CS130 Lab Exercise 1

Creating an 'Icon'

Please go to https://skilldrick.github.io/easy6502/#first-program (Links to an external site.).

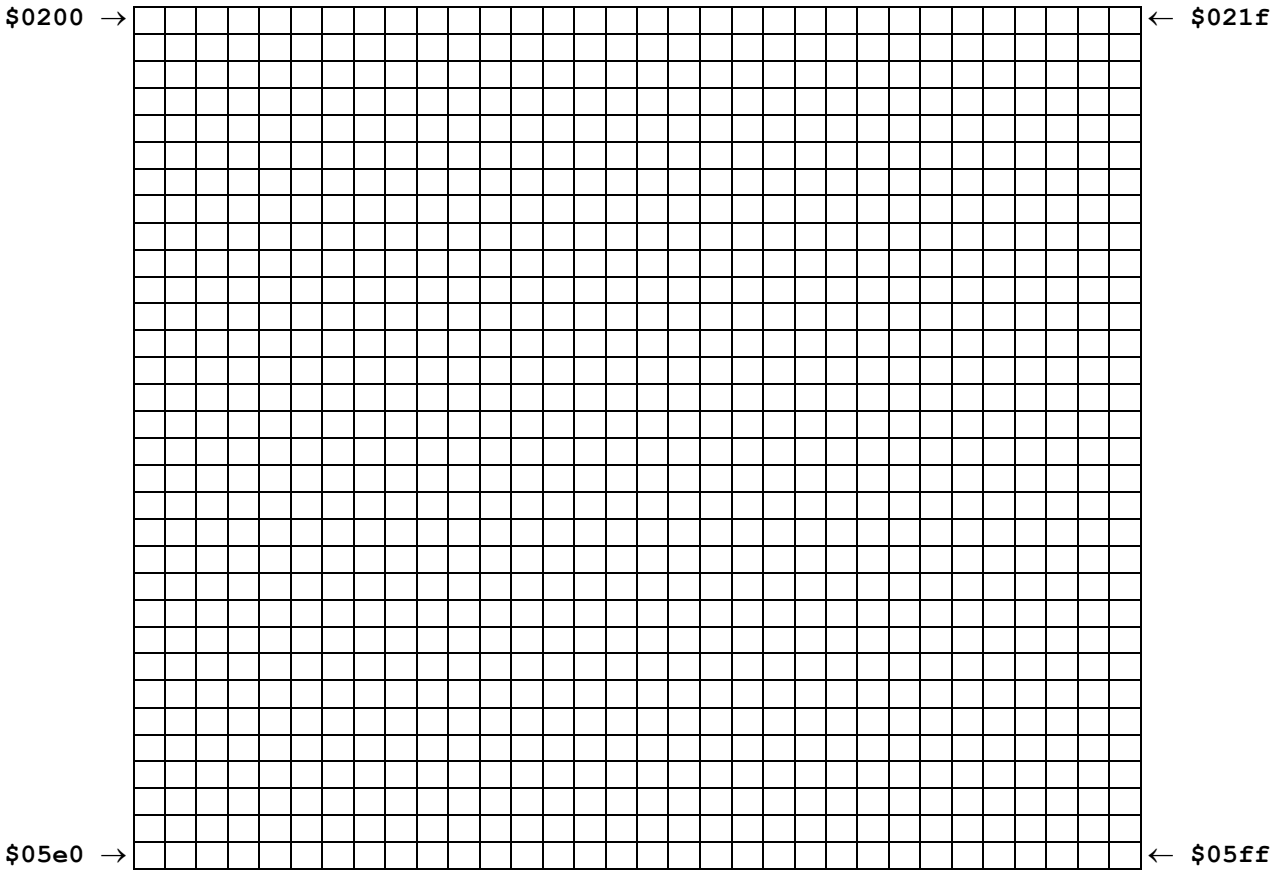
Exercises:

1. Run the program as is. Then, after reading the text, alter the code to change the color of each of the three pixels. Capture a screenshot and paste it into a document labeled: {Your Last Name} CS130 Lab 1.docx (without the curly brackets, of course!)
2. Change the location of one of the pixels to now show at the bottom-right corner (memory location **`$05ff`**). Capture a screenshot and paste it into the document, labeling it Step 2.
3. Create a so-called “Color Palette” of the 16 available colors. Display this palette horizontally, starting with the leftmost pixel positioned at **`$0400`**. Capture a screenshot and paste it into the document, labeling it Step 3.

4a. Clear all instructions and create a ‘font’, representing your own last initial. Size this font as 17 pixels high. Capture a screenshot and paste it into the same document, labeled Step 4.

4b. Submit your “code” as a separate text file e-mailed to me: {Your Last Name} CS130 Lab 1.Txt

Here is a grid that shows the address of the pixels:



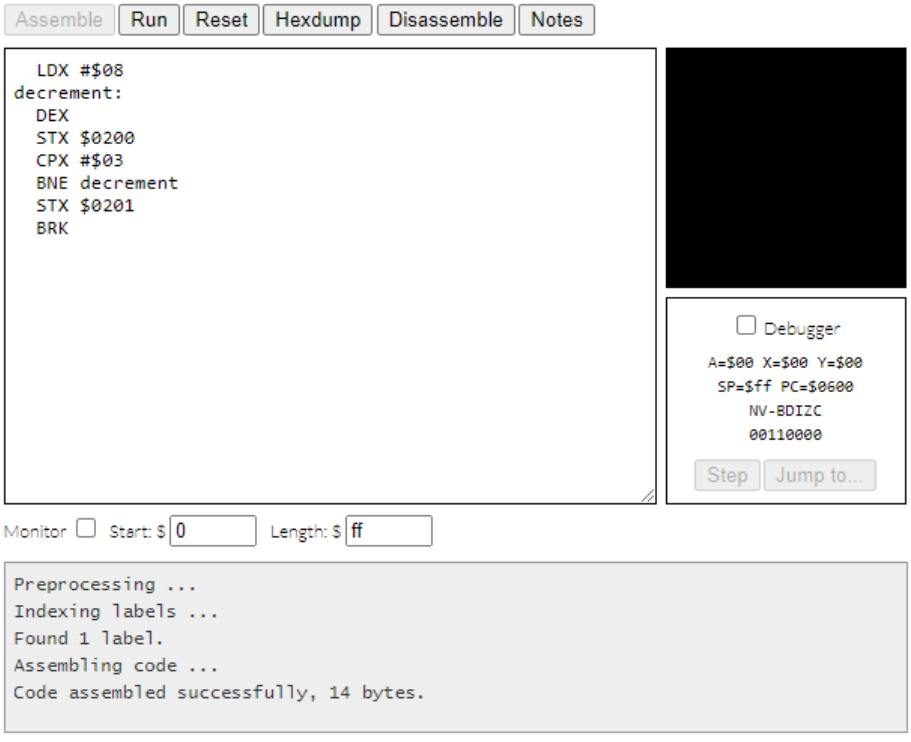
(Continue on the next page.)

Branching

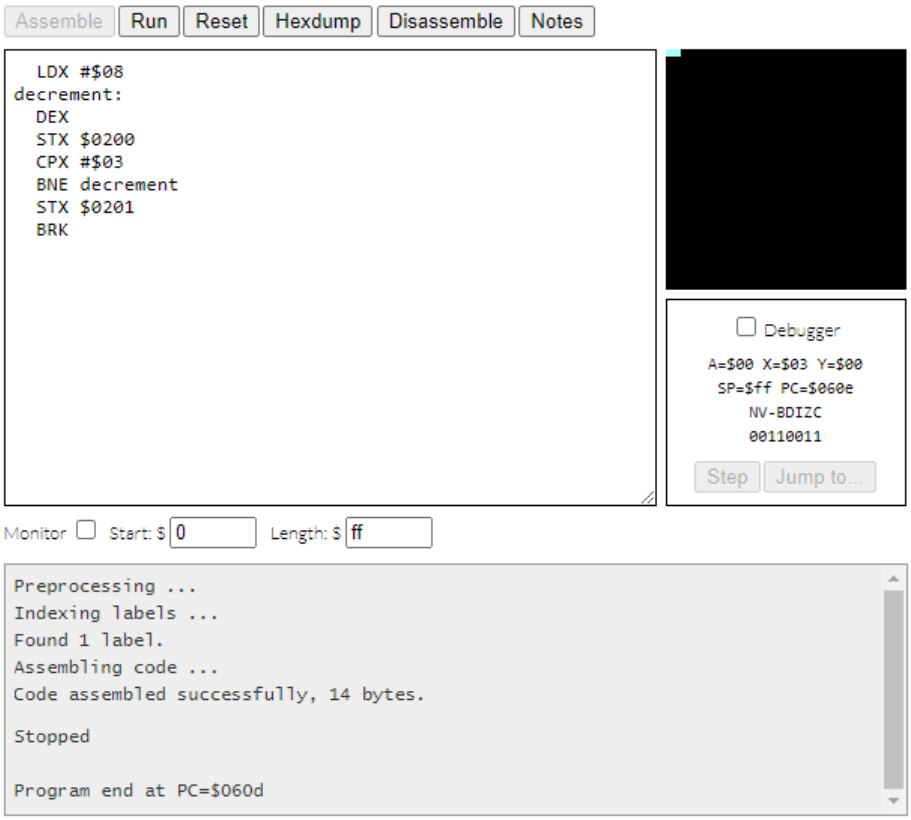
Please go to: https://skilldrick.github.io/easy6502/#branching and step through the program.

Refer also to the document "Instruction Sequence - Branching Example.PDF" that is posted on "Canvas" for Week #1 for the lab exercises (Section 19424, NOT 19423).
So far, we're only able to write basic programs without any branching logic. Let's change that.
The 6502 assembly language has a bunch of branching instructions, all of which branch based on whether certain flags are set or not. In this example we'll be looking at **BNE**: "Branch on Not Equal".

Program before being assembled. Next, click "Assemble"



Program after being assembled. Next, click "Run" to execute this program.



Program after being run. Note Program Counter.

Step-by-Step

To get a better understanding of exactly what is happening, let's start over by first clicking on "Reset". Locate and click the box to the left of the text "Debugger". With each click of the mouse over the button for "Step", you will execute one step of the program. The number of memory locations you progress will, of course, depend upon the instruction being executed.

Watch the "Program Counter", labeled **PC** in the screenshots above. Notice that the memory addresses do not increment the same amount each time but, rather, the increment depends upon where you are in the program. As you step, refer to the "Instruction Sequence" document mentioned above and follow along. You can see that the program ends when the instruction at address **060d** is executed (**BRK**, for "Break"). See the last line of the figure on this page.

What's Happening?

First, we load the value **$08** into the **X** register. The next line is a *label*. Labels just mark certain points in a program so we can return to them later. After the label, we decrement **X**, store it to **$0200** (the top-left pixel), and then compare it to the value **$03**. **CPX** compares the value in the **X** register with another value and sets one or more so-called "Flags". If the two values are equal, the **Z** flag is set to **1**, otherwise it is set to **0**.

The next line, **BNE decrement**, will shift execution to the **decrement** label if the **Z** flag is set to **0** (meaning that the two values in the **CPX** comparison were not equal). Otherwise, it does nothing and we store **X** to **$0201** and then finish the program.

In assembly language, you will usually use labels with branch instructions. When assembled though, this label is converted to a single-byte relative offset (a number of bytes to go backwards or forwards from the next instruction) so branch instructions can only go forward and back around 256 bytes. This means they can only be used to move around local code. For moving further, you will need to use a "jump" instruction. In this program, we go back **8** memory locations as explained in the companion "Instruction Sequence" document.

Exercises

5. Step thru the given program. Watch the **X** value. How many times does it decrement? Add this number to your spiffy submittal document.
6. The opposite of **BNE** is **BEQ**. (i. e. the value in **X** matches the Hex value that follows **CPX** in the previous instruction). Replace **BNE** with **BEQ**. Reassemble and step thru it again. Keep watch of the **X**-register (value). How many times does it decrement now? Include this value in your document submittal.
7. If the count is different, why? (You do not have to write an answer for why; just ponder it.)

If You have extra time

(Not to be submitted; for your benefit only)

a. **BCC** and **BCS** ("Branch on Carry Clear" and "Branch on Carry Set") are used to branch on the Carry flag. Write a program that uses one of these two.

Submit your finished work (document and text file) [ONLY] to RSturlaCS130@GMail.com