Running:
The program is run by invoking an installation of Python3, followed by Monitor.py, followed by an optional intex HEX filename.
EX:
Python3 Monitor.py {test1.obj}

Functions:

Monitor.main : Gets user input, translates to values actually usable to call functions, then calls those functions on the emulator.

Monitor.prompt_for_input : get a normal value from user and return true, or get an exit code and return false

Memory.parse_intel_hex : loads a given file, interprets it as an intex hex file, and then passes the parsed values to the memory setting function

Memory.display_registers : prints the contents of the special registers in the format specified by the project writeup.

Memory.display_register_headers : print the formatted header which labels the content from the registers

Memory.initalize_registers : resets all special registers to default values

Memory.save_values_to_memory : takes in a starting location and a list of memory values, then stores those values into memory starting at the starting location

Memory.display_values_from_range : display the data in memory from the eight floored starting value to the ending value. Format with 8 values per line and the memory location as a tag before the eight values.

Memory.display_single_values : display the data within memory at the given location

Memory.push_to_stack : push a value to the stack and decrement the stack pointer

Memory.pop_from_stack : pop a value off the stack and increment the stack pointer

Helper.get_hex_string_from_decimal_number: convert a base 10 value to an appropriately formatted hex string

Helper.get_decimal_number_from_hex_string : get the decimal values of a string interpreted as hexadecimal

Helper.get_decimal_int_from_signed_byte : convert a base 10 integer to a signed byte

Helper.get_signed_byte_from_decimal_int : convert a signed byte to a base 10 int

Helper.get_twos_compliment : converts a signed byte to its opposite signed value

Helper.get_ones_compliement: converts a signed byte's bits to be used with compares

Helper.combine_bytes: turns a lo_byte and hi_byte in a decimal int

Processor.ALU : process a given opcode using a decode table

Processor.clear_status_bits : clear the selected bit from the status register

Processor.is_negative : return if a signed byte is negative and set the bit in the status register according

Processor.is_zero : return is signed byte is zero and set the bit in the status register accordingly

Processor.execute_at_location : retrieve and execute instructions starting at a given address, Incrementing the program counter as needed

Processor.generic_store : store a register in a specified memory location

Processor.generic_load : fill a register with a value determined by the addressing mode

Processor.generic_compare : set status bits according to the comparison between a given register and value

Processor.add : add/subtract two signed bytes and set status bits as needed

Processor.generic_branch : branch to a given location if the specific status bit is low/high

Processor.resolve_params : given an addressing mode and parameters sent to the alu, convert to more friendly values (add offsets, etc.)

Processor.get_additional_args : given an addressing mode, adjust the PC to the next instruction and read the needed parameters

Testing:

Tests are located in test_functions.py. Static methods and conversions were tested to ensure no data was lost going back and forth.
All example runs of the program provided by the documentation were directly translated to tests. Command line object loading was intended to be added, but the intended implementation would be operating system specific, and therefore testing accuracy would be ambiguous. Manual testing of all example test cases was performed before submission.

Grading: I would like to be graded for an A level. The compare functions are all tied to Processor.generic_compare(), and are my only concern for failure. They may be fixed depending upon when this is pulled.