Opcode Project

Microprocessors are typically programmed using an instruction set to achieve the desired logic for computation. We want to build a prototype of the instruction set for our custom microprocessor. We will implement the sub-set of instructions today in a simulator.

Our prototype microprocessor has 4 registers namely A, B, C & D (which can store 32-bit signed integers). It supports the following instructions

Instruction	**Explanation**	**Comments**
SET A 10	A = 10	This instruction sets the value of register A to 10 (which is an integer value) The value can be negative since the Register can store signed integer values.
ADR C D	C = C + D	This instruction adds the content of register D into register C. The updated value is stored back in Register C.
ADD A 12	A = A + 12	This instruction adds the integer 12 to the existing value as stored in Register A, the updated value is stored back in Register A. The value (12) can be negative as well in which case, the ADD acts as a subtraction operation.
MOV A B	A ← B	This instruction moves/updates the value of register A with the value which was stored in Register B. The value of register B remains unchanged. Ex: If A = 10, B = 15, then MOV A B would result in A = 15, B = 15

INR C	C= C + 1	This instruction increments the value stored in register C by 1. The updated value is stored back in register C.
DCR A	A = A - 1	This decrements the value of register A. The updated value is stored back in register A.

RST	A = 0
	B = 0
	C = 0
	D = 0
		This command resets all the currently stored values across all the available registers (A, B, C, and D)

Note - The above instructions deviate from the original specifications of the 8085 or its successors. Since this is a prototype processor, we have simplified/modified the instruction set.

Expectations

- You can implement the functionality in one of the programming languages (Java, Golang, Ruby, JS, Python, or C#)
- Please don’t create REST APIs or any UI for the same.
- You can test the functionality using unit test cases.
  - The addition of the unit test cases is desired and expected. Please avoid using main methods or driver methods to test the functionality.
- The focus is on the core functionality and how it is implemented v/s how it is presented (UI or API structure).
- Please use the coding best practices (SOLID, DRY, etc.) and ensure that you are writing modular & extensible code (to accommodate any future changes in functionality)
  - Ex: What if we want to support logical operations like AND, OR, XOR, etc. how is the implementation open to allow these extensions?
  - Ex: What if we want to add more registers - E & F, how can you take care of this extension/stretch?
- The submitted code would be judged on both coding best practices and functional correctness.