(CO assignment?) Simple Computers

You are to write an interpreter for a simple computer. This computer uses a processor with a small number of machine instructions. Furthermore, it is equipped with 32 byte of memory, one 8-bit accumulator (accu) and a 5-bit program counter (pc). The memory contains data as well as code, which is the usual von Neumann architecture.

The program counter holds the address of the instruction to be executed next. Each instruction has a length of 1 byte - the highest 3 bits define the type of instruction and the lowest 5 bits define an optional operand which is always a memory address (`xxxxx`). For instructions that don't need an operand the lowest 5 bits have no meaning (`-----`). Here is a list of the machine instructions and their semantics:

`000xxxxx`	`STA x`	store the value of the accu into memory byte x
`001xxxxx`	`LDA x`	load the value of memory byte x into the accu
`010xxxxx`	`BEQ x`	if the value of the accu is 0 load the value x into the pc
`011-----`	`NOP`	no operation
`100-----`	`DEC`	subtract 1 from the accu
`101-----`	`INC`	add 1 to the accu
`110xxxxx`	`JMP x`	load the value x into the pc
`111-----`	`HLT`	terminate program

In the beginning, program counter and accumulator are set to 0. After fetching an instruction but before its execution, the program counter is incremented. You can assume that programs will terminate.

Input Specification

The input file contains several test cases. Each test case specifies the contents of the memory prior to execution of the program. Byte 0 through 31 are given on separate lines in binary representation. A byte is denoted by its highest-to-lowest bits. Input is terminated by EOF.

Output Specification

For each test case, output on a line the value of the accumulator on termination in binary representation, again highest bits first.

Sample Input

```
00111110
10100000
01010000
11100000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00111111
10000000
00000010
11000010
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
11111111
10001001
```

Sample Output

```
10000111
```