Group C:

Ian Gotcher

Hyrum Cox

Dallon Monson

David Rowley

Class Definition Document Additional Documentation

Class: Machine

Purpose of the class: Represents the core of UVSim, has memory, an accumulator, a program counter, and several objects for performing operations. Machine also has a tick function that, when called in a loop, allows the machine to interpret and execute all functions, after which it terminates the program. Also interprets instructions and passes off information to its several operation classes for processing.

+ tick() - Purpose: meant to be called in a loop, when called, it causes the machine to interpret an instruction, stopping the running of the machine if the instruction is invalid. Input parameters: none. Return value: none. Pre-condition: There must be a Machine object initialized with a parsed memory. Post-condition: The program must have either executed all of the instructions or have found an invalid instruction.

+ is_running():bool - Purpose: allows class users to determine if the machine is running. If the program is running, then tick can be called again and the program can continue being executed. If the program is not running, then UVSim has encountered an invalid instruction or has been halted, and tick should not be called again. Input parameters: none. Return value: Boolean value determining whether program is running or not. Pre-condition: Machine must be initialized. Post-condition: none.

+ set_memory_at_address(index, value) - Purpose: allows the setting of a register in memory from outside of the machine object. Input parameters: index - an int that is the index of the register in memory; value - an int that is a 4 letter word to be placed in the memory register at the index. Return value: none. Pre-condition: Machine must be initialized. Post-condition: Memory[index] must be equal to the value passed as a parameter.

+ get_memory_at_address(index):int - Purpose: allows the obtaining of the value in a register of memory from outside of the machine object. Input parameters: index - an int that is the index of the register in memory. Return value: an int that is the value in the memory register at the index. Pre-condition: Machine must be initialized. Post-condition: none.

+ set_accumulator(value) - Purpose: allows the setting of the value in the accumulator from outside of the machine object. Input parameters: value - an int that is the value to be placed in the accumulator. Return value: none. Pre-condition: Machine must be initialized. Post-condition: The value in the accumulator must be equal to the value passed as a parameter.

+ get_accumulator():int - Purpose: allows the obtaining of the value in the accumulator from outside of the machine object. Input parameters: none. Return value: an int that is the value in the accumulator. Pre-condition: Machine must be initialized. Post-condition: none.

+ get_needs_input():int - Purpose: Allows Machine class users to get the memory index value that needs input from outside of the Machine object. Input Parameters:

none. Return value: An int that is the location in memory that needs input. Pre-condition: Machine must be initialized. Post-condition: none.

+ set_needs_input(memory_index) - Purpose: Allows Machine class users to set needs_input to the index value in memory that needs input. Input parameters: memory_index an int that has two digits and is the index in memory that needs input. Return value: none. Pre-condition: Machine must be initialized. Post-condition. The Machine’s needs_input must be equal to the value passed in as the parameter memory_index.

+ reset() - Purpose: Calling this function allows the user to run the program again as it resets the Machine. Input parameters: none. Return value: none. Pre-condition: Machine must be initialized, and the current program must have terminated before being reset and run again. Post-condition: The program must have started again, with the accumulator and program_counter being 0, and the is_running flag set to True.

+ get_program_counter():int - Purpose: allows class users to obtain the current program counter of the machine object.. Input parameters: none. Return value: returns the int value in the program counter. Pre-condition: Machine must be initialized. Post-condition: none.

+ set_program_counter(value) - Purpose: allows the setting of the value in the program counter, so that the machine is made to execute an instruction at a particular index in memory. Input parameters: value - an int that is the index in memory that the program is to start executing instructions from after this function is called. Return value: none. Pre-condition: Machine must be initialized. Post-condition: The value in the program counter must be equal to the parameter value.

- interpret_instructions(instruction):int - Purpose: allows the machine to interpret instructions, taking an instruction and deciding which class of operations to use. Input parameters: instruction - an int with 4 digits which has a 2-digit opcode and a 2-digit operand. Return value: returns an int, -1 is if there is an invalid instruction, 0 if the instruction was correctly interpreted. Pre-condition: Machine must be initialized, tick() must be called from outside the object, and there must be a valid instruction to interpret passed as a parameter. Post-condition: The instruction must have been executed and this function shall have returned 0, or this function shall have returned -1, indicating that an invalid instruction was passed.

Class: Memory

Purpose of the class: Intended to simplify the creation and usage of “memory” with 100 indices. The parse function shall return a memory. The machine class shall take a memory parameter. Essentially a specialized list - used in the same way as a list, but initializes with 100 indices each holding 0.

- __setitem__(index, value) - Purpose: This is a Python magic function, allows users of memory to set the value in an index in the same way as a list (“memory[x] = y” instead of something like “memory.set(x, y)”). Input parameters: index - an int, 0 <= index <= 99, the index that the user wants to set; value - an int, is the value that is to be put into the index of the list. Return value: none. Pre-condition: memory must be initialized. Post-condition: The value in the memory index passed as a parameter must be holding the value passed as a parameter.

- __getitem__(index) - Purpose: This is a python magic function, allows users of memory to get the value in an index in the same way as a list (“a = memory[x]” instead of something like “a = memory.get(x)”). Input parameters: index - an int, 0 <= index <= 99, the index that the user wants to get the value from. Return value: returns whatever variable is in the memory index (since this is to be used as a machine memory, this should always be a 4-digit word int). Pre-condition: memory must be initialized. Post-condition: none.

Class: Parser

Purpose of the class: Is responsible for parsing the input .txt files and converting the files into a memory usable by the machine class. Raises errors if the file contains incorrectly formatted instructions or if the file cannot be opened.

+ parse(file_name):memory - Purpose: Converts the input .txt files into memory usable by the machine class. Raises errors if the file contains incorrectly formatted instructions or if the file cannot be opened. Input parameters: file_name - a string that is the name of the .txt file containing the instructions to be passed to a machine. Return value: returns a memory containing usable instructions for a machine object. Pre-conditions: parser is initialized, a .txt file containing valid instructions exists and is accessible by the UVSim Python program. Post-conditions: none.

Class: Control

Purpose of the class: Further interprets instructions already partially interpreted by machine’s interpret_instruction, once they are interpreted, the operations are then carried out by functions in this class.

- __call__(op_code, memory_index) - Purpose: A python magic function, further interprets instructions already partially interpreted by machine’s interpret_instruction, then calls the appropriate operation based on the op_code. Input parameters: op_code - the 2nd digit of a 4-digit word int, indicates which operation to call; memory_index - the 3rd and 4th digits of a 4-digit word int, indicates which memory_index to operate upon/using. Return value: none. Pre-conditions: The machine object and the operation object must be initialized, with the control object being called with said machine object as a parameter. Post-condition: The correct instruction must be called and given the memory index.

- branch(memory_index) - Purpose: allows the user to change the program counter to a different area of memory, which the program will continue executing instructions from. Input parameters: memory_index - a 2-digit int that is the memory index that the program is to branch to. Return value: none. Pre-conditions: Must be called by op_br. Post-conditions: the value in the program counter must be equal to the memory address that was passed as a parameter.

- branch_neg(memory_index) - Exactly the same as branch, except that it only branches if the value in the accumulator is negative. Otherwise, nothing happens and the program continues.

- branch_zero(memory_index) - Exactly the same as branch, except that it only branches if the value in the accumulator is zero. Otherwise, nothing happens and the program continues.

- halt() - Purpose: Allows the user to halt the program, changes the running boolean flag to false. Input parameters: none. Return value: none. Pre-conditions: none. Post-condition: running boolean flag must be false and the program must cease execution.

Class: IO

Purpose of the class: Further interprets instructions already partially interpreted by machine's interpret_instruction, once they are interpreted, the operations are then carried out by functions in this class.

- __call__(op_code, memory_index) - Purpose: A python magic function, further interprets instructions already partially interpreted by machine's interpret_instruction, then calls the appropriate operation based on the op_code. Input parameters: op_code - the 2nd digit of a 4-digit word int, indicates which operation to call; memory_index - the 3rd and 4th digits of a 4-digit word int, indicates which memory_index to operate upon/using. Return value: none. Pre-conditions: The machine object and the operation object must be initialized, with the IO object being called with said machine object as a parameter. Post-condition: The correct instruction must be called and given the memory index.

- read(memory_index) - Purpose: reads a word from keyboard input and puts it into a register in the Machine's memory. Input parameters: memory_index - a 2-digit int that is the memory index where the program will store the keyboard input. Return value: none. Pre-conditions: none. Post-conditions: The register at memory_index must contain the keyboard input.

- write(memory_index) - Purpose: writes a word from a memory location to screen. Input parameters: memory_index a 2-digit int that is the memory index where the program will get the word to print to screen. Return value: noen. Pre-conditions: none. Post-conditions: The program must have output the word from memory_index.

Class: Arithmetic

Purpose of the class: Further interprets instructions already partially interpreted by machine's interpret_instruction, once they are interpreted, the operations are then carried out by functions in this class.

- __call_(op_code, memory_index) - Purpose: A python magic function, further interprets instructions already partially interpreted by machine's interpret_instruction, then calls the appropriate operation based on the op_code. Input parameters: op_code - the 2nd digit of a 4-digit word int, indicates which operation to call; memory_index - the 3rd and 4th digits of a 4-digit word int, indicates which memory_index to operate upon/using. Return value: none. Pre-conditions: The machine object and the operation object must be initialized, with the arithmetic object being called with said machine object as a parameter. Post-condition: The correct instruction must be called and given the memory index.

- add(memory_index) - Purpose: gets a value from memory_index and adds it to the value in the accumulator, leaving the result in the accumulator. Input parameters:

memory_index - a 2-digit int that is the memory index where the value to be added to the value in the accumulator is to be retrieved from. Return value: none. Pre-conditions: none. Post-condition: The accumulator must be equal to the sum of the value that was previously in the accumulator and the value in memory at memory_index.

- subtract(memory_index) - Exactly the same as add(), except this function does subtraction instead of addition.

- divide(memory_index) - Exactly the same as add(), except this function does (integer) division instead of addition.

- multiply(memory_index) - Exactly the same as add(), except this function does multiplication instead of addition.

Class: LoadStore

Purpose of the class: Further interprets instructions already partially interpreted by machine's interpret_instruction, once they are interpreted, the operations are then carried out by functions in this class.

- __call__(op_code, memory_index) - Purpose: A python magic function, further interprets instructions already partially interpreted by machine's interpret_instruction, then calls the appropriate operation based on the op_code. Input parameters: op_code - the 2nd digit of a 4-digit word int, indicates which operation to call; memory_index - the 3rd and 4th digits of a 4-digit word int, indicates which memory_index to operate upon/using. Return value: none. Pre-conditions: The machine object and the operation object must be initialized, with the LoadStore object being called with said machine object as a parameter. Post-condition: The correct instruction must be called and given the memory index.

- load(memory_index) - Purpose: Allows the user to set the value in the accumulator to a value in memory at memory_index. Input parameters: memory_index - a 2-digit int that is the memory index where the value that the accumulator is to be set to is. Return value: none. Pre-conditions: none. Post-condition: The value in the accumulator must be equal to the value in memory at memory_index.
- store(memory_index) - Purpose: Allows the user to store the value in the accumulator in a register in memory. Input parameters: memory_index a 2-digit int that is the memory index where the value in the accumulator will be stored. Return value: none Pre-conditions: none. Post-condition: The value in memory at memory_index must be equal to the value in the accumulator.

Class: GUI

Purpose of the class: This class contains all the code necessary to create the UVSim GUI using the TKinter standard Python library. It creates the window and all UI elements while also interfacing with the other classes in the project.

- make_window() - Purpose: This function creates the operating system window as well as all the UI elements in the GUI. Input parameters: none. Return value: none. Pre-conditions: GUI object must be initialized, and a file must not be given as a command line argument. Post-conditions: A window with all GUI elements must appear on the user’s OS.

- import_memory() - Purpose: This function runs when the user clicks the "Import" button and allows the user of the program to select a file using the OS's file dialog, once the user selects a valid file, the text is parsed and converted into a usable memory, and a Machine object is instantiated using that memory, the GUI also shows the updated memory containing the instructions and data imported from the file. Input parameters: none. Return values: none. Pre-conditions: Window must be made. Post-conditions: A machine must be instantiated with the memory from the file and the memory column must show the contents of the memory.

- run() - Purpose: This function runs when the user clicks the "Run" button and allows the user of the program to run the program that they have loaded into the Machine, resets the machine so that the program can be run again if the user clicks the Run button again. Input parameters: none. Return values: none. Pre-conditions: The user must have imported a memory file. Post-conditions: UVSim must have executed the program that the user has imported, and UVSIM must be reset so that the program can be run again.

- update_memory_labels() - Purpose: When a memory is imported, this function updates the GUI element showing the new contents of the memory registers. Input parameters: none. Return values: none. Pre-conditions: A user must have imported a memory file. Post-conditions: The memory GUI element must show the new contents of the memory registers.

- print_to_output(text, end = "\n") - Purpose: This function allows the program to output text to the GUI window, allowing for messages such as "Welcome to the UVSim" to be displayed. Input parameters: text - a string that will be displayed in the window, end - by default set to a newline character. Return values: none. Pre-conditions: none. Post-conditions: The window must be displaying the text that was passed as a parameter.

- wait_for_input() - Purpose: Allows the program to pause and obtain user input, validates and stores this input, outputs an error message if input is invalid and allows the user to try again. Input parameters: none. Return values: none. Pre-conditions: There must be an instruction in the memory for the program to obtain input from the user. Post-conditions: The user's input must be stored in memory at the appropriate location as specified by the instruction.

Class: Input

Purpose of the class: This class handles the obtaining and validation of user inputs.

- get_validity() - Purpose: allows users to obtain the validity bool attribute from outside of the class. Input parameters: none. Return values: Returns a bool that indicates whether or not the input object is valid. Pre-conditions: Input must be instantiated. Post-conditions: none.

- set_validity(validity) - Purpose: allows users to set the validity bool attribute from outside of the class. Input parameters: validity - a bool that is to indicate the validity of the input class. Return values: none. Pre-conditions: Input must be instantiated. Post-conditions: The validity of the object must be equal to the validity passed as a parameter.

- validate_input(input) - Purpose: Checks if an input is a valid word, sets the validity of the input object accordingly. Input parameters: input - an input that shall be tested to determine its validity as a word. Return values: none. Pre-conditions: none. Post-conditions: The validity attribute must be set to True if the input is valid, and False if the input is invalid.

- check_exit(input):bool - Purpose: Checks if the user wants to exit the input, returns True if the input is equal to “exit”. Input parameters: input - an input that will be checked for equivalence to “exit”. Return values: returns a bool that is true if input = “exit”, false otherwise. Pre_conditions: none. Post_conditions: none.

+ get_input():string - Purpose: A public function that gets and validates input from the user by asking for input, checking if the user wants to exit, validating the input, returning an error message if the input is invalid, and returning the word. Input parameters: none. Return values: returns the validated word input by the user. Pre-conditions: The input object must be instantiated. Post-conditions: none.