Japanese scientists have made organic molecules perform parallel computations like neurons in the human brain. They created this promising new approach with a ring-like molecule called 2,3-dichloro-5,6-dicyano-p-benzoquinone, or DDQ.

Today, computer chips can process data at 10 trillion (10^13) bits per second. But, even though neurons in the human brain fire at a rate of 100 million (10^8) times per second, the brain still outperforms the best computers at various tasks. The main reason being that calculations done by computer chips happen in isolated pipelines one at a time.

Where Do Control Signals Come From?

Control Signals

Instruction(32 bits):

- Inst: Instruction
- Addr: Address
- Op: Operation
- Fun: Function
- Rd: Register
- Rs: Register
- Rz: Register
- Rt: Register
- MemWr: Memory Write
- MemtoReg: Memory to Register
- RegWr: Register Write
- RegDst: Register Destination
- ALUsrc: ALU Source
- ExtOp: External Operation
- ALUctr: ALU Control
- nPC_sel: nPC Select
- MemSel: Memory Select
- Mux: Mux

Control Signals:

- Inst:
- Addr:
- Op:
- Fun:
- Rd:
- Rs:
- Rz:
- Rt:
- MemWr:
- MemtoReg:
- RegWr:
- RegDst:
- ALUsrc:
- ExtOp:
- ALUctr:
- nPC_sel:
- MemSel:
- Mux:

Summary of the Control Signals (1/2)

- add: R[rd] ← R[rs] + R[rt]; PC ← PC + 4
- sub: R[rd] ← R[rs] - R[rt]; PC ← PC + 4
- ori: R[rt] ← R[rs] + zero_ext(imm16); PC ← PC + 4
- lw: R[rt] ← MEM[R[rs] + sign_ext(imm16)]; PC ← PC + 4
- beq if (R[rs] == R[rt]) then PC ← PC + 4
Summary of the Control Signals (2/2)

<table>
<thead>
<tr>
<th>Summary of the Control Signals (2/2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>RegDst</td>
</tr>
<tr>
<td>ALUSrc</td>
</tr>
<tr>
<td>MemWrite</td>
</tr>
<tr>
<td>Jump</td>
</tr>
<tr>
<td>ALUctr[1]</td>
</tr>
<tr>
<td>ALUctr[0]</td>
</tr>
</tbody>
</table>

Boolean Exprs for Controller

\[ \text{Op} - 0 - 5 \text{ are really Instruction bits 26-31} \]
\[ \text{Func} - 0 - 5 \text{ are really instruction bits 0-5} \]
\[ \text{rtype} = \text{op}, \text{op}, \text{op}, \text{op}, \text{op}, \text{op} \]
\[ \text{ori} = \text{op}, \text{op}, \text{op}, \text{op}, \text{op}, \text{op} \]
\[ \text{lw} = \text{op}, \text{op}, \text{op}, \text{op}, \text{op}, \text{op} \]
\[ \text{sw} = \text{op}, \text{op}, \text{op}, \text{op}, \text{op}, \text{op} \]
\[ \text{beq} = \text{op}, \text{op}, \text{op}, \text{op}, \text{op}, \text{op} \]
\[ \text{jump} = \text{op}, \text{op}, \text{op}, \text{op}, \text{op}, \text{op} \]
\[ \text{add} = \text{rtype} + \text{func}, \text{func}, \text{func}, \text{func}, \text{func}, \text{func} \]
\[ \text{sub} = \text{rtype} + \text{func}, \text{func}, \text{func}, \text{func}, \text{func}, \text{func} \]

How do we implement this in gates?

Controller Implementation

```
<table>
<thead>
<tr>
<th>Controller Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>Add</td>
</tr>
<tr>
<td>Ori</td>
</tr>
<tr>
<td>Rw</td>
</tr>
<tr>
<td>Jump</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Controller Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Op</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>Add</td>
</tr>
<tr>
<td>Ori</td>
</tr>
<tr>
<td>Rw</td>
</tr>
<tr>
<td>Jump</td>
</tr>
</tbody>
</table>
```

Boolean Exprs for Controller

\[ \text{RegDst} = \text{add + sub} \]
\[ \text{ALUSrc} = \text{ori + lw + sw} \]
\[ \text{MemtoReg} = \text{lw} \]
\[ \text{RegWrite} = \text{add + sub + ori + lw} \]
\[ \text{MemWrite} = \text{sw} \]
\[ \text{nPCsel} = \text{beq} \]
\[ \text{Jump} = \text{jump} \]
\[ \text{ExtOp} = \text{lw + sw} \]
\[ \text{ALUctr[0]} = \text{sub + beq} \]
\[ \text{ALUctr[1]} = \text{ori} \]

(assume ALUctr is 00 ADD, 01 SUB, 10 OR)

How do we implement this in gates?

Call home, we've made HW/SW contact!
Administrivia

• Due to time constraints – we can only allow the use of a maximum of 2 slip days on Project 4.

• Thus, while we always encourage you to get your work done on time, if you still have 3 slip days left, you may want to consider using one prior to project 4.

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>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>800ps</td>
</tr>
<tr>
<td>sw</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>700ps</td>
</tr>
<tr>
<td>R-format</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>100 ps</td>
<td>600ps</td>
<td></td>
</tr>
<tr>
<td>beq</td>
<td>200ps</td>
<td>100 ps</td>
<td>200ps</td>
<td>100 ps</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
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) I/Fetch: Instruction Fetch, Increment PC
2) D/Decode: Instruction Decode, Read Registers
3) E/Exec:
   - Mem-ref: Calculate Address
   - Arith-log: Perform Operation
4) M/Mem:
   - Load: Read Data from Memory
   - Store: Write Data to Memory
5) W/Write: Write Data Back to Register

Single Cycle Datapath

Pipeline registers

- Need registers between stages
  - To hold information produced in previous cycle
Chapter 4: The Processor

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