Summary

• Fundamentals of computer architecture
  – Main elements of a computer
  – Inside the processor
  – Assembly language

• Performance metrics
  – Clock rate
  – CPI
Below Your Program

• **Application software**
  – Written in high-level language (HLL)

• **System software**
  – Compiler: translates HLL code to machine code
  – Operating System: service code
    • Handling input/output
    • Managing memory and storage
    • Scheduling tasks & sharing resources

• **Hardware**
  – Processor, memory, I/O controllers
Levels of Program Code

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability

- **Assembly language**
  - Textual representation of instructions

- **Hardware representation**
  - Binary digits (bits)
  - Encoded instructions and data

---

High-level language program (in C)

```c
swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Assembly language program (for MIPS)

```assembly
swap:
    mul $2, $5, 4
    add $2, $4, $2
    lw $15, 0($2)
    lw $16, 4($2)
    sw $16, 0($2)
    sw $15, 4($2)
    jr $31
```

Binary machine language program (for MIPS)

```
00000000000000101000010000000000000110000
00000000000000001100000000000100001
10001100110011000000000000000000000000
1000110000000000000000000000000000000
101011001111001000000000000000000000
101011000110000000000000000000000000
000000000000000000000000000000000000
```

---

Computer Organization / Architectures for Embedded Computing
Components of a Computer

- Same components for all kinds of computer
  - Desktop, server, embedded

- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers
Anatomy of a Computer

Output device

Network cable

Input device

Input device
Opening the Box

- Hard drive
- Processor
- Fan with cover
- Spot for memory DIMMs
- Spot for battery
- Motherboard
- Fan with cover
- DVD drive
Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data
Inside the Processor

- AMD Barcelona: 4 processor cores
Understanding Performance

• Algorithm
  – Determines number of operations executed

• Programming language, compiler, architecture
  – Determine number of machine instructions executed per operation

• Processor and memory system
  – Determine how fast instructions are executed

• I/O system (including OS)
  – Determines how fast I/O operations are executed
Defining Performance

Which airplane has the best performance?

- Boeing 777
- Boeing 747
- BAC/Sud Concorde
- Douglas DC-8-50

**Passenger Capacity**
- Douglas DC-8-50
- BAC/Sud Concorde
- Boeing 747
- Boeing 777

**Cruising Range (miles)**
- Douglas DC-8-50
- BAC/Sud Concorde
- Boeing 747
- Boeing 777

**Cruising Speed (mph)**
- Douglas DC-8-50
- BAC/Sud Concorde
- Boeing 747
- Boeing 777

**Passengers x mph**
- Douglas DC-8-50
- BAC/Sud Concorde
- Boeing 747
- Boeing 777
Response Time and Throughput

• Response time
  – How long it takes to do a task

• Throughput
  – Total work done per unit time
    • e.g., tasks/transactions/… per hour

• How are response time and throughput affected by
  – Replacing the processor with a faster version?
  – Adding more processors?

• We’ll focus on response time for now…
Relative Performance

• Define: Performance = 1 / Execution Time

• “X is $n$ times faster than Y”

\[
\frac{\text{Performance}_X}{\text{Performance}_Y} = \frac{\text{Execution Time}_Y}{\text{Execution Time}_X} = n
\]

• Example: time taken to run a program
  – 10s on A, 15s on B
  – $\text{Execution Time}_B / \text{Execution Time}_A = 15s / 10s = 1.5$
  – So A is 1.5 times faster than B
Measuring Execution Time

• Elapsed time
  – Total response time, including all aspects
    • Processing, I/O, OS overhead, idle time
  – Determines system performance

• CPU time
  – Time spent processing a given job
    • Discounts I/O time, other jobs’ shares
  – Comprises user CPU time and system CPU time
  – Different programs are affected differently by CPU and system performance
Operation of digital hardware governed by a constant-rate clock

- **Clock period**: duration of a clock cycle
  - e.g., $T_{\text{clk}} = 250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- **Clock frequency (rate)**: cycles per second
  - e.g., $f_{\text{clk}} = 1/T_{\text{clk}} = 4.0\text{GHz} = 4,000\text{MHz} = 4.0 \times 10^9\text{Hz}$
CPU Time

\[ \text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \]

\[ = \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}} \]

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count
CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes $1.2 \times$ clock cycles
- How fast must Computer B clock be?

\[
\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}
\]

\[
\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A
\]

\[
= 10s \times 2\text{GHz} = 20 \times 10^9
\]

\[
\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4\text{GHz}
\]
Instruction Count and CPI

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction (CPI)
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

Clock Cycles = Instruction Count \times Cycles per Instruction

CPU Time = Instruction Count \times CPI \times Clock Cycle Time

\[
\text{Instruction Count} \times \text{CPI} = \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}
\]
CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

\[
\begin{align*}
\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\
&= l \times 2.0 \times 250\text{ps} = l \times 500\text{ps} \\
\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\
&= l \times 1.2 \times 500\text{ps} = l \times 600\text{ps} \\
\frac{\text{CPU Time}_B}{\text{CPU Time}_A} &= \frac{l \times 600\text{ps}}{l \times 500\text{ps}} = 1.2
\end{align*}
\]

A is faster…

…by this much
CPI in More Detail

If different instruction classes take different numbers of cycles:

\[
\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)
\]

Weighted average CPI:

\[
\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)
\]

Relative frequency
CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

<table>
<thead>
<tr>
<th>Class</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPI for class</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>IC in sequence 1</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>IC in sequence 2</td>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Sequence 1:
- IC = 5
- Clock Cycles = 2×1 + 1×2 + 2×3
  = 10
- Avg. CPI = 10/5 = 2.0

Sequence 2:
- IC = 6
- Clock Cycles = 4×1 + 1×2 + 1×3
  = 9
- Avg. CPI = 9/6 = 1.5
A Simple Example

<table>
<thead>
<tr>
<th>Op</th>
<th>Freq</th>
<th>CPI&lt;sub&gt;i&lt;/sub&gt;</th>
<th>Freq x CPI&lt;sub&gt;i&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>50%</td>
<td>1</td>
<td>.5</td>
</tr>
<tr>
<td>Load</td>
<td>20%</td>
<td>5</td>
<td>1.0</td>
</tr>
<tr>
<td>Store</td>
<td>10%</td>
<td>3</td>
<td>.3</td>
</tr>
<tr>
<td>Branch</td>
<td>20%</td>
<td>2</td>
<td>.4</td>
</tr>
<tr>
<td>CPI =</td>
<td></td>
<td></td>
<td>2.2</td>
</tr>
</tbody>
</table>

**CC = Clock Cycles**

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

\[
\text{CPU time new} = 1.6 \times \text{IC} \times \text{CC} \quad \text{so} \quad \frac{\text{CPU}_{\text{old}}}{\text{CPU}_{\text{new}}} = \frac{2.2}{1.6} \text{ means 37.5% faster}
\]

- How does this compare with using branch prediction to shave a cycle off the branch time?

\[
\text{CPU time new} = 2.0 \times \text{IC} \times \text{CC} \quad \text{so} \quad \frac{\text{CPU}_{\text{old}}}{\text{CPU}_{\text{new}}} = \frac{2.2}{2.0} \text{ means 10% faster}
\]

- What if two ALU instructions could be executed at once?

\[
\text{CPU time new} = 1.95 \times \text{IC} \times \text{CC} \quad \text{so} \quad \frac{\text{CPU}_{\text{old}}}{\text{CPU}_{\text{new}}} = \frac{2.2}{1.95} \text{ means 12.8% faster}
\]
Performance Summary

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, $T_{clk}$

CPU Time = \( \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}} \)

- CPI
- $T_{clk}$
MIPS as a Performance Metric

MIPS: Millions of Instructions Per Second

- **Pitfall**: Doesn’t account for
  - Differences in ISAs between computers
  - Differences in complexity between instructions

\[
\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} = \frac{\text{Instruction count}}{\text{Instruction count} \times \text{CPI} \times \text{Clock rate}} \times 10^6 = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}
\]

- **CPI** varies between programs on a given CPU
Amdahl’s Law

**Pitfall:** Improving an aspect of a computer and expecting a proportional improvement in overall performance

\[
T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}
\]

\[= \left( \frac{f}{\text{improvement factor}} + (1-f) \right) T_{\text{original}} \]

**Speedup:**

\[
\text{Speedup} = \frac{T_{\text{original}}}{T_{\text{improved}}} = \frac{1}{\frac{f}{\text{improvement factor}} + (1-f)}
\]

**Corollary:** make the common case fast!
SPEC CPU Benchmark

• Programs used to measure performance
  – Supposedly typical of actual workload

• Standard Performance Evaluation Corp (SPEC)
  – Develops benchmarks for CPU, I/O, Web, …

• SPEC CPU2006
  – Elapsed time to execute a selection of programs
    • Negligible I/O, so focuses on CPU performance
  – Normalize relative to reference machine
  – Summarize as geometric mean of performance ratios
    • CINT2006 (integer) and CFP2006 (floating-point)

\[ \sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i} \]
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>IC $\times 10^9$</th>
<th>CPI</th>
<th>$T_{clk}$ (ns)</th>
<th>Exec time</th>
<th>Ref time</th>
<th>SPECratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>perl</td>
<td>Interpreted string processing</td>
<td>2,118</td>
<td>0.75</td>
<td>0.40</td>
<td>637</td>
<td>9,777</td>
<td>15.3</td>
</tr>
<tr>
<td>bzip2</td>
<td>Block-sorting compression</td>
<td>2,389</td>
<td>0.85</td>
<td>0.40</td>
<td>817</td>
<td>9,650</td>
<td>11.8</td>
</tr>
<tr>
<td>gcc</td>
<td>GNU C Compiler</td>
<td>1,050</td>
<td>1.72</td>
<td>0.47</td>
<td>24</td>
<td>8,050</td>
<td>11.1</td>
</tr>
<tr>
<td>mcf</td>
<td>Combinatorial optimization</td>
<td>336</td>
<td>10.00</td>
<td>0.40</td>
<td>1,345</td>
<td>9,120</td>
<td>6.8</td>
</tr>
<tr>
<td>go</td>
<td>Go game (AI)</td>
<td>1,658</td>
<td>1.09</td>
<td>0.40</td>
<td>721</td>
<td>10,490</td>
<td>14.6</td>
</tr>
<tr>
<td>hmmer</td>
<td>Search gene sequence</td>
<td>2,783</td>
<td>0.80</td>
<td>0.40</td>
<td>890</td>
<td>9,330</td>
<td>10.5</td>
</tr>
<tr>
<td>sjeng</td>
<td>Chess game (AI)</td>
<td>2,176</td>
<td>0.96</td>
<td>0.48</td>
<td>37</td>
<td>12,100</td>
<td>14.5</td>
</tr>
<tr>
<td>libquantum</td>
<td>Quantum computer simulation</td>
<td>1,623</td>
<td>1.61</td>
<td>0.40</td>
<td>1,047</td>
<td>20,720</td>
<td>19.8</td>
</tr>
<tr>
<td>h264avc</td>
<td>Video compression</td>
<td>3,102</td>
<td>0.80</td>
<td>0.40</td>
<td>993</td>
<td>22,130</td>
<td>22.3</td>
</tr>
<tr>
<td>omnetpp</td>
<td>Discrete event simulation</td>
<td>587</td>
<td>2.94</td>
<td>0.40</td>
<td>690</td>
<td>6,250</td>
<td>9.1</td>
</tr>
<tr>
<td>astar</td>
<td>Games/path finding</td>
<td>1,082</td>
<td>1.79</td>
<td>0.40</td>
<td>773</td>
<td>7,020</td>
<td>9.1</td>
</tr>
<tr>
<td>xalancbmk</td>
<td>XML parsing</td>
<td>1,058</td>
<td>2.70</td>
<td>0.40</td>
<td>1,143</td>
<td>6,900</td>
<td>6.0</td>
</tr>
</tbody>
</table>

Geometric mean: 11.7

High cache miss rates
Comparing Relative Performance

Guiding principle in reporting performance measurements is reproducibility. List everything another experimenter would need to duplicate the experiment:

- version of the operating system
- compiler settings
- input set used
- specific computer configuration:
  - clock rate, cache and memory sizes and speed, etc.

Benchmark set revised periodically:
- Designers target performance specifically for common benchmarks
Other Performance Metrics

Power consumption – especially in the embedded market where battery life is important – For power-limited applications, the most important metric is energy efficiency
Next Class

• Assembly Instructions

• Instruction Set Architecture (ISA)

• MIPS ISA
Computer Fundamentals
Metrics and Performance

Computer Organization
Architectures for Embedded Computing

Friday, 20 September 13