Assembly and Machine Language Design Document:

1. Instructions and their opcodes

Before actually coding the interpreter and the assembler, we had first to choose which kinds of opcodes to add to our language given the constraints we were given. Since we used only one digit for the opcode, there were only 20 possible permutations (since the number was signed), which limited to a great extent the flexibility of the language we were about to create. After taking some inspiration from assembly languages such as x86, here is the list of the instructions with their respective opcodes:

Instruction Name	AL Opcode	ML Opcode
Move	MOV	+0
Add	ADD	+1
Subtract	SUB	-1
Multiply	MUL	+2
Divide	DIV	-2
Square	SQR	+3
Square root	SQRT	-3
Equal	EQU	+4
Not equal	NEQU	-4
Greater than or equal	GTE	+5
Less than	SLT	-5
Assign to array	ATA	+6
Copy from array	CFA	-6
Read	READ	+7
Print	PRNT	-7
Assign Label	ASG	
Halt	HALT	+9

2. General Instruction Format:

1. Assembly language:

We chose to allocate a variable length for each opcode in the assembly language since we had no size restrictions. For the opcodes, the string length for labels and symbols is also variable. Still, for immediate values (i.e., digits), we chose to limit it to 4 since, in our machine language, only 4 bits are allocated for each opcode. Hence the maximum value a digit can have is 9999.

For the addressing modes, we chose two:

- Either immediate addressing mode, in which case the operand directly holds the value.
- Direct memory addressing, in which case the operand is either a label or a symbol, is later swapped with the memory address it references while being converted to the machine code.

→ Following these design choices, the first operand can either be an immediate numeric value or a symbol. In contrast, the second operand can either be an immediate numeric value, a symbol, or a label. The only exception is in the "Assign label" instruction, whose first operand is the label name and the second operand is always 0000.

The table below helps to visualize the instructions:

opcode	operand 1	operand 2

2. Machine Language:

According to the specifications of the project, we had ten digits available for instruction. As such, we divided that open space between the different parts of the instruction.

- Operands need four digits of space, and since in this project we are required to have two of them, we end up with two numbers left.
- Keeping in mind that we need one digit for the opcode, we were left with one unused operand. After some discussions, we agreed that this digit could help us determine whether the operands used in the instruction are actual values or addresses. As such, we used it as some kind of memory access mode indicator. To illustrate, please refer to the list below.

Note that we will be using the word “flag” to reference the latter digit as it is more reflective of its nature.

- If the flag has a value of 0, then both operands are immediate values.
- If the flag has value 1, then both operands are memory locations.
- If the flag has value 2, then the first operand is an immediate value, and the second operand is a memory location.
- If the flag has value 3, then the first operand is a memory location, and the second operand is an immediate value.

The table below helps to visualize the instructions:

opcode	flag	a four-digit operand	a four-digit operand

3. Descriptions of the instructions:

In this description, we will use X to designate the aforementioned flag.

- **Move:**

  General Assembly format: MOV OP1 OP2
  General ML format: +0X OP1 OP2

  Operation performed:

  1. If the first operand is an immediate value: Initializes the accumulator’s content to the value of op1. In this case, the second operand holds the value 0000.
  2. If both operands are memory addresses: Initializes the memory location’s content indicated in op2 to the value of the memory location indicated in op1.
  3. If the second operand is a memory address: Initializes the memory location’s content specified in op2 to the accumulator’s value. In this case, the first operand holds the value 0000.
  4. If the first operand is a memory location: initialize the accumulator’s content to the value of the memory location specified in op1. In this case, the second operand holds the value 0000.

- **Add:**

General Assembly format: ADD OP1 OP2
General ML format: +1X OP1 OP2

The operation performed: In general, the second operand always specifies a memory address location. Therefore we only have two cases:

1. If both operands are both memory locations: Adds the content of the memory location specified in op1 to the value in the accumulator and stores the result in the memory location specified in op2.
2. If the first operand is an immediate value: Adds the value of op1 to the value in the accumulator and store the result in the memory location specified in op2.

- **Subtract:**

General Assembly format: SUB OP1 OP2
General ML format: -1X OP1 OP2

Operation performed:

1. If both operands are both memory locations: Subtracts the content of the memory location specified in op1 to the value from the value in the accumulator and stores the result in the memory location specified in op2.
2. If the first operand is an immediate value: Subtracts the value of op1 from the value in the accumulator and stores the result in the memory location specified in op2.

- **Multiply:**

General Assembly format: MUL OP1 OP2
General ML format: +2X OP1 OP2

Operation performed:

1. If both operands are both memory locations: Multiplies the content of the memory location specified in op1 to the value by the value in the accumulator and stores the result in the memory location specified in op2.
2. If the first operand is an immediate value: Multiplies the value of op1 by the value in the accumulator and stores the result in the memory location specified in op2.

- **Divide**:

  General Assembly format: DIV OP1 OP2
  General ML format: -2X OP1 OP2

  Operation performed:

  1. If both operands are both memory locations: Divides the content of the memory location specified in op1 to the value by the value in the accumulator and stores the result in the memory location specified in op2.
  2. If the first operand is an immediate value: It divides the value of op1 by the value in the accumulator and stores the result in the memory location specified in op2.

- **Square:**

  General Assembly format: SQR OP1 OP2
  General ML format: +3X OP1 OP2

  Operation performed:

  1. If the first operand is a memory location: Squares the content of memory location specified in op1 and stores the result in the memory location specified in op2.
  2. If the first operand is an immediate value: Squares the value of op1 and stores the result in the memory location specified in op2.

- **Square root:**

  General Assembly format: SQRT OP1 OP2
  General ML format: -3X OP1 OP2

  Operation performed:

  1. If the first operand is a memory location: Square roots the content of memory location specified in op1 and stores the result in the memory location specified in op2.
  2. If the first operand is an immediate value: Square roots the value of op1 and stores the result in the memory location specified in op2.

- **Equal:**

General Assembly format: EQU OP1 OP2
General ML format: +4X OP1 OP1

The operation performed: The second operand always holds the memory address that we need to jump to. Therefore, we have two cases:

1. Suppose the first operand is a memory address: Checks whether the value in the memory address specified in op1 is equal to the accumulator's value. If that's the case, then it jumps to the memory location specified in op2.
2. Suppose the first operand is an immediate value: Checks whether the value in op1 is equal to the accumulator's value. If that's the case, then it jumps to the memory location specified in op2.

- **Not equal:**

General Assembly format: NEQU OP1 OP2
General ML format: -4X OP1 OP2

Operation performed:

1. Suppose the first operand is a memory address: Checks whether the value in the memory address specified in op1 is not equal to the accumulator's value. If that's the case, then it jumps to the memory location specified in op2.
2. Suppose the first operand is an immediate value: Checks whether the value in op1 is not equal to the accumulator's value. If that's the case, then it jumps to the memory location specified in op2.

- **Greater than or equal:**

General Assembly format: GTE OP1 OP2
General ML format: +5X OP1 OP2

Operation performed:

1. Suppose the first operand is a memory address: Checks whether the value in the memory address specified in op1 is greater than or equal to the accumulator's value. If that's the case, then it jumps to the memory location specified in op2.
2. Suppose the first operand is an immediate value: Checks whether the value in op1 is greater than or equal to the accumulator's value. If that's the case, then it jumps to the memory location specified in op2.

- **Less than:**

  General Assembly format: LT OP1 OP2
  General ML format: -5X OP1 OP2

  Operation performed:

  1. If the first operand is a memory address: Checks whether the value in the memory address specified in op1 is less than the value in the accumulator. If that's the case, then it jumps to the memory location specified in op2.
  2. If the first operand is an immediate value: Checks whether the value in op1 is less than the value in the accumulator, if that's the case, then it jumps to the memory location specified in op2.

- **Assign to array:**
  General Assembly format: ATA OP1 OP2
  General ML format: +63 OP1 OP2

  The operation performed: It computes the memory location by adding the index value specified in op1 to the array address specified in op2 and stores the accumulator's content in the memory location calculated.
- **Copy from array:**

  General Assembly format: CFA OP1 OP2
  General ML format: -63 OP1 OP2

  The operation performed: Copies the memory computed content by adding the index specified in op1 to the array address specified in op2 into the accumulator.

- **Read:**

  General Assembly format: READ OP1 0000
  General ML format: +73 OP1 0000

  The operation performed: Gets the input from the input part and stores it in the address specified in op1. In this case, the value in op2 is 0000.

- **Print:**

  General Assembly format: PRNT OP1 0000
  General ML format: -7X OP1 0000

The operation performed: We only make use of the first operand. In this case, the value of the second operand is 0000:

1. The first operand is a memory address: It gets the value from the memory address specified in op1 and displays it to the screen.
2. In case the first operand is an immediate value: It displays the value in op1 to the screen.

- **Assign label:**

  General Assembly format: ASG OP1 0000

  The operation performed: It assigns the label specified in OP1 to the line after it.

- **Halt:**

  General Assembly format: HALT 0000 0000
  General ML format: +9X 0000 0000

  The operation performed: It stops the execution of the program.