Program #1 – Emulator

William Doering

Note to Grader

I would like to be graded at a C level. I do have a few B level opcodes done. I will give them below in case they can help my grade or help with your scripts.

Indirect: JMP
Relative: `BCC, BNE`
Absolute: `ASL, JMP, JSR, LDA, LSR, ROL, ROR`

Running the Program

To run the program, you must first compile it using the command given below while in the program's directory. It will take an optional object file formatted using the Intel HEX file format. It will load this file into the emulator's memory. After it is running it will have 4 separate functions along with an exit command.

```
$ python3.6 main.py [file.obj]
```

Read Single

To read the value in a specific memory space, type the address to that space in hex format. It will output the address you typed along with the value stored there.

```
> 200
200   h4
```

Read Multiple

To read the values in a range of memory, type the starting address followed by a period and then the ending address in hex. This will output the values in chunks of 8. There will be an address at the beginning of each chunk for ease of viewing.

```
> 200.20d
200   h4 7a 99 00 ab cd 74 e4
208   00 69 00 ba 04 20
```

Assign

To assign value(s) to memory, type in the starting address followed by a colon. Then type in the new value for each following address in order, separated by space.

```
> 300: aa ab ac ad ae af a1 a2 a3 a4
```

Run

To run a program, type the start location of the program, followed by an ‘R’. It will then run the program until it encounters an interrupt (interrupt flag can be set by opcode functions or invalid/unsupported opcode). Sample output below.

```
> 400: 88 e8 98 0a 2a 48 8a 6a a8 68 aa 18 00
> 400R
PC   OPC  INS   AMOD OPRND  AC XR YR SP NV-BDIZC
400  88   DEY   impl -- --  00 00 FF FF 10100000
401  E8   INX   impl -- --  00 01 FF FF 00100000
402  98   TYA   impl -- --  FF 01 FF FF 10100000
403  0A   ASL      A -- --  FE 01 FF FF 10100001
404  2A   ROL      A -- --  FD 01 FF FF 10100001
405  48   PHA   impl -- --  FD 01 FF FE 10100001
406  8A   TXA   impl -- --  01 01 FF FE 00100001
407  6A   ROR      A -- --  80 01 FF FE 10100001
408  A8   TAY   impl -- --  80 01 80 FE 10100001
409  68   PLA   impl -- --  FD 01 80 FF 10100001
40A  A8   TAY   impl -- --  FD FD 80 FF 10100001
40B  18   CLC   impl -- --  FD FD 80 FF 10100000
40C  00   BRK   impl -- --  FD FD 80 FC 10110100
```

Exit

To exit, type exit and press enter. There needs to be a space on either side of the word or it needs to be on a new line with a space following it to work. End of file will stop the running emulation, but not the program.

Testing

For testing, I first used the files given and the sample commands in Appendix A of the assignment. I then redirected input from a text file where I wrote test cases for every function. In this manner, I was able to implement unit testing for the opcodes.

Functions

Below is a list of the functions in the program with their descriptions.

main()

This starts the program. It calls the parse file and gets user input. It then passes it on to evaluate.

evaluate()

This evaluates the user input and determines what command the user is giving. It then calls the appropriate function for that command.

print_one()

This will print the value at the address given.

print_range()

This will print the values at the range of addresses given.

edit_mem()
This will edit the values at and after the address given.

parse_file()
Parses the incoming file and stores it in memory

scream_and_die()
Output closing message and exit the program.

prog_run()
This runs the program. It will start at the given program counter and print out the program’s state for each operation. When the program has it’s interrupt flag set, it will end execution of the emulated program.

op_print()
Prints the status of the registers after each operation.

inv_error()
This is an error function used for development, testing and error handling. If an unimplemented op code is given, it will return with “Invalid Operation ##”.

sign_add()
Adds two signed bytes together, along with a carry flag. Returns results and a status byte.

sign_sub()
Subtracts two signed bytes from eachother. Returns results and a status byte.

class Memory
This holds all the memory and registers for the emulator.

Table opc_table
This table holds all the operations indexed by their op codes. This allows for quick lookup and execution of the code.

Opcode Operation Functions

I put all the opcode operations in a separate file called “operations.py”. This contains a function for every opcode supported by the emulator. Here I list the opcode, it’s function, and it’s addressing mode.

brk()

Opcode 00: Force Break. This uses the implied addressing mode.

ora_zpg()

Opcode 05: OR Memory with Accumulator. This uses the zeropage addressing mode.

asl_zpg()

Opcode 06: Shift Left One Bit. This uses the zeropage addressing mode.

php()

Opcode 08: Push Processor Status on Stack. This uses the implied addressing mode.

ora_imme()

Opcode 09: OR Memory with Accumulator. This uses the immediate addressing mode.

asl_a()

Opcode 0A: Shift Left One Bit. This uses the accumulator addressing mode.

clc()

Opcode 18: Clear Cary Flag. This uses the implied addressing mode.

jsr()

Opcode 20: Jump to New Location Saving Return Address. This uses the indirect addressing mode.

and_zpg()

Opcode 25: AND Memory with Accumulator. This uses the zeropage addressing mode.

rol_zpg()

Opcode 26: Rotate One Bit Left. This uses the zeropage addressing mode.

plp()

Opcode 28: Pull Processor Status from Stack. This uses the implied addressing mode.

and_imme()

Opcode 29: AND Memory with the Accumulator: This uses the immediate addressing mode.

rol_a()

Opcode 2A: Rotate One Bit Left This uses the accumulator addressing mode.

sec()

Opcode 38: Set Carry Flag. This uses the implied addressing mode.

eor_zpg()

Opcode 45: Exclusive-OR Memory with the Accumulator. This uses the zeropage addressing mode.

lsr_zpg()
Opcode 46: Shift One Bit Right in Memory. This uses the zeropage addressing mode.

pha()
Opcode 48: Push Accumulator on Stack. This uses the implied addressing mode.’

eor_imme()
Opcode 49: Exclusive-OR Memory with the Accumulator. This uses the immediate addressing mode.

lsr_a()
Opcode 4A: Shift One Bit Right. This uses the accumulator addressing mode.

jmp_abs()
Opcode 4C: Jump to new location. This uses the accumulator addressing mode.

cli()
Opcode 58: Clear Interrupt Disable Bit. This uses the implied addressing mode.

rts()
Opcode 60: Return form subroutine. This uses the implied addressing mode.

ror_zpg()
Opcode 66: Rotate one bit right. This uses the zeropage addressing mode.

pla()
Opcode 68: Pull Accumulator from stack. This uses the implied addressing mode.

adc_imme()
Opcode 69: Add memory to the accumulator with the carry. This uses the implied addressing mode.

ror_a()
Opcode 6A: Rotate One Bit Right. This uses the accumulator addressing mode.

jmp_ind()
Opcode 6C: Jump to a new location. This uses the indirect addressing model.

adc_abs()
Opcode 6D: Add memory to the accumulator with the carry. This uses the absolute addressing mode.

sei()
Opcode 78: Set Interupt Disable Status. This uses the implied addressing mode.

sty_zpg()
Opcode 84: Store index Y in memory. This uses the zeropage addressing mode.

sta_zpg()
Opcode 85: Store the accumulator in memory. This uses the zeropage addressing mode.

stx_zpg()
Opcode 86: Store index X in memory. This uses the zeropage addressing mode.

dey()
Opcode 88: Decrement Index Y by One. This uses the implied addressing mode.

txa()
Opcode 8A: Transfer Index X to Accumulator. This uses the implied addressing mode.

sta_abs()
Opcode 8D: Store the accumulator in memory. This use the absolute addressing mode.

bcc()
Opcode 90: Branch on carry clear flag. This uses the relative addressing mode.

tya()
Opcode 98: Transfer Index Y to Accumulator. This uses the implied addressing mode.

txs()
Opcode 9A: Transfer Index X to Stack Register. This uses the implied addressing mode.

ldy_imme()
Opcode A0: Lead index Y with memory. This uses the immediate addressing mode.

ldx_imme()
Opcode A2: Load index X with memory. This uses the immediate addressing mode.

ldy_zpg()
Opcode A4: Load index Y with memory. This uses the zeropage addressing mode.

lda_zpg()
Opcode A5: Load the accumulator with memory. This uses the zeropage addressing mode.

ldx_zpg()
Opcode A6: Load index X with memory. This uses the zeropage addressing mode.

tay()
Opcode A8: Transfer Accumulator to Index Y. This uses the implied addressing mode.

lda_imme()
Opcode A9: Load the accumulator with memory. This uses the immediate addressing mode.

tax()
Opcode AA: Transfer Accumulator to Index X. This uses the implied addressing mode.

lda_abs()
Opcode AD: Load the accumulator with memory. This uses the absolute addressing mode.

clv()
Opcode B8: Clear Overflow Flag. This uses the implied addressing mode.

tsx()
Opcode BA: Transfer Stack Pointer to Index X. This uses the implied addressing mode.

cpy_imme()
Opcode C0: Compare memory and index y. This uses the immediate addressing mode.

cpy_zpg()
Opcode C4: Compare memory with index y. This uses the zeropage addressing mode.

cmp_zpg()
Opcode C5: Compare memory with the accumulator. This uses the zeropage addressing mode.

dec_zpg()
Opcode C6: Decrement memory by one. This uses the zeropage addressing mode.

iny()
Opcode C8: Increment Index Y by One. This uses the implied addressing mode.

cmp_imme()
Opcode C9: Compare memory with the accumulator. This uses the immediate addressing mode.

dex()
Opcode CA: Decrement Index X by One. This uses the implied addressing mode.

bne()
Opcode D0: Branch on result not zero (zero flag). This uses the relative addressing mode.

cld()
Opcode D8: Clear Decimal Mode. This uses the implied addressing mode.

cpx_imme()
Opcode E0: Compare memory and index X. This uses the immediate addressing mode.

cpx_zpg()
Opcode E4: Compare memory and index X. This uses the zeropage addressing mode.

sbc_zpg()
Opcode E5: Subtract memory from the accumulator with the carry. This uses the zeropage addressing mode.

inc_zpg()
Opcode E6: Increment memory by one. This uses the zeropage addressing mode.

inx()
Opcode E8: Increment Index X by One. This uses the implied addressing mode.

nop()
Opcode EA: No Operation. This uses the implied addressing mode.

sed()
Opcode F8: Set Decimal Flag. This uses the implied addressing mode.