T34 Emulator

Adam Kraus

December 12, 2021

1 T34 Program

This program is an emulator for the 6502. It has most of the functionality implemented, not including indexed addressing modes.

1.1 Monitor Functions

The monitor has the ability to do the following:

1.1.1 Run Program

If desired, you can start running the emulator at a location.
Ex: "200R" runs the emulator starting at address $0200.

1.1.2 Display Memory Range

You can display a range of memory locations with at most 8 bytes per line.
Ex. "200.20F" prints the memory values at [200, 20F).

1.1.3 Edit Memory Locations

If no file is supplied or you want to manually set memory values, you can do that.
Ex: "200: A9 B4" sets the memory at address $0200 to $A9 and at $0201 to $B4.

1.1.4 Display Memory Location

You can display a singular memory value.
Ex: "200" displays the value at address $0200.

1.1.5 Exit Monitor

You can exit the monitor if finished running programs.
Ex: "exit"

1.2 Addressing Modes

These are the currently implemented addressing modes. More information on the instructions and what they perform can be found in the Appendix B for the T34.

1.2.1 Implied

These instructions mainly consist of transferring values between registers and setting or clearing values in the status register.

Ex: TAY transfer the accumulator into the Y register.

1.2.2 Accumulator

These instructions act directly on the accumulator.

Ex: ASL A shifts the accumulator left 1 bit.

1.2.3 Immediate

These instructions take in an additional byte to use.

Ex: LDY $FF loads $FF into the y register.

1.2.4 Relative

These instructions take in an additional byte to use for branching. The byte is signed, so the offset is between -127 and 128.

Ex: BCC $04 branches 6 bytes forward if the carry flag in the status register is not set.

1.2.5 Absolute

These instructions take in two additional bytes to determine where to execute.

Ex: INC $00 $03 increases the value in memory at location $0300

1.2.6 Zero Page

The zero page is the first 256 bytes in memory, being 0000 to 00FF. The zero page instructions take in an additional byte to determine where in the zero page to execute.

Ex: ORA $03 takes the bitwise OR of the value at address $03 and the accumulator, and stores it back in the accumulator.

1.2.7 Indirect

The only instruction that uses indirect addressing is JMP. It takes in two additional bytes to calculate and address. The program counter is then set to the address stored at the location, not from the additional bytes.

Ex: JMP $FF $11 jumps to the address stored at the address $11FF.

2 Functions

2.1 sublists chunks(list, n)

Used by displayMemRange to split a list into sublists of a certain size. Returns the list of sublists.

2.2 void displayMem(loc)

Takes in an address location and prints out the corresponding memory value at that location.

2.3 void displayMemRange(start, end)

Takes in a beginning and ending address and displays them with at most 8 memory locations per row. Prints out from [start, end).

2.4 void editMem(loc, newMem)

Takes in a starting memory location and new memory values starting at that location.

2.5 void runProg(loc)

Takes in an address location for where to start running the program. Fetches the opcode from that address, and executes it. Keeps interpreting instructions at the program counter until the break flag is set.

2.6 value set_bit(value, bit)

Takes in an integer and a bit location. Returns the integer with the bit at that location set to a 1.

Ex: set_bit(0000, 3) = 0100

2.7 value clear_bit(value, bit)

Takes in an integer and a bit location. Returns the integer with the bit at that location set to a 0.

Ex: clear_bit(1111, 3) = 1011

2.8 value check_bit(value, bit)

Takes in an integer and a bit location. Returns true if that bit is a 1, false if it is a 0

Ex: check_bit(0100, 3) = true

2.9 pc_change interpret()

Interprets instruction starting at the current program counter. Prints out operand, instruction name, addressing mode, operands, x and y register, accumulator, stack pointer, and status register after completing each instruction. Returns the change to the program counter for branching and instructions that use additional bytes.

2.10 void file_input()

Handles input from object file if supplied. Validates file integrity using checksum involving address, byte count, and record type. Only accepts records of type 00 and 01.

2.11 main()

Calls file_input, then handles input from the monitor. Calls appropriate function based on command input from the monitor.

3 Final Grading

I wish to be graded for a B.

3.1 B Programs

Both programs swap values between the accumulator and a memory location. They both have a value in memory at $0200, which is 1F. Both programs are located in the programs directory in the repository.

3.1.1 Swapping with X temporary

The first program swaps values between A and a memory location using the X register as a temporary placeholder. Below is the pseudo assembly instructions for the program.

0300: LDA #58
0302: JSR $1000
1000: TAX
1001: LDA $0200
1004: STX $0200
1007: RTS
0305: BRK

Once you jump to the subroutine, it transfer A into X, loads the memory value into A, and stores the value of X back into memory.

This program is called swapx.obj, and is located in the programs folder.

3.1.2 Swapping with zero page temporary

The second program swaps values between A and a memory location using the zero page as a temporary placeholder. Below is the pseudo assembly instructions for the program.

0300: LDA #58
0302: JSR $1000
1000: STA $00
1002: LDA $0200
1005: STA $01
1007: LDA $00
1009: STA $0200
100C: LDA $01
100E: RTS
0305: BRK

This program uses the space of the accumulator, and 2 zero page locations. It stores A at $0000 in the zero page, then loads the memory location into X, and stores it back into the zero page at $0001. It then reloads A with its original value from the zero page at $0000, and stores it into memory. Finally it loads the original memory value from the zero page into A.

This program is called swapzpg.obj, and is located in the programs folder.

4 Running Program

Program is run from command line using python3.

Ex: python3 ./t34.py code2_2.obj
Ex: python3 ./t34.py test1.obj
Ex: python3 ./t34.py