## Latest NRC Rankings!!

<table>
<thead>
<tr>
<th>Quality of Faculty</th>
<th>1982</th>
<th>1995</th>
<th>Change</th>
<th>Ranking</th>
<th>1982</th>
<th>1995</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stanford</td>
<td>5.0</td>
<td>4.97</td>
<td>-0.03</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>MIT</td>
<td>4.9</td>
<td>4.91</td>
<td>0.01</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>CMU</td>
<td>4.8</td>
<td>4.76</td>
<td>-0.04</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>Berkeley</td>
<td>4.5</td>
<td>4.88</td>
<td>0.38</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Cornell</td>
<td>4.3</td>
<td>4.64</td>
<td>0.34</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Illinois</td>
<td>3.8</td>
<td>4.09</td>
<td>0.29</td>
<td>6</td>
<td>8</td>
<td>8</td>
<td>-2</td>
</tr>
<tr>
<td>UCLA</td>
<td>3.8</td>
<td>na</td>
<td></td>
<td>6</td>
<td>na</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>Yale</td>
<td>3.5</td>
<td>na</td>
<td></td>
<td>8</td>
<td>na</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>U.Wash.</td>
<td>3.4</td>
<td>4.04</td>
<td>0.64</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>USC</td>
<td>3.2</td>
<td>na</td>
<td></td>
<td>10</td>
<td>na</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>UT/Austin</td>
<td>3.2</td>
<td>4.18</td>
<td>0.98</td>
<td>10</td>
<td>7</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>Wisconsin</td>
<td>3.2</td>
<td>4.00</td>
<td>0.80</td>
<td>10</td>
<td>10</td>
<td>10</td>
<td>0</td>
</tr>
<tr>
<td>Maryland</td>
<td>3.1</td>
<td>na</td>
<td></td>
<td>13</td>
<td>na</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>Princeton</td>
<td>3.0</td>
<td>4.31</td>
<td>1.31</td>
<td>14</td>
<td>6</td>
<td>6</td>
<td>8</td>
</tr>
</tbody>
</table>
Review: Instruction Set Evolution

- Single Accumulator (EDSAC 1950)
- Accumulator + Index Registers (Manchester Mark I, IBM 700 series 1953)
- Separation of Programming Model from Implementation
- High-level Language Based (B5000 1963)
- Concept of a Family (IBM 360 1964)
- General Purpose Register Machines
- Complex Instruction Sets (Vax, Intel 432 1977-80)
- Load/Store Architecture (CDC 6600, Cray 1 1963-76)
- RISC (Mips, Sparc, HP PA, PowerPC, . . .1987-91)
## Review: 80x86 v. DLX

### Instruction Counts

<table>
<thead>
<tr>
<th>SPEC pgm</th>
<th>x86</th>
<th>DLX</th>
<th>DLX÷86</th>
</tr>
</thead>
<tbody>
<tr>
<td>gcc</td>
<td>3,771,327,742</td>
<td>3,892,063,460</td>
<td>1.03</td>
</tr>
<tr>
<td>espresso</td>
<td>2,216,423,413</td>
<td>2,801,294,286</td>
<td>1.26</td>
</tr>
<tr>
<td>spice</td>
<td>15,257,026,309</td>
<td>16,965,928,788</td>
<td>1.11</td>
</tr>
<tr>
<td>nasa7</td>
<td>15,603,040,963</td>
<td>6,118,740,321</td>
<td>0.39</td>
</tr>
</tbody>
</table>
Review From Last Time
Intel Summary

- Archeology: history of instruction design in a single product
  - Address size: 16 bit vs. 32-bit
  - Protection: Segmentation vs. paged
  - Temp. storage: accumulator vs. stack vs. registers

- “Golden Handcuffs” of binary compatibility affect design 20 years later, as Moore predicted

- Not too difficult to make faster, as Intel has shown

- HP/Intel announcement of common future instruction set by 2000 means end of 80x86???

- “Beauty is in the eye of the beholder”
  - At 50M/year sold, it is a beautiful business
Pipelining: Its Natural!

- Laundry Example
- Ann, Brian, Cathy, Dave each have one load of clothes to wash, dry, and fold
- Washer takes 30 minutes
- Dryer takes 40 minutes
- “Folder” takes 20 minutes
• Sequential laundry takes 6 hours for 4 loads
• If they learned pipelining, how long would laundry take?
Pipelined Laundry
Start work ASAP

- Pipelined laundry takes 3.5 hours for 4 loads
Pipelining Lessons

- Pipelining doesn’t help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to “fill” pipeline and time to “drain” it reduces speedup
5 Steps of DLX Datapath

Figure 3.1, Page 130

Instruction Fetch

Instr. Decode Reg. Fetch

Execute Addr. Calc

Memory Access

Write Back

IR

L

M

D
Pipelined DLX Datapath

Figure 3.4, page 137

- Data stationary control
  - local decode for each instruction phase / pipeline stage
Visualizing Pipelining

Figure 3.3, Page 133

- Time (clock cycles)
Its Not That Easy for Computers

- Limits to pipelining: **Hazards** prevent next instruction from executing during its designated clock cycle
  - **Structural hazards**: HW cannot support this combination of instructions
  - **Data hazards**: Instruction depends on result of prior instruction still in the pipeline
  - **Control hazards**: Pipelining of branches & other instructions that change the PC

- Common solution is to **stall** the pipeline until the hazard is resolved, inserting one or more “bubbles” in the pipeline
One Memory Port/Structural Hazards
Figure 3.6, Page 142

Time (clock cycles)

Load

Instr 1

Instr 2

Instr 3

Instr 4
One Memory Port/Structural Hazards

Time (clock cycles)

Load
Instr 1
Instr 2
Instr 3

Figure 3.7, Page 143
**Speed Up Equation for Pipelining**

**Speedup from pipelining** = \( \frac{\text{Ave Instr Time unpipelined}}{\text{Ave Instr Time pipelined}} \)

= \( \frac{\text{CPI}_{\text{unpipelined}} \times \text{Clock Cycle}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}} \times \text{Clock Cycle}_{\text{pipelined}}} \)

= \( \frac{\text{CPI}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}} \)

**Ideal CPI** = \( \frac{\text{CPI}_{\text{unpipelined}}}{\text{Pipeline depth}} \)

**Speedup** = \( \frac{\text{Ideal CPI} \times \text{Pipeline depth} \times \text{Clock Cycle}_{\text{unpipelined}}}{\text{CPI}_{\text{pipelined}} \times \text{Clock Cycle}_{\text{pipelined}}} \)
Speed Up Equation for Pipelining

\[ \text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}} \]

\[ \text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}} \]
Example: Dual-port vs. Single-port

- Machine A: Dual ported memory
- Machine B: Single ported memory, but its pipelined implementation has a 1.05 times faster clock rate
- Ideal CPI = 1 for both
- Loads are 40% of instructions executed

\[
\text{SpeedUp}_A = \frac{\text{Pipeline Depth}}{(1 + 0)} \times \frac{\text{clock}_{\text{unpipe}}}{\text{clock}_{\text{pipe}}} = \text{Pipeline Depth}
\]

\[
\text{SpeedUp}_B = \frac{\text{Pipeline Depth}}{(1 + 0.4 \times 1)} \times \left(\frac{\text{clock}_{\text{unpipe}}}{\text{clock}_{\text{unpipe}} / 1.05}\right) = \frac{\text{Pipeline Depth}}{1.4} \times 1.05 = 0.75 \times \text{Pipeline Depth}
\]

\[
\frac{\text{SpeedUp}_A}{\text{SpeedUp}_B} = \frac{\text{Pipeline Depth}}{(0.75 \times \text{Pipeline Depth})} = 1.33
\]

- Machine A is 1.33 times faster
Data Hazard on R1

Figure 3.9, page 147

Time (clock cycles)

<table>
<thead>
<tr>
<th>IF</th>
<th>ID/RF</th>
<th>EX</th>
<th>MEM</th>
<th>WB</th>
</tr>
</thead>
</table>

- add r1, r2, r3
- sub r4, r1, r3
- and r6, r1, r7
- or r8, r1, r9
- xor r10, r1, r11
Three Generic Data Hazards

$\text{Instr}_i$ followed be $\text{Instr}_j$

- **Read After Write (RAW)**
  $\text{Instr}_j$ tries to read operand before $\text{Instr}_i$ writes it
Three Generic Data Hazards

Instr\textsubscript{i} followed be Instr\textsubscript{j}

- **Write After Read (WAR)**
  Instr\textsubscript{j} tries to write operand before Instr\textsubscript{i} reads it

- Can’t happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages,
  - Reads are always in stage 2, and
  - Writes are always in stage 5
Three Generic Data Hazards

Instr\textsubscript{i} followed be Instr\textsubscript{j}

- **Write After Write (WAW)**
  - Instr\textsubscript{j} tries to write operand before Instr\textsubscript{i} writes it
  - Leaves wrong result (Instr\textsubscript{i} not Instr\textsubscript{j})

- Can’t happen in DLX 5 stage pipeline because:
  - All instructions take 5 stages, and
  - Writes are always in stage 5

- Will see WAR and WAW in later more complicated pipes
Forwarding to Avoid Data Hazard

Figure 3.10, Page 149

Time (clock cycles)

Instr. Order

add r1, r2, r3
sub r4, r1, r3
and r6, r1, r7
or r8, r1, r9
xor r10, r1, r11
HW Change for Forwarding

Figure 3.20, Page 161
Data Hazard Even with Forwarding

Figure 3.12, Page 153

Instr. Order

lw r1, 0(r2)

sub r4, r1, r6

and r6, r1, r7

or r8, r1, r9
Data Hazard Even with Forwarding
Figure 3.13, Page 154

Instr.
Order

Time (clock cycles)

lw r1, 0(r2)
sub r4, r1, r6
and r6, r1, r7
or r8, r1, r9
Software Scheduling to Avoid Load Hazards

Try producing fast code for

\[
\begin{align*}
a &= b + c; \\
d &= e - f;
\end{align*}
\]

assuming \(a, b, c, d, e,\) and \(f\) in memory.

<table>
<thead>
<tr>
<th>Slow code:</th>
<th>Fast code:</th>
</tr>
</thead>
<tbody>
<tr>
<td>LW Rb,b</td>
<td>LW Rb,b</td>
</tr>
<tr>
<td>LW Rc,c</td>
<td>LW Rc,c</td>
</tr>
<tr>
<td>ADD Ra,Rb,Rc</td>
<td>LW Re,e</td>
</tr>
<tr>
<td>SW a,Ra</td>
<td>ADD Ra,Rb,Rc</td>
</tr>
<tr>
<td>LW Re,e</td>
<td>LW Rf,f</td>
</tr>
<tr>
<td>LW Rf,f</td>
<td>SW a,Ra</td>
</tr>
<tr>
<td>SUB Rd,Re,Rf</td>
<td>SUB Rd,Re,Rf</td>
</tr>
<tr>
<td>SW d,Rd</td>
<td>SW d,Rd</td>
</tr>
</tbody>
</table>

RHK.SP96 27
Compiler Avoiding Load Stalls

% loads stalling pipeline

- gcc: 31% scheduled, 54% unscheduled
- spice: 14% scheduled, 42% unscheduled
- tex: 25% scheduled, 65% unscheduled
Pipelining Summary

• Just overlap tasks, and easy if tasks are independent
• Speed Up ≤ Pipeline Depth; if ideal CPI is 1, then:

\[
\text{Speedup} = \frac{\text{Pipeline Depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle Unpipelined}}{\text{Clock Cycle Pipelined}}
\]

• Hazards limit performance on computers:
  – Structural: need more HW resources
  – Data: need forwarding, compiler scheduling
  – Control: discuss next time