class CPU

Constant Class Attributes

| MAX = 9999 | Sets the maximum sized "valid" instruction |
| --- | --- |
| MIN = -9999 | Sets the minimum sized "valid" instruction |
| ACCUMULATOR\_DEFAULT = 0000 | Sets the Default Accumulator value when booting up |
| MAX\_INSTRUCTION\_LIMIT = 100 | Forces the CPU to Stop when running more than 100 consecutive commands. Is configurable but useful for testing |

Attributes

| Accumulator : Float | Holds the arthimatic value between instructions  Persists between instructions |
| --- | --- |
| Register : Int | Variable used for all instructions to be read from, or any data to be loaded. Does *not* persist between instructions |
| Pointer : Int | Variable used to hold the address that will read the next instruction from |
| Halted : Boolean | Variable used to see if the CPU has halted |
| Memory : Memory Object | Variable used to reference the Memory object |

Methods

**boot\_up()**

Used to reset the attributes to the constant default values

**run()**

Used to 'start' the CPU and will continuously read/run instructions and update the pointer until a halt command is hit

Raises ValueError : If operation() method cannot run the instruction given to it

Raises ValueError : If Memory cannot read from the pointer's address a valid instruction

Raises Halt : If Halt instruction is ran

**decypher\_instruction(word)**

Splits the instruction into the first two digits and last digits (operator/operand)

Return : Returns a tuple of the operator and operand. If a 4-digit instruction is given, the operator will be the first 2 digits and the operand will be the last 2 digits

Raises ValueError : If a negative operation is attempted to split

Raises ValueError : If operation is not in valid range (not 4 digits or under 1000)

**operation(word)**

Runs the decypher\_instruction() method

Takes the operator/operand combo and runs the operator through a match switch.

Then runs the correct operation with the corect operand

Raises Halt : If Halt instruction is given

Raises ValueError : If operator that is given does not correspond with a valid operation

**read\_from\_memory(addresss)**

Used for access to the Memory class. Uses memory to set the value at an address into the register

**load\_to\_memory(address, value)**

Transition method to put data in the register into memory

**op\_READ(operand, gui)**

Reads from the gui given to get a value.

Places that value into the memory adress given by the operand

**op\_WRITE(operand, gui)**

Gets value from the memory address given by the opearnd

Prints value to the screen via the gui

**op\_LOAD(operand)**

Loads the value at the address/operand location into the accumulator

**op\_STORE(operand)**

Loads the value of the accumulator into the address/operand location in memory

**op\_ADD(operand)**

Loads the value at the address/operand location into the register

Adds the register and the accumulator and leaves value in the accumulator

**op\_SUBTRACT(operand)**

Loads the value at the address/operand location into the register

Subtracts the register from the accumulator and leaves value in the accumulator

**op\_MULTIPLY(operand)**

Loads the value at the address/operand location into the register

Multiplies the register and the accumulator and leaves value in the accumulator

**op\_DIVIDE(operand)**

Loads the value at the address/operand location into the register

Divides the register from the accumulator and leaves value in the accumulator

**op\_BRANCH(operand)**

Moves the pointer to the given value as the operand

**op\_BRANCHNEG(operand)**

Moves the pointer to the given value as the operand IF the accumulator is currently negative

**op\_BRANCHZERO(operand)**

Moves the pointer to the given value as the operand IF the accumulator is currently ZERO

**op\_HALT()**

Raises Halt

This cascades up into the run() method which then ends the CPU

class Memory

Attributes

size : Int

The total amount of memory cells that can be accessed

memory : Array of String

The reference to the array of all the memory cells

Methods

**validate\_address(address)**

Validates whether an address is within bounds or not

Raises IndexError : If address is out of bounds

**@staticmethod**

**validate\_word(word)**

static method used for determining if a word is valid to be inputted into memory

Raises ValueError : If the length of the word is NOT 5 - Must be 4-digits and +/- sign

Raises ValueError : If the first character of the word is neighter "+" or "-"

Raises ValueError : If the other 4 characters are not digits

Raises ValueError : If the word is not considered a string

**read(address)**

Used for the CPU to read a specific value from a specific address

Address is validated before performing using validate\_address() method

**write(address, word)**

Used for the CPU to write a specific value to a specific addres

Address is validated before performing using validate\_address() method

Word is validated before performing using validate(word)

**clear()**

Resets all memory cells to "+0000"

**\_\_str\_\_()**

Creates a single string that shows the entire memory and all its values

**word\_to\_int(word)**

Static method : used for converting a word to an integer when passing between CPU and Memory

**int\_to\_word(number)**

Static method : Used for converting an integer to a word when passing between Memory and CPU

Raises ValueError if Number isn't in range (-9999 to 9999)

class Bootstrapper

Attributes

Memory : Memory Object

CPU : CPU Object

Methods

**load\_program(file\_name)**

Opens the file at the given file\_name

Raises IndexError : If a file is too large, and too much memory is tried to be written, only the first 100 lines are accepted into memory and returns successfully

Raises ValueError : If an incorect instruction is read, instead of failing the value "+0000" is written to memory instead and then continues on

**run\_program**

Runs the CPU run method

class App

Methods

**load\_file(file\_path / Optional)**

If a file a path is not given, a dialog box is given to allow the user to pick a file to load

Once a file path is given, or file picked to load, it is then read and placed in the program\_text field

**clear\_program()**

Clears the program\_text field to allow a new file to be loaded

**load\_memory()**

Takes the text from the program\_text field and writes all the values into the memory module

Raises IndexError : If the index is larger than the memory it simply returns and stops loading. All valid instructions beforehand are kept

Raises ValueError : If an invalid instruction is read into memory it returns and stops loading.

All valid instructions beforehand are kept.

**adjust\_memory\_font\_size()**

Dynamically adjusts the font size based on the widget's size

**update\_memory\_text()**

Used to update the gui text fields after the CPU has updated in some way

All values such as the pointer, accumulator that are seen in the GUI are updated to their real-time values

**run\_program()**

Runs the bootstrapper module to run the CPU class with the given memory and values

**step\_program()**

Reads one operation at a time, and updates the text by running update\_memory\_text()

**reset\_program()**

Resets the program and all labels/text widgits to default values

**instruction\_window()**

Displays the instructions set window with the text of all the currently valid instructions and what each instruction does

