BPC 1 S4 - Binary Matrices

Time limit: 1.0s **Memory limit:** 256M

We can use an $N \times N$ binary matrix to encode unsigned integers with N^2 bits in the following way: Let $a_{i,j}$ $(1 \leq i, j \leq N)$ be the cell's value in the matrix at row i and column j. $a_{i,j} \in \{0, 1\}$. The value represented by the binary matrix is the sum of $a_{i,j} \times 2^{(N-i)N+N-j}$ for all $1 \leq i, j \leq N$. Less formally, the cell in the top left corner is the most significant bit, the cell in the 1^{st} row 2^{nd} column is the second most significant, and the cell in the 1^{st} row 3^{rd} column is the third most significant, etc.

You have pieces of memory called registers, where each register stores a 32×32 binary matrix. Each register is identified by an uppercase letter from A to Z. Registers A and B initially contain arbitrary values, while the rest contain all zeros. You are to output a program that sorts A and B, setting `A := min(A, B)` and `B := max(A, B)`. You may use other registers in the process, but their final value does not matter. Comparing two binary matrices is defined as comparing the integer values they represent.

You may only use instructions of the following types. All instructions first compute the value you would get by applying the operation on the two source operands, then set the destination operand equal to this value. All destination operands are registers, identified by a letter from A to Z. Source operands can be registers or integer constants based on the type of operation. Integer constants must be in the range $[0, 31]$.

- `OR DST S1 S2` - calculates the bitwise OR of matrices $S1$ and $S2$.
- `AND DST S1 S2` - calculates the bitwise AND of matrices $S1$ and $S2$.
- `XOR DST S1 S2` - calculates the bitwise XOR of matrices $S1$ and $S2$.
- `LEFT DST S1 X2` - shifts every bit in $S1$ left by $X2$ positions. Zeros will be shifted in from the right as necessary, setting the rightmost $X2$ bits in each row to 0.
- `RIGHT DST S1 X2` - shifts every bit in $S1$ right by $X2$ positions. Zeros will be shifted in from the left as necessary, setting the leftmost $X2$ bits in each row to 0.
- `UP DST S1 X2` - shifts every bit in $S1$ up by $X2$ positions. Zeros will be shifted in from the bottom as necessary, setting the $X2$ bits closest to the bottom of each column to 0.
- `DOWN DST S1 X2` - shifts every bit in $S1$ down by $X2$ positions. Zeros will be shifted in from the top as necessary, setting the $X2$ bits closest to the top of each column to 0.
- `ADD DST S1 S2` - performs row-wise addition on matrices $S1$ and $S2$. Specifically, each row in each matrix is converted to a 32-bit integer, with the leftmost bit being the most significant, and corresponding rows from each matrix are added together. If the result of this addition is greater than or equal to 2^{32} for a specific row, the overflow bit is discarded. Note that, because of this, this operation is not equivalent to adding together the total values of the matrices.

Input Specification

There is no input. The program you output should work for arbitrary initial values of A and B and will not know these values in advance.

Output Specification

You should output at most $1\,000$ instructions, each on a separate line. The instructions will be executed in the order you output them.

Additionally, the checker for this problem will have the same policy on whitespace as the `standard` checker. This means that trailing whitespace, empty lines, and missing a newline character at the end of the last line are tolerated. This is required to allow the submission of plain text files (`Text` language in the submission editor) since the editor strips trailing newline characters at the end of the file, which would make it impossible for plain text submissions to pass under an `identical` checker.

You will receive a verdict of `Presentation Error` if there was anything wrong with the format of your output. You will only receive `Wrong Answer` if your output corresponded to a set of valid instructions and it sorted the values in registers A and B incorrectly.

Scoring

Your score will depend on the total number of different registers your program accesses.

Number of registers used	Score
≥ 8	60%
7	70%
6	80%
5	90%
4	100%

Sample Output

```
XOR C A B
ADD D C B
RIGHT Z C 2
DOWN Z Z 1
```

Explanation for Sample

The sample output would not receive AC on the problem since it does not sort the matrices in registers A and B. It is provided only to clarify the output format. Here is what the operations would do if registers stored 4×4 matrices. Note that the real test data will only have registers containing 32×32 matrices as specified earlier.

Initial state:

```
A       B       C       D       Z
0010    1100    0000    0000    0000
1101    0110    0000    0000    0000
0110    0000    0000    0000    0000
0010    1011    0000    0000    0000
```

After `XOR C A B`:

```
A       B       C       D       Z
0010    1100    1110    0000    0000
1101    0110    1011    0000    0000
0110    0000    0110    0000    0000
0010    1011    1001    0000    0000
```

After `ADD D C B`:

```
A       B       C       D       Z
0010    1100    1110    1010    0000
1101    0110    1011    0001    0000
0110    0000    0110    0110    0000
0010    1011    1001    0100    0000
```

After `RIGHT Z C 2`:

```
A       B       C       D       Z
0010    1100    1110    1010    0011
1101    0110    1011    0001    0010
0110    0000    0110    0110    0001
0010    1011    1001    0100    0010
```

After `DOWN Z Z 1`:

```
A       B       C       D       Z
0010    1100    1110    1010    0000
1101    0110    1011    0001    0011
0110    0000    0110    0110    0010
0010    1011    1001    0100    0001
```