Instruction Set Architecture Simulator

In any programming language, write a simulator for your own instruction set architecture with at least 4 general-purpose 64-bit registers (it is allowed to use data types of length 8 bytes, such as long long or uint64_t for registers). The simulator should function as an interpreter. It should be possible to load the source assembly code from a file, and properly perform the necessary lexical, syntactic, and semantic analysis of the code. The instruction set of the simulated machine (guest) must include:

Basic arithmetic operations (ADD, SUB, MUL, DIV)

Basic bitwise logical operations (AND, OR, NOT, XOR)

Instruction for moving data between registers (MOV)

Instruction for inputting data from standard input (similar to the corresponding system call)

Instruction for outputting data to standard output (similar to the corresponding system call)

Implement a simple single-step debugging support. Enable execution and viewing of the values of all registers and specified memory addresses at set breakpoints in the assembly code during execution. In this mode, it should be possible to step to the next instruction (NEXT or STEP console commands) and jump to the next breakpoint (CONTINUE).

Implement memory management. The simulated machine (guest) should have a 64-bit address space. Enable direct and indirect addressing. The contents of each memory address should be 1 byte in length. Properly allow addresses to be specified as numbers. Allow access to all addresses in the address space, including read and write, using the appropriate instructions (MOV or LOAD/STORE).

Implement the instructions needed for unconditional and conditional branching (JMP, CMP, JE, JNE, JGE, JL).

Instead of directly interpreting strings, allow assembly instructions to be translated into guest machine code (bytecode) and stored in the guest's address space (analogous to the code or text segment). Therefore, instructions should be interpreted and executed from the guest's memory. Define and use a register that will serve as the program counter (instruction pointer). Construct an example of assembly code for the guest that is self-modifying (after being translated into guest machine code) and in which the modified portion of the code is executed both before and after the relevant modification.