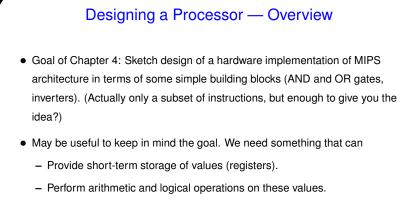


Slide 1

## Minute Essay From Last Lecture Students who said anything about their spring break: Most common was not doing much, maybe going home, getting some rest. But a few traveled: Taiwan (with the choir), Belgium, Cuba. (Mine was relatively relaxing, and I got caught up with grading.) Most students thought the exam was about what they thought it would be except for the length relative to allotted time. I'll try to gauge that better for the second exam!

Slide 3



- Provide longer-term storage of values (memory).
- Transfer data between registers and memory.
- Repeatedly fetch and execute instructions, allowing for both sequential execution and branching/jumps.

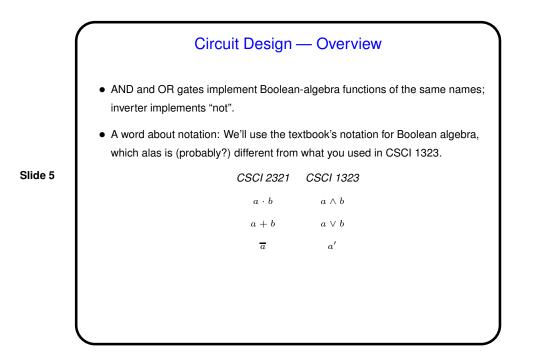


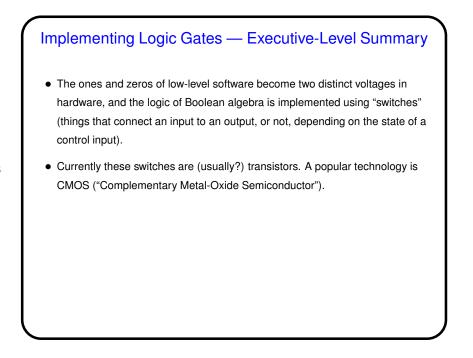
- Key components of the design (Figures 4.1 and 4.2):
  - Something to implement memory.
  - Something to implement instructions: "ALU" (arithmetic/logic unit).
  - Something to implement registers: "register file".

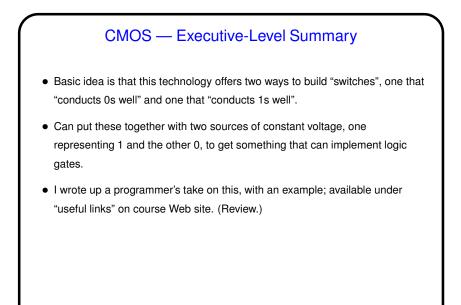
Slide 4

- Something to implement fetch/decode/execute cycle: "control logic".

The first three together make up the "data path". Analogy: It's a puppet, with "control" pulling its strings.

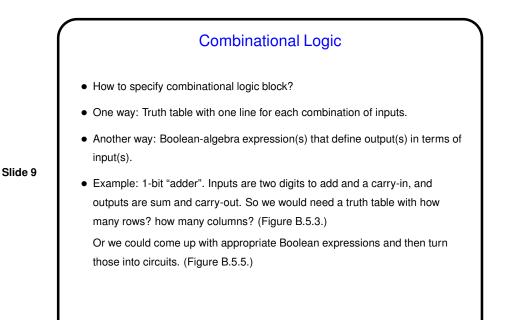






Circuit Design, Continued
Two basic types of blocks:
"Combinational logic" blocks implement Boolean functions/operations — map input(s) to output(s) without a notion of persistent state. (Think of these as "pure" functions that don't change any variables but can have multiple outputs.)
"Sequential logic" blocks also implement Boolean functions/operations but include a notion of persistent state. (Think of these as methods in object-oriented programming, which map input(s) to output(s) but also have access to member variables that can be read/written.)

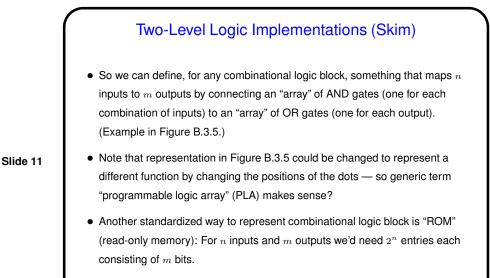
Slide 7



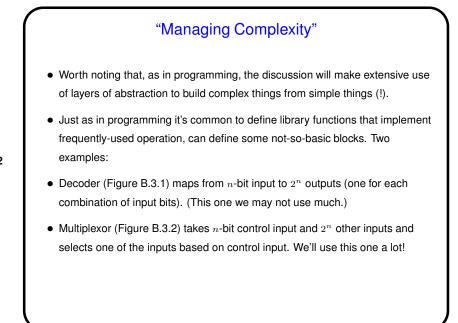
## Function of the product of sums. (Why? Think about truth-table representation.)

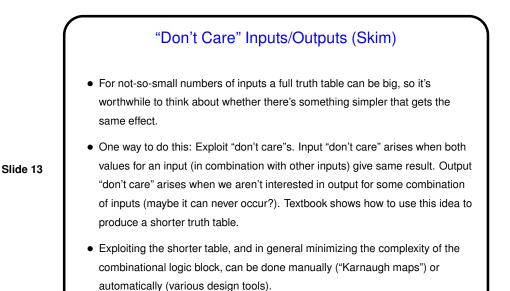
Slide 10

## 5



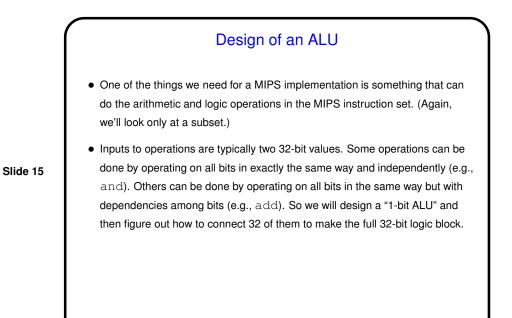
• For either of these the process of turning a truth table into implementation can be automated(!).





Arrays of Logic Elements

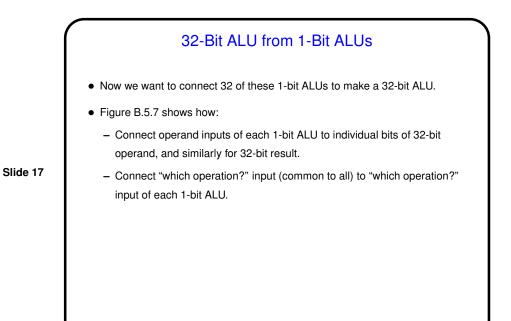
- Descriptions so far (except for decoder) have been in terms of single-bit inputs. But often want to work on larger collections (e.g., 32 bits of a register).
- To do this, can build an "array" of identical logic blocks.
- If inputs/outputs are not in some way connected, can just indicate that input/output values are more than one bit ("bus"). Examples: 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.)

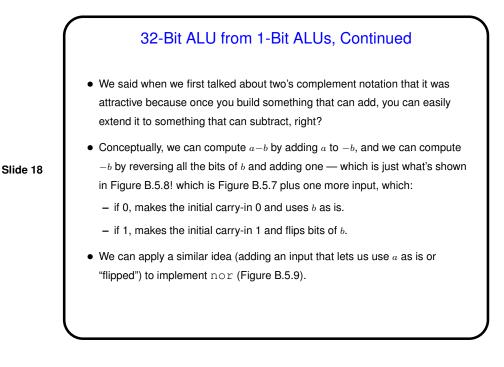


1-Bit ALU
Figures B.5.1 through B.5.6 show how we can build up something that performs and, or, and add on 1-bit values (plus carry-in and carry-out values for add).
Result (B.5.6) is a logic block with inputs

two 1-bit operands
2-bit "which operation?"
1-bit carry-in
and outputs
1-bit result
1-bit carry-out

8





9

