Didn’t I learn how to do addition in second grade? UNC courses aren’t what they used to be...

Finally; time to build some serious functional blocks

We’ll need a lot of boxes

• How to add and subtract using combinational logic
• Setting flags
• Adding faster
Review: Binary Representations

- Unsigned numbers, each increasingly significant bit has a weight of the next larger power of 2.
- Signed 2's complement representation the most significant bit is a negative power of 2.

\[ v = \sum_{i=0}^{n-1} 2^i b_i \]

\[ v = -2^{n-1} b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i \]

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 1  | 0  | 1  | 0  | 1  | 1  | 0 |

| 4294967254 | -or- | -42 |

- Why?
  - They are compatible. The same logic can be used for both
  - Only "adders" are needed for both addition and subtraction
Here's an example of binary addition as one might do it by "hand":

\[
\begin{array}{c}
\text{A:} & 1101 \\
\text{B:} & + 0101 \\
\hline
\text{10010}
\end{array}
\]

Let's start by building a block to add one column: This functional block is called a "Full-adder"

Then we can cascade them to add two numbers of any size...
**Design of a “Full Adder”**

1) Start with a truth table:

2) Write down equations for the “i” outputs

\[
\begin{align*}
    \text{CO} &= (!\text{CI} \land \text{A} \land \text{B}) \lor (\text{CI} \land !\text{A} \land \text{B}) \\
    &\quad \lor (\text{CI} \land \text{A} \land !\text{B}) \\
    \text{S} &= (!\text{CI} \land !\text{A} \land \text{B}) \lor (!\text{CI} \land \text{A} \land !\text{B}) \\
    &\quad \lor (\text{CI} \land !\text{A} \land !\text{B}) \\
\end{align*}
\]

\[
\begin{array}{cccc|cc}
\text{C}_i & \text{A} & \text{B} & \text{C}_o & \text{S} \\
\hline
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 \\
0 & 1 & 0 & 0 & 1 \\
0 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 \\
1 & 0 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 \\
\end{array}
\]

3) Simplifying a bit

\[
\begin{align*}
    \text{CO} &= (\text{CI} \land (\text{A} \lor \text{B})) \lor (\text{A} \land \text{B}) \\
    \text{S} &= \text{CI} \land (\text{A} \land \text{B})
\end{align*}
\]
As a Logic Diagram

- Our equations:
  \[ CO = (CI \& (A ^ B)) \mid (A \& B) \]
  \[ S = CI \^ (A ^ B) \]

- A little tricky, but finally
  Only 5 gates/bit
An Aside: Why Full Adder?

Suppose you don’t want/need a carry-in?

Then you get a "half adder" with 2 inputs and 2 outputs:

- Half-adder equations:
  - CO = A & B
  - S = A ^ B

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>CO</th>
<th>S</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
**Subtraction: A - B = A + (-B)**

- Recall the trick was to "complement and add 1"
- How to complement?
  \[ \sim = \text{bitwise complement} \]

- So now a unit that can either add or subtract
**Reverse Subtract: -A + B**

- And with a few more XOR gates we can subtract either the A or the B operands.
Condition Flags

Besides the sum, one often wants four other bits of information from an arithmetic unit, the condition flags.

\( Z \) (zero): result is = 0  \hspace{1cm} \text{big NOR gate}  
\( N \) (negative): result is < 0  
\( C \) (carry): indicates the most significant bit produced a carry, e.g., "1 + (-1)"  \hspace{1cm} \text{CO}_{31} \text{ (of last FA)}  
\( V \) (overflow): indicates an unexpected change in sign e.g., "(2^{30} - 1) + 1"  

Signed comparison:  
\( H = N \oplus V \)  
\( le = Z \lor (N \oplus V) \)  
\( eq = Z \)  
\( ne = \neg Z \)  
\( ge = \neg(N \oplus V) \)  
\( gt = \neg(Z \lor (N \oplus V)) \)

Unsigned comparison:  
\( hi = C \land \neg Z \)  
\( ls = \neg C \lor Z \)  
\( lo = C \) \text{ (same as cc)}  
\( hs = C \) \text{ (same as cs)}
How fast is an Add?

Determined by $T_{pd}$ of the FA chain

Worse-case path: carry propagation from LSB to MSB, e.g., when adding -1 to 1.

$$t_{PD} = (t_{PD,XOR} + t_{PD,AND} + t_{PD,OR}) + (N-2)*(t_{PD,OR} + t_{PD,AND}) + t_{PD,XOR} \approx \Theta(N)$$
**WE CAN ADD “MUCH” FASTER**

Using more gates we can speed up adding considerably if we add 2 “free” extra outputs from our adder

- **P**, Propagate, means the carry-out depends entirely on the carry-in
- **G**, generates a carry-out regardless of the carry-in

<table>
<thead>
<tr>
<th>Ci</th>
<th>A</th>
<th>B</th>
<th>Co</th>
<th>S</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>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
CARRY-SKIP ADDERS

If all full adders in a contiguous block have their Propagate true, then the incoming carry-in can "skip" over the entire block!

Requires extra AND gates and a MUX, but reduces the worst case add-time
**Full Carry-Lookahead**

The fastest adders use full carry look-ahead.

- Given the $P$s and $G$s of a block, one can simultaneously compute the carry-ins for all bits as well as the block using the 3-level SOP methods discussed last lecture.

- Results in an $\Theta(\log_2(N))$, $T_{pd}$, like an $N$-input AND gate, using $\approx 2x$ more gates.
Next Time

We get shifty, no, Bool!