C. Azembler

time limit per test: 5 seconds
memory limit per test: 256 megabytes
input: standard input
output: standard output

After the Search Ultimate program that searched for strings in a text failed, Igor K. got to think: "Why on Earth does my program work so slowly?" As he double-checked his code, he said: "My code contains no errors, yet I know how we will improve Search Ultimate!" and took a large book from the shelves. The book read "Azembler. Principally New Approach".

Having carefully thumbed through the book, Igor K. realised that, as it turns out, you can multiply the numbers dozens of times faster. "Search Ultimate will be faster than it has ever been!" — the fellow shouted happily and set to work.

Let us now clarify what Igor's idea was. The thing is that the code that was generated by a compiler was far from perfect. Standard multiplying does work slower than with the trick the book mentioned.

The Azembler language operates with 26 registers (eax, ebx, ..., ezx) and two commands:

- $[x]$ — returns the value located in the address x. For example, [eax] returns the value that was located in the address, equal to the value in the register eax.
- lea x, y — assigns to the register x, indicated as the first operand, the second operand's address. Thus, for example, the "lea ebx, [eax]" command will write in the ebx register the content of the eax register: first the [eax] operation will be fulfilled, the result of it will be some value that lies in the address written in eax. But we do not need the value — the next operation will be lea, that will take the [eax] address, i.e., the value in the eax register, and will write it in ebx.

On the first thought the second operation seems meaningless, but as it turns out, it is acceptable to write the operation as

lea ecx, [eax + ebx],

lea ecx, [k*eax]

or even

lea ecx, [ebx + k*eax],

where k = 1, 2, 4 or 8.

As a result, the register ecx will be equal to the numbers eax + ebx, k*eax and ebx + k*eax correspondingly. However, such operation is fulfilled many times, dozens of times faster that the usual multiplying of numbers. And using several such operations, one can very quickly multiply some number by some other one. Of course, instead of eax, ebx and ecx you are allowed to use any registers.

For example, let the eax register contain some number that we should multiply by 41. It takes us 2 lines:

lea ebx, [eax + 4*eax] // now ebx = 5*eax

lea eax, [eax + 8*ebx] // now eax = eax + 8*ebx = 41*eax

Igor K. got interested in the following question: what is the minimum number of lea operations needed to multiply by the given number n and how to do it? Your task is to help him.

Consider that at the initial moment of time eax contains a number that Igor K. was about to multiply by n, and the registers from ebx to ezx contain number 0. At the final moment of time the result can be located in any register.

Input

The input data contain the only integer n ($1 \le n \le 255$), which Igor K. is about to multiply.

Output

On the first line print number p, which represents the minimum number of lea operations, needed to do that. Then print the program consisting of p commands, performing the operations. It is guaranteed that such program exists for any n from 1 to 255.

Use precisely the following format of commands (here k is equal to 1, 2, 4 or 8, and x, y and z are any, even coinciding registers):

lea x, [y]

lea x, [y + z]

lea x, [k*y]

lea x, [y + k*z]

Please note that **extra spaces at the end of a command are unacceptable**.

Examples

input

```
41
```

output

```
2
lea ebx, [eax + 4*eax]
lea ecx, [eax + 8*ebx]
```

input

```
2
```

output

```
1
lea ebx, [eax + eax]
```

input

```
4
```

output

```
1
lea ebx, [4*eax]
```