the book starting on page 104. I plotted the shape, the picture is below. And I am looking into how to store them, I think page 268 will be useful to learning how to actually store the shape tables



Setting Yet Another Goal

3/27

Today I will continue reading about making and implementing sprites on the apple ii in assembly using this book. Sprites start on page 100 and beyond. I will also implement a better version of my code that checks to see if the player is on an edge and if they are, they can't move out of the screen view. I implemented the better version of the code and saved it under a different name so I can keep the original. The new one is saved at PLOTBTTN. I also experimented with a

better way to write coroutines. Below is an example. In figure 1, the coroutine name is on the same line as an instruction, this works perfectly fine, until you want to make something happen before LDA #00, this is why figure 2 is better. Figure 2 has a ":" after the coroutine name, this means that you can have the coroutine name without an instruction. Because of this, If I wanted to add something above LDA #00, I could easily do that with the insert command. It didn't matter at first, but it is more useful as I get more advanced. I learned this while looking at the original assembly source code for the original NES mario game. Here is the link to the full code. At first I wondered why they did this but it makes sense that a game made over a long period of time would benefit from flexible code. It also looks neater and more organized. I will spend the rest of the day reading the book I found about apple ii assembly programming (it was written specifically for game programming on the apple ii). I will research more about sprites, how to make them, and how to move them.

Figure 1

```
KEYIN LDA #00
      JSR $FD1B
      JSR $FDDA
      JMP KEYIN
      END
```

Figure 2

```
KEYIN:
   LDA #00
   JSR $FD1B
   JSR $FDDA
   JMP KEYIN
   END
```

Victory

3/20

Today is the day I will see if the solution to my problem is the order of the subroutines. I start today like every other flex friday by praying to all the computer gods hoping that I can get some code to run. I will update the log after I try to fix my code.

Update: IT WORKED, THE FIX I MADE WORKED below is the final version that works. Because I know how to save programs, I saved it as bttnplot, so now I have my first assembly script that I made. Before, I was just following tutorials, but this is the first assembly program that I thought about how to make it work and then fixed it when it didn't work. You can move the dot on screen with W,A,S, and D, and exit the script by pressing Q. In the apple ii asm code doc, I made an untested version that also checks to make sure that the space that the user wants to move is on the screen, if it isn't, it doesn't move the pixel. Below is also a picture of the result, you have to trust me to say that it does move.

BTTNPLOT (not in the script just a way for me to remember what I saved it as)

```
JSR     $FB40 ;LORES GR
        LDY #05
        LDX #05
KEYIN   LDA #00
        JSR $FD1B ; get button pressed
        CMP #$C1 ; A
```

```
        BEQ AMOV
        CMP #$C4 ; D
        BEQ DMOV
        CMP #$D3 ;S
        BEQ SMOV
        CMP #$D7 ;W
        BEQ WMOV
        CMP #$D1 ;Q
        BEQ EXIT
        JMP KEYIN
CSAVE JSR $FF4A ; saves registers
        JSR $F832 ; clear low res screen
        JSR $FF3F ; restores registers
        LDA #44 ;COLOR15
        STA $0030
        RTS
AMOV JSR CSAVE
        DEY
        TXA
        JSR $F800 ; plots point
        JMP KEYIN
DMOV JSR CSAVE
        INY
        TXA
        JSR $F800
        JMP KEYIN
SMOV JSR CSAVE
        INX
        TXA
        JSR $F800
        JMP KEYIN
WMOV JSR CSAVE
        DEX
        TXA
        JSR $F800
        JMP KEYIN
EXIT    BRK
        END
```

3/19
I know I have not gotten the pixel move code working (I am not going to try my solution to the broken code until tomorrow so I can have something to do) but, I was looking into high res graphics and producing shape tables here is the book I am reading https://archive.org/details/Apple_Graphics_Arcade_Game_Design/page/n101/mode/1up page 100 is the start of where I'm reading

Breakthrough

3/17
9:25pm
I just woke up to write this because I think I figured out my problem of why the script isn't working. The actual code itself is fine, I am doing everything right. The problem is the ORDER! Right now, I have it so the program sets everything up (sets to low res, loads the value 5 into the x and y register and sets the color) and then it goes into the csave subroutine that saves the registers, clears the screen, and then restores the register, after that it does rts (return to subroutine). In high level languages like c#, you can have subroutines that won't be run unless they are called. In assembly you can't have that. ORDER MATTERS! When I put rts at the end, the computer doesn't return to the right subroutine because I didn't jsr into it from the correct

one. My theory is that if I put the keyin subroutine before it, the csave code won't be run until it is called from one of the move subroutines. I will try this in the morning

Learning How to Save Code

3/16
I have discovered how to save, load, and replace lines.
For saving I type "SA filename" or "SA filename , s6,d2" (slot 6 and disk 2 to save to another floppy disk)
For loading I type "LO filename"
And for rewriting lines I type "M 2" (2 is the line number so "M 15" would modify line 15. After you type that and press enter, the next line is what you want the line to be replaced by.
For adding a line above another line I type "i 2" (2 is the line number so "i 15" would insert a line above line 15).
Because of the M(odify) command I can now, write a program, save it and then continue writing or modify a line. Now I don't have to rewrite my program every time I want to make a change.

First draft of dot moving code

3/6 (entered on 5/15 because I saw that I didn't have an entry for 3/6 which)
Entered in the code for the moving dot with keyboard script, it didn't work, so I spent the whole day trying to figure out why it didn't work.
3/4
I created a google doc that I can brainstorm my code in before typing it into the actual assembler. Using the doc, I can see every line of code at once, edit any line easily, and save the code I've written. HERE is the link

Adding another milestone: move a dot using keyboard input

3/2
I know it's not friday, but I wanted to see if I could do what I thought I could do (use my knowledge to test for a certain key if pressed). It turns out I can. The program below tests to see if a key is pressed, if one is, it tests to see if that key is the "A" key, if it is, it prints out "A", if not it waits for another key.
KEYIN LDA #0000

```
        JSR $FD1B
        CMP #$C1
        BEQ PRNT
        BNE KEYIN
PRNT JSR $FDED
        JMP KEYIN
        BRK
        END
```

If I wanted to test for more than one key, I could add “CMP #(high-order keycode)” and then “BEQ PRNT” on the next line testing for W,A,S, and D would look like this

```
KEYIN LDA #0000
        JSR $FD1B
        CMP #$C1 (A)
        BEQ PRNT
        CMP #$C4 (D)
        BEQ PRNT
        CMP #$D3 (S)
        BEQ PRNT
        CMP #$D7 (W)
        BEQ PRNT
        BNE KEYIN
PRNT JSR $FDED
        JMP KEYIN
        BRK
        END
```

I could combine this with parts of the “looping dot” code (the one that increases the x and y value by one and plots the result and does this continuously), to test if a certain key was pressed, and if it was, increase x and y value and plot a point (highlighted in yellow because I added the highlighted text Long after the other text (specifically on 5/8). 3/2 was also the day that I combined some of the keycode checking script and the dot plot script to make a program that when the “A” key is pressed, the screen is cleared, the x and y registers are incremented by 1, and a new dot is plotted. Also was the day that I made the first version of the dot moving script (not working, but wrote all of the code)

A **really** good day

2/28

I looked up how to use the low res graphics and plot a point in assembly the code is below (DPLOT)

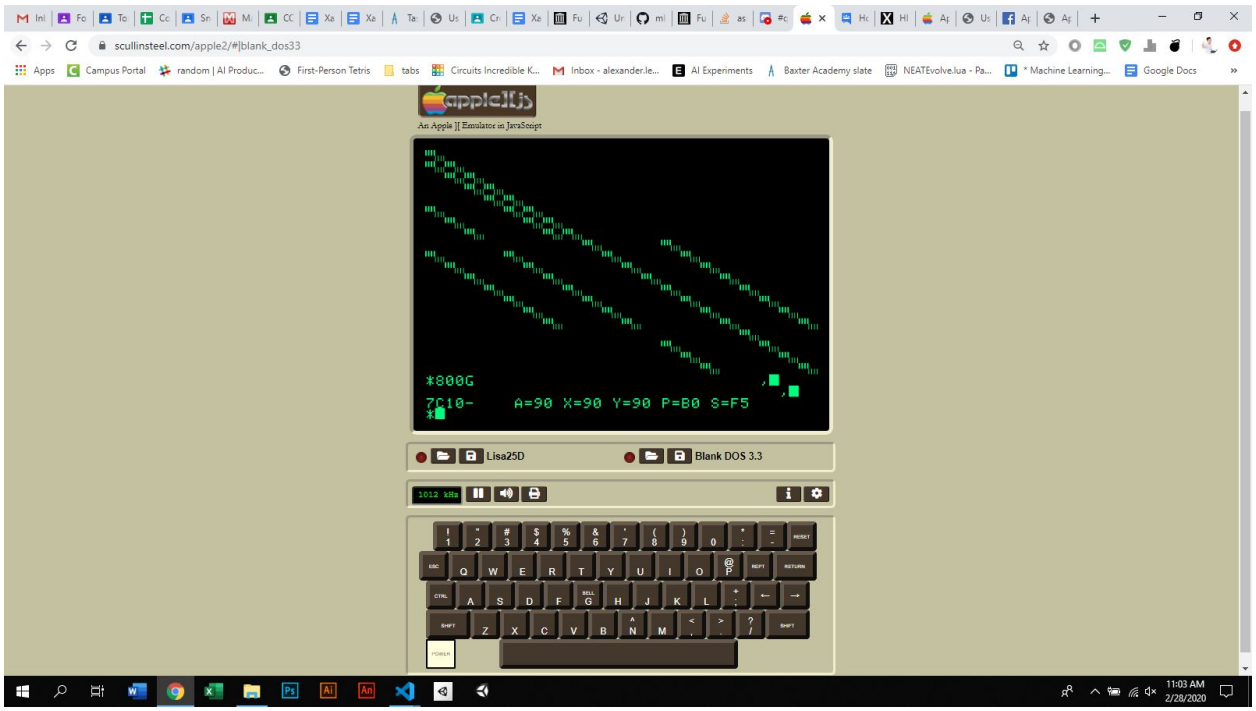
```
 JSR $FB40 ;LORES GR
 LDA #44 ;COLOR15
 STA $0030
 LDY #15 ;COL
 LDA #05 ;ROW
 JSR $F800 ;PLOT
```

I will experiment with adding to the registers to move the dot around.

I was able to successfully move the dot by adding to the registers. The acumulator gets scrambled everytime a point is plotted, so I am using the x register to keep the value for the accumulator. I can add or subtract from the x register and then use txa (transfer x to the acumulator) to set the correct value. Below is the code that I used to test if this approach works.

```
        JSR $FB40 ;LORES GR
         LDA #44 ;COLOR15
        STA $0030
        LDY #0
        LDX #0
        LDA #0
LOOP JSR $F800
        INY
        INX
        TXA
        JMP LOOP
        BRK
        END
```

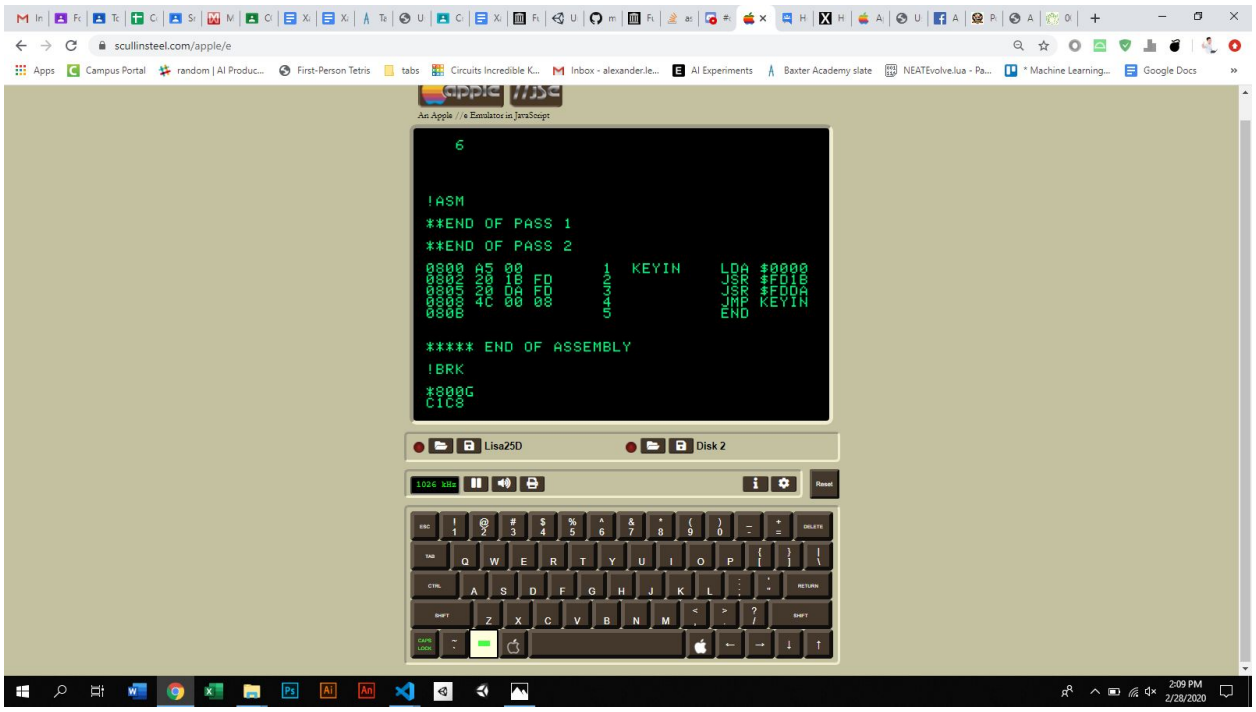
Below is a picture of the result, it works. There are extra dots that were plotted because I did not have an end condition so the extra dots are from the registers overflowing



I can clear the screen using clrtop ($f832)

I was reading up on the key input and keycodes and I made a program that reads the key pressed and then prints the keycode. This is useful because next week I can write a program that checks if a key pressed was a certain key. Even though this is simple and has been done by every programmer since the apple ii came out, this is a big step in my assembly programming skills. Below is the program and the outputs (each keycode is 2 characters long, and I just pressed random keys.) (BTTNCD)

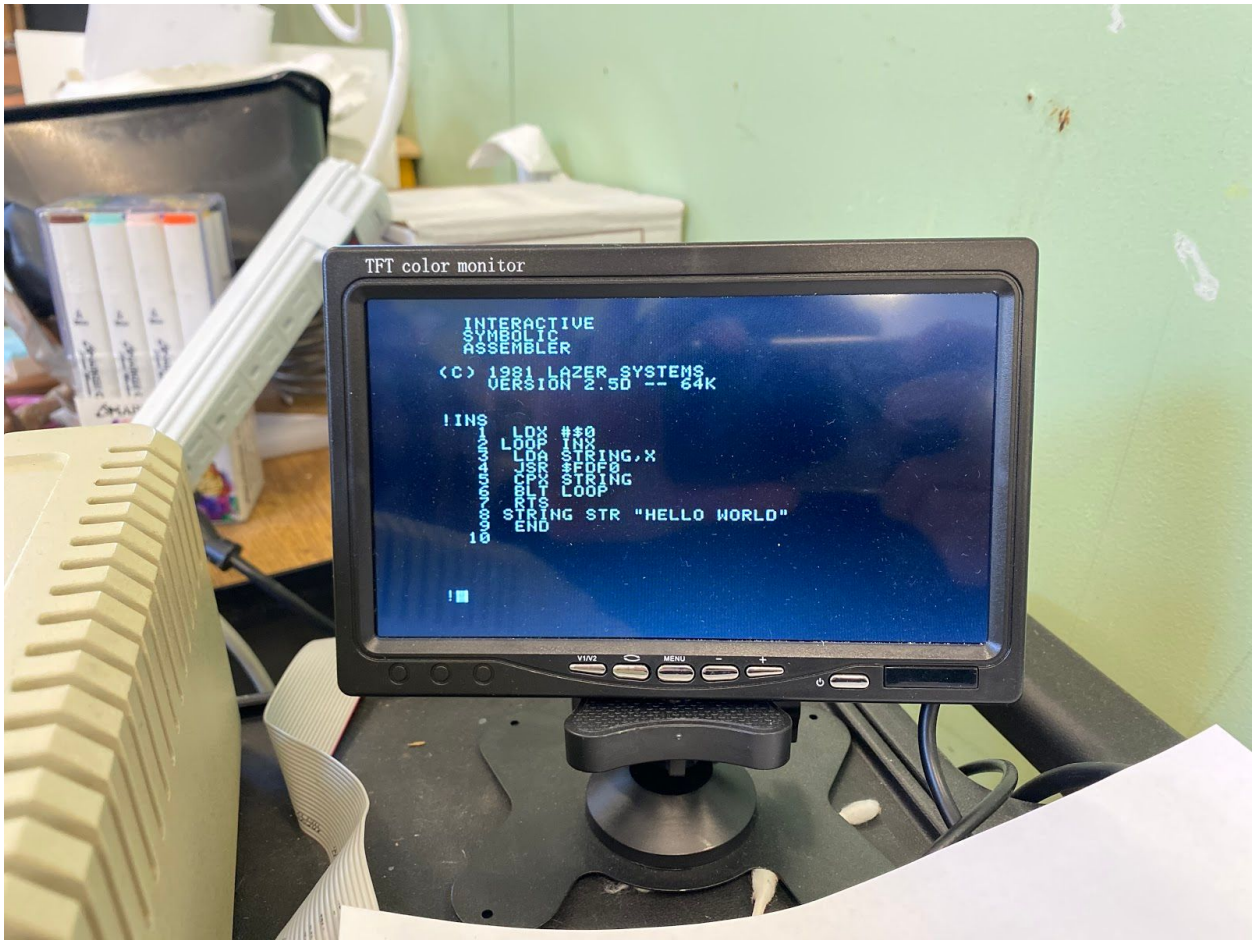
```
KEYIN LDA #0000
      JSR $FD1B
      JSR $FDDA
      JMP KEYIN
      END
```

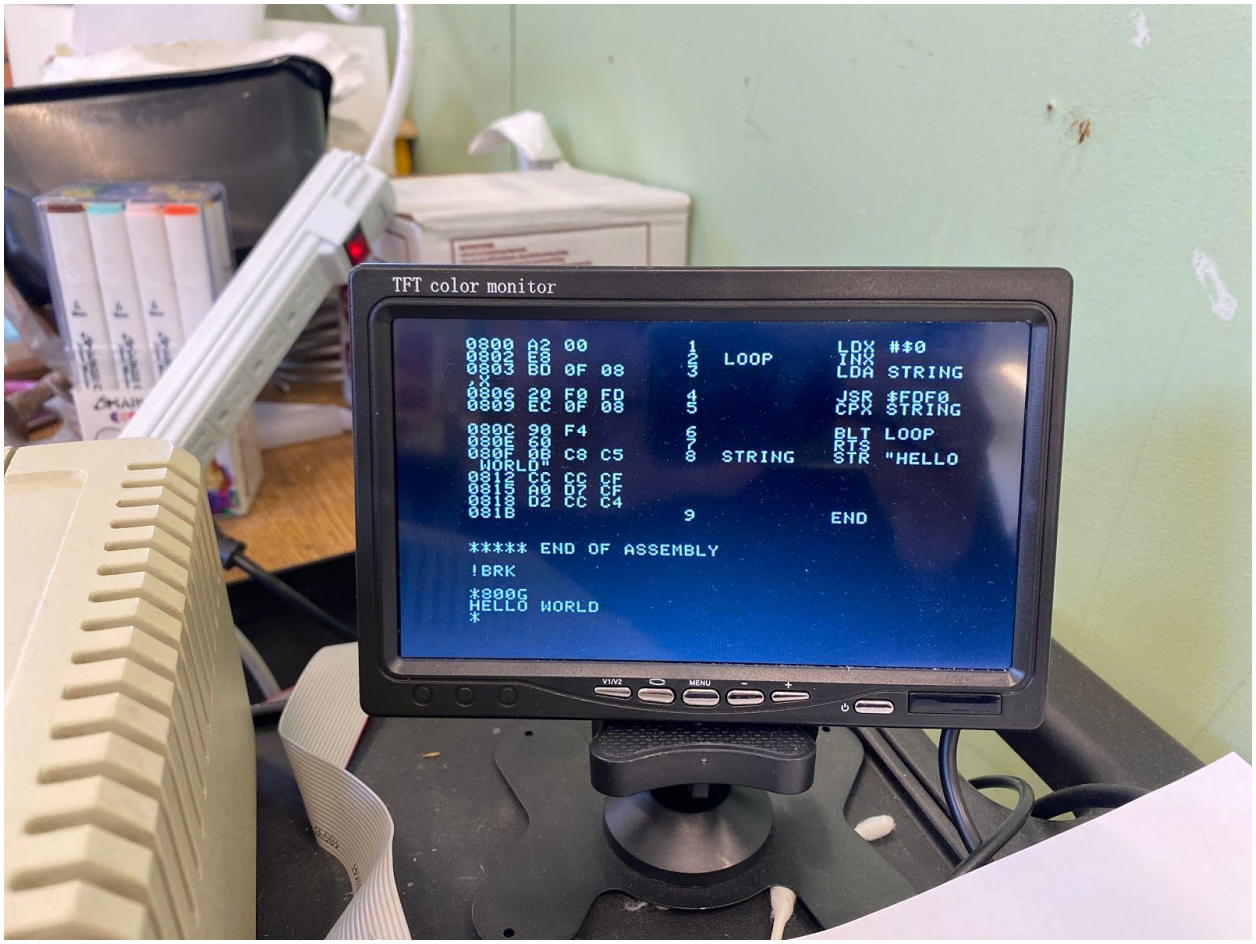


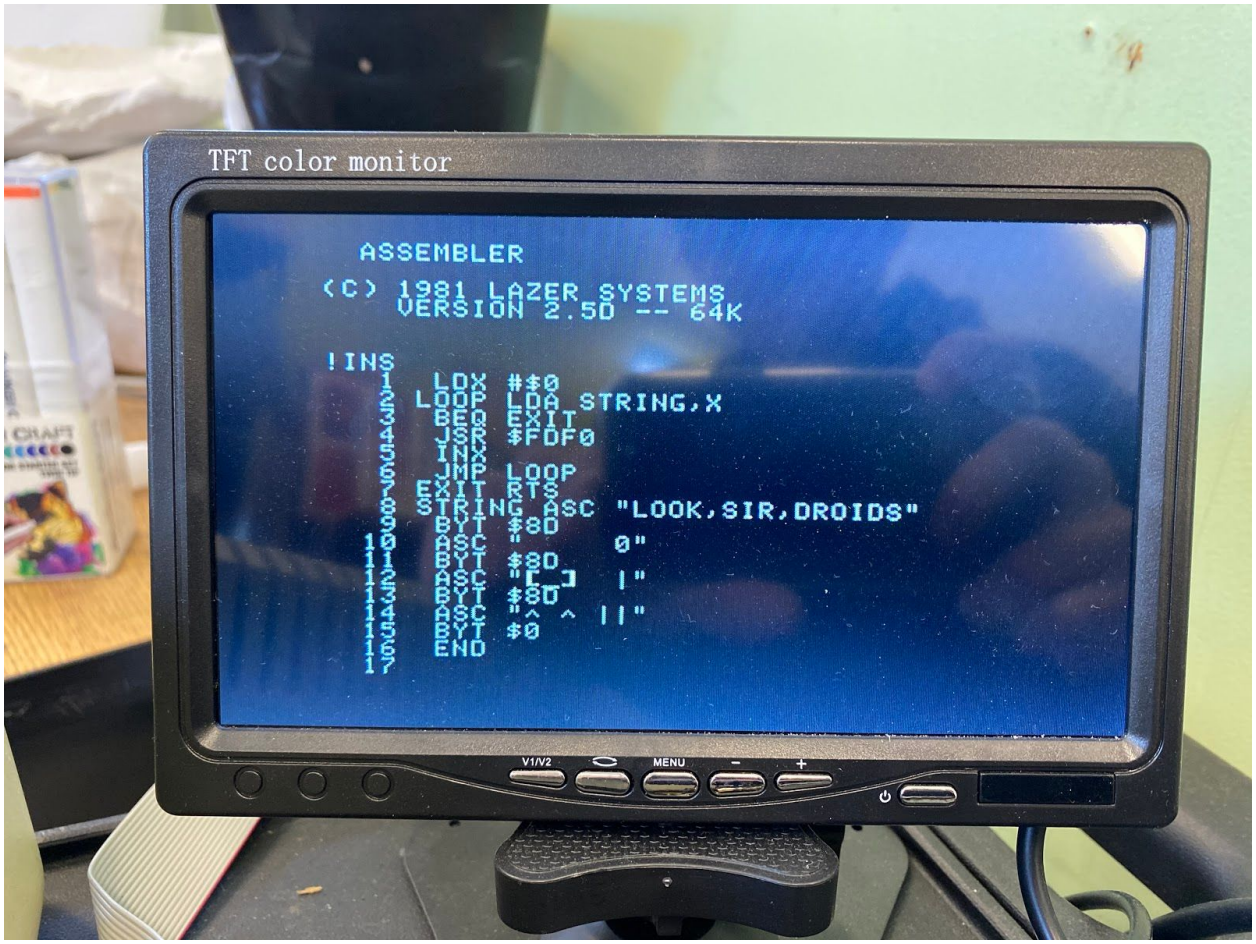
Learning About Keycodes and the Low Res Screen

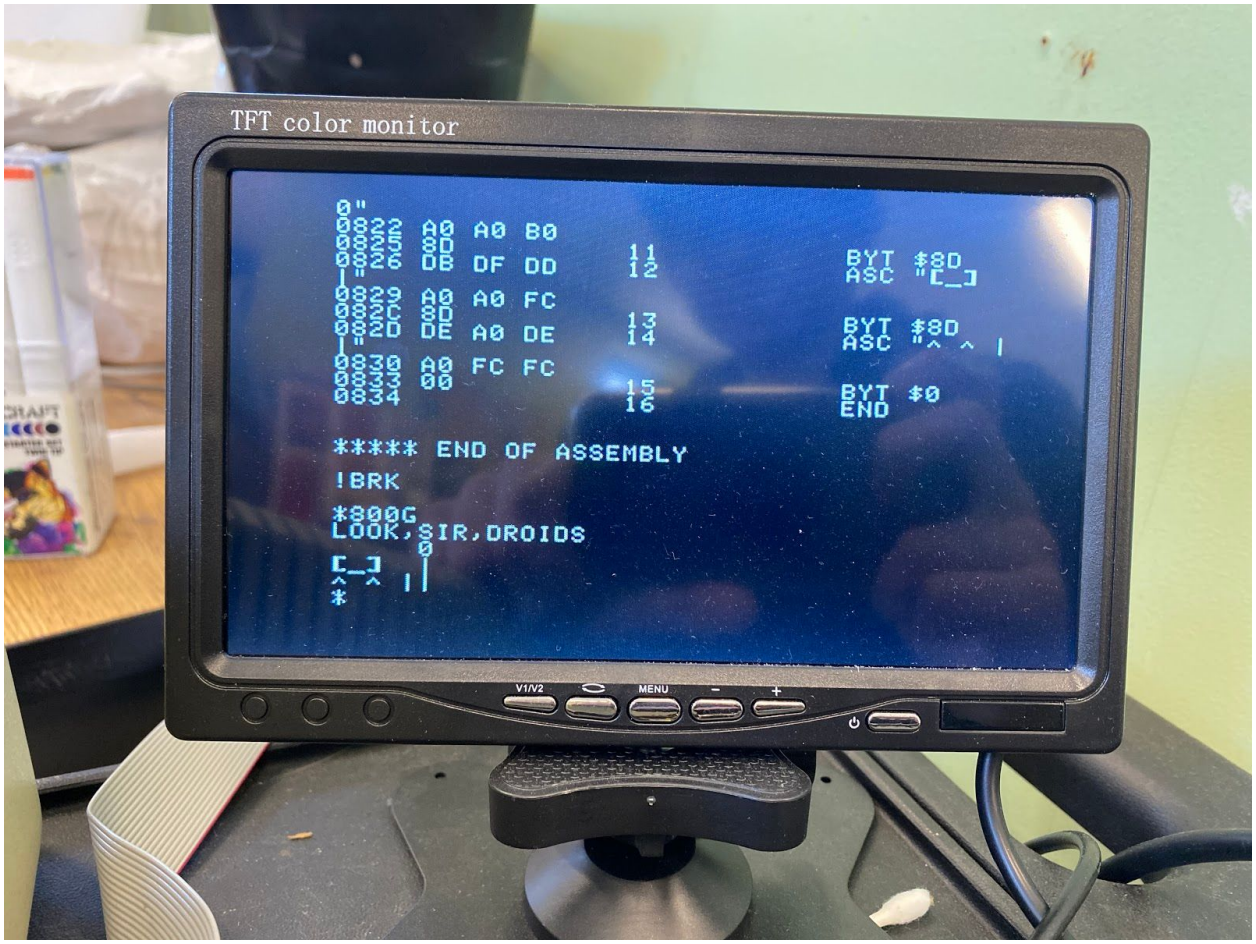
2/14

Today I brought in a working iie. I loaded up LISA on it and was able to type the hello world script I also worked on getting multiple lines to print




TFT color monitor
0800 A2 00 1 LDX #$0
0802 E8 2 LOOP INX
0803 BD 0F 08 3 LDA STRING
,X
0806 20 F0 FD 4 JSR $FDF0
0809 EC 0F 08 5 CPX STRING
080C 90 F4 6 BLT LOOP
080E 60 7 RTS
080F 0B C8 C5 8 STRING STR "HELLO
WORLD"
0812 CC CC CF
0815 A0 D7 CF
0818 D2 CC C4
081B 9 END
***** END OF ASSEMBLY
!BRK
*800G
HELLO WORLD
*
V1/V2
MENU


TFT color monitor
ASSEMBLER
(C) 1981 LAZER SYSTEMS
VERSION 2.5D -- 64K
!INS
1 LDX #$0
2 LOOP LDA STRING,X
3 BEQ EXIT
4 JSR $FDF0
5 INX
6 JMP LOOP
7 EXIT RTS
8 STRING ASC "LOOK,SIR,DROIDS"
9 BYT $8D
10 ASC " 0"
11 BYT $8D
12 ASC "[ ] |"
13 BYT $8D
14 ASC "^ ^ ||"
15 BYT $0
16 END
17
V1/V2
MENU
-
+



1/31

I tried to get LISA working on a ii+. It wasn't working so I switched to a iie. LISA works but the only iie with a keyboard that is at school has a broken 8 key. The 8 key is important because when you write programs, they start at address 800. To run a program you have to type 800g and that's impossible with a broken 8 key. I will bring in a fully working iie to program with next week. I got the ii+ working, the cpu wasn't installed properly so it was an easy fix. I ordered anti static bags for all of the components and will order more keyboards

1/28

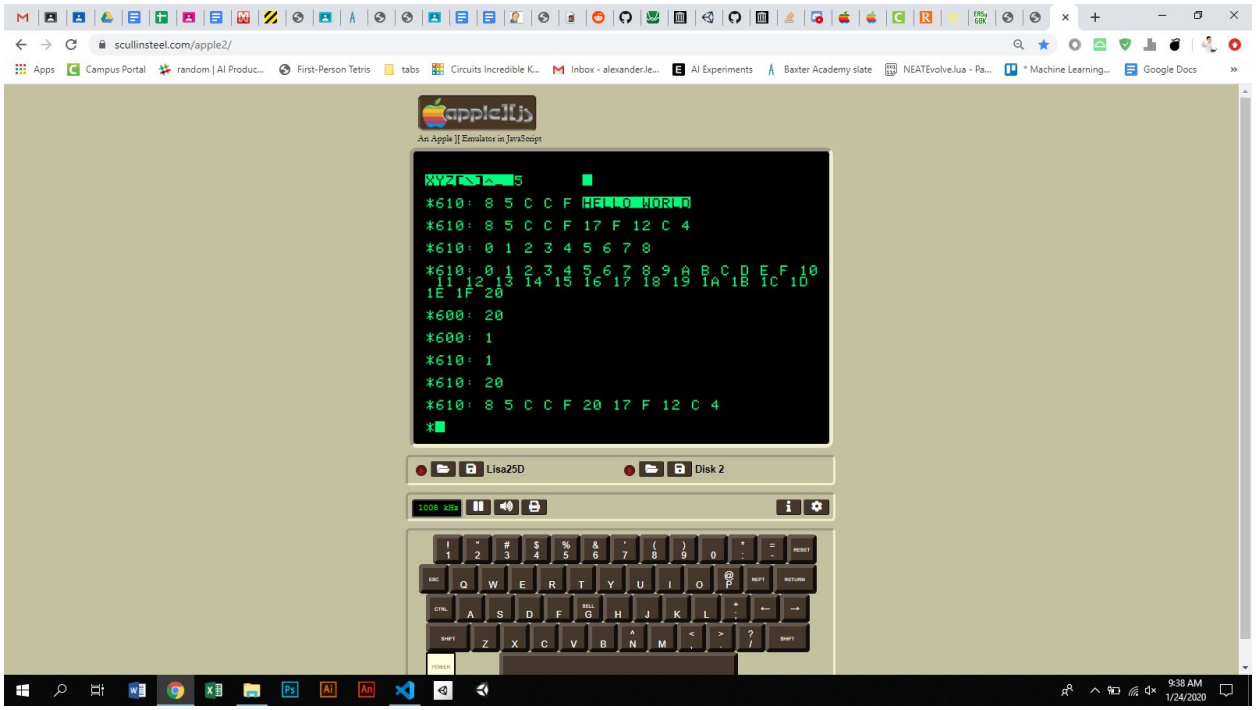
I was able to print hello world, it's not my code but I understand how it works, here it is.

```
(HLLOWRLD)
        LDX #$0
 LOOP INX
        LDA STRING,X
        JSR $FDF0
        CPX STRING
        BLT LOOP
        RTS
STRING STR "hello world"
         END
```

By typing “BYT $8D”after a string and before the next string (if it’s in ASC) will cause the ASC text to be printed on different lines. After the last string, the next line needs to be “BYT $0”

1/24

I started to better understand how memory location works. I was able to display a dot on a 6502 assembly emulator and display hello world on an apple ii emulator, I didn't do it the “correct” way storing a string and iterating through it. Instead, I used trial and error to find the memory locations of pixels on the screen and set the value of each one to a letter, this is inefficient but it helped me understand how memory works and how letters and characters are stored. I also am learning to count in hex.



“First Hello World” and What is Assembly?

1/17

I swapped in a new memory expansion for an apple iigs. I am working on reading the manual to understand how to use a ii gs. I am on chapter 5 of the assembly book. I will post update on the keyboard situation when I get more information

1/10

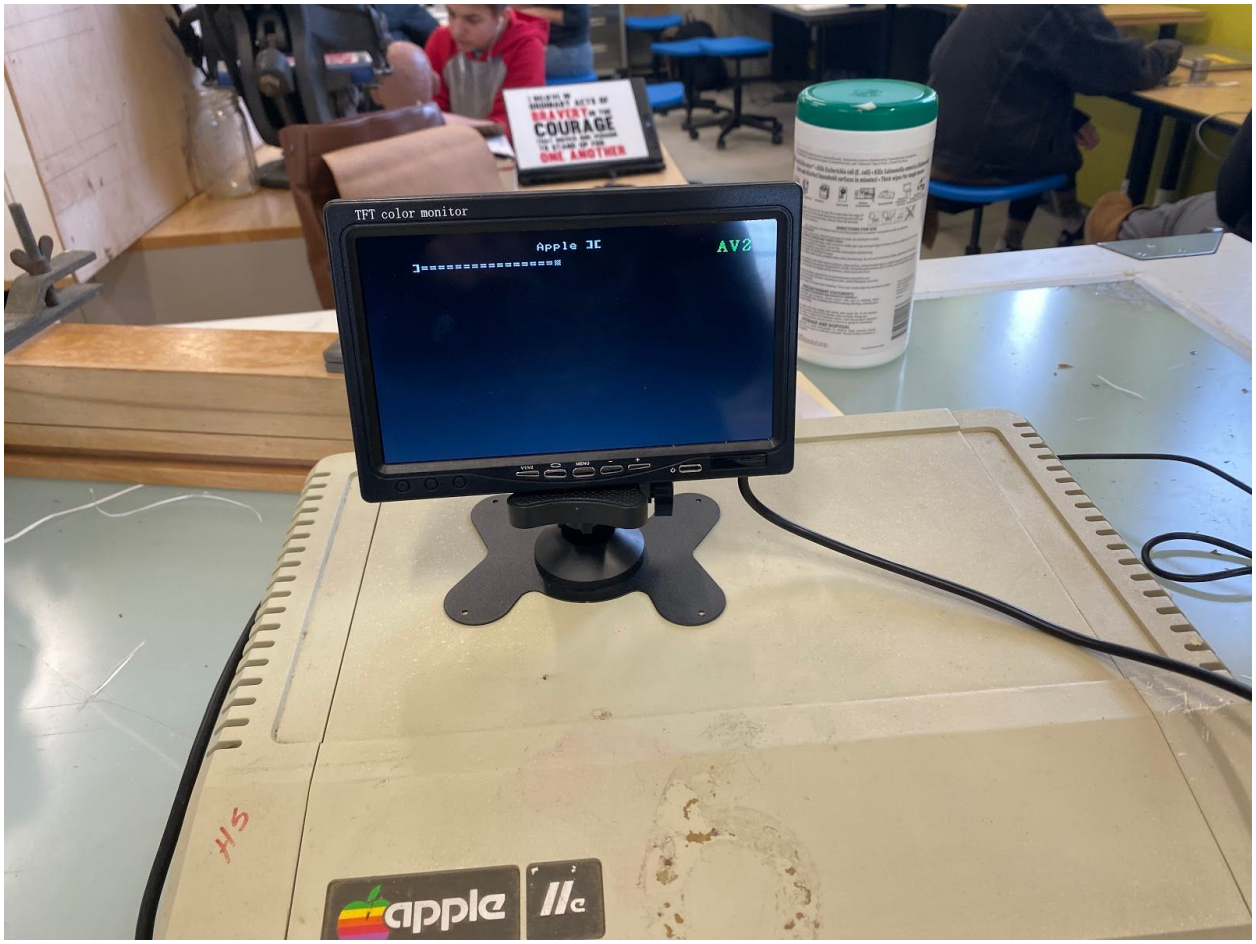
I brought in a new non working Apple iie. It was missing the power supply so I cleaned it and switched in a power supply from a working Apple iie. The new one works completely

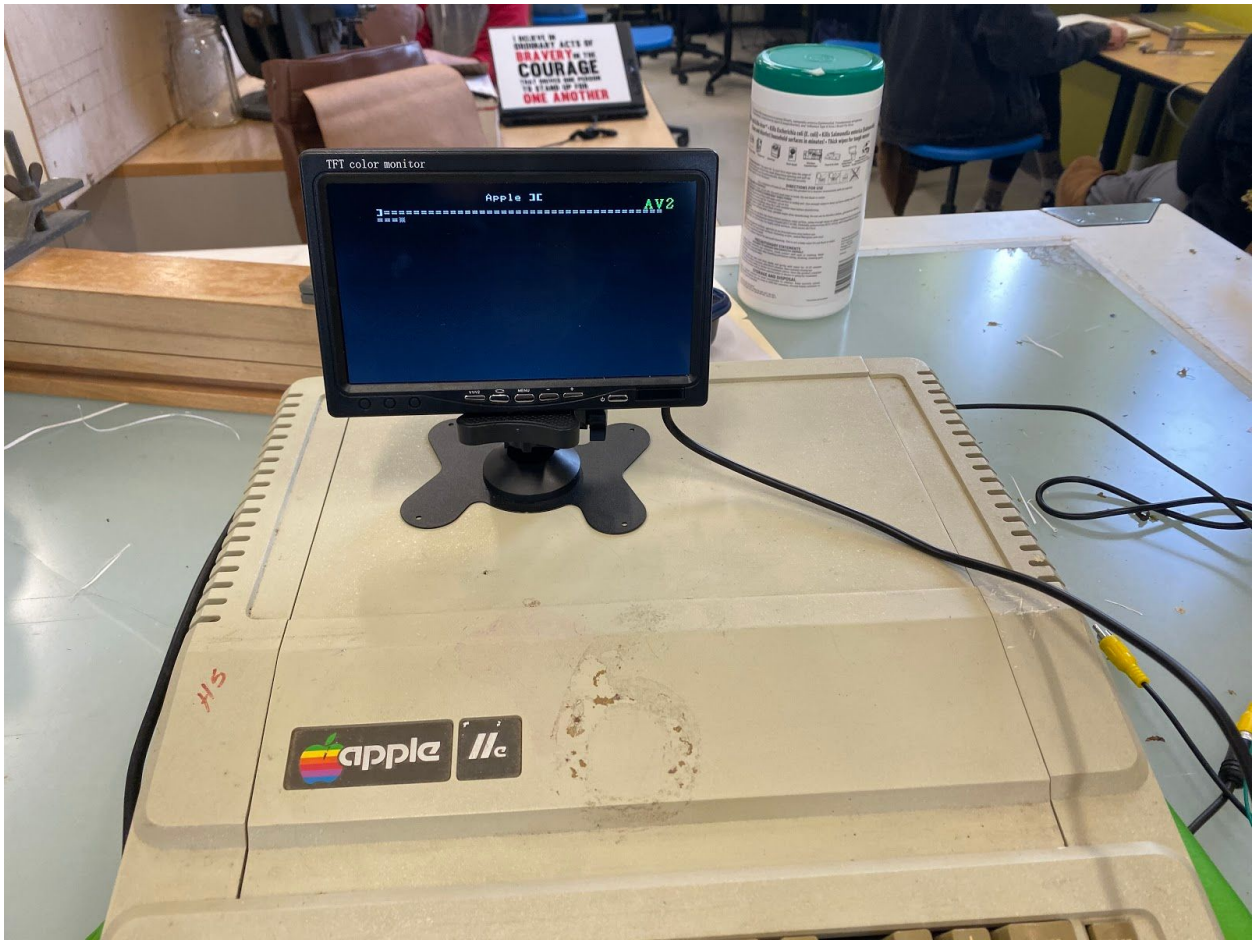
12/20

I switched the broken keyboard with a working keyboard. I have a replacement keyboard coming.

12/13

I put in the missing chip and booted the iie up. I did a boot test and everything is ok. I need to replace the keyboard. I will change the assembler because the current one does not work with the iie.


TFT color monitor
Apple]C
AV2
COURAGE
ONE ANOTHER
apple
IIc



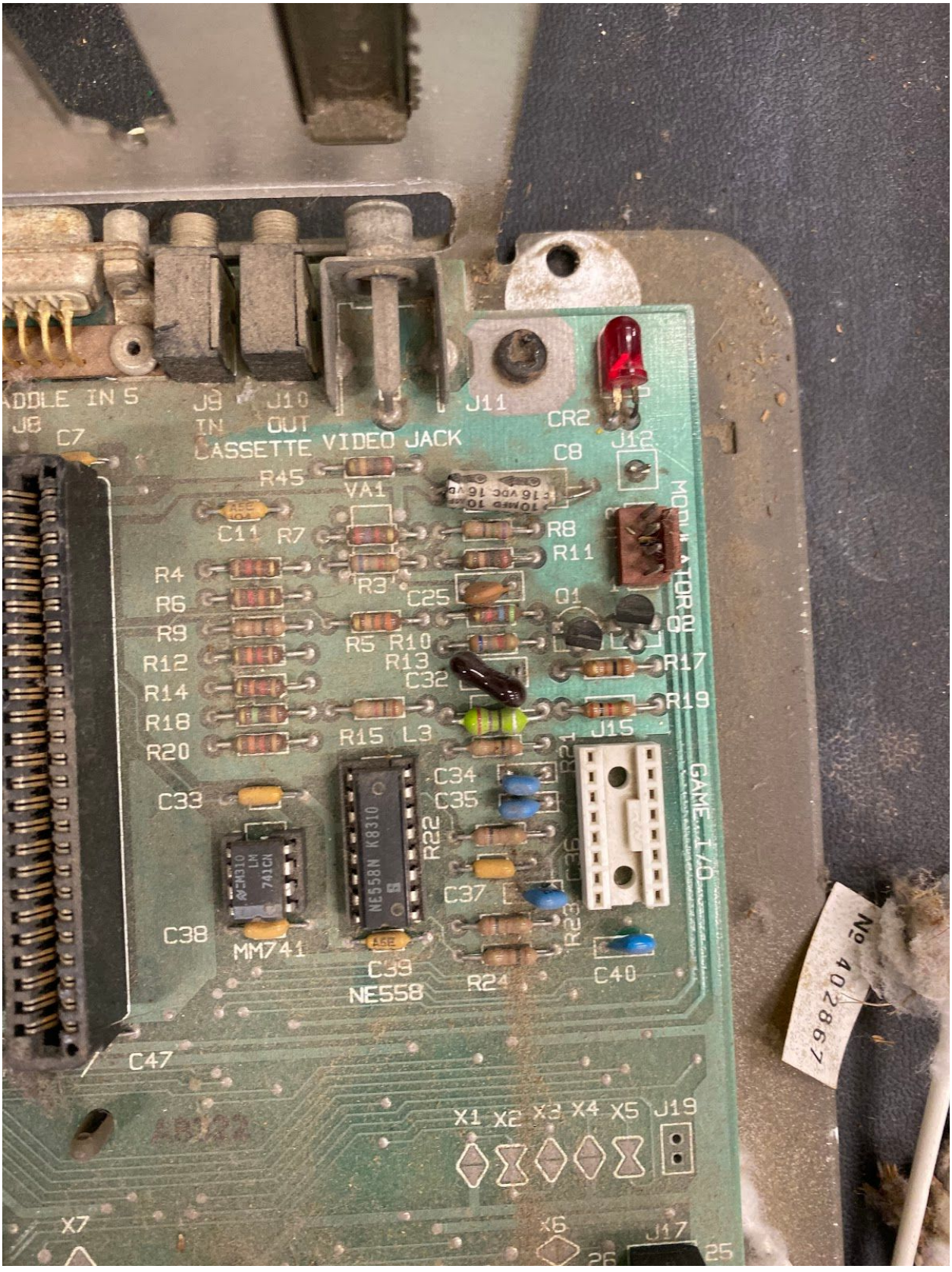
12/6

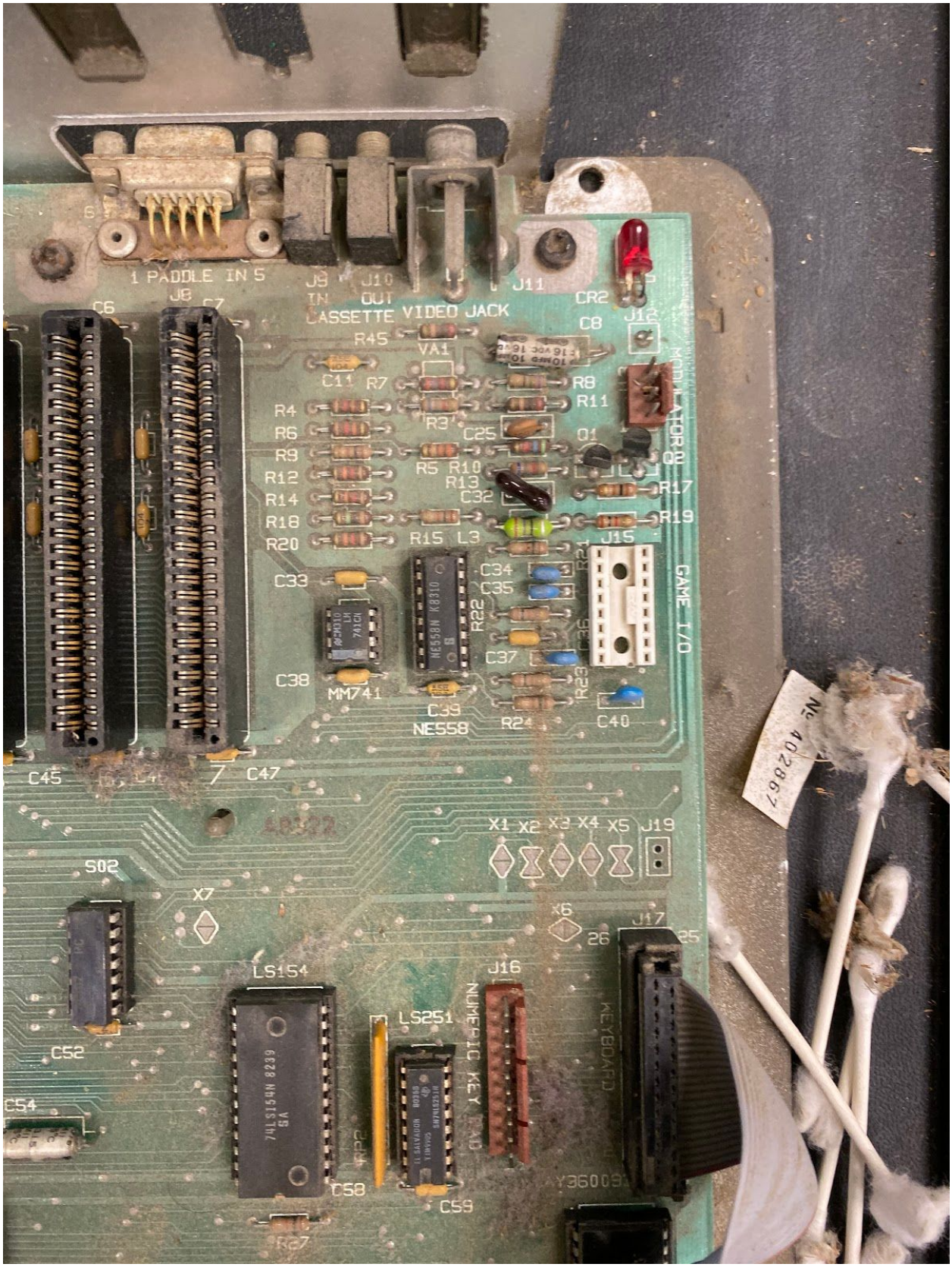
I bought the f74166 chip that was missing. I will see if I can get the Lisa assembler up and running. Due to the conditions of two of the computers, I will be trying to repair only one, and will be using the other for parts.

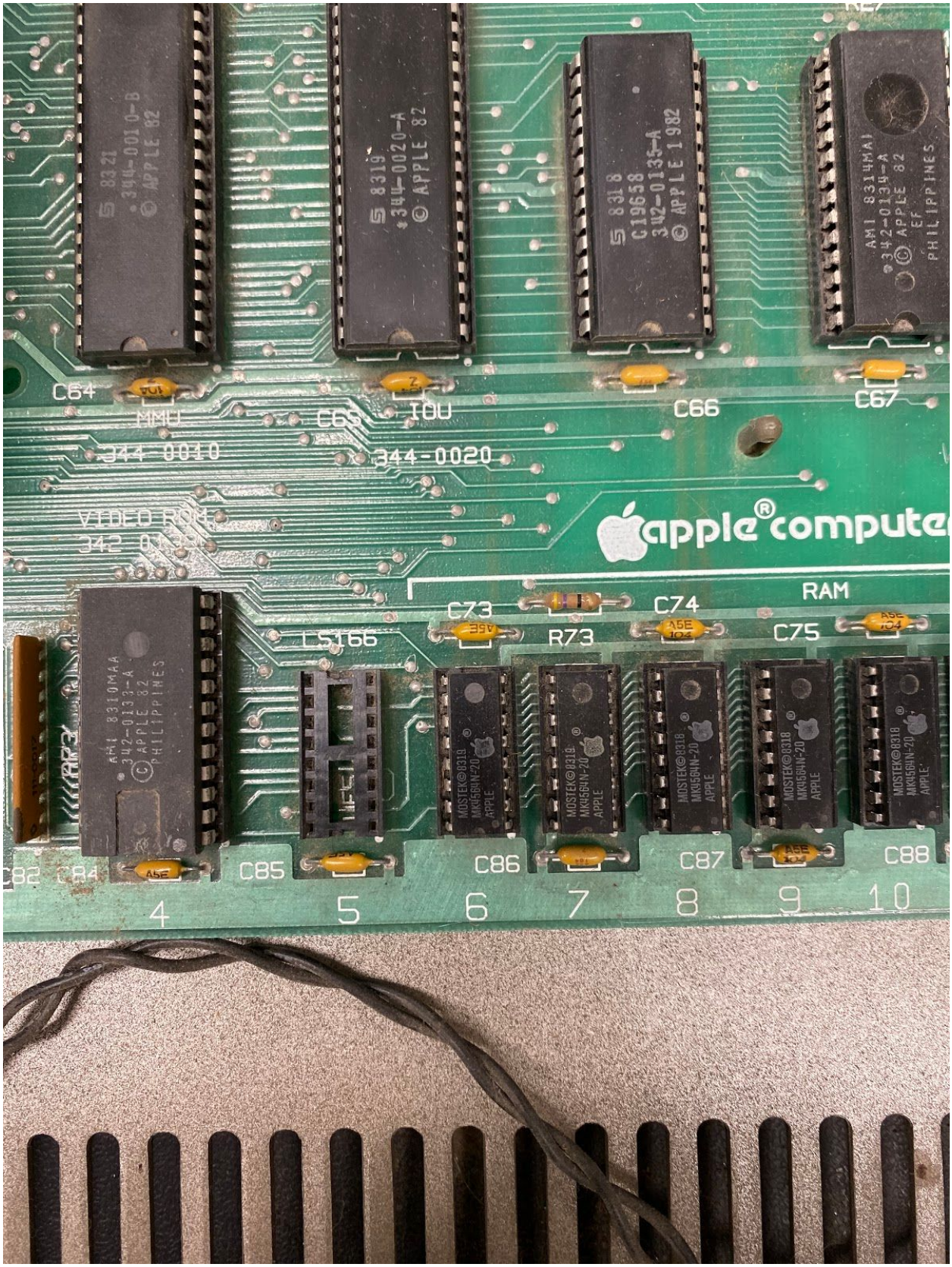
11/22

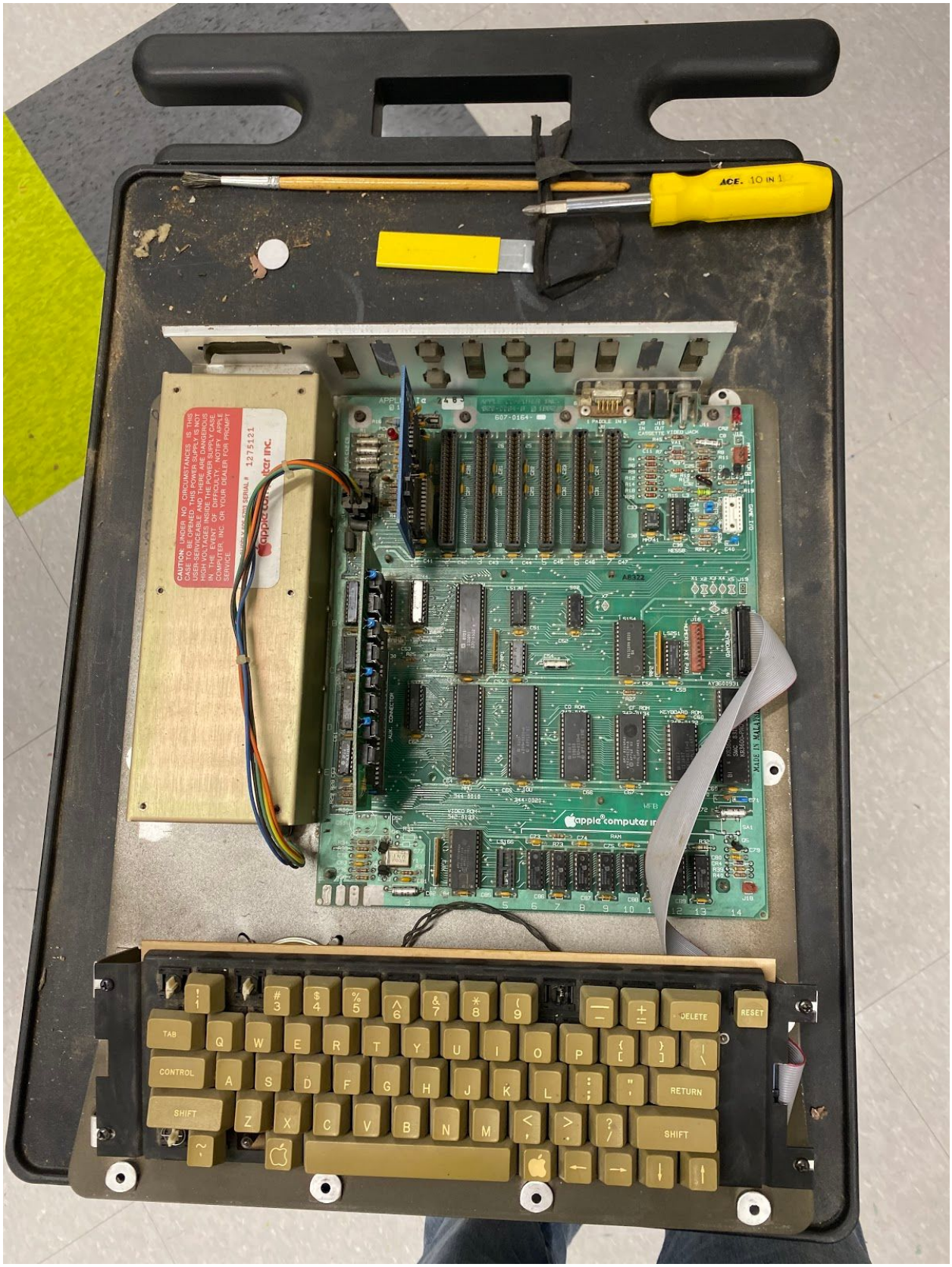
I brought in a non working Apple iie and started to clean it. It is missing a F74166 chip. I will order one soon. Here are some before pictures and one after picture. Next week I will order some air in a can so I can fully clean it. The assembler that I will use for my programming is called LISA.


7" TFT LCD MONITOR
Padarsey
AVAILABLE FOR VCD/DVD/GPS/CAMERA
HIGH-RESOLUTION PICTURE
FULL COLOR DISPLAY
LOW POWER CONSUMPTION
CAUTION: UNDER NO CIRCUMSTANCES IS THIS COMPUTER TO BE OPENED BY ANYONE OTHER THAN USER-SERVICEABLE AND THERE ARE DANGEROUS HIGH VOLTAGES INSIDE THE POWER SUPPLY CASE. DO NOT REMOVE IT. IF YOU NEED SERVICE, CONTACT APPLE INC. OR YOUR DEALER FOR PROMPT SERVICE.
apple computer inc.
DELETE
RESET
TAB
RETURN
CONTROL
SHIFT
SHIFT


ADDLE IN 5
J8
C7
J9
IN
J10
OUT
CASSETTE VIDEO JACK
J11
CR2
J12
C8
R45
VA1
C11
R7
R8
R11
R4
R6
R9
R12
R14
R18
R20
R3
C25
Q1
R5
R10
R13
C32
R17
R19
R15
L3
J15
C33
C34
C35
R22
C36
C37
R23
C38
MM741
NE558
C39
R24
C40
C47
X1
X2
X3
X4
X5
J19
X7
X6
J17
26
25
GAME I/O
Nº 402867


1 PADDLE IN 5
J8
J9
J10
J11
IN
OUT
CASSETTE VIDEO JACK
CR2
J12
C6
C7
C8
R45
VA1
C11
R7
R8
R11
R4
R6
R9
R12
R14
R18
R20
R3
C25
Q1
Q2
R5
R10
R13
R17
C32
R19
R15
L3
J15
C33
C34
C35
R22
C36
C37
C38
MM741
C39
NE558
R24
C40
GAME I/O
C45
C47
X1
X2
X3
X4
X5
J19
S02
X7
X6
J17
26
25
LS154
J16
LS251
NUMERIC KEY PAD
KEYBOARD
C52
C54
C58
C59
R27
Y360091


MMU
344-0010
IOU
344-0020
VIDEO ROM
342-0133
apple® computer
RAM
C64
C65
C66
C67
C73
C74
C75
R73
LS166
C82
C84
C85
C86
C87
C88
4
5
6
7
8
9
10

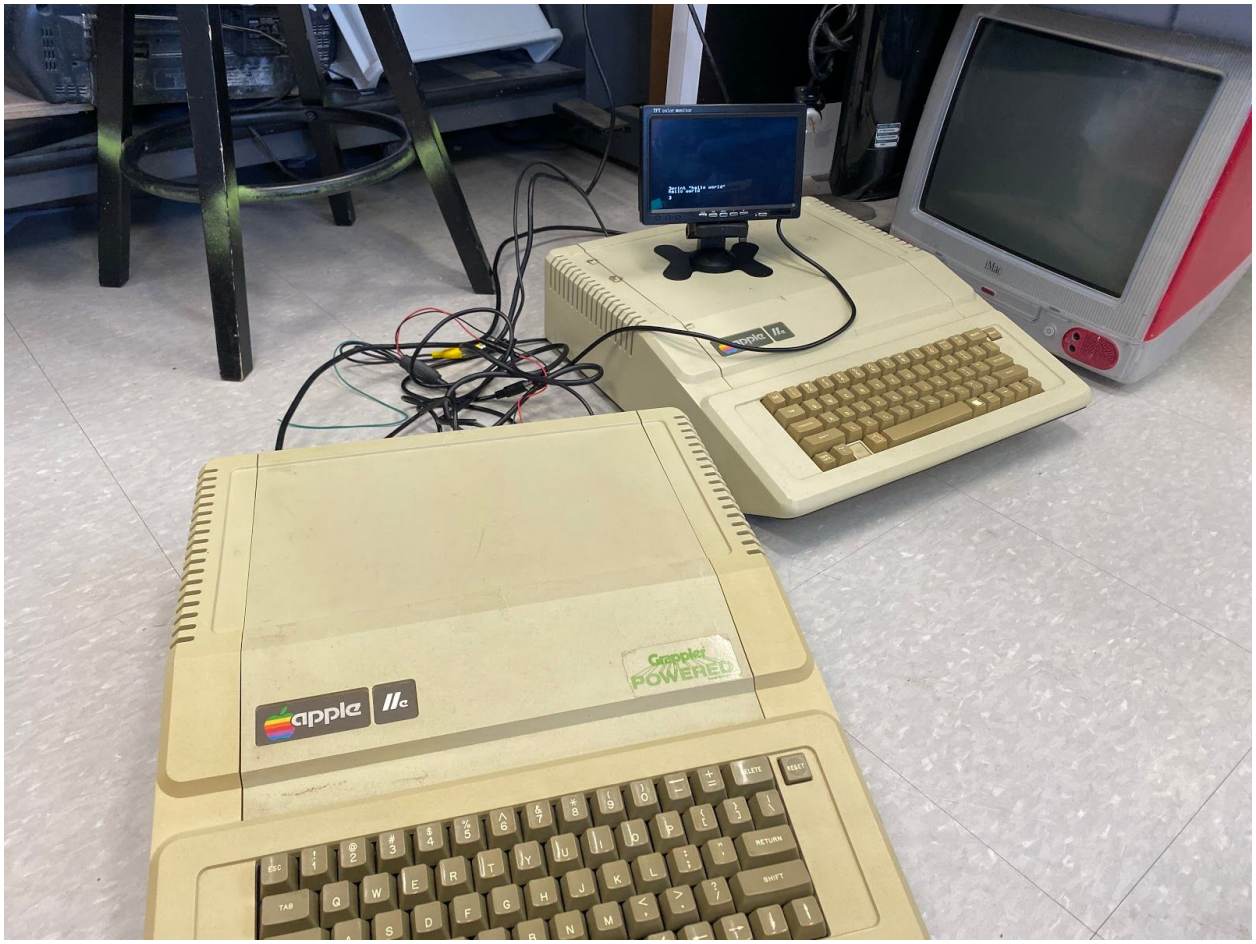

ACE. 10 IN 1
CAUTION: UNDER NO CIRCUMSTANCES IS THIS CASE TO BE OPENED. THE POWER SUPPLY IS NOT USER-SERVICEABLE AND THERE ARE DANGEROUS VOLTAGES INSIDE THE POWER SUPPLY CASE. IN THE EVENT IT IS DEFECTIVE, NOTIFY APPLE COMPUTER, INC. OR YOUR DEALER FOR PROMPT SERVICE
1275121
apple computer inc.
DELETE
RESET
TAB
CONTROL
RETURN
SHIFT
SHIFT

Cleaning and Diagnosing the Computers

11/15
I got a 7 inch monitor so I don’t have to carry a heavy crt display


TFT color monitor
Apple //e
AV2
MENU
apple IIe
RESET
DELETE
iMac


TFT color monitor
]PRINT "HELLO WORLD"
HELLO WORLD
]
apple IIe
iMac
ESC
TAB
CONTROL
SHIFT
CAPS LOCK
DELETE
RETURN
SHIFT
RESET


apple IIc
Grappler
POWERED
RESET
DELETE
RETURN
SHIFT
TAB
ESC

Getting A Monitor

11/8
I found a space, I will have two computers at school and rotate them after I work on them. Below is a picture of the first computer I bought in. It works and will be the computer I use to program on


CAUTION: UNDER NO CIRCUMSTANCES IS THIS POWER SUPPLY IS NOT CASE TO BE OPENED AND THERE ARE DANGEROUS USER-SERVICEABLE PARTS INSIDE THE POWER SUPPLY CASE HIGH VOLTAGE IS PRESENT IF DIFFICULTY, NOTIFY APPLE IN COMPUTER INC. OR YOUR DEALER FOR PROMPT SERVICE
POWER SUPPLY 605-5703 SERIAL #061302
apple computer inc.
apple computer inc.
ESC
TAB
CONTROL
SHIFT
CAPS LOCK
DELETE
RETURN
SHIFT
RESET

I am still waiting for a response for a request for a room
I will continue to read the book and wait for a response, If I don't hear back next week I will find another method

10/25
I am still waiting for a final response about what room I can use
I will continue to read the 6502 programming book,

10/18/19
I sent out an email for what room I can use

10/6/19 here are the links to some resources I am using
http://www.appleii-box.de/D06b_resurectionIIe.htm
http://www.classiccmp.org/cini/pdf/Apple/Apple%20II%20(Redbook)%20Reference%20Manual%2030th%20Anniversary.pdf
https://www.willegal.net/appleii/appleii-repair.htm
http://www.appleoldies.ca/anix/Using-6502-Assembly-Language-by-Randy-Hyde.pdf 4-6
http://www.appleoldies.ca/anix/index.htm#lisa
https://archive.org/stream/6502_Assembly_Language_Subroutines/6502_Assembly_Language_Subroutines_djvu.txt

The Beginning

Original Goals

Oct 18th- start learning 6502 assembly and set programming goal
Jan 17th- have all the computers diagnosed and parts ordered
Feb 28th- have all computers fixed
Apr 3rd- be able to write hello world in assembly
May 3rd- be able to have a pixel appear on the screen