_1_

## Write a TM program that prints 1.

*(You need only three lines.)*

Write a TM program that computes 3 + 4 and prints the result.

*(You need only **five** lines.)*

_3_

Write a TM program
that squares its argument
and prints the result.

*(You need only **four** lines!)*

_4_

Same as #3, but the code jumps to the multiplication instruction and jumps back.

*(This took me eight lines…)*

## What does it mean to "jump to an instruction"?

*The caller puts arguments in a specific place and then transfers control.*

*The callee accesses the arguments, does its work, stores its result, and branches back.*

## The Subroutine Design Pattern

call a function:

```
load R1,n
load R0,PC+2
branch SQUARE
```

the function:

```
SQUARE:
  [ use R1 ]
  branch @R0
```

.

## The Subroutine Design Pattern

Without a call stack and
other program overhead,
this is simply ***branch and return***.

## Branch and Return in TM

There is no instruction
to branch unconditionally.

R7 is the program counter.
Load an address there — boom.

*(Only* ***5 + 3*** *lines...)*

_4_

Same as #3, but the code jumps to the multiplication instruction and jumps back.

*(This took me eight lines...)*

_5_

## Same as #4, but by calling both `main(n)` and `square(n)`

*(Fourteen lines for me...)*