CENG311 Computer Architecture

Computer Abstractions

IZTECH, Fall 2023

05 October 2023

Computer Architecture

Computer architecture is the science and art of selecting and interconnecting hardware components to create a computer that meets functional, performance and cost goals

Why Study Computer Architecture?

Make computers faster, cheaper, smaller, more reliable

Enable new applications

- Deep neural networks
- Self driving cars

Enable better solutions to problems

- Performance improvement

Understand why computers work the way they do

Computer Systems

Personal computers

General purpose, variety of software

Supercomputers

High-end scientific and engineering calculations

Embedded computers

Hidden as components of systems

Personal mobile devices

Smart phones, tablets, electronic glasses

Computer System Design Goals

Functional

- Needs to be correct

High performance

Low cost

Reliable

- Continue to perform correctly

Low power/energy

Secure

Computer System Structure

Application software

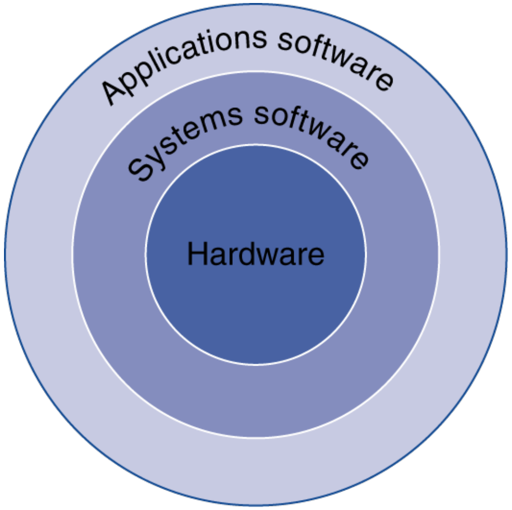
- Written in high-level language (C, Java)

System software

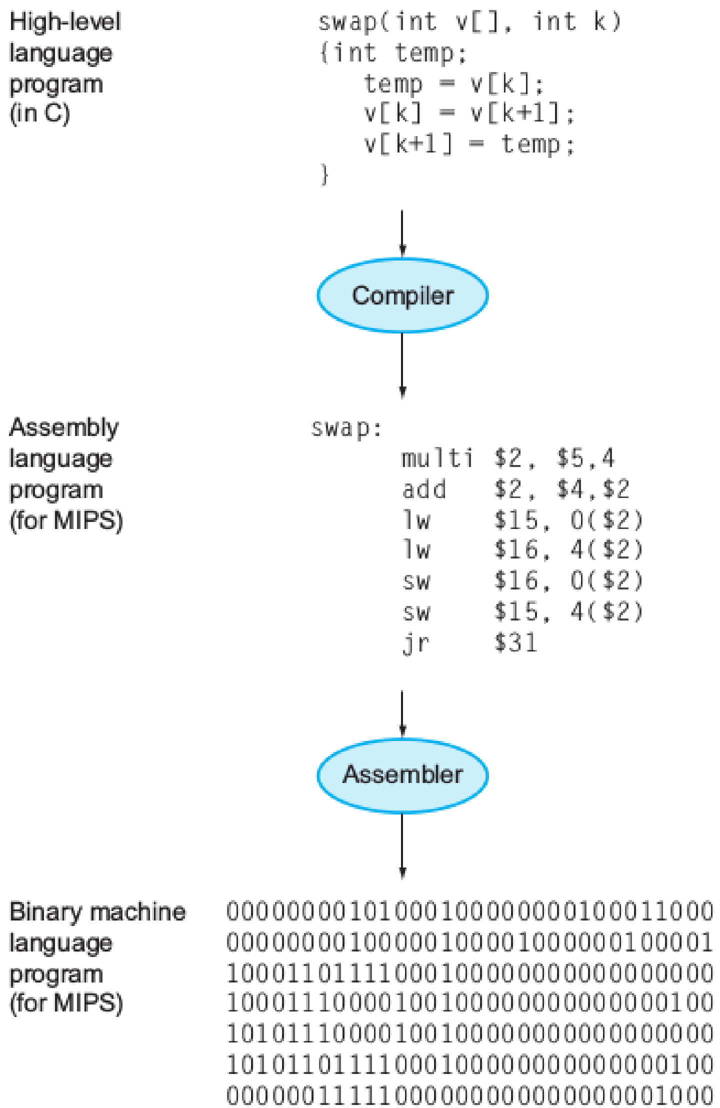
- Compiler: translates HLL code to machine code
- Operating system
  - Handling input/output
  - Managing memory and storage
  - Scheduling tasks & sharing resources

Hardware

- Processor, memory, I/O controllers



From a High-Level Language to the Language of Hardware



Computer Architect's View

Systems Prog.

"C" as a model of computation

Programmer's view of how a computer system works

How does an assembly program end up executing as digital logic?

What happens in-between?

How is a computer designed using logic gates and wires to satisfy specific goals?

Architect/microarchitect's view: How to design a computer that meets system design goals.

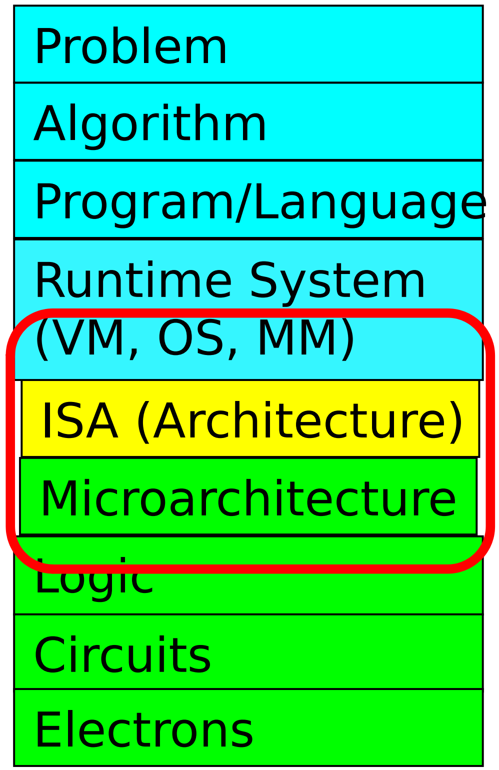
Choices critically affect both the SW programmer and the HW designer

HW designer's view of how a computer system works

Digital logic as a model of computation

Digital Design

Levels of Transformation



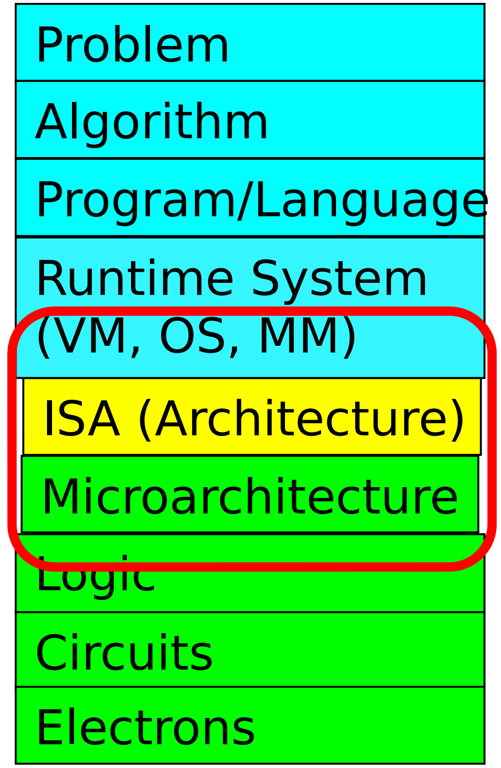
Y. Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.

Levels of Transformation

Algorithm

Step-by-step procedure that is **guaranteed to terminate** where **each step is precisely stated** and **can be carried out by a computer**

Many algorithms for the same problem



ISA
(Instruction Set Architecture)

Interface/contract between SW and HW

What the programmer assumes hardware will satisfy

Microarchitecture
An implementation of the ISA

Digital logic circuits
Building blocks of micro-arch (e.g., gates)

The Power of Abstraction

Levels of transformation create abstractions

Interface to the lower level, not how the lower level is implemented

e.g., high-level language programmer does not really need to know what the ISA is and how a computer executes instructions

Abstraction improves productivity

No need to worry about decisions made in underlying levels

e.g., programming in Java vs. C vs. assembly vs. binary vs. by specifying control signals of each transistor every cycle

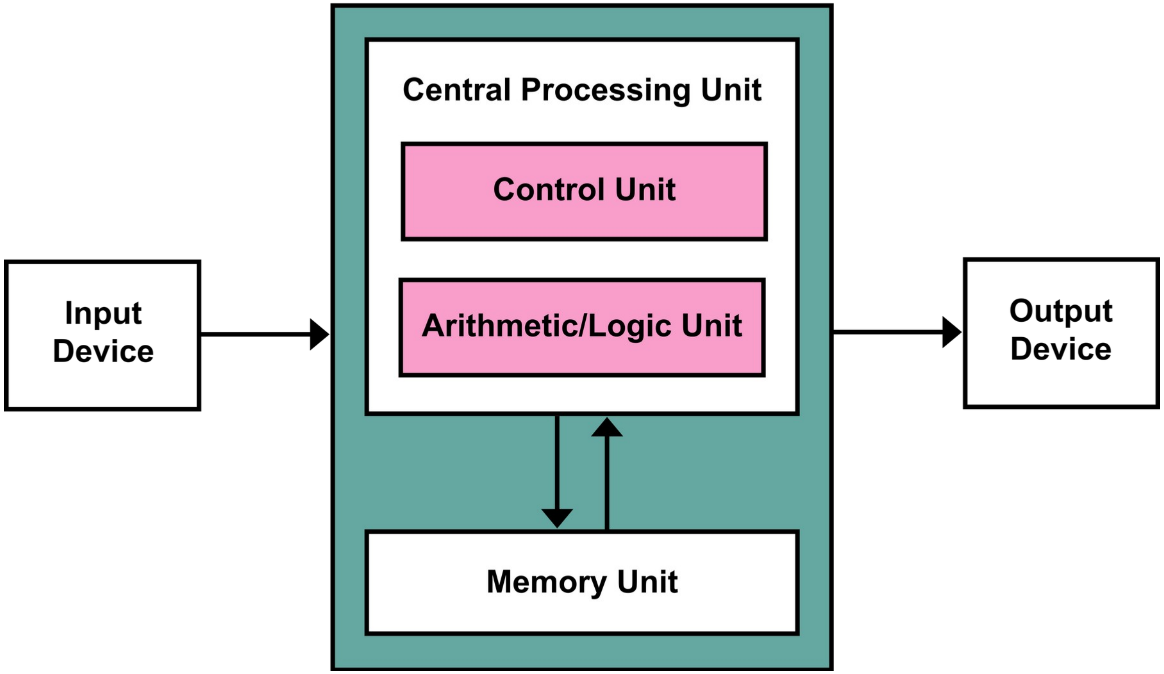
Crossing the Abstraction Layers

To understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer

To enable you to be comfortable in making design and optimization decisions that cross the boundaries of different layers and system components

The von Neumann Architecture

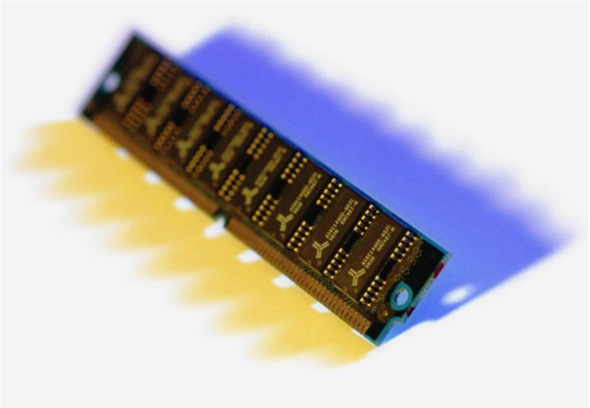
- Main memory
- Central processing unit (CPU)
- Interconnection



Main Memory

Collection of locations, each of which is capable of storing both instructions and data

Every location consists of an address, which is used to access the location, and the contents of the location

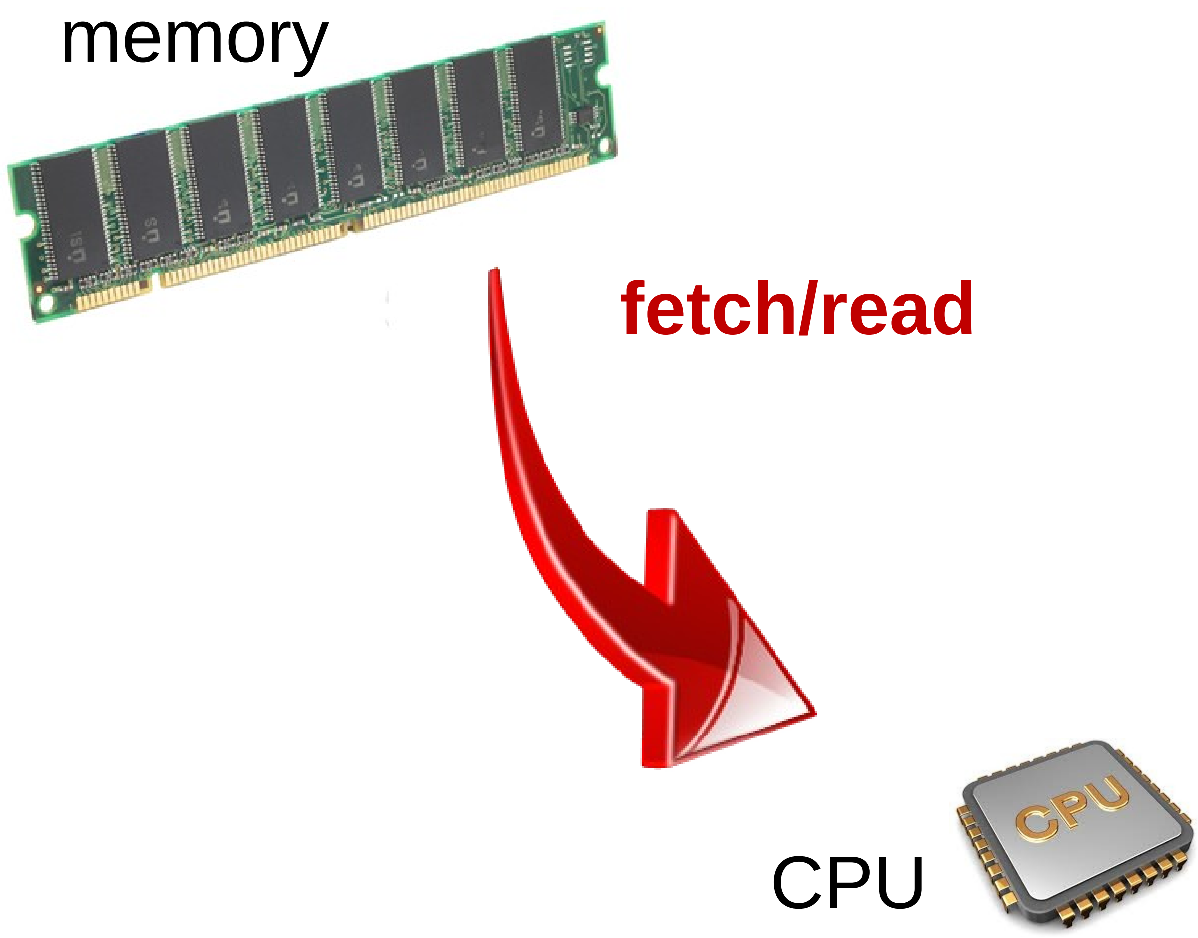


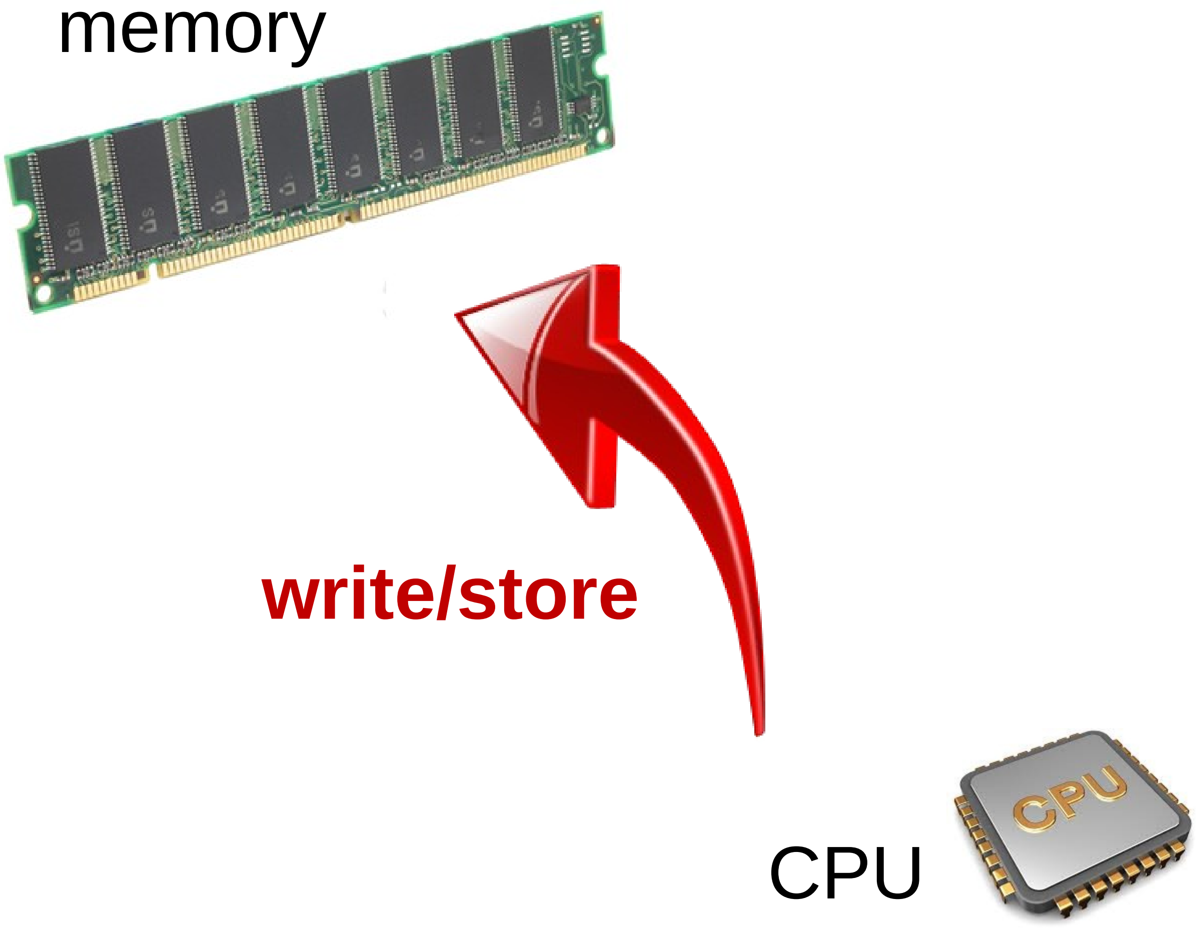
Central Processing Unit (CPU)

Control unit : responsible for deciding which instruction in a program should be executed

Arithmetic and logic unit (ALU) : responsible for executing the actual instructions

Register : quickly accessible location available to CPU


memory
fetch/read
CPU


memory
write/store
CPU
CPU

Example Program

Compute sin(x) using taylor expansion: sin(x) = x – x^3/3! + x^5/5! + x^7/7! + ...

```
void sinx(int N, int terms, float* x, float* result)
{
   for (int i=0; i<N; i++)
   {
      float value = x[i];
      float numer = x[i] * x[i] * x[i];
      int denom = 6;  // 3!
      int sign = -1;

      for (int j=1; j<=terms; j++)
      {
         value += sign * numer / denom
         numer *= x[i] * x[i];
         denom *= (j+3) * (j+4);
         sign *= -1;
      }

      result[i] = value;
   }
}
```

Compile Program

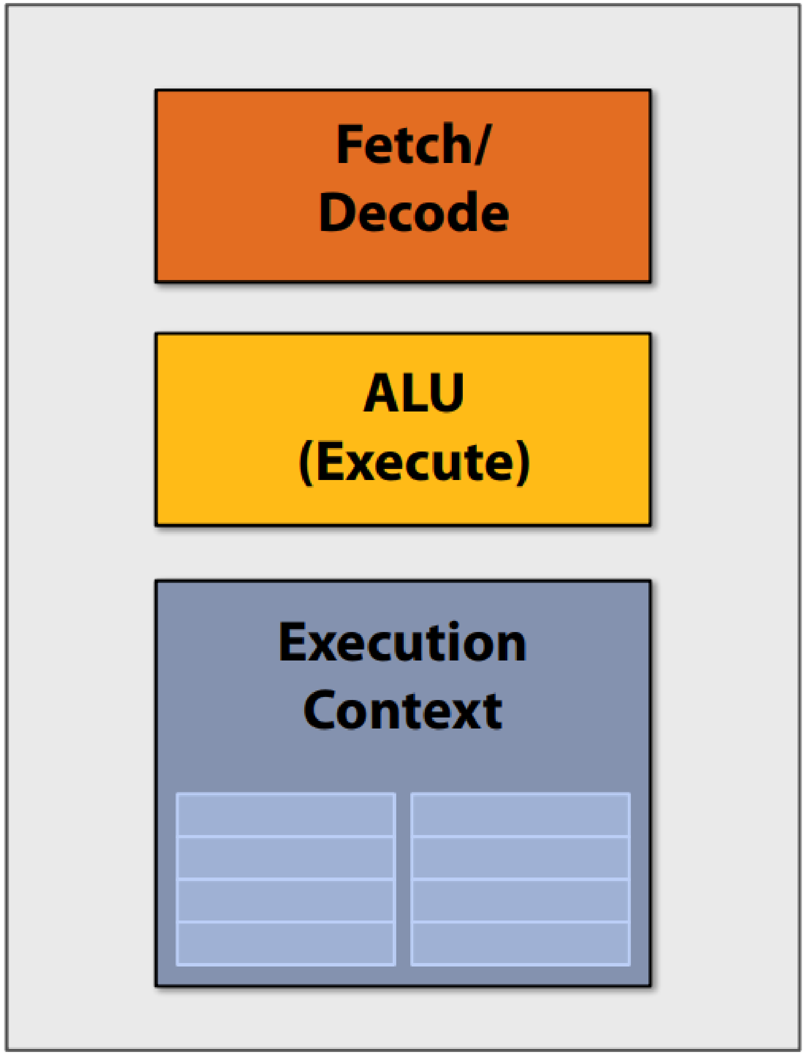
```
void sinx(int N, int terms, float* x, float* result)
{
   for (int i=0; i<N; i++)
   {
      float value = x[i];
      float numer = x[i] * x[i] * x[i];
      int denom = 6;  // 3!
      int sign = -1;

      for (int j=1; j<=terms; j++)
      {
         value += sign * numer / denom
         numer *= x[i] * x[i];
         denom *= (j+3) * (j+4);
         sign *= -1;
      }

      result[i] = value;
   }
}
```

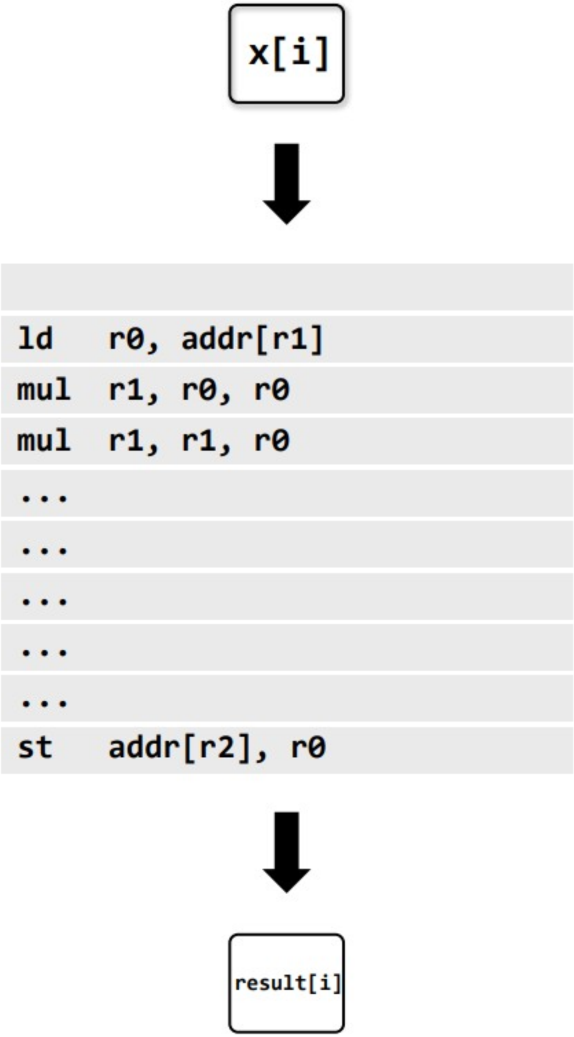
x[i]

```
ld    r0, addr[r1]
mul   r1, r0, r0
mul   r1, r1, r0
...
...
...
...
...
st    addr[r2], r0
```

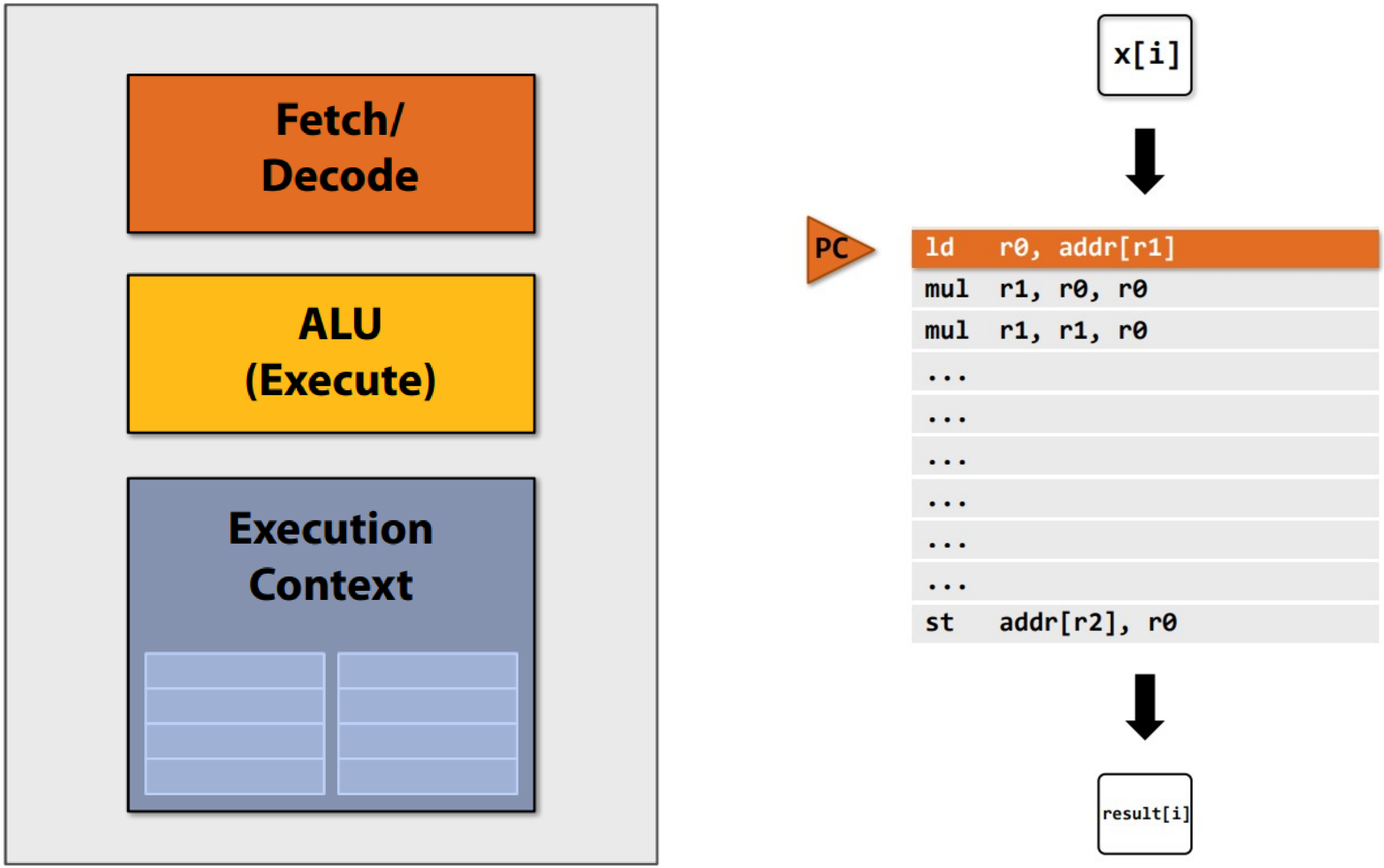
result[i]

Execute Program

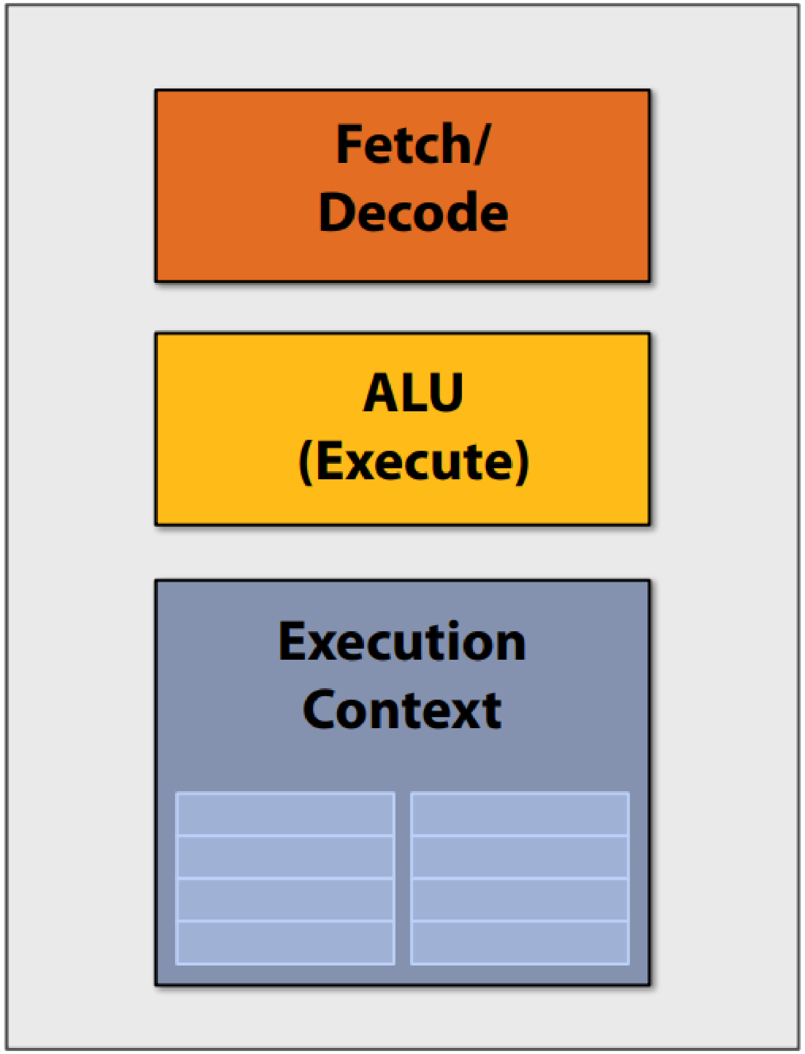


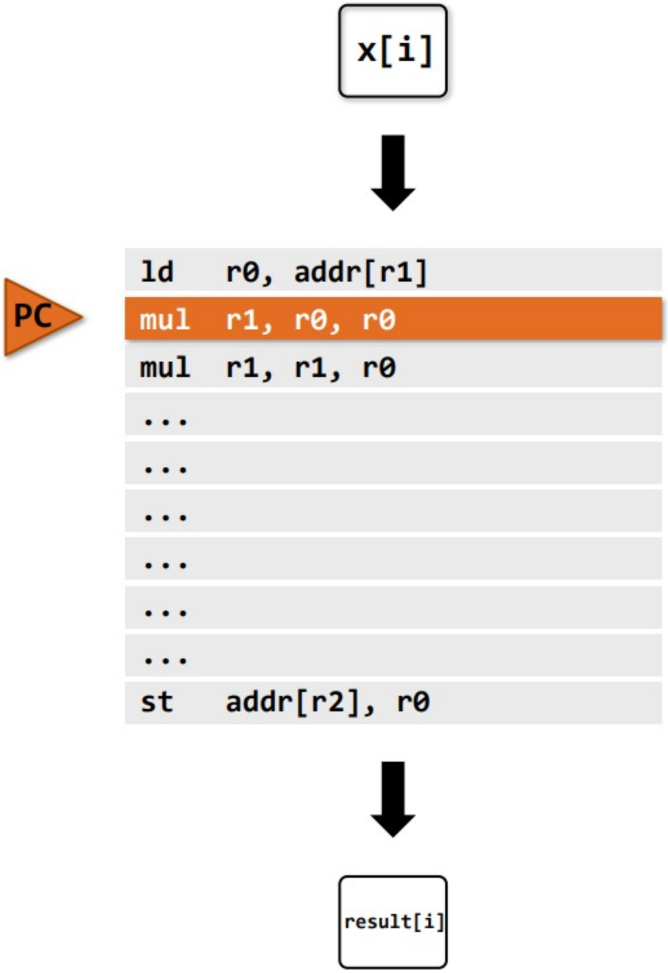


Execute One Instruction per clock

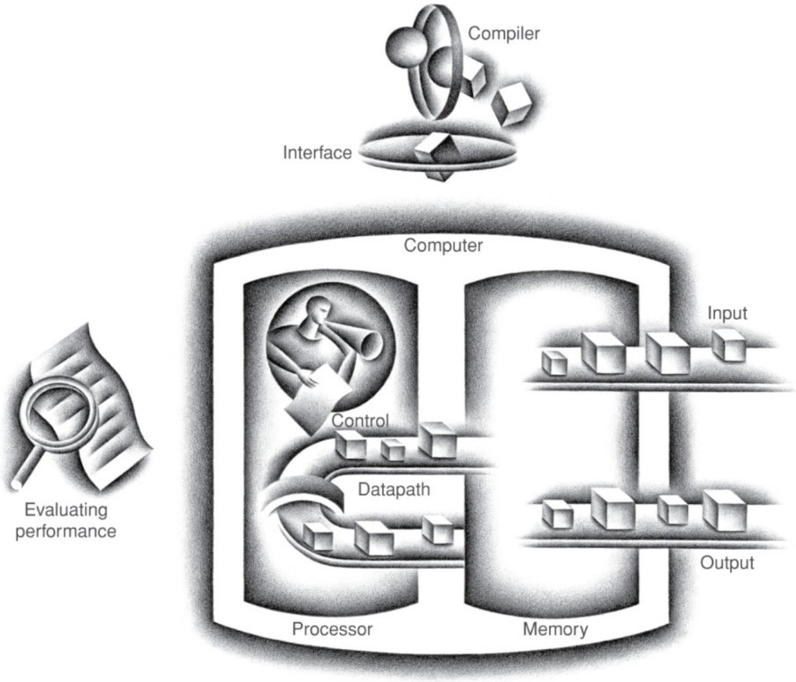


Execute One Instruction per clock

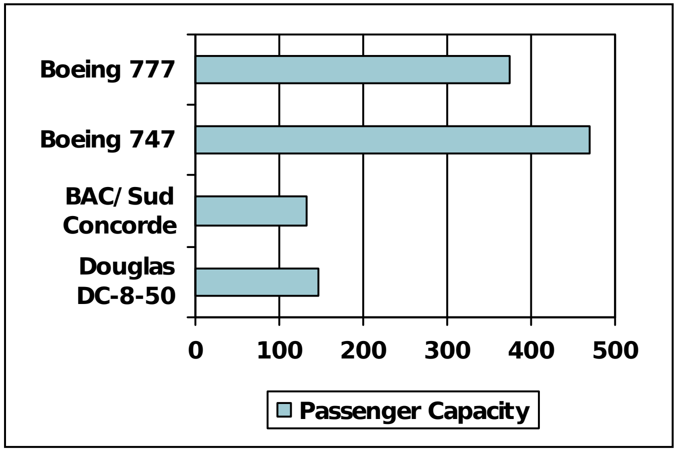


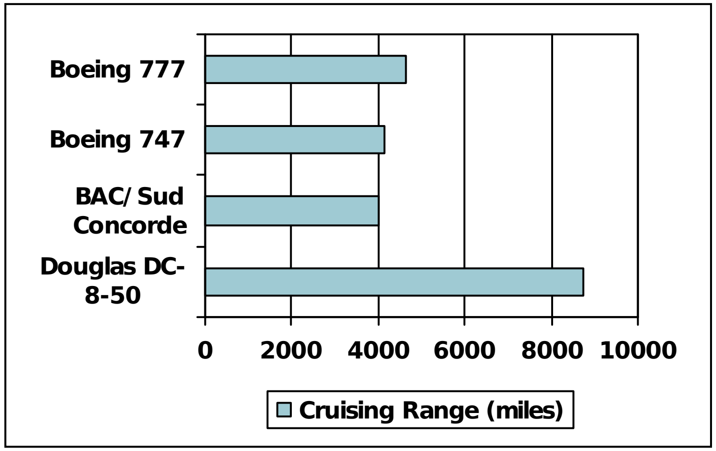


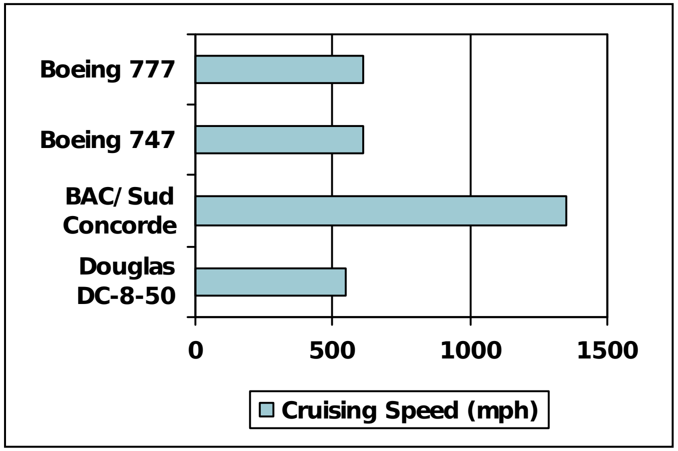
Computer System View

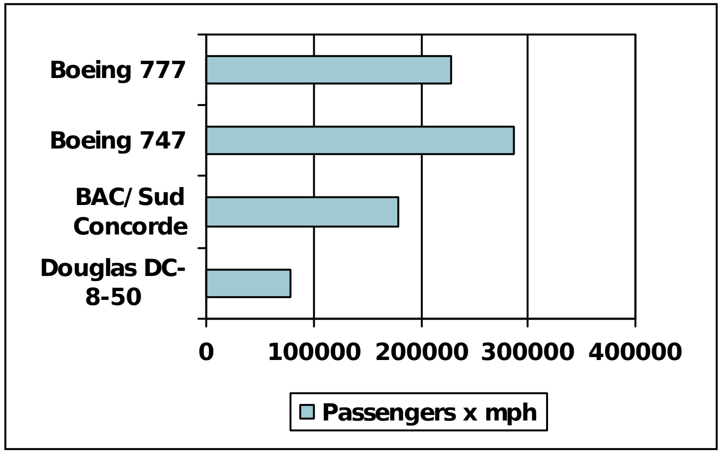


Defining Performance









Latency vs Throughput

Latency

Elapsed time (How long it takes to do a task)

Throughput

Total work done per unit time

e.g., tasks/transactions/... per hour

Example: move people 10 miles

Car: capacity=5, speed=60 miles/hour

Bus: capacity=60, speed=20 miles/hour

Latency: car=10 min, bus=30 min

Throughput: car=15 PPH (count return), bus=60 PPH

Processor Performance

When discussing processor performance, we will focus primarily on execution time for a single job (Latency)

Measuring Execution Time

Elapsed time ~ Execution 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

Performance = 1 / Execution Time

Relative Performance

$\text{Performance}_X = 1 / \text{Execution time}_X$

$\text{Performance}_X > \text{Performance}_Y$

$1 / \text{Execution time}_X > 1 / \text{Execution time}_Y$

$\text{Execution time}_Y > \text{Execution time}_X$

"X is *n* times faster than Y"

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

Relative Performance Example

If computer A runs a program in 10s, and computer B runs the same program in 15s, how much faster is A than B?

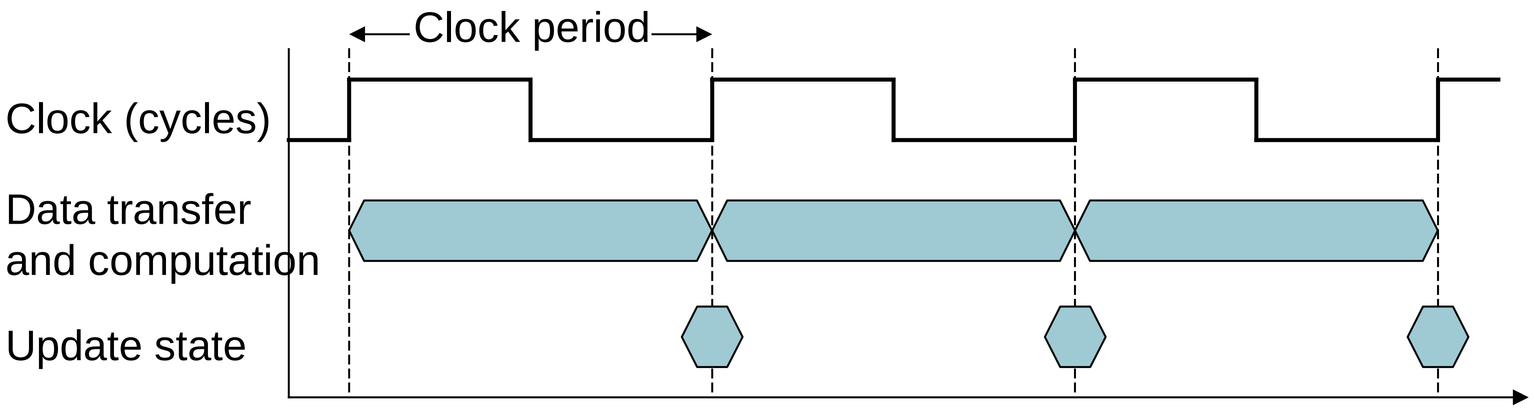
Execution Time$_B$ / Execution Time$_A$

$$15s / 10s = 1.5$$

So A is 1.5 times faster than B

CPU Clock

Determines when events take place in the hardware



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^{9}Hz

CPU Time

CPU execution time for a program

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

How to improve performance?

- Reducing number of clock cycles
- Increasing clock rate

CPU Time Example

A program runs in 10 seconds on computer A, which has a 2 GHz clock rate

Designing a computer B: The program runs 6 seconds, but 1.2x more clock cycles

Computer B clock rate?

CPU Time Example

A program runs in 10 seconds on computer A, which has a 2 GHz clock rate

Designing a computer B: The program runs 6 seconds, but 1.2x more clock cycles

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

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

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

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

How Many Cycles are Required for a Program?

Different instructions take different amounts of time on different machines

- Division takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers

Could assume that

- # of cycles = # of instructions
- This assumption may be used for simplicity, but incorrect

Assembly Instructions

High-level language program (in 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)

```
swap:
      multi $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)

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

Instruction Count

Compiler generates instructions to execute, and the computer has to execute the instructions to run the program, the execution time must depend on the number of instructions in a program

Instruction (IC) count: the total number of instruction executions involved in a program

Instruction Count and CPI

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

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

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

Instruction Count for a program

- Determined by program, ISA and compiler

Average cycles per instruction

- Determined by CPU hardware
- If different instructions have different CPI
  - Average CPI affected by instruction mix

CPI Example

Computer A: Cycle Time = 250ps, CPI = 2.0

Computer B: Cycle Time = 500ps, CPI = 1.2

A and B implements the same ISA

Which is faster for this program? How much?

CPI Example

Computer A: Cycle Time = 250ps, CPI = 2.0

Computer B: Cycle Time = 500ps, CPI = 1.2

A and B implements the same ISA

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

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

A is faster…

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

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

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

…by this much

CPI Example

Instruction Class	**A**	**B**	**C**
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

Which code sequence is faster? What is the CPI?

CPI Example

Instruction Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

Sequence 1: IC = 5

Clock Cycles

$= 2\times1 + 1\times2 + 2\times3$

$= 10$

Avg. CPI = 10/5 = 2.0

Sequence 2: IC = 6

Clock Cycles

$= 4\times1 + 1\times2 + 1\times3$

$= 9$

Avg. CPI = 9/6 = 1.5

CPI in 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

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

Performance Summary

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

Algorithm: affects IC, possibly CPI

Programming language: affects IC, CPI

Compiler: affects IC, CPI

Instruction set architecture: affects IC, CPI, T_c

How to Determine the Values

We can measure the CPU execution time by running the program

Clock cycle time is usually published as part of the documentation for a computer

We can measure the instruction count by using software tools that profile the execution or by using a simulator of the architecture

CPI, however, depends on a wide variety of design details in the computer, including both the memory system and the processor structure

Computer Benchmarks

A benchmark is a program or set of programs used to evaluate computer performance

Benchmarks allow us to make performance comparisons based on execution times

Benchmarks should

- Be representative of the type of applications run on the computer
- Not be overly dependent on one or two features of a computer

Benchmarks can vary greatly in terms of their complexity and their usefulness

SPEC CPU Benchmark

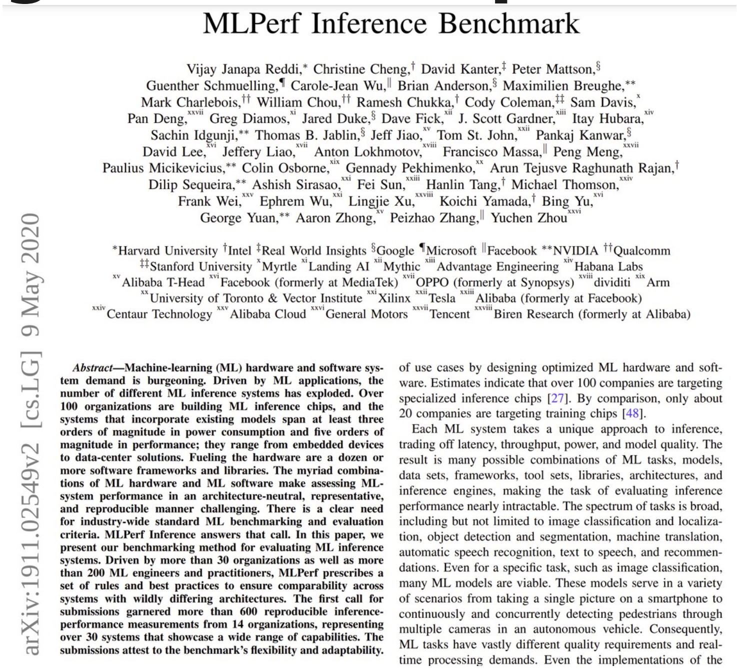
SPECINTC2006 benchmarks running on a 2.66 GHz Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

MLPerf Inference Benchmark

Industry-wide standard ML benchmarking and evaluation

Driven by more than 30 organizations and more than 200 ML engineers and practitioners



MLPerf Inference Benchmark

Vijay Janapa Reddi,[*] Christine Cheng,[†] David Kanter,[‡] Peter Mattson,[§]
Guenther Schmuelling,[¶] Carole-Jean Wu,[‖] Brian Anderson,[§] Maximilien Breughe,[**]
Mark Charlebois,[††] William Chou,[††] Ramesh Chukka,[†] Cody Coleman,[‡‡] Sam Davis,[x]
Pan Deng,[xxvii] Greg Diamos,[xi] Jared Duke,[§] Dave Fick,[xii] J. Scott Gardner,[xiii] Itay Hubara,[xiv]
Sachin Idgunji,[**] Thomas B. Jablin,[§] Jeff Jiao,[xv] Tom St. John,[xxii] Pankaj Kanwar,[§]
David Lee,[xvi] Jeffery Liao,[xvii] Anton Lokhmotov,[xviii] Francisco Massa,[‖] Peng Meng,[xxvii]
Paulius Micikevicius,[**] Colin Osborne,[xix] Gennady Pekhimenko,[xx] Arun Tejusve Raghunath Rajan,[†]
Dilip Sequeira,[**] Ashish Sirasao,[xxi] Fei Sun,[xxiii] Hanlin Tang,[†] Michael Thomson,[xxiv]
Frank Wei,[xxv] Ephrem Wu,[xxi] Lingjie Xu,[xxviii] Koichi Yamada,[†] Bing Yu,[xvi]
George Yuan,[**] Aaron Zhong,[xv] Peizhao Zhang,[‖] Yuchen Zhou[xxvi]

[*]Harvard University [†]Intel [‡]Real World Insights [§]Google [¶]Microsoft [‖]Facebook [**]NVIDIA [††]Qualcomm
[‡‡]Stanford University [x]Myrtle [xi]Landing AI [xii]Mythic [xiii]Advantage Engineering [xiv]Habana Labs
[xv]Alibaba T-Head [xvi]Facebook (formerly at MediaTek) [xvii]OPPO (formerly at Synopsys) [xviii]dividiti [xix]Arm
[xx]University of Toronto & Vector Institute [xxi]Xilinx [xxii]Tesla [xxiii]Alibaba (formerly at Facebook)
[xxiv]Centaur Technology [xxv]Alibaba Cloud [xxvi]General Motors [xxvii]Tencent [xxviii]Biren Research (formerly at Alibaba)

***Abstract*—Machine-learning (ML) hardware and software system demand is burgeoning. Driven by ML applications, the number of different ML inference systems has exploded. Over 100 organizations are building ML inference chips, and the systems that incorporate existing models span at least three orders of magnitude in power consumption and five orders of magnitude in performance; they range from embedded devices to data-center solutions. Fueling the hardware are a dozen or more software frameworks and libraries. The myriad combinations of ML hardware and ML software make assessing ML-system performance in an architecture-neutral, representative, and reproducible manner challenging. There is a clear need for industry-wide standard ML benchmarking and evaluation criteria. MLPerf Inference answers that call. In this paper, we present our benchmarking method for evaluating ML inference systems. Driven by more than 30 organizations as well as more than 200 ML engineers and practitioners, MLPerf prescribes a set of rules and best practices to ensure comparability across systems with wildly differing architectures. The first call for submissions garnered more than 600 reproducible inference-performance measurements from 14 organizations, representing over 30 systems that showcase a wide range of capabilities. The submissions attest to the benchmark's flexibility and adaptability.**

of use cases by designing optimized ML hardware and software. Estimates indicate that over 100 companies are targeting specialized inference chips [27]. By comparison, only about 20 companies are targeting training chips [48].

Each ML system takes a unique approach to inference, trading off latency, throughput, power, and model quality. The result is many possible combinations of ML tasks, models, data sets, frameworks, tool sets, libraries, architectures, and inference engines, making the task of evaluating inference performance nearly intractable. The spectrum of tasks is broad, including but not limited to image classification and localization, object detection and segmentation, machine translation, automatic speech recognition, text to speech, and recommendations. Even for a specific task, such as image classification, many ML models are viable. These models serve in a variety of scenarios from taking a single picture on a smartphone to continuously and concurrently detecting pedestrians through multiple cameras in an autonomous vehicle. Consequently, ML tasks have vastly different quality requirements and real-time processing demands. Even the implementations of the

Amdahl's Law

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

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

Amdahl's Law Example

Multiply accounts for 80s/100s

How much improvement in multiply performance to get 2× overall?

$$50 = \frac{80}{n} + 20 \qquad n = 2.67$$

Amdahl's Law Example

Multiply accounts for 80s/100s

How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

Summary of Performance Evaluation

Good benchmarks, such as the SPEC benchmarks, can provide an accurate method for evaluating and comparing computer performance

Amdahl’s law provides an efficient method for determining speedup due to an enhancement

Make the common case fast!

References

Digital Design and Computer Architecture - Lecture 1: Introduction and Basics, ETH Zurich, by Onur Mutlu

https://www.youtube.com/watch?v=VcKjvwD930o

Chapter 1 Hennessy/Patterson, Computer Abstractions and Technology