ALU Summary

• We can build an ALU to support MIPS addition
• Our focus is on comprehension, not performance
• Real processors use more sophisticated techniques for arithmetic
• Where performance is not critical, hardware description languages allow designers to completely automate the creation of hardware!

Chapter Five

The Processor: Datapath & Control

• We're ready to look at an implementation of the MIPS
• Simplified to contain only:
  – memory-reference instructions: lw, sw
  – arithmetic-logical instructions: add, sub, and, or, slt
  – control flow instructions: beq, j
• Generic Implementation:
  – use the program counter (PC) to supply instruction address
  – get the instruction from memory
  – read registers
  – use the instruction to decide exactly what to do
• All instructions use the ALU after reading the registers

More Implementation Details

• Abstract / Simplified View:

  Two types of functional units:
  – elements that operate on data values (combinational)
  – elements that contain state (sequential)
State Elements

- Unclocked vs. Clocked
  - Clocks used in synchronous logic
  - When should an element that contains state be updated?

An unclocked state element

- The set-reset latch
  - Output depends on present inputs and also on past inputs

Latches and Flip-flops

- Output is equal to the stored value inside the element
  - (don’t need to ask for permission to look at the value)
- Change of state (value) is based on the clock
- Latches: whenever the inputs change, and the clock is asserted
- Flip-flop: state changes only on a clock edge
  - (edge-triggered methodology)

A clocking methodology defines when signals can be read and written
  - Wouldn’t want to read a signal at the same time it was being written

D-latch

- Two inputs:
  - The data value to be stored (D)
  - The clock signal (C) indicating when to read & store D
- Two outputs:
  - The value of the internal state (Q) and its complement
D flip-flop

- Output changes only on the clock edge

Our Implementation

- An edge triggered methodology
- Typical execution:
  - read contents of some state elements,
  - send values through some combinational logic
  - write results to one or more state elements

Register File

- Built using D flip-flops

Abstraction

- Make sure you understand the abstractions!
- Sometimes it is easy to think you do, when you don’t

Do you understand? What is the “Mux” above?
Register File

• Note: we still use the real clock to determine when to write

Simple Implementation

• Include the functional units we need for each instruction

Building the Datapath

• Use multiplexors to stitch them together

Control

• Selecting the operations to perform (ALU, read/write, etc.)
• Controlling the flow of data (multiplexor inputs)
• Information comes from the 32 bits of the instruction
• Example:

  Instruction Format:
  000000 10001 10010 01000 00000 10000
  op   rs  rt  rd  shamt  funct

• ALU's operation based on instruction type and function code
Control

- e.g., what should the ALU do with this instruction
  Example: lw $1, 100($2)

<table>
<thead>
<tr>
<th>op</th>
<th>rs</th>
<th>rt</th>
<th>16 bit offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>2</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

- ALU control input
  0000  AND
  0001  OR
  0010  add
  0110  subtract
  0111  set-on-less-than
  1100  NOR

- Why is the code for subtract 0110 and not 0011?