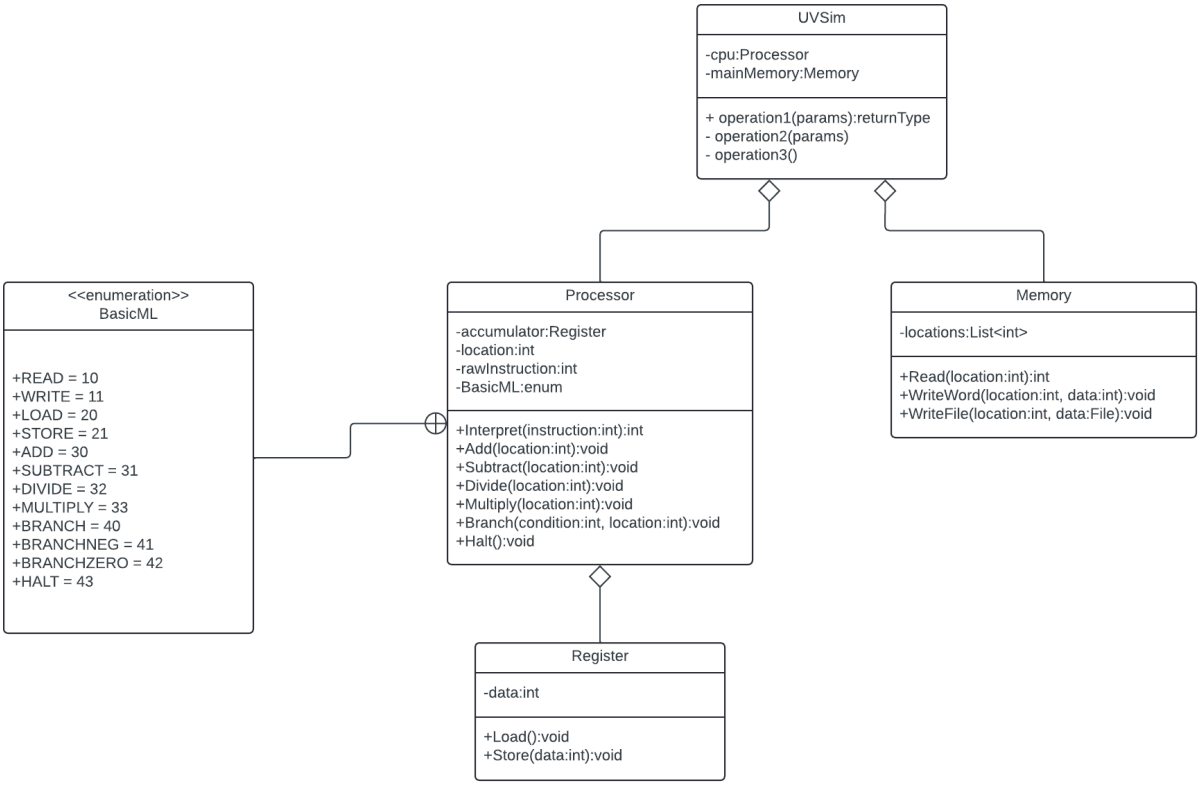
9/12/2024
Scrum Master:
Chris Aldous

Attendance:
Sam Christmas
Samuel Griffey
Elijah Rutherford
Chris Aldous

Agenda Items:

- Milestone 2 Requirements
    - Console Application / Working Prototype
        - Class Structure
            1. Class Diagram



            2. Class Templates
            3. Data Structures
    - Design Document
        - 2 User Stories
            1. As a student, using the UVSim, I want to be able to load a text file into main memory.

2. As a student, using the UVSim, once I have loaded a file I want to execute the program and see the output.
    - 10 - 15 Use Cases

- Unit Tests
    - At least two for the use cases
- README

Action Items:

- Elijah:
    - Class Templates
- Sam Christmas:
    - Class Diagram
- Chris Aldous
    - Avalonia GUI
- Sam Griffey
    - Functional Logic

9/19/2024
Scrum Master:
Elijah Rutherford

Attendance:
Sam Christmas
Samuel Griffey
Elijah Rutherford
Chris Aldous

Action Items:

- Elijah:
    - Test Functions
- Sam Christmas:
    - State Transition Diagram
    - README
- Chris Aldous
    - Test Functions
- Sam Griffey
    - Add Github Issues

Agenda Items:

- UVSim Design Document

UVSim is a software simulator designed to help computer science students learn machine language and computer architecture through practical experience. It allows students to write, load, and execute BasicML programs in a controlled virtual environment. This program is aimed to help computer science students, as well as the instructors teaching computer programming and computer architecture courses.

One of the objectives of the program is to provide an intuitive interface for writing and executing BasicML code. Another objective is to enable students to understand fundamental concepts of machine language and architecture while facilitating hands-on practice in a controlled environment.

The features of UVSim will include the ability to load and execute BasicML programs and the ability to perform arithmetic operations.

User Stories:

As a student**,** I want to write and execute BasicML programs so that I can become familiar with machine language and gain a better understanding of computer architecture. By using the BasicML simulator, I will develop my coding skills as well as be more prepared for exams.

As an educator**,** I want to analyze the BasicML programs submitted by my students so that I can assess their understanding of key concepts in computer architecture and provide useful and constructive feedback. This will help to make the understanding of this abstract subject easier to grasp and establish a good foundation of computer architecture knowledge.

Use Cases: (Require System, Actors, and Scenario)

System is UVSim for all use cases

Use Case 1: Write a word to a location memory

Primary Actor: Student

Scenario: The student will be able to load instructions and data into memory.

Use Case 2: Read a word from a location in memory

Primary Actor: Student

Scenario: Upon completing the program execution, the user can request output to be displayed using the Write command. The system retrieves the specified memory content and presents it on the screen.

Use Case 3: Load a word into accumulator

Actor: Student

Scenario: The student will be able to successfully load a word from memory into the accumulator.

Use Case 4: Store a word from the accumulator into memory

Actor: Student

Scenario: The student will be able to store the contents of the accumulator into a memory location.

Use Case 5: Add

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the add operator. This operator correctly adds the word in the accumulator to a word from memory and the results are stored in the accumulator.

Use Case 6: Subtract

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the subtract operator. This operator correctly subtracts the word in the accumulator from a word in memory and the results are stored in the accumulator.

Use Case 7: Divide

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the divide operator. This operator correctly divides the word in the accumulator by a word from memory and the results are stored in the accumulator.

Use Case 8: Multiply

Primary Actor: Student

Scenario: The user writes a BasicML program that includes the subtract operator. This operator correctly subtracts the value in the accumulator from a value stored in memory and the results are stored in the accumulator.

Use Case 9: Halt Program Execution

Primary Actor: Student

Scenario: While the program is running, the user chooses to halt execution. The system stops processing the program.

Use Case 10: Branch to a memory location

Primary Actor: System

Scenario: The user is able to use branch command to control program execution.

Use Case 11: Load Instructions from Text File

Actor: Student

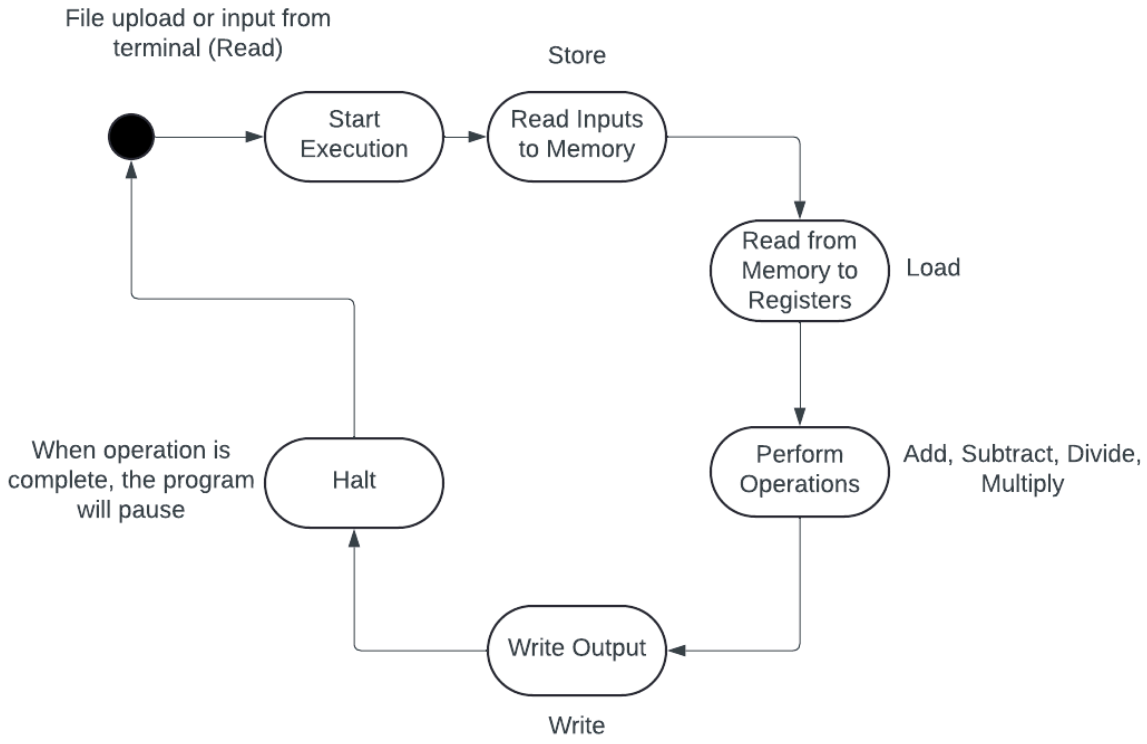
Scenario: The student selects a valid text file. The instructions from the file are successfully loaded into the memory registers, starting at the specified memory location.

Use Case 12: Execute Program

Primary Actor: Student

Scenario: The student executes the program they have loaded into UVSim using the user interface.

State Transition Diagram



Use Case Diagram

README.txt

Cloning the Repository

Begin by cloning the repository. This can be done by running this command in the command line: git clone https://github.com/captaincobbs/CS_2450_Application.git

Building the Application

Build the application by navigating to the cloned directory and selecting the shortcut to the executable file (`Application\bin\Debug\net8.0\CS_2450_Application.exe`)

in the main directory and run it.

Writing and Executing BasicML

Once the application has been built, you should be able to start executing BasicML files by following the input prompts in the terminal.

BasicML instructions (from Group Project Milestone 2 description) :

I/O operation:
READ = 10 Read a word from the keyboard into a specific location in memory.
WRITE = 11 Write a word from a specific location in memory to screen.

Load/store operations:
LOAD = 20 Load a word from a specific location in memory into the accumulator.
STORE = 21 Store a word from the accumulator into a specific location in memory.

Arithmetic operation:
ADD = 30 Add a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator)
SUBTRACT = 31 Subtract a word from a specific location in memory from the word in the accumulator (leave the result in the accumulator)
DIVIDE = 32 Divide the word in the accumulator by a word from a specific location in memory (leave the result in the accumulator).
MULTIPLY = 33 multiply a word from a specific location in memory to the word in the accumulator (leave the result in the accumulator).

Control operation:
BRANCH = 40 Branch to a specific location in memory

BRANCHNEG = 41 Branch to a specific location in memory if the accumulator is negative.
BRANCHZERO = 42 Branch to a specific location in memory if the accumulator is zero.
HALT = 43 Pause the program

The last two digits of a BasicML instruction are the operand – the address of the memory location containing the word to which the operation applies.

Example of a BasicML program:

10 00 // READ input into memory location 00

20 00 // LOAD value from memory location 00 into accumulator

30 01 // ADD value from memory location 01 to value in the accumulator

21 02 // STORE accumulator value into memory location 02

11 02 // WRITE value stored in memory location 02 to screen

43 // HALT