F • Microprocessor Simulation

Problem

Consider a small microprocessor that has the following properties:

- Each word is four bits.
- Addresses are two words. The high word always comes first. That is, the high word of a two-word address will always occupy the lower word of memory.
- Memory is 256 words.
- There are two accumulators, A and B, each storing one word.
- There are nine instruction codes. Each instruction requires at least one word to store the code that specifies the instruction. Four instructions have arguments and require an additional two words.

Each 4 bit number can have the values from 0 to 15, inclusive, in base 10. We will write these using hexadecimal in the usual way, i.e. A means 10, B means 11, etc.

These are the nine instructions:

Code	Words	Description
0	3	LD: Load accumulator A with the contents of memory at the specified argument.
1	3	ST: Write the contents of accumulator A to the memory location specified by the argument.
2	1	SWP: Swap the contents of accumulators A and B.
3	1	ADD: Add the contents of accumulators A and B. The low word of the sum is stored in A, and the high word in B.
4	1	INC: Increment accumulator A. Overflow is allowed; that is, incrementing F yields 0.
5	1	DEC: Decrement accumulator A. Underflow is allowed; that is, decrementing 0 yields F.
6	3	BZ: If accumulator A is zero, the next command to be executed is at the location specified by the argument. If A is not zero, the argument is ignored and nothing happens.
7	3	BR: The next command to be executed is at the location specified by the argument.
8	1	STP: Stop execution of the program.

The microprocessor always begins by executing the command at location 00. It executes the commands in sequence until it reaches the Stop command.

The examples below show partial programs and describe their affect.

Program	Description
01A8	Load accumulator A with the contents of memory location 1A (26 in decimal) and stop.
01A512F8	Load accumulator A with the contents of memory location 1A (26 in decimal), decrement it, store the result to memory location 2F, then stop.

The input will consist of several lines of exactly 256 hex characters. Each line is the contents of memory, beginning with address 00 and ending with address FF. The end of the input is indicated by a memory state that has a stop instruction (an "8") at address 00. The input programs will never "fall of the end of memory", that is, you will never execute an instruction that is located between addresses F0 and FF, inclusive.

Output

For each memory state, you should simulate execution beginning with address 00. When the stop instruction is reached, you will dump the contents of memory to the output as a single string of 256 hex characters followed by a newline character.

Examples

This program reads two one-word numbers from 10 and 11, and stores the two-word sum at 12 and 13.

```
0102011311321128FF00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
```

The correct output is:

```
0102011311321128FF1E00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000
```

This program computes n^2, where n is stored at 40, and stores the two-word result at FE and FF.

```
040563B14004220FF31FF041320FE31FE00C2042314200032041314170080000F03000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000001
```

The correct output is:

```
040563B14004220FF31FF041320FE31FE00C2042314200032041314170080000011F00000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
00000000000000000000000000000000000000000000000000000000000000000000000000000000E1
```