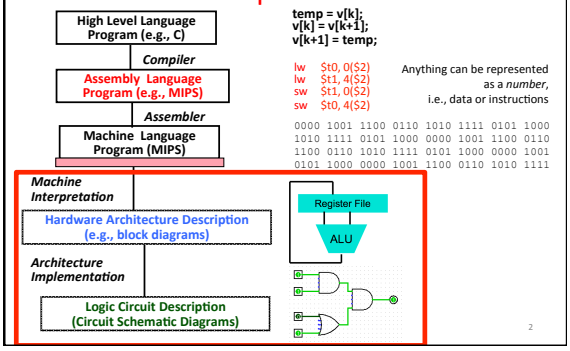
CS 61C:
Great Ideas in Computer Architecture
Finite State Machines

Instructors:
Krste Asanovic & Vladimir Stojanovic
http://inst.eecs.berkeley.edu/~cs61c/sp15

Levels of Representation/ Interpretation



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)
  - Sequential Logic (SL)
    - Circuits that "remember" or store information
    - aka "State Elements"
    - E.g., memories and registers (Registers)

Uses for State Elements

- Place to store values for later re-use:
  - Register files (like $1-$31 in 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

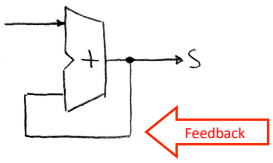
Accumulator Example

Why do we need to control the flow of information?

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

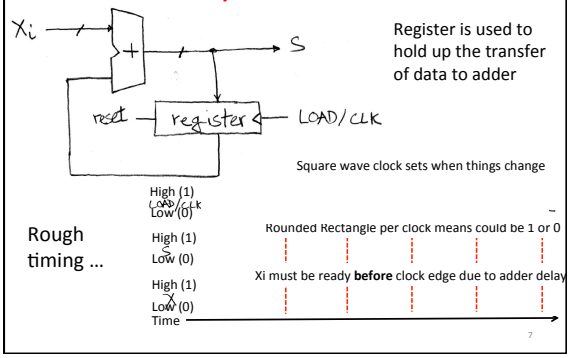
Assume:
- Each X value is applied in succession, one per cycle
- After n cycles the sum is present on S

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?

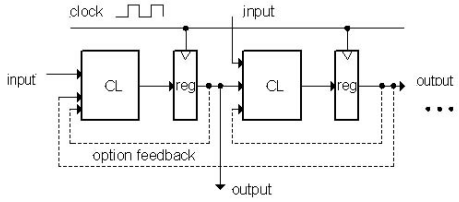


Register is used to hold up the transfer of data to adder

Square wave clock sets when things change

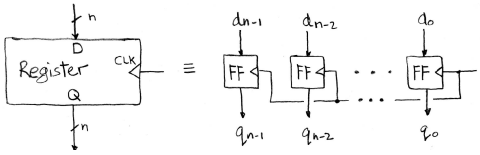
Rough timing …

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)

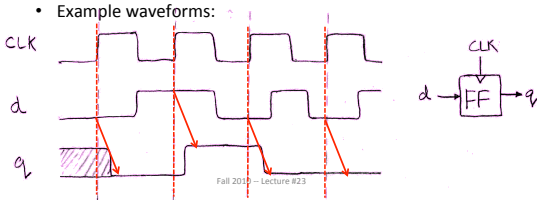
Register Internals



- 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"

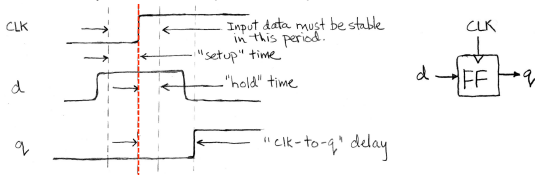
Flip-Flop Operation

- Edge-triggered d-type flip-flop
  - This one is "positive edge-triggered"
- "On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored."
- Example waveforms:



Flip-Flop Timing

- Edge-triggered d-type flip-flop
  - This one is "positive edge-triggered"
- "On the rising edge of the clock, the input d is sampled and transferred to the output. At all other times, the input d is ignored."
- Example waveforms (more detail):



Camera Analogy Timing Terms

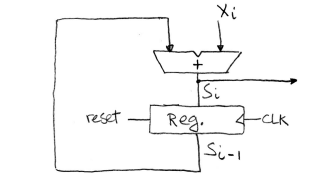
- Want to take a portrait – timing right before and after taking picture
- *Set up time* – don't move since about to take picture (open camera shutter)
- *Hold time* – need to hold still after shutter opens until camera shutter closes
- *Time click to data* – time from open shutter until can see image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

Hardware Timing Terms

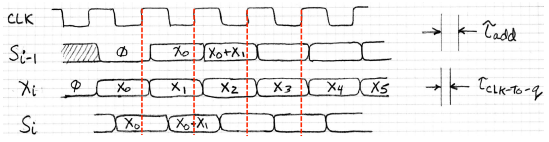
- Setup Time: when the input must be stable *before* the edge of the CLK
- Hold Time: when the input must be stable *after* the edge of the CLK
- "CLK-to-Q" Delay: how long it takes the output to change, measured from the edge of the CLK

13

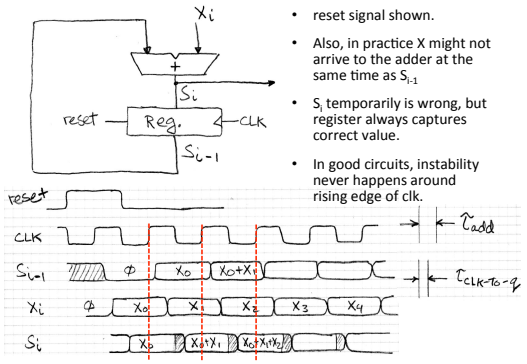
Accumulator Timing 1/2



- Reset input to register is used to force it to all zeros (takes priority over D input).
- S_{i-1} holds the result of the i^{th}-1 iteration.
- Analyze circuit timing starting at the output of the register.



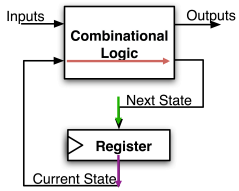
Accumulator Timing 2/2



- reset signal shown.
- Also, in practice X might not arrive to the adder at the same time as S_{i-1}
- S_i temporarily is wrong, but register always captures correct value.
- In good circuits, instability never happens around rising edge of clk.

Maximum Clock Frequency

- What is the maximum frequency of this circuit?

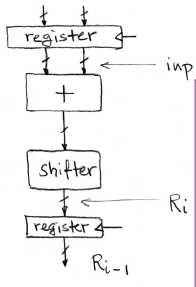


Hint:
Frequency = 1/Period

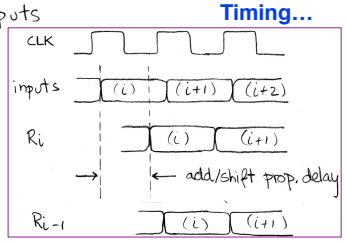
Max Delay = CLK-to-Q Delay + CL Delay + Setup Time

16

Critical Paths



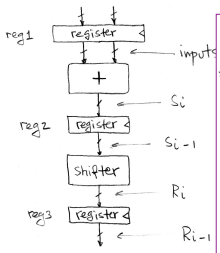
Timing…

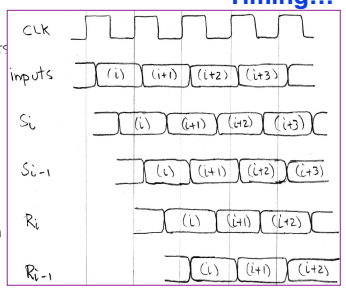


Note: delay of 1 clock cycle from input to output.
Clock period limited by propagation delay of adder/shifter.

Pipelining to improve performance

Timing…



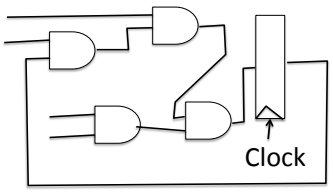


- Insertion of register allows higher clock frequency.
- More outputs per second (higher bandwidth)
- But each individual result takes longer (greater latency)

Recap of Timing Terms

- Clock (CLK) - steady square wave that synchronizes system
- Setup Time - when the input must be stable before the rising edge of the CLK
- Hold Time - when the input must be stable after the rising edge of the CLK
- "CLK-to-Q" Delay - how long it takes the output to change, measured from the rising edge of the CLK
- Flip-flop - one bit of state that samples every rising edge of the CLK (positive edge-triggered)
- Register - several bits of state that samples on rising edge of CLK or on LOAD (positive edge-triggered)

Clickers/Peer Instruction



Clock->Q 1ns
Setup 1ns
Hold 1ns
AND delay 1ns

What is maximum clock frequency?

- A: 5 GHz
- B: 200 MHz
- C: 500 MHz
- D: 1/7 GHz
- E: 1/6 GHz

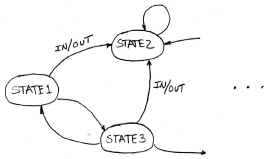
Administrivia

- Project 1-1 due 3/01
- Midterm is next Thursday 2/26, in class
  - Covers up to and including the previous lecture
  - 1 handwritten, double sided, 8.5"x11" cheat sheet
  - We'll give you MIPS green sheet
- Review Sessions:
  - TA: Monday 2/23, 7-9pm, 10 Evans
  - HKN: Saturday 2/21, 1-4pm, 100 Genetics Plant Biology

21

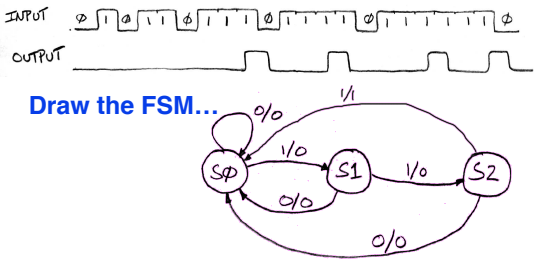
Finite State Machines (FSM) Intro

- You have seen FSMs in other classes.
- Same basic idea.
- The function can be represented with a "state transition diagram".
- With combinational logic and registers, any FSM can be implemented in hardware.



FSM Example: 3 ones…

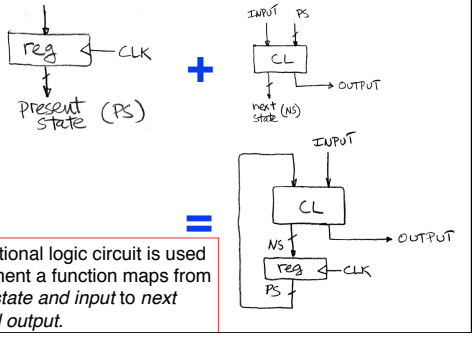
FSM to detect the occurrence of 3 consecutive 1's in the input.



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

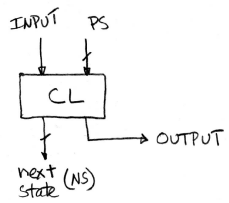
… Therefore a register is needed to hold the a representation of which state the machine is in. Use a unique bit pattern for each state.



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

FSM Combinational Logic

Specify CL using a truth table.

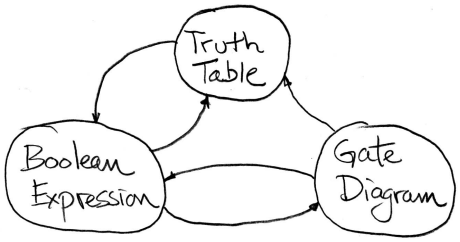


Truth table…

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

Moving between Representations

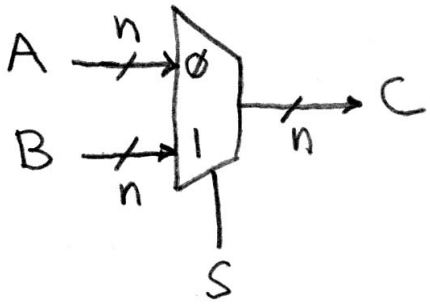
- Use this table and techniques we learned last time to transform between alternative views of same logic function



Building Standard Functional Units

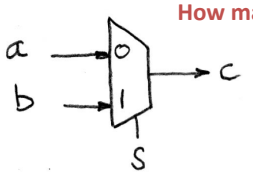
- Data multiplexers
- Arithmetic and Logic Unit
- Adder/Subtractor

Data Multiplexer ("Mux") (here 2-to-1, n-bit-wide)



N instances of 1-bit-wide mux

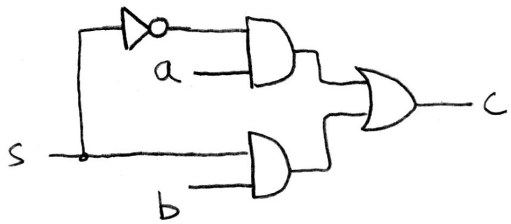
How many rows in TT?



$$
\begin{aligned}
c &= \bar{s}a\bar{b} + \bar{s}ab + s\bar{a}b + sab \\
&= \bar{s}(a\bar{b} + ab) + s(\bar{a}b + ab) \\
&= \bar{s}(a(\bar{b} + b)) + s((\bar{a} + a)b) \\
&= \bar{s}(a(1)) + s((1)b) \\
&= \bar{s}a + sb
\end{aligned}
$$

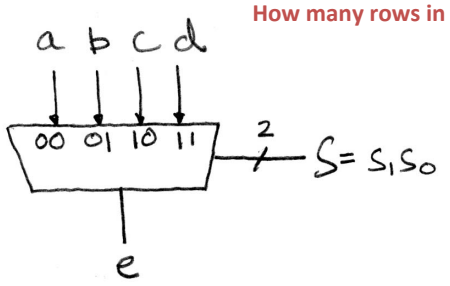
How do we build a 1-bit-wide mux?

$$\bar{s}a + sb$$



4-to-1 multiplexer?

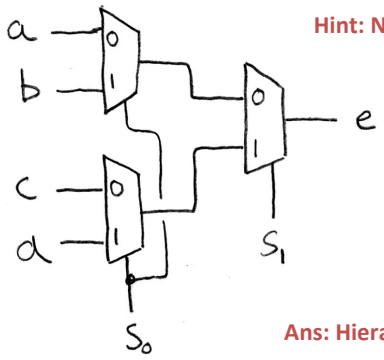
How many rows in TT?



$$e = \overline{s_1}\,\overline{s_0}a + \overline{s_1}s_0b + s_1\overline{s_0}c + s_1s_0d$$

Another way to build 4-1 mux?

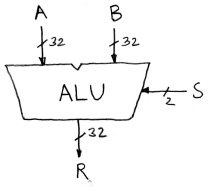
Hint: NCAA tourney!



Ans: Hierarchically!

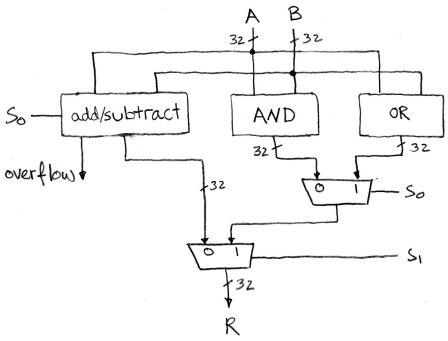
Arithmetic and Logic Unit

- Most processors contain a special logic block called the "Arithmetic and Logic Unit" (ALU)
- We'll show you an easy one that does ADD, SUB, bitwise AND, bitwise OR

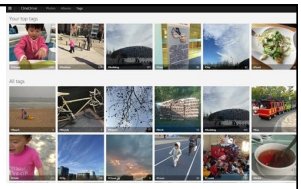


when S=00, R=A+B
when S=01, R=A-B
when S=10, R=A AND B
when S=11, R=A OR B

Our simple ALU



In the News: Microsoft, Google beat Humans at Image Recognition (EE Times)



- On ImageNet benchmark image database, systems from Microsoft and Google performed better than humans at recognizing images
- Both companies used deep artificial neural networks to train on image database

35

How to design Adder/Subtractor?

- Truth-table, then determine canonical form, then minimize and implement as we've seen before
- Look at breaking the problem down into smaller pieces that we can cascade or hierarchically layer

Adder/Subtractor – One-bit adder LSB…

	a_3	a_2	a_1	a_0
+	b_3	b_2	b_1	b_0
	s_3	s_2	s_1	s_0

a_0	b_0	s_0	c_1
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

$s_0 =$

$c_1 =$

Adder/Subtractor – One-bit adder (1/2)…

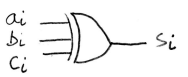
	a_3	a_2	a_1	a_0
+	b_3	b_2	b_1	b_0
	s_3	s_2	s_1	s_0

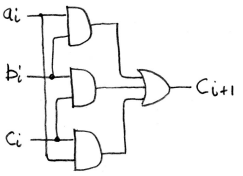
a_i	b_i	c_i	s_i	c_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$s_i =$

$c_{i+1} =$

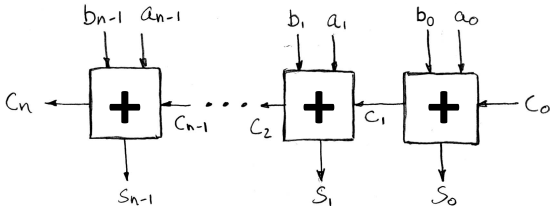
Adder/Subtractor – One-bit adder (2/2)





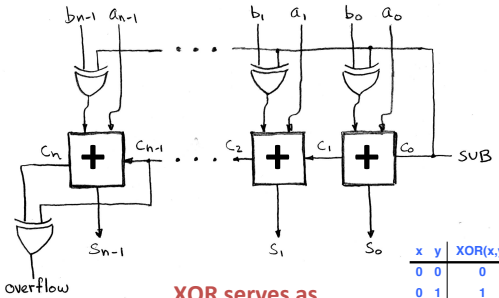
$$s_i = \text{XOR}(a_i, b_i, c_i)$$
$$c_{i+1} = \text{MAJ}(a_i, b_i, c_i) = a_i b_i + a_i c_i + b_i c_i$$

N 1-bit adders ⇒ 1 N-bit adder



What about overflow?
Overflow = c_n?

Extremely Clever Subtractor



XOR serves as conditional inverter!

x	y	XOR(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

In Conclusion

- Finite State Machines have clocked state elements plus combinational logic to describe transition between states
- Standard combinational functional unit blocks built hierarchically from subcomponents

42