Group Project Milestone 2

Design Document

User Stories:

- **User Story 1**: As an accountant, I want a calculator with a memory, so that I can keep track of values and easily make adjustments as time passes.
- **User Story 2**: As a physicist, I want a way to automate calculation, so that I can automatically input values and get the resulting appropriate values for that calculation.

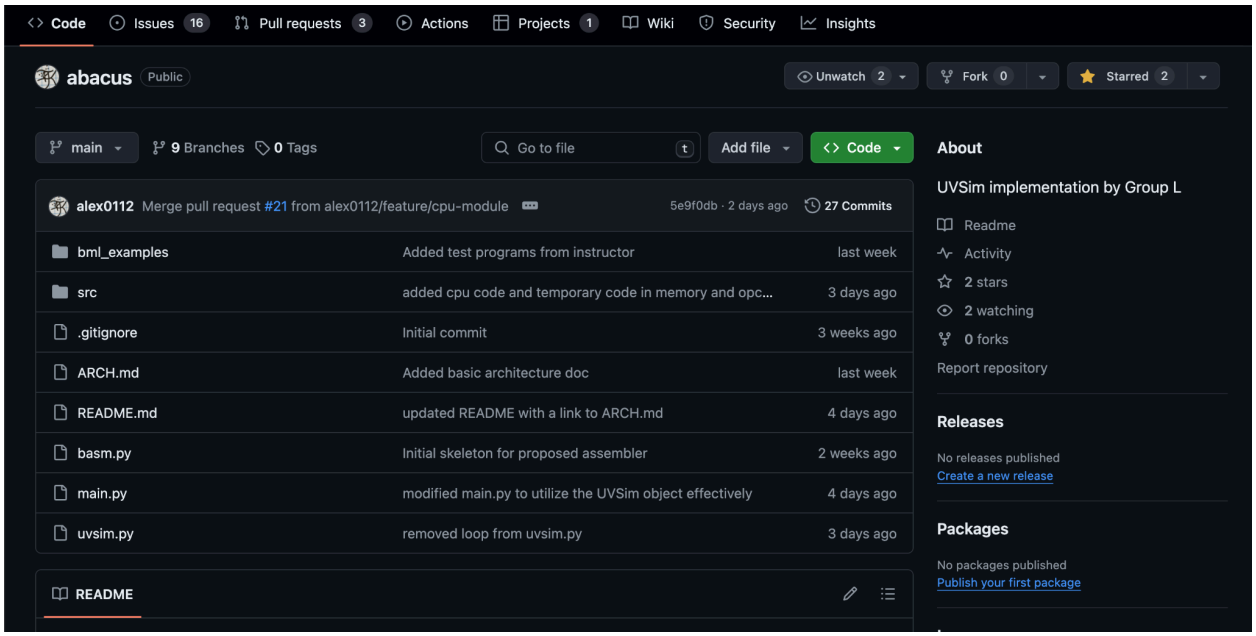
Use Cases:

- **Use Case 1**: Read takes a word imputed and saves it to the specified memory location
- **Use Case 2**: Write takes a word from a specified memory location and outputs the word to the screen
- **Use Case 3**: Load takes a word from a specified memory location and saves it in the accumulator
- **Use Case 4**: Store takes the word in the accumulator and moves it to a specified memory location
- **Use Case 5**: The Arithmetic functions take a word from memory and correctly adds, subtracts, multiplies or divides with the word in the accumulator and stores the last four whole numbers in the accumulator.
    - **Use Case 5a**: The Arithmetic functions take a word from memory and correctly divides it with the word in the accumulator and stores the last four whole numbers in the accumulator.
    - **Use Case 5b**: The Arithmetic functions take a word from memory and correctly adds it with the word in the accumulator and stores the last four whole numbers in the accumulator.
    - **Use Case 5c**: The Arithmetic functions take a word from memory and correctly subtracts it with the word in the accumulator and stores the last four whole numbers in the accumulator.
    - **Use Case 5d**: The Arithmetic functions take a word from memory and correctly multiplies it with the word in the accumulator and stores the last four whole numbers in the accumulator.
- **Use Case 6**: Branch sends the next command to be at the specified memory location and continue from that location if the word in the accumulator is positive.
- **Use Case 7**: BranchNeg sends the next command to be at the specified memory location and continue from that location if the word in the accumulator is negative.
- **Use Case 8**: BranchZero sends the next command to be at the specified memory location and continue from that location if the word in the accumulator is exactly Zero.

- **Use Case 9**: Halt Immediately ends the program
- **Use Case 10**: All opcodes execute the first two numbers of the words to the memory location of the last two numbers of that word.

Working Prototype

Link to our source control base: https://github.com/alex0112/abacus



Unit Test(s) Spreadsheet:

test_cpu.py:

Test Name	Description	Use Case Reference	Inputs	Expected Outputs	Pass/Fail Criteria
test_init	Initialize CPU and check initial values	Initialization	None	CPU halted: False, current: 0, acc: +0000	CPU halted is False, current is 0, acc is +0000
test_overflow	Set current counter	Current Counter	cpu.current = 100	IndexError	Raises IndexError

	beyond upper bound				
test_underflow	Set current counter below lower bound	Current Counter	cpu.current = -1	IndexError	Raises IndexError
test_halt_from_index	Halt CPU when current counter reaches 99	Halting	cpu.current = 99	CPU halted: True	CPU halted is True
test_halt_from_instruction	Halt CPU with HALT instruction	Halting	Opcode('4300')	CPU halted: True	CPU halted is True
test_read_good_word	Read valid word into memory	READ	Opcode('1010'), io_device.read = lambda: '1234'	memory.read(10) == 1234	memory.read(10) == 1234
test_read_bad_word	Read invalid word into memory	READ	Opcode('10100'), io_device.read = lambda: '1234'	ValueError	Raises ValueError
test_write	Write word from memory to IO device	WRITE	Opcode('1120'), memory.write(20, 5678)	io_device.write(5678) == '5678'	io_device.write(5678) == '5678'
test_write_bad_word	Write invalid word from memory to IO device	WRITE	Opcode('11100')	ValueError	Raises ValueError
test_load	Load word from memory into accumulator	LOAD	Opcode('2030'), memory.write(30, Opcode('9101'))	cpu.acc == Opcode('9101')	cpu.acc == Opcode('9101')
test_load_bad_word	Load invalid word from memory into accumulator	LOAD	Opcode('20200')	ValueError	Raises ValueError

test_store	Store word from accumulator into memory	STORE	Opcode('+1337'), cpu.acc = Opcode('+1337'), cpu.store(memory, 42)	memory.read(42) == 1337	memory.read(42) == 1337
test_store_bad_word	Store invalid word from accumulator into memory	STORE	Opcode('21200')	ValueError	Raises ValueError
test_add	Add word from memory to accumulator	ADD	Opcode('3040'), memory.write(40, Opcode('1000'))	cpu.acc == Opcode('+2234')	cpu.acc == Opcode('+2234')
test_add_bad_word	Add invalid word from memory to accumulator	ADD	Opcode('30100')	ValueError	Raises ValueError
test_subtract	Subtract word from memory from accumulator	SUBTRACT	Opcode('3150'), memory.write(50, Opcode('1234'))	cpu.acc == Opcode('+4444')	cpu.acc == Opcode('+4444')
test_subtract_bad_word	Subtract invalid word from memory from accumulator	SUBTRACT	Opcode('31100')	ValueError	Raises ValueError
test_multiply	Multiply word from memory with accumulator	MULTIPLY	Opcode('3260'), memory.write(60, Opcode('0002'))	cpu.acc == Opcode('+2468')	cpu.acc == Opcode('+2468')
test_multiply_bad_word	Multiply invalid word from memory with accumulator	MULTIPLY	Opcode('32100')	ValueError	Raises ValueError

test_divide	Divide accumulator by word from memory	DIVIDE	Opcode('3370'), memory.write(70, Opcode('0002'))	cpu.acc == Opcode('+0617')	cpu.acc == Opcode('+0617')
test_divide_bad_word	Divide accumulator by invalid word from memory	DIVIDE	Opcode('33100')	ValueError	Raises ValueError
test_divide_by_zero	Divide accumulator by zero	DIVIDE	Opcode('3380'), memory.write(80, Opcode('0000'))	ZeroDivisionError	Raises ZeroDivisionError
test_branch	Branch to specific memory address	BRANCH	Opcode('4040')	cpu.current == 40	cpu.current == 40

test_memory.py:

Test Name	**Description**	**Use Case Reference**	**Inputs**	**Expected Outputs**	**Pass/Fail Criteria**
test_init	Initialize Memory object and check if created	Initialization	None	Memory object exists	Memory object exists
test_init_from_list	Initialize Memory and write/read a value	Initialization	mem.write(0, -1234)	mem.read(0) == -1234	mem.read(0) == -1234
test_bad_init_from_list	Initialize Memory and write out-of-bounds value	Initialization	mem.write(10000000000000, 0)	IndexError	Raises IndexError

test_read	Write and read multiple values from Memory	READ	mem.write(0, 1234), mem.write(1, 4567), mem.write(2, 8910)	mem.read(0) == 1234, mem.read(1) == 4567, mem.read(2) == 8910	mem.read(0) == 1234, mem.read(1) == 4567, mem.read(2) == 8910
test_write	Write a value to Memory and verify	WRITE	mem.write(12, 1234)	mem.read(12) == 1234	mem.read(12) == 1234
test_writenext	Write to next available memory location	Write Next Available	mem.write(0, 1234), mem.write(1, 5678), mem.write(2, 8910), mem.writenext(1112)	mem.read(3) == 1112	mem.read(3) == 1112

test_opcode.py:

Test Name	**Description**	**Use Case Reference**	**Inputs**	**Expected Outputs**	**Pass/Fail Criteria**
test_init_with_plus	Initialize Opcode with a plus sign	Initialization	Opcode("+1234")	str(op) == "+1234"	str(op) == "+1234"
test_init_with_minus	Initialize Opcode with a minus sign	Initialization	Opcode("-1234")	str(op) == "-1234"	str(op) == "-1234"
test_init_with_no_sign	Initialize Opcode without a sign	Initialization	Opcode("1234")	str(op) == "+1234"	str(op) == "+1234"2
test_init_with_too_long	Initialize Opcode with too long value	Initialization	Opcode("12345")	ValueError	Raises ValueError
test_init_with_too_short	Initialize Opcode with	Initialization	Opcode("123")	ValueError	Raises ValueError

	too short value				
test_init_with_non_num	Initialize Opcode with non-numeric characters	Initialization	Opcode("12a4")	ValueError	Raises ValueError
test_init_with_empty	Initialize Opcode with an empty string	Initialization	Opcode("")	ValueError	Raises ValueError
test_init_with_too_large	Initialize Opcode with too large value	Initialization	Opcode("+12345")	ValueError	Raises ValueError
test_init_with_too_small	Initialize Opcode with too small value	Initialization	Opcode("+123")	ValueError	Raises ValueError
test_name	Get the name of the Opcode	Get Name	Opcode("+1034")	op.name == "READ"	op.name == "READ"
test_noop	Get the name of an invalid Opcode	Get Name	Opcode("+9999")	op.name == "NOOP"	op.name == "NOOP"
test_plus	Get the sign of a positive Opcode	Get Sign	Opcode("+1234")	op.sign == "+"	op.sign == "+"
test_minus	Get the sign of a negative Opcode	Get Sign	Opcode("-1234")	op.sign == "-"	op.sign == "-"
test_operand	Get the operand of an Opcode	Get Operand	Opcode("+1234")	op.operand == "34"	op.operand == "34"
test_str_from_minus	String representation of a negative Opcode	String Representation	Opcode("-1234")	str(op) == "-1234"	str(op) == "-1234"
test_str_from_	String	String	str(op) ==	str(op) ==	str(op) ==

plus	representation of a positive Opcode	Representation	"+1234"	"+1234"	"+1234"
test_str_from_no_sign	String representation of an Opcode without sign	String Representation	Opcode("1234")	str(op) == "+1234"	str(op) == "+1234"
test_basic_equality	Equality comparison of two identical Opcodes	Equality	Opcode("+0000"), Opcode("+0000")	op1 == op2	op1 == op2
test_basic_inequality	Inequality comparison of two different Opcodes	Equality	Opcode("+1000"), Opcode("+0001")	op1 != op2	op1 != op2
test_basic_add	Addition of two Opcodes	Addition	Opcode("+0000"), Opcode("+1000")	result == Opcode("+1000")	result == Opcode("+1000")
test_add_negative_and_positive	Addition of a positive and a negative Opcode	Addition	Opcode("+0000"), Opcode("-1000")	result == Opcode("-1000")	result == Opcode("-1000")
test_add_negative_and_negative	Addition of two negative Opcodes	Addition	Opcode("-1000"), Opcode("-1000")	result == Opcode("-2000")	result == Opcode("-2000")
test_overflow	Addition resulting in overflow	Addition	Opcode("+9999"), Opcode("+0001")	OverflowError	Raises OverflowError
test_basic_subtraction	Subtraction of two Opcodes	Subtraction	Opcode("+0000"), Opcode("+1000")	result == Opcode("-1000")	result == Opcode("-1000")
test_subtract_a_negative	Subtraction of a negative Opcode	Subtraction	Opcode("-9999"), Opcode("+0001"	OverflowError	result == Opcode("+1000")

			)		
test_underflow	Subtraction resulting in underflow	Subtraction	Opcode("+0000"), Opcode("+0001")	result == Opcode("+0000")	Raises OverflowError
test_mul	Multiplication of two Opcodes	Multiplication	Opcode("+0001"), Opcode("-0001")	result == Opcode("-0001")	result == Opcode("+0000")
test_mul_neg	Multiplication of a positive and a negative Opcode	Multiplication	Opcode("+0004"), 2	result == Opcode("+0002")	result == Opcode("-0001")
test_integer_div	Division of an Opcode by an integer	Division	Opcode("+0004"), 2	result == Opcode("+0002")	result == Opcode("+0002")
test_opcode_div	Division of two Opcodes	Division	Opcode("+0004"), Opcode("+0002")	result == Opcode("+0002")	result == Opcode("+0002")
test_opcode_div_with_remainder	Division of two Opcodes with remainder	Division	Opcode("+0005"), Opcode("+0002")	result == Opcode("+0002")	result == Opcode("+0002")

test_io.py:

Test Name	**Description**	**Use Case Reference**	**Inputs**	**Expected Outputs**	**Pass/Fail Criteria**
test_init	Initialize IODevice	Initialization	IODevice()	io_device exists	io_device is not None
test_read	Test reading from IODevice	Read Operation	io_device.read = lambda: "1234"	"1234"	io_device.read() == "1234"

test_write	Test writing to IODevice	Write Operation	io_device.write = lambda x: x, "1234"	"1234"	io_device.write("1234") == "1234"

Other Documentation

Our README.txt file exists on our GitHub at the following link:
https://github.com/alex0112/abacus

Click the link below to see a detailed view of the proposed Architecture of our UVSim machine:
https://github.com/alex0112/abacus/blob/main/ARCH.md

Meeting Reports - [Sprint 2]

Meeting Report for 05/27/2024

Attendees:

- Scott Mottola
- Jackson Jacobson
- Jordan Paxman
- Alex Larsen
- Kainny Godinez

Action Items:

- Task Creation: Develop a comprehensive list of tasks necessary for project progression.
- Task Assignment: Ensure each team member selects a task from the list by the next meeting.

Next Meeting: Scheduled for Thursday, 05/30/2024, at 6:30 PM to follow up on task assignments and initial progress.

Follow-up Meeting: 05/30/2024

A follow-up meeting was conducted successfully. A list of issues has been created and assigned in the GitHub repository for effective tracking and accountability.

Task Assignments:

- Scott Mottola: Implement opcodes for MULTIPLY, DIVIDE, SUBTRACT, and ADD.
- Jackson Jacobson: Manage the opcodes for STORE, LOAD, READ, and WRITE.
- Jordan Paxman: Develop the opcodes class.
- Alex Larsen: Focus on developing the CPU abstraction layer.
- Kainny Godinez: Construct the memory abstraction layer.

To-Do Before Next Meeting:

- Code Implementation: Each member is to advance on their assigned coding tasks.

Next Meeting: Scheduled for Tuesday, 06/04/2024, at 6:30 PM. This meeting will review progress on coding tasks and address any emerging issues.

Additional Meeting: 06/06/2024

Attendees:

- Scott Mottola
- Jackson Jacobson
- Jordan Paxman
- Alex Larsen
- Kainny Godinez

Purpose:

- Code Review and Integration: Review the integration of all individual code contributions, ensure cohesion, and address any areas of concern.
- Merge Conflict Resolution: Identify and resolve any merge conflicts that have arisen during the integration process.
- Testing: Verify that all unit tests are functioning correctly post-integration.

Specific Responsibilities:

- **Alex Larsen**: Tasked with configuring the framework of our code to ensure our project supports all integrated components effectively from all of the teams contributions.
- **Jackson Jacobson**: Responsible for compiling all of the unit tests together and the unit test spreadsheet. He also created our Group Project Milestone 2 submission file.
- **Scott Mottola:** Focused on ensuring that his last assigned tasks were ready to be merged and function correctly with the integrated code, he additionally worked on unit tests for his arithmetic cases. Additionally, he also created our Design Document for this project.
- **Jordan Paxman and Kainny Godinez**: Focus on ensuring that their last assigned tasks are ready to be merged and function correctly with the integrated code. Each of them has also created tests for their respective code sections to ensure functionality and compatibility.