Simulating a register machine

FOCS

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

A simple register machine

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

Each register holds an arbitrary natural number (≥ 0)

A program is a sequence of instructions (indexed from 0)

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

Example: addition

```
# R0 + R1 =? R2

# start
0      DEC 0, 3   #equal?
1      INC 1
2      JMP 0

# equal?
3      DEC 1, 6   #empty0
4      DEC 2, 7   #reject
5      JMP 3      #equal?
# empty0
6      DEC 2, 8   #accept
# reject
7      FALSE
# accept
8      TRUE
```

Example: multiplication

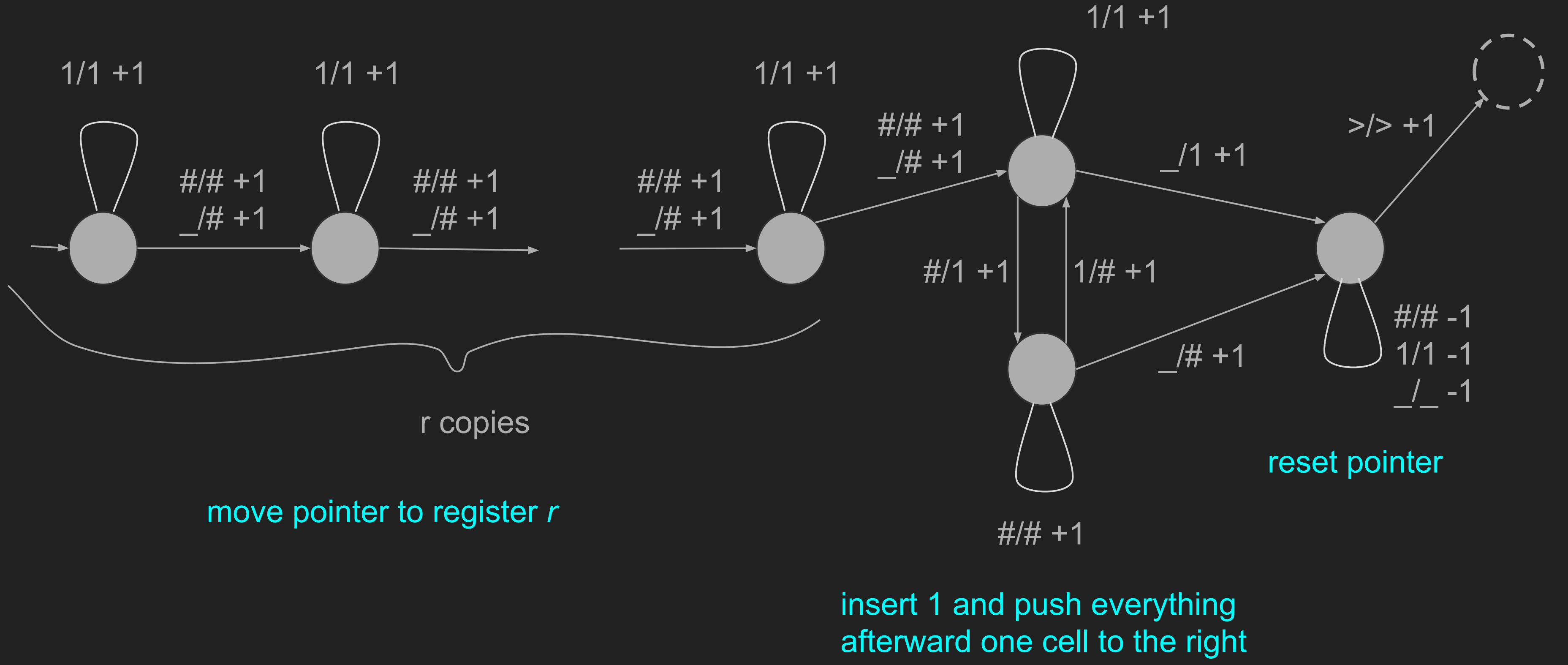
```
# R0 * R1 =? R2

# clear R3 = prod
0    DEC 3, 2   #clearR4
1    JMP 0

# clear R4 = temp
2    DEC 4, 4   #outloop
3    JMP 2

# outloop
4    DEC 0, 12  #equal?
# inloop
5    DEC 1, 9   #next
6    INC 3
7    INC 4
8    JMP 5      #inloop
# next
9    DEC 4, 4   #outloop
10   INC 1
11   JMP 9      #next

# equal?
12   DEC 3, 15  #empty
13   DEC 2, 16  #reject
14   JMP 12     #equal?
# empty
15   DEC 2, 17  #accept
# reject
16   FALSE
# accept
17   TRUE
```

Simulating a program with a Turing machine

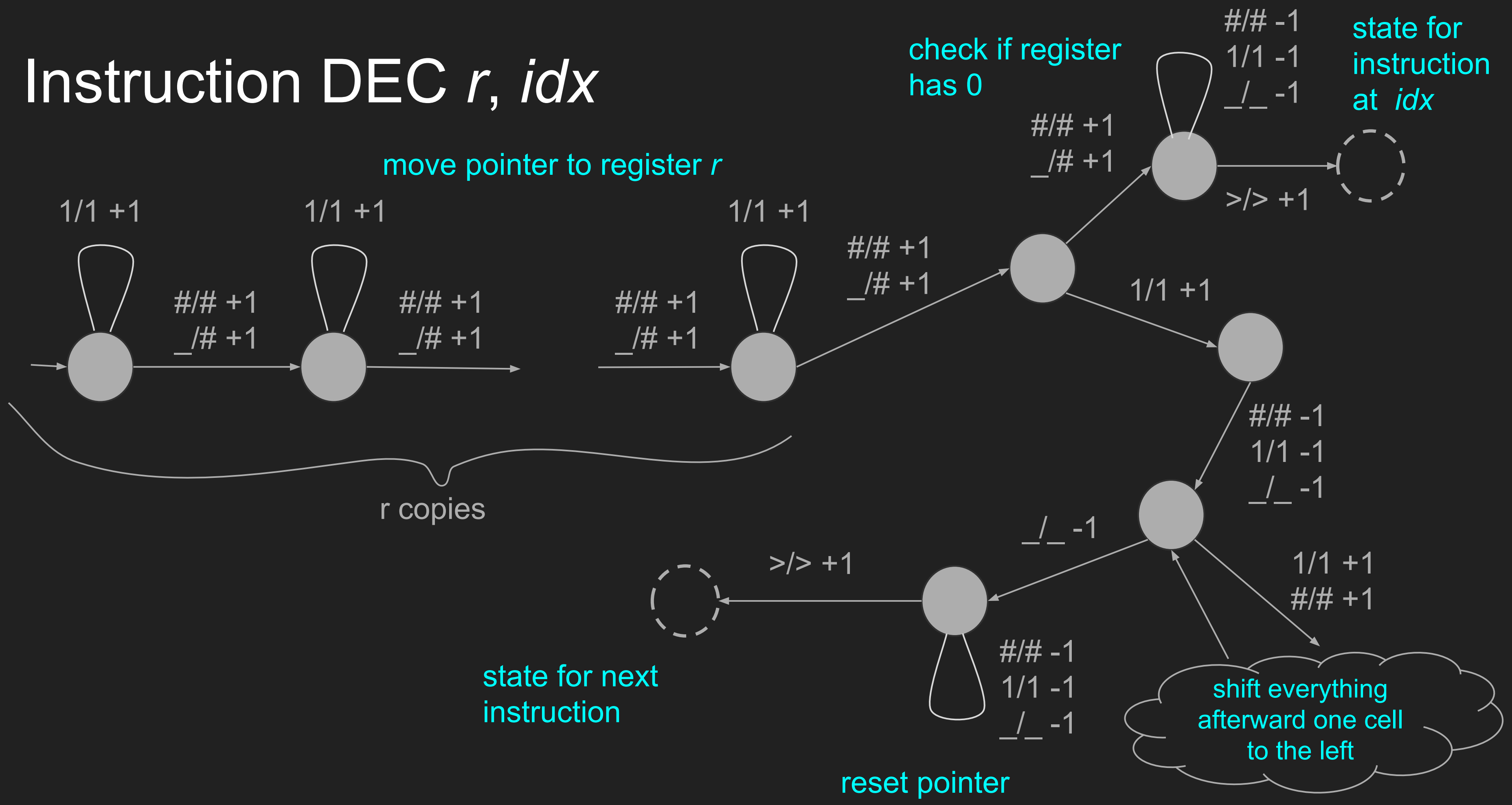
Translate a register machine program into a Turing machine

- 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
    - $>n_0\#n_1\#n_2\#n_3\#n_4\#\ldots$
    - 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*



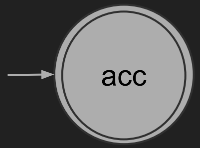
Instruction DEC *r*, *idx*



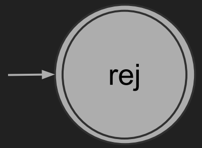
Instruction JMP *idx*

Instructions TRUE and FALSE

TRUE

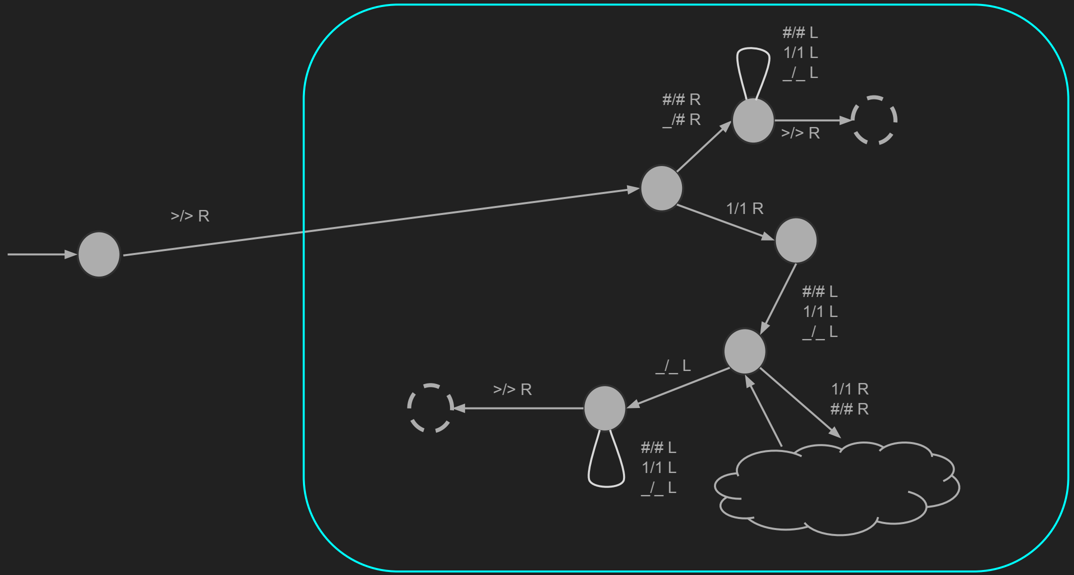


FALSE



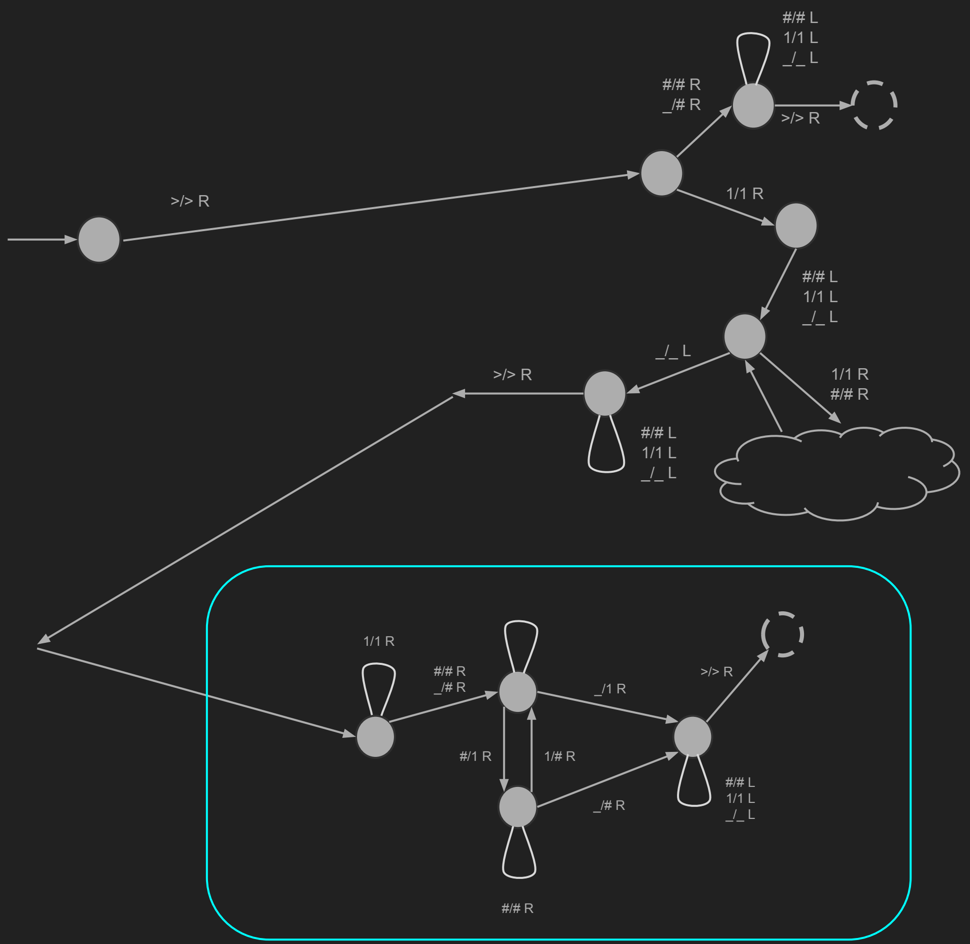
Example

\# start:
0 DEC 0, 3
1 INC 1
2 JMP 0
\# stop:
3 TRUE



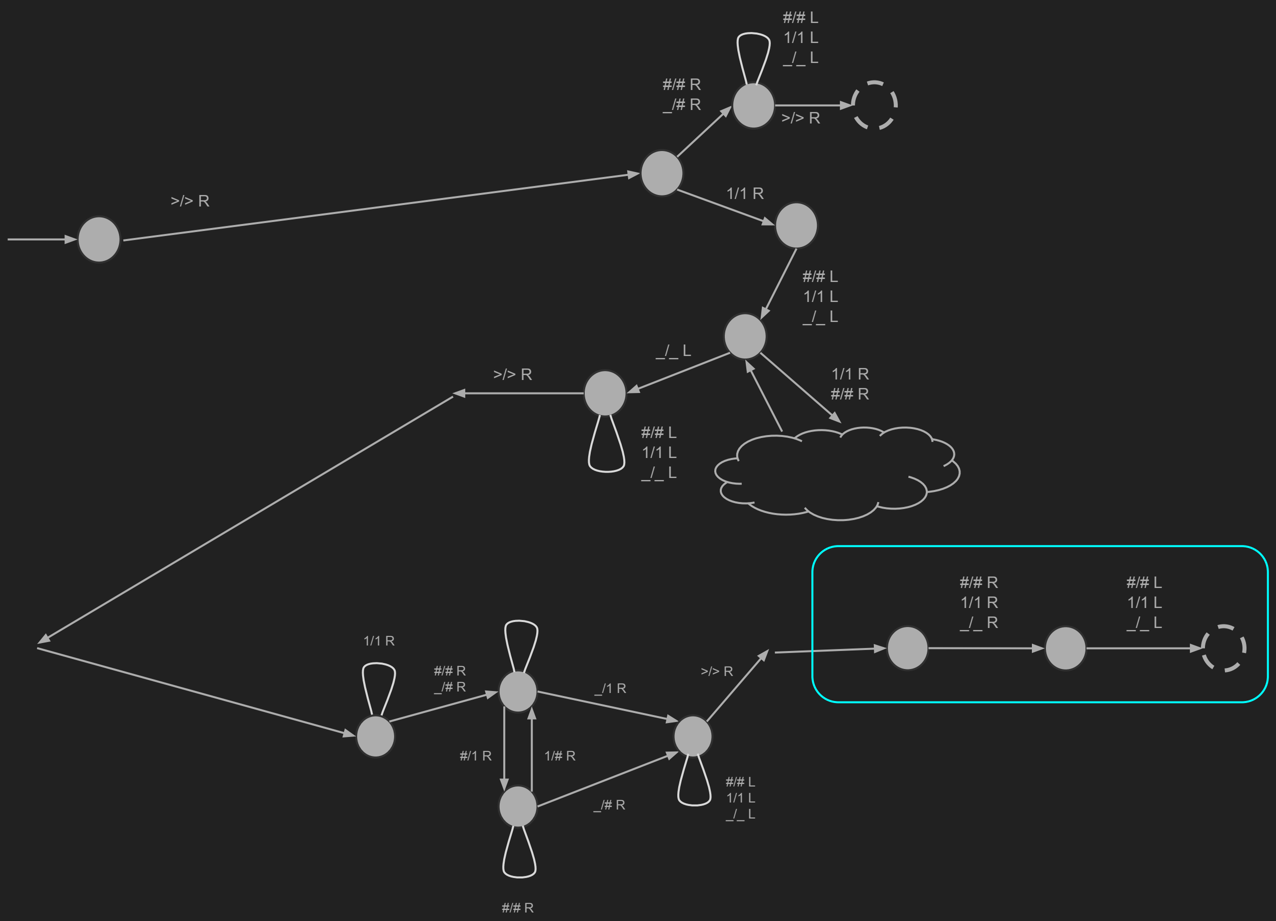
Example

\# start:
0 DEC 0, 3
1 INC 1
2 JMP 0
\# stop:
3 TRUE



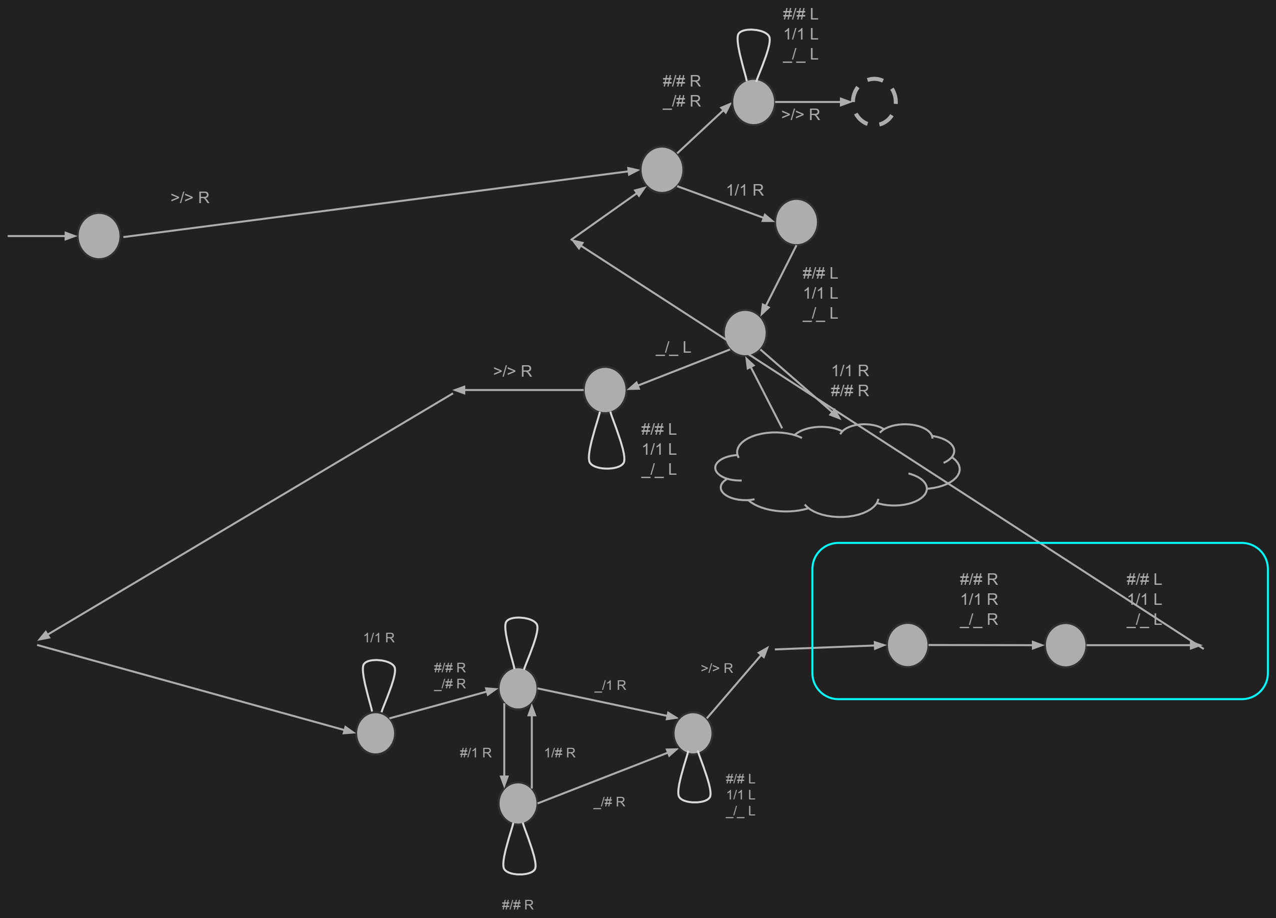
Example

start:
0 DEC 0, 3
1 INC 1
2 JMP 0
stop:
3 TRUE



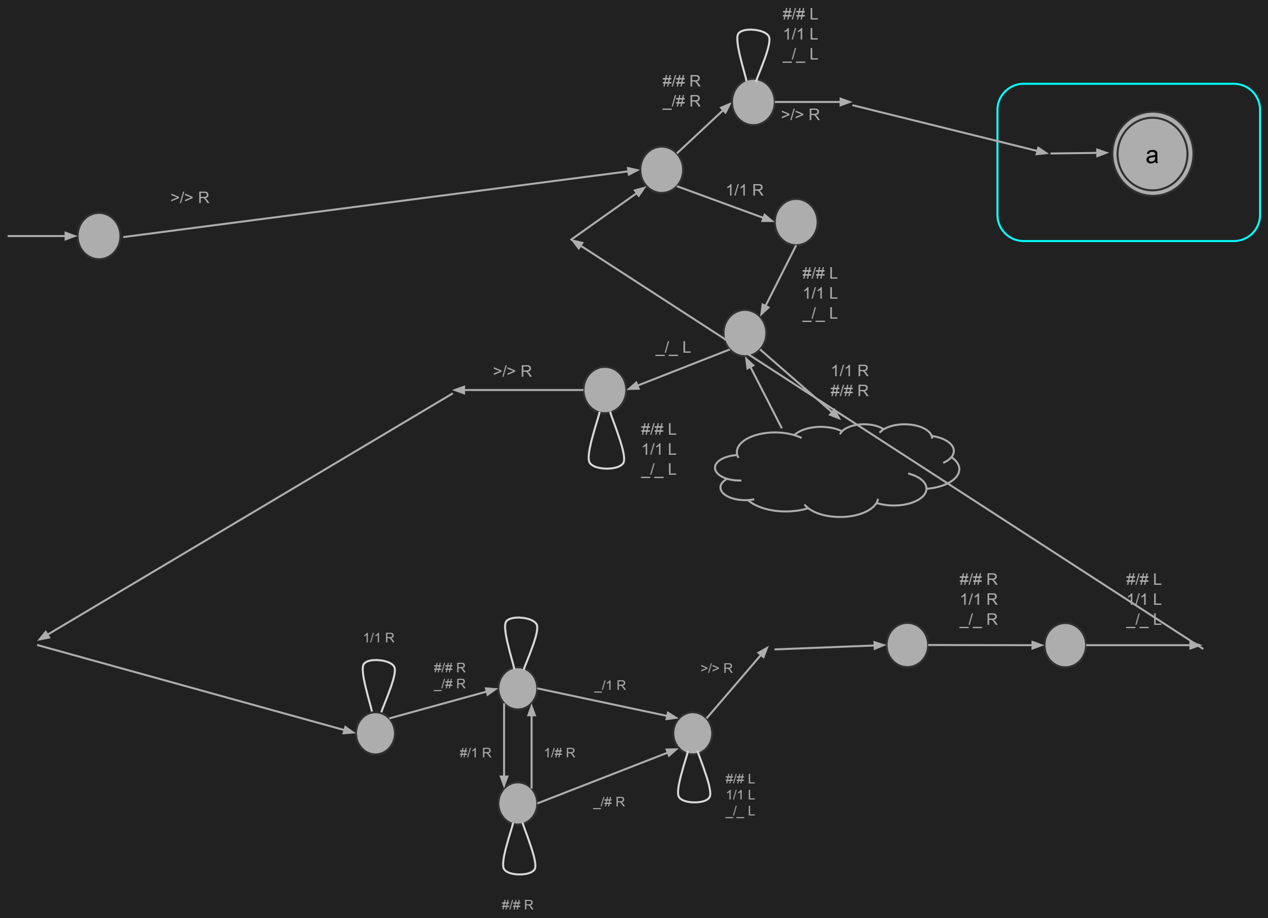
Example

\# start:
0 DEC 0, 3
1 INC 1
2 JMP 0
\# stop:
3 TRUE



Example

start:
0 DEC 0, 3
1 INC 1
2 JMP 0
stop:
3 TRUE



Higher-level languages

Once you have a register machine 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