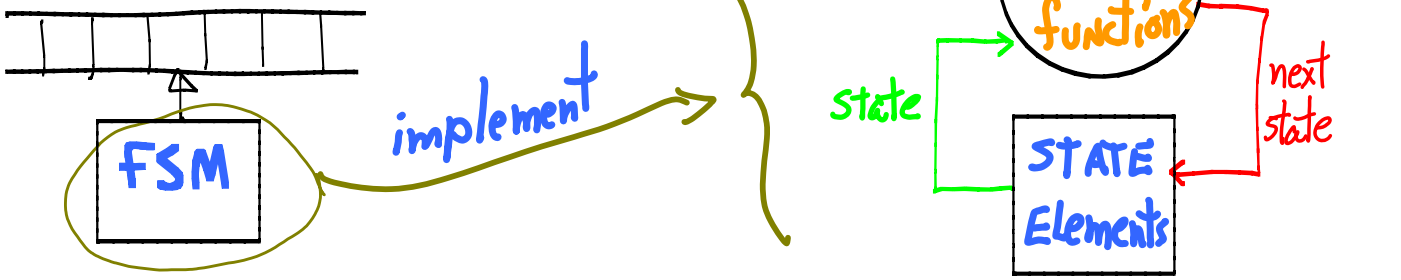
TM Implementation



To **build any TM, WE NEED**:

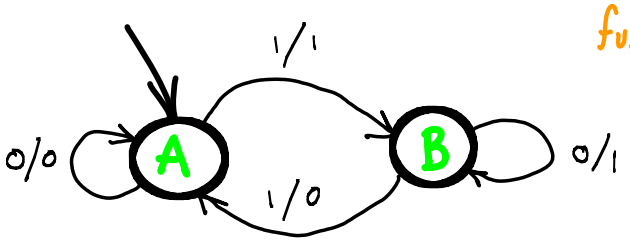
--- **(1.) FSM**:
- **state**
- **logic functions** (**output** and **next-state**)

--- **(2.) Tape**: **methods to R/W symbols**, we'll use registers (RAM).

--- **(3.) Symbol set** = a set of fixed length bit strings, e.g.,
- **S = {0,1}** (**2 symbols**)
- **S = {00, 01, 10, 11}** (**4 symbols**)
- **S = {000, 001, 010, 011, 100, 101, 110, 111}** (**8 symbols**)

Build FSM

even-odd parity



{symbols} = {0, 1}

{states} = {A, B} encode → {0, 1}

functions

OUT : {states} X {symbols} → {symbols}

next_state : {states} X {symbols} → {states}

OUT

(A, 0)	===>	0
(A, 1)	===>	1
(B, 0)	===>	1
(B, 1)	===>	0

encode →

OUT

(0, 0)	===>	0
(0, 1)	===>	1
(1, 0)	===>	1
(1, 1)	===>	0

next_state

(A, 0)	===>	A
(A, 1)	===>	B
(B, 0)	===>	B
(B, 1)	===>	A

encode →

next_state

(0, 0)	===>	0
(0, 1)	===>	1
(1, 0)	===>	1
(1, 1)	===>	0

We Need:
- functions
- STATE ELEMENTS

--- **in hardware** for **simulator/computer/UTM**.
--- **in description language** for **any TM**.

Boolean functions

Universal (able to **simulate any TM**)

--- **language** to **describe any TM**,
--- **Simulator** that **understands that language**.

LANGUAGE

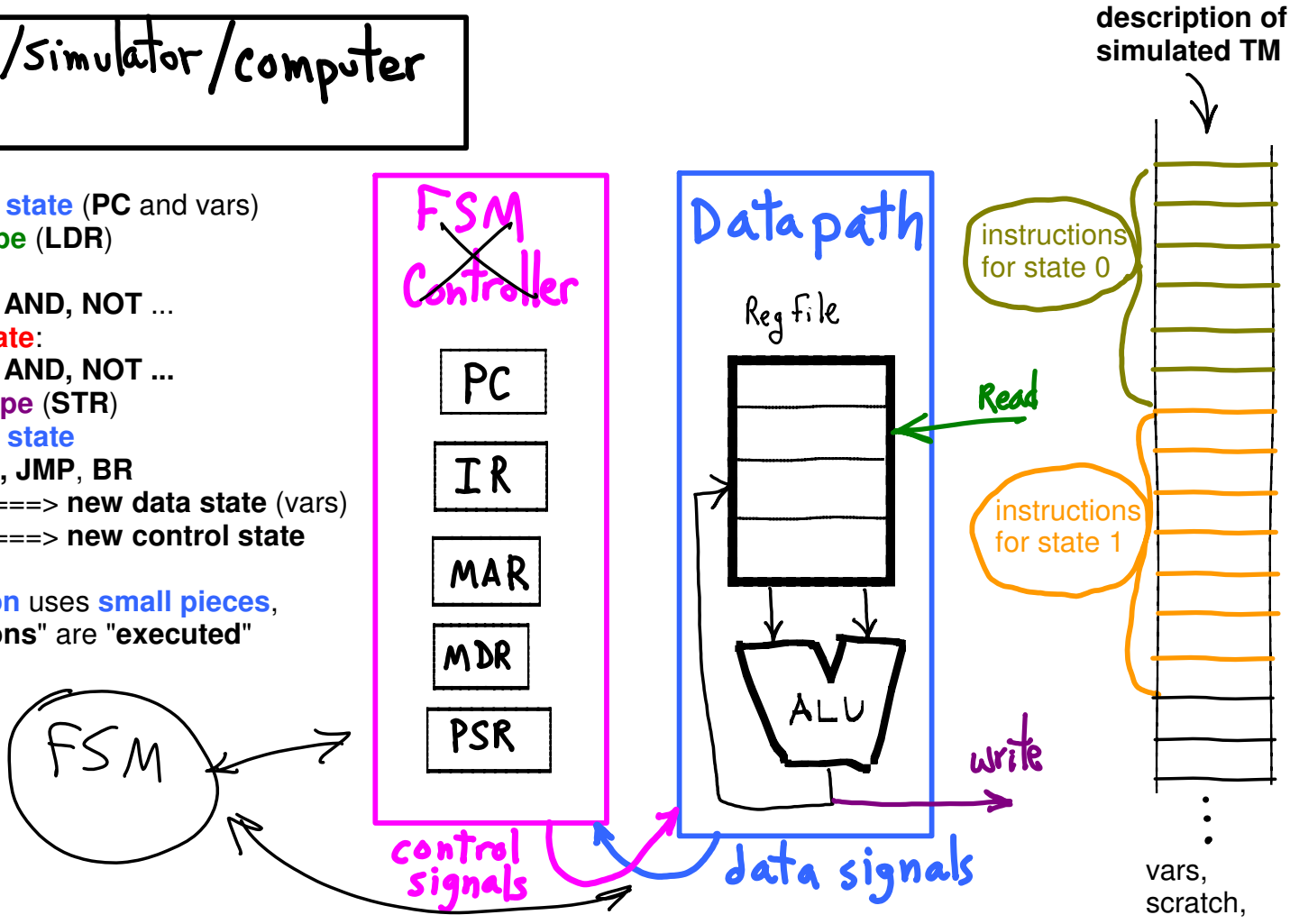
able to describe:

--- Arbitrary **set of states**, including regs (= vars)
--- Arbitrary **set of symbols**
--- Arbitrary **branching** **(via binary trees)**
--- **RW to tape**
--- Arbitrary **functions** (**next-state**, **output**)

UTM/simulator/computer

--- **current state** (**PC** and vars)
--- **read tape** (**LDR**)
--- **OUT**:
    ADD, AND, NOT ...
--- **next-state**:
    ADD, AND, NOT ...
--- **write tape** (**STR**)
--- **change state**
    PC++, JMP, BR
    STR ===> new data state (vars)
    STR ===> new control state

Description uses **small pieces**,
"**instructions**" are "**executed**"



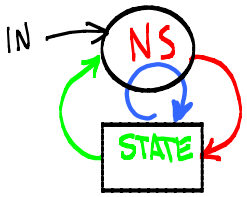
FSM Controller uses **registers** (e.g., PC) to **remember**:
--- **simulated machine's state** (control + data states)
--- simulated **symbols read** (RegFile)
--- simulated **write symbols** (RegFile)
--- **step of simulation** **(UTM's controller's state)**
--- **partial steps** of **function evaluations** (next-state, output)
        data registers, PSR, on tape, ...

STATE ELEMENTS

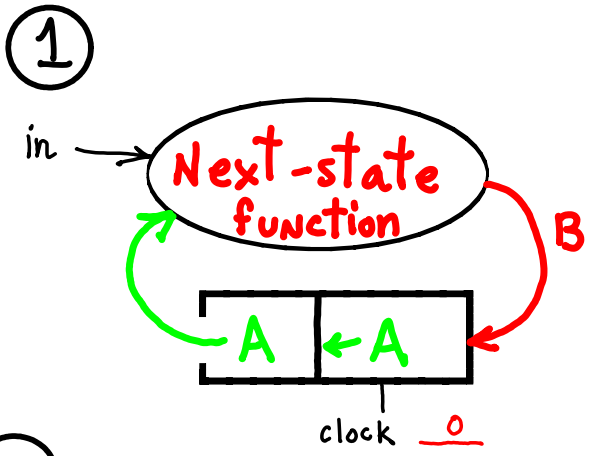
pos. edge-triggered FF

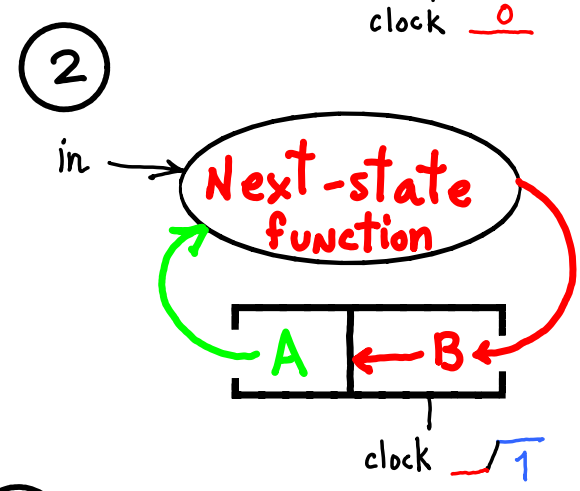
Problem:

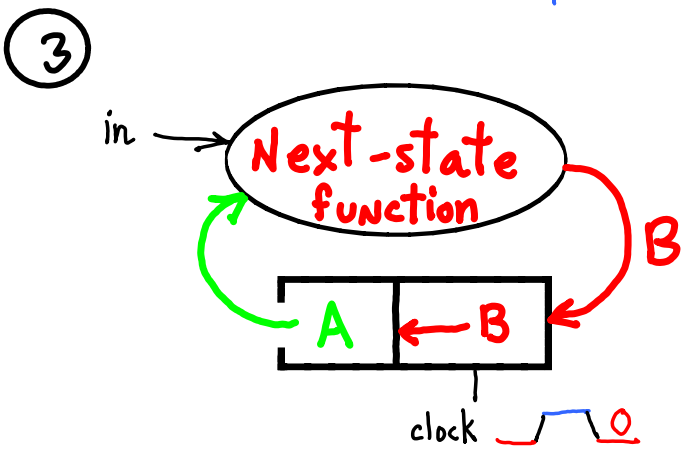
Katy bar the door



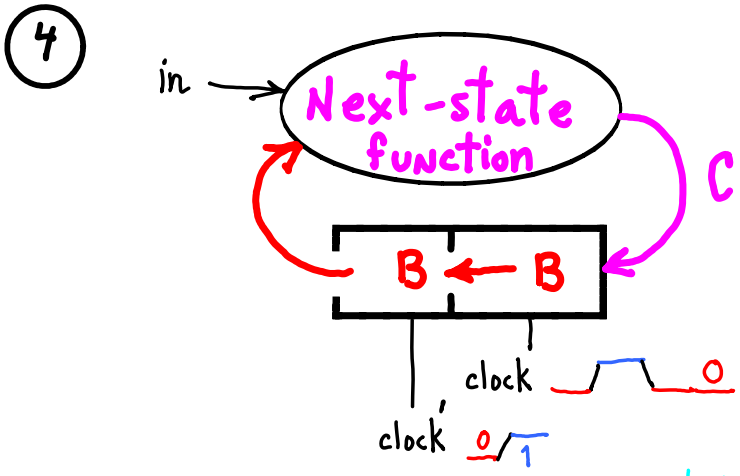
Feedback loop:
state change,
NS change,
state change,
NS change, ...

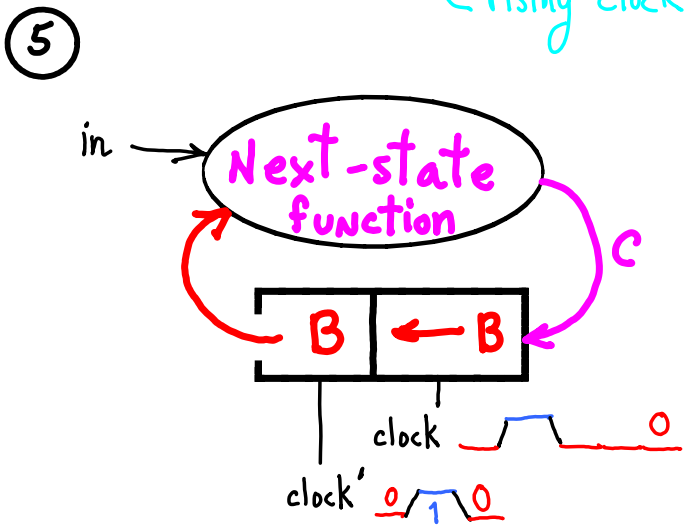






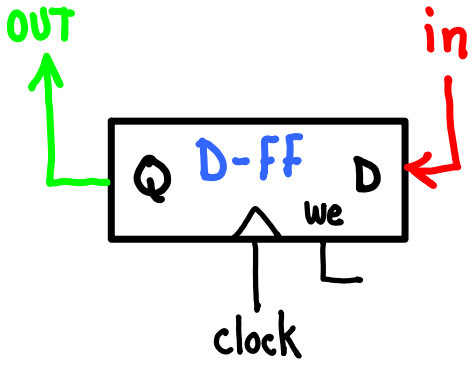
input "sampled"
"latched"





Back at (1)

This is what we need for implementing STATE.



Describing Functions

Two Ways

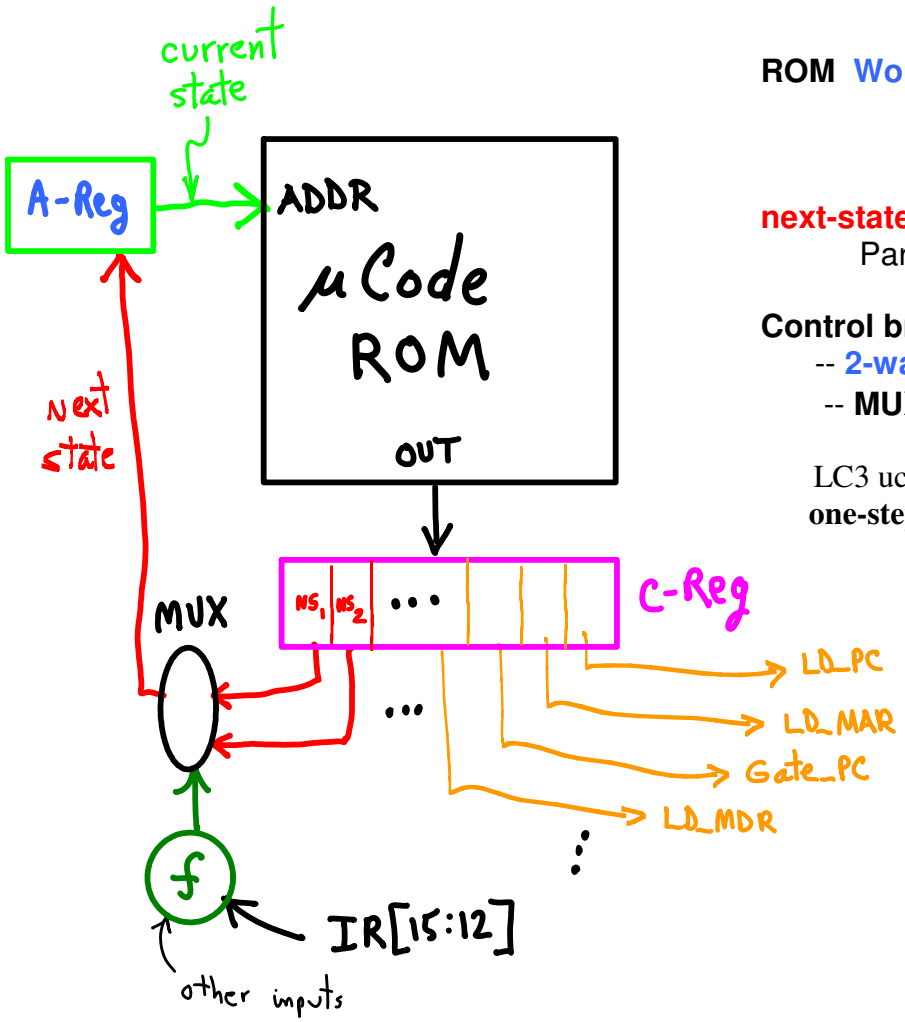
(A) **for any** x
describe **how to evaluate** $f(x)$ E.g. $f(x) = 2x$

(B) **provide a table** giving $f(x)$ **for any** x

E.g.

(x)	$f(x)$
0	1
1	0

(B) microcoded controller next-state & output functions



A-Reg == **controller's current state**
addresses **uCode ROM**
gives memory **word at output**

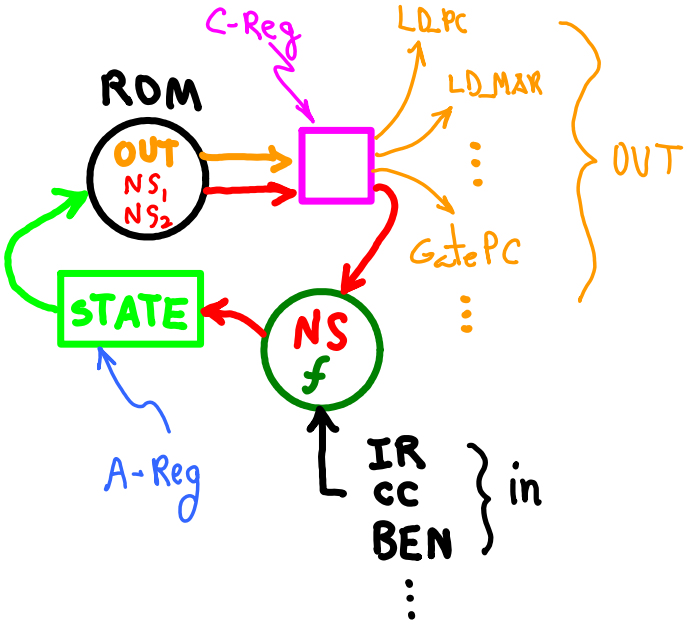
ROM Word == **datapath control bits**
C-Reg has **current control word**
controls datapath

next-state fields of C-Reg
Part of **Controller's next-state** function

Control branching (the rest of **next-state** function)
-- **2-way**, **NS1** or **NS2**
-- **MUX.select** = *f* (**STATE**, **IR**, **...)**

LC3 ucode branching also includes
one-step, 16-way branching (DECODE).

As a general FSM,
it looks like this,



Advantages of **ucode controller:**

-- easier to **change**
-- easier to **figure out**
-- easier to **expand**
install bigger ROM.

Advantages of "**random logic**" **controller**:

-- **faster**
-- **smaller** (?)
-- **distributed throughout machine**

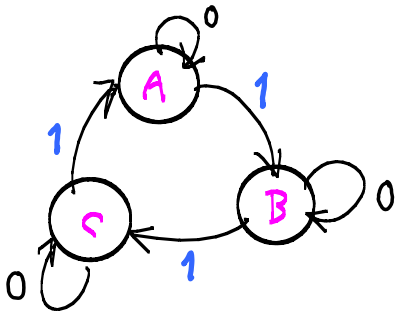
Caveat: The C-Reg is just to make the picture clearer, it doesn't actually exist in LC3.

WE HAVE (suppose for now)

--- 1-bit state elements
--- 1-bit function elements

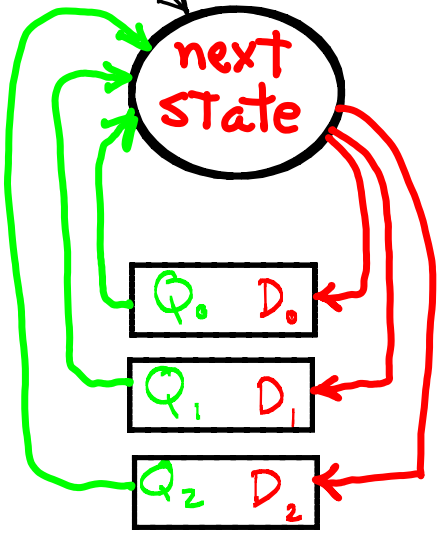
HOW to put them together?

A mod-3 machine



STATE ENCODING

1-hot encoding

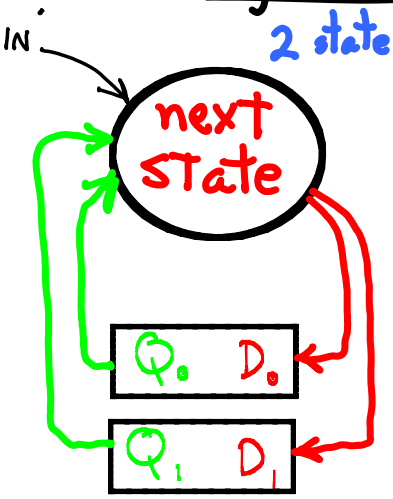


STATE	Q_2	Q_1	Q_0
A	0	0	1
B	0	1	0
C	1	0	0

3 state elements

binary encoding

2 state elements



STATE	Q_1	Q_0
A	0	0
B	0	1
C	1	0

Describe next-state function

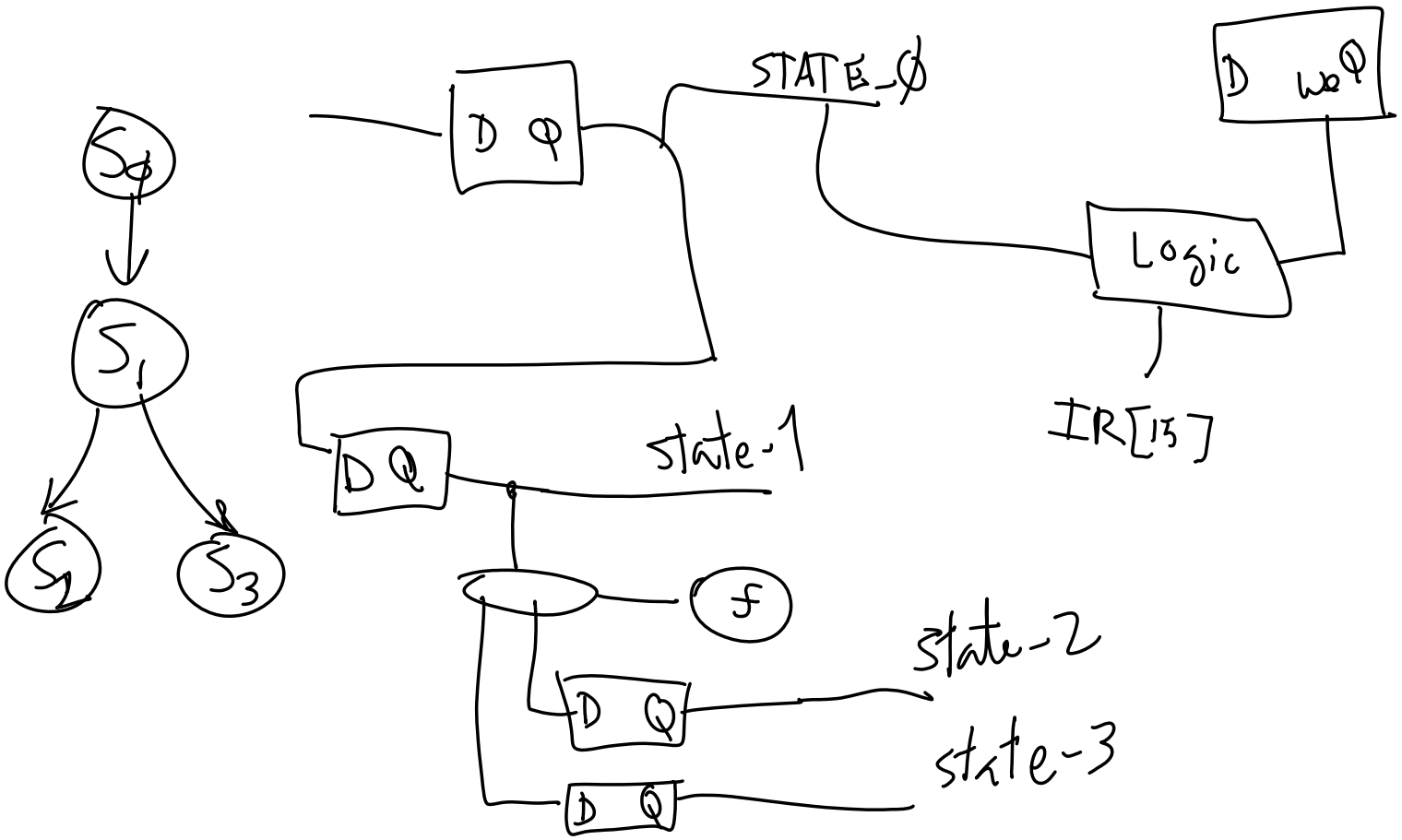
IN	Q2	Q1	Q0		D2	D1	D0	
0	0	0	1	(A)	0	0	1	(A)
1	0	0	1	(A)	0	1	0	(B)
0	0	1	0	(B)	0	1	0	(B)
1	0	1	0	(B)	1	0	0	(C)
0	1	0	0	(C)	1	0	0	(C)
1	1	0	0	(C)	0	0	1	(A)
*	*	*	*		x	x	x	

* rows not shown, don't matter?
X is for **don't care**, either 0 or 1.

x $f(x)$

IN	Q1	Q0		D1	D0	
0	0	0	(A)	0	0	(A)
1	0	0	(A)	0	1	(B)
0	0	1	(B)	0	1	(B)
1	0	1	(B)	1	1	(C)
0	1	1	(C)	1	1	(C)
0	1	1	(C)	0	0	(A)
*	*	*		x	x	

* rows cannot be reached.


S_0
S_1
S_2
S_3
D Q
STATE_0
D Q
Logic
IR[15]
D Q
state-1
f
state-2
D Q
state-3
D Q

IF we have a **universal language** (able to describe **any TM**)

All we need to know is **How To Build**:

--- **1-bit state elements**?

--- **1-bit functions**?