Assembly Simulator

Assembly Simulator

Assembly Language is a low-level programming language that is specific to a particulars computer architecture. Each command has a specific structure:

    label COMMAND OPERANDS

The commands are represented by mnemonics and perform low-level arithmetic and logical operations generally computed by an Arithmetic-Logic-Unit (ALU). Operands include constants, registers, and addresses that can trigger operations. Labels are used to mark the line of a command so that the ALU can jump or branch to that location to implement higher-level operations such as an if-statement or loop.

Task

Write an assembly language emulator for the Xtreme-8000 processor. The processor consists of an ALU and a register to store comparisons (only set using the COMP command). There are no accumulators, temporary data registers, or temporary address registers. The processor has a maximum of 256-byte memory that it can address through its commands. The full list of commands is:

`PRINT A1`	Print the hexadecimal byte stored at memory location A1.
`PRINT A1,A2`	Print the hexadecimal bytes stored in locations A1 to A2 (both addresses are inclusive).
`MOVE #N,A1`	Move constant N to memory location A1.
`MOVE #N,(A1)`	Move constant N to the dereferenced address A1.
`MOVE (A1),A2`	Move the dereferenced address A1 into A2.
`MOVE (A1),(A2)`	Move the dereferenced address A1 into the dereferenced address A2.
`MOVE A1,(A2)`	Move the byte at A1 to the dereferenced address A2.
`MOVE A1,A2`	Move byte from memory location A1 to A2.
`ADD #N,A1`	Add constant N to the value stored at A1. Result stored in A1.
`ADD A1,A2`	Add value from A1 to A2. Result stored in A2.
`SUB #N,A1`	Subtract constant N from the value stored at A1 and store the result in A1.
`SUB A1,A2`	Subtract value A1 from A2, and store the result in A2.
`AND #N,A1`	Logical AND between constant N and A1. Result stored in A1.
`AND A1,A2`	Logical AND between values at A1 and A2. Result stored in A2.
`OR #N,A1`	Logical OR between constant N and A1. Result stored in A1.
`OR A1,A2`	Logical OR between values at A1 and A2. Result stored in A2.
`XOR #N,A1`	Logical XOR between constant N and A1. Result stored in A1.
`XOR A1,A2`	Logical XOR between values at A1 and A2. Result stored in A2.
`COMP #N,A1`	Compare constant N to A1. Result stored in the processor to use in the following branch operation.
`COMP A1,A2`	Compare values stored at A1 and A2. Result stored in the processor to use in the following branch operation.
`BEQ label`	Branch to label if result of comparison is equal.
`BNE label`	Branch to label if result of comparison is not equal.
`BGT label`	Branch to label if result of comparison is true (A1 > A2).
`BLT label`	Branch to label if result of comparison is true (A1 < A2).
`BGE label`	Branch to label if result of comparison is true (A1 >= A2).
`BLE label`	Branch to label if result of comparison is true (A1 <= A2).

Input

The input will contain:

- 0 <= Size of memory in bytes <= 0xFF
- Program to execute
  - Each line has an optional label, a command, and a list of comma separated operands (as needed for each command).

Output

Output the result of any print statement.

Note: There is a newline character at the end of the last line of the output.

Sample Input 1

```
0F
PRINT 00,0F
MOVE #FF,00
PRINT 00,0F
MOVE 00,01
PRINT 00,0F
ADD 00,02
PRINT 00,0F
ADD 01,02
PRINT 00,0F
```

Sample Output 1

```
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FF 00 00 00 00 00 00 00 00 00 00 00 00 00
FF FF FE 00 00 00 00 00 00 00 00 00 00 00 00 00
```