Basics of Logic Design: Boolean Algebra, Logic Gates

Computer Science 250
Admin

• Midterm 1:
  – Coming soon!

• Homework:
  – Homework 2: pushed back to Monday

• Reading
  – Appendix C (parts 1, 2, 3, 5, 7, 8, 10, 11)
What We’ve Done, Where We’re Going

Top Down

Software

Interface Between HW and SW

Instruction Set Architecture, Memory, I/O

Hardware

(CAlmost) Bottom UP to CPU
Fundamental Law of CS Revisited

• Everything is a number
  – If it’s a number, you can compute with it
  – If it’s not a number (and you can’t represent it as one), you can’t compute on it

• Numbers are just bits
  – Binary encodings: 0s and 1s

• Computers physically made of transistors
  – Electrically controlled switches
How Many Transistors Are We Talking About?

Pentium III
• Processor Core 9.5 Million Transistors
• Total: 28 Million Transistors

Pentium 4
• Total: 42 Million Transistors

Core2 Duo (two processors)
• Total: 290 Million Transistors

Core2 Duo Extreme (4 processors, 8MB cache)
• Total: 590 Million Transistors

How do they design such a thing?
Abstraction!

• Use of abstraction (key to design of any large system)
  – Put a few (2-8) transistors into a logic gate
  – Combine gates into logical functions (add, select, ….)
  – Combine adders, muxes, etc together into RLMs
    (“Random Logic Macros”)
    Units with well defined interfaces for large tasks: e.g., decode
  – Combine a dozen of those into a core…
  – Stick 4 cores on a chip…
You are here:

• Use of **abstraction** (key to design of any large system)
  – Put a few (2-8) transistors into a **logic gate**
  – Combine gates into logical functions (add, select, …)
  – Combine adders, muxes, etc together into RLMs
    (“Random Logic Macros”)
    Units with well defined interfaces for large tasks: e.g., decode
  – Combine a dozen of those into a core …
  – Stick 4 cores on a chip …
Boolean Algebra

• First step to logic: Boolean Algebra
  – Manipulation of True / False (1/0)
  – After all: everything is just 1s and 0s…

• Have inputs (variables): A, B, C, P, Q…
• Negation (NOT): !A (or ~A or A)
• AND: A & B (or A * B)
• OR: A | B (or A + B)
• XOR: A ^ B
Truth Tables

- Can represent as Truth Table: shows outputs for all inputs

<table>
<thead>
<tr>
<th>a</th>
<th>NOT(a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Truth Tables

- Can represent as Truth Table: shows outputs for all inputs

\[
\begin{array}{c|c}
\text{a} & \text{NOT}(a) \\
\hline
0 & 1 \\
1 & 0 \\
\end{array}
\quad
\begin{array}{c|c|c}
\text{a} & \text{b} & \text{AND}(a,b) \\
\hline
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]
Truth Tables

- Can represent as Truth Table: shows outputs for all inputs

\[
\begin{array}{c|c}
 a & \text{NOT}(a) \\
 0 & 1 \\
 1 & 0 \\
\end{array}
\quad
\begin{array}{c|c}
 a & \text{AND}(a,b) \\
 0 & 0 \\
 0 & 1 \\
 1 & 0 \\
 1 & 1 \\
\end{array}
\quad
\begin{array}{c|c}
 a & \text{OR}(a,b) \\
 0 & 0 \\
 0 & 1 \\
 1 & 0 \\
 1 & 1 \\
\end{array}
\]
Truth Tables

- Can represent as Truth Table: shows outputs for all inputs

<table>
<thead>
<tr>
<th>a</th>
<th>NOT(a)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>AND(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>OR(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>XOR(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>XNOR(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>NOR(a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Any Inputs Any Outputs

- Can have any # of inputs, any # of outputs
- Can have arbitrary functions:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>f₁</th>
<th>f₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Let’s write a Truth Table for a function…

- Example:
  \((A \land B) \lor \neg C\)

Start with Empty TT

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s write a Truth Table for a function…

- Example:
  \((A \& B) \mid !C\)

Start with Empty TT
- Column Per Input
- Column Per Output

Fill in Inputs
- Counting in Binary

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
Let’s write a Truth Table for a function…

• Example:
  
  \[(A \& B) \mid !C\]

Start with Empty TT

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Fill in Inputs

Counting in Binary
Let’s write a Truth Table for a function…

• Example:
  
  \[(A \& B) \mid !C\]

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>Output</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Start with Empty TT
  
  Column Per Input
  
  Column Per Output

Fill in Inputs
  
  Counting in Binary
Let’s write a Truth Table for a function...

• Example:
  \((A \& B) \mid !C\)

Start with Empty TT

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Column Per Input
Column Per Output

Fill in Inputs
Counting in Binary

Compute Output
\((0 \& 0) \mid !0 = 0 \mid 1 = 1\)
Let’s write a Truth Table for a function…

• Example:
  \[(A \land B) \lor \lnot C\]

Start with Empty TT
  Column Per Input
  Column Per Output

Fill in Inputs
  Counting in Binary

Compute Output
  \[(0 \land 0) \lor \lnot 1 = 0 \lor 0 = 0\]
Let’s write a Truth Table for a function…

• Example:
  
  \[(A \& B) \mid !C\]

Start with Empty TT

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

Compute Output

\[(0 \& 1) \mid !0 = 0 \mid 1 = 1\]
Let’s write a Truth Table for a function…

• Example:
  
  \((A \& B) | !C\)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Start with Empty TT

Column Per Input

Column Per Output

Fill in Inputs

Counting in Binary

Compute Output
You try one…

- Try one yourself:
  \((\neg A \lor B) \& \neg C\)
You try one…

• Try one yourself:
  \((!A \lor B) \& !C\)

Answer:

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</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>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Suppose I turn it around…

- Given a Truth Table, find the formula?

Hmmm..
Suppose I turn it around…

• Given a Truth Table, find the formula?

Hmmm..
Could write down every “true” case
Then OR together:

Suppose I turn it around…

• Given a Truth Table, find the formula?

Hmmm..

Could write down every “true” case

Then OR together:

\[ (!A \& !B \& !C) \mid \]
\[ (!A \& !B \& C) \mid \]
\[ (!A \& B \& !C) \mid \]
\[ (A \& B \& !C) \mid \]
\[ (A \& B \& C) \]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</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>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Suppose I turn it around…

• Given a Truth Table, find the formula?

Hmmm..
Could write down every “true” case
Then OR together:

(!A & !B & !C) |  
(!A & !B & C)  |  
(!A & B & !C)  |  
(A & B & !C)  |  
(A & B & C)
Suppose I turn it around…

• This approach: “sum of products”
  – Works every time
  – Result is right…
  – But really ugly

\[
\begin{align*}
(!A \& !B \& !C) \text{ | } \\
(!A \& !B \& C) \text{ | } \\
(!A \& B \& !C) \text{ | } \\
(A \& B \& !C) \text{ | } \\
(A \& B \& C)
\end{align*}
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</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>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Suppose I turn it around…

- This approach: “sum of products”
  - Works every time
  - Result is right…
  - But really ugly

\[
\begin{align*}
(!A & !B & !C) & | \\
(!A & !B & C) & | \\
(!A & B & !C) & | \\
(A & B & !C) & | \\
(A & B & C)
\end{align*}
\]

Could just be (A & B) here?
Suppose I turn it around…

• This approach: “sum of products”
  – Works every time
  – Result is right…
  – But really ugly

\[
\begin{align*}
(!A & !B & !C) \lor \\
(!A & !B & C) \lor \\
(!A & B & !C) \lor \\
(A&B)
\end{align*}
\]
Suppose I turn it around…

• This approach: “sum of products”
  – Works every time
  – Result is right…
  – But really ugly

\[
\begin{align*}
(!A & !B & !C) & \mid \\
(!A & !B & C) & \mid \\
(!A & B & !C) & \mid \\
(A&B) & \\
\end{align*}
\]

Could just be (!A & !B) here
Suppose I turn it around…

- This approach: “sum of products”
  - Works every time
  - Result is right…
  - But really ugly

(\bar{A} \& \bar{B}) \mid

(\bar{A} \& B \& \bar{C}) \mid

(A\&B)

Could just be (\bar{A} \& \bar{B}) here

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</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>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Suppose I turn it around…

- This approach: “sum of products”
  - Works every time
  - Result is right…
  - But really ugly

\[(\neg A \land \neg B) \lor (\neg A \land B \land \neg C) \lor (A\land B)\]

Looks nicer…

Can we do better?
Suppose I turn it around…

- This approach: “sum of products”
  - Works every time
  - Result is right…
  - But really ugly

\[
(\neg A \land \neg B) \lor (\neg A \land B \land \neg C) \lor (A \land B)
\]

This has a lot in common:

\( \neg A \land (\text{something}) \)
Suppose I turn it around…

- This approach: “sum of products”
  - Works every time
  - Result is right…
  - But really ugly

\[(\neg A \& \neg (B \& C)) \mid (A \& B)\]
Just did some of these by intuition.. but

• Somewhat intuitive approach to simplifying
• This is math so there are formal rules
  – Just like “regular” algebra
Boolean Function Simplification

• Boolean expressions can be simplified by using the following rules (bitwise logical):
  
  \[ \begin{align*} 
  & A \land A = A \\
  & A \land 0 = 0 \\
  & A \land 1 = A \\
  & A \land \neg A = 0 \\
  & \neg \neg A = A \\
  \end{align*} \]

\[ \begin{align*} 
  & A \lor A = A \\
  & A \lor 0 = A \\
  & A \lor 1 = 1 \\
  & A \lor \neg A = 1 \\
  \end{align*} \]

  
  \[ \neg \neg A = A \]

  
  \[ \begin{align*} 
  & \text{\& and | are both commutative and associative} \]
  \[ \begin{align*} 
  & \text{\& and | can be distributed: } A \land (B \lor C) = (A \land B) \lor (A \land C) \\
  & \text{\& and | can be subsumed: } A \lor (A \land B) = A \\
  \end{align*} \]
DeMorgan’s Laws

• Two (less obvious) Laws of Boolean Algebra:
  – Let us push negations inside, flipping & and |

\[
\neg (A \land B) = (\neg A) \lor (\neg B)
\]

\[
\neg (A \lor B) = (\neg A) \land (\neg B)
\]
Suppose I turn it around…

• One more simplification on early example:

\[
(!A \ & \ !((B \ & \ C)) \ | \\
(A \ & \ B)
\]

= 

\[
(!A \ & \ (!B \ | \ !C)) \ | \\
(A \ & \ B)
\]
Simplification Example:

! ( !A | ! (A & (B | C)) )
Simplification Example:

! ( !A | ! (A & (B | C)) )

Demorgan’s

!!A & !! (A & (B | C))
Simplification Example:

! (!A | !(A & (B | C)))

Demorgan’s

!!A & !! (A & (B | C))

Double Negation Elimination

A & (A & (B | C))
Simplification Example:

! ( !A | !(A & (B | C)) )

Demorgan’s

!!A & !!(A & (B | C))

Double Negation Elimination

A & (A & (B | C))

Associativity of &

(A & A) & (B | C)
Simplification Example:

\[ \lnot (\lnot A \lor (A \land (B \lor C))) \]

Demorgan’s

\[ \lnot \lnot A \land \lnot \lnot (A \land (B \lor C)) \]

Double Negation Elimination

\[ A \land (A \land (B \lor C)) \]

Associativity of \&

\[ (A \land A) \land (B \lor C) \]

\[ A \land A = A \]

\[ A \land (B \lor C) \]
You try this:

Come up with a formula for this Truth Table
Simplify as much as possible

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
You try this:

Come up with a formula for this Truth Table

Simplify as much as possible

Sum of Products:

\[ (!A \land !B \land !C) \lor
\begin{align*}
( & !A \land B \land !C) \lor
\end{align*}
\]

\[ ( A \land !B \land C) \lor
\begin{align*}
( & A \land B \land C)
\end{align*}
\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
You try this:

Simplify:

\((!A \land !B \land !C) \lor (!A \land B \land !C)\)
You try this:

Simplify:

\((!A \& !B \& !C) \mid (!A \& B \& !C)\)

Regroup (associative/commutative):

\(((!A \& !C) \& !B) \mid ((!A \& !C) \& B)\)
You try this:

Simplify:

(!A & !B & !C) | (!A & B & !C)

Regroup (associative/commutative):

((!A & !C) & !B) | ((!A & !C) & B)

Distribute:

(((!A & !C) & !B) | (!A & !C)) & 

((!A & !C) | B) & (!B | B)

(Applied twice for blue text)
You try this:

Simplify:

\(!A \& !B \& !C) \mid (!A \& B \& !C)\)

Regroup (associative/commutative):

\(((!A \& !C) \& !B) \mid ((!A \& !C) \& B)\)

Distribute:

\((((!A \& !C) \& !B) \mid (!A \& !C)) \&

\(((!A \& !C) \mid B) \& (!B \mid B)\)

Subsume:

\(!A \& !C) \& (!B \mid B)\)
You try this:

Simplify:

\[ (!A \land \neg B \land \neg C) \lor (!A \land B \land \neg C) \]

Regroup (associative/commutative):

\[ ((\neg A \land \neg C) \land \neg B) \lor ((\neg A \land \neg C) \land B) \]

Distribute:

\[ (((\neg A \land \neg C) \land \neg B) \lor (\neg A \land \neg C)) \land ((\neg A \land \neg C) \land B) \land (\neg B \lor B) \]

Subsume:

\[ (\neg A \land \neg C) \land (\neg B \lor B) \]

OR identities:

\[ (\neg A \land \neg C) \land \text{true} = (\neg A \land \neg C) \]
You try this:

Come up with a formula for this Truth Table
Simplify as much as possible

Sum of Products:

(!A & !C) | 
(A & !B & C) | 
(A & B & C)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
You try this:

Come up with a formula for this Truth Table
Simplify as much as possible

Sum of Products:

\[(\neg A \land \neg C) \lor (A \land C)\]
Applying the Theory

• Lots of good theory
• Can reason about complex boolean expressions
• But why is this useful?
Boolean Gates

- **Gates** are electronic devices that implement simple Boolean functions (building block of hardware)

Examples

\[
\begin{align*}
&\text{AND}(a,b) & \quad & \text{OR}(a,b) & \quad & \text{NOT}(a) \\
&\text{XOR}(a,b) & \quad & \text{NAND}(a,b) \\
&\text{NOR}(a,b) & \quad & \text{XNOR}(a,b)
\end{align*}
\]
Drew’s Guide to Remembering your Gates

• This one looks like it just points its input where to go
  – It just produces its input as its output
  – Called a buffer

[Diagram of a buffer]
Drew’s Guide to Remembering your Gates

• This one looks like it just points its input where to go
  – It just produces its input as its output
  – Called a buffer

• A circle always means negate

\[ \text{Circle} = \text{NOT} \]
Drew’s Guide to Remembering Your Gates

- And Gates have a straight edge, like an A (in AND)

```
AND(a, b)
```

![Diagram of an AND gate with inputs a and b and output AND(a, b)]

Straight like an A

- OR Gates have a curved edge, like an O (in OR)

```
OR(a, b)
```

![Diagram of an OR gate with inputs a and b and output OR(a, b)]

Curved, like an O
Drew’s Guide to Remembering Your Gates

- If we stick a circle on them…

- We get NAND (NOT-AND) and NOR (NOT-OR)
  - NAND\((a,b)\) = NOT(AND\((a,b)\))
Drew’s Guide to Remembering Your Gates

• XOR looks like OR (curved line)
  – But has two lines (like an X does)

  ![XOR Circuit Diagram]

• Can put a dot for XNOR
  – XNOR is 1-bit “equals” by the way

  ![XNOR Circuit Diagram]
Boolean Functions, Gates and Circuits

- Circuits are made from a network of gates.

$$( \neg A \land \neg C ) \lor ( A \land C )$$
A few more words about gates

- Gates have inputs and outputs
  - If you try to hook up two outputs, bad things happen
    (your processor catches fire)
  - If you don’t hook up an input, it behaves kind of randomly
    (also not good, but not set-your-chip-on-fire bad)
Let’s Make a Useful Circuit

• Pick between 2 inputs (called 2-to-1 MUX)
  – Short for multiplexor

• What might we do first?
Let’s Make a Useful Circuit

• Pick between 2 inputs (called 2-to-1 MUX)
  – Short for multiplexor

• What might we do first?
  – Make a truth table?
    • S is selector:
      • S=0, pick A
      • S=1, pick B

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Let’s Make a Useful Circuit

• Pick between 2 inputs (called 2-to-1 MUX)
  – Short for multiplexor

• What might we do first?
  – Make a truth table?
    • S is selector:
      • S=0, pick A
      • S=1, pick B

• Next: sum-of-products
  (A & B & S)  
  (A & !B & !S)  
  (A & B & !S)  
  (A & B & S)
Let’s Make a Useful Circuit

• Pick between 2 inputs (called 2-to-1 MUX)
  – Short for multiplexor

• What might we do first?
  – Make a truth table?
    • S is selector:
      • S=0, pick A
      • S=1, pick B

• Next: sum-of-products

• Simplify
  \[(A \ & \ !S) \ | \ (B \ & \ S)\]

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>S</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<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>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</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>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
**Circuit Example: 2x1 MUX**

Draw it in gates:

\[
\text{MUX}(A, B, S) = (A \& \neg S) \lor (B \& S)
\]

So common, we give it its own symbol:
Example 4x1 MUX

The / 2 on the wire means “2 bits”
Binary Math : Addition

• Remember binary addition:

\[
\begin{array}{c}
11 \\
00011101 \\
+ \quad 00101011 \\
\hline
\quad 00
\end{array}
\]

How would we implement this in hardware?
Implementing Addition

• First, one bit addition.
  – Three inputs: Carry In (CI), A, B
  – Two outputs Carry Out (CO), Sum (S)

• Go around room for truth table:

<table>
<thead>
<tr>
<th>CI</th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Implementing Addition

• First, one bit addition.
  – Three inputs: Carry In (CI), A, B
  – Two outputs Carry Out (CO), Sum (S)

• Go around room for truth table:

<table>
<thead>
<tr>
<th>CI</th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Half Adder

• Ignore CI for a second (assume is 0)
  – Can simplify a lot and build “half adder”
    • Formula for S?
    • Formula for CO?

<table>
<thead>
<tr>
<th>Cl</th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Half Adder

• Ignore CI for a second (assume is 0)
  – Can simplify a lot and build “half adder”
  • Formula for S? \( A \text{xor} B \)
  • Formula for CO? \( A \text{ and} B \)

<table>
<thead>
<tr>
<th>CI</th>
<th>A</th>
<th>B</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</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>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Half Adder

- Half adder:
- 1 XOR and 1 AND
- Can anyone guess why its called a half adder?
Implementing Addition

- Re-visit Truth table, but..
  - Use Half-Sum and Half-CO (results of Half-Adder)
- Go around room for truth table:

<table>
<thead>
<tr>
<th>Cl</th>
<th>Half-Sum</th>
<th>Half-CO</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Implementing Addition

- Re-visit Truth table, but..
  - Use Half-Sum and Half-CO (results of Half-Adder)
- Go around room for truth table:

<table>
<thead>
<tr>
<th>CI</th>
<th>Half-Sum</th>
<th>Half-CO</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>!!!</td>
<td>!!!</td>
</tr>
</tbody>
</table>
Implementing Addition

• Re-visit Truth table, but..
  – Use Half-Sum and Half-CO (results of Half-Adder)

• Go around room for truth table:

<table>
<thead>
<tr>
<th>CI</th>
<th>Half-Sum</th>
<th>Half-CO</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<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>
<td>1</td>
</tr>
</tbody>
</table>
### Implementing Addition

- **Formulas:**
  - Sum?
  - CO?

<table>
<thead>
<tr>
<th>CI</th>
<th>Half-Sum</th>
<th>Half-CO</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<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>
<td>1</td>
</tr>
</tbody>
</table>
## Implementing Addition

- **Formulas:**
  - Sum? $\text{CI} \text{ xor} \text{ Half-Sum}$
  - CO? $(\text{CI} \text{ and} \text{ Half-Sum}) \text{ OR} \text{ Half-CO}$

<table>
<thead>
<tr>
<th>CI</th>
<th>Half-Sum</th>
<th>Half-CO</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<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>
<td>1</td>
</tr>
</tbody>
</table>
Implementing Addition

- **Formulas:**
  - Sum? \( CI \) xor Half-Sum
  - CO? \( (CI \text{ and Half-Sum}) \) OR Half-CO

<table>
<thead>
<tr>
<th>CI</th>
<th>Half-Sum</th>
<th>Half-CO</th>
<th>S</th>
<th>CO</th>
</tr>
</thead>
<tbody>
<tr>
<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>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<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>
<td>1</td>
</tr>
</tbody>
</table>
• Full Adder
• 2 Half Adders + an OR Gate
Ripple Carry

- Full Adder = Add 1 Bit
  - Can chain together to add many bits
  - Upside: Simple
  - Downside?
Ripple Carry

- Full Adder = Add 1 Bit
  - Can chain together to add many bits
  - Upside: Simple
  - Downside? Slow
    - Let’s see why
• Cout depends on Cin
  – 2 “gate delays” through full adder for carry
Ripple Carry

- Carries form a chain
  - Need CO of bit N is CI of bit N+1
- For few bits (e.g., 4) no big deal
  - For realistic numbers of bits (e.g., 32, 64), slow
Adding

• Adding is important
  – Ubiquitous: need it to be fast

• Real hardware uses fancier adders
  – Much faster
  – Much more complicated
  – Exploit parallelism
  – We’re not going to go into them
Subtraction

• 2’s complement makes subtraction easy:
  – Remember: A - B = A + (-B)
  – And: -B = ~B + 1
    🔺 that means flip bits ("not")
  – So we just flip the bits and start with CI = 1
  – Fortunate for us: makes circuits easy

•
•
•

1

0110101
→
0110101

- 1010010
+ 0101101
32-bit Adder/subtractor

- Inputs: A, B, Add/Sub (0=Add, 1 = Sub)
- Outputs: Sum, Cout, Ovf (Overflow)
32-bit Adder/subtractor

- By the way:
  - That thing has about 3,000 transistors
  - Aren’t you glad we have abstraction?
Remember Shifting

• Left shift (<<)
  – Moves left, bringing in 0s at right, excess bits “fall off”
  – $10010001 << 2 = 01000100$
  – $x << k$ corresponds to $x \times 2^k$

• Logical (or unsigned) right shift (>>)
  – Moves bits right, bringing in 0s at left, excess bits “fall off”
  – $10010001 >> 3 = 00010010$
  – $x >> k$ corresponds to $x / 2^k$ for unsigned $x$

• Arithmetic (or signed) right shift (>>)
  – Moves bits right, bringing in (sign bit) at left
  – $10010001 >> 3 = 11110010$
  – $x >> k$ corresponds to $x / 2^k$ for signed $x$
Shift: Implementation

• Suppose an 8-bit number
  \[ b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \]
  Shifted left by a 3 bit number
  \[ s_2 s_1 s_0 \]

• Option 1: Truth Table?
  – 2048 rows? Not appealing
Lets simplify

- Simpler problem: 8-bit number shifted by 1 bit number (shift amount selects each mux)
Let's simplify

- Simpler problem: 8-bit number shifted by 2 bit number (new muxes selected by 2\textsuperscript{nd} bit)
Now shifted by 3-bit number

- Full problem: 8-bit number shifted by 3 bit number (new muxes selected by 3rd bit)
Now shifted by 3-bit number

- Shifter in action: shift by 000
Now shifted by 3-bit number

- Shifter in action: shift by 010
Now shifted by 3-bit number

- Shifter in action: shift by 011
ALU: Arithmetic and Logic Unit

ALU can do math/logic
- Adder / Subtractor
- Shifter
- And/Or/Xor/Not (just simple gates)

- Take function code, mux picks output of correct sub-unit
  - MIPS: func code in R-type specifies ALU operation
Summary

• Boolean Algebra & functions
• Logic gates (AND, OR, NOT, etc)
• Multiplexors
• ALU: Add, Subtract, Shift, Logical
• Reading Chapter C (sub-parts listed at start of slides)