State

Hakim Weatherspoon
CS 3410
Computer Science
Cornell University

The slides are the product of many rounds of teaching CS 3410 by Professors Weatherspoon, Bala, Bracy, and Sirer.
Announcements

Make sure you are

• Registered for class, can access CMS
• Have a Section you can go to.
• Lab Sections are required.
  • “Make up” lab sections only Friday 11:40am or 1:25pm
  • Bring laptop to Labs
• Project partners are required for projects starting w/ project 2
  • Have project partner in same Lab Section, if possible
  • WICC hosting a partner finding event Feb 12 @ 6pm in 3rd floor lounge of Gates
Announcements

Make sure to go to your Lab Section this week

Completed **Proj1** due **before** winter break, Friday, Feb 16th

Note, a Design Document is due when you submit Proj1 final circuit

Work **alone**

Work alone, **BUT** use your resources

- Lab Section, Piazza.com, Office Hours
- Class notes, book, Sections, CSUGLab
Announcements

Check online syllabus/schedule
  • http://www.cs.cornell.edu/Courses/CS3410/2018sp/schedule
  • Slides and Reading for lectures
  • Office Hours
  • **Pictures of all TAs**
  • Project and Reading Assignments
  • **Dates to keep in Mind**
    • Prelims: Thur Mar 15th and Thur May 3rd
    • **Proj 1: Due next Friday, Feb 16th before Winter break**
    • Proj3: Due before Spring break
    • Final Project: May 15th

Schedule is subject to change
Collaboration, Late, Re-grading Policies

“White Board” Collaboration Policy
- Can discuss approach together on a “white board”
- Leave, watch a movie (e.g. Strange Things), and write up solution independently
- Do not copy solutions

Late Policy
- Each person has a total of four “slip days”
- Max of two slip days for any individual assignment
- Slip days deducted first for any late assignment, cannot selectively apply slip days
- For projects, slip days are deducted from all partners
- 25% deducted per day late after slip days are exhausted

Regrade policy
- Submit written regrade request within a week of receiving score
A Single cycle processor
Review

- We can generalize 1-bit Full Adders to 32 bits, 64 bits ...
Review

- We can generalize 1-bit Full Adders to 32 bits, 64 bits ...
Example: A Calculator

A → 8
B → 8
S → 0=add, 1=sub
Example: A Calculator

0 = add
1 = sub
Example: A Calculator

128 A
0b1000 0000

128 B
0b1000 0000

S

0 = add
1 = sub

0b1000 0000

adder

decoder
Example: A Calculator

- 128 A
  0b1000 0000

- 128 B
  0b1000 0000

- S

0 = add
1 = sub

adder

decoder
Review: Efficiency and Generality

- We can generalize 1-bit Full Adders to 32 bits, 64 bits ...
- How long does it take to compute a result?
We can generalize 1-bit Full Adders to 32 bits, 64 bits ...

How long does it take to compute a result?

- A) 2 ns
- B) 2 gate delays
- C) 10 ns
- D) 10 gate delays
- E) 8 gate delays
Review: Efficiency and Generality

- We can generalize 1-bit Full Adders to 32 bits, 64 bits ...
- How long does it take to compute a result?
Review: Efficiency and Generality

- We can generalize 1-bit Full Adders to 32 bits, 64 bits ...
- How long does it take to compute a result?
- Can we store the result?

<table>
<thead>
<tr>
<th>t=10</th>
<th>t=8</th>
<th>t=6</th>
<th>t=4</th>
<th>t=2</th>
<th>t=0</th>
</tr>
</thead>
<tbody>
<tr>
<td>S3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Speed of a circuit is affected by the number of gates in series (on the *critical path* or the *deepest level of logic*).

![Diagram of combinational logic](image)
4-bit Ripple Carry Adder

- First full adder, 2 gate delay
- Second full adder, 2 gate delay
- ...
Stateful Components

Until now is **combinational** logic

- Output is computed when inputs are present
- System has no internal state
- Nothing computed in the present can depend on what happened in the past!

Need a way to record data
Need a way to build **stateful** circuits
Need a state-holding device
Goals for Today

State

• How do we store one bit?
  • Attempts at storing (and changing) one bit
    – Set-Reset Latch
    – D Latch
    – D Flip-Flops
    – Master-Slave Flip-Flops
  • Register: storing more than one bit, N-bits

Basic Building Blocks

• Decoders and Encoders
How do we store one bit?
First Attempt: Unstable Devices
First Attempt: Unstable Devices

Does not work!

- Unstable
- Oscillates wildly!
Second Attempt: Bistable Devices

- Stable and unstable equilibria?

![A Simple Device](image)

In stable state, $\overline{A} = B$

How do we change the state?
Third Attempt: Set-Reset Latch
Third Attempt: Set-Reset Latch

Set-Reset (S-R) Latch
Stores a value Q and its complement
**Third Attempt: Set-Reset Latch**

Set-Reset (S-R) Latch

Stores a value $Q$ and its complement

<table>
<thead>
<tr>
<th>$S$</th>
<th>$R$</th>
<th>$Q$</th>
<th>$\overline{Q}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$Q$ will be 0 if $R$ is 1

$\overline{Q}$ will be 1

### Truth Table

<table>
<thead>
<tr>
<th>$A$</th>
<th>$B$</th>
<th>$A \lor B$</th>
<th>$\overline{A \lor B}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$\overline{Q}$ will be 1
Third Attempt: Set-Reset Latch

Set-Reset (S-R) Latch
Stores a value Q and its complement

\[
\begin{array}{c|c|c|c}
S & R & Q & \overline{Q} \\
0 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 \\
\end{array}
\]

\[Q \text{ will be } 1\]
\[\overline{Q} \text{ will be } 0 \text{ if } S \text{ is } 1\]

What are the values for Q and \(\overline{Q}\)?

- a) 0 and 0
- b) 0 and 1
- c) 1 and 0
- d) 1 and 1

iClicker Question
**Third Attempt: Set-Reset Latch**

Set-Reset (S-R) Latch

Stores a value \( Q \) and its complement

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>( \bar{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

If \( Q \) is 1, will stay 1
If \( Q \) is 0, will stay 0

If \( \bar{Q} \) is 0 will stay 0
If \( \bar{Q} \) is 1 will stay 1

\[ A \quad B \quad \text{OR} \quad \text{NOR} \]
| 0 0 0 1 |
| 0 1 1 0 |
| 1 0 1 0 |
| 1 1 1 0 |
Third Attempt: Set-Reset Latch

Set-Reset (S-R) Latch

Stores a value $Q$ and its complement

$Q$ will be 0 since $R$ is 1

$\bar{Q}$ will be 0 since $S$ is 1

What happens when $S,R$ changes from 1,1 to 0,0?
What’s wrong with the SR Latch?

A. Q is undefined when S=1 and R=1
   (That’s why this is called the forbidden state.)
B. Q oscillates between 0 and 1 when the inputs transition from 1,1 \(\rightarrow\) 0,0
C. The SR Latch is problematic b/c it has two outputs to store a single bit.
D. There is nothing wrong with the SR Latch!
Third Attempt: Set-Reset Latch

Set-Reset (S-R) Latch
Stores a value Q and its complement

<table>
<thead>
<tr>
<th></th>
<th>R</th>
<th>Q</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>Q</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>forbidden</td>
<td></td>
</tr>
</tbody>
</table>

What happens when S,R changes from 1,1 to 0,0?

Q and \( \overline{Q} \) become unstable and will oscillate wildly between values 0,0 to 1,1 to 0,0 to 1,1 ...
Third Attempt: Set-Reset Latch

Set-Reset (S-R) Latch
Stores a value \( Q \) and its complement

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>( \overline{Q} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>( \overline{Q} )</td>
</tr>
<tr>
<td></td>
<td></td>
<td>hold</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>reset</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>set</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>forbidden</td>
<td></td>
</tr>
</tbody>
</table>
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.
Next Goal
How do we avoid the forbidden state of S-R Latch?
Fourth Attempt: (Unclocked) D Latch

Fill in the truth table?

<table>
<thead>
<tr>
<th>D</th>
<th>Q</th>
<th>\overline{Q}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>OR</th>
<th>NOR</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Fourth Attempt: (Unclocked) D Latch

Fill in the truth table?

Data (D) Latch
- Easier to use than an SR latch
- No possibility of entering an undefined state

When D changes, Q changes
- ... immediately (...after a delay of 2 Ors and 2 NOTs)

Need to control when the output changes
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding the forbidden state.
Next Goal
How do we coordinate state changes to a D Latch?
Aside: Clocks

Clock helps coordinate state changes

- Usually generated by an oscillating crystal
- Fixed period
- Frequency = 1/period
Clock Disciplines

Level sensitive

• State changes when clock is high (or low)

Edge triggered

• State changes at clock edge
  
  positive edge-triggered

  negative edge-triggered
Clock Methodology

• Negative edge, synchronous

Edge-Triggered → signals must be stable near falling edge

“near” = before and after

t\text{setup} \quad t\text{hold}
Round 2: D Latch (1)

- Inverter prevents SR Latch from entering 1,1 state

<table>
<thead>
<tr>
<th></th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Reset**

**Set**
Round 2: D Latch (1)

Inverter prevents SR Latch from entering 1,1 state

<table>
<thead>
<tr>
<th>D</th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Reset

Set
Round 2: D Latch (1)

- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

C = 1, D Latch *transparent*: set/reset (according to D)
C = 0, D Latch *opaque*: keep state (ignore D)

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q</th>
<th>(\bar{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

No Change
Reset
Set
Round 2: D Latch (1)

C = 1, D Latch *transparent*: set/reset (according to D)

C = 0, D Latch *opaque*: keep state (ignore D)

- Level sensitive
- Inverter prevents SR Latch from entering 1,1 state
- C enables changes

<table>
<thead>
<tr>
<th>S</th>
<th>R</th>
<th>Q</th>
<th>(\bar{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>(\bar{Q}) hold</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1 reset</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0 set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>forbidden</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>C</th>
<th>D</th>
<th>Q</th>
<th>(\bar{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>(\bar{Q})</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>(\bar{Q})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1 set</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0 reset</td>
</tr>
</tbody>
</table>

No Change

Reset

Set
What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1
What is the value of Q at A & B?

a) $A = 0, B = 0$

b) $A = 0, B = 1$

c) $A = 1, B = 0$

d) $A = 1, B = 1$
Level Sensitive D Latch

Clock high:
set/reset (according to D)
Clock low:
keep state (ignore D)

<table>
<thead>
<tr>
<th>clk</th>
<th>D</th>
<th>Q</th>
<th>(\overline{Q})</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q</td>
<td>(\overline{Q})</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Q</td>
<td>(\overline{Q})</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Round 3: D Flip-Flop

- Edge-Triggered
- Data captured when clock high
- Output changes only on falling edges
Round 3: D Flip-Flop

Clock = 1: L1 transparent, L2 opaque

When CLK rises (0→1), now X can change, Q does not change

Clock = 0: L1 opaque, L2 transparent

When CLK falls (1→0), Q gets X, X cannot change
What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1
What is the value of Q at A & B?

a) A = 0, B = 0
b) A = 0, B = 1
c) A = 1, B = 0
d) A = 1, B = 1
Edge-Triggered D Flip-Flop

- Edge-Triggered
- Data captured when clock is high
- Output changes only on falling edges
**Edge-Triggered D Flip-Flop**

- **Edge-Triggered**
- **Data captured when clock is high**
- **Output changes only on falling edges**

![Diagram of D Flip-Flop with clock inputs and output transitions](image)
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flop (aka Master-Slave D Flip-Flop) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.
Next Goal

How do we store more than one bit, $N$ bits?
Registers

Register

• D flip-flops in parallel
• shared clock
• extra clocked inputs: write_enable, reset, ...

 clk

4-bit reg

 D0

 D1

 D2

 D3

 clk

4
Set-Reset (SR) Latch can store one bit and we can change the value of the stored bit. But, SR Latch has a forbidden state.

(Unclocked) D Latch can store and change a bit like an SR Latch while avoiding a forbidden state.

An Edge-Triggered D Flip-Flop (aka Master-Slave D Flip-Flop) stores one bit. The bit can be changed in a synchronized fashion on the edge of a clock signal.

An \( N \)-bit register stores \( N \)-bits. It is created with \( N \) D-Flip-Flops in parallel along with a shared clock.
An Example: What will this circuit do?
An Example: What will this circuit do?

Reset
Run

4-bit reg

WE R

Clk

Decoder


Cout +1

16

+1

S[4]

4
7-Segment LED

- photons emitted when electrons fall into holes
7-Segment LED

- photons emitted when electrons fall into holes

Decoder Example: 7-Segment LED

\[
\begin{array}{cccccccc}
\text{s} & \text{d0} & \text{d1} & \text{d2} & \text{d3} & \text{d4} & \text{d5} & \text{d6} & \text{d7} \\
5 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1
\end{array}
\]
Decoder Example: 7-Segment LED Decoder

3 inputs
• encode 0 – 7 in binary

7 outputs
• one for each LED
### 7 Segment LED Decoder Implementation

<table>
<thead>
<tr>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
### 7 Segment LED Decoder Implementation

<table>
<thead>
<tr>
<th>b2</th>
<th>b1</th>
<th>b0</th>
<th>d6</th>
<th>d5</th>
<th>d4</th>
<th>d3</th>
<th>d2</th>
<th>d1</th>
<th>d0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

![Image of a 7-segment LED display with labels d0 to d6]
Basic Building Blocks We have Seen

- Binary Encoder
  - $2^N \downarrow N \downarrow N$

- Binary Decoder
  - $N \downarrow 2^N$

- Multiplexor
  - $N \rightarrow 0, 1, 2, \ldots, 2^M - 1$

- Encoders

- N Input wires:

- Log₂(N) outputs wires

- **E.g. Voting:**
  Can only vote for one out of N candidates, so N inputs.

- But can encode vote efficiently with binary encoding.
Example Encoder Truth Table

A 3-bit encoder with 4 inputs for simplicity.
A 3-bit encoder with 4 inputs for simplicity

- $o_2 = \overline{abcd}$
- $o_1 = \overline{abcd} + \overline{abcd}$
- $o_0 = \overline{abcd} + \overline{abcd}$
Basic Building Blocks Example: Voting

Ballots

The 3410 optical scan vote reader machine
Basic Building Blocks We have Seen

![Diagram of binary encoder and decoder](image)
Recap

We can now build interesting devices with sensors

• Using combinaionial logic

We can also store data values (aka Sequential Logic)

• In state-holding elements
• Coupled with clocks
Summary

We can now build interesting devices with sensors

- Using combinational logic

We can also store data values

- Stateful circuit elements (D Flip Flops, Registers, ...)
- Clock to synchronize state changes