Boolean Exprs for Controller

RegDst = add + sub
ALUSrc = ori + lw + sw
MemtoReg = lw
RegWrite = add + sub + ori + lw
MemWrite = sw
nPCsel = beq
Jump = jump
ExtOp = lw + sw
(assume ALUctr is 00 ADD, 01 SUB, 10 OR)

How do we implement this in gates?

Controller Implementation

Controller Implementation

High Level Language Program (e.g., C)
Assembly Language Program (e.g., MIPS)
Machine Language Program (MIPS)

Machine Interpretation

Architecture Description (e.g., block diagrams)
Logic Circuit Description (Circuit Schematic Diagrams)

temp = v[k];

v[k] = v[k+1];

v[k+1] = temp;

lw $t0, 0($2)
lw $t1, 4($2)
sw $t1, 0($2)
sw $t0, 4($2)

Logic Circuit Description

Call home, we’ve made HW/SW contact!

How many hours of fun from proj3 so far?

a) 0 <= F <= 4
b) 4 < F <= 8
c) 8 < F <= 12
d) 12 < F <= 16
e) 16 < F

Administrivia/Clicker
**Administrivia/Clicker**

- How many Gflop/s right now?
  a) $0 \leq F \leq 4$
  b) $4 < F \leq 8$
  c) $8 < F \leq 12$
  d) $12 < F \leq 16$
  e) $16 < F$

**Review: Single-cycle Processor**

- Five steps to design a processor:
  1. Analyze instruction set → datapath requirements
  2. Select set of datapath components & establish clock methodology
  3. Assemble datapath meeting the requirements
  4. Analyze implementation of each instruction to determine setting of control points that effects the register transfer.
  5. Assemble the control logic
    - Formulate Logic Equations
    - Design Circuits

**Single Cycle Performance**

- Assume time for actions are
  - 100ps for register read or write; 200ps for other events
- Clock rate is?

<table>
<thead>
<tr>
<th>Instr</th>
<th>Instr fetch</th>
<th>Register read</th>
<th>ALU op</th>
<th>Memory access</th>
<th>Register write</th>
<th>Total time</th>
</tr>
</thead>
<tbody>
<tr>
<td>lw</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>100ps</td>
<td>800ps</td>
<td></td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>100ps</td>
<td>700ps</td>
<td></td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>100ps</td>
<td>600ps</td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100ps</td>
<td>200ps</td>
<td>100ps</td>
<td>500ps</td>
<td></td>
</tr>
</tbody>
</table>

- What can we do to improve clock rate?
- Will this improve performance as well?
  Want increased clock rate to mean faster programs

**Gotta Do Laundry**

- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, fold, and put away
  - Washer takes 30 minutes
  - Dryer takes 30 minutes
  - “Folder” takes 30 minutes
  - “Stasher” takes 30 minutes to put clothes into drawers

**Sequential Laundry**

- Sequential laundry takes 8 hours for 4 loads

- Time 6 PM 7 8 9 10 11 12 1 2 AM
- Task Order

- 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
Pipelined Laundry

- Pipelined laundry takes 3.5 hours for 4 loads!

Pipelining Lessons (1/2)

- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Multiple tasks operating simultaneously using different resources
- Potential speedup = Number pipe stages
- Time to "fill" pipeline and time to "drain" it reduces speedup: 2.3X v. 4X in this example

Pipelining Lessons (2/2)

- Suppose new Washer takes 20 minutes, new Stasher takes 20 minutes. How much faster is pipeline?
- Pipeline rate limited by slowest pipeline stage
- Unbalanced lengths of pipe stages reduces speedup

Steps in Executing MIPS

1) *Ifetch*: Instruction Fetch, Increment PC
2) *Dec*: Instruction Decode, Read Registers
3) *Exec*:
   - Mem-ref: Calculate Address
   - Arith-log: Perform Operation
4) *Mem*:
   - Load: Read Data from Memory
   - Store: Write Data to Memory
5) *Wb*: Write Data Back to Register

Single Cycle Datapath

- Need registers between stages
  - To hold information produced in previous cycle
More Detailed Pipeline

IF for Load, Store, ...

ID for Load, Store, ...

EX for Load

MEM for Load

WB for Load – Oops!

Wrong register number
Corrected Datapath for Load

So, in conclusion

• You now know how to implement the control logic for the single-cycle CPU.
  – (actually, you already knew it!)
• Pipelining improves performance by increasing instruction throughput: exploits ILP
  – Executes multiple instructions in parallel
  – Each instruction has the same latency
• Next: hazards in pipelining:
  – Structure, data, control