Library Documentation

<Interpreter> *Interpreter.h*

The Interpreter takes the input of the program, runs error checks to make sure it is valid input then puts it into memory.

Int numbers - input is stored once it is saved as an integer.

string word - this is where the argument is stored once it is taken in as a string

Int stopCase -What that the Interpreter looks for to halt taking the programming

Int incrementor - this counts how many words get put into memory

Int wordLength - so it only takes words that are this same length

allDigits - this is true after the first character is off and the rest of the word are digits

vmMemory\* memory = vmMemory::getInstance();- a pointer to access the memory class

maxMemory - int, the maximum amount of words that can be put into memory.

Precondition:

Postcondition: All the program input has been checked for errors and input into memory.

<Memory> *Memory.h*

It reads and writes words from and to memory. The words manipulated are represented by integers of four digits each.

Precondition:

Postcondition:

Functions:

< *int write(int address, int\* data);* >

*write:* It writes a word to a designated location in memory

Returns zero

< *int read(int address, int\* data);* >

read: It reads a word from a designated location in memory.

Returns zero

< *int write\_multiple(int address, int\* data, int length);* >

write\_multiple: It writes memory addresses with integers at data address + offset

< *void dump(void);* >

dump: It prints out the entire memory array

No return

<Executor> *Executor.h*

It uses the read and write Memory class capabilities to access the words in memory, and to extract the opcode and operand.

Functions:

< *string GetOpcode();* >

GetOpcode: Reads a word from memory and extracts the instruction opcode within the word.

Returns the opcode as a string.

< *int GetOperand();* >

GetOperand: Reads a word from memory and extracts the instruction operand within the word.

Returns the operand as an integer value.

< *void CallInvoker();* >

CallInvoker: It Calls the *execute* member function in the Invoker class and uses the instruction opcode and operand to be executed by the virtual machine. After the execution of a program the dump member function in the Memory class is called to display the entire memory.

No return

<Invoker> *Invoker.h*

It executes the instructions fetched from memory.

Functions

< *int executeCommand(string code, int argument);* >

exectueCommand: takes the opcode and the operand for the execution of the respective instruction defined in the instruction set.

Returns zero if instruction is a valid instruction

Returns negative one if instruction is not valid

<OpCode> *OpCode.h*

*Precondition:*

*Postcondition:*

*Functions:*

<OpBranch> *OpBranch.h*

*Precondition:* The PC counter will be at a location.

*Postcondition: The PC Counter will branch to a specified location.*

*Functions:*

*void OpBranch::execute(int argument).*

<OpBranchNeg> *OpBranchNeg.h*

*Precondition:* The accumulator will have a negative value stored in it.

*Postcondition: T*he PC counter will branch to a specific location.

*Functions:*

*void OpBranchNeg::execute(int argument)*

<OpBranchZero> *OpBranchZero.h.*

*Precondition:* The accumulator will have a value of zero stored in it.

*Postcondition:* The PC counter will branch to a specific location.

*Functions:*

*void OpBranchZero::execute(int argument)*

<OpAdd> *OpAdd.h*

*Precondition:* A value in the accumulator.

*Postcondition: Add a word form a location in memory from the value in the accumulator and left in the accumulator.*

*Functions:*

*void OpAdd::execute(int argument)*

<OpDivide> *OpDivide.h*

*Precondition:* A value stored in the accumulator.

*Postcondition:* The value stored in the accumulator will be divided by a word from a location in memory and left in the accumulator.

*Functions:*

void OpDivide::execute(int argument)

<OpMultiply> *OpMultiply.h*

*Precondition:* A value stored in the accumulator.

*Postcondition:* The value stored in the accumulator will be multiplied by a word from a location in memory and left in the accumulator.

*Functions:*

void OpMultiply::execute(int argument)

<OpLoad> *OpLoad.h*

*Precondition: A word in a location of memory*

*Postcondition: Loads a word from a location in memory into the accumulator.*

*Functions:*

*void OpLoad::execute(int argument)*

<OpStore> *OpStore.h*

*Precondition: A word in the accumulator*

*Postcondition: Stores the word in the accumulator into a location of memory.*

*Functions:*

*void OpStore::execute(int argument)*

<OpSubtract> *OpSubtract.h*

*Precondition:* A value in the accumulator

*Postcondition: Subtract a word form a location in memory from the value in the accumulator and left in the accumulator.*

*Functions:*

*void OpSubtract::execute(int argument)*

<OpRead> *OpRead.h*

*Precondition:* this class takes a integer and a string as an argument.

*Postcondition: Read a word from input unto a location in memory*

*Functions:*

*this->memory->write(arguement, &inputWord);*

<OpWrite> *OpWrite.h*

*When the Opcode is 11, this class will take that argument and write it into memory*

*Precondition: A word in a location of memory.*

*Postcondition: Write the word from a specific location in memory to the screen.*

*Functions:*

*void OpWrite::execute(int argument)*

*CLASS BranchOpCode*

This is an abstract class, it defines the opcode classes to be used by the invoker.

*Precondition: A word in a location of memory,*

*Postcondition: Write the word from a specific location in memory to the screen.*

*Functions:*

vmMemory\* memory – a pointer to memory to read from memory

*virtual std::string getCode()*

*Class BranchInvoker*

*Precondition:*

*Postcondition:*

*Functions:*

*Class Fecade*

*Precondition:*

*Postcondition:*

*Functions:*

*Class GUIOpCode*

*Precondition:*

*Postcondition:*

*Functions:*

*Class LastInvoker*

*Precondition:*

*Postcondition:*

*Functions:*

*Class MathInvoker*

*Precondition:*

*Postcondition:*

*Functions:*

*Class MemoryInvoker.*

*Precondition:*

*Postcondition:*

*Functions:*