Computing abstractions

- Application software
  - Written in high-level language
- 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
From high-level language to machine language

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;
}
```

Compiler

Assembly language program (for MIPS)

```assembly
swap:
    muli $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
```

Assembler

Binary machine language program (for MIPS)

```
0000000001010000100000000000001100
0000000001010000100000000000001100
1000110001100100100000000000000000
1000110011110010000000000000000010
1010110011110010000000000000000000
1010110001100100100000000000000000
00000011110000000000000000100
```
What is performance?

- **Response time**
  - How long it takes to execute 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?
Relative performance

- Define Performance = 1/Execution Time
- “X is \( n \) time faster than Y”

\[
\text{Performance}_X / \text{Performance}_Y = \text{Execution time}_Y / \text{Execution time}_X = n
\]

- Example: time taken to run a program
  - 20s on A, 10s on B
  - Execution Time\(_B \) / Execution Time\(_A \) = 20s / 10s = 2
  - So A is 2\( \times \) faster than B
What can be done to improve the performance?

Improvement to performance can be attained through:

- better algorithms
- better compilers
- **better architectures**
- better organization and design

**focus of this course**
Measuring execution time

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

- Processor time
  - Time spent processing a given job
    - Discounts I/O time, other jobs’ shares
  - Comprises user processor time and system CPU time
  - Different programs are affected differently by processor and system performance
Processor clocking

- Operation of digital hardware governed by a constant-rate clock

- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s

- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = 4.0×10^9Hz
Factors that determine execution time

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

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, \( T_c \)
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?

\[
\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\
= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps} \\
\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\
= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps} \\
\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2
\]
Average CPI

- 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
    \[= 2 \times 1 + 1 \times 2 + 2 \times 3\]
    \[= 10\]
  - Avg. CPI = 10/5 = 2.0

- Sequence 2: IC = 6
  - Clock Cycles
    \[= 4 \times 1 + 1 \times 2 + 1 \times 3\]
    \[= 9\]
  - Avg. CPI = 9/6 = 1.5
Bounding max improvement: Amdahl’s law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

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

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

\[
20 = \frac{80}{n} + 20
\]

- **Corollary:** make the common case fast