

# Simulating a simple CPU

FOCS, Spring 2022

# Turing machines as CPUs

Modern computers are driven by a CPU

If we can show how to simulate a CPU using a Turing machine, we get that whatever a modern computer can do can basically be done by a Turing machine

What's a CPU?

- registers holding finite values
- finite memory holding finite values
- simple instructions to transfer values between registers and memory

# A simple CPU

Arbitrarily many registers, numbered 0, 1, 2, 3, ...

Each register holds an arbitrary natural number  $\geq 0$

Instructions:

- INC  $r$                     increment register  $r$
- DEC  $r$ ,  $addr$             if register  $r > 0$ , decrement it; else, jump to  $addr$
- JMP  $addr$                 jump to  $addr$
- TRUE                      stop and return *true*
- FALSE                    stop and return *false*

# Example: addition

# R0 + R1 =? R2

start:

DEC 0, compare

INC 1

JMP start

compare:

DEC 1, empty

DEC 2, reject

JMP compare

empty:

DEC 2, accept

reject:

FALSE

accept:

TRUE

# Example: multiplication

# R0 \* R1 =? R2

clear3:

# clear R3 = prod

DEC 3, clear4

JMP clear3

clear4:

# clear R4 = temp

DEC 4, loop0

JMP clear4

loop0

DEC 0, compare

loop1

DEC 1, next

INC 3

INC 4

JMP loop1

next:

DEC 4, loop0

INC 1

JMP next

compare:

DEC 3, empty

DEC 2, reject

JMP compare

empty:

DEC 2, accept

reject:

FALSE

accept:

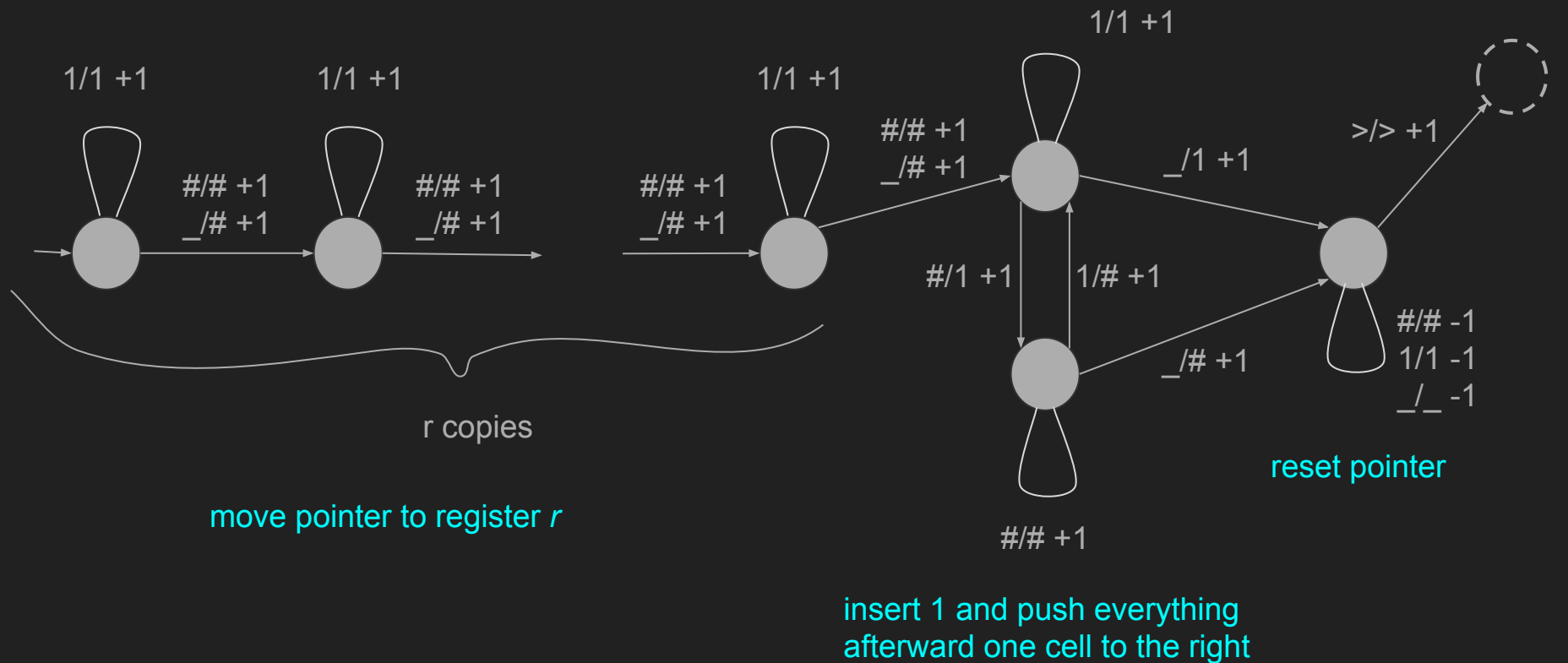
TRUE

# Simulating a program with a Turing machine

Translate a CPU program into a Turing machine

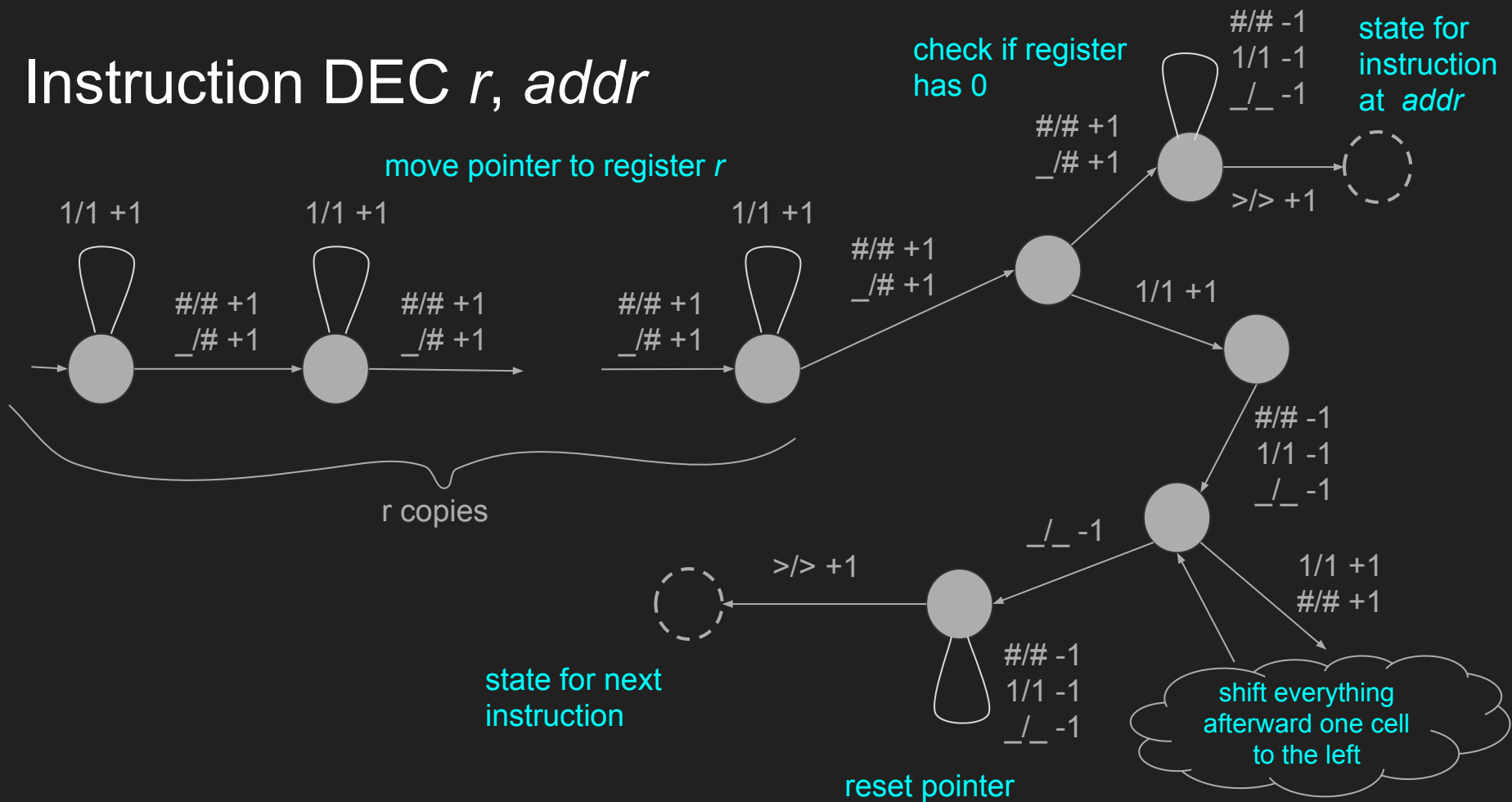
- Translate each instruction is a set of states in the Turing machine
  - Each set of states for an instruction has an "entry" state
- Jumping to an instruction is jumping to the "entry" state of the corresponding set of states
- The tape holds a value for each register
  - $>n_0\#n_1\#n_2\#n_3\#n_4\#\dots$
  - each stored in *unary* for simplicity     $0 = ;$  ;  $1 = 1$ ;  $2 = 11$ ;  $3 = 111$ ;  $10 = 1111111111$ ; ...
- At the beginning of each instruction, tape pointer is on the first register

# Instruction INC $r$



# Instruction DEC $r$ , $addr$

move pointer to register  $r$



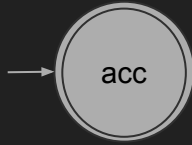


# Instruction JMP *addr*

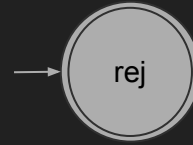


# Instructions TRUE and FALSE

TRUE



FALSE



# Example

start:

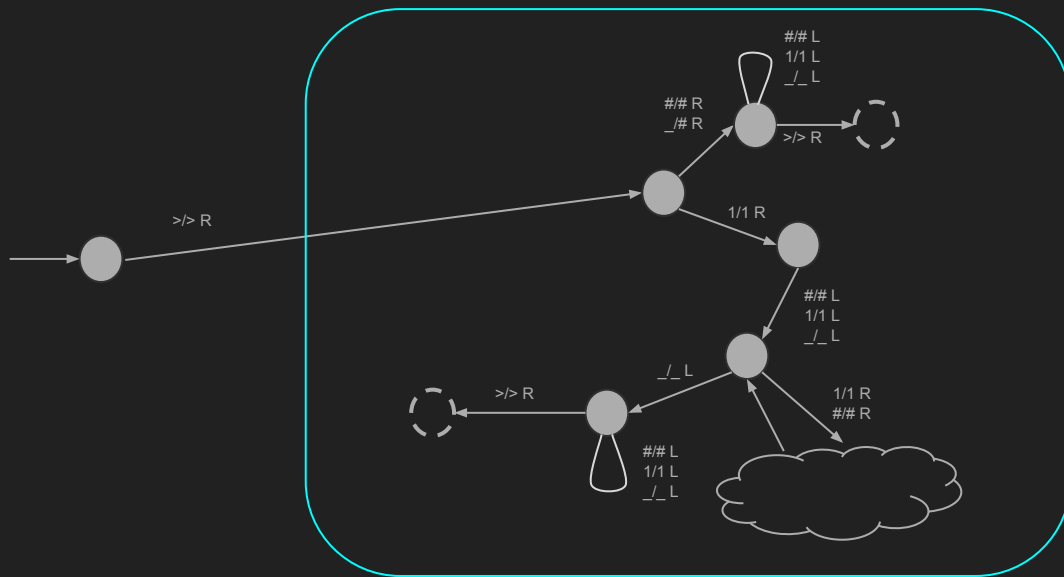
DEC 0, stop

INC 1

JMP start

stop:

TRUE



# Example

start:

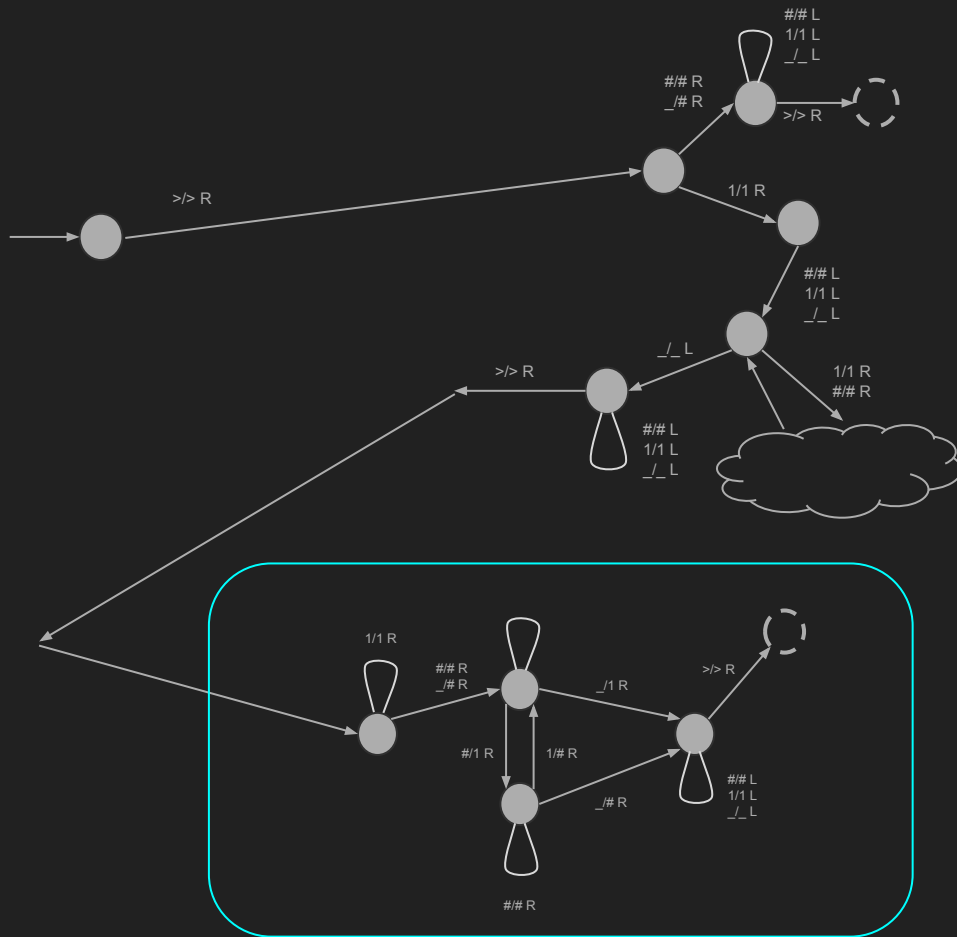
DEC 0, stop

INC 1

JMP start

stop:

TRUE



# Example

start:

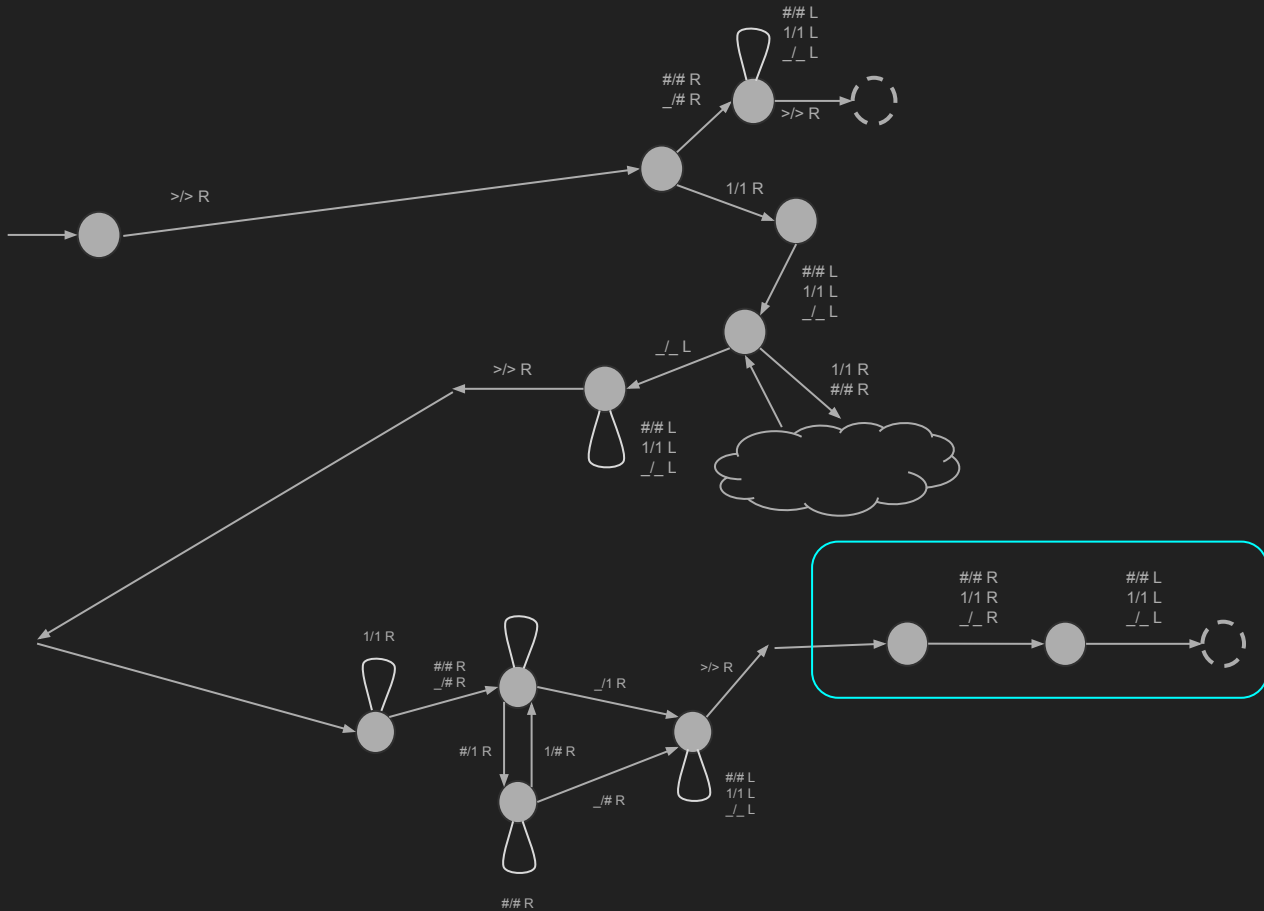
DEC 0, stop

# INC 1

# JMP start

stop:

TRUE



# Example

start:

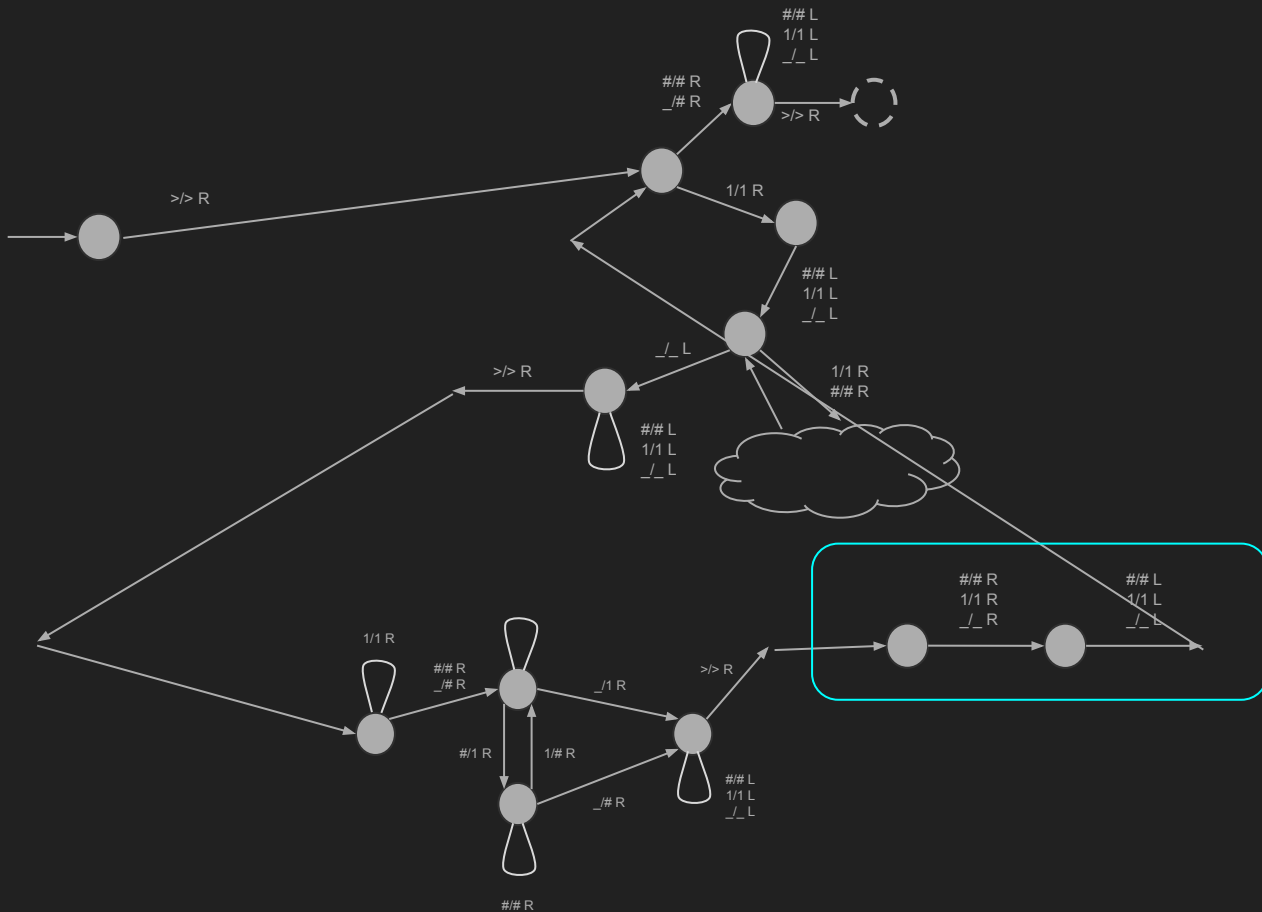
DEC 0, stop

INC 1

**JMP start**

stop:

TRUE



# Example

start:

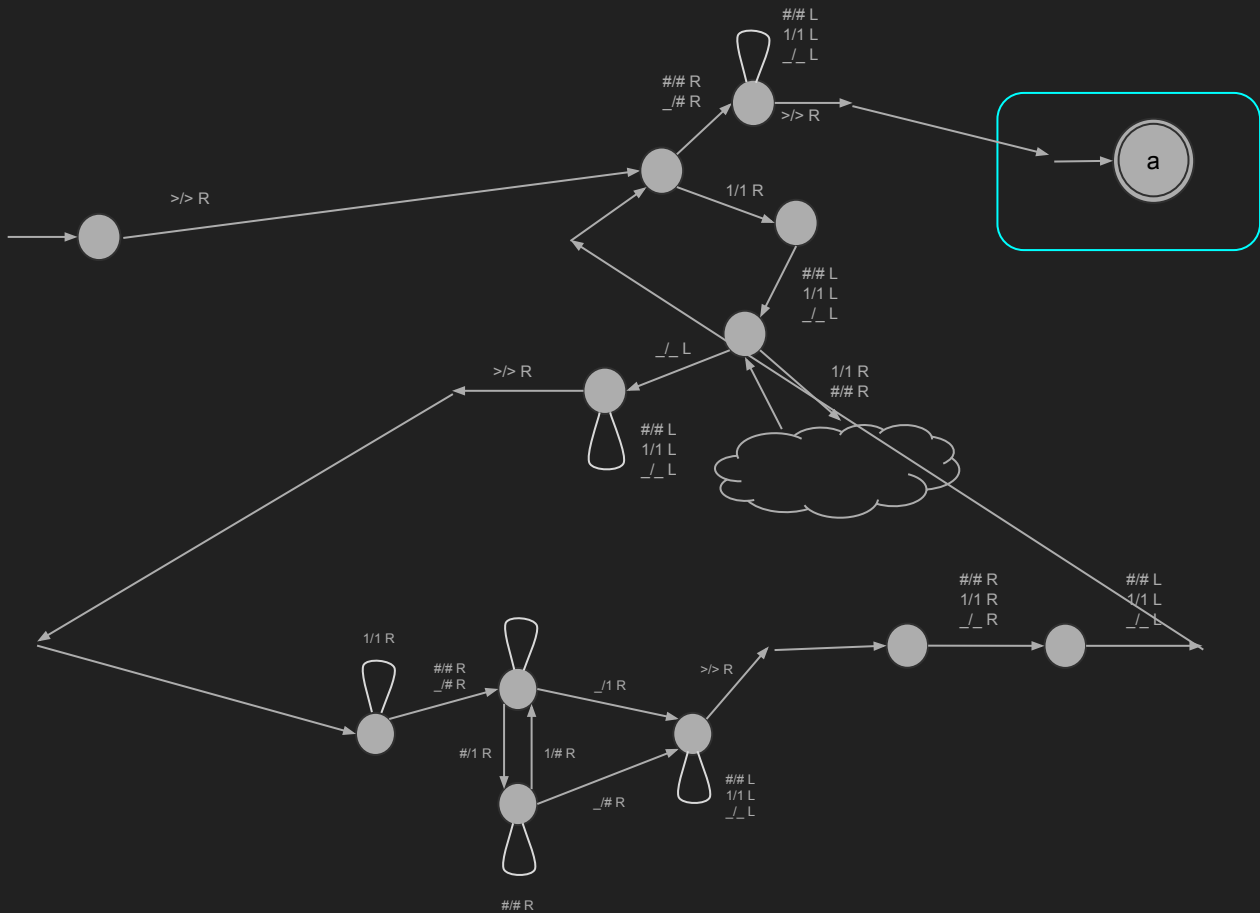
DEC 0, stop

# INC 1

# JMP start

stop:

TRUE



# Higher-level languages

Once you have a small CPU language, you can use it as the target of **compilation** for higher-level languages

At this point though, this is less about Turing machines, and more about programming languages implementation, with Turing machines as a (very slow) execution model