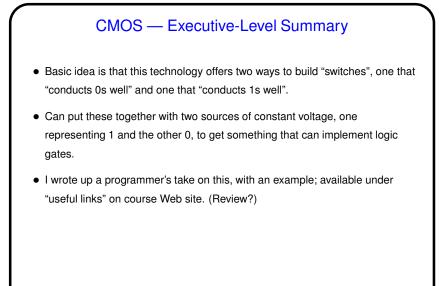
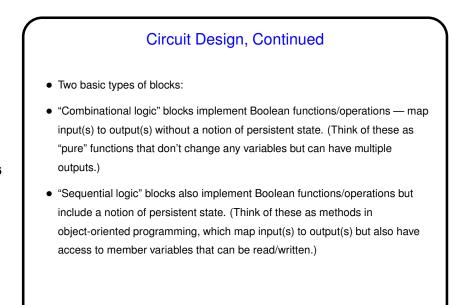
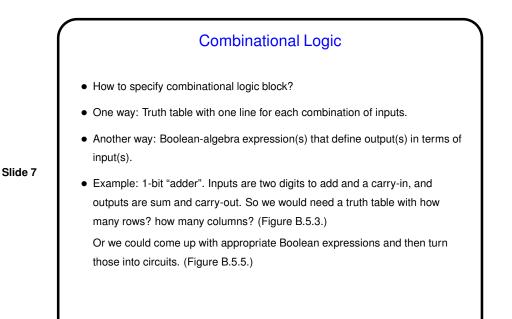


Slide 5





Slide 6



Fwo-Level Logic
Constructing logic blocks that implement arbitrary Boolean algebra expressions could take some thought.
However, any Boolean-algebra expression can be represented in one of two forms, sum of products or product of sums. (Why? Think about truth-table representation.)

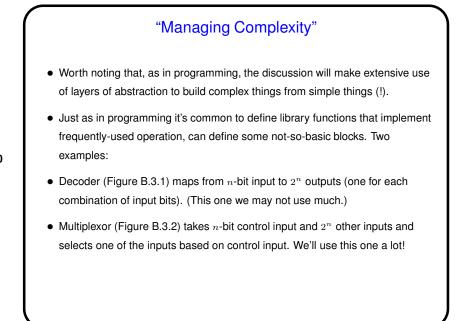
Slide 8



So we can define, for any combinational logic block, something that maps n inputs to m outputs by connecting an "array" of AND gates (one for each combination of inputs) to an "array" of OR gates (one for each output). (Example in Figure B.3.5.)

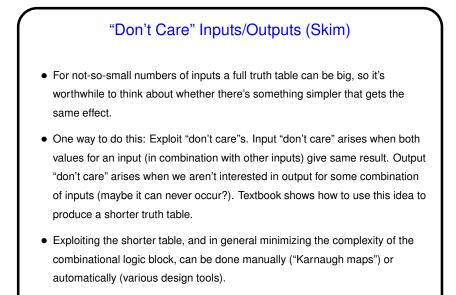
Slide 9

- Note that representation in Figure B.3.5 could be changed to represent a different function by changing the positions of the dots so generic term "programmable logic array" (PLA) makes sense?
- Another standardized way to represent combinational logic block is "ROM" (read-only memory): For n inputs and m outputs we'd need 2ⁿ entries each consisting of m bits.
- For either of these the process of turning a truth table into implementation can be automated(!).



Slide 10

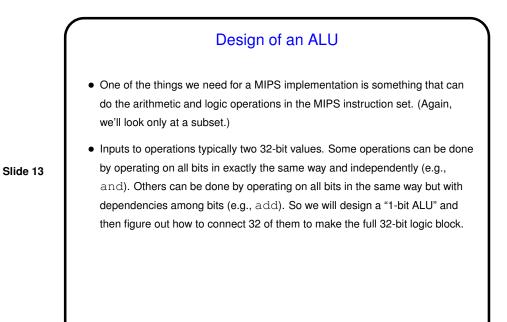
Slide 11



Arrays of Logic Elements

- Descriptions so far (except for decoder) have been in terms of single-bit inputs. But often want to work on larger collections (e.g., 32 bits of a register).
- To do this, can build an "array" of identical logic blocks.
- If inputs/outputs are not in some way connected, can just indicate that input/output values are more than one bit ("bus"). Examples: bitwise AND of 32-bit values, Figure B.3.6.
- If inputs/outputs are connected, idea still works but picture must indicate connections. Example: addition of 32-bit values using 32 single-bit "adder" blocks, each with three inputs (two operands and carry-in) and two outputs (value and carry-out). (Figure shortly.)

Slide 12



Figures B.5.1 through B.5.6 show how we can build up something that performs and, or, and add on 1-bit values (plus carry-in and carry-out values for add).

Slide 14

- 2-bit "which operation?"

- two 1-bit operands

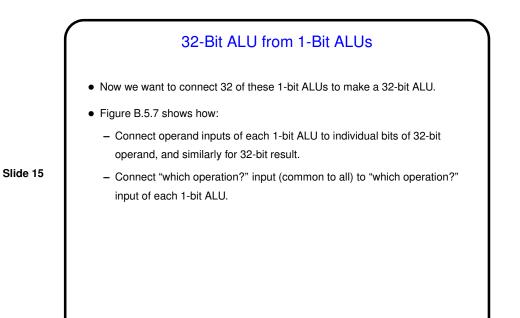
• Result (B.5.6) is a logic block with inputs

- 1-bit carry-in

and outputs

- 1-bit result
- 1-bit carry-out

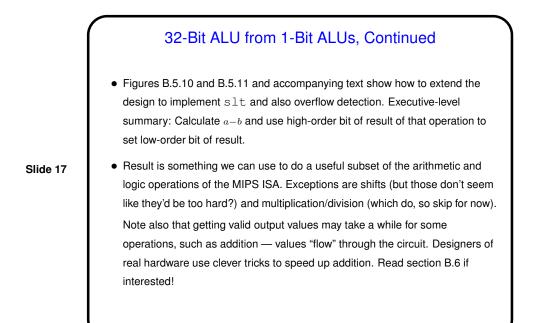
Slide 16



32-Bit ALU from 1-Bit ALUs, Continued
We said when we first talked about two's complement notation that it was attractive because once you build something that can add, you can easily extend it to something that can subtract, right?
Conceptually, we can compute a-b by adding a to -b, and we can compute -b by reversing all the bits of b and adding one — which is just what's shown in Figure B.5.8! which is Figure B.5.7 plus one more input, which:

if 0, makes the initial carry-in 0 and uses b as is.
if 1, makes the initial carry-in 1 and flips bits of b.

We can apply a similar idea (adding an input that lets us use a as is or "flipped") to implement nor (Figure B.5.9).



 Minute Essay

 • Questions? For most of you I'm guessing this is mostly new and unfamiliar, but after some exposure I hope it will make sense!

 Slide 18