$\boxed{\text{Lecture 22}}$

Andrea LaPaugh

April 25, 1996

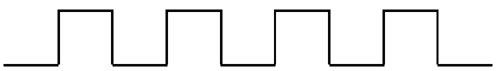
(Lecture 21 notes with Lecture 20 notes)

Registers

Want R1 <- R1 + R2

Still reading old value while storing new value?

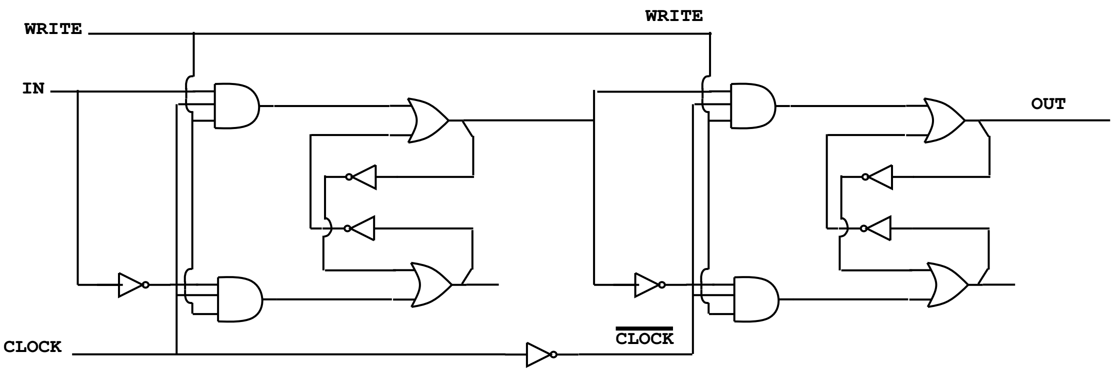
Solution: use a clock
regular (in time) alternation of 1 and 0



while clock=1 register value is stable
can compute with it

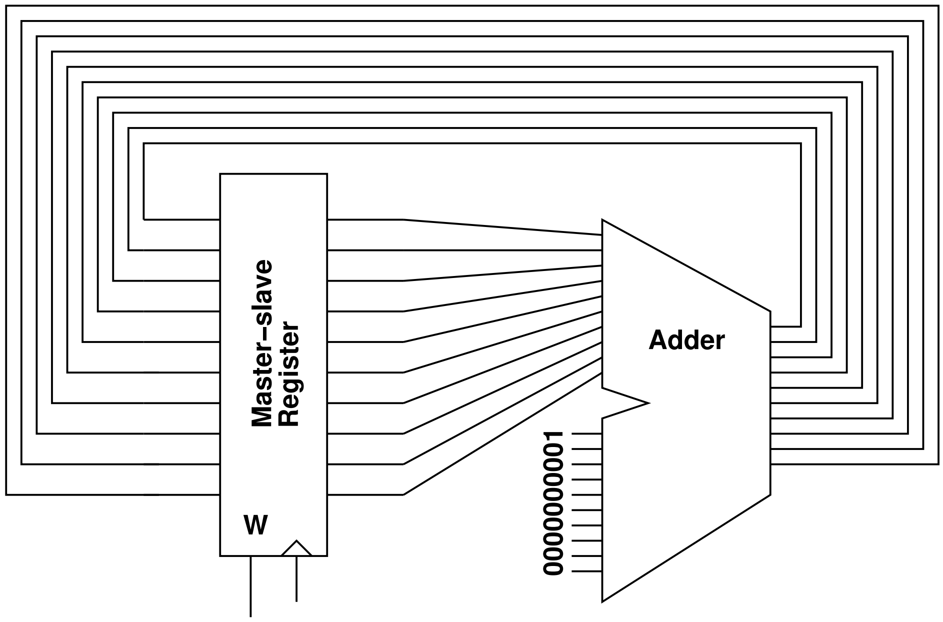
clock <- 0 receive new register value
change on "falling edge" of clock

use master-slave flip-flop



Counter

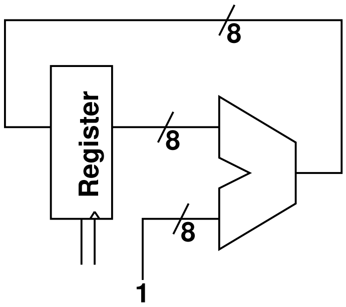
Master-slave flip-flop with feedback



Internal state changes when clock goes to 1

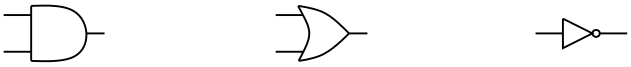
Output changes when clock goes from 1 to 0

But it's tiresome drawing all those wires:

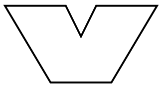


Building blocks

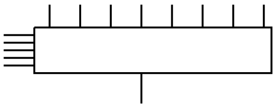
AND, OR, NOT



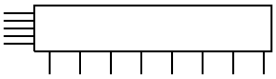
Adder



Multiplexer



Decoder



Flip-flop

Master-Slave Register

Memory

Counter

Clock

Control

Multiplexers allow adders, registers, memory, ...
to take inputs from various places -- choice

How control?

e.g. where get load signals?

Model for controllers: Finite State Machines

a machine with

S states

N input bits

M output bits

A computation step of the machine is:

read inputs

compute new state and outputs based on
current state and inputs

change to new state

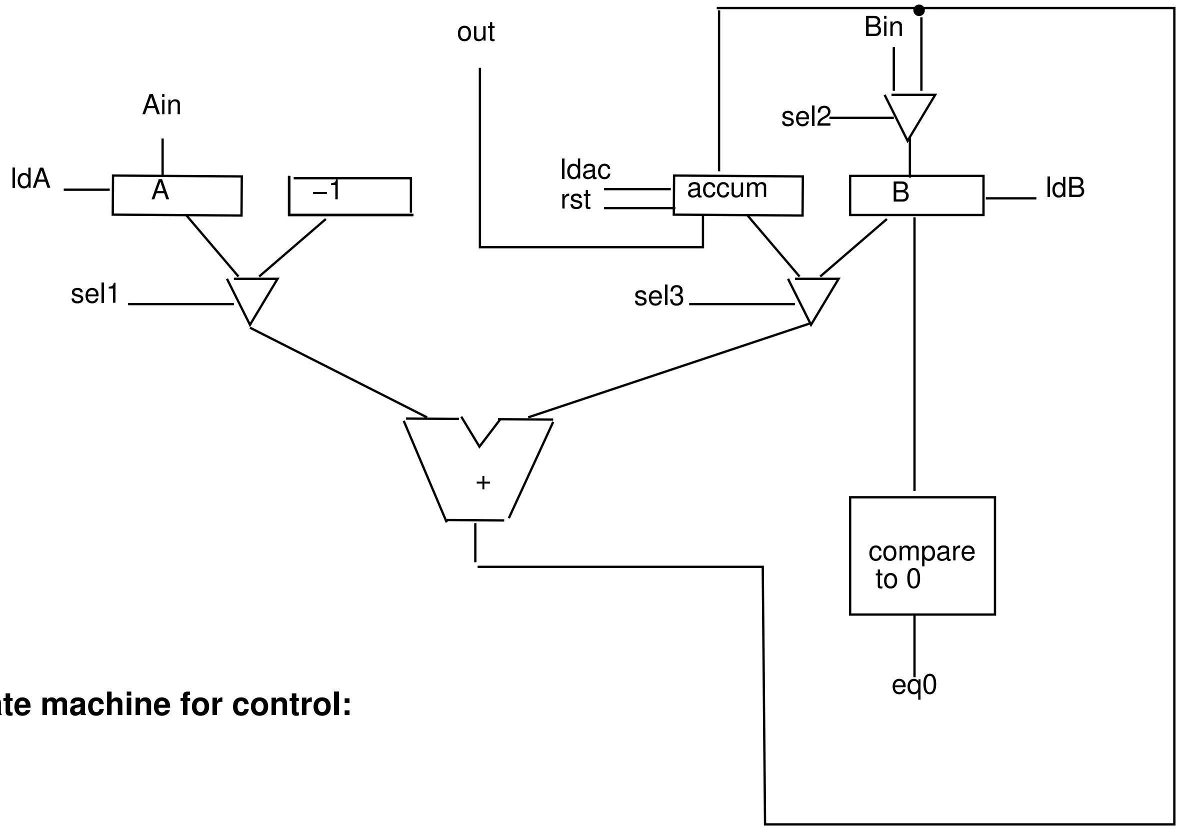
Repeat computation step until reach ``halt'' state

Machine has memory, held in ``state''

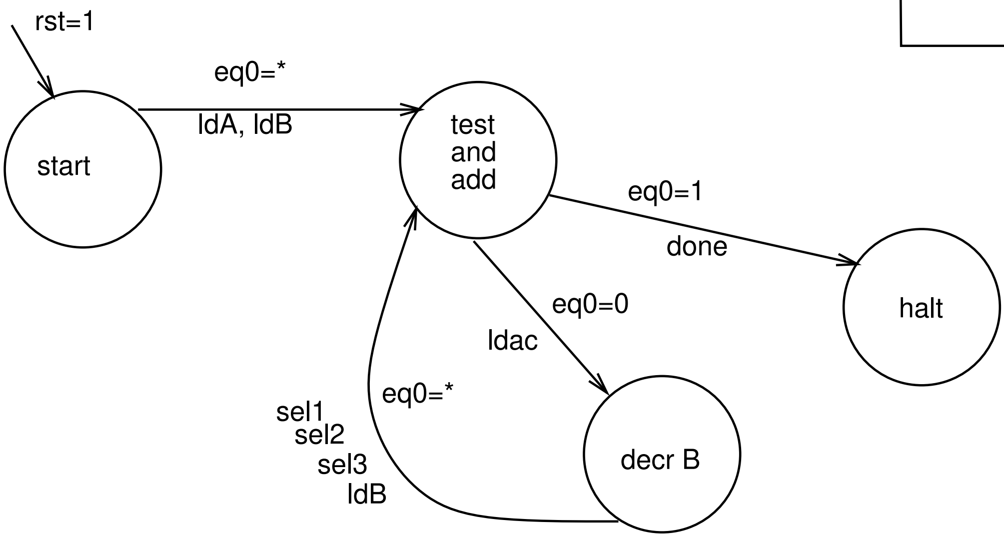
Moves are at discrete times (new state vs old state)

FSM example

Multiply in hardware using an adder:



Finte state machine for control:



fsm input: eq0 (from comparator), rst (reset)
fsm output: ldA, ldB, ldac, sel1, sel2, sel3, done
start state always entered on reset=1;
other states entered when reset=0; value of eq0 shown

How build an FSM?

encode states using bits (unary vs binary)

function from state bits and input bits

to new state bits and output bits

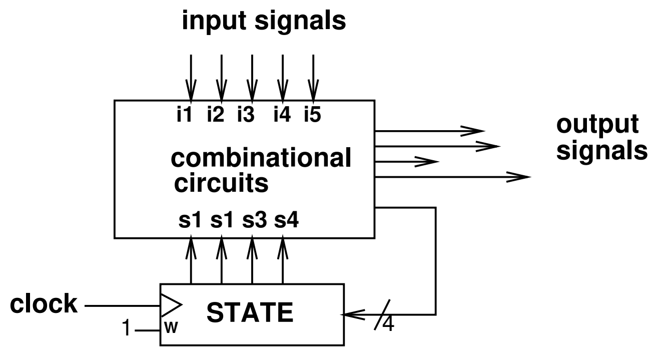
is really just several Boolean functions

one for each new state bit and new output bit

implement each function as combinational logic,

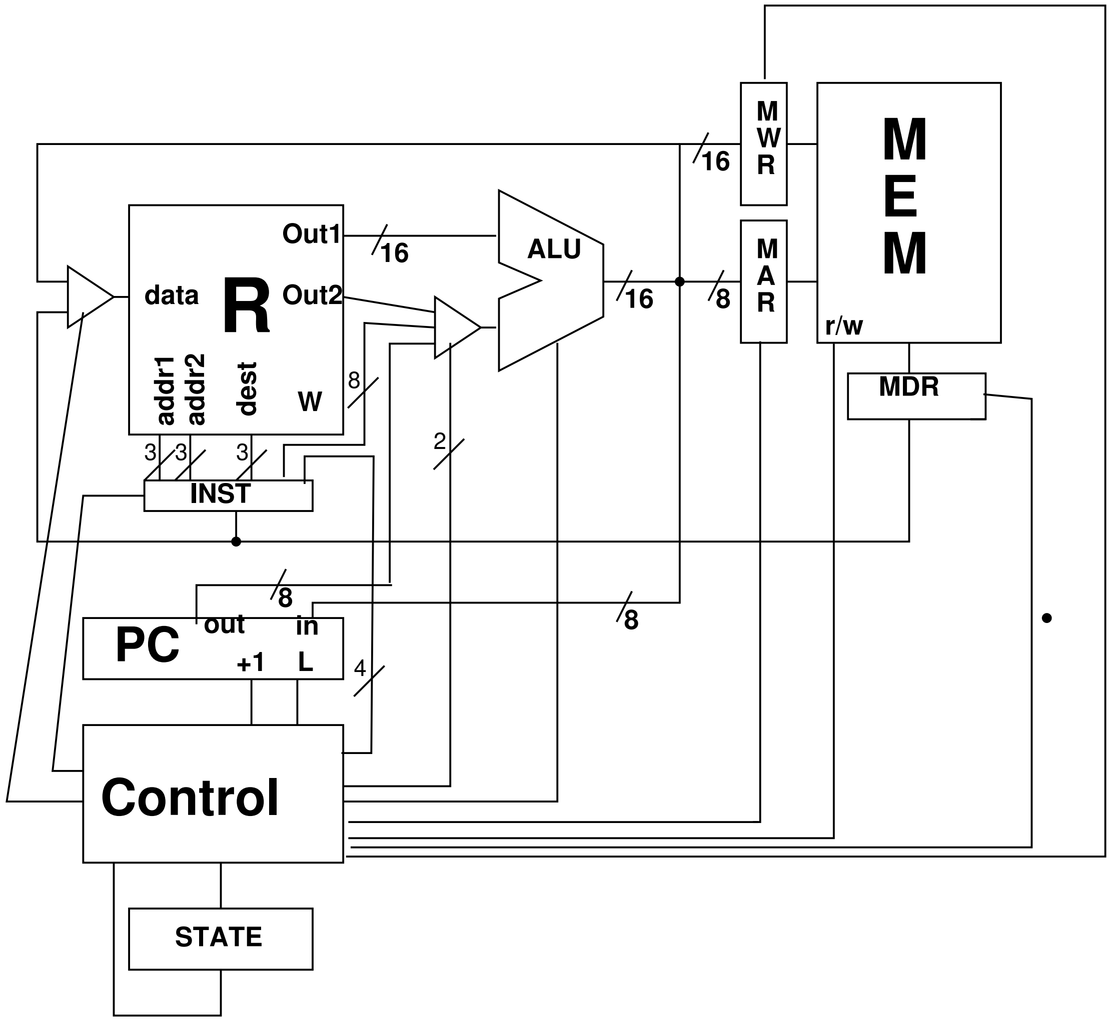
add master-slave clocked register to hold state,

and you have a fsm:



Basic Machine Organization

(Not all details shown)



Two types of wires
- data: connect registers (copy signals)
- control: enable connections

Two places where compute (vs. move data)
- ALU (multiple fuctions)
- Control

machine control

Sequences of control signals "move" information

Example:

20: 1234   R2 <- R3 + R4

Step	Control Signal
0	PC -> ALU -> MAR
1	MEM -> MDR, PC+1 -> PC
2	MDR -> INST, INST[4-6]->addr1, INST[0-2]->addr2, INST[8-10]->dest
3	R[addr1]->ALU1, R[addr2]->ALU2, ALU1+ALU2->R[dest]

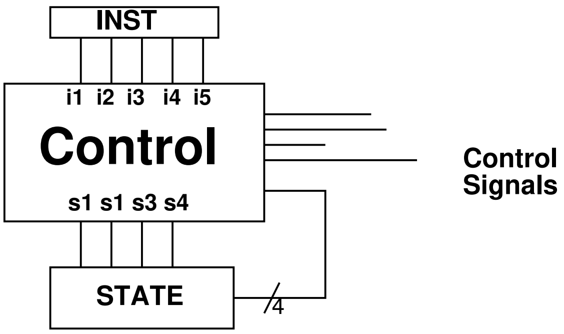
Final Step

Q: How can sequence of control signals

be made to happen?

A: Implement a finite state machine using

instruction op code as input



FETCH-INCREMENT-EXECUTE cycle

implemented by fsm

In each clock cycle, different control signals raised

Crucial distinctions

clock cycle of machine: externally generated periodic pulse

instruction cycle of machine: it may take many clock cyles

to do one instruction