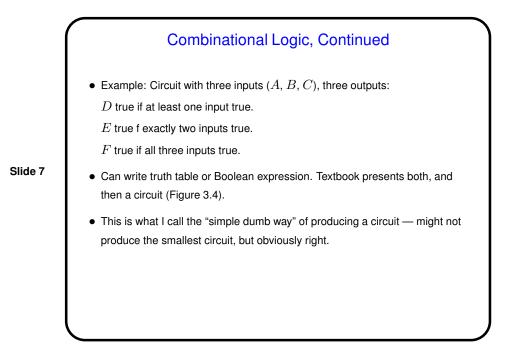
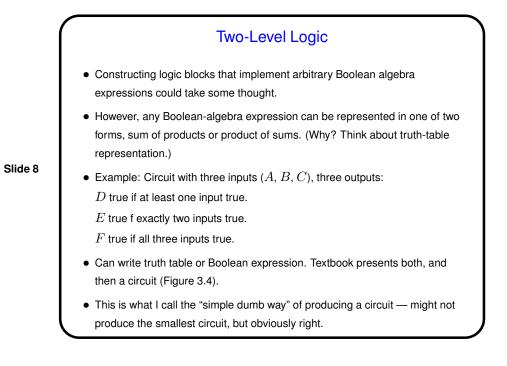


Combinational Logic
How to specify combinational logic block?
One way: Truth table with one line for each combination of inputs.
Another way: Boolean-algebra expression(s) that define output(s) in terms of input(s).



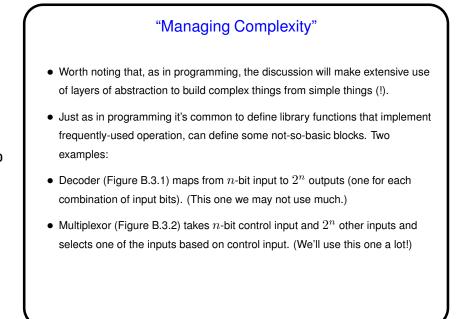


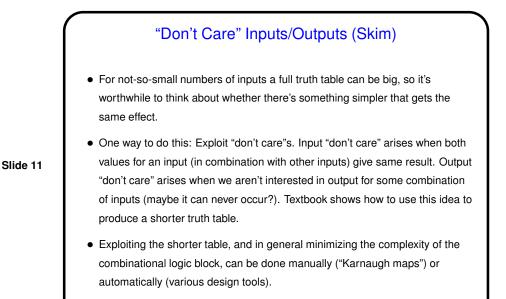


• So we can define, for any combinational logic block, something that maps *n* inputs to *m* outputs by connecting an "array" of AND gates (one for each combination of inputs) to an "array" of OR gates (one for each output). (Example in Figure B.3.5.)

Slide 9

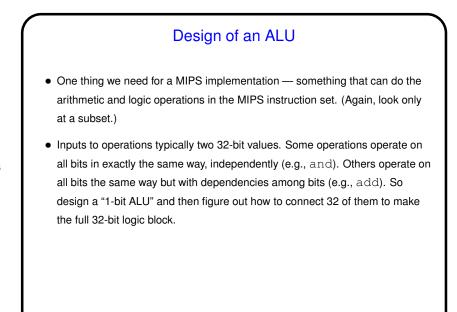
- Note that representation in Figure B.3.5 could be changed to represent a different function by changing the positions of the dots so generic term "programmable logic array" (PLA) makes sense?
- Another standardized way to represent combinational logic block is "ROM" (read-only memory): For n inputs 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(!).

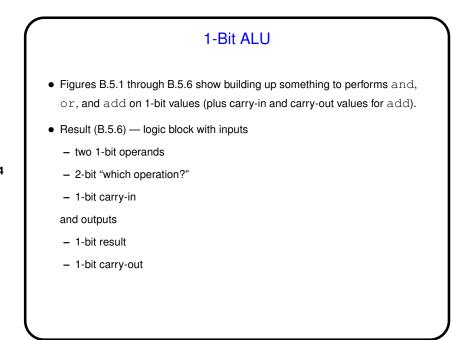


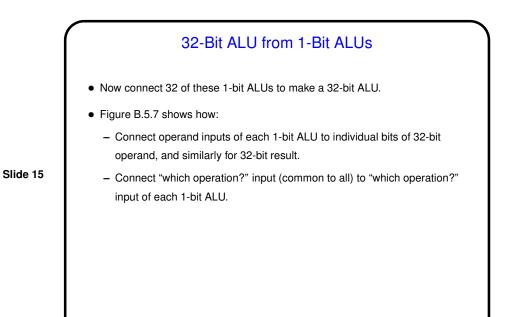


Arrays of Logic Elements

- 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: Figure B.3.6 (bitwise AND of 32-bit values).
- 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



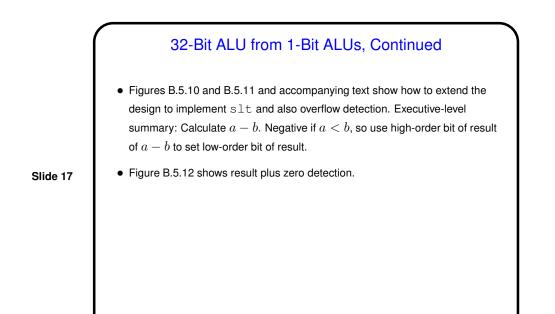


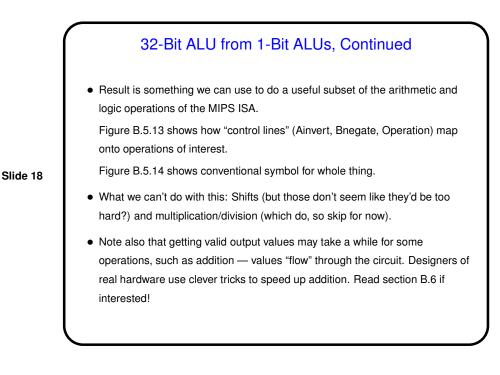


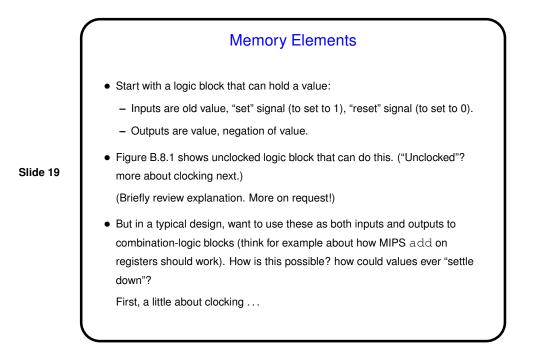
32-Bit ALU from 1-Bit ALUS, Continued
We keep saying that about two's complement notation that it's attractive because once you build something that can add, you can easily extend it to something that can subtract, right?
Conceptually, we can compute a - b by adding a to -b, and we can compute -b by 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:

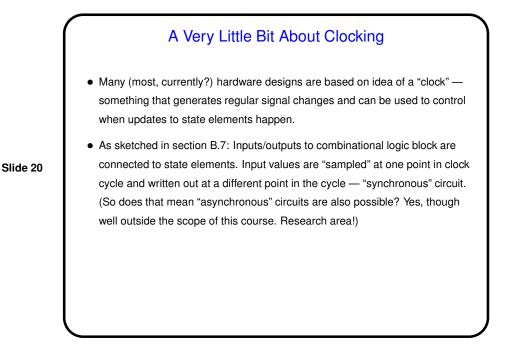
if 0, makes the initial carry-in 0 and uses b as is.
if 1, makes the initial carry-in 1 and flips bits of b.

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). Clever?





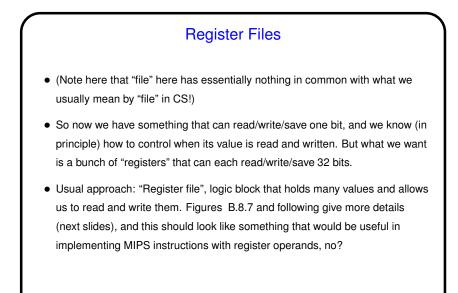




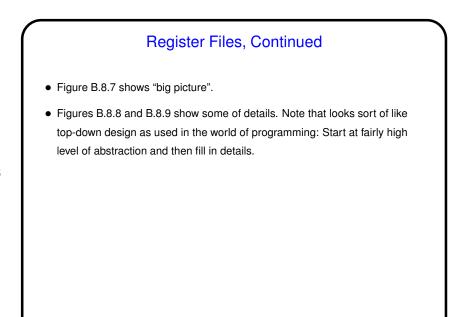


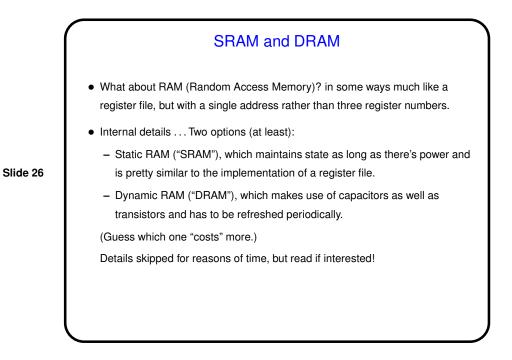
- If input and output of combinational logic are different, all is well. But if they're not (Figure B.7.2)? How can values ever "settle down"?
- So introduce between state element 1 (input) and CL block, some kind of barrier/switch that can either let bits flow or not, and the same thing between CL block and state element 2, with only one of those barriers letting bits flow at a time.

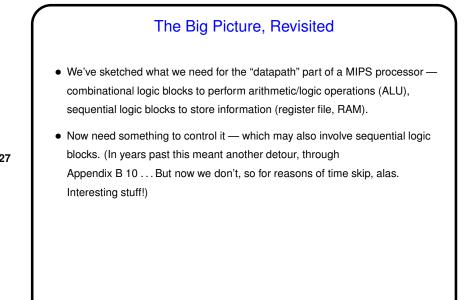
Hemory Elements, Continued Figure B.8.2 shows such a barrier ("latch") — circuit that stores one bit and only samples data input when clock input is 1. Details interesting but not really crucial for this course! Notice how figures use the "layers of abstraction" idea: E.g., first show details of a "latch", then show using it as a black box to build something more complex.

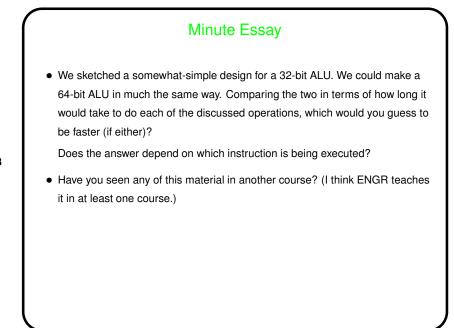


	Register Files, Continued
Slide 24	 Inputs: Two (multi-bit) register numbers saying which registers we want to "read" (use as input to some operation). One (multi-bit) register number saying which register we (might) want to "write" (change the value of). One (32-bit) value to (maybe) save in a register. A "yes do a write" bit. Outputs: Two (32-bit) values representing the contents of the two registers selected by the "read register" numbers used as input.









Slide 28

Minute Essay Answer

The 64-bit ALU will be slower for some operations (such as add), since "values" have "flow" through 64 1-bit ALUs rather than 32. (However, as students have sometimes pointed out, if the ALU is doing all the operations anyway even though only one is being used, in some sense they

do all take the same amount of time.)

Slide 29