214 Code Generation

Your employer needs a backend for a translator for a very SIC machine (Simplified Instructional Computer, apologies to Leland Beck). Input to the translator will be arithmetic expressions in postfix form and the output will be assembly language code.

The target machine has a single register and the following instructions, where the operand is either an identifier or a storage location.

L	load the operand into the register
A	add the operand to the contents of the register
S	subtract the operand from the contents of the register
M	multiply the contents of the register by the operand
D	divide the contents of the register by the operand
N	negate the contents of the register
ST	store the contents of the register in the operand location

An arithmetic operation replaces the contents of the register with the expression result. Temporary storage locations are allocated by the assembler for an operand of the form "`$n`" where `n` is a single digit.

Input

The input file consists of several legitimate postfix expressions, each on a separate line. Expression operands are single letters and operators are the normal arithmetic operators (`+`, `-`, `*`, `/`) and unary negation (`@`).

Output

Output must be assembly language code that meets the following requirements:

1. One instruction per line with the instruction mnemonic separated from the operand (if any) by one blank.
2. One blank line must separate the assembly code for successive expressions.
3. The original order of the operands must be preserved in the assembly code.
4. Assembly code must be generated for each operator as soon as it is encountered.
5. As few temporaries as possible should be used (given the above restrictions).
6. For each operator in the expression, the minimum number of instructions must be generated (given the above restrictions).

Sample input

```
AB+CD+EF++GH+++
AB+CD+-
```

Sample output

```
L A
A B
ST $1
L C
A D
ST $2
L E
A F
A $2
ST $2
L G
A H
A $2
A $1

L A
A B
ST $1
L C
A D
N
A $1
```