Cairo University

Cairo University, Faculty of Computers and Artificial Intelligence

VOLE MACHINE REPORT

A1-Task4Task5-S5-20231116-20231134-20231042

NOVEMBER 1, 2024

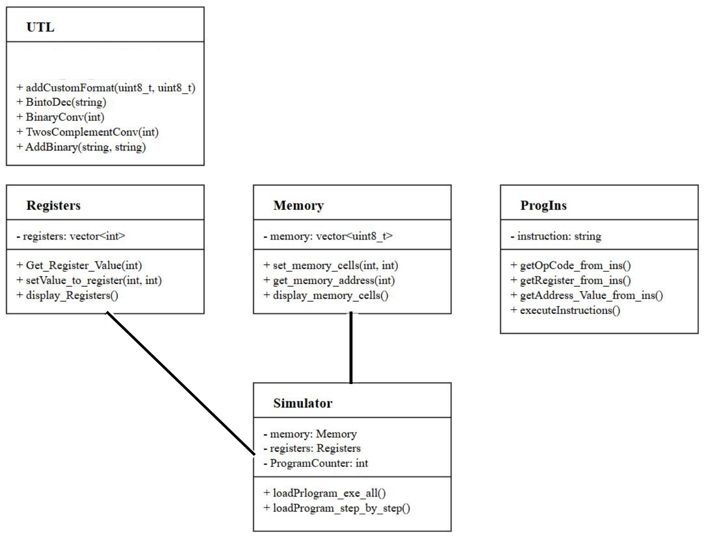
Name	ID	What he makes
Amr Khaled Ahmed Abd El-Hamid 20231116@stud.fci-cu.edu.eg	20231116	1. Register: Class: 1.1. Get_Register_Value: function 1.2. setValue_to_register: function 1.3. display_Registers: function 2. Instructions: Class 2.1. getOpCode_from_ins: function 2.2. getRegister_from_ins: function 2.3. getAddress_Value_from_ins: function 2.4. executeInstructions: function 3. Simulator: Class 3.1. loadProgram_exe_all: function 3.2. loadProgram_step_by_step: function 4. Opcode:1,2,4 5. Slow print: function 6. Menu 7. Defensive programming
Mohamed Ahmed Mohamed Abd El-Wahab 20231134@stud.fci-cu.edu.eg	20231134	1. Memory: Class 1.1. set_memory_cells: function 1.2. get_memory_address: function 1.3. display_memory_cells: function 2. hexa_char: function 3. hexa_unsigned_chars: function 4. char_to_hex: function 5. Normalize: function 6. AddFloatsBitwise: function 7. Opcode: 3,6, B,C
George Malak Magdy 20231042@stud.fci-cu.edu.eg	20231042	1. BintoDec: function 2. BinaryConv: function 3. TwosComplementConv: function 4. AddBinary: function 5. Opcode: 5,7,8,9, A
Each member on team contributes on:	1. Testing 2. Associate All program components 3. Validation	

Illustrative table

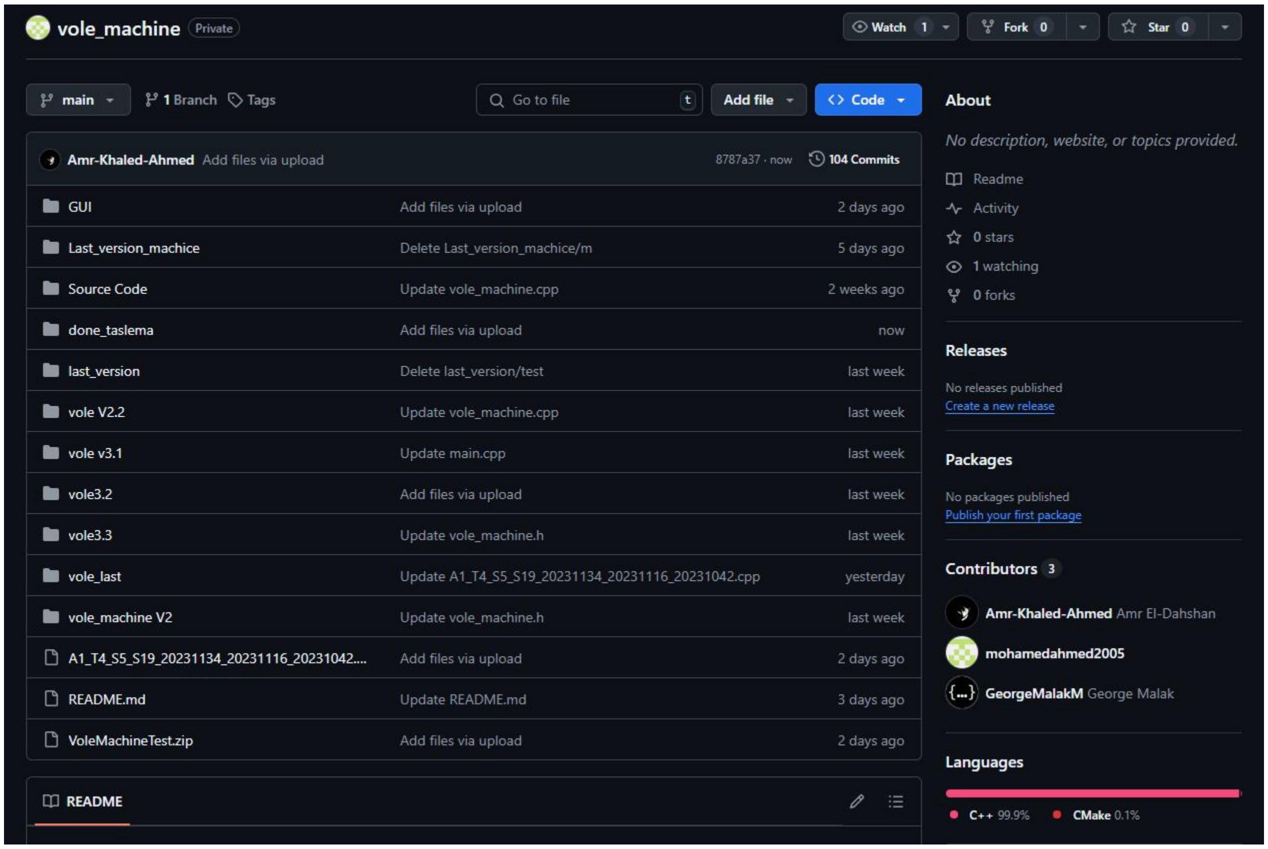
Contents

Vole Machine Simulator

UML Diagram:



GitHub repository:


vole_machine Private

Watch 1 Fork 0 Star 0

main 1 Branch Tags Go to file Add file Code

Amr-Khaled-Ahmed Add files via upload 8787a37 · now 104 Commits

GUI	Add files via upload	2 days ago
Last_version_machice	Delete Last_version_machice/m	5 days ago
Source Code	Update vole_machine.cpp	2 weeks ago
done_taslema	Add files via upload	now
last_version	Delete last_version/test	last week
vole V2.2	Update vole_machine.cpp	last week
vole v3.1	Update main.cpp	last week
vole3.2	Add files via upload	last week
vole3.3	Update vole_machine.h	last week
vole_last	Update A1_T4_S5_S19_20231134_20231116_20231042.cpp	yesterday
vole_machine V2	Update vole_machine.h	last week
A1_T4_S5_S19_20231134_20231116_20231042....	Add files via upload	2 days ago
README.md	Update README.md	3 days ago
VoleMachineTest.zip	Add files via upload	2 days ago

README

About

No description, website, or topics provided.

Readme
Activity
0 stars
1 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Contributors 3

Amr-Khaled-Ahmed Amr El-Dahshan
mohamedahmed2005
GeorgeMalakM George Malak

Languages

C++ 99.9% CMake 0.1%

GitHub link: https://github.com/mohamedahmed2005/vole_machine.git

Classes

Memory:

- **Memory definition:** memory is a vector which contains 256 cells on hexadecimal representation.
- We will make constructor to make all elements on vector equal to 0
- **Set memory cells:** On execution program, instructions must be loaded on memory cells, and we should handle overflow.

- **Get memory cells**: after execution program, we will return value on cells to be used on program later.
- **Display memory cells**: program will output 256 cells of memory and values which is inside each memory cell.
- We will make destructor to avoid memory leak.

Register:

- **Definition:** The Registers class manages a set of registers used for temporary storage of values during program execution.
- **Constructor**: Initializes all register values to 0.
- **Get_Register_Value:** Retrieves the value stored in a specified register.
- **setValue_to_register**: Updates the value of a specified register, ensuring the index is valid.
- **display_Registers:** Outputs the current values of all registers for debugging and tracking purposes.
- **Destructor**: Cleans up any resources, although simple integer storage typically requires minimal management.

ProgIns

- **Definition:** The ProgIns class encapsulates program instructions, providing methods to extract components from instructions and execute them.
- **Constructor**: Initializes the instruction string.
- **getOpCode_from_ins:** Extracts the opcode from the instruction for processing.
- **getRegister_from_ins:** Retrieves the register index specified in the instruction.
- **getAddress_Value_from_ins:** Extracts the address or value from the instruction for memory operations.
- **executeInstructions:** Implements logic to execute the instruction based on the opcode and interacts with the memory and registers as necessary.

Simulator:

- **Definition:** The Simulator class coordinates the operation of the virtual machine, managing both memory and registers while executing instructions.
- I use an association from Registers Class and Memory class.
- **Constructor**: Initializes the simulator, including the memory and registers.
- **loadProgram_exe_all**: Loads all instructions into memory for execution and give all data afte r execute it.
- **loadProgram_step_by_step**: Allows instructions to be loaded and executed step by step, with ask for continue or not.

UTL :

- Slow Print Function: This function takes a millisecond delay and prints every character after this delay.
- BintoDec: This function takes a binary number represented as a string (e.g., "101") and returns its decimal equivalent as an integer. For example, inputting "101" will return the decimal value 5.

- hexa_char: the Function of this function is to convert a single hex digit to an unsigned character.
- hexa_unsigned_chars: the Function of this function is to convert a hex string to an unsigned char.
- char_to_hex: the function of this function is to convert an unsigned char to a hex string.
- Normalize_IEEE: This function converts floating numbers to IEEE “8-bits” Representation.
- Add_two_floating_numbers: this Function to add two 8-bit floating-point numbers in custom format.
- BinaryConv: This function accepts an integer and returns its binary representation as a string. For instance, if you input the integer 5, it will return "101". (Note: this function is only called within **TwosComplementConv**).
- TwosComplementConv: This function first checks if a given number is positive or negative. If the number is positive, it converts it directly to binary using **BinaryConv**. If the number is negative, it

takes the positive version of the number, converts it to binary, inverts all binary digits (changing 1s to 0s and 0s to 1s), and finally adds 1, following the rules of two's complement representation.

- AddBinary: This function takes two binary numbers represented as strings, adds them, and returns the result as a binary string, ignoring any overflow.

Menu:

- ***Options***:
  - ***1 - Load a new program***: *This option allows the user to load and execute a program.*
  - ***2 - Exit***: *This option terminates the program and displays a farewell message.*
- ***User Input***: *The user is prompted to enter a choice. Input validation ensures only valid choices are accepted.*

Execution Menu *(appears after selecting to load a new program):*

- ***Options***:
  - ***1 - Execute line by line***: *This option allows the user to step through the program one instruction at a time.*

    - **2 - Execute the entire program at once**: This option loads and executes all instructions in a single step.
- **User Input**: Similar to the main menu, the user is prompted to enter a choice, with error handling for invalid inputs.

The Machine's Language

c	Format	Operand	Description
1	Load	RXY	LOAD the register R with the bit pattern found in the memory cell whose address is XY. Example.
2	Load	RXY	LOAD the register R with the bit pattern XY. Example: 20A3 would cause the value A3 to be placed in register 0.
3	Store	RXY	STORE the bit pattern in register R into the memory cell at address XY. Example.

4	Move	ORS	MOVE the bit pattern found in register R to register S. Example.
5	Add	RST	ADD the bit patterns in registers S and T as though they were two's complement representations and leave the result in register R. we have used the functions: • TwosComplementConv • BinaryConv • AddBinary • BintoDec
6	FAdd	RST	ADD the bit patterns in registers S and T as though they represented values in floating-point notation and leave the floating-point result in register R. we have used the functions: • hexa_char • hexa_unsigned_chars • char_to_hex • decode_to_decimal • encode_from_decimal • Normalize_IEEE • Add_two_floating_numbers
7	OR	RST	OR the bit patterns in registers S and T and place the result in register R. Example.
8	AND	RST	AND the bit patterns in registers S and T and place the result in register R. Example.
9	EXCLUSIVE OR	RST	EXCLUSIVE OR the bit patterns in registers S and T and place the result in register R. Example.
A	Rotate	ROX	ROTATE the bit pattern in register R one bit to the right X times. Each time place the bit that started at the low-order end at the high-order end. Example.

B	Jump	RXY	JUMP to the instruction located in the memory cell at address XY if the bit pattern in register R is equal to the bit pattern in register number 0. Otherwise, continue with the normal sequence of execution. (The jump is implemented by copying XY into the program counter during the execute phase.
C	Halt	000	HALT execution. Example: C000 would cause program execution to stop