ECE 550D
Fundamentals of Computer Systems and Engineering

Fall 2023

Finite State Machines

Xin Li & Dawei Liu
Duke Kunshan University

Slides are derived from work by
Andrew Hilton, Tyler Bletsch and Rabih Younes (Duke)

Last time…

- Who can remind us what we did last time?
  - Flip-flops
  - Registers

Finite Storage = Finite States

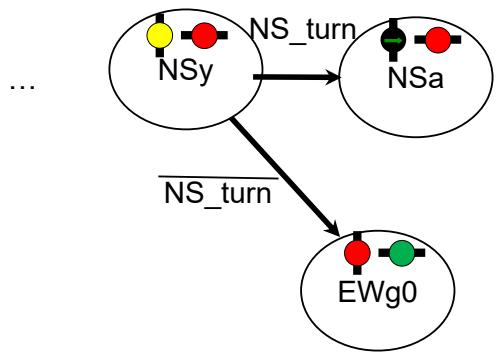
- Computers have finite storage (inside processor):
  - Design in fixed number of DFFs
  - Result: finite number of states (N bits => 2^N states)
- Useful to talk about finite state machines
  - Ubiquitous in processor design
    - Basically how the processor works out many multi-step processes

FSM: Input + Current State = Output + New State

- Finite State Machines
  - Output = g(Input, Current State)
  - New State = f(Input, Current State)
- Example: Traffic Light
  - Input: NS_turn, EW_turn
  - Outputs: which lights are on
    - NS_green
    - NS_g_arrow
    - NS_yellow
    - NS_y_arrow
    - NS_red
    - EW_green
    - …

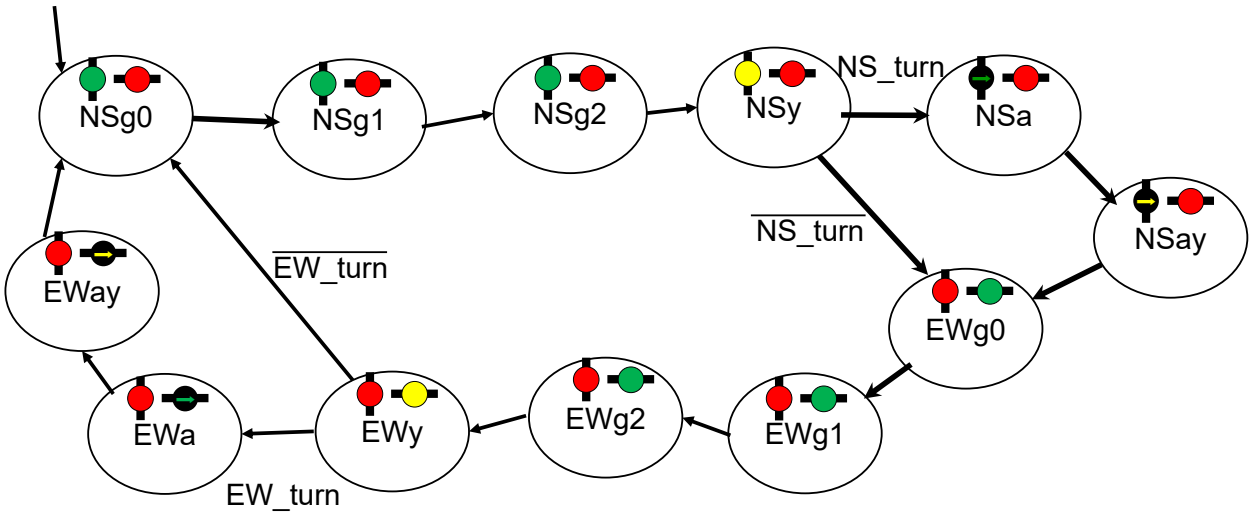
Inductive sensor in road detects car in turn lane

State Diagrams



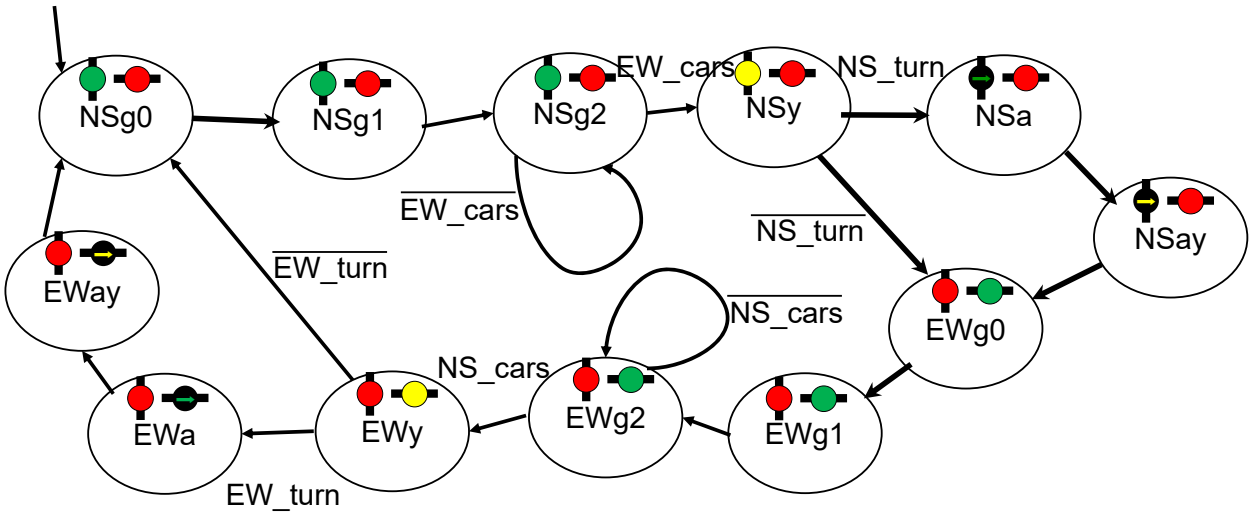
- Can draw state machine as a diagram
  - Circles for states
  - Arrows for transitions (possibly with a choice based on inputs)

State Diagrams



- Full diagram for our traffic light
  - Note start state: NSg0
- Note: real traffic lights have more states
  - Longer greens relative to yellows. All red in before next green…

State Diagrams



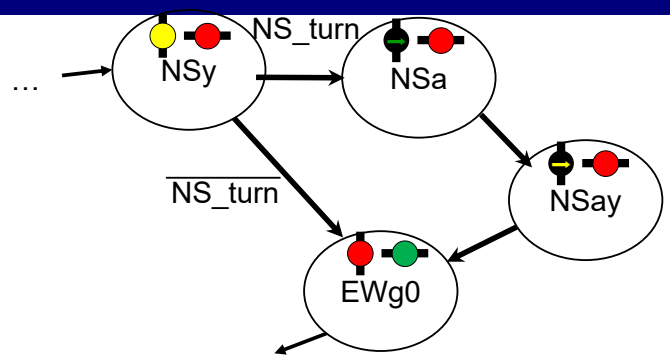
- Could make it smarter/fancier with more inputs
  - E.g., stay green unless opposing traffic present
  - Perfectly fine to have self-loops (stay in same state)

Transition function

Why state_d and state_q?

Will latch state in DFFs from one cycle to next.

state_d = next one
state_q = current one



- State diagrams describes **transition function** pictorially
  - next_state = f (inputs, current_state)
  - Easy to translate into VHDL:

```
state_d <=  EWg0 when state_q = NSy and not NS_turn else
            NSa  when state_q = NSy and NS_turn else
            NSay when state_q = NSa else
            EWg0 when state_q = NSay else ….
```

Can define these as constants

Output function

- Also need an output function:
  - For each output signal, compute as function of inputs and state
    - (or maybe just state, as in traffic lights)

State	ns_g	ns_ga	ns_y	ns_ya	ns_r	ew_g	ew_ga	ew_y	ew_ya	ew_r
NSg	1	0	0	0	0	0	0	0	0	1
NSy										
NSa										
NSay										
EWg										
EWy										
EWa										
EWay										

Output function

- Also need an output function:
  - For each output signal, compute as function of inputs and state
    - (or maybe just state, as in traffic lights)

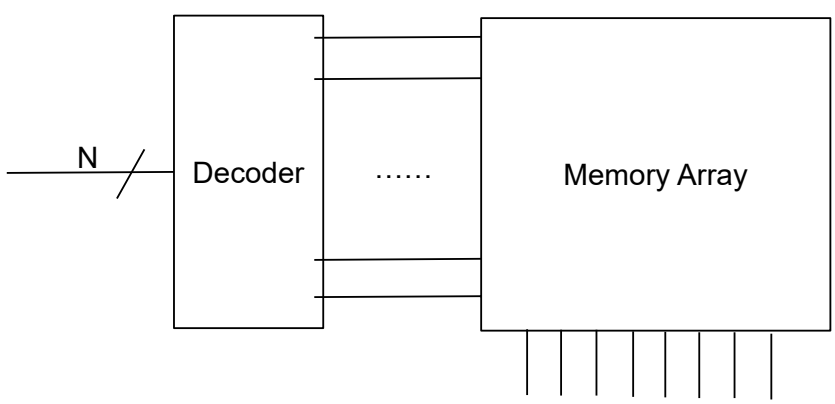
State	ns_g	ns_ga	ns_y	ns_ya	ns_r	ew_g	ew_ga	ew_y	ew_ya	ew_r
NSg	1	0	0	0	0	0	0	0	0	1
NSy	0	0	1	0	0	0	0	0	0	1
NSa	0	1	0	0	0	0	0	0	0	1
NSay	0	0	0	1	0	0	0	0	0	1
EWg	0	0	0	0	1	1	0	0	0	0
EWy	0	0	0	0	1	0	0	1	0	0
EWa	0	0	0	0	1	0	1	0	0	0
EWay	0	0	0	0	1	0	0	0	1	0

Hardware implementation

State	ns_g	ns_ga	ns_y	ns_ya	ns_r	ew_g	ew_ga	ew_y	ew_ya	ew_r
NSg	1	0	0	0	0	0	0	0	0	1
NSy	0	0	1	0	0	0	0	0	0	1
NSa	0	1	0	0	0	0	0	0	0	1
NSay	0	0	0	1	0	0	0	0	0	1
EWg	0	0	0	0	1	1	0	0	0	0
EWy	0	0	0	0	1	0	0	1	0	0
EWa	0	0	0	0	1	0	1	0	0	0
EWay	0	0	0	0	1	0	0	0	1	0

- Hardware implementation option 1:
  - Logic from the truth table

Hardware implementation: ROM



- Can also use ROM
  - Read Only Memory
  - Address goes into decoder
  - One hot word line goes into memory array
  - Data comes out on bit lines
- More details soon (when we do RAMs)

Take a moment to draw an FSM…

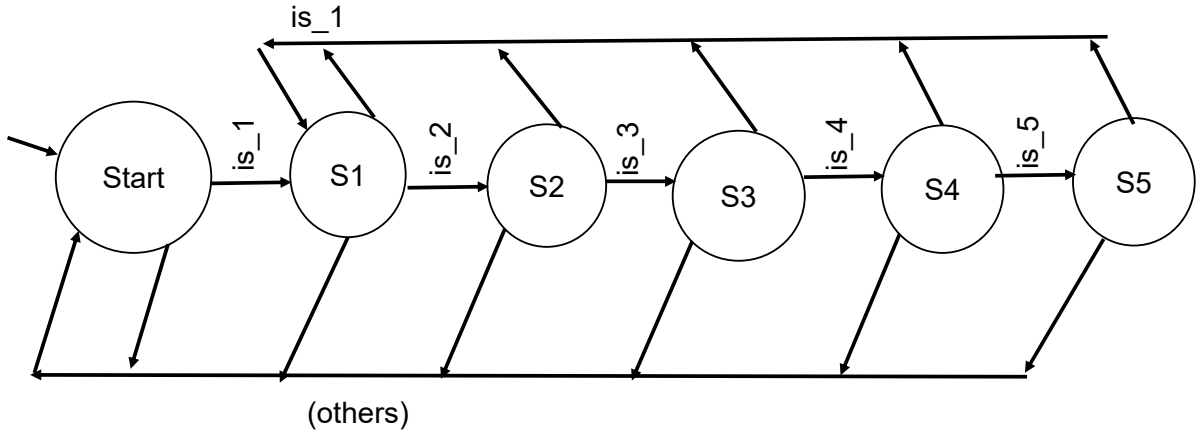
- Take a minute to draw an FSM for a combination lock
  - Combination: 12345

"So the combination is... one, two, three, four, five? That's the stupidest combination I've ever heard in my life! That's the kind of thing an idiot would have on his luggage!"—Dark Helmet (Spaceballs, the movie)

- Inputs:
  - One hot is_0, is_1, is_2, …
- Outputs:
  - Unlock

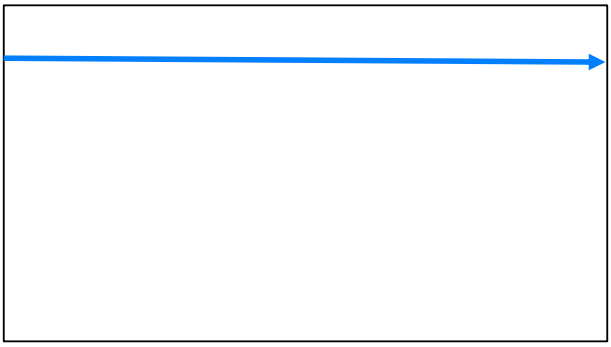
- Draw transitions as state diagram, note which states have unlock on.
  - Feel free to abbreviate "all other cases" by leaving arrow label blank

Combination Lock



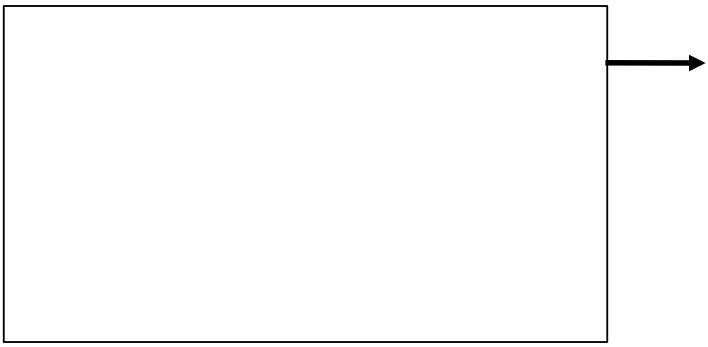
- is_1 always takes us to S1
- Correct input moves us "right"
- Other: back to start
- S5 unlocks

VGA controller: FSM



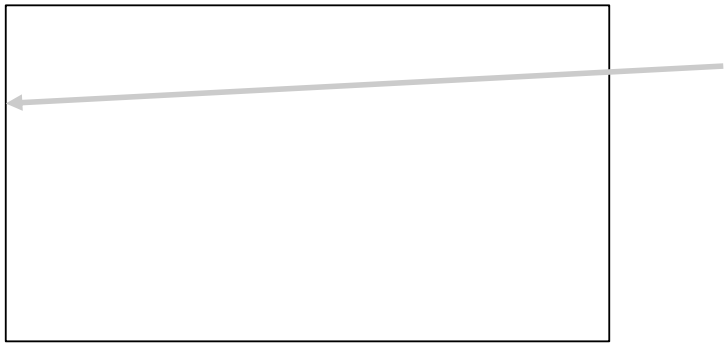
- Recitation will have FSM to implement
  - VGA controller
  - Scan row from left to right, sending out data pixel by pixel
    - One pixel per cycle

VGA controller: FSM



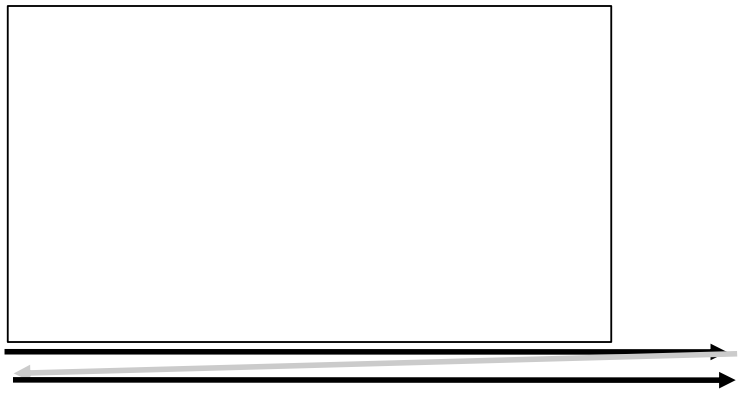
- Recitation will have FSM to implement
  - VGA controller
  - Scan row from left to right, sending out data pixel by pixel
    - One pixel per cycle
  - Then period of black (all 0 pixel) with some control signals

VGA controller: FSM



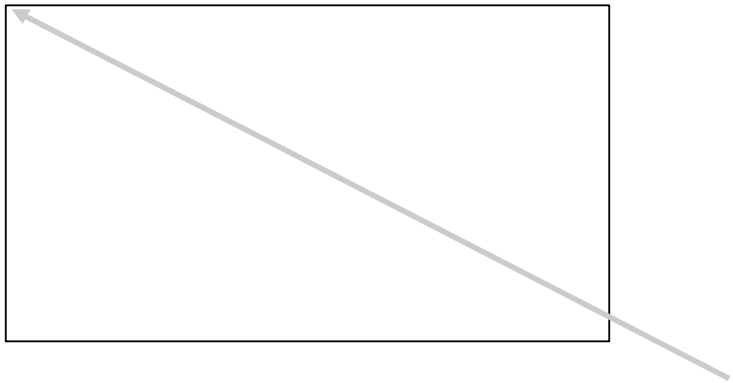
- Recitation will have FSM to implement
  - VGA controller
  - Scan row from left to right, sending out data pixel by pixel
    - One pixel per cycle
  - Then period of black (all 0 pixel) with some control signals
  - Then restart on next row

VGA controller: FSM



- VGA controller
  - Trace blank rows
    - All black, goes through same horizontal states as real rows
  - Reach last row

VGA controller: FSM



- VGA controller
  - Trace blank rows
    - All black, goes through same horizontal states as real rows
  - Reach last row
  - Then reset to top left corner

Division: math with an FSM

```
      15224 R 1
3 | 45673
    3
    15
    15
     06
      6
      07
       6
       13
       12
        1
```

- We have talked about add, sub
  - Pretty easy math to implement in hardware
- What about divide?
  - Much more complicated
  - Multi-step process
  - Well suited to FSM

Division: Binary

$$11 \overline{)101101}$$

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

$$\begin{array}{r} 0 \\ 11 \overline{)101101} \end{array}$$

1 11 > 1

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      00
11 |101101
```

10 11 > 10

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      001
11 |101101
```

101 11 <= 101

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      001
11 |101101
```

10 101 – 11 = 10

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      0011
11 |101101
```

101 11 <= 101

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      0011
11 |101101
```

10 101 – 11 = 10

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      00111
11 |101101
```

100 11 <= 100

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      00111
11 | 101101
```

1 100 – 11 = 1

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      001111
11 | 101101
```

11 11 <= 11

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

Division: Binary

```
      001111
11 | 101101
```

0 11– 11 = 0

- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times

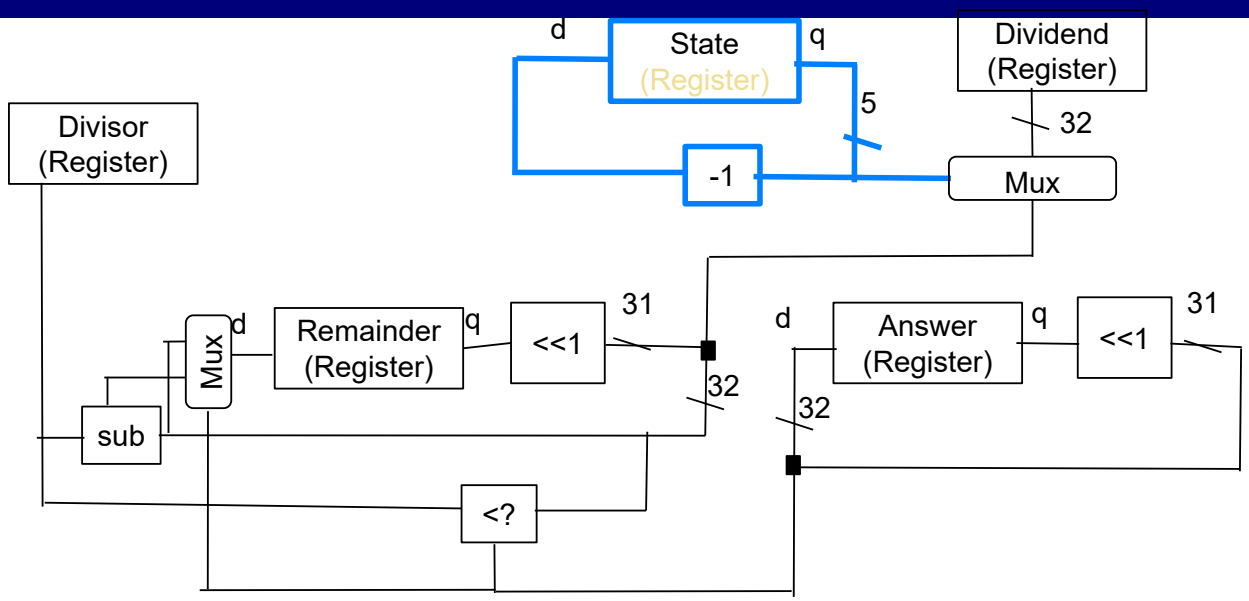
Division: Binary

```
      001111  = Answer
11 | 101101
```

Remainder =0 Done

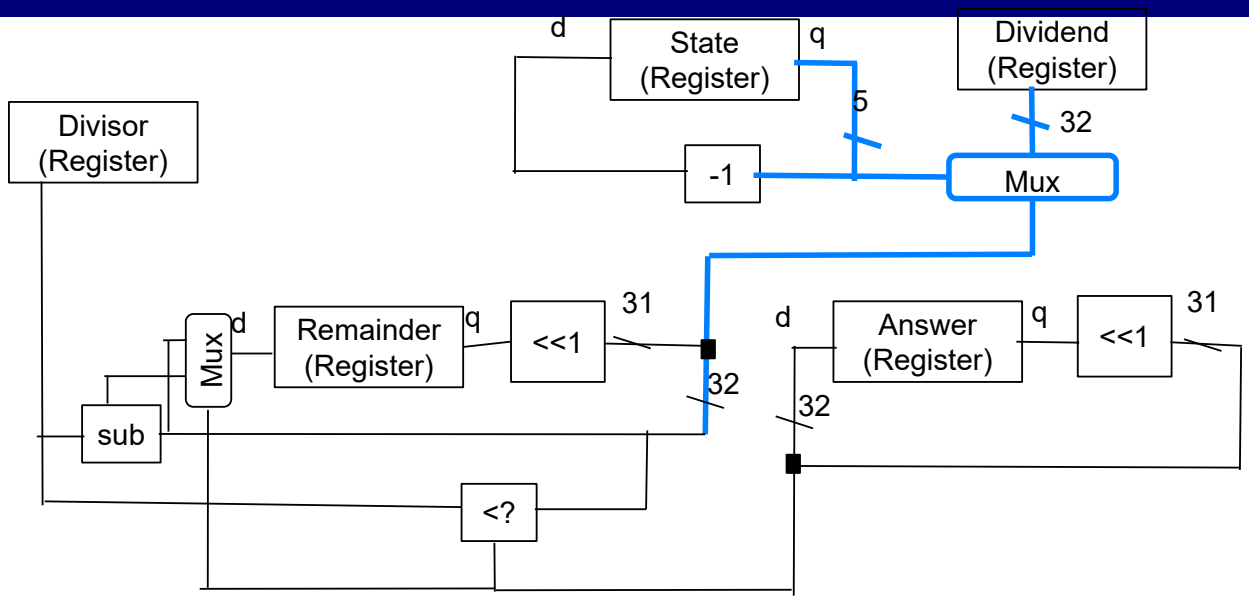
- Binary long division similar to decimal
  - But a little simpler, because it goes in 1 or 0 times
  - 45 / 3 = 15 remainder 0

Division FSM/Circuit



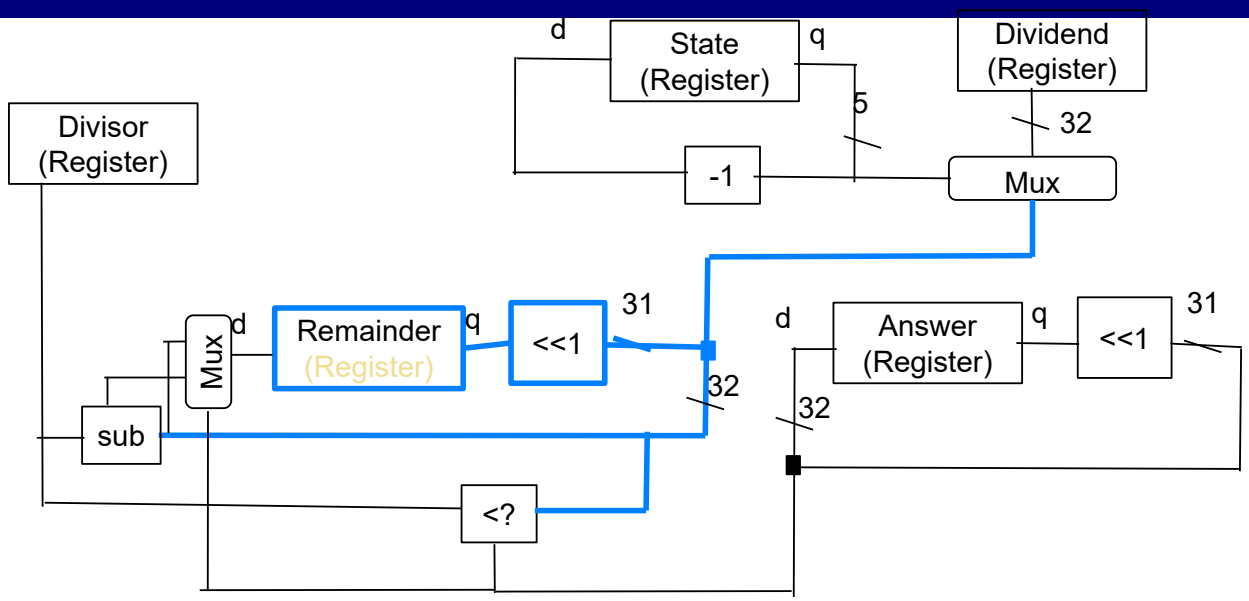
- 32 bit division: 32 states (5 bits)
  - Decrement state # each cycle (count down which bit)

Division FSM/Circuit



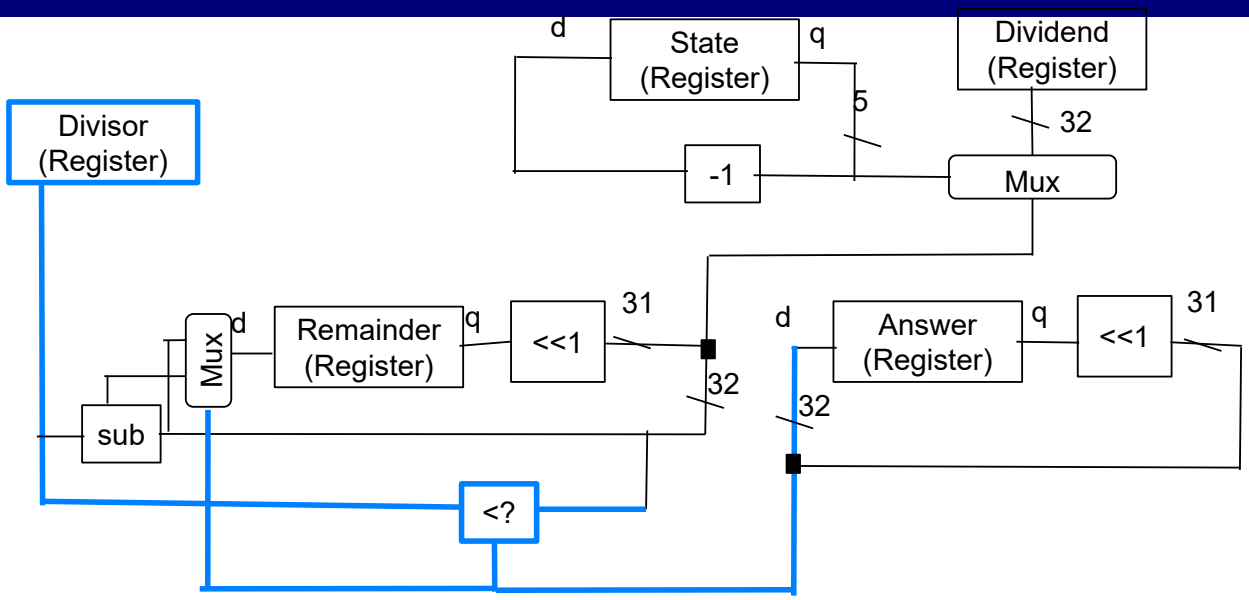
- Use State # to pick out which bit of Dividend

Division FSM/Circuit



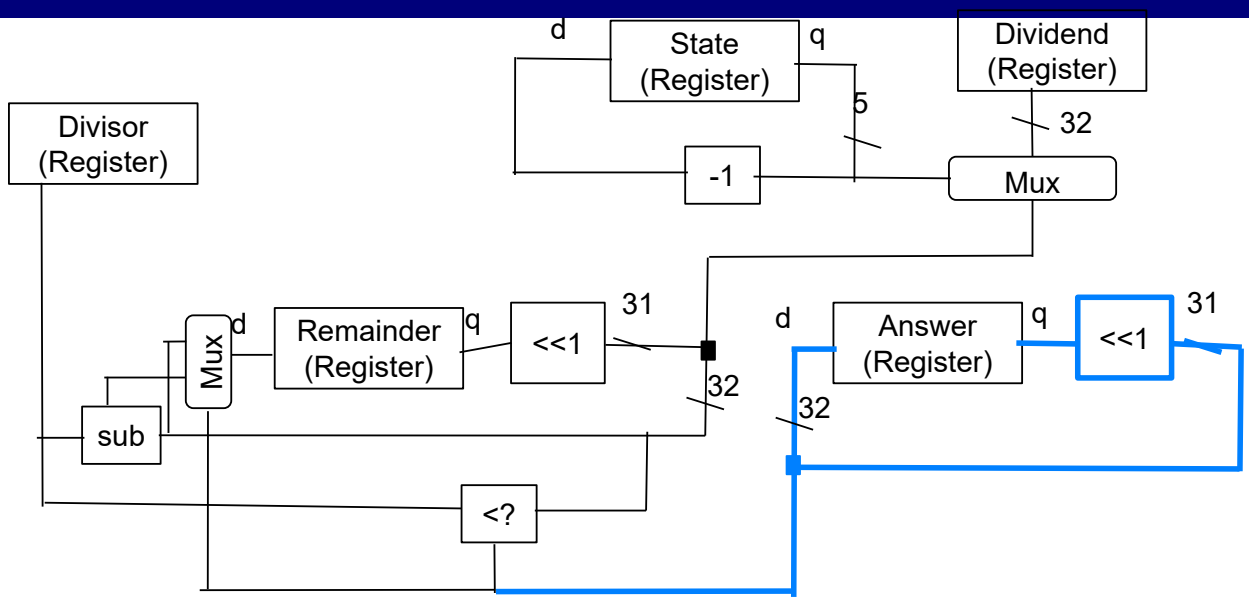
- Shift remainder left 1, concatenate dividend bit at right

Division FSM/Circuit



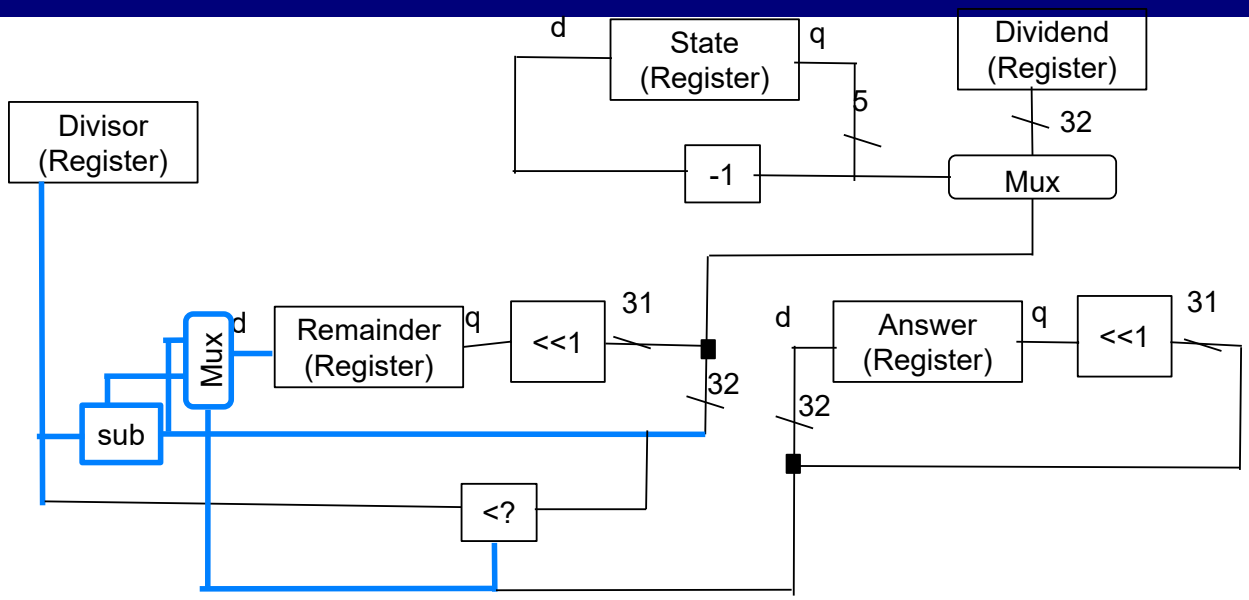
- Check if divisor is < result… used for two things
  - Mux selector on remainder_d
  - Lowest bit of answer_d

Division FSM/Circuit



- For answer, shift old answer <<1, concatenate in < result

Division FSM/Circuit



- For remainder, pick from two things (based on < result)
  - Result of shifting old remainder and concatenating dividend bit
  - That minus the divisor

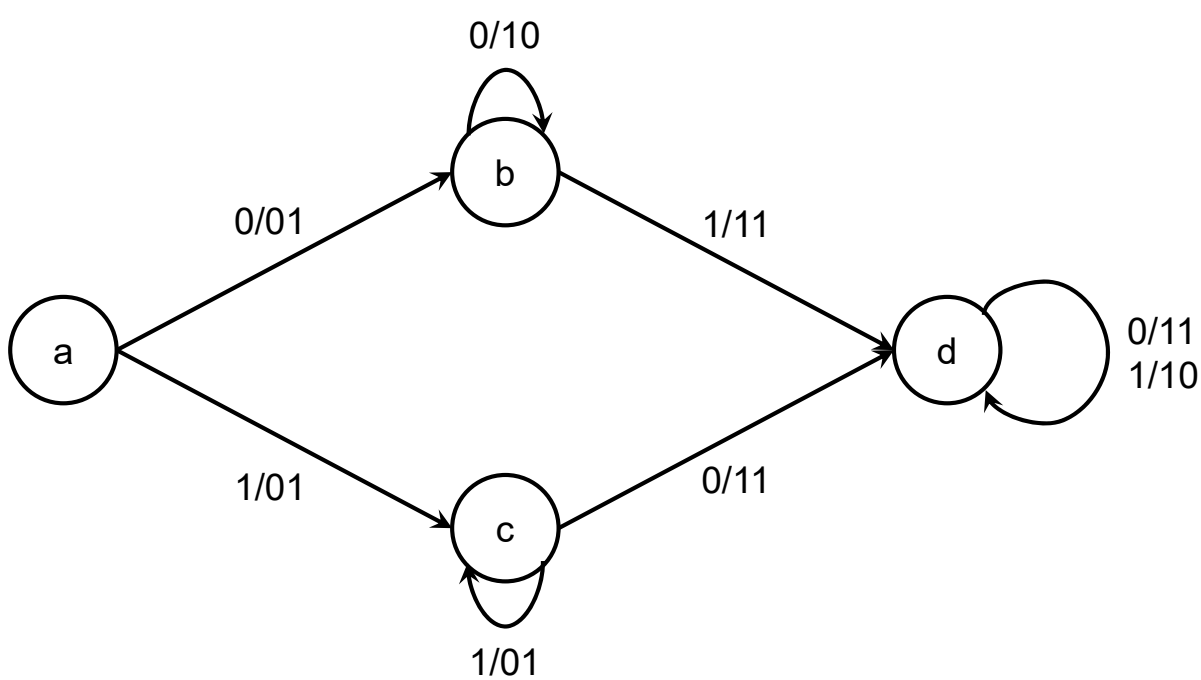
Mealy Machine

- New State = f(Input, Current State)
- Output = g(Input, Current State)

Current State	Input = 0		Input = 1	
	New State	Output	New State	Output
a	b	01	c	01
b	b	10	d	11
c	d	11	c	01
d	d	11	d	10

Mealy Machine

- State diagram



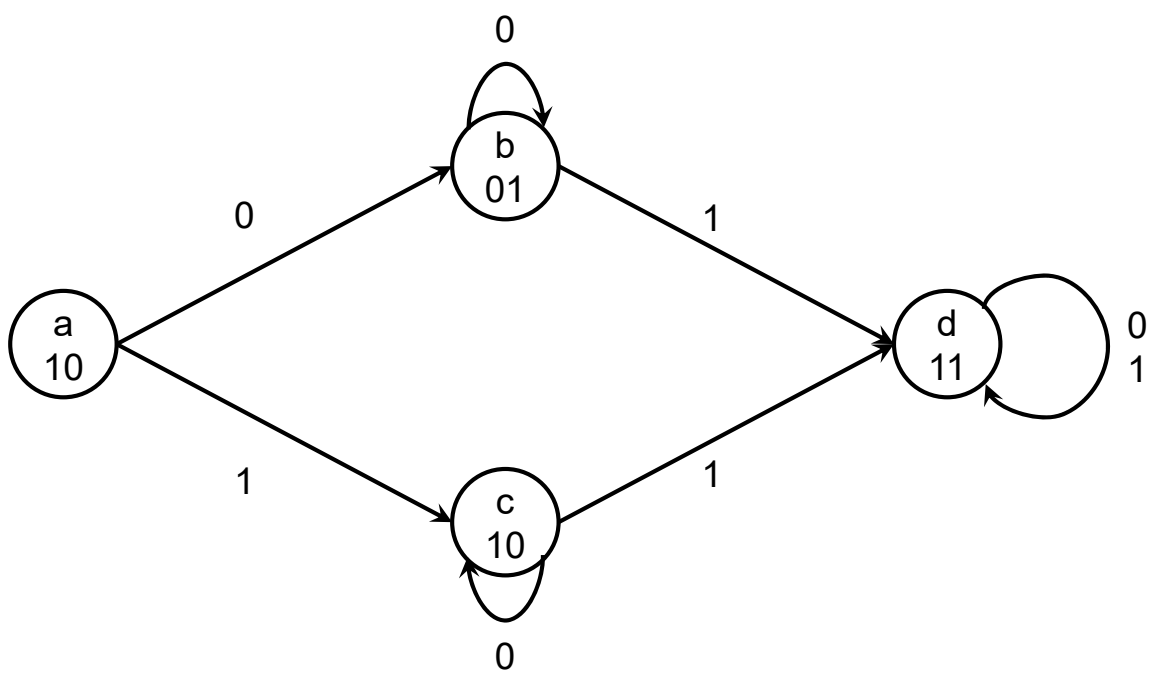
Moore Machine

- New State = f(Input, Current State)
- Output = g(Current State)

Current State	Input = 0	Input = 1	Output
	New State	New State	
a	b	c	10
b	b	d	01
c	c	d	10
d	d	d	11

Moore Machine

- State diagram



Excitation Table

Current State	New State	Input
a	b	0
a	c	1
b	b	0
b	d	1
c	c	0
c	d	1
d	d	x

Summary

- Finite State Machine
  - Finite states (encoded in some way: binary nums, one-hot…)
  - Transition function: (state * inputs) -> state
    - Helpful to draw as diagram
  - Output function: (state * inputs) -> outputs
- Examples:
  - Traffic light
  - VGA controller
  - Combination lock
  - Division
- Mealy machine and Moore machine