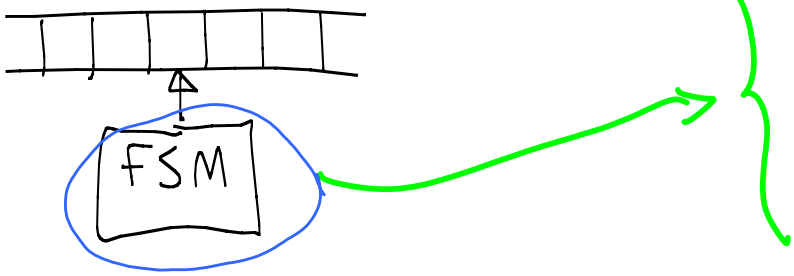
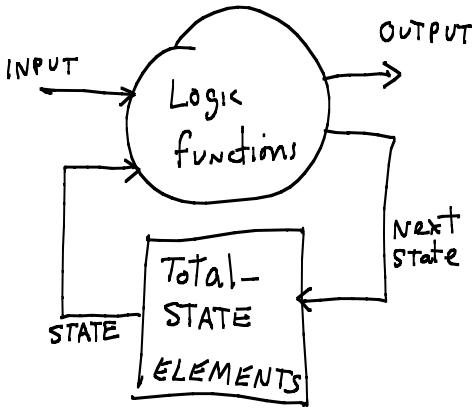
TM-implementation





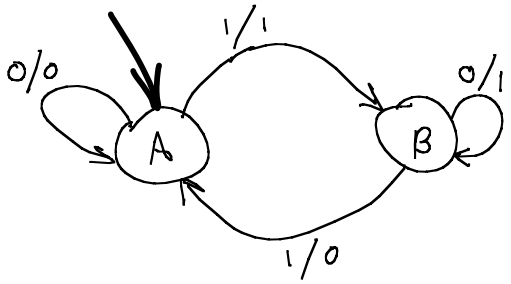
We take liberties with the terms "TM" and "UTM", but in basic concepts we're ok.
To build any TM, WE NEED:

--- FSM:
  --- state
  --- logic functions (output and next-state)

--- Tape: a variety of methods to RW symbols, we'll use registers (RAM).

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

Build FSM | even-odd parity



functions

$$\begin{cases} OUT : S \times \Sigma \rightarrow \Sigma \\ Next\_State : S \times \Sigma \rightarrow S \end{cases}$$

Σ = Symbols = {0,1}

S = States = {A,B} $\xrightarrow{encode}$ {0,1}

{A,B,C,D} $\xrightarrow{encode}$ {00,01,10,11}

OUT

(A,0) → 0
(A,1) → 1
(B,0) → 1
(B,1) → 0

Next_State

(A,0) → A
(A,1) → B
(B,0) → B
(B,1) → A

We Need:

- functions
- STATE ELEMENTS

Boolean functions

--- in hardware for our simulator/computer/UTM.
--- in desc. language so we can describe any TM.

For our "UTM/simulator" to be universal (able to simulate any TM)
WE MUST HAVE:
--- a description language able to describe any TM,
--- a UTM/Simulator that can understand that language.

LANGUAGE

Language (ISA) needs to be 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)

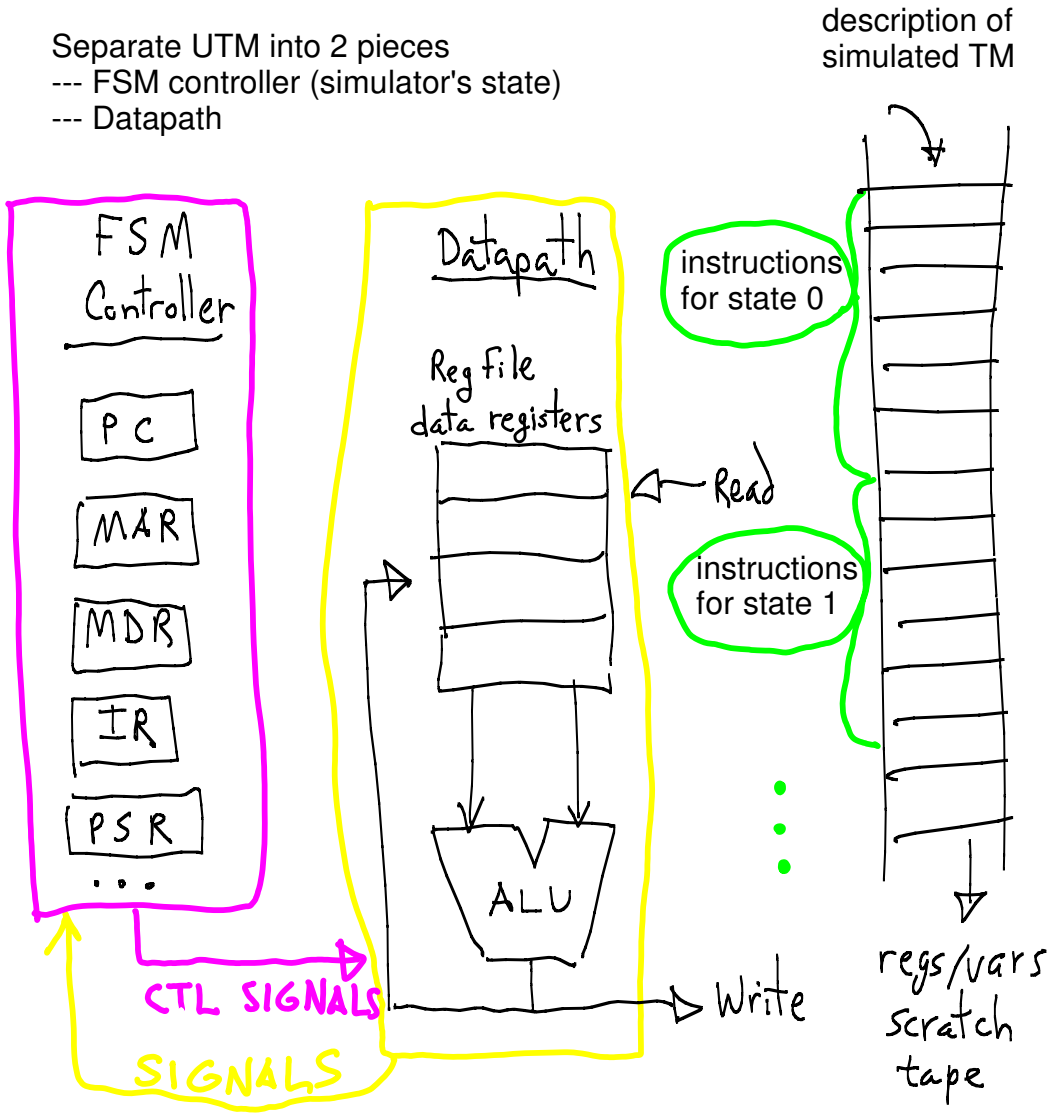
UTM/Simulator simulates using

--- current state (PC and var memory)
--- read tape (LDR => reg)
--- output function:
    ADD, AND, NOT ...
--- next-state function:
    ADD, AND, NOT ...
--- write tape (STR)
--- change state
    PC ++, JMP, BRp, BRz, BRn
    STR => data state (var memory)

Represent description using small pieces, "instructions" that are "executed" by machine cycle:

--- Instruction fetch
--- Operand fetch
--- Execute
    --- Function evaluation
    --- Change state (branching)
--- Operand store

Separate UTM into 2 pieces
--- FSM controller (simulator's state)
--- Datapath

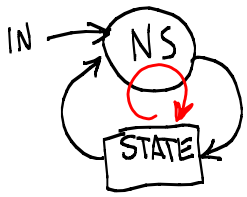


FSM Controller uses registers (e.g., PC) to remember:
--- simulated machine's state (control state + data reg) (simulated data reg is var on tape.)
--- simulated symbols read (RegFile)
--- simulated machine's write symbols (RegFile)
--- its own machine state (controller state), ie. step of simulation
--- partial steps of function evaluations (next-state, output)
    these are in data registers, mostly, but also PSR, on tape, ...

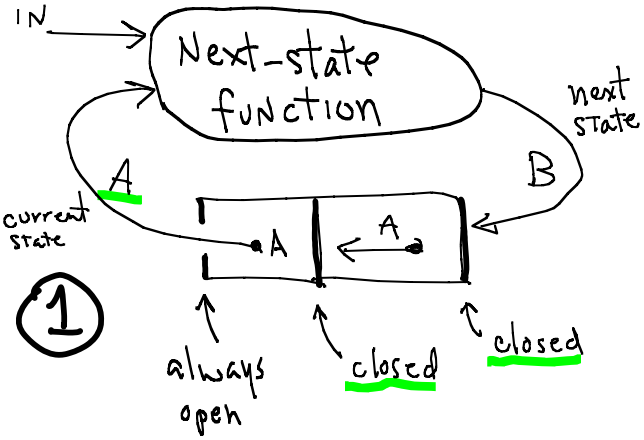
STATE ELEMENTS

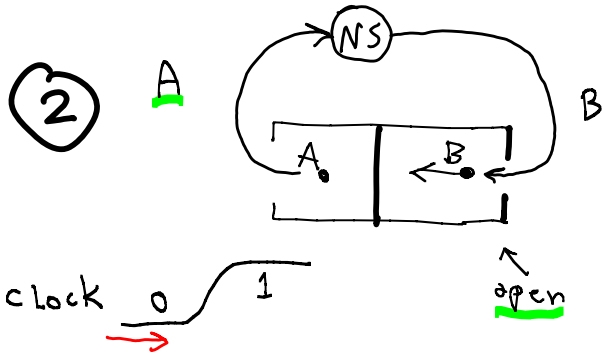
pos. edge-triggered FF

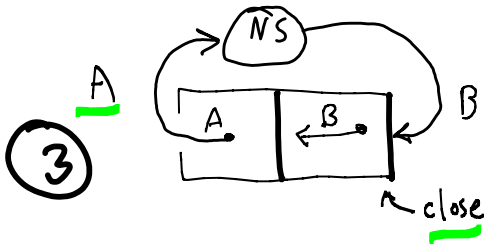
Problem:

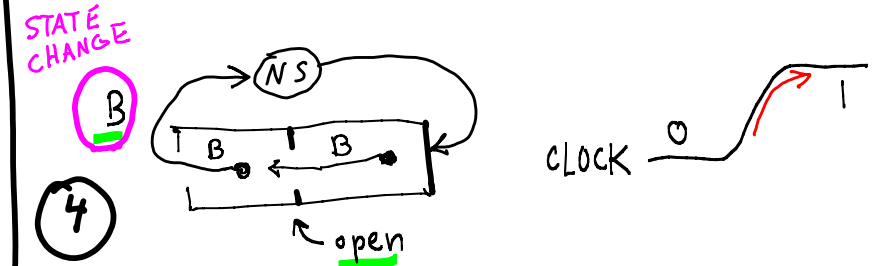


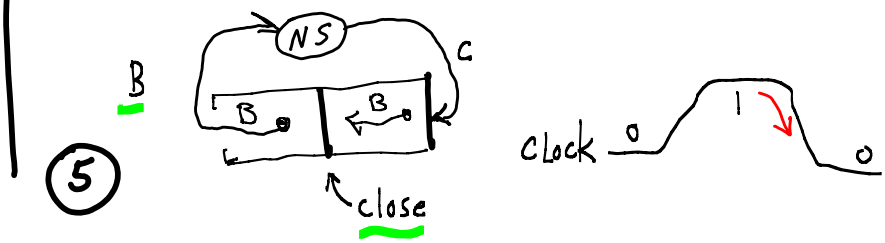
Feedback loop:
state change, NS change, state change, NS change, ...



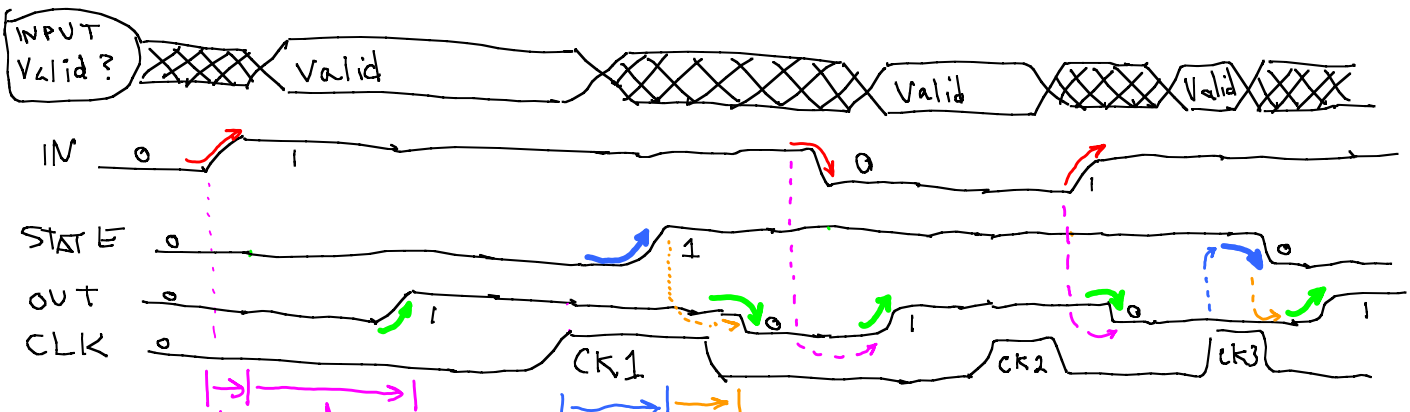




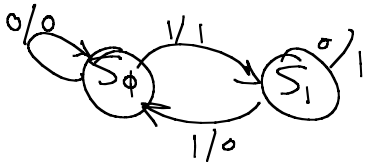


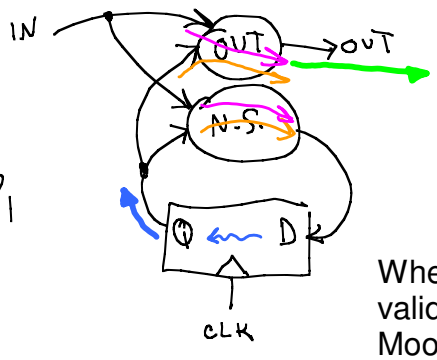


Valid / not Valid Timing, things take Time



delay until OUT responds To IN 0 → 1

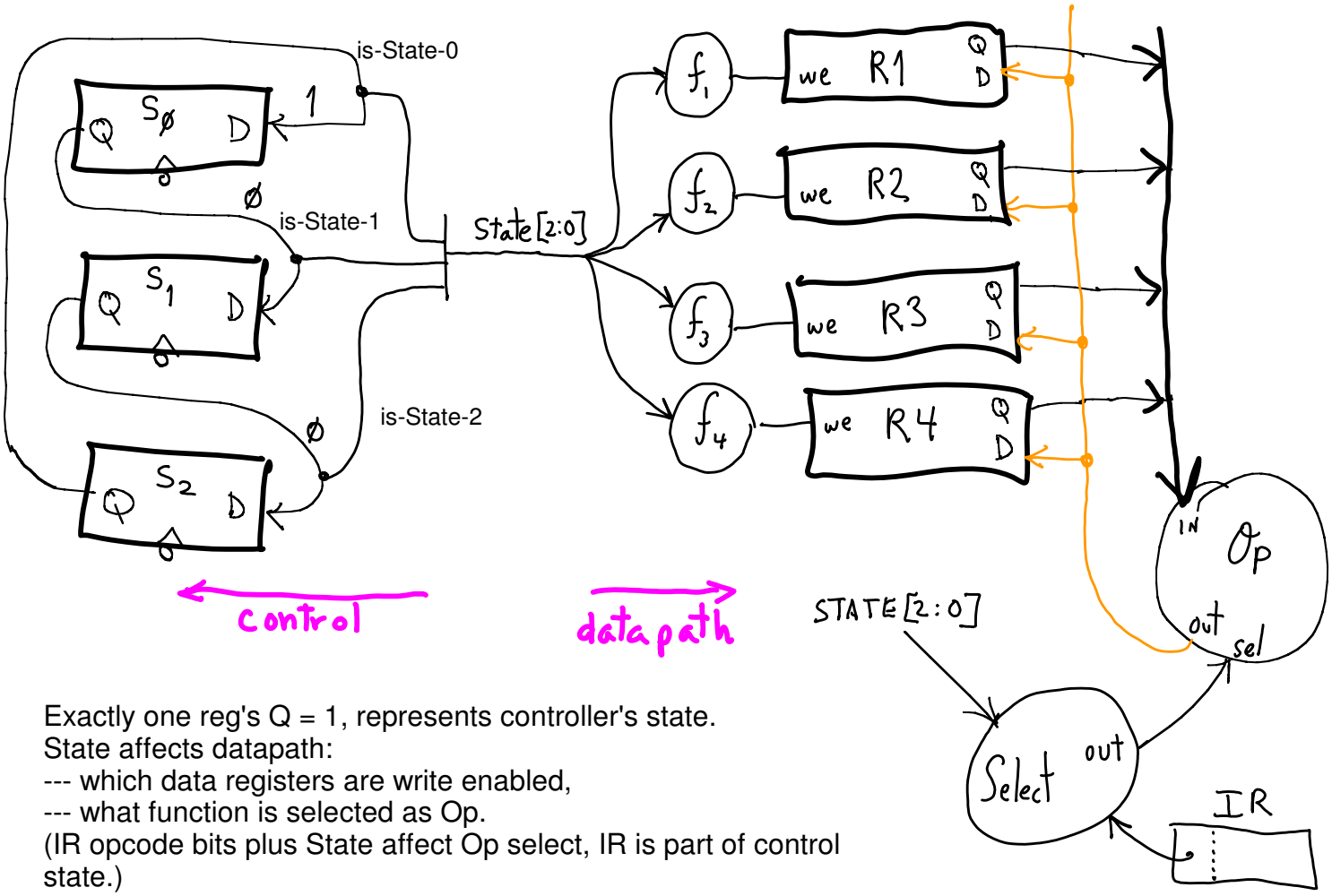




When is OUT valid? Do better w/ Moore machine?

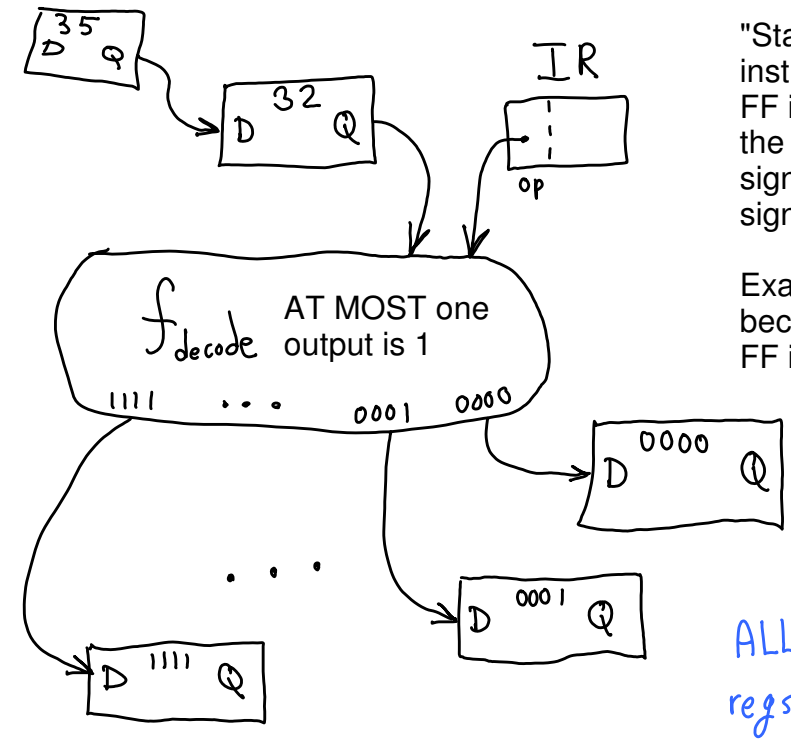
One-hot FSM controller

Data registers and functions
(write enable, select Operation/function)



Exactly one reg's Q = 1, represents controller's state.
State affects datapath:
--- which data registers are write enabled,
--- what function is selected as Op.
(IR opcode bits plus State affect Op select, IR is part of control state.)

Control Branching in 1-hot ?

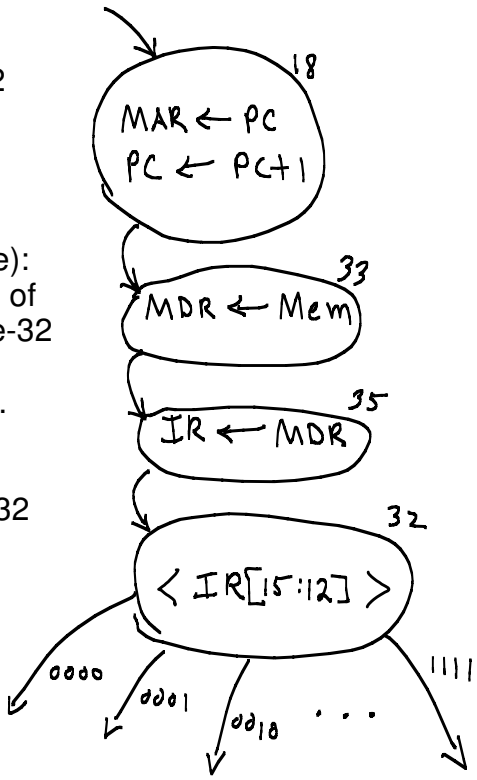


LC3 Decode state-32

Control branching

"State-0001" (ADD instruction's first state): FF input is a function of the value of the State-32 signal and the 4-bit signal from IR[15:12].

Exactly ONE FF will become hot, if state-32 FF is hot.



ALL control state regs are write enabled

Can we describe functions such as Select? How? Two possibilities:

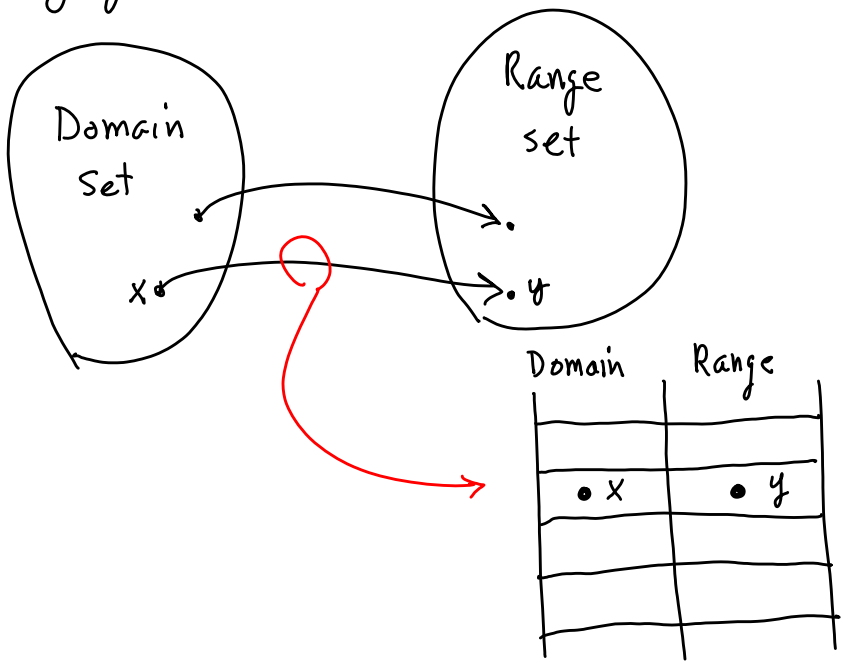
(1) describe for any x how to calculate f(x).
(2) show for all x, the value of f(x).

We choose (2) ==> a table.

INPUTS:					OUTPUTS:					
state-32	IR[15]	IR[14]	IR[13]	IR[12]	1111	1110	1101	...	0001	0000
1	0	0	0	0	0	0	0	...	0	1
1	0	0	0	1	0	0	0	...	1	0
...	...	...	...	...	...	...	...	...	...	...
1	1	1	1	0	0	1	0	...	0	0
1	1	1	1	1	1	0	0	...	0	0

(ignore all rows for state-32 == 0; they are not relevant: all outputs are 0.)

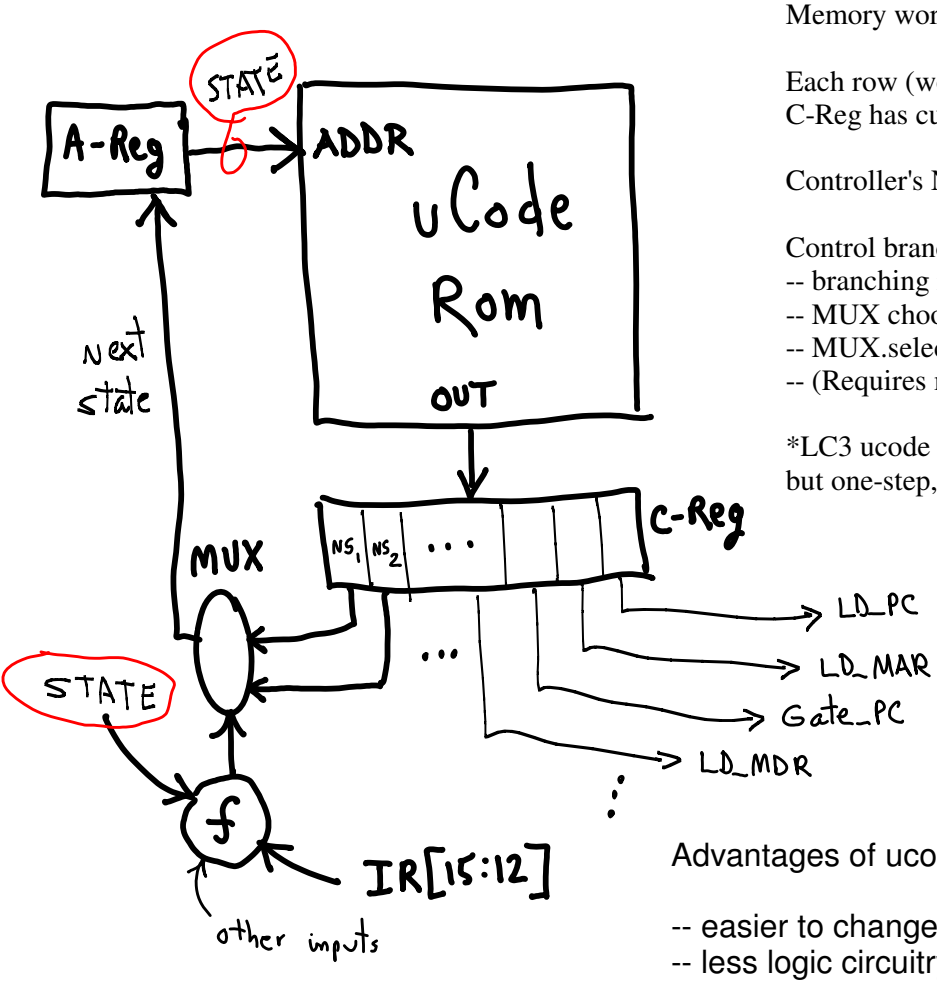
Any function can be written as a table*



what about relations that are not functions?
(x, y)
(x, z)
partial functions?

* what about $\mathbb{R}$?
* what about $2^{\mathbb{R}}$ (subsets of $\mathbb{R}$)?

OR, microcoded controller



A-Reg has controller's current state.
STATE is used as address to ROM.
Memory word selected by A-Reg's content (address)

Each row (word) of ROM has datapath control bits.
C-Reg has current "control word", its outputs control datapath.

Controller's Next-state bits are in next-state fields of C-Reg.

Control branching, One possibility:
-- branching is two-way, NS1 or NS2
-- MUX chooses one
-- MUX.select = f(STATE, IR, other inputs)
-- (Requires multi-stage branching for 16-way decode for LC3 ISA)

*LC3 ucode controller has different branch scheme. Harder to grasp, but one-step, 16-way branching.

Control signals to Datapath

Advantages of ucode controller:

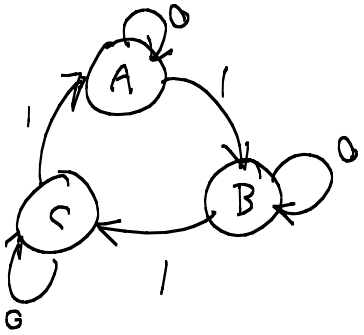
-- easier to change
-- less logic circuitry to figure out
-- easier to expand to include new functionality: install bigger ROM.

Advantages of "random logic" controller:

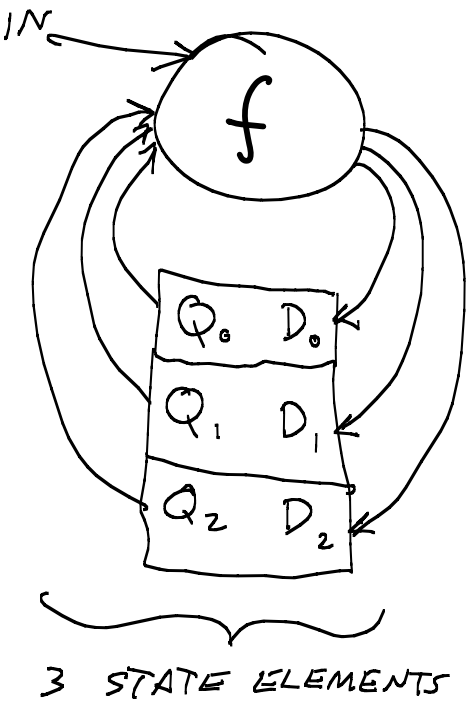
-- faster
-- smaller (?)
-- easier to distribute throughout machine

Let's get back to simpler things (FSM).
We will have:
--- 1-bit state elements
--- 1-bit function elements
How do we put them together to form a FSM?

A mod-3 machine:



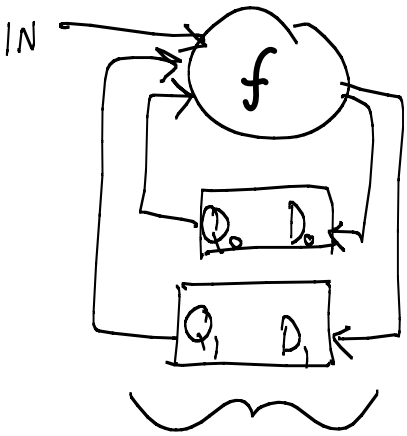
STATE ENCODINGS



3 STATE ELEMENTS

One-hot

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



2 state elements

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

Can we describe f ? Let's use a table.

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

("*" row is for all other rows not shown. "X" is for don't care, either 0 or 1, since cannot happen.)

Can we describe f ? Let's use a table.

inputs: IN	Q1	Q0		outputs: 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.)

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?