CS2450 Group Project

By Jonah Burton, Michael Findlay, Gaby Rodriguez, James Durfee, and Cortland Niccum

Documentation by Jonah Burton

Table of Contents:

A Quick Summary:

This group project is an exercise in team-building and community oriented learning; around the focal point of a basic virtual machine. The program itself is made to simulate machine language and computer architecture; with an accumulator register and 250-words of memory executing BasicML code. Programs may be loaded from a file and run sequentially unless otherwise directed via branch instructions. The organization of these elements is relatively simple which makes this an attractive option for a student or teacher.

It can provide a number of basic arithmetic operations as well as conditional branching, reading and writing from input, and handling error codes.

Future Changes:

Modifications to this program could include new functionality around editing the code in-app; so as to demonstrate the cascading effects of small changes to machine-read code, or breakpoints so as to capture and analyze the program at a variety of points. The real-world applications of these two features would be fairly large; as high-level demonstration would become much easier with this kind of technique.

This is the first of our relevant documents for the project's full and comprehensive documentation; a key design framework:

UVSim - High-Level Functionality Document

System Overview

UVSim (Universal Virtual Simulator) is a virtual machine designed for computer science students to learn machine language and computer architecture. It simulates a simple CPU with an accumulator register and a 250-word memory, executing BasicML instructions. Programs are loaded from a file into memory and executed sequentially unless modified by branching instructions.

User Stories

1. A Student Runs UVSim to Learn Machine Language

As a computer science student,

I want to execute my BasicML programs on UVSim,

So that I can learn how machine language works through hands-on execution.

2. A Teacher Tests Student Programs

As a computer science professor,

I want to verify students' machine code execution using UVSim,

So that I can ensure they understand machine instructions and program flow.

Use Cases

1. Load a Value into Memory

Description: The user loads a specific value into memory at a designated address.

Steps:

1. User provides an instruction with opcode `021XXX`.
2. The value stored in the accumulator is saved to memory at location `XXX`.
3. The system confirms the storage.

2. Retrieve a Value from Memory

Description: The user loads a value from memory into the accumulator.

Steps:

1. User provides an instruction with opcode `020XXX`.
2. The value at memory location `XXX` is copied into the accumulator.
3. The system updates the accumulator value.

3. Perform Addition

Description: The system adds a value from memory to the accumulator.

Steps:

1. User provides an instruction with opcode `030XXX`.

2. The value at memory location `XXX` is added to the accumulator.

3. The system stores the result in the accumulator.

4. Perform Subtraction

Description: The system subtracts a value in memory from the accumulator.

Steps:

1. User provides an instruction with opcode `031XXX`.

2. The value at memory location `XXX` is subtracted from the accumulator.

3. The system updates the accumulator with the new value.

5. Perform Multiplication

Description: The system multiplies the accumulator by a value in memory.

Steps:

1. User provides an instruction with opcode `033XXX`.

2. The value at memory location `XXX` is multiplied by the accumulator.

3. The system updates the accumulator with the result.

6. Perform Division

Description: The system divides the accumulator by a value in memory, handling division by zero.

Steps:

1. User provides an instruction with opcode `032XXX`.
2. The system checks if the divisor is zero. If yes, execution halts with an error message.
3. Otherwise, the system performs the division and updates the accumulator.

7. Branching to a Memory Address

Description: The system jumps execution to a different memory address unconditionally.

Steps:

1. User provides an instruction with opcode `040XXX`.
2. The instruction counter updates to `XXX`.
3. Execution continues from the new memory location.

8. Branching on Negative Value

Description: The system jumps to a new memory address if the accumulator contains a negative value.

Steps:

1. User provides an instruction with opcode `041XXX`.
2. If the accumulator is negative, the instruction counter updates to `XXX`.
3. Execution continues from the new memory location.

9. Branching on Zero Value

Description: The system jumps to a new memory address if the accumulator contains zero.

Steps:

1. User provides an instruction with opcode `042XXX`.
2. If the accumulator is zero, the instruction counter updates to `XX`.
3. Execution continues from the new memory location.

10. Writing to Output

Description: The system prints a value stored in memory to the screen.

Steps:

1. User provides an instruction with opcode `011XXX`.
2. The system retrieves the value at memory location `XXX`.
3. The value is printed to the console.

11. Reading from Input

Description: The system takes user input and stores it in memory.

Steps:

1. User provides an instruction with opcode `010XXX`.
2. The system prompts the user for an input value.
3. The value is stored at memory location `XXX`.

12. Halt Execution

Description: The system stops executing instructions when a halt command is encountered.

Steps:

1. User provides an instruction with opcode `043XXX`.
2. The system sets the `running` flag to `False`.
3. Execution stops immediately.

13. Handling Invalid Opcodes

Description: The system detects an invalid opcode and terminates execution with an error.

Steps:

1. The system encounters an unrecognized opcode.
2. An error message is printed.
3. Execution halts to prevent unexpected behavior.

14. Handling Out-of-Bounds Memory Access

Description: The system prevents memory accesses beyond allocated space.

Steps:

1. The user attempts to load/store data in an invalid memory location (outside 0-249).
2. The system detects the issue and prevents execution from continuing.

3. An error message is displayed, and execution stops.

15. Execution of a Full Program from File

Description: The system reads and executes a complete BasicML program from a file.

Steps:

1. The user provides the filename containing BasicML instructions.
2. The system loads instructions into memory.
3. The system executes instructions sequentially until encountering `HALT` or an error.

16. Convert Old Format File

Description: The system reads and converts from a 4-digit to a 6-digit format version of the file.

Steps:

1. The user selects a 4-digit format file.
2. The system adds leading zeroes to each opcode and address.
3. The system writes a 6-digit format version of the file.

17. Open Multiple Files

Description: The system creates a new file tab allowing editing/running multiple files in one app instance

Steps:

1. The user selects "Open File"
2. A new file tab is created.

3. The user can edit/run multiple files simultaneously within one app instance

A second document, under our Github readme:

GroupProject2450

UVSim - BasicML Virtual Machine

How to Run

1. Open a terminal.
2. Run `python uvsim.py`.
3. Enter the program file when prompted (e.g., `Test1.txt`).
4. The UVSim will execute the instructions in the file.

Input File Format

- Each line in the input file should contain a **signed six-digit number**.
- Instructions are stored sequentially in memory (up to 250 lines).
- The **last line** of the file should be `-99999` to mark the end.

Supported Commands

| Opcode | Instruction | Description |

|--------| ---------- | ---------------------------------- |

| 010XXX | READ | Read input into memory[XX] |

| 011XXX | WRITE | Print memory[XX] |

| 020XXX | LOAD | Load memory[XX] into accumulator |

| 021XXX | STORE | Store accumulator into memory[XX] |

| 030XXX | ADD | Add memory[XX] to accumulator |

| 031XXX | SUBTRACT | Subtract memory[XX] from accumulator |

| 032XXX | DIVIDE | Divide accumulator by memory[XX] |

| 033XXX | MULTIPLY | Multiply accumulator by memory[XX] |

| 040XXX | BRANCH | Jump to memory[XX] |

| 041XXX | BRANCHNEG | Jump if accumulator is negative |

| 042XXX | BRANCHZERO | Jump if accumulator is zero |

| 043XXX | HALT | Stop execution |

File Format Compatibility

UVSIM now supports both 4-digit and 6-digit instruction formats.

- Files in 4-digit format can be **converted to** 6-digit using the built-in conversion tool.

- Converted files have leading zeroes added to the opcode and address.

- All instructions within a file must match the same format (mixing 4-digit and 6-digit is not allowed)

Running Unit Tests

To run the unit tests, execute:

```

python -m unittest uvsim.py

```

When prompted to select a file, select "Test1.txt"

This will automatically verify all functionalities.

Error Handling

- **Division by zero** is detected and will halt execution.
- **Invalid opcodes** will print an error and stop execution.
- **Out-of-bounds memory access** is prevented.

>>>>>>> origin/main

A simple Unit Test chart:

Test Name	Description	Use Case	Inputs	Expected Output	Success Criteria
test_load	Tests loading a value into memory	Load Operation	Memory[10] = 25	Memory[10] = 25	Memory matches expected value
test_addition	Tests adding a value from memory	ADD Operation	Accumulator=10, Memory[5]=15	Accumulator=25	Sum is correct
test_subtraction	Tests subtracting a value from memory	SUBTRACT Operation	Accumulator=50, Memory[5]=20	Accumulator=30	Subtraction is correct
test_divide_by_zero	Tests division by zero handling	DIVIDE Operation	Accumulator=50, Memory[5]=0	Simulation stops	Error handled
test_store	Tests storing accumulator value	STORE Operation	Accumulator=99	Memory[5]=99	Stored correctly
test_halt	Tests halting execution	HALT Operation	Instruction=4300	Simulation stops	Execution stops as expected
test_multiply	Tests multiplication	MULTIPLY Operation	Accumulator=5, Memory[5]=4	Accumulator=20	Multiplication is correct
test_branch	Tests branching to a specific location	BRANCH Operation	Memory[0]=4005, Memory[5]=4300	Instruction Counter=5	Branch executed correctly
test_branchneg	Tests branching on negative accumulator	BRANCHNEG Operation	Accumulator=-1, Memory[0]=4105, Memory[5]=4300	Instruction Counter=5	Branch on negative executed
test_branchzero	Tests branching on zero accumulator	BRANCHZERO Operation	Accumulator=0, Memory[0]=4205, Memory[5]=4300	Instruction Counter=5	Branch on zero executed
test_invalid_opcode	Tests handling of invalid opcodes	Invalid Opcode	Memory[0]=9999	Simulation stops	Unknown opcode error handled
test_memory_bounds	Tests memory bounds access	Memory Operation	Memory[0]=2099 (accessing memory[99])	Accumulator=0	Accessed valid memory location
convert_4_to_6_digits	Converts a 4-digit file to 6-digit format	Convert Old Format File	Test1.txt	All instructions have leading zeroes added	Converted file has valid 6-digit instructions only
reject_mixed_format	Runs a file with mixed formats	Mixing Formats	A file with mixed formats	Error: Mixed formats	Error handled
reject_over250	Tests out-of-bounds memory	Out-of-Bounds	A file with more than 250 lines	Error: Memory overflow	Execution stops as expected

Meeting descriptions:

Jan 25 2025
Group members: Jonah Burton, Michael Findlay, Gaby Rodriguez, James Durfee, and Cortland Niccum

Ideal meeting day: Tuesdays after 5pm, using google meets

Scrum order: Gaby, Michael, Cortland, James, Jonah

Programming Language: Java or Python, gonna decide once everyone can get in a meeting. Jonah and Michael didn't make it to the meeting this week.

Source control: Github

Project Management interface: Asana

Notes:

Attendees: Gaby, James, and Cortland

Got to know each other and learned what everyone was experienced in. We determined one of the best days for us to meet would be Tuesdays after 5pm.

Feb 4, 2025

Notes:

Attendees: Gaby, James, and Cortland

Short meeting to discuss task delegation and team member participation. We decided to meet again to make more formal decisions.

Feb 12, 2025

Notes:

Attendees: Gaby, James, Jonah

Quick team discussion, we decided to meet back after the makeup meeting and watch the video about milestone 3. Next meeting plan is to delegate milestone 3 tasks.

March 8, 2025

Notes:

Attendees: James, Gaby

We were not able to discuss anything as nobody else showed up.

April 7, 2025

Notes:

Attendees: James, Gaby

Decided to convert files with opcode length of 4 to opcode length of 6 during runtime.

April 22, 2025

Attendees: James, Gaby

Assigned roles for completion of final project submission.

FINAL SUBMISSION ASSIGNMENTS

James - Write Script
Edit video

Gaby - Make Powerpoint

Cortland - Record Powerpoint explanation
Record Demonstration

Jonah - Ensure all documentation is present and updated

These pieces of documentation have been presented to give a full recount of how the project was made and by whom; with detail paid towards the use cases and operation of the final product. Hopefully these have been illuminating and help readers to understand the intention and development of this project.

END