CSC 317 Project 2 Report

Nathaniel Fagrey

October 2019

Introduction

This document outlines how to run the T34 Emulator. It also lists all the functions used and how they were tested.

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

- def operation: This is the main function for the Run Program operation. It reads in the opcodes from a given address.
- def printHeader: Prints the header for the program outputs.
- def printInstruction: Will print the PC address, Instruction, Opcode, Addressing mode, Operands, and Register values for each program command.
- def doASLOpA: Left shifts the Accumulator register.
- def doBRKOpI: Breaks program
- def doCLCOpI: Clears the carry flag.
- def doCLDOpI: Clears decimal flag.
- def doCLIOpI: Clears Interrupt flag.
- def doCLVOpI: Clear overflow.
- def doDEXOpI: Decrement the X register.
- def doDEYOpI: Decrement the Y register.
- def doINXOpI: Increment the X register.
- def doINYOpI: Increment the Y register.
- def doLSROpA: Shift one bit right in the Accumulator.
- def doNOPOpI: No operation.
- def doPHAOpI: Push Accumulator onto the stack.
- def doPHPOpI: Push Status Register onto the stack.
- def doPLAOpI: Pull Accumulator from the stack.
- def doPLPOpI: Pull Status Register from the stack.
- def doROLOpA: Push MSB of Accumulator into the carry, and set LSB of Accumuator with the previous value of the carry.
- def doROROpA: Push LSB of Accumulator into the carry, and set MSB of Accumuator with the previous value of the carry.
- def doSECOpI: Set carry flag.
- def doSEDOpI: Set decimal flag.
- def doSEIOpI: Set interrupt flag.

- def doTAXOpI: Transfer Accumulator to Register X.
- def doTAYOpI: Transfer Accumulator to the Register Y.
- def doTSXOpI: Transfer Stack pointer to Register X.
- def doTXAOpI: Transfer Register X to Accumulator.
- def doTXSOpI: Transfer Register X to Stack pointer.
- def doTYAOpI: Transfer Register Y to Accumulator.
- def doFlags: Given a register, this method sets the flags that are needed, and returns the register, making sure it at 8-bits.
- isNegative: Will return true if a the value in a given register is negative, false otherwise.
- clearTopBits: Cleared the top 32 bits of a register to mock actual 8-bit registers.

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 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 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.