Program #1 – Emulator

William Doering

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 (either a break 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. The other two forms of exit are not implemented.

Testing

For testing, I first used the files given and the sample commands in Appendix A of the assignment. I then made a random file using the same format as the test files and typed in random commands to see that it works in general. There is no error checking on input so the commands need to be typed correctly.

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.

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 ##".

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.

brk()

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

php()

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

plp()

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

pha()

Opcode 48: Push Accumulator on Stack. This uses the implied addressing mode.

lsr_a()

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

cli()

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

pla()

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

ror_a()

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

sei()

Opcode 78: Set Interupt Disable Status. This uses the implied 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.

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.

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

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

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

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

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