ECE232: Hardware Organization and Design

Part 10: Control Design

http://www.ecs.umass.edu/ece/ece232/

Datapath With Control
R-Format Instruction: add $t1, $t2, $t3

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>Memto-Reg</th>
<th>Reg Write</th>
<th>Mem Read</th>
<th>Mem Write</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>R-format 1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Load Instruction

<table>
<thead>
<tr>
<th>Instruction</th>
<th>RegDst</th>
<th>ALUSrc</th>
<th>Memto-Reg</th>
<th>Reg Write</th>
<th>Mem Read</th>
<th>Mem Write</th>
<th>Branch</th>
<th>ALUOp1</th>
<th>ALUOp0</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Branch-on-Equal Instruction

Simple combinational logic
Single-Cycle Machine: Appraisal

- All instructions complete in one clock cycle (CPI = 1)
- Some instructions take more steps than others
  - lw is most expensive (5 steps, vs. 4 for R-type and sw, 3 for beq)
- Clock cycle must cover longest instruction ⇒ inefficient
  - suppose mult is added?
  - 32-shift/add steps ⇒ would delay every other instruction

Cycle time and speedup computation

- Assume:
  - 2ns for instruction/data memory
  - 1ns for decode/register read
  - 2ns for ALU and
  - 1ns for register write
- Single-cycle datapath clock period = 8ns
- Assume an instruction mix of 24% loads, 12% stores, 44% R-format, 18% branches, and 2% jumps
- Assuming a variable-cycle datapath, average clock period = \(8 \times 0.24 + 7 \times 0.12 + 6 \times 0.44 + 5 \times 0.18 + 3 \times 0.02 = 6.36\) ns
- Possible Speed-up = 1.26
Multicycle Implementation (MIPS-lite v.2)

- Want more efficient implementation
- Each step will take one clock cycle (not each instruction) [CPI > 1]
  - shorter clock cycle: cycle time constrained by longest step, not longest instruction
  - simpler instructions take fewer cycles
  - higher overall performance
- More complex control: finite state machine
- Versatile (can extend for new instructions: swap, mult-add etc.)

Clocking: single-cycle vs. multi-cycle

**Single-cycle Implementation**

- Clock
- `add $t0,$t1,$t2`
- `beq $t0,$t1,L`

**Multicycle Implementation**

- Clock
- `add $t0,$t1,$t2`
- `beq $t0,$t1,L`

Multicycle Implementation: less waste=higher performance
How fast can we run the clock?

- Depends on how much we want to be done per clock cycle
  - Can do: several “inexpensive” datapath operations per clock
    - simple gates (AND, OR, ...)
    - single datapath registers (PC)
    - sign extender, left shifter, multiplexor
  - OR: exactly one “expensive” datapath operation per clock
    - ALU operation
    - Register File access (2 reads, or 1 write)
    - Memory access (read or write)

Multicycle Datapath (overview)

- One ALU (no extra adders)
- One Memory (no separate IMem, DMem)
- New Temporary Registers (“clocked”/require clock input)
Multicycle Implementation

- Datapath changes
  - one memory: both instructions and data (because can access on separate steps)
  - one ALU (eliminate extra adders)
  - extra "invisible" registers to capture intermediate (per-step) datapath results
- Controller changes
  - controller must fire control lines in correct sequence and correct time
  ⇒ controller must remember current execution step, advance to next step

Datapath + Control Points
FSM diagram for multi-cycle machine

FSM controller: execution cycles 3-5
Cycle 1

MemRead
MemWrite
IorD
IRWrite
RegDst
RegWrite
ALUSrcA
PCWrite
PCSrc

PC
Read
Mem
Address
Read
Data
Write
Data
M
D
R
ALU
ALU Out
ALUSrcE
ALUOp
MemtoReg
ALU Control
Sgn Ext
(0:5)
A
B

MemRead
ALUSrcA = 0
IorD = 0
IRWrite
ALUSrcB = 1
ALUOp = 0
PCWrite
PCSrc = 0