TEXAS TECH UNIVERSITY
Department of Computer Science

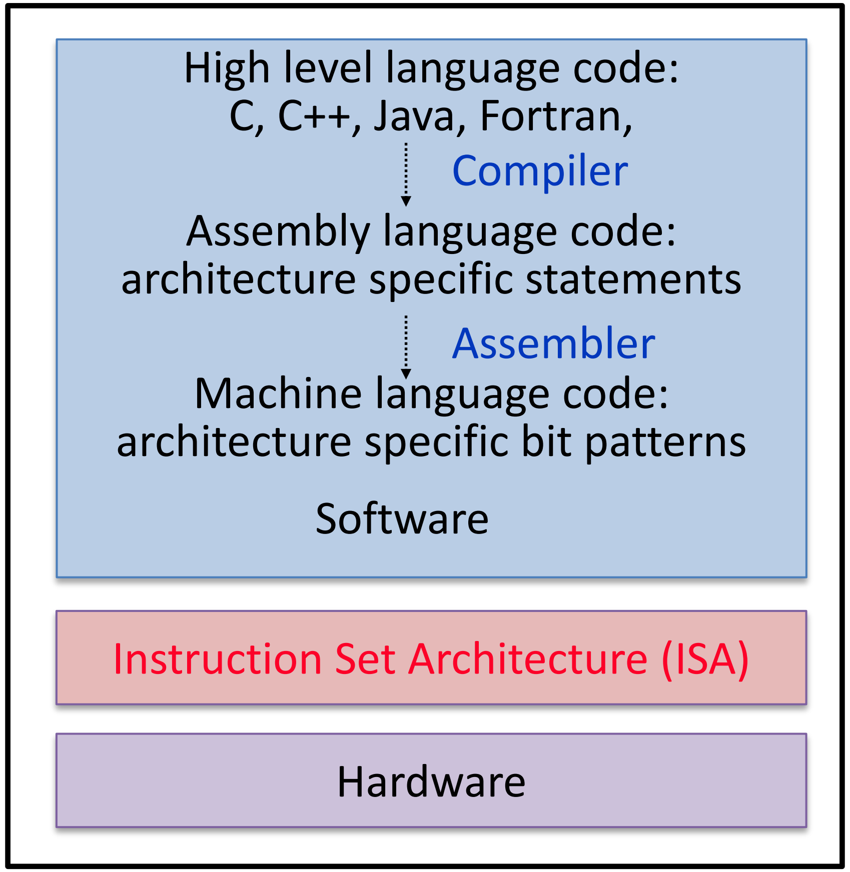
# CS5375 Computer Systems Organization and Architecture

## Lecture 3

Instructor: Yong Chen, Ph.D.
Department of Computer Science
Texas Tech University
Yong.Chen@ttu.edu, 806-834-0284

# Review of Last Lecture

- Computer performance improvements primarily come from two perspectives
  - Semiconductor technology and computer architecture improvements
- Trends in Architecture: computer architecture has moved to multi-processor architecture (multicore/manycore)
  - Limit of single-core processor, e.g power consumption is being reached (power wall)
- New models for performance:
  - Exploits parallelism
  - Flynn’s Taxonomy: SISD, SIMD, MISD, MIMD
- Defining Computer Architecture
  - Instruction Set Architecture:
    - An abstract specification, can have many implementations
    - ISA creates its own software ecosystem



# Outline

- Trends in Technology
- Measuring, Reporting and Summarizing Performance

# Trends in Technology

- Integrated circuit technology
  - Transistor density: 35%/year
  - Die (chip) size: 10-20%/year
  - Integration overall: 40-55%/year or doubling every 18 to 24 months (Moore's law) (we are also approaching post Moore's law era)

- DRAM capacity: 25-40%/year (slowing)

- Flash capacity: 50-60%/year
  - 8-10X cheaper/bit than DRAM

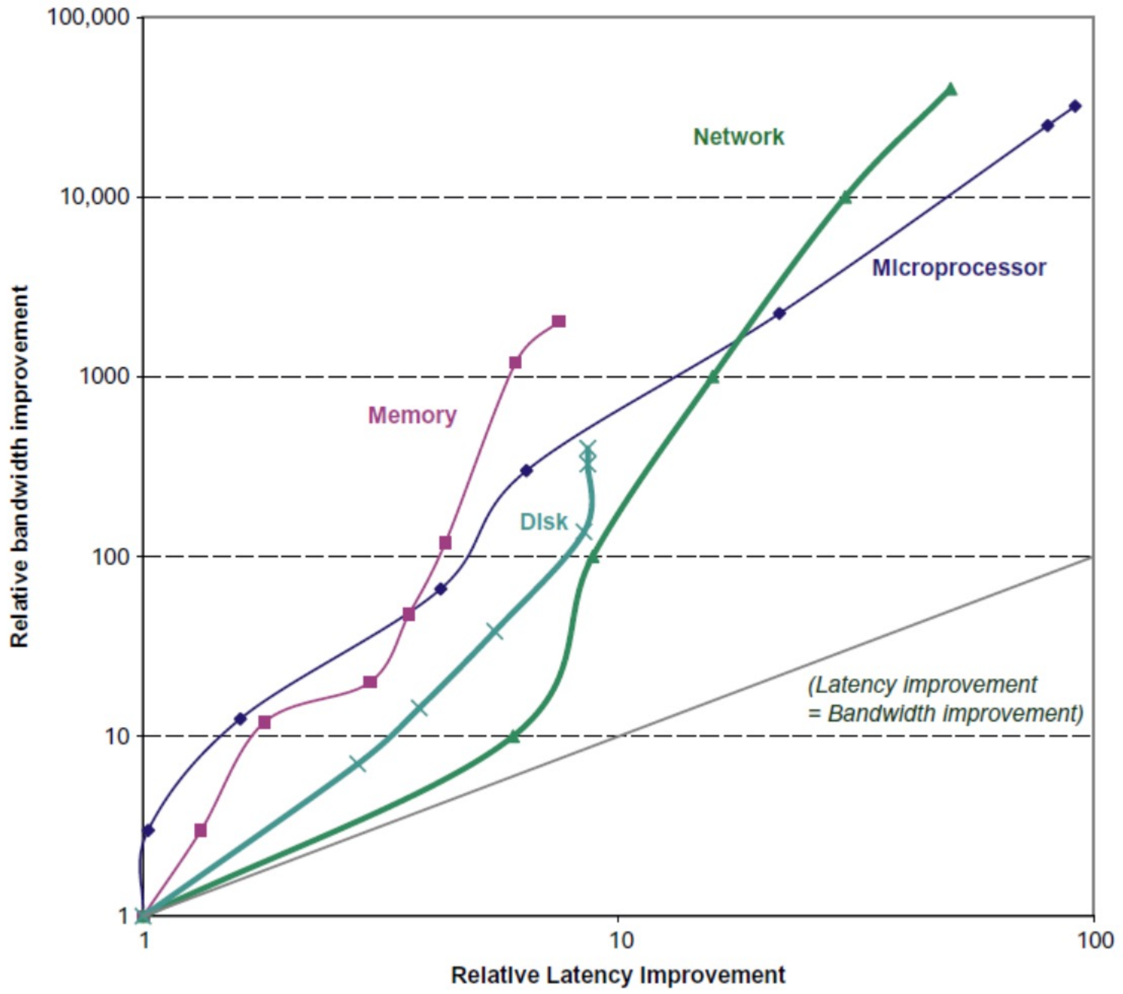
- Magnetic disk technology: 40%/year, recently slowed to 5%/year
  - Density increases may no longer be possible, maybe increase from 7 to 9 platters
  - 8-10X cheaper/bit than Flash
  - 200-300X cheaper/bit than DRAM

# Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time (e.g., MB/s for disk transfer)
  - 32,000-40,000X improvement for processors
  - 300-1200X improvement for memory and disks

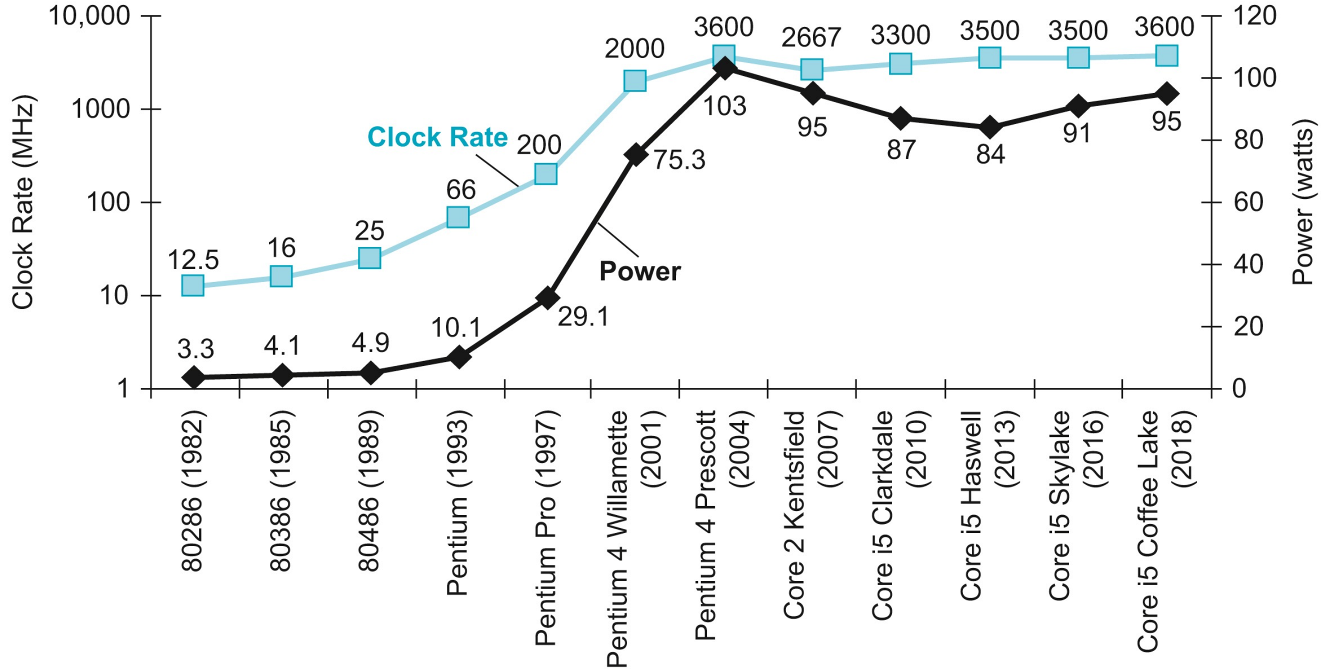
- Latency or response time
  - Time between start and completion of an event (e.g., milliseconds for a disk access)
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks

# Bandwidth and Latency




Log-log plot of bandwidth and latency milestones

# Power Wall




Clock rate and power for Intel x86 microprocessors over nine generations and 36 years.

- In CMOS (Complementary Metal Oxide Semiconductor) IC technology

Power: 1/2 X Capacitive load X Voltage$^2$ X Frequency

×40

5V → 1V

×1000

# Power Wall (cont.)

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction
  - What's the new power required?

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}{}^2 \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further (otherwise causing transistors too leaky)
  - We can't remove more heat
  - Battery life and energy bills

- How else can we improve performance?

Uniprocessor (unicore processor) ->
Multiprocessor (multicore processor)

# Multiprocessors

- Multicore processors
  - More than one processor (core) per chip

- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer (free ride)
  - Hard to do (free ride is over; requires programmer's extensive efforts)
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

# Reducing Power

- Techniques for reducing power:
  - Turn off the clock of inactive modules (e.g., floating point units)
  - Dynamic Voltage-Frequency Scaling (DVFS)
  - Design for typical case: e.g., low power state for DRAM, disks
  - Overclocking of a single core (e.g., Intel Turbo mode) for single-threaded code while turning off other cores

# Outline

- Trends in Technology
- Measuring, Reporting and Summarizing Performance

# Performance

- Define (Absolute) Performance = 1/Execution Time
- Define Relative Performance: "Computer X is $n$ time faster than Computer Y" (Speedup of X relative to Y)

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

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

"improve performance" == "increase performance"
"improve execution time" == "decrease execution time"

# Measuring Execution Time

- Elapsed time (or wall-clock time, or response time)
  - The total time required for the computer to complete a task
  - Counts everything: disk and memory accesses, I/O, OS (operating system) overhead, CPU execution time, etc.
  - A useful number, but often not good for comparison purposes

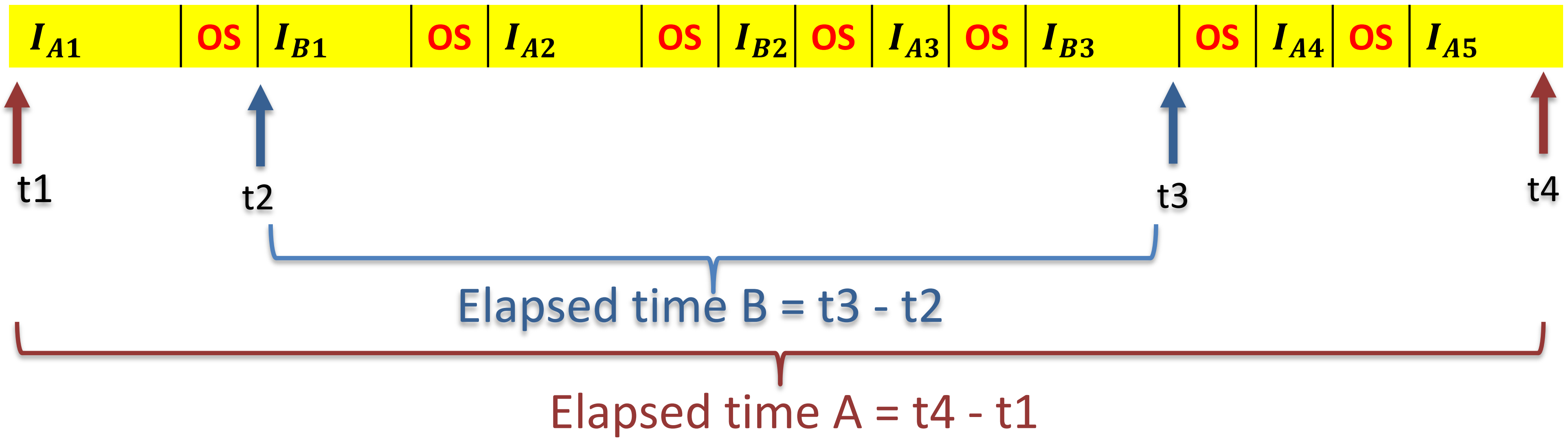
- CPU execution time (or CPU time)
  - The actual time the CPU spends computing for a specific task
    - Does not count I/O time, or time spent running other programs
  - Can be broken up into user CPU time and system CPU time
    - user CPU time: the CPU time spent in a program itself
    - system CPU time: the CPU time spent in the OS performing tasks on behalf of the program
  - Different programs are affected differently by CPU and system performance

Try out the *time* command on Unix/Linux system

# Elapsed time

| Program | A | |
|---|---|---|
| $I_{A1}$ | Read A | (IO) |
| $I_{A2}$ | Read B | (IO) |
| $I_{A3}$ | B=B-2 | (CPU) |
| $I_{A4}$ | A=A+B | (CPU) |
| $I_{A5}$ | Print A | (IO) |

| Program | B | |
|---|---|---|
| $I_{B1}$ | Read A | (IO) |
| $I_{B2}$ | A= A X 2 | (CPU) |
| $I_{B3}$ | Print A | (IO) |



# CPU Clocking

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

- Clock period (clock cycle time): the length/duration of a clock cycle
  - e.g., 250ps = 0.25ns = $250 \times 10^{-12}$s
  - Also called tick, clock tick, clock, cycle, clock cycle

- Clock frequency (clock rate): cycles per second (Hz)
  - e.g., 4.0GHz = $4.0 \times 10^9$Hz => $4.0 \times 10^9$ cycles per second => $0.25 * 10^{-9}$ seconds per cycle

- Clock frequency (or clock rate) is the inverse of the clock period (or clock cycle time)

- Instead of reporting execution time in seconds, we often use cycles (or the number of clock cycles)

# CPU Time

- CPU execution time for a program can be written as:

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing the number of clock cycles
  - Reducing the clock cycle time
  - Increasing the clock rate

**Question:** How to improve performance?
[*Reducing/Increasing*?] the number of required clock cycles for a program
[*Reducing/Increasing*?] the clock cycle time
[*Reducing/Increasing*?] the clock rate

- Hardware designer often faces a trade-off between the number of clock cycles needed for a program and the length of each cycle
  - Many techniques that decrease the number of clock cycles may also increase the clock cycle time

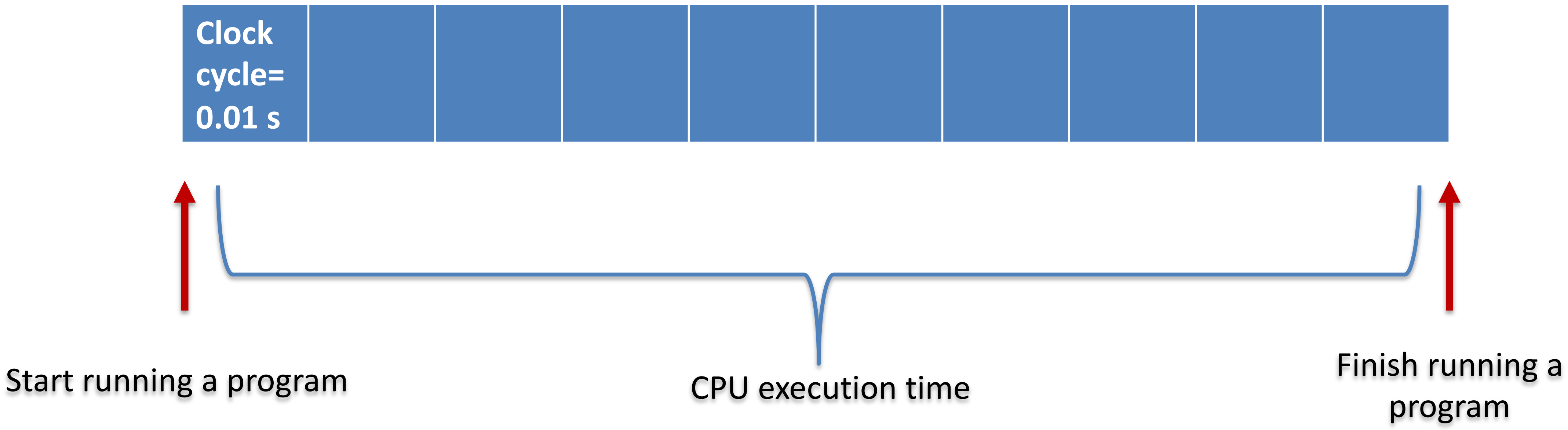
# Example

CPU clock cycle time = 0.01 s

# CPU clock cycle = 10

CPU execution time = 10 x 0.01 =0.1 s

Clock Rate = 100



# CPU Time Example

- Suppose our favorite program takes 10s CPU time on Computer A with 2GHz clock
- To design a new Computer B
  - Aim for 6s CPU time
  - Can do faster clock, but causes 1.2 × clock cycles
- How fast must Computer B clock be?

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

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10\text{s} \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

# Another View of CPU Time

- Another view of the execution time is:
  - The number of instructions executed multiplied by the average time per instruction
- In other words

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

- Instruction Count
  - The number of instructions executed by a program
  - Determined by program, ISA and compiler
- CPI (Cycles Per Instruction)
  - The average number of clock cycles per instruction for a program
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

CPI can be less than 1.0, as some processors fetch and execute multiple instructions per cycle (i.e. "multiple issue"). Some designers invert CPI to IPC (Instructions Per Cycle), e.g., 0.5 CPI == 2.0 IPC.

# The Classic CPU Performance Equation

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- These formulas are particularly useful because they indicate three key factors that affect performance
  - Instruction Count, CPI, Clock Cycle Time or Clock Rate

# CPI Example

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

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$

$$= \text{I} \times 2.0 \times 250\text{ps} = \text{I} \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$

$$= \text{I} \times 1.2 \times 500\text{ps} = \text{I} \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{\text{I} \times 600\text{ps}}{\text{I} \times 500\text{ps}} = 1.2$$

…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

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

# CPI Example

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

| Class | A | B | C |
|---|---|---|---|
| CPI for class | 1 | 2 | 3 |
| IC (Instructions Count) in sequence 1 | 2 | 1 | 2 |
| IC (Instructions Count) in sequence 2 | 4 | 1 | 1 |

- Sequence 1:
  Instructions Count? = 5

  Clock Cycles? $= 2\times1 + 1\times2 + 2\times3$
  $= 10$

  Average CPI? $= 10/5 = 2.0$

- Sequence 2:
  Instructions Count? = 6

  Clock Cycles? $= 4\times1 + 1\times2 + 1\times3$
  $= 9$

  Average CPI? $= 9/6 = 1.5$

# Performance Summary

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

$$= \text{Seconds / Program}$$

- Performance depends on
  - Algorithm: affects IC (Instructions Count), CPI (by favoring slower or faster instructions)
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Architecture/Instruction Set Architecture (ISA): affects IC, CPI, $T_c$ (Clock Cycle Time)

# Measuring Performance

- To evaluate and compare two computers, the workload matters (when comparing the execution time on two computers)

- Workload
  - A set of programs run on a computer that is either the actual collection of applications run by a user or constructed from real programs to approximate such a mix

- Benchmarks
  - A set of programs (standard, agreed upon) chosen specifically to measure performance
  - From a user’s perspective, need to know which programs are commonly run, so that the right benchmarks can be chosen to predict the performance of the actual workload

# Measuring Performance (cont.)

- Benchmarks
  - Kernels: small, key pieces of real applications, e.g. matrix multiply
  - Toy programs: small or tiny real programs, e.g. sorting
  - Synthetic benchmarks: “fake” programs invented to match the profile and behavior of real applications
  - Benchmark suites: a collection of benchmark applications, e.g. SPEC (Standard Performance Evaluation Corporation) CPU2017, TPC-C (Transaction-Processing Council)

# SPEC CPU Benchmark

- SPEC CPU2017 benchmark suite
  - https://www.spec.org/benchmarks.html
  - A set of Integer (called SPECspeed® 2017 Integer) and Floating-Point (SPECspeed® 2017 Floating Point) benchmarks for comparing the execution time
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

Where Execution time ratio$_i$ is the execution time, normalized to the reference computer, for the *i*-th program of a total of *n* in the workload, and

$$\prod_{i=1}^{n} a_i \text{ means the product } a_1 \times a_2 \times \ldots \times a_n$$

# SPECspeed 2017 Integer benchmarks on a 1.8 GHz Intel Xeon E5-2650L

SPECRatio = Execution Time$_{reference}$ / Execution Time$_{target}$

| Description | Name | Instruction Count x $10^9$ | CPI | Clock cycle time (seconds x $10^{-9}$) | Execution Time (seconds) | Reference Time (seconds) | SPECratio |
|---|---|---|---|---|---|---|---|
| Perl interpreter | perlbench | 2684 | 0.42 | 0.556 | 627 | 1774 | 2.83 |
| GNU C compiler | gcc | 2322 | 0.67 | 0.556 | 863 | 3976 | 4.61 |
| Route planning | mcf | 1786 | 1.22 | 0.556 | 1215 | 4721 | 3.89 |
| Discrete Event simulation - computer network | omnetpp | 1107 | 0.82 | 0.556 | 507 | 1630 | 3.21 |
| XML to HTML conversion via XSLT | xalancbmk | 1314 | 0.75 | 0.556 | 549 | 1417 | 2.58 |
| Video compression | x264 | 4488 | 0.32 | 0.556 | 813 | 1763 | 2.17 |
| Artificial Intelligence: alpha-beta tree search (Chess) | deepsjeng | 2216 | 0.57 | 0.556 | 698 | 1432 | 2.05 |
| Artificial Intelligence: Monte Carlo tree search (Go) | leela | 2236 | 0.79 | 0.556 | 987 | 1703 | 1.73 |
| Artificial Intelligence: recursive solution generator (Sudoku) | exchange2 | 6683 | 0.46 | 0.556 | 1718 | 2939 | 1.71 |
| General data compression | xz | 8533 | 1.32 | 0.556 | 6290 | 6182 | 0.98 |
| Geometric mean | – | – | – | – | – | – | 2.36 |

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

# Readings

- Chapter 1, 1.4-1.9