IEEEXTREME PROGRAMMING COMPETITION 14.0

ARM Constant Multiplication

Time limit: *1000 ms*
Memory limit: *256 MB*

In this challenge, we will explore how a compiler for a 32-bit ARM processor may implement multiplication by an unsigned integer constant without using a multiply instruction, which would consume relatively a lot of energy. You are allowed to use the four instructions `MOV`, `ADD`, `SUB`, and `RSB` operating on registers where the last register may be logically shifted left or right (`LSL` or `LSR`). The allowed instructions have the following syntax:

- `MOV Rd, Ra, LSL/LSR #i` : Move the shifted value of `Ra` to `Rd` .
- `ADD Rd, Rb, Ra, LSL/LSR #i` : Add the shifted value of `Ra` to `Rb` , and save the result in `Rd` .
- `SUB Rd, Rb, Ra, LSL/LSR #i` : Subtract the shifted value of `Ra` from `Rb` , and save the result in `Rd` .
- `RSB Rd, Rb, Ra, LSL/LSR #i` : Subtract `Rb` from the shifted value of `Ra` , and save the result in `Rd` . `RSB` stands for Reverse SuBtract.

`Rd` is the destination register where the result of the instruction is saved. `Rb` and `Ra` are operand registers, and their values do not change during the instruction, though `Rd` may be the same as `Rb` and/or `Ra` in which case the value of the register may change. The value of `Ra` is shifted left (`LSL`) or right (`LSR`) by `i` bits before it is used by the instruction. That is, `Ra LSL #i` means the value of `Ra << i` , and `Ra LSR #i` means the value of `Ra >> i` . For `LSL` , the value of `i` must be between 0 and 31 inclusive. For `LSR` , the value of `i` must be between 1 and 32 inclusive. Note that the range of `i` is different for `LSL` and `LSR` . This is because the shifting amount i must fit into 5 bits in an instruction. All registers store 32-bit integers. Bits are discarded when shifted outside the 32-bit range. If you logically shift right by 32 (`LSR #32`), you end up with 0 in the register. Overflow bits are discarded in addition, and underflow bits are discarded in subtraction.

For each test case you will be given a single constant unsigned integer multiplier C. You should return a sequence of ARM instructions that implement the multiplication by this constant multiplier. Assume the multiplicand is in the register `R0` , and the product should end up in the same register `R0` afterwards. That is, if the value of the register `R0` is X in the beginning, its value should become $C \cdot X$ truncated to 32 bits in the end. You are allowed to use 12 additional registers `R1` , `R2` , ... `R12` to save any intermediate results. The values in `R1` , ... `R12` are initially all zeros. You are asked to minimize the number of instructions so that the multiplication can be executed as quickly as possible.

Standard input

The input has a single integer T on the first line, the number of test cases.

Each test case has a single line with one unsigned 32-bit integer, the constant unsigned integer multiplier C.

Standard output

For each test case, output a sequence of instructions to perform the multiplication, one instruction per line. Print a single line with `END` to mark the end of the output for each test case. Each instruction should be one of the four classes given above and follow the provided formats. Note that the shift operation is required and you should set `i` to zero for `LSL` to use the value of `Ra` without any shift.

Scoring

For each test case, your solution will be scored based on k, the number of instructions used ($k \geq 0$). It can be shown that for any C, it is possible to implement the multiplication using no more than 32 instructions. Your submission will get a score of $(1.0 - k/33)^2$ if it correctly performs multiplication and satisfies $k \leq 32$. A submission that performs multiplication incorrectly or has $k > 32$ will receive zero points.

A test file will only receive a positive final score if **every** test case in the test file receives a positive score, in which case the final score of the test file is the average of the scores your solution receives for all the test cases in this test file times 5.

Constraints and notes

- $T = 100$ for all the hidden test files.
- Each C is an unsigned 32-bit integer.

Input	Output	Explanation

`1`
`8`
	`MOV R0, R0, LSL #3`
	`END`
		`R0` is shifted to the left by 3 bits, getting `R0 << 3` . This equals `8 * R0` and is the product saved back to `R0` .

`1`
`9`
	`ADD R0, R0, R0, LSL #3`
	`END`
		The `ADD` instruction adds 8 times `R0` to `R0` , getting `9 * R0` , and saves it back to `R0` .

`2`
`115`
`105`
	`ADD R1, R0, R0, LSL #1`
	`SUB R1, R1, R0, LSL #4`
	`ADD R0, R1, R0, LSL #7`
	`END`
	`RSB R1, R0, R0, LSL #2`
	`MOV R0, R0, LSL #5`
	`ADD R0, R0, R1, LSL #0`
	`RSB R0, R0, R0, LSL #2`
	`END`
		Let the initial value in `R0` be X The `ADD` instruction adds $2X$ from `R0, LSL #1` and X from `R0` , getting $3X$ and saves it to `R1` . The `SUB` instruction gets `R1 - R0 * 16` which is $3X - 16X = -13X$ and saves it in `R1` . The last `ADD` instruction obtains `R1 + 128 * R0` which is $-13X + 128X = 115X$.
		
		Note that in the last test case you can do better using only two instructions:
		• `RSB R0, R0, R0, LSL #4`
		• `RSB R0, R0, R0, LSL #3`
		
		The first `RSB` places `15 * R0` in `R0` . The second `RSB` calculates `7 * R0` . Before the second instruction, `R0` has $15X$, and therefore the second `RSB` will give us $105X$ in `R0` as a result.