CSC317 Project3

Nathaniel Fagrey

December 2019

Introduction

This document outlines how to run the T34 Emulator. It also lists all the functions used and how they were tested. For Part 3 for this Project I wrote all the operations necessary to get to an A for the program, and I want to be graded up to an A.

Running the Program

To run the program:

$ python3 t34.py <objectFile>

If an object file is specified on the command line, it must contain a program in Intel Hex format. Also, the program will only work if python3 is used. Earlier versions of python will not work.

Functions

There is one main class used in this project phase: the Memory class. In this class there were 6 methods:

Memory Class:

- parseUserData - parses the user data entered on the command line and determines what action needs to be done.
- storeData - stores the given data in the correct address in memory.
- getMultipleData - returns data given a start address and an end address.
- storePrograms - stores a program given in a file in a memory address.
- checkMemory - continually checks memory to make sure that the memory bounds are not exceeded or that the address given is out of bounds.
- getData - given an address, it returns a single value for that address. Right now not necessary, but it will be helpful in the future.

RunProgram Class:

This class contains all the methods that carry out the operations for a given opcode. This class is very large has has 100+ methods. This is way too many to put in this document (and for the grader to read), therefore, not all methods will be listed. I will briefly outline the types of methods and what they did. For a given operation such as ADC there could be 8 addressing modes for the operation each with their own special opcode. I therefore divided the opcodes into categories based on their addressing mode:

- Implied category
- Accumulator category
- Immediate category
- Absolute category
- Zero-page category
- Relative category
- indirect category
- Indexed category

For most these addressing modes with the same operation, the way in which the operation worked was very similar. However, I ended up just re-writing pretty much the same code for the operation until I hit the Indexed Category. This category is actually composed of 4 addressing modes (indirect indexed, indexed indirect, absolute indexed, and zero-page indexed). These 4 modes are almost identical except how we determine the address in memory to get data. To avoid re-writing very similar code like I had done for the previous categories, I created a parser method for these 4 modes. This parser determines which of of the modes we are in, what the correct address is, and calls the correct operation for that opcode. This greatly reduced the time and code I needed to complete this part and made me wish I had done something similar for the previous parts.

Testing

Memory

For testing the Memory, the program and functions were tested in two main ways. One was just manually entering data and programs on the command line and printing out that data. This was a quick way to make sure that the store data, print data, and store program functions worked.
However, to test the code more rigorously, a Testing mode was implemented. This mode can be activated by specifying the file TESTING.txt on the command line as follows:

$ python3 t34.py tests/TESTING.txt

TEXTING.txt is previously filled with random data that was then stored in the program's "memory". That data was printed out and checked with the original data to make sure all data was in the correct place and was preserved. To fill TESTING.txt with random data run following command:

$ python3 tests/fillTestingFile.py

Specific tests were also done (most of these were to check edge cases):

- Made sure the values stored were actually stored and remained in memory even after multiple access to memory.
- Made sure that my error checking method would prevent memory allocated over our specified amount or that our addresses could not go beyond 2^{16}.
- Made sure that multiple programs can be given in a file in Intel hex format and stored in memory.

Run a Program

For running a program, I tested running a program in two main ways. First, I compared my output to the examples in the document, making sure they were the same.

Second, I created some specific tests for the more complicated operations, creating my own programs, and confirming that the outputs were what I expected them to be. I actually caught several bugs doing this. Some of the specific tests were:

- Made sure that the ASL preformed correctly
- Made sure the ROR op worked
- Made sure the ROL op worked
- Checked the PLA, PHA, PLP, and PHA methods work correctly
- Made sure that the data was actually added to memory, and stayed there after the run program method exited exited.
- Made sure that LSR and carry bit worked
- Made sure that the registers still contained their values
- Confirmed that multiple programs could be loaded in and run, and they would not affect each other.
- created a test for all the indirect indexed because there was no tests given for them in the document

For the final part of the project, I added a python script that would allow you to run every function in the RunProgram class just to make sure that all functions could be interpreted.

$ python3 doAllFunctions.py

I also added specific tests for some of the more generic operations like Add, Sub, ADC, SBC, my sign extend function, etc, to make sure they could handle different values and still output what I expected. To run these tests:

$ python3 testOperations.py

I also used pretty diff online compare my results with the ones given in the document.