CS 61C:
Great Ideas in Computer Architecture
Introduction to Hardware: Representations and State

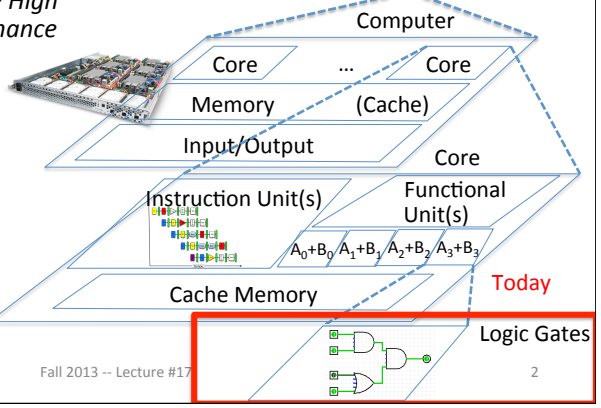
Instructor:
Randy H. Katz
http://inst.eecs.Berkeley.edu/~cs61c/fa13

You are Here!

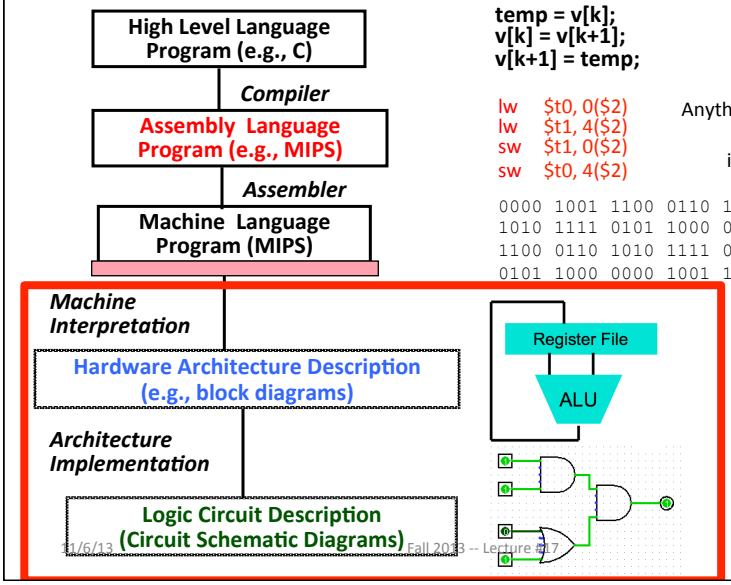
Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time
- Programming Languages

Harness Parallelism & Achieve High Performance



Levels of Representation/ Interpretation



Anything can be represented as a *number*, i.e., data or instructions

Agenda

- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Logisim
- State Machines
- And in Conclusion, …

Agenda

- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Logisim
- State Machines
- And in Conclusion, …

Hardware Design

- Next several weeks: how a modern processor is built, starting with basic elements as building blocks
- Why study hardware design?
  - Understand capabilities and limitations of hw in general and processors in particular
  - What processors can do fast and what they can't do fast (avoid slow things if you want your code to run fast!)
  - Background for more in depth hw courses (CS 152)
  - Hard to know what will need for next 30 years
  - There is just so much you can do with standard processors: you may need to design own custom hw for extra performance
    - Even some commercial processors today have customizable hardware!

Synchronous Digital Systems

Hardware of a processor, such as the MIPS, is an example of a Synchronous Digital System

Synchronous:

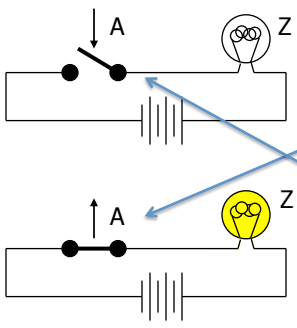
- All operations coordinated by a central clock
  - "Heartbeat" of the system!

Digital:

- Represent all values by two discrete values
- Electrical signals are treated as 1's and 0's
  - 1 and 0 are complements of each other
- High /low voltage for true / false, 1 / 0

Switches: Basic Element of Physical Implementations

- Implementing a simple circuit (arrow shows action if wire changes to "1" or is *asserted*):



Close switch (if A is "1" or asserted) and turn on light bulb (Z)

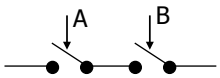
Open switch (if A is "0" or unasserted) and turn off light bulb (Z)

Z ≡ A

Switches (cont'd)

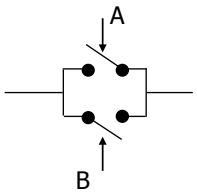
- Compose switches into more complex ones (Boolean functions):

AND



Z ≡ A and B

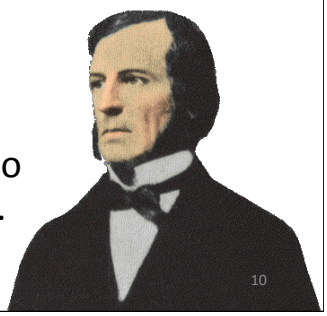
OR



Z ≡ A or B

Historical Note

- Early computer designers built ad hoc circuits from switches
- Began to notice common patterns in their work: ANDs, ORs, …
- Master's thesis (by Claude Shannon) made link between work and 19th Century Mathematician George Boole
  - Called it "Boolean" in his honor
- Could apply math to give theory to hardware design, minimization, …



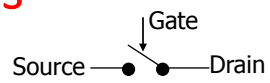
Transistors

- High voltage (V_{dd}) represents 1, or true
- Low voltage (0 volts or Ground) represents 0, or false
- Let threshold voltage (V_{th}) decide if a 0 or a 1
- If switches control whether voltages can propagate through a circuit, can build a computer
- Our switches: CMOS transistors

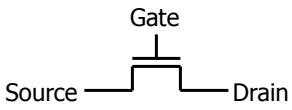
CMOS Transistor Networks

- Modern digital systems designed in CMOS
  - MOS: Metal-Oxide on Semiconductor
  - C for complementary: use *pairs* of normally-open and normally-closed switches
    - Used to be called COS-MOS for complementary-symmetry - MOS
- CMOS transistors act as voltage-controlled switches
  - Similar, though easier to work with, than relay switches from earlier era
  - Use energy primarily when switching

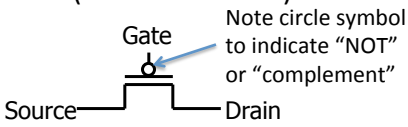
CMOS Transistors



- Three terminals: source, gate, and drain
  - Switch action:
    if voltage on gate terminal is (some amount) higher/lower than source terminal then conducting path established between drain and source terminals (switch is closed)



n-channel transitor
open when voltage at Gate is low
closes when:
voltage(Gate) > voltage (Threshold)
(High resistance when gate voltage Low, Low resistance when gate voltage High)



p-channel transistor
closed when voltage at Gate is low
opens when:
voltage(Gate) > voltage (Threshold)
(Low resistance when gate voltage Low, High resistance when gate voltage High)

CMOS Circuit Rules

- Don't pass weak values => Use Complementary Pairs
  - N-type transistors pass weak 1's (V_{dd} - V_{th})
  - N-type transistors pass strong 0's (ground)
  - Use N-type transistors only to pass 0's (N for negative)
  - Converse for P-type transistors: Pass weak 0s, strong 1s
    - Pass weak 0's (V_{th}), strong 1's (V_{dd})
    - Use P-type transistors only to pass 1's (P for positive)
  - Use pairs of N-type and P-type to get strong values
- Never leave a wire undriven
  - Make sure there's always a path to V_{dd} or gnd
- Never create a path from V_{dd} to gnd (ground)

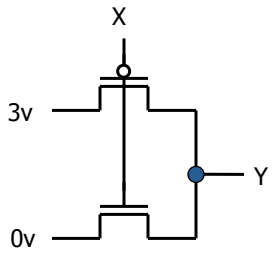
Administrivia

Agenda

- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Logisim
- State Machines
- And in Conclusion, …

MOS Networks

p-channel transistor
closed when voltage at Gate is low
opens when:
voltage(Gate) > voltage (Threshold)



n-channel transitor
open when voltage at Gate is low
closes when:
voltage(Gate) > voltage (Threshold)

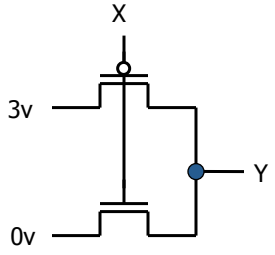
what is the relationship between x and y?

x	y
0 volts (gnd)	
3 volts (Vdd)	

Called an *invertor* or *not gate*

MOS Networks

n-channel transitor
open when voltage at Gate is low
closes when voltage(Gate) > voltage (Source) + ε



p-channel transistor
closed when voltage at Gate is low
opens when voltage(Gate) < voltage (Source) – ε

what is the relationship between x and y?

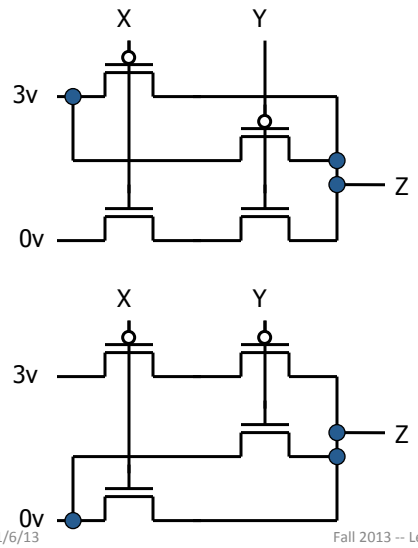
x	y
0 volts (gnd)	3 volts (Vdd)
3 volts (Vdd)	0 volts (gnd)

Called an *invertor* or *not gate*

$P = ½ C V^2 f$

- Dynamic Energy (when switching) is proportional to Capacitance * Voltage^2
- Since pulse is 0 -> 1 -> 0 or 1 -> 0 -> 1, Energy of a single transition is proportional to ½ *Capacitance * Voltage^2
- Power is just energy per transition times frequency of transitions: proportional to ½ * Capacitance * Voltage^2 * Frequency

Two Input Networks

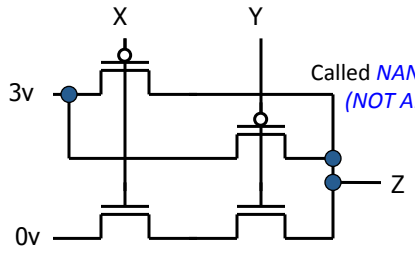


what is the relationship between x, y and z?

x	y	z
0 volts	0 volts	
0 volts	3 volts	
3 volts	0 volts	
3 volts	3 volts	

x	y	z
0 volts	0 volts	
0 volts	3 volts	
3 volts	0 volts	
3 volts	3 volts	

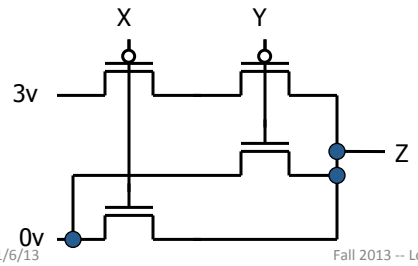
Two Input Networks: Peer Instruction



Called *NAND gate (NOT AND)*

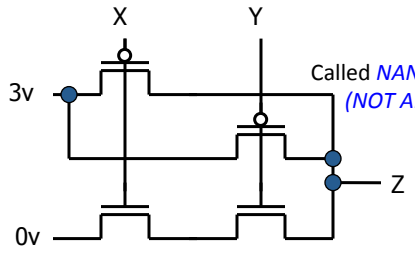
what is the relationship between x, y and z?

x	y	z
0 volts	0 volts	3 volts
0 volts	3 volts	3 volts
3 volts	0 volts	3 volts
3 volts	3 volts	0 volts



x	y	z				
		A	B	C	D	
0 volts	0 volts	0	0	3	3	volts
0 volts	3 volts	0	3	0	3	volts
3 volts	0 volts	0	3	0	3	volts
3 volts	3 volts	3	3	0	0	volts

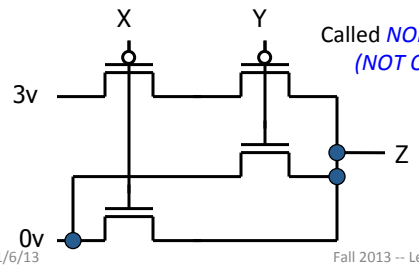
Two Input Networks



Called *NAND gate (NOT AND)*

what is the relationship between x, y and z?

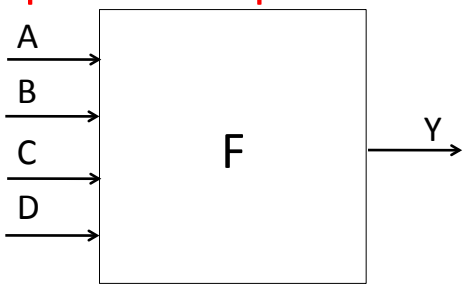
x	y	z
0 volts	0 volts	3 volts
0 volts	3 volts	3 volts
3 volts	0 volts	3 volts
3 volts	3 volts	0 volts



Called *NOR gate (NOT OR)*

x	y	z
0 volts	0 volts	3 volts
0 volts	3 volts	0 volts
3 volts	0 volts	0 volts
3 volts	3 volts	0 volts

Truth Tables
List outputs for all possible inputs



a	b	c	d	y
0	0	0	0	F(0,0,0,0)
0	0	0	1	F(0,0,0,1)
0	0	1	0	F(0,0,1,0)
0	0	1	1	F(0,0,1,1)
0	1	0	0	F(0,1,0,0)
0	1	0	1	F(0,1,0,1)
0	1	1	0	F(0,1,1,0)
0	1	1	1	F(0,1,1,1)
1	0	0	0	F(1,0,0,0)
1	0	0	1	F(1,0,0,1)
1	0	1	0	F(1,0,1,0)
1	0	1	1	F(1,0,1,1)
1	1	0	0	F(1,1,0,0)
1	1	0	1	F(1,1,0,1)
1	1	1	0	F(1,1,1,0)
1	1	1	1	F(1,1,1,1)

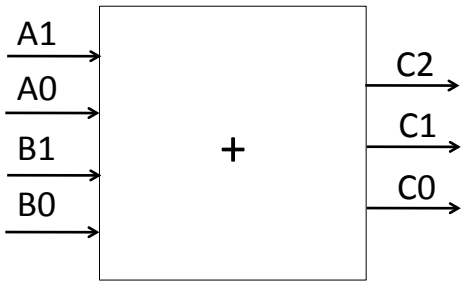
# Truth Table Example #1:
y= F(a,b): 1 iff a ≠ b

a	b	y
0	0	0
0	1	1
1	0	1
1	1	0

Truth Table Example #2: 2-bit Adder

A a_1a_0	B b_1b_0	C $c_2c_1c_0$

How Many Rows?



Truth Table Example #3: 32-bit Unsigned Adder

A	B	C
000 ... 0	000 ... 0	000 ... 00
000 ... 0	000 ... 1	000 ... 01
.	.	.
.	.	.
.	.	.
111 ... 1	111 ... 1	111 ... 10

How Many Rows?

Truth Table Example #4: 3-input Majority Circuit

a	b	c	y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

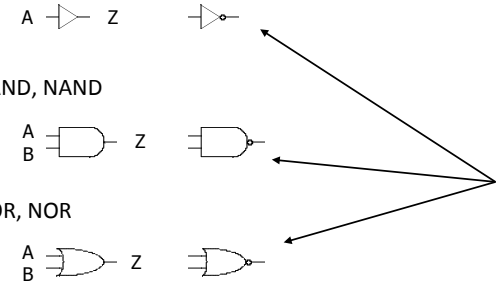
Y =

This is called *Sum of Products* form;
Just another way to represent the TT as a logical expression

More simplified forms
(fewer gates and wires)

Combinational Logic Symbols

- Common combinational logic systems have standard symbols called logic gates
  - Buffer, NOT
  - AND, NAND
  - OR, NOR



Easy to implement with CMOS transistors (the switches we have available and use most)

Agenda

- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Logisim if there is time
- State Machines
- And in Conclusion, …

Boolean Algebra

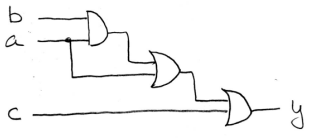
- Use plus for OR
  - "logical sum"
- Use product for AND (a•b or implied via ab)
  - "logical product"
- "Hat" to mean complement (NOT)
- Thus

  $ab + a + \overline{c}$

= $a \bullet b + a + \overline{c}$

= (a AND b) OR a OR (NOT c)

Boolean Algebra: Circuit & Algebraic Simplification



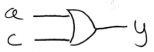
original circuit

$$y = ((ab) + a) + c$$

equation derived from original circuit

$$= ab + a + c$$
$$= a(b + 1) + c$$
$$= a(1) + c$$
$$= a + c$$

algebraic simplification



simplified circuit

Laws of Boolean Algebra

$X \overline{X} = 0$	$X + \overline{X} = 1$	Complementarity
$X\ 0 = 0$	$X + 1 = 1$	Laws of 0's and 1's
$X\ 1 = X$	$X + 0 = X$	Identities
$X\ X = X$	$X + X = X$	Idempotent Laws
$X\ Y = Y\ X$	$X + Y = Y + X$	Commutativity
$(X\ Y)\ Z = Z\ (Y\ Z)$	$(X + Y) + Z = Z + (Y + Z)$	Associativity
$X\ (Y + Z) = X\ Y + X\ Z$	$X + Y\ Z = (X + Y)\ (X + Z)$	Distribution
$X\ Y + X = X$	$(X + Y)\ X = X$	Uniting Theorem
$\overline{X}\ Y + X = X + Y$	$(\overline{X} + Y)\ X = X\ Y$	United Theorem v. 2
$\overline{X\ Y} = \overline{X} + \overline{Y}$	$\overline{X + Y} = \overline{X}\ \overline{Y}$	DeMorgan's Law

Boolean Algebraic Simplification Example

$$y = ab + a + c$$

Boolean Algebraic Simplification Example

$$\begin{aligned} y &= ab + a + c \\ &= a(b+1) + c && \textit{distribution, identity} \\ &= a(1) + c && \textit{law of 1's} \\ &= a + c && \textit{identity} \end{aligned}$$

a	b	c	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Agenda

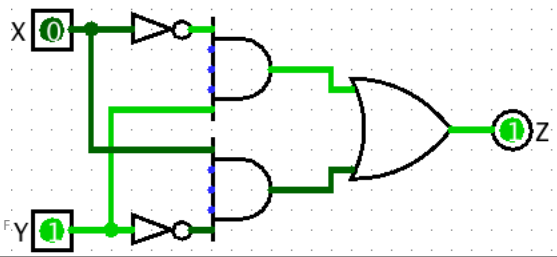
- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Logisim
- State Machines
- And in Conclusion, …

Logisim

- Free schematic capture/logic simulation program in Java
  - "A graphical tool for designing and simulating logic circuits"
  - Search and download version 2.7.1, online tutorial
  - *ozark.hendrix.edu/~burch/**logisim**/*
- Drawing interface based on toolbar
  - Color-coded wires aid in simulating and debugging a circuit
  - Wiring tool draws horizontal and vertical wires, automatically connecting to components and to other wires.
- Circuit layouts used as "subcircuits" of other circuits, allowing hierarchical circuit design
- Included circuit components: inputs and outputs, gates, multiplexers, arithmetic circuits, flip-flops, RAM memory

Logisim Wires

- Blue wires: value at that point is "unknown"
- Gray wires: not connected to anything
- OK when in process of building a circuit
- When finished => wires not be blue or gray
- If connected, all wires should be green
  - Bright green a 1
  - Dark green a 0



Common Mistakes in Logisim

- Connecting wires together
- Using input for output
- Connecting to edge without connecting to actual input
  - Unexpected direction of input

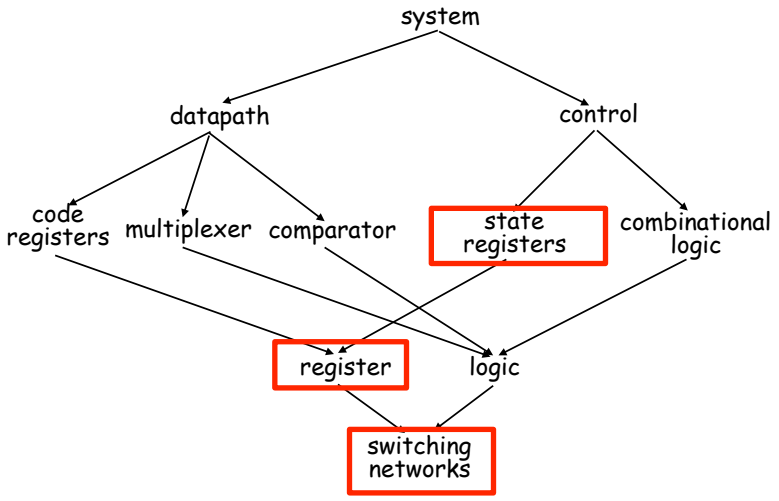
Agenda

- Switching Networks, Transistors
- Gates and Truth Tables for Circuits
- Boolean Algebra
- Logisim
- State Machines
- And in Conclusion, …

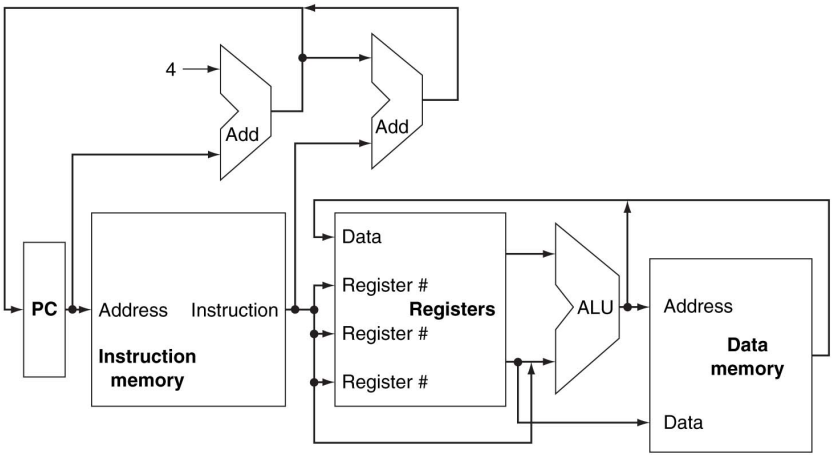
Type of Circuits

- *Synchronous Digital Systems* consist of two basic types of circuits:
  - Combinational Logic (CL) circuits
    - Output is a function of the inputs only, not the history of its execution
    - E.g., circuits to add A, B (ALUs)
    - Last lecture was CL
  - Sequential Logic (SL)
    - Circuits that "remember" or store information
    - aka "State Elements"
    - E.g., memories and registers (Registers)
    - Today's lecture is SL

Design Hierarchy



A Conceptual MIPS Datapath

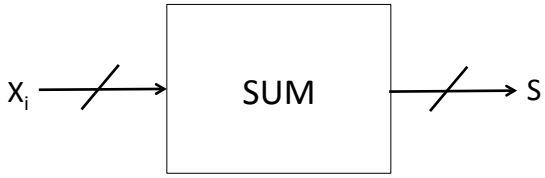


Uses for State Elements

- Place to store values for later re-use:
  - Register files (like $1-$31 on the MIPS)
  - Memory (caches, and main memory)
- *Help control flow of information between combinational logic blocks*
  - State elements hold up the movement of information at input to combinational logic blocks to allow for orderly passage

Accumulator Example

Why do we need to control the flow of information?



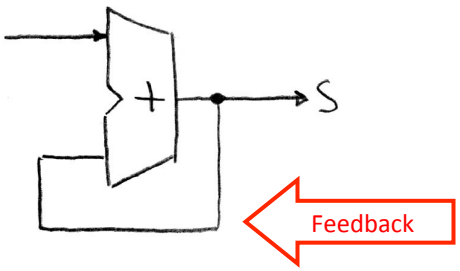
Want:

```
S=0;
for (i=0;i<n;i++)
   S = S + Xi
```

Assume:

- Each X value is applied in succession, one per cycle
- After n cycles the sum is present on S

First Try: Does this work?

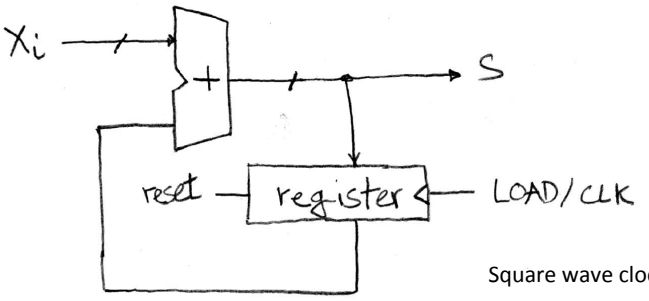


No!

Reason #1: How to control the next iteration of the 'for' loop?

Reason #2: How do we say: 'S=0'?

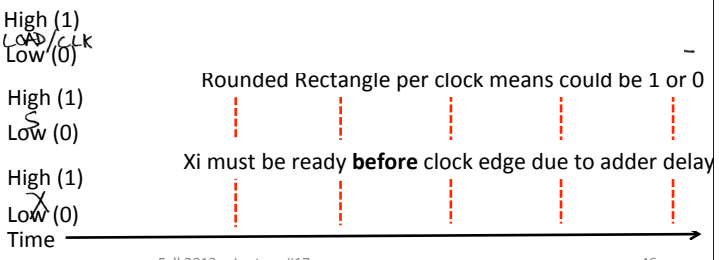
Second Try: How About This?



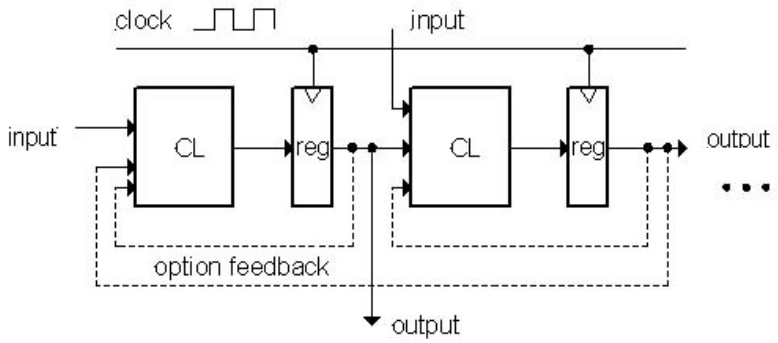
Register is used to hold up the transfer of data to adder

Square wave clock sets when things change

Rough timing …

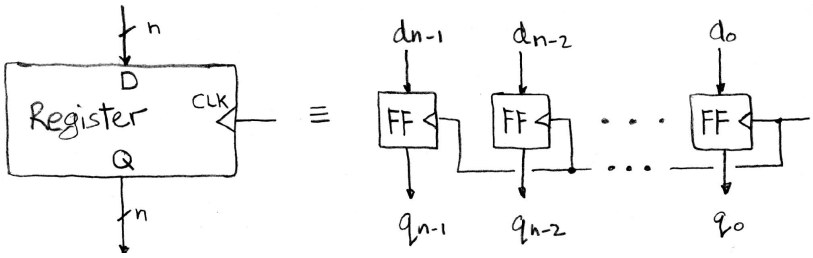


Model for Synchronous Systems



- Collection of Combinational Logic blocks separated by registers
- Feedback is optional
- Clock signal(s) connects only to clock input of registers
- Clock (CLK): steady square wave that synchronizes the system
- Register: several bits of state that samples on rising edge of CLK (positive edge-triggered) or falling edge (negative edge-triggered)

Register Internals



- n instances of a "Flip-Flop"
- Flip-flop name because the output flips and flops between 0 and 1
- D is "data input", Q is "data output"
- Also called "D-type Flip-Flop"

Camera Analogy Timing Terms

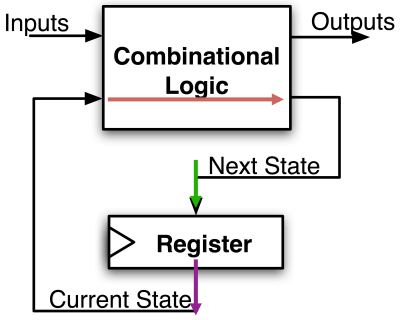
- Want to take a portrait – timing right before and after taking picture
- *Set up time* – don't move since about to take picture (open camera shutter)
- *Hold time* – need to hold still after shutter opens until camera shutter closes
- *Time click to data* – time from open shutter until can see image on output (viewfinder)

Hardware Timing Terms

- Setup Time: when the input must be stable *before* the edge of the CLK
- Hold Time: when the input must be stable *after* the edge of the CLK
- "CLK-to-Q" Delay: how long it takes the output to change, measured from the edge of the CLK

FSM Maximum Clock Frequency

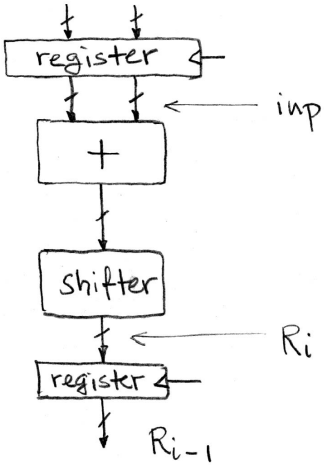
- What is the maximum frequency of this circuit?



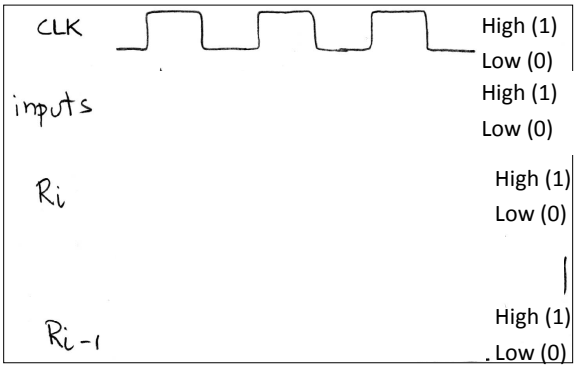
Hint:
Frequency = 1/Period

Max Delay = Setup Time + CLK-to-Q Delay + CL Delay

Pipelining to Improve Performance: BEFORE (1/2)



Timing…

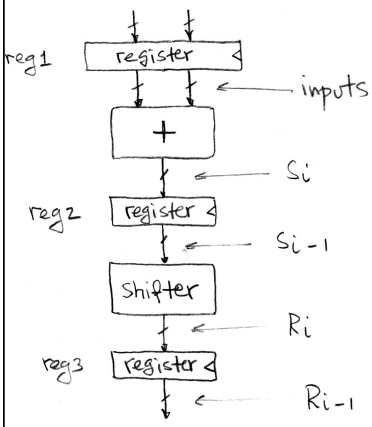


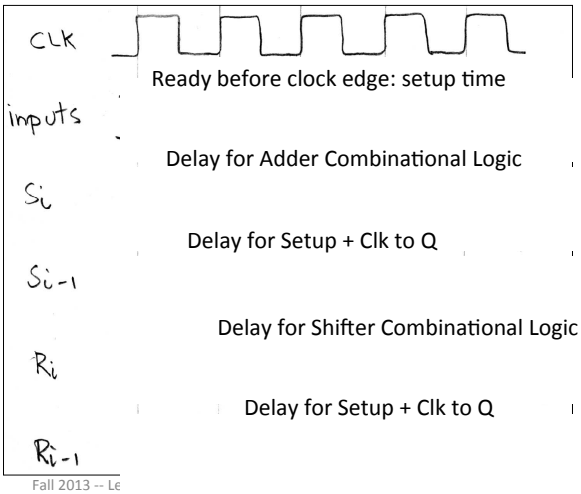
Note: delay of 1 clock cycle from input to output.
Clock period limited by propagation delay of adder/shifter

Pipelining to Improve Performance (2/2)

- Insertion of register allows higher clock frequency
- More outputs per second

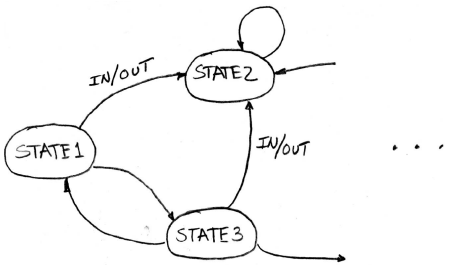
Timing…





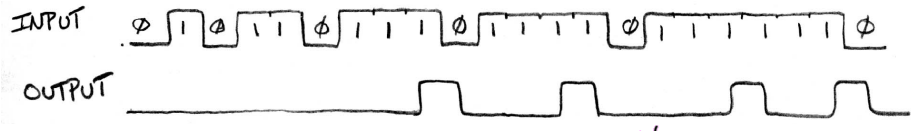
Another Great (Theory) Idea: Finite State Machines (FSM)

- You may have seen FSMs in other classes (e.g., CS70)
- Same basic idea
- Function can be represented with a "state transition diagram"
- With combinational logic and registers, any FSM can be implemented in hardware

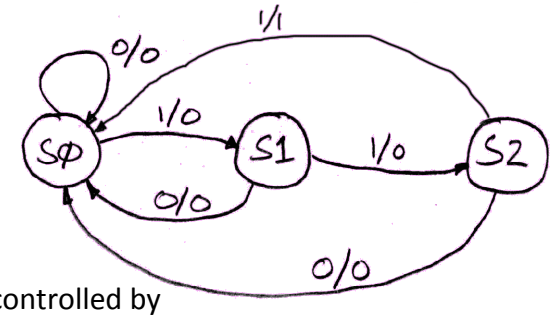


Example: 3 Ones FSM

FSM to detect the occurrence of 3 consecutive 1's in the Input



Draw the FSM …



Assume state transitions are controlled by the clock: On each clock cycle the machine checks the inputs and moves to a new state and produces a new output …

Hardware Implementation of FSM

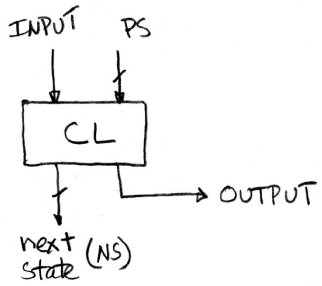
Register needed to hold a representation of the machine's state.
Unique bit pattern for each state.

Combinational logic circuit is used to implement a function maps from *present state (PS)* and *input* to *next state (NS)* and *output*.

The register is used to break the feedback path between Next State (NS) and Prior State (PS), controlled by the clock

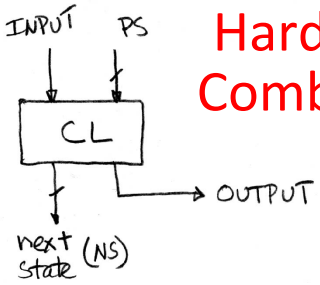
Hardware for FSM: Combinational Logic

Can look at its functional specification, truth table form



Truth table …

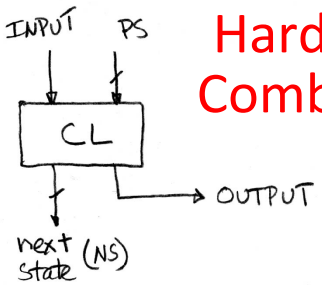
PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1



Hardware for FSM: Combinational Logic

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

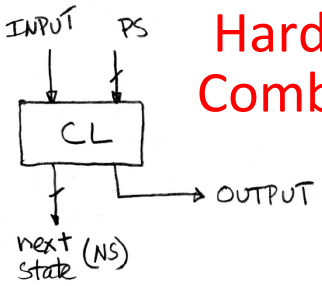


Hardware for FSM: Combinational Logic

Alternative Truth Table format: list only cases where value is a 1.Then restate as logic equations using PS1, PS0, Input

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1



Hardware for FSM: Combinational Logic

Alternative Truth Table format: list only cases where value is a 1.Then restate as logic equations using PS1, PS0, Input

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

NS bit 0 is 1

PS	Input
00	1

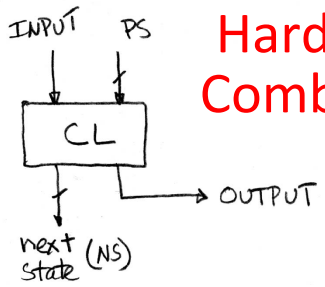
NS bit 1 is 1

PS	Input
01	1

Output is 1

PS	Input
10	1

Hardware for FSM: Combinational Logic



Alternative Truth Table format: list only cases where value is a 1.Then restate as logic equations using PS1, PS0, Input

Truth table …

PS	Input	NS	Output
00	0	00	0
00	1	01	0
01	0	00	0
01	1	10	0
10	0	00	0
10	1	00	1

- NS0 = $\overline{PS1} \bullet \overline{PS0} \bullet Input$
  - NS0 = ~PS1•~PS0•Input

NS bit 0 is 1

PS	Input
00	1

- NS1 = $\overline{PS1} \bullet PS0 \bullet Input$
  - NS1 = ~PS1•PS0•Input

NS bit 1 is 1

PS	Input
01	1

- Output= $PS1 \bullet \overline{PS0} \bullet Input$
  - Output= PS1•~PS0•Input

Output is 1

PS	Input
10	1

And in Conclusion, …

- Multiple Hardware Representations
  - Analog voltages quantized to represent logic 0 and logic 1
  - Transistor switches form gates: AND, OR, NOT, NAND, NOR
  - Truth table mapped to gates for combinational logic design
  - Boolean algebra for gate minimization
- State Machines
  - Finite State Machines: made from *Stateless* combinational logic and *Stateful* "Memory" Logic (aka Registers)
  - Clocks synchronize D-FF change (Setup and Hold times important!)
  - Pipeline long-delay CL for faster clock cycle— Split the *critical path*