Chapter Three: Building an ALU to support MIPS integer arithmetic
(Material from Appendix B.5 on CD-ROM)

Review: Boolean Algebra & Gates

• Problem: Consider a logic function with three inputs: A, B, and C.
  
  Output D is true if at least one input is true
  Output E is true if exactly two inputs are true
  Output F is true only if all three inputs are true

• Show the truth table for these three functions.
• Show the Boolean equations for these three functions.
• Show an implementation consisting of inverters, AND, and OR gates.

See Appendix B of textbook (on CDROM)
An ALU (arithmetic logic unit)

- Let's build an ALU to support the andi and ori instructions
  - we'll just build a 1 bit ALU, and use 32 of them

- Possible Implementation (sum-of-products):

Review: The Multiplexor

- Selects one of the inputs to be the output, based on a control input

\[ S \]
\[ A \to 0 \]
\[ B \to 1 \]
\[ C \]

note: we call this a 2-input mux even though it has 3 inputs!

- Let's build our ALU using a MUX:
Different Implementations

- Not easy to decide the “best” way to build something
  - Don’t want too many inputs to a single gate
  - Don’t want to have to go through too many gates
  - for our purposes, ease of comprehension is important
- Let’s look at a 1-bit ALU for addition:

\[ \text{cout} = a \ b + a_c + b_c \]
\[ \text{sum} = a \ \text{xor} \ b \ \text{xor} \ c \]

- How could we build a 1-bit ALU for add, and, and or?
- How could we build a 32-bit ALU?

Building a 32 bit ALU
What about subtraction \((a - b)\)?

- Two's complement approach: just negate \(b\) and add.
- How do we negate?

- A very clever solution:

```
<table>
<thead>
<tr>
<th>Operation</th>
<th>CarryIn</th>
<th>CarryOut</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Adding a NOR function

- Can also choose to invert \(A\). How do we get \(A\) nor \(B\)?
Tailoring the ALU to the MIPS

- Need to support the set-on-less-than instruction (slt)
  - remember: slt is an arithmetic instruction
  - produces a 1 if rs < rt and 0 otherwise
  - use subtraction: \((a-b) < 0\) implies \(a < b\)
- Need to support test for equality (beq $t5, $t6, $t7)
  - use subtraction: \((a-b) = 0\) implies \(a = b\)

Supporting slt

Use this ALU for bits 0-30

Use this ALU for most significant bit (bit 31)
Supporting slt

Test for equality

- Notice control lines:

  0000 = and
  0001 = or
  0010 = add
  0110 = subtract
  0111 = slt
  1100 = nor

- Note: zero is a 1 when the result is zero!
Conclusion

• We can build an ALU to support the MIPS instruction set
  – key idea: use multiplexor to select the output we want
  – we can efficiently perform subtraction using two’s complement
  – we can replicate a 1-bit ALU to produce a 32-bit ALU
• Important points about hardware
  – all of the gates are always working
  – the speed of a gate is affected by the number of inputs to the gate
  – the speed of a circuit is affected by the number of gates in series
    (on the “critical path” or the “deepest level of logic”)
• Our primary focus: comprehension, however,
  – Clever changes to organization can improve performance
    (similar to using better algorithms in software)
  – we’ll look next at multiplication