



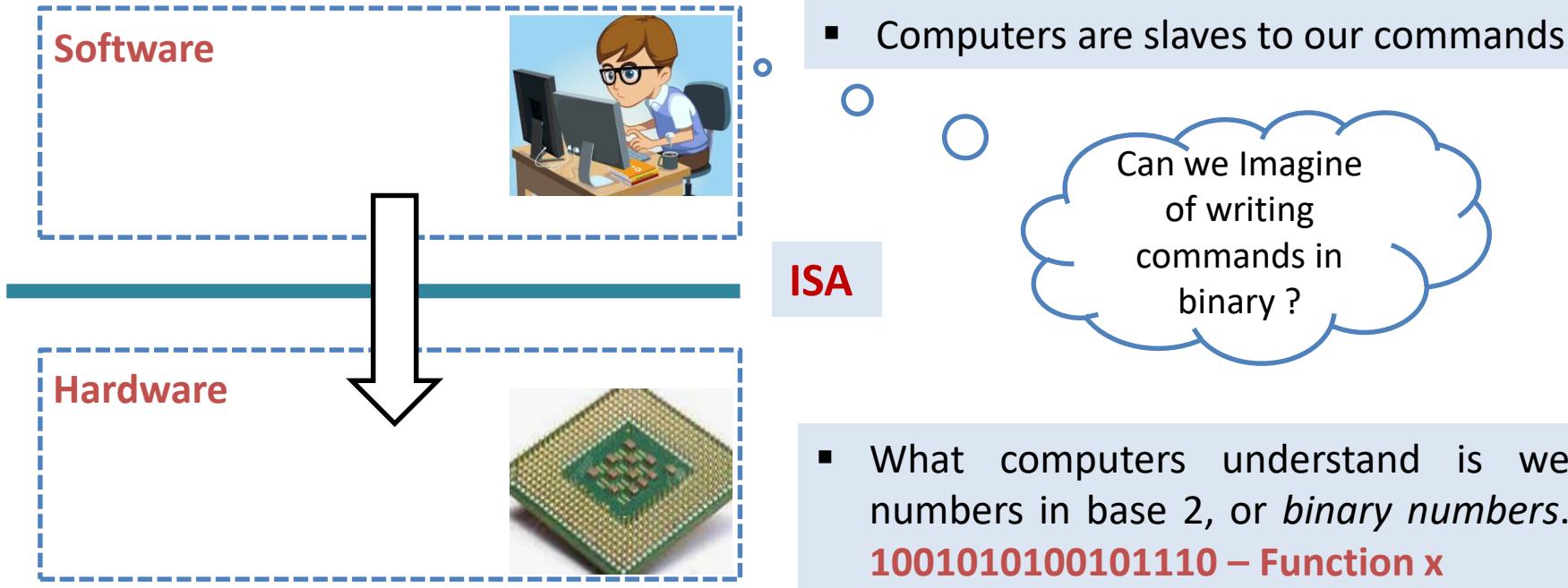
PES
UNIVERSITY
ONLINE

RISC V Architecture

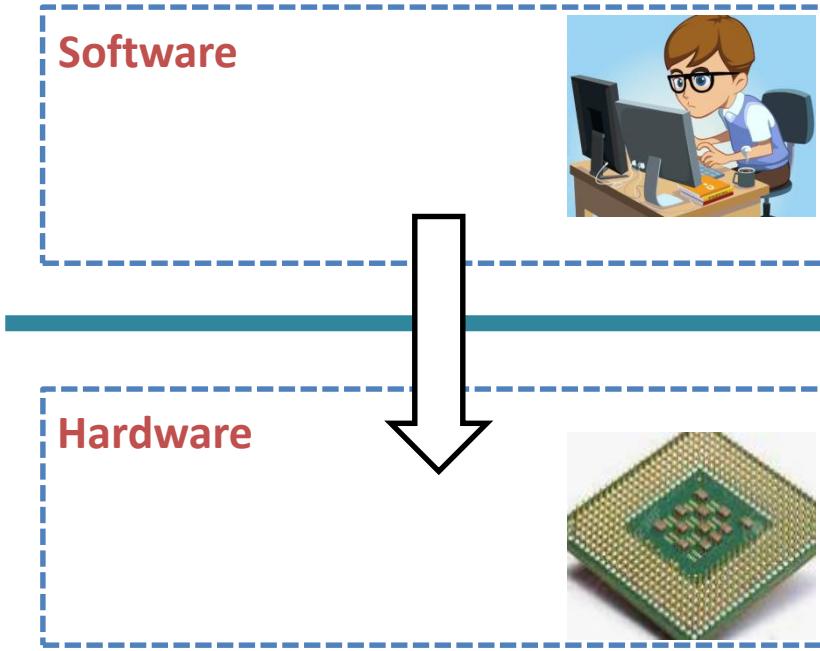
Mahesh Awati & Vinay Reddy

Department of Electronics and Communication Engg.

Interaction between hardware and software



Interaction between hardware and software



From a High-Level Language to
the Language of Hardware

ISA- Language what
Processor can understand

Computer Architecture –
High speed Architecture -
Memory Hierarchy, Pipelining,
Parallelism,

RISC V ARCHITECTURE

Introduction

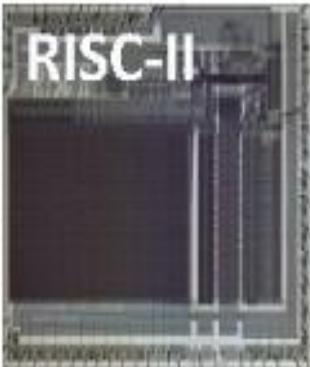
- Modern computer technology requires professionals to **understand both hardware and software**.
- The **interaction between hardware and software at a variety of levels** also offers a framework for understanding the fundamentals of computing.
- **How programs to run faster**, they must become parallel.
- **How programs to run efficiently** on parallel computers



RISC V ARCHITECTURE

Introduction

RISC-V (pronounced “risk-five”) stands for ‘Reduced Instruction Set Computer (RISC) Five’. The number five refers to the number of generations of [RISC architecture](#) that were developed at the **University of California**, Berkeley since 1981.



David Patterson



Andrew Waterman



Yunsup Lee



Krste Asanovic

Why RISC-V ????

- Lots of **small processors are requirements** of the day to build systems
- Existing licensing terms are **quite difficult and are expensive.**
- More importantly, it **was restricted in terms of the freedom to change it.**

The only reason for using RISC-V is because you want the freedom to change it and add your own things to it." - It is open source

A **license-free, royalty-free**
High-quality,
Suitable for all types of computing system,
RISC ISA specification,
Originally from UC Berkeley

RISC V ARCHITECTURE

Introduction

This course content broadly has two parts



- 1. Software:** Unit1,2 and 3 in which we understand the ISA of RISC-V with introduction Computer Architecture
- 2. Hardware:** Unit4 and 5 in which covers computer architecture concepts in understanding the how hardware plays role in performance

RISC V ARCHITECTURE

Computer Abstractions and Technology:

Introduction, Eight Great Ideas in Computer Architecture, Technologies for building processors and Memory, Performance, The Power Wall, The Switching from uniprocessor to Multiprocessor, Benchmarking Intel i7, Fallacies and Pitfalls, Concluding Remarks.

Instructions-Part1

Instructions: The Language of Computer: Introduction, Operations of Computer hardware, Operands of Computer hardware, Signed and Unsigned numbers, Representing Instruction in Computer, Logical operations, Instructions for making decisions, supporting procedures in Computer hardware, Communicating the people.

Instructions-Part2

RISC-V Addressing for wide immediate and addresses, Parallelism and Instructions: Synchronization, Translating and Starting Programs, A C Sort Example to put it All together, Arrays versus Pointers, Advanced Material: Compiling C and Interpreting Java, Real Stuff: MIPS Instructions, x86 Instructions, The Rest of RISC-V Instructions, Fallacies and Pitfalls, Concluding Remarks,

RISC V ARCHITECTURE

Arithmetic for Computer

Introduction, Addition and Subtraction, Multiplication, Division, Floating Point, Parallelism and Computer Arithmetic: Sub word parallelism, Real Stuff, Streaming SIMD Extensions, Advanced Vector Extensions in x86, Going Faster: Subword parallelism and Matrix Multiply, Fallacies and Pitfalls

The Processor

Introduction, Logic Design Conventions, building a Datapath, A Simple Implementation Scheme, an overview of pipelining, Pipelined Datapath and Control, Data Hazards, Exceptions, Parallelism via Instructions, Real stuff: ARM –A53 and Intel Core i7 Pipelines, Going Faster: Instruction Level Parallelism and Matrix Multiply, Advanced Topic, Fallacies and Pitfalls, Concluding Remarks

RISC V ARCHITECTURE

UNIT 1 – Computer Abstractions and Technology

Mahesh Awati & Vinay Reddy

Department of Electronics and Communication Engineering

Computer Abstractions and Technology

Syllabus/Topics



1. Introduction
2. Seven Great Ideas in Computer Architecture
3. Below Your Program
4. Technologies for Building Processors and Memory
5. Performance
6. The Power Wall
7. The Sea Change: The Switch from Uniprocessors to Multiprocessors
8. Real Stuff: Benchmarking the Intel Core i7
9. Going Faster: Matrix Multiply in Python
10. Fallacies and Pitfalls

Can you think of a place where computers are not used today?



Applications that were economically infeasible suddenly become practical.

In the recent past, few applications were “computer science fiction.”



Computer Abstractions and Technology

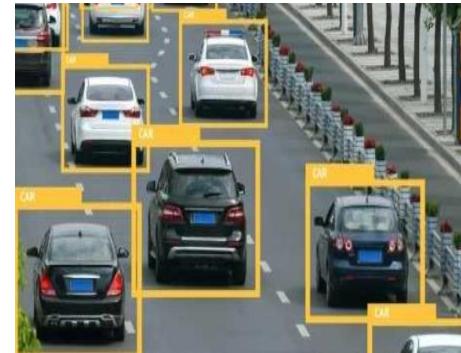
Introduction



What where those “computer science fiction” applications?

1. Computers in automobiles

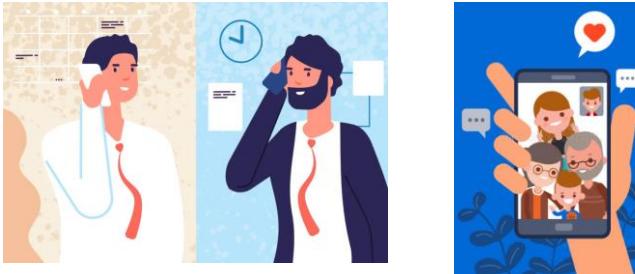
- Reduce pollution
 - Improve fuel efficiency via engine controls
 - Increase safety through
 - blind spot warnings,
 - lane departure warnings,
 - moving object detection, and
 - air bag inflation to protect occupants in a crash.



What where those “computer science fiction” applications?

2. Cell phones

Allowing person-to-person communication to almost anyone anywhere in the world.



3. Human genome project – Biomedical

You will soon be able to acquire your own genome, **allowing medical care to be tailored to you.**

What where those “computer science fiction” applications?

4. World Wide Web

The web has replaced libraries and newspapers.



5. Search engines

As the content of the web grew in size and in value, finding relevant information became increasingly important - rely on search engines



Is that all?

More of it....

Today's science fiction suggests tomorrow's killer applications: already on their way are –

- Glasses that augment reality



- The cashless society



- Cars that can drive themselves

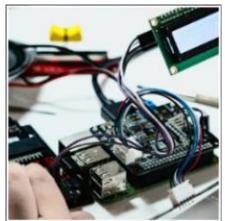
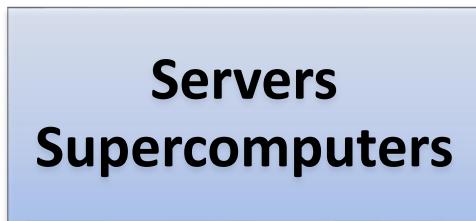


Traditional Classes of Computing Applications and Their Characteristics

Broadly computers are used in three dissimilar classes of applications



**Personal
Computers**



**Embedded
Computers**

1. Personal computers (PCs):

- A computer designed for use by an individual.
- Personal computers emphasize delivery of good performance to single users.
- Low costs and usually execute third-party software.

Traditional Classes of Computing Applications and Their Characteristics

2. Servers:

- A computer used for **running larger programs for multiple users.**
- Typically accessed only via a network.

Supercomputer:

- A class of computers with the **highest performance and cost.**
- They are configured as servers.
- These are used for **high-end scientific and engineering calculations**, such as weather forecasting, oil exploration, protein structure determination, and other large-scale problems



The Cray-1 supercomputer was designed by American electrical engineer and supercomputer architect Seymour Cray in the 1970s.

Traditional Classes of Computing Applications and Their Characteristics

3. Embedded computer:

- A computer inside another device used for running one predetermined application or collection of software.
- A popular term today is Internet of Things (IoT) which suggests many small devices that all communicate wirelessly over the Internet.



Purpose-built computing platforms designed for a specific task or fixed functionality

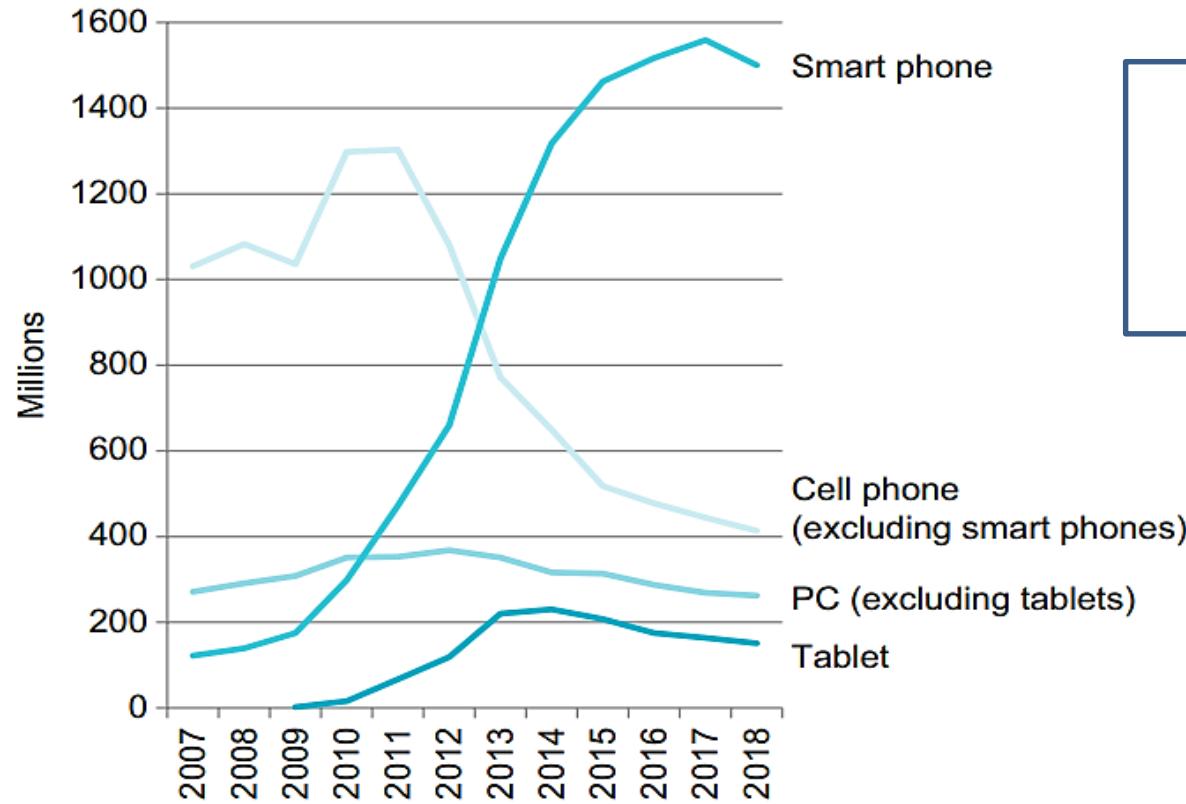
Computer Abstractions and Technology

Introduction

The 2^x vs. 10^y bytes

| Decimal term | Abbreviation | Value | Binary term | Abbreviation | Value | % Larger |
|--------------|--------------|-----------|-------------|--------------|-----------|----------|
| kilobyte | KB | 10^3 | kibibyte | KiB | 2^{10} | 2% |
| megabyte | MB | 10^6 | mebibyte | MiB | 2^{20} | 5% |
| gigabyte | GB | 10^9 | gibibyte | GiB | 2^{30} | 7% |
| terabyte | TB | 10^{12} | tebibyte | TiB | 2^{40} | 10% |
| petabyte | PB | 10^{15} | pebibyte | PiB | 2^{50} | 13% |
| exabyte | EB | 10^{18} | exbibyte | EiB | 2^{60} | 15% |
| zettabyte | ZB | 10^{21} | zebibyte | ZiB | 2^{70} | 18% |
| yottabyte | YB | 10^{24} | yobibyte | YiB | 2^{80} | 21% |
| ronnabyte | RB | 10^{27} | robibyte | RiB | 2^{90} | 24% |
| queccabyte | QB | 10^{30} | quebibyte | QiB | 2^{100} | 27% |

Post-PC Era



Graph shows the rapid growth over time of tablets and smart phones versus that of PCs and traditional cell phones.



FIGURE 1.2 The number manufactured per year of tablets and smart phones, which reflect the post-PC era, versus personal computers and traditional cell phones. Smart

Post-PC Era

Replacing the PC - The Personal Mobile Device (PMD)

- Small wireless devices to connect to the Internet.
- They rely on batteries for power, and software is installed by downloading apps.
- Unlike PCs, they no longer have a keyboard and mouse

Smart phones



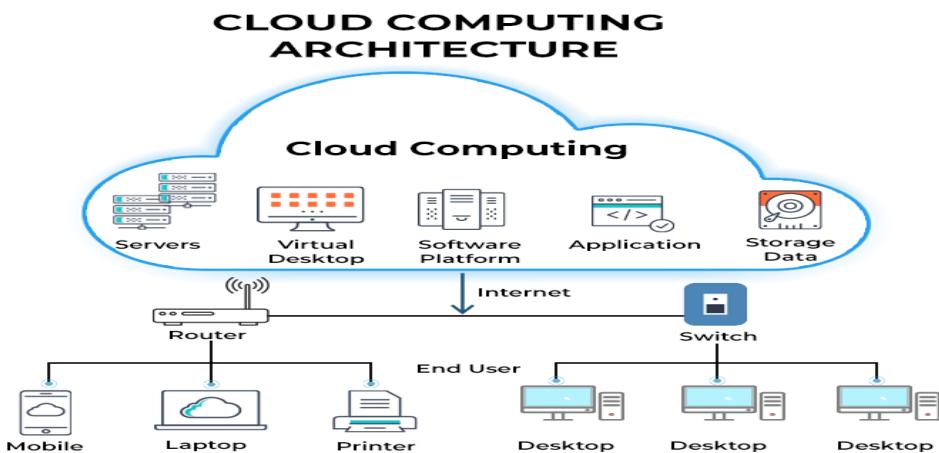
tablets.



Post-PC Era

Taking over from the Conventional Server - Cloud Computing

- Refers to large collections of servers that provide services over the Internet.
- Some providers rent dynamically varying numbers of servers as a utility.
- Relies upon giant datacenters that are now known as Warehouse Scale Computers (WSCs).



Companies like Amazon and Google build these WSCs containing 50,000 servers and then let companies rent portions of them so that they can provide software services to PMDs without having to build WSCs of their own.

Post-PC Era

Software as a Service (SaaS)

WebRISC-V

- Delivers software and data as a service over the Internet
- Done via a thin program such as a browser that runs on local client devices, instead of binary code that must be installed, and runs wholly on that device.
- Examples include web search and social networking.



- *A SaaS application may be accessed through a browser or through an app.*
- *Online email applications that users access through a browser, such as Gmail and Office 365, are common examples of SaaS applications.*

What You Can Learn in this Course

- ✓ Successful programmers have always been concerned about the **performance of their programs**
- ✓ In the 1960s and 1970s a primary constraint on computer performance was the **size of the computer's memory**. Thus, programmers often followed a simple credo: minimize memory space to make programs fast.
- ✓ In last two decades – Memory is not a constraint. -**the parallel nature of processors and the hierarchical nature of memories**
- ✓ today's programmers need to worry about **energy efficiency of their programs** running either on the Personal Mobil Devices (PMD) or in the Cloud, which also requires understanding what is below your code.

What You Can Learn in this Course

Program written High Level Language (HLL)

translated

Language of hardware

How are programs written in a HLL, translated into the language of the hardware

How does the hardware execute the resulting program?

Software

Interface

Hardware

What is the interface between the software and the hardware

how does software instruct the hardware to perform needed functions?

What You Can Learn in this Course

Performance of Program

What determines the Performance of Program ?

How programmer **Improve the Performance of Program ?**

Techniques used by hardware designer to **Improve performance**

Techniques can be used by hardware designers to **improve energy efficiency?**

What are the **reasons for and the consequences of the switch from sequential processing to parallel processing?**

Understanding Program Performance

The performance of a program depends on a combination of -

- the effectiveness of the algorithms used in the program.
- the software systems used to create and translate the program into machine instructions, and
- the effectiveness of the computer in executing those instructions, which may include input/output (I/O) operations.



Abstraction



Pipelining



Parallelism



Common Case Fast



Prediction



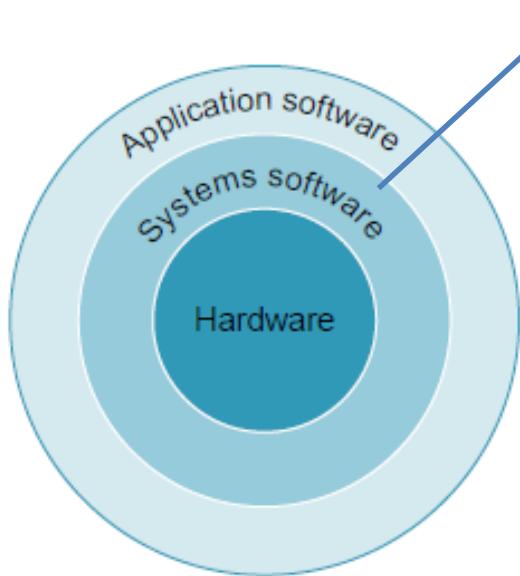
Hierarchy



Dependability

2. Use Abstraction to Simplify Design/Below your Program

- Simplified view of hardware and software as hierarchical layers.
- Hardware in the center and application software outermost.
- A variety of **systems software** sitting between the hardware and the application software

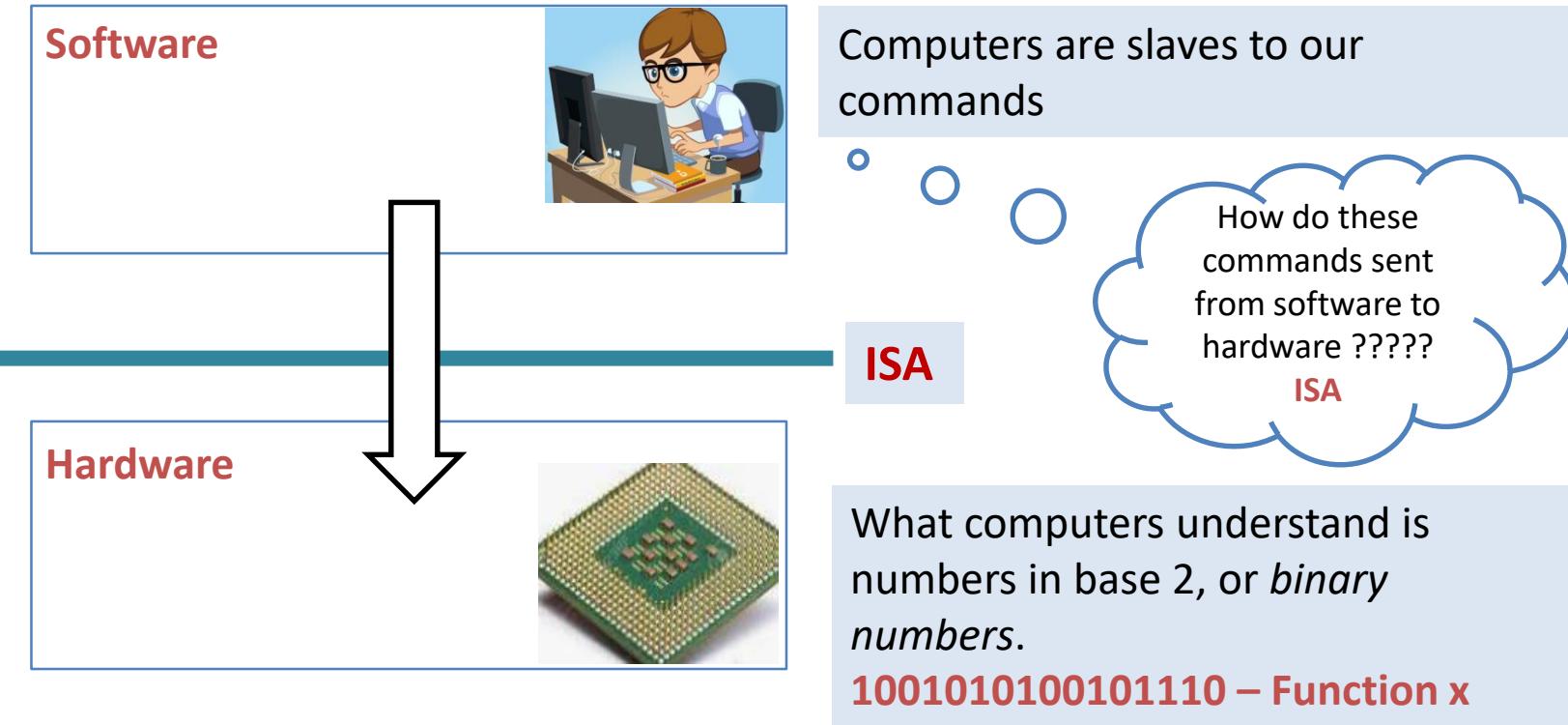


Software that provides services that are commonly useful, including **operating systems**, **compilers**, loaders, and assemblers

- An **operating system** interfaces between a user's program and the hardware and provides a variety of services and supervisory functions.
 - ✓ Handling basic input and output operations
 - ✓ Allocating storage and memory
 - ✓ Provides protected sharing of the computer resources multiple applications.
- An **compiler** A program that translates high-level language into assembly language statements.

Use Abstraction to Simplify Design/Below your Program

Interface between Software and Hardware – Role of Instruction Set Architecture (ISA)

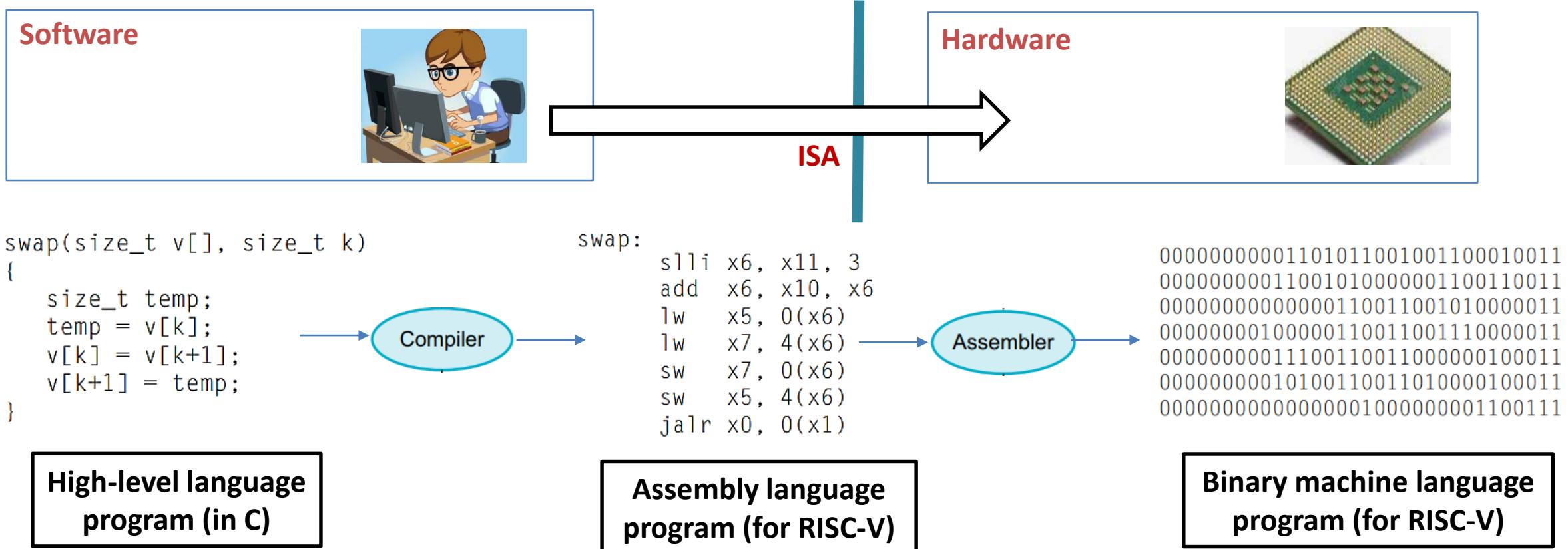


Computer Abstractions and Technology

Seven Great Ideas in Computer Architecture

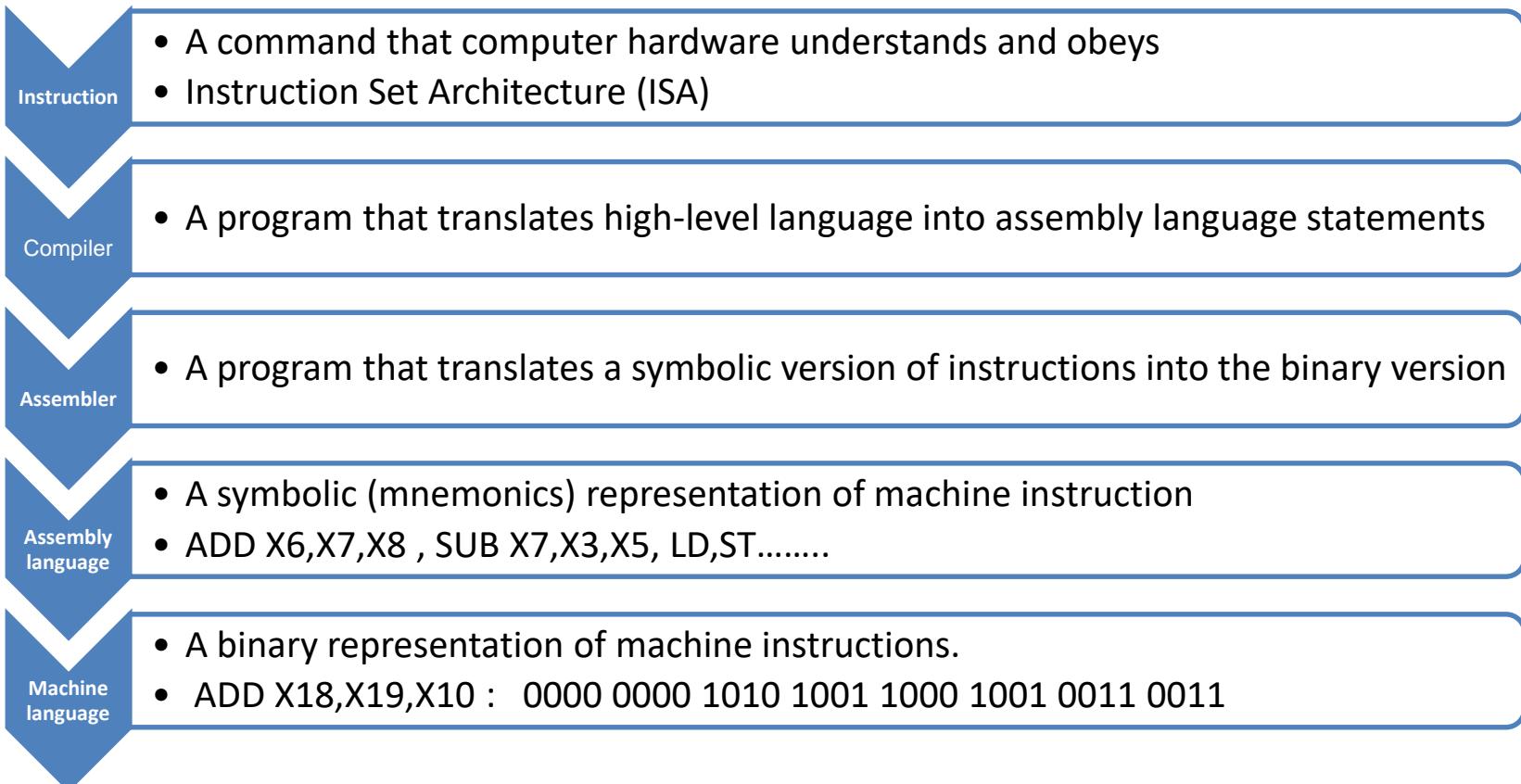
Use Abstraction to Simplify Design/Below your Program

From a High-Level Language to the Language of Hardware



Use Abstraction to Simplify Design/Below your Program

Things to know

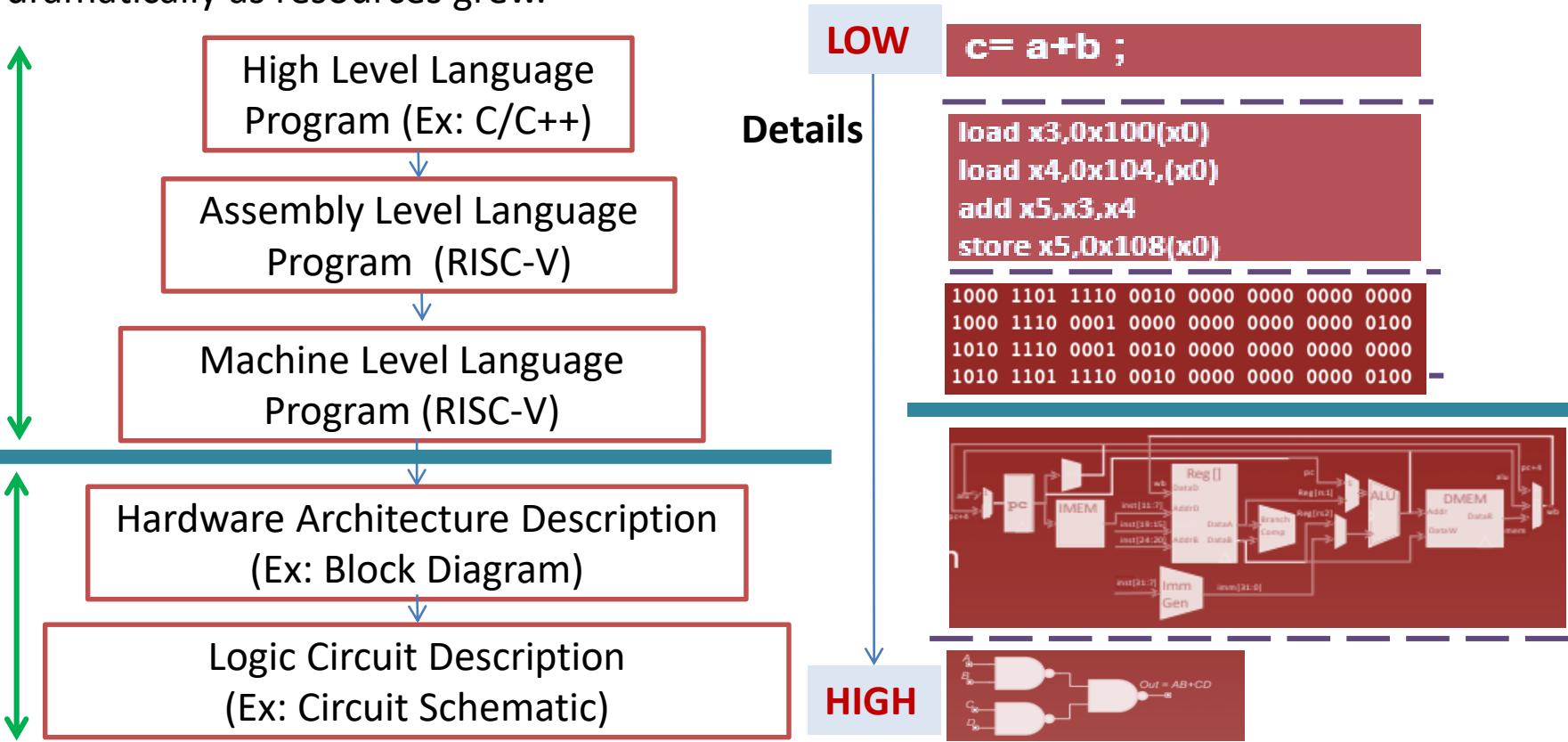


Computer Abstractions and Technology

Seven Great Ideas in Computer Architecture

Use Abstraction to Simplify Design/Below your Program

Both computer architects and programmers had to invent techniques to make themselves more productive, for otherwise design time would lengthen as dramatically as resources grew.



- Abstraction Hides Implementation Details
 - Remembering physical address of a,b, and c
 - Remembering registers in which a,b, and c are stored
 - Loading a,b from memory
 - Storing result in Memory

Why Use Abstraction ?
Simplify the design:
Helps us to **cope with enormous complexity**

Use Abstraction to Simplify Design/Below your Program

Summary

- A compiler enables a programmer to write this high-level language expression:
 $A + B$
- The compiler would compile it into this assembly language statement: add A, B
- The assembler would translate this statement into the binary instructions that tell the computer to add the two numbers A and B.

Benefits of High-level programming:

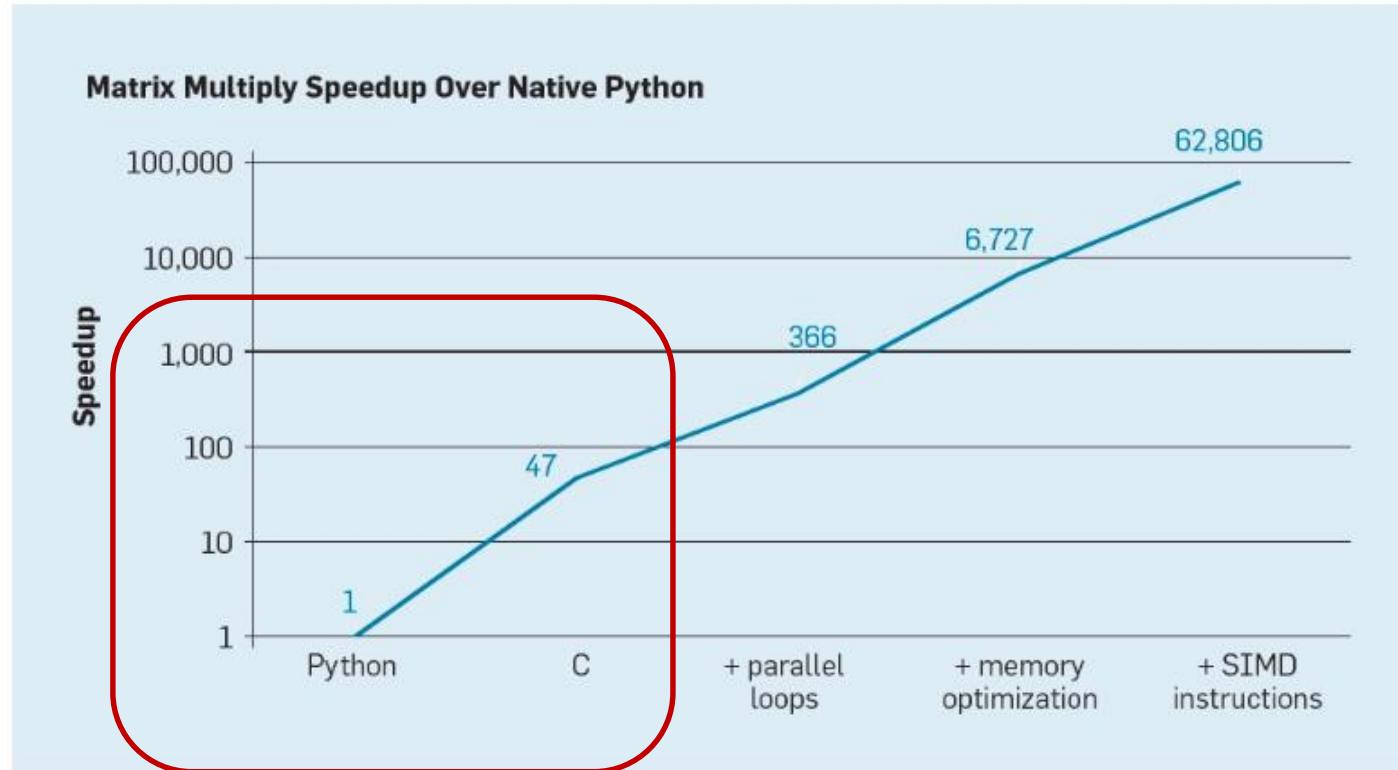
- Allow the programmer to think in a more natural language, using English words and algebraic notation
- Improved **programmer productivity**.
- **Programming languages allow programs to be independent of the computer** on which they were developed.

Computer Abstractions and Technology

Seven Great Ideas in Computer Architecture



Abstraction & inefficiency



Potential speedup of matrix multiply in Python for four optimizations

The C version of the code is 47 times faster than Python version.

Why??

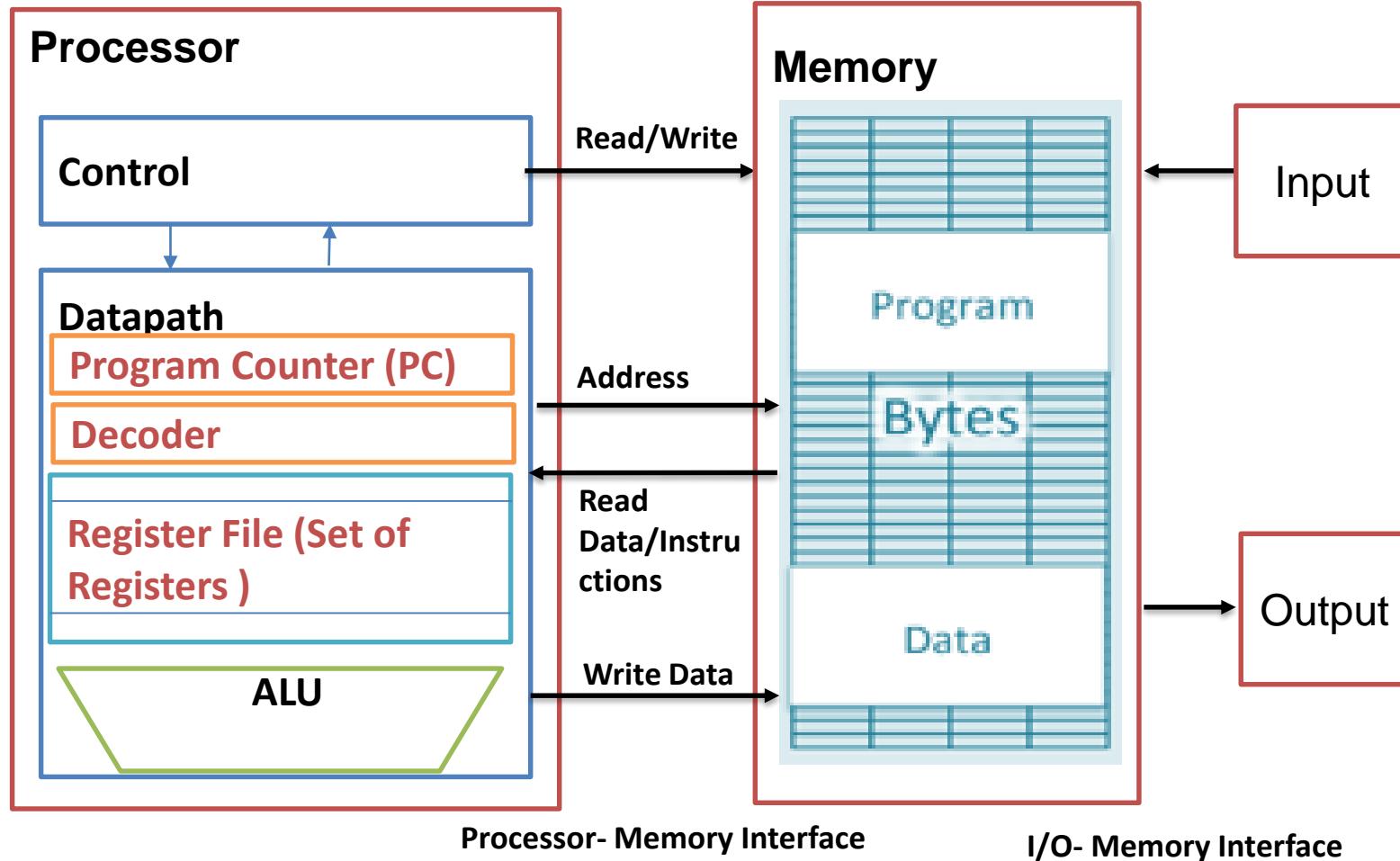


Takeaway: Abstraction is Good but it should not result in inefficiency

Computer Abstractions and Technology

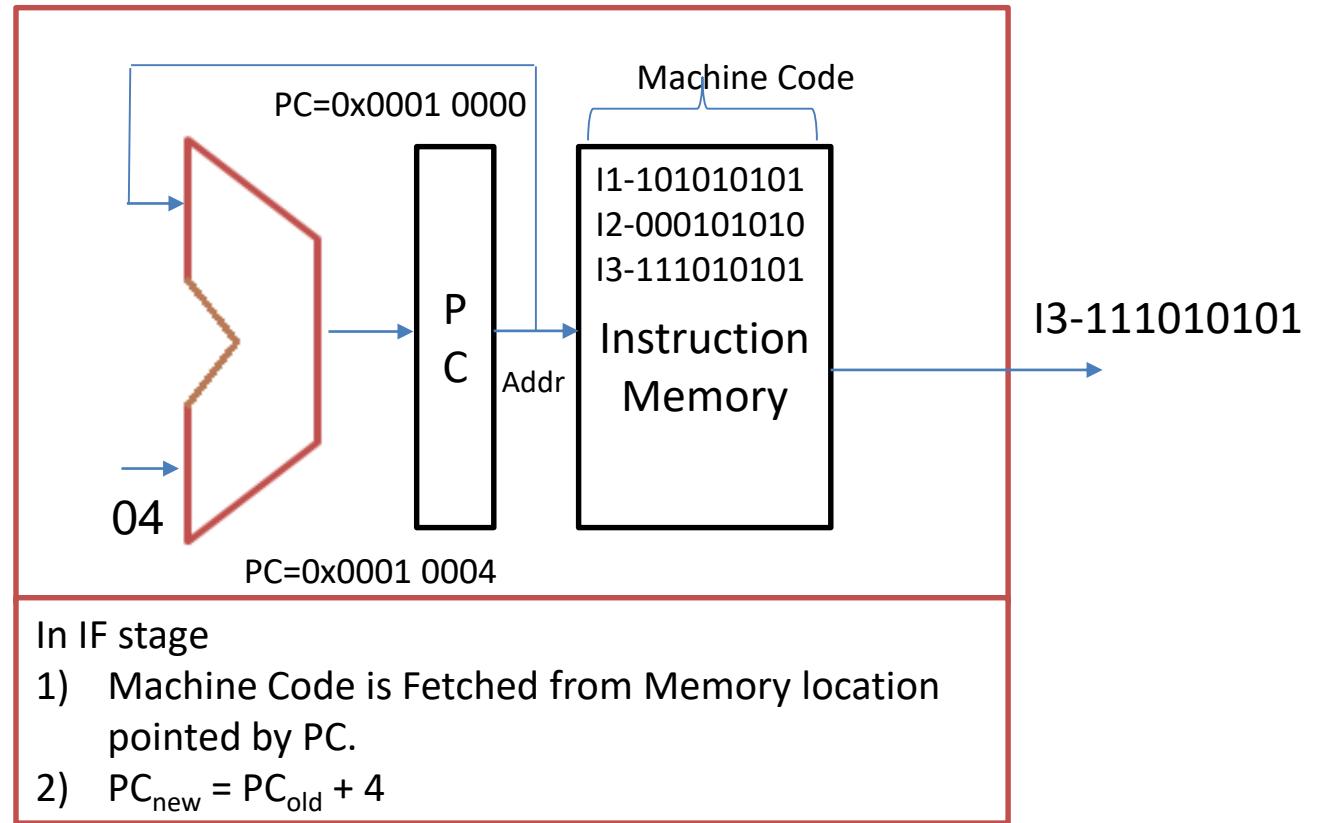
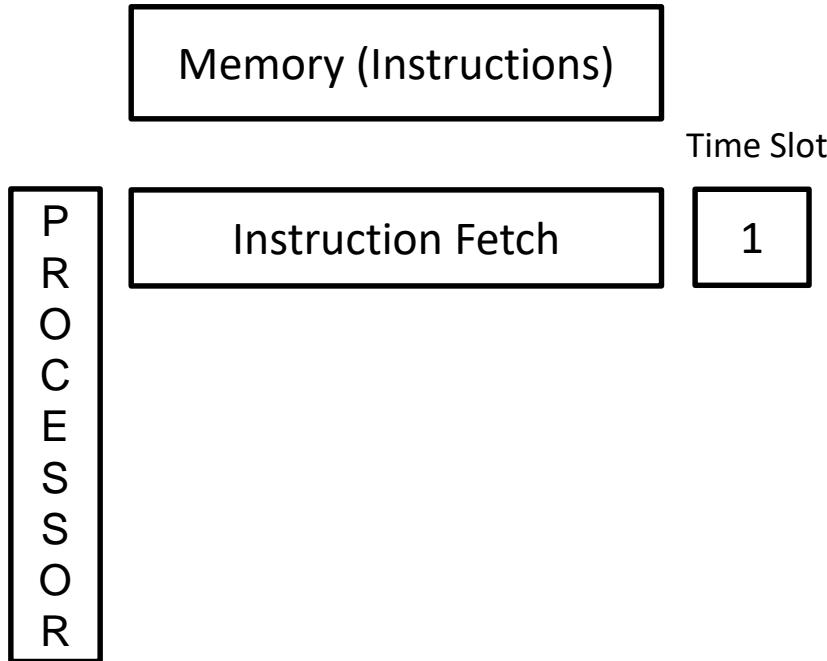
Components of Computer

Things to know



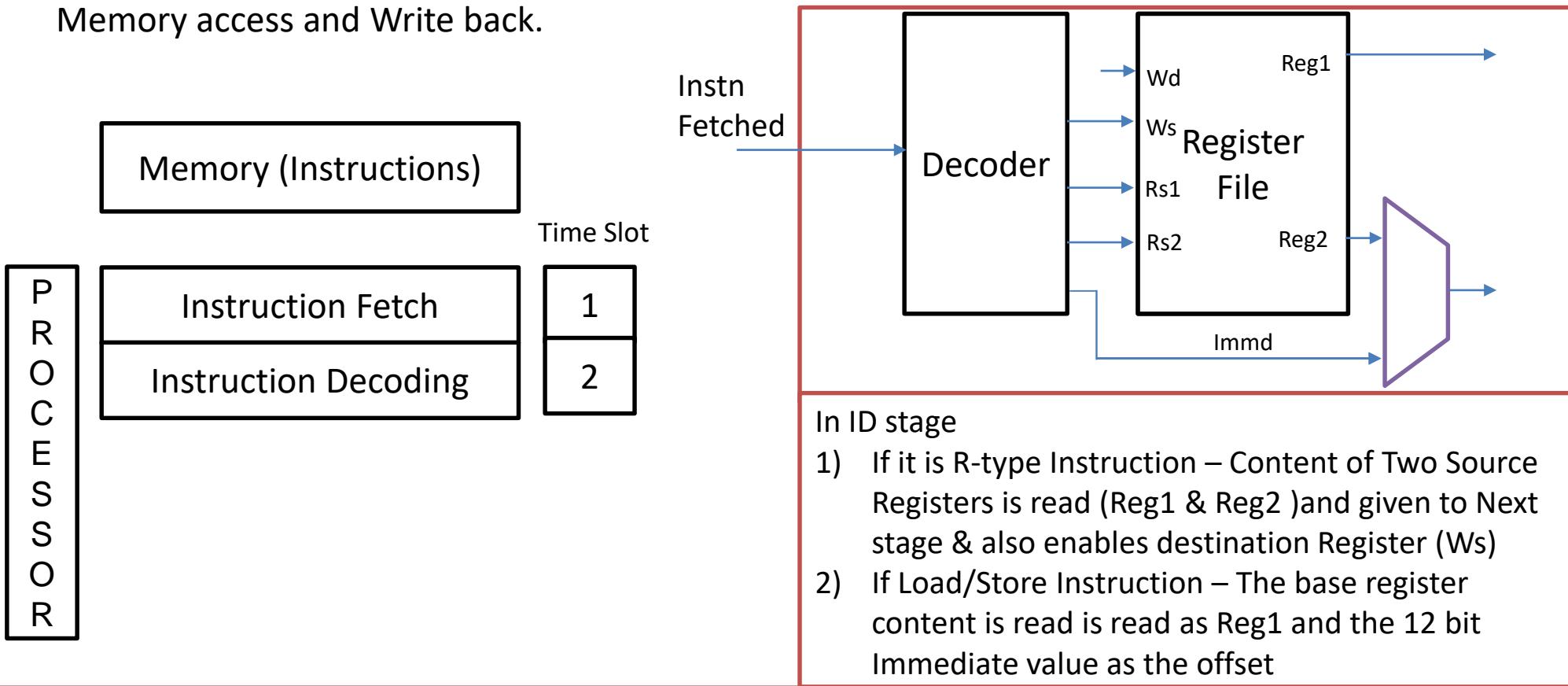
Performance via Pipelining

- First let us understand the steps involved in Non-Pipelined mode of execution.
- Instruction Execution involves different stages like Fetch , Decode, Execute, Memory access and Write back.



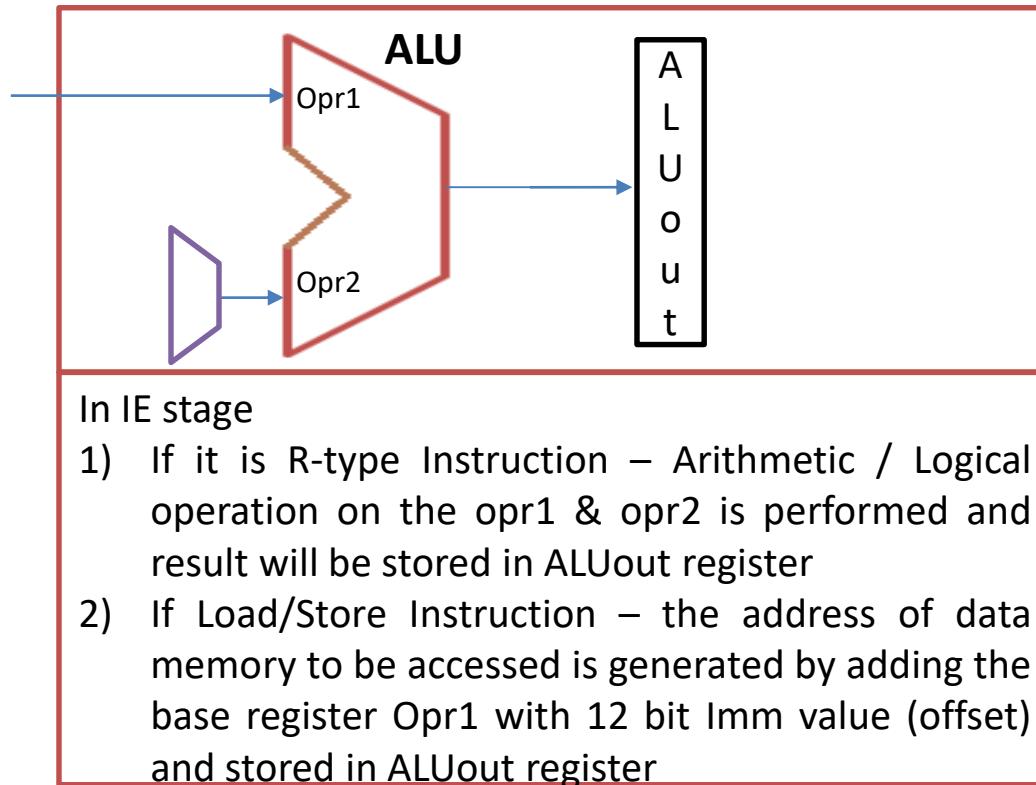
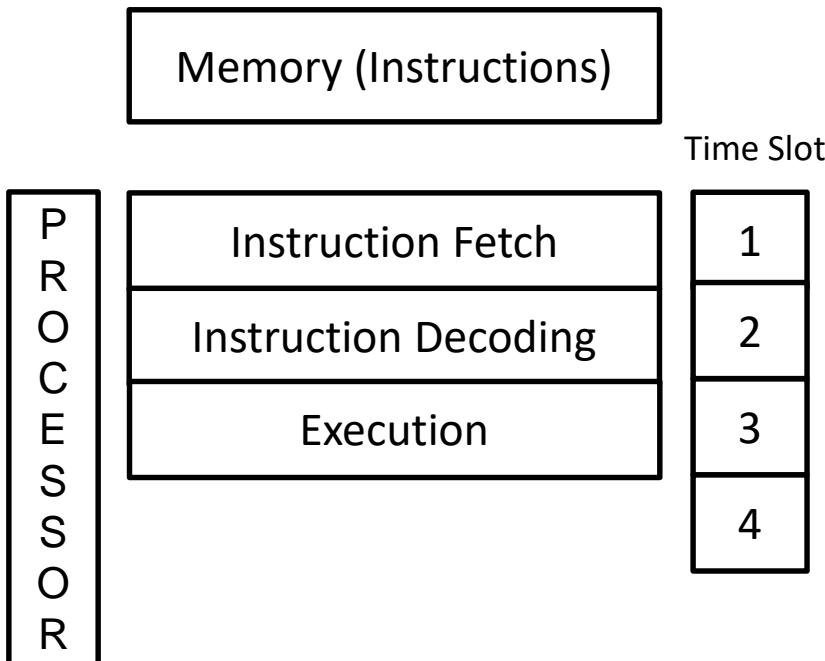
Performance via Pipelining

- First let us understand the steps involved in Non-Pipelined mode of execution.
- Instruction Execution involves different stages like Fetch , Decode, Execute, Memory access and Write back.



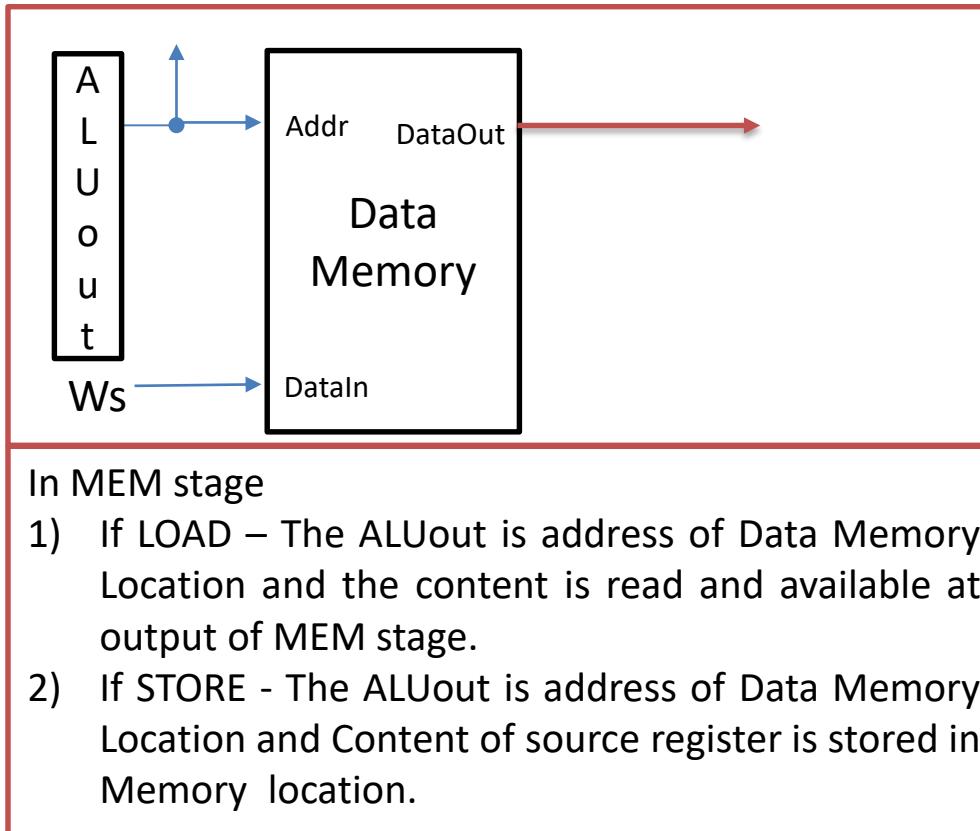
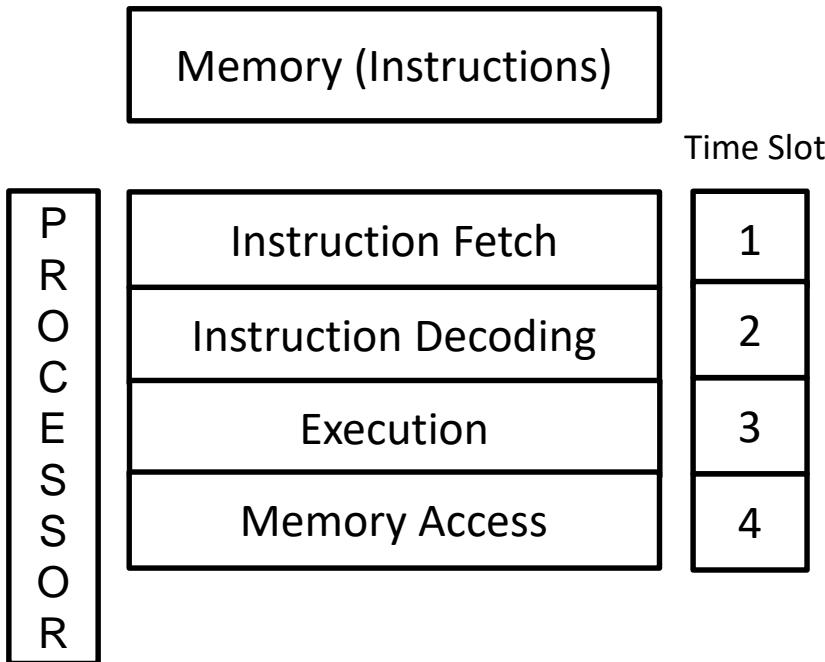
Performance via Pipelining

- First let us understand the steps involved in Non-Pipelined mode of execution.
- Instruction Execution involves different stages like Fetch , Decode, Execute, Memory access and Write back.



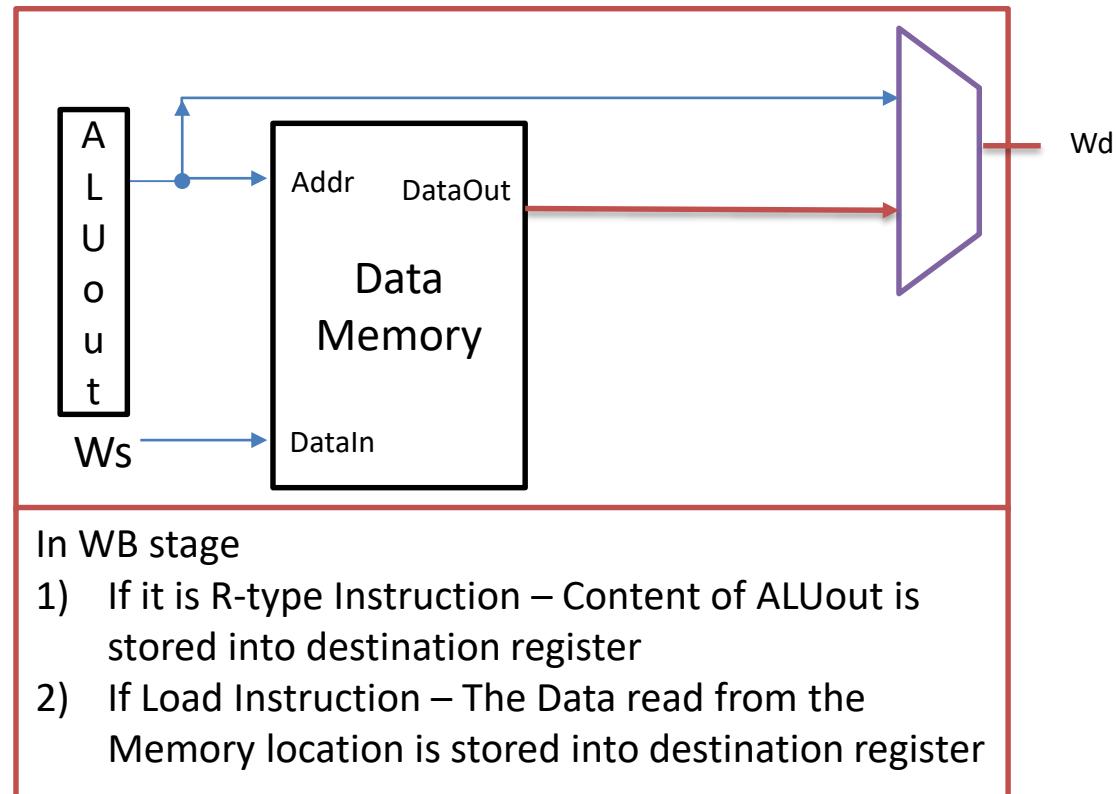
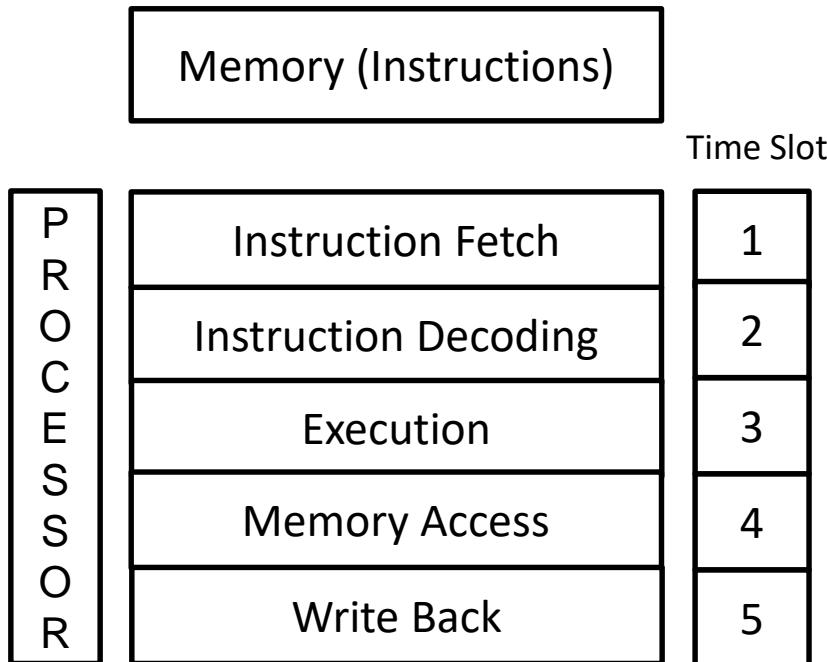
Performance via Pipelining

- First let us understand the steps involved in Non-Pipelined mode of execution.
- Instruction Execution involves different stages like Fetch , Decode, Execute, Memory access and Write back.



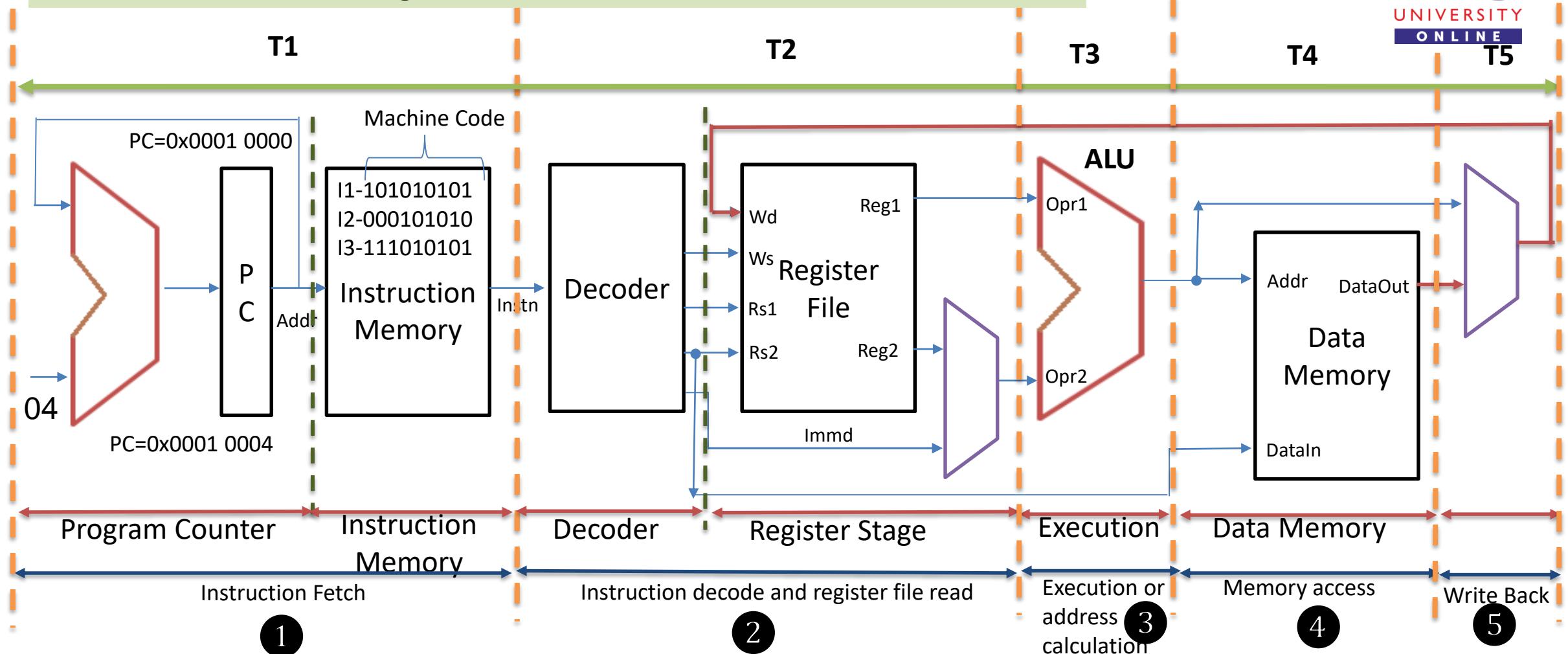
Performance via Pipelining

- First let us understand the steps involved in Non-Pipelined mode of execution.
- Instruction Execution involves different stages like Fetch , Decode, Execute, Memory access and Write back.



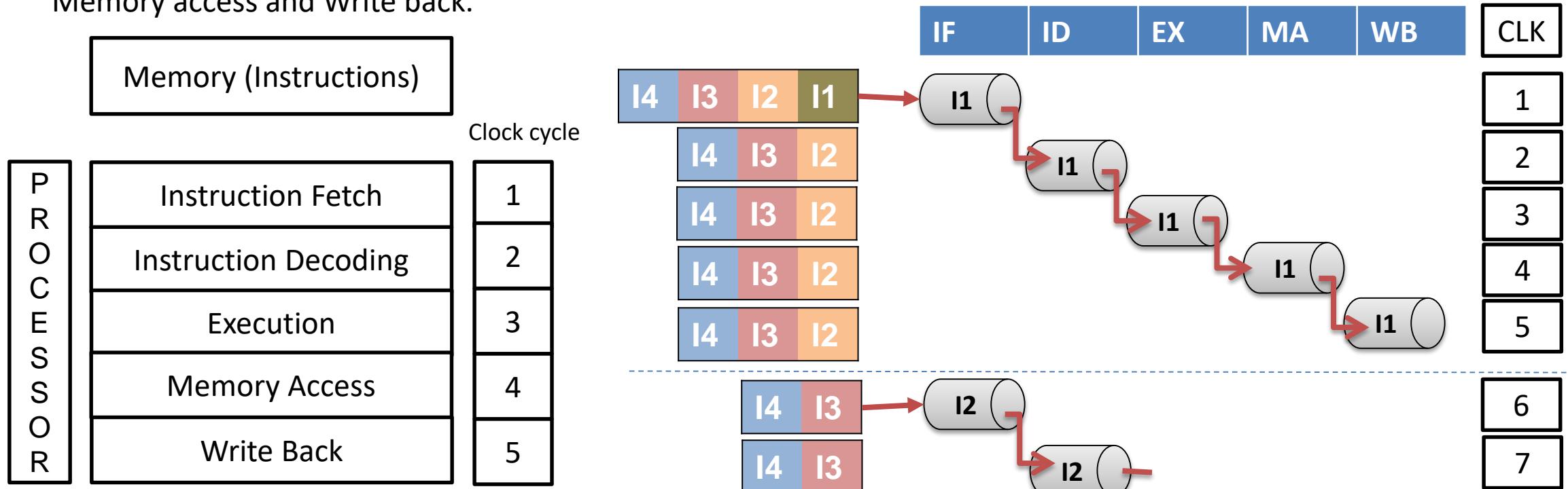


Instruction Execution Stages



Performance via Pipelining

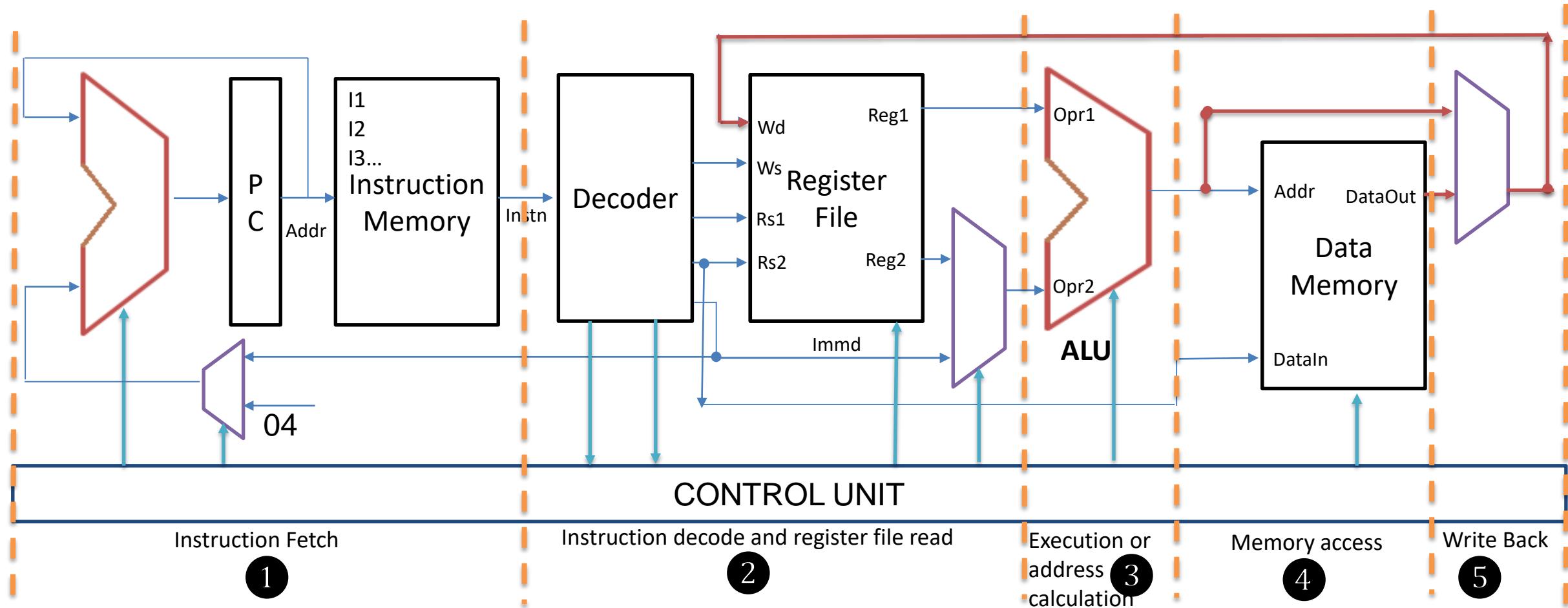
- First let us understand the steps involved in **Non-Pipelined mode** of execution.
- Instruction Execution involves different stages like Fetch , Decode, Execute, Memory access and Write back.



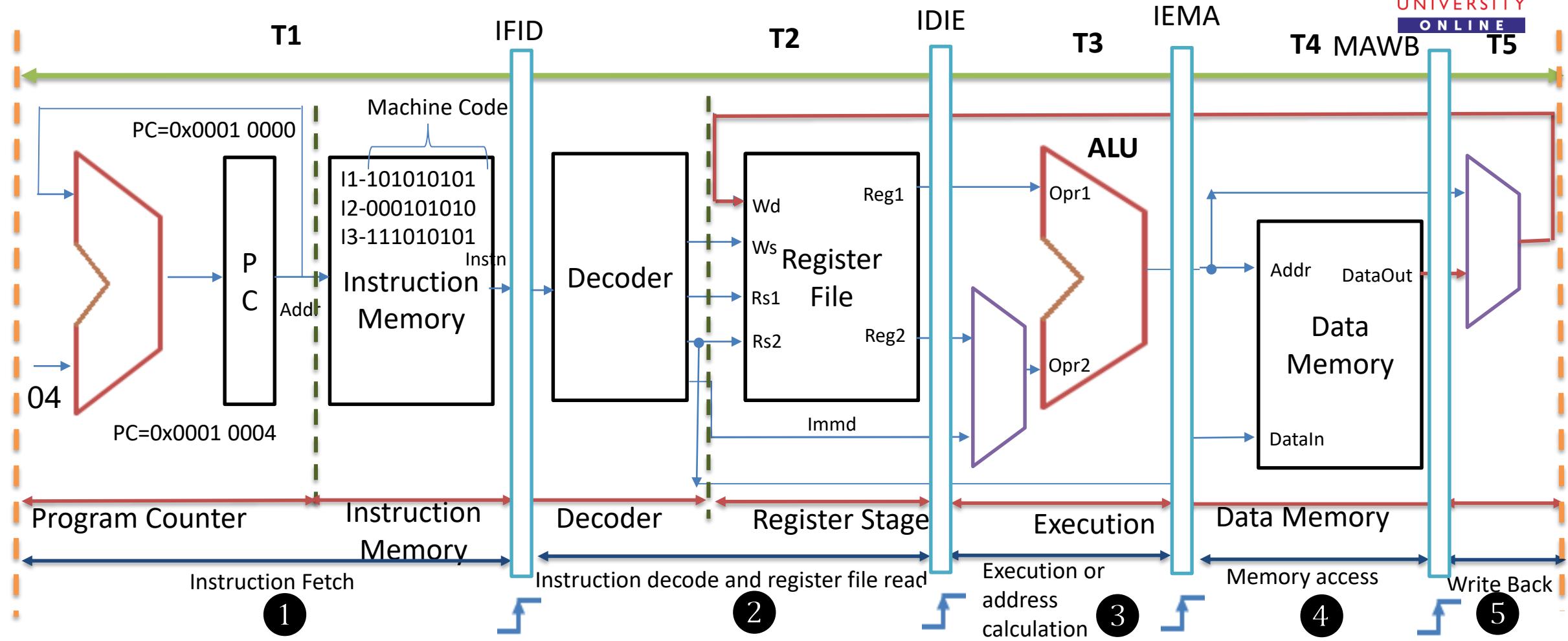
Computer Abstractions and Technology

Data Path and Control

- Datapath - Designed to support data transfers required by instructions.
- Anything that stores data or operates on data within a processor is called data path.

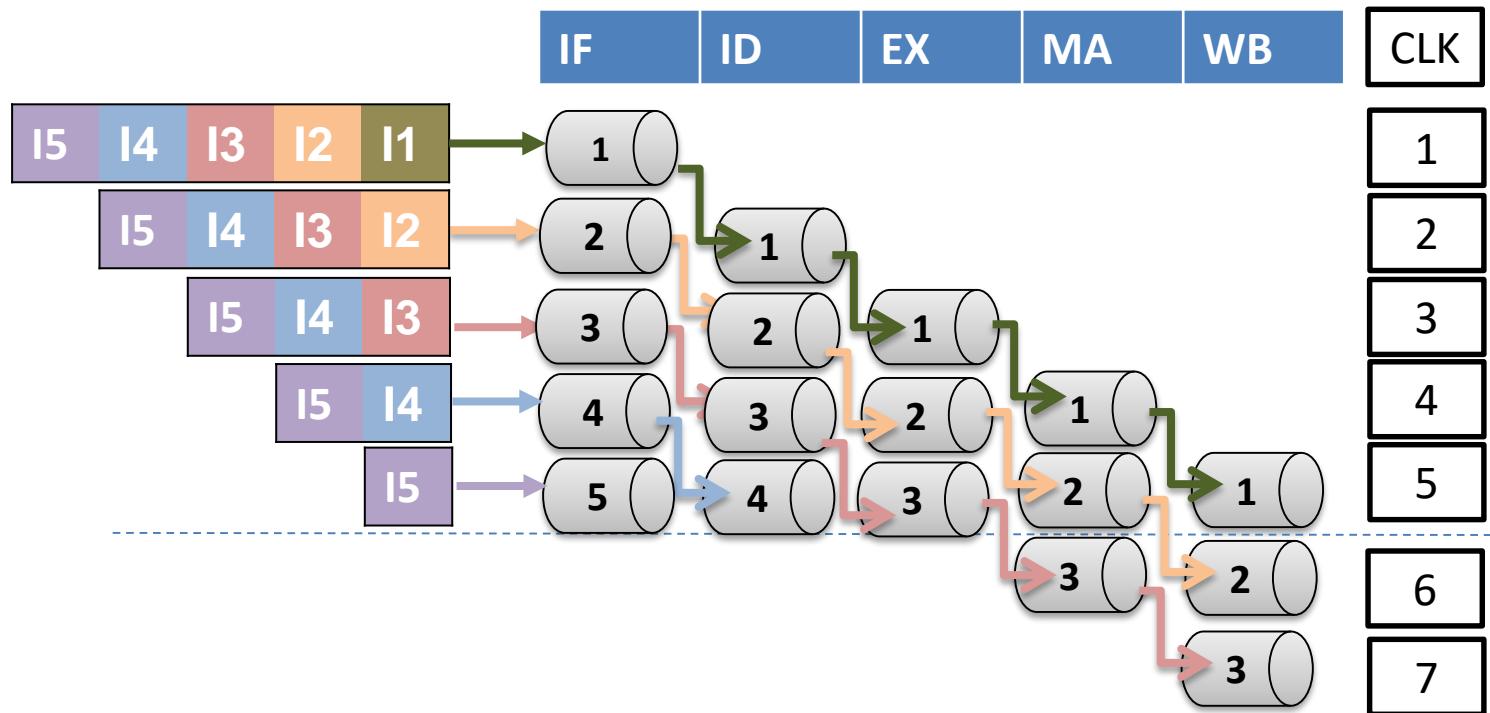


5 Stage Pipelined Processor



Performance via Pipelining

- If these stages are independently performing the task in a sequence, then the **pipelined approach of execution** can be used.
- Processor may have Single, Two, Three, Five or Six stages of pipeline.



Comparison of Non-Pipelined Verses Pipelined Mode

Non Pipeline (Single Cycle Processor)

1. Time Interval of Clock of Single Cycle Processor = $T_{clk} = T_1 + T_2 + T_3 + T_4 + T_5$

Ex:

| | |
|----------------------|--------------|
| Instruction Fetch | $T_1 = 160n$ |
| Instruction Decoding | $T_2 = 120n$ |
| Execution | $T_3 = 130n$ |
| Memory Access | $T_4 = 100n$ |
| Write Back | $T_5 = 140n$ |

$$T_{clk} = T_1 + T_2 + T_3 + T_4 + T_5 = (160 + 120 + 130 + 100 + 140)nSec = 550n$$

2. Time taken by a program with N number of Instructions takes

$$T_{total} = N \times T_{clk}$$

Comparison of Non-Pipelined Verses Pipelined Mode

5 stage Pipeline Processor

1. Time Interval of Clock of Single Cycle Processor = T_{clk} = Maximum {T1,T2,T3,T4,T5}

Ex:

| | |
|----------------------|---------|
| Instruction Fetch | T1=160n |
| Instruction Decoding | T2=120n |
| Execution | T3=130n |
| Memory Access | T4=100n |
| Write Back | T4=140n |

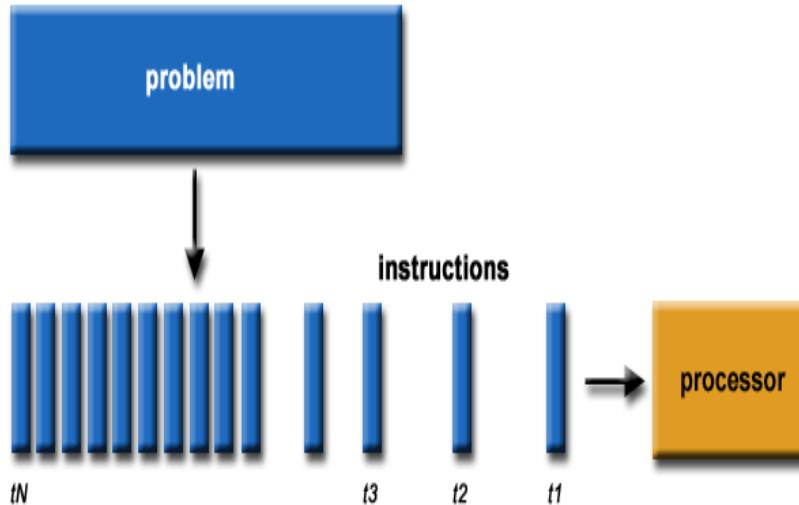
$$T_{clk} = \text{Maximum } \{T1, T2, T3, T4, T5\} = \text{Maximum } \{160n, 120n, 130n, 100n, 140n\}$$

2. Time taken by a program with N number of Instructions takes (Ideally)

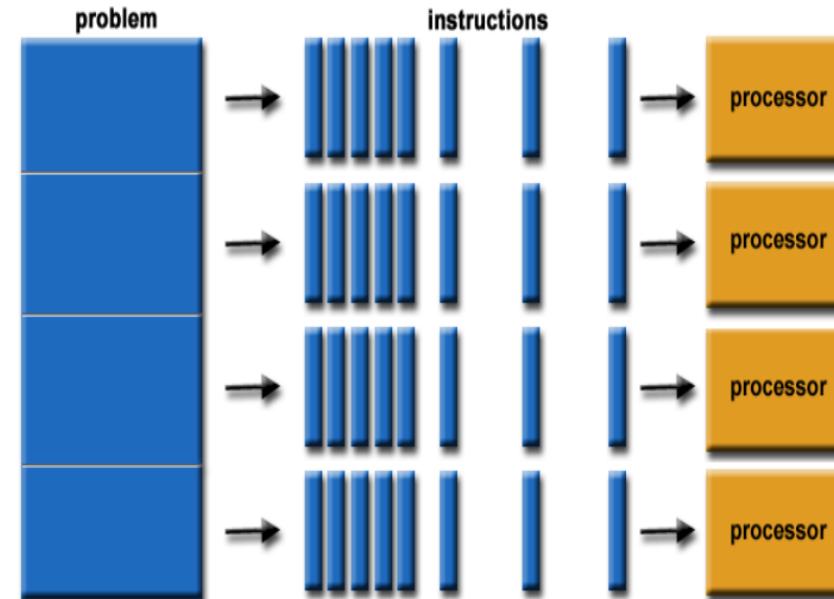
$$T_{total} = (5 + N - 1) \times T_{clk}$$

Performance via Parallelism

Sequential computing



Parallel computing



Doing different parts of a task in parallel accomplishes the task in less time than doing them sequentially

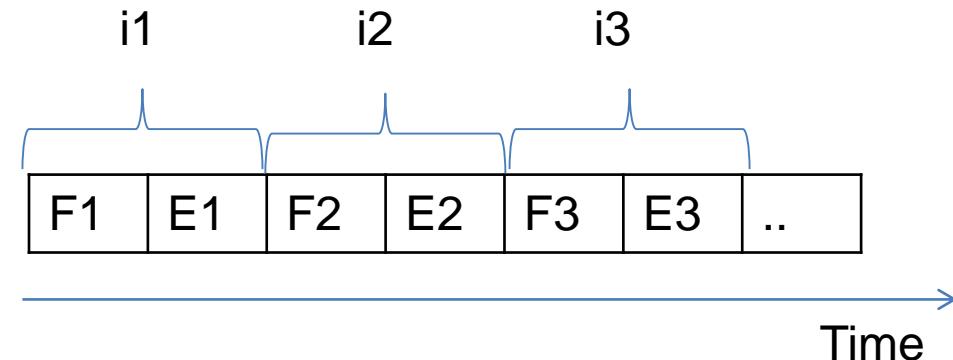
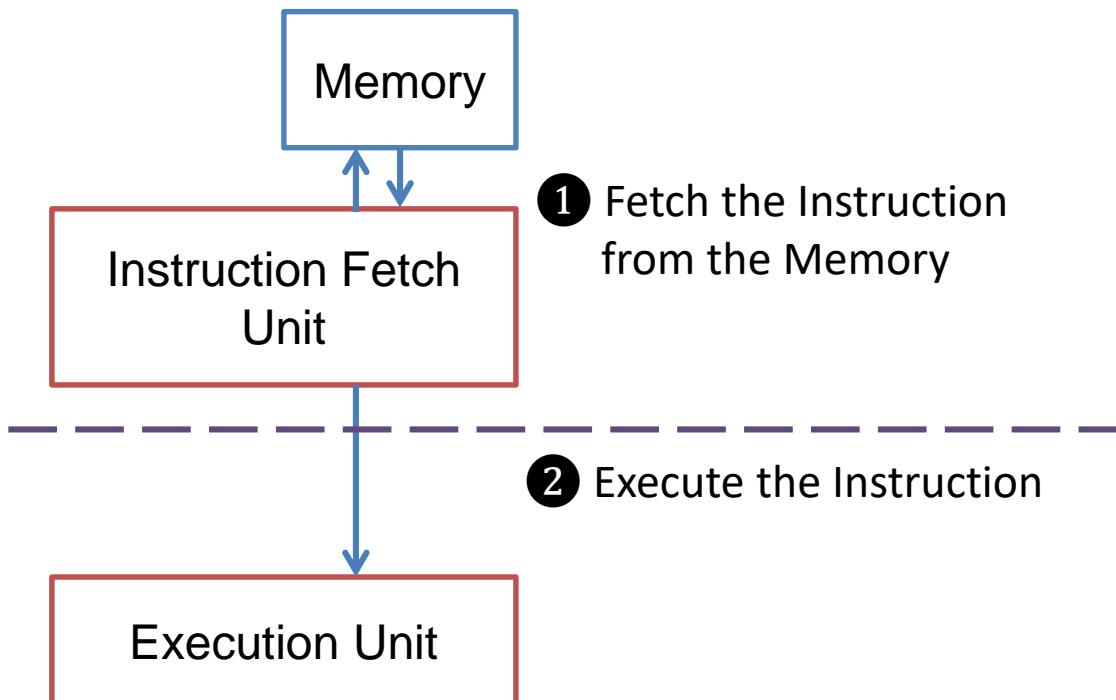
Sequential computing is a computational model in which operations are performed in order, one at a time on one processor or computer.

Parallel computing is a computational model where a problem or program is broken into multiple smaller sequential computing operations some of which are performed simultaneously in parallel.

Performance via Parallelism

- Computer architects have offered designs that get more performance by **computing operations in parallel**

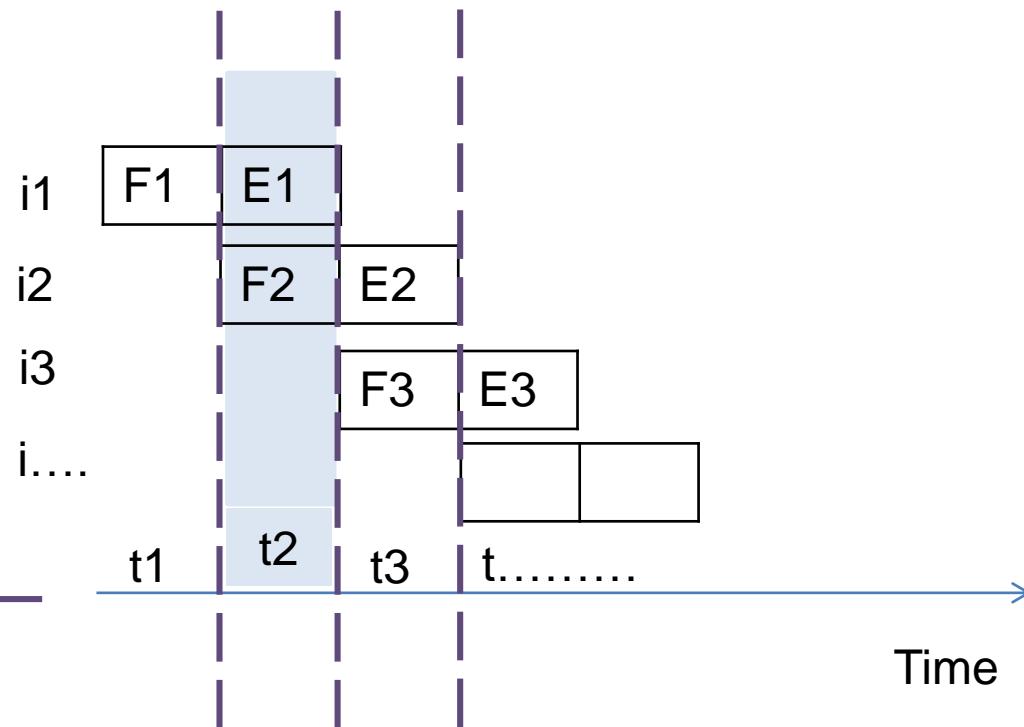
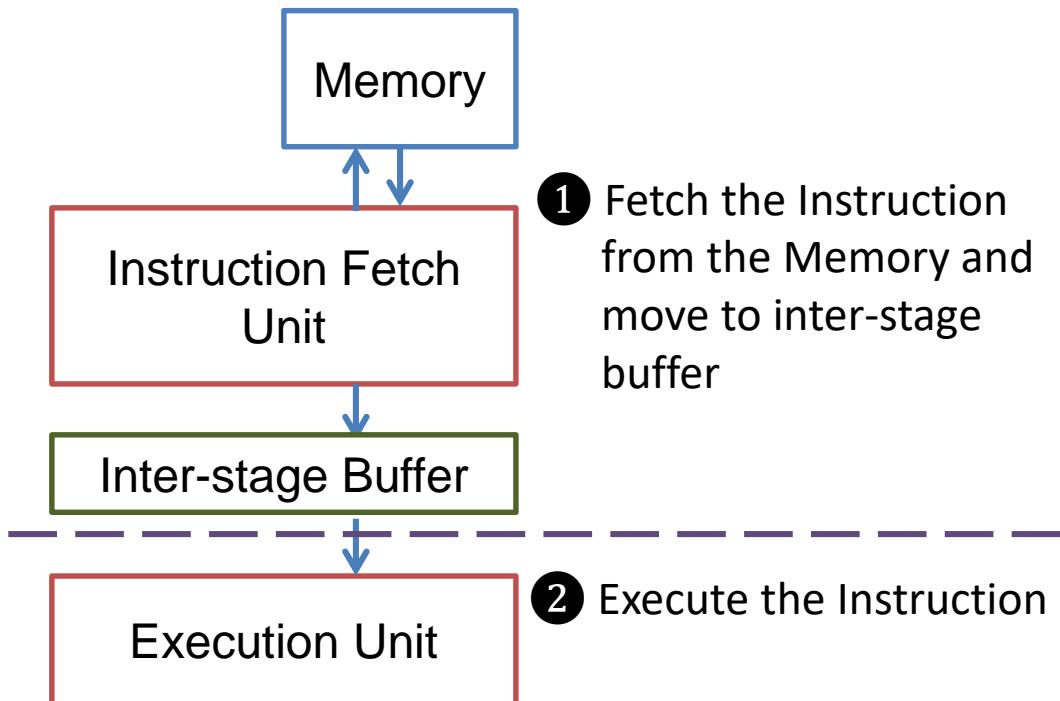
Sequential Operation - Datapath



Performance via Parallelism

- Computer architects have offered designs that get more performance by computing operations in parallel

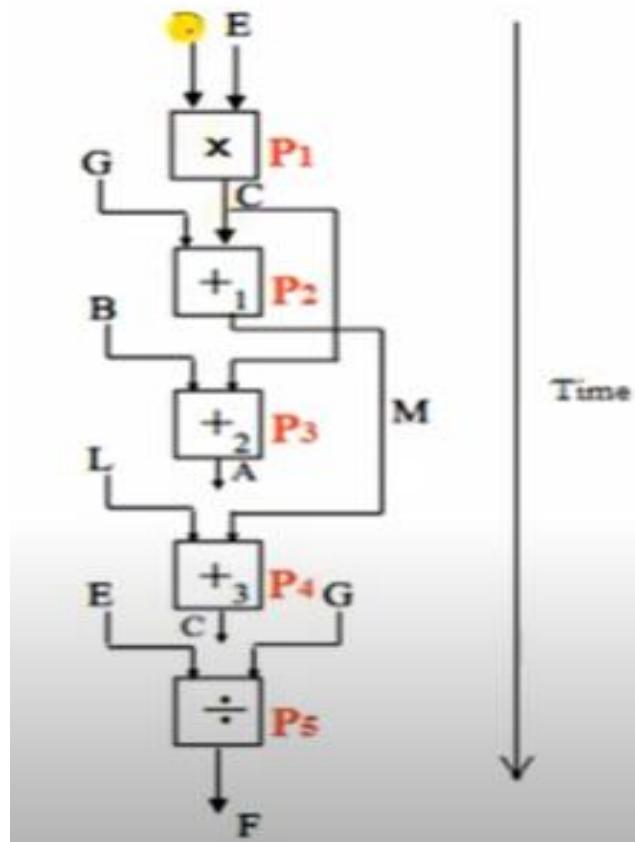
Parallelism in Datapath



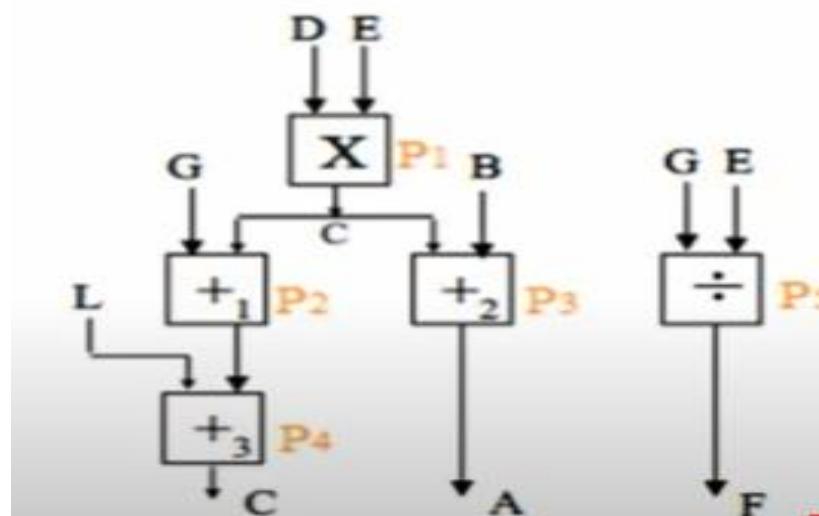
Seven Great Ideas in Computer Architecture

Find how many steps are needed in case of

- a) Sequential execution and
- b) Parallel execution if **two adders** are available



$$\begin{array}{ll} P1: & C = D \times E \\ P2: & M = G + C \\ P3: & A = B + C \\ P4: & C = L + M \\ P5: & F = D / E \end{array}$$



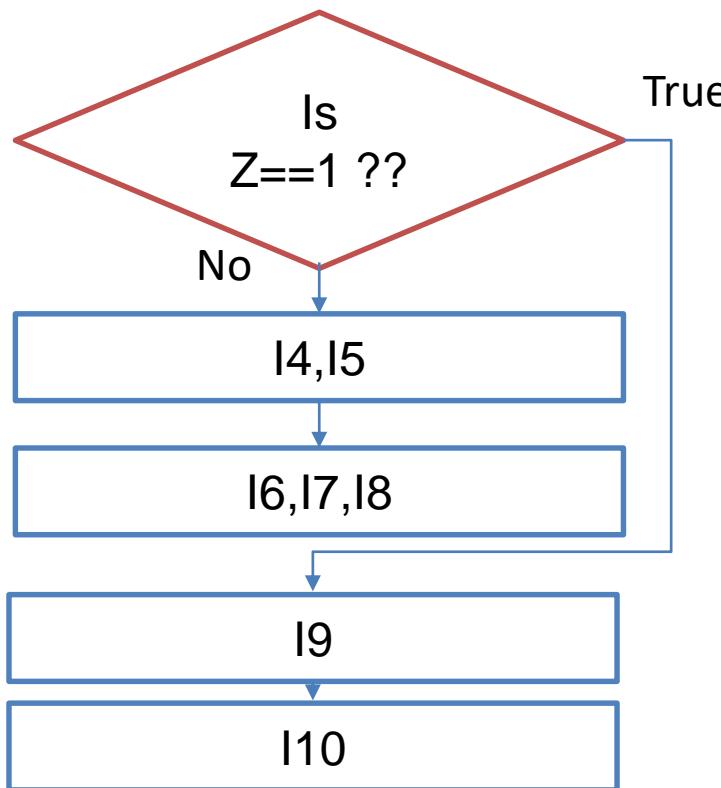
- ✓ Data dependency
- ✓ Resource dependency

Performance via Prediction

In some cases, it can be faster on average to guess and start working rather than wait until you know for sure, assuming that the mechanism to recover from a mis-prediction is not too expensive and your prediction is relatively accurate.

PC

| Address | Label | Instructions |
|---------|-------|--------------|
| 0x1000 | | I1 |
| 0x1004 | | I2 |
| 0x1008 | | BZ Next |
| 0x100C | | I4 |
| 0x1010 | | I5 |
| 0x1014 | | I6 |
| 0x1018 | | I7 |
| 0x101C | | I8 |
| 0x1020 | Next: | I9..... |



Execution of branch instruction or branching by the direct modification of PC.

By the time a branch instruction gets into execution stage of pipeline, regardless of branch instruction outcome, It has fetched next successive instruction and are in pipeline. Therefore the RISC-V core will **flush its pipeline**.

| Cycle | IF | ID | IE | MA | WB |
|-------|----|----|----|----|----|
| 1 | I1 | X | X | X | X |
| 2 | I2 | I1 | X | X | X |
| 3 | I3 | I2 | I1 | X | X |
| 4 | I4 | I3 | I2 | I1 | X |
| 5 | I5 | I4 | I3 | I2 | I1 |
| 6 | | | | | |

**if I3 is a
Conditional
Branch
Instruction**

If branch
condition is
TRUE Branch
location gets
updated in
PC in IE stage in
RISC-V
PC= ALUout

PC →

| Address | Label | Instructions |
|---------|-------|--------------|
| 0x1000 | | I1 |
| 0x1004 | | I2 |
| 0x1008 | | BZ Next |
| 0x100C | | I4 |
| 0x1010 | | I5 |
| 0x1014 | | I6 |
| 0x1018 | | I7 |
| 0x101C | | I8 |
| 0x1020 | Next: | I9..... |



Execution of branch instruction or branching by the direct modification of PC.

By the time a branch instruction gets into execution stage of pipeline, regardless of branch instruction outcome, It has fetched next successive instruction and are in pipeline. Therefore the RISC-V core will **flush its pipeline**.

| Cycle | IF | ID | IE | MA | WB |
|-------|----|----|----|----|----|
| 1 | I1 | X | X | X | X |
| 2 | I2 | I1 | X | X | X |
| 3 | I3 | I2 | I1 | X | X |
| 4 | I4 | I3 | I2 | I1 | X |
| 5 | I5 | I4 | I3 | I2 | I1 |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |

if I3 is a
Conditional Branch
Instruction

PC

If branch condition
is TRUE ; Branch
location gets
updated in
PC in IE stage in
RISC-V
PC= ALUout

FLUSH the Pipeline
0x1020 | Next: | I9

What will
happen to I4
and I5 which
are in Pipeline
????



Mr. Think

Execution of branch instruction or branching by the direct modification of PC.

By the time a branch instruction gets into execution stage of pipeline, regardless of branch instruction outcome, It has fetched next successive instruction and are in pipeline. Therefore the RISC-V core will **flush its pipeline**.

| Cycle | IF | ID | IE | MA | WB |
|-------|-----|-----|----|----|----|
| 1 | I1 | X | X | X | X |
| 2 | I2 | I1 | X | X | X |
| 3 | I3 | I2 | I1 | X | X |
| 4 | I4 | I3 | I2 | I1 | X |
| 5 | | | I3 | I2 | I1 |
| 6 | I9 | | | I3 | I2 |
| 7 | I10 | I9 | | | I3 |
| 8 | I11 | I10 | I9 | | |

I4 and I5 which were in pipeline will be **FLUSHED** & Refilling of pipeline starts from branched location

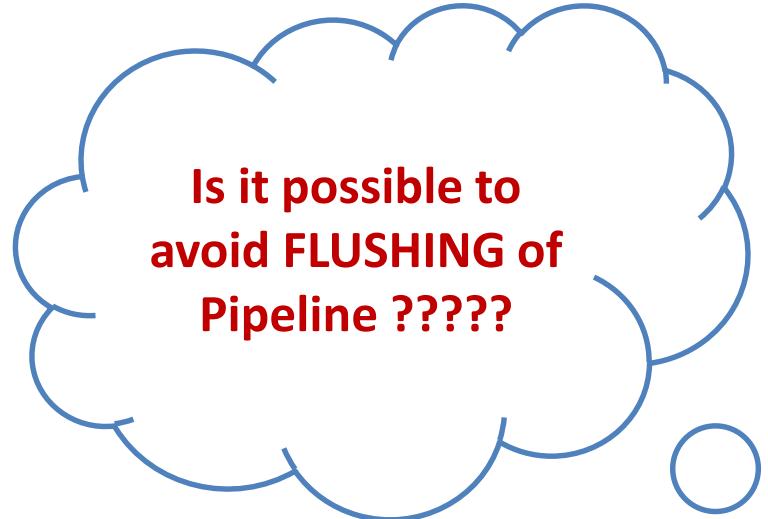
PC →

| Address | Label | Instructions |
|---------|-------|--------------|
| 0x1000 | | I1 |
| 0x1004 | | I2 |
| 0x1008 | | BZ Next |
| 0x100C | | I4 |
| 0x1010 | | I5 |
| 0x1014 | | I6 |
| 0x1018 | | I7 |
| 0x101C | | I8 |
| 0x1020 | Next: | I9..... |

Branch Prediction

In computer architecture, a **branch predictor** is a digital circuit that tries to **guess which way a branch will go before this is known definitively**. This reduces effect of pipeline flush.

Branch prediction **reduces the effect of a pipeline flush by predicting possible branches and loads the new branch address prior to the execution of the instruction**

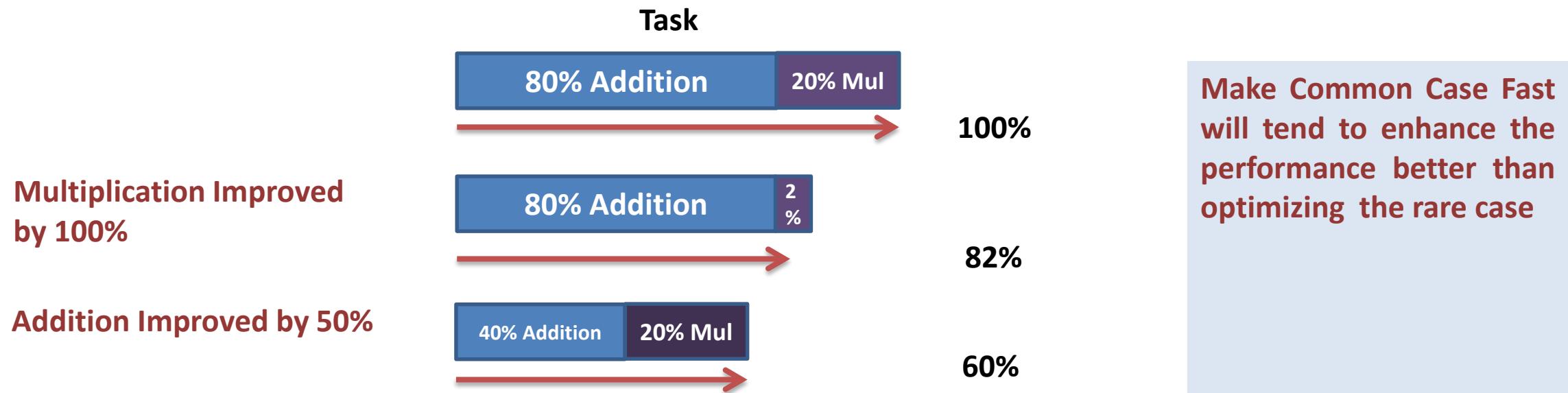


Is it possible to
avoid FLUSHING of
Pipeline ?????



Make the Common Case Fast

Making the **common case fast** will tend to enhance performance better than optimizing the rare case. Ironically, the common case is often simpler than the rare case and hence is usually easier to enhance..

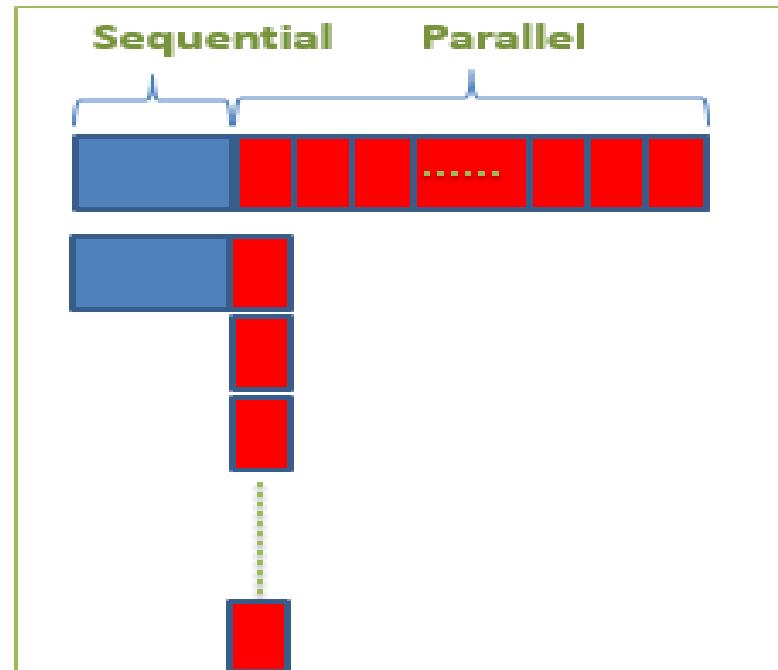


Make the Common Case Fast

Making the **common case fast** will tend to enhance performance better than optimizing the rare case. Ironically, the common case is often simpler than the rare case and hence is usually easier to enhance..

Amdahl's Law:

- ✓ A program needs 20 hours to complete
- ✓ No matter what, 1 hour needs to run sequentially
- ✓ Only, rest of the 19 hours can run in parallel

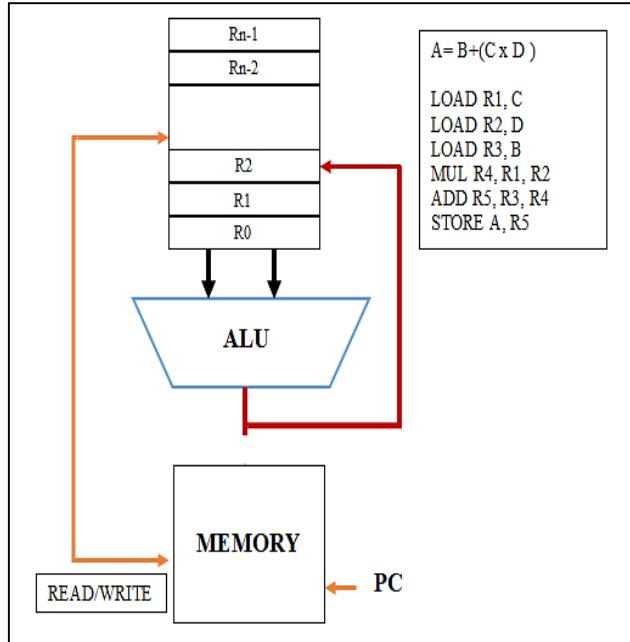


Make the Common Case Fast

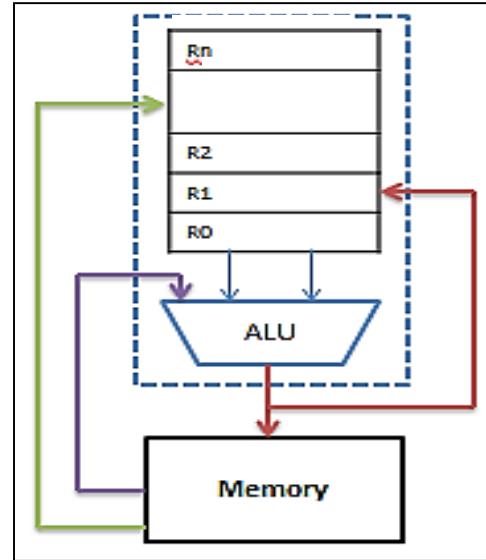
The load-store computer architecture separates instructions into

1. Memory access operations (load and stores) and
2. Into operations that operate on the data in the register file (register to register or register to immediate).

Load-Store



Direct Memory Access

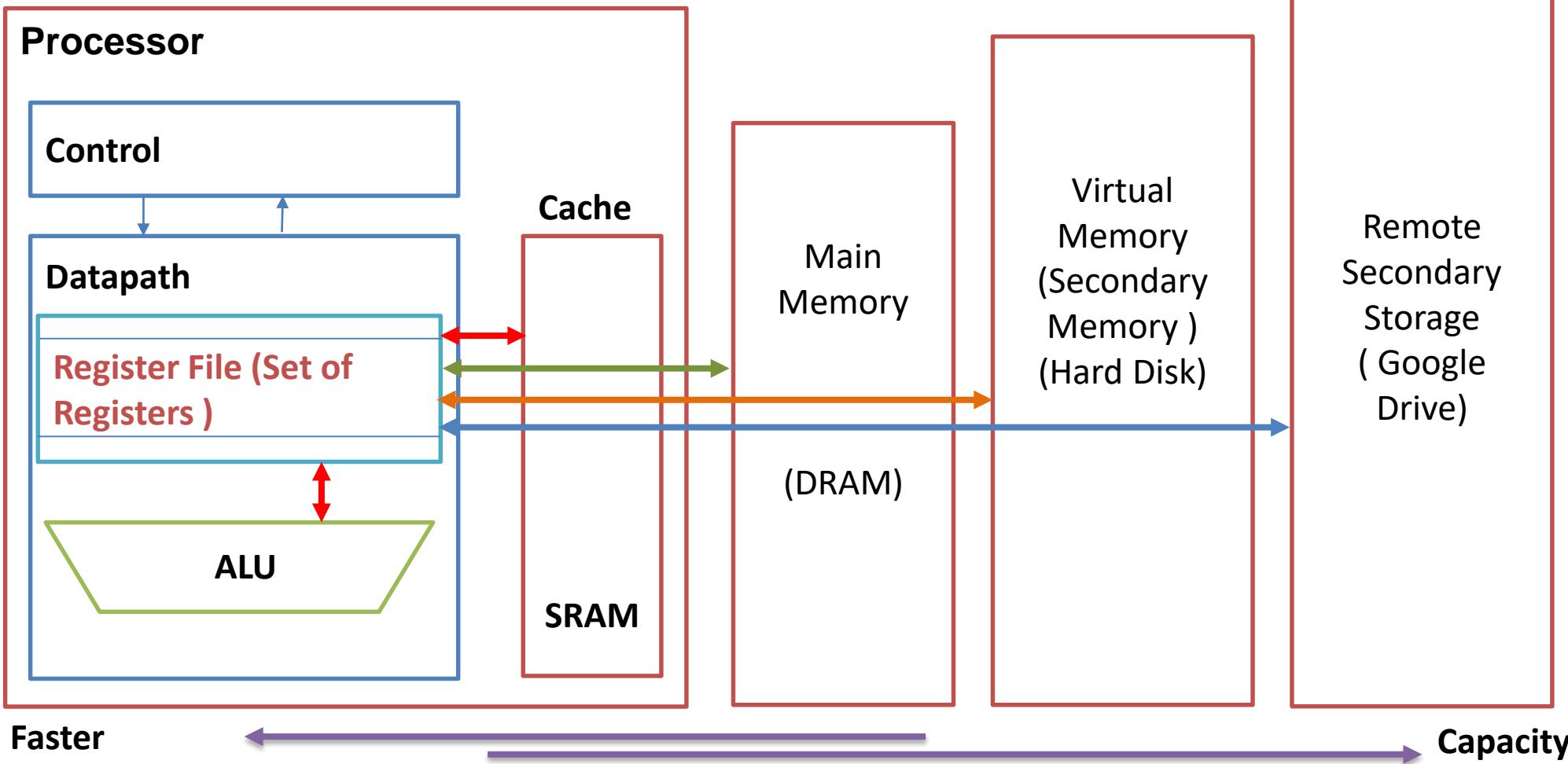


By not combining memory accesses with data manipulation operations, the processor's complexity is reduced which enables making the common case fast

Computer Abstractions and Technology

Seven Great Ideas in Computer Architecture

Hierarchy of Memories



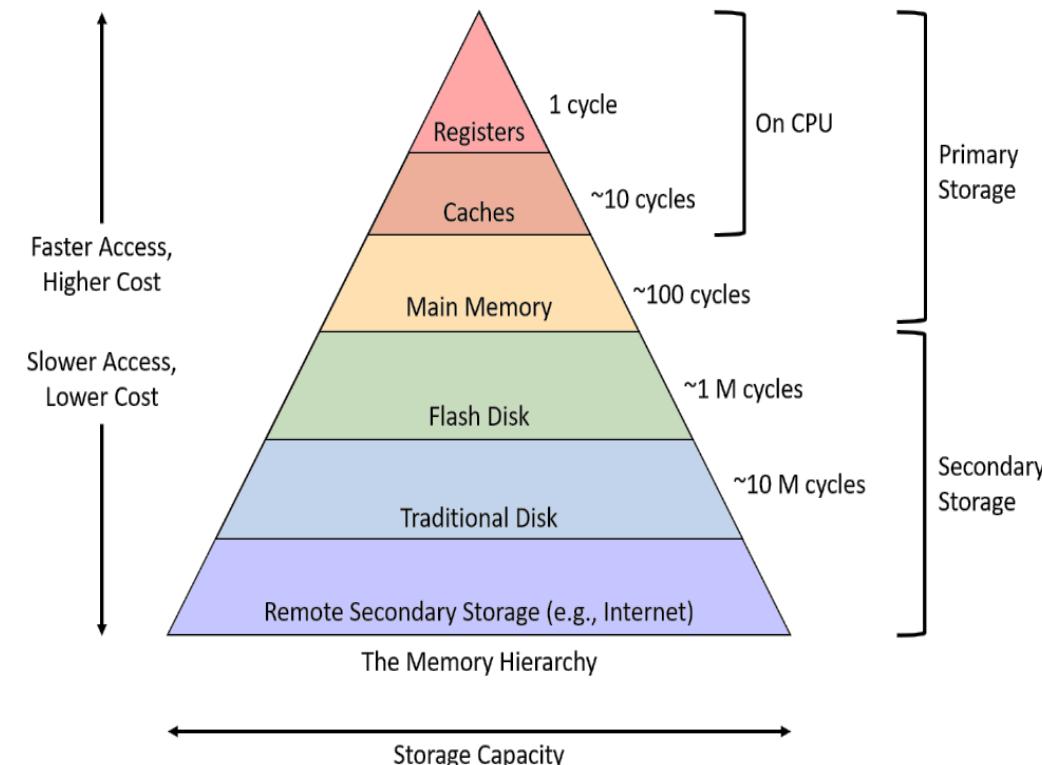
Hierarchy of Memories

Programmers want the memory to be

- Fast
- Large
- Cheap

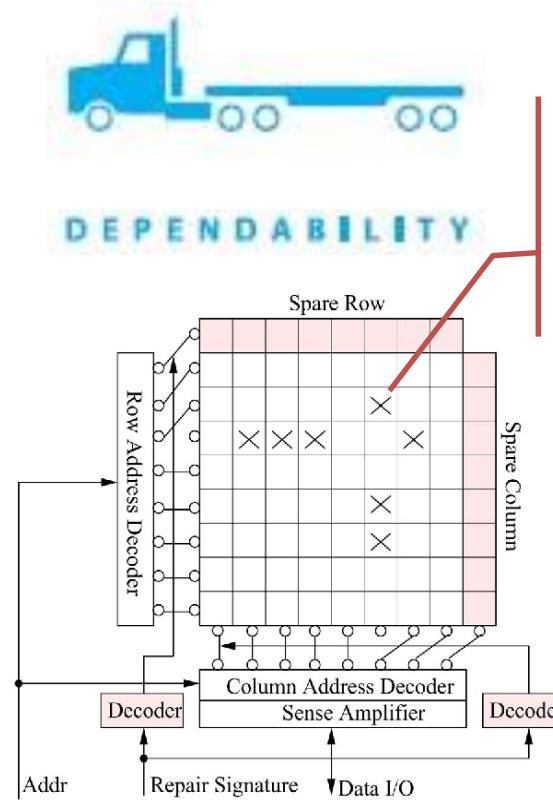
Why memory is more important?

- Memory speed often shapes performance
- Capacity limits the size of problems that can be solved
- The cost of memory today is often the majority of computer cost



Dependability via Redundancy

- Computers not only need to be fast; they **need to be dependable**.
- Since any **physical device can fail**, we make systems **dependable** by including **redundant components that can take over when a failure occurs and to help detect failures**.
- Failing piece does not make whole system fail
- Increasing transistor density reduce the cost of redundancy .



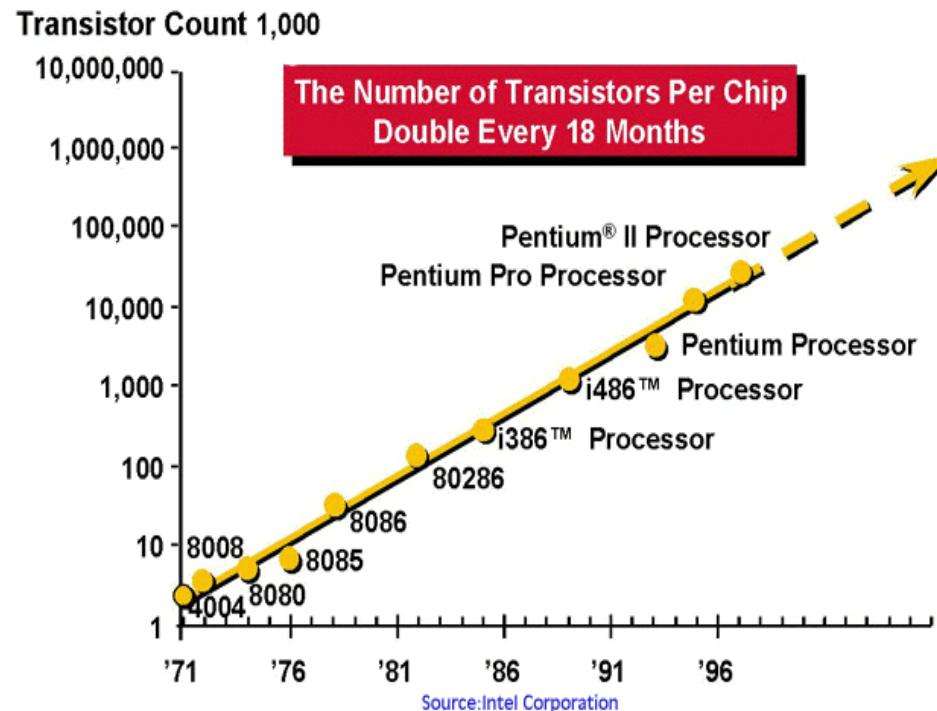
DRAM Architecture

Failing Column/ row does not make DRAM IC manufactured to be rejected

Moore's Law



- One of the founders of Intel
- Integrated circuit resources(Number of Transistors) would double every 18-24 Months. His prediction was accurate.



- **No exponential growth can last forever**, and Moore's law is no longer accurate.

Summary

Use Abstraction to Simplify Design

Hierarchy of Memories

Performance via Pipelining

Performance via Parallelism

Performance via Prediction

Make the Common Case Fast

Dependability via Redundancy

Technologies used over time

| Year | Technology used in computers | Relative performance/unit cost |
|------|--------------------------------------|--------------------------------|
| 1951 | Vacuum tube | 1 |
| 1965 | Transistor | 35 |
| 1975 | Integrated circuit | 900 |
| 1995 | Very large-scale integrated circuit | 2,400,000 |
| 2020 | Ultra large-scale integrated circuit | 500,000,000,000 |

FIGURE 1.10 Relative performance per unit cost of technologies used in computers over time. Source: Computer Museum, Boston, with 2013 extrapolated by the authors. See  [Section 1.13](#).

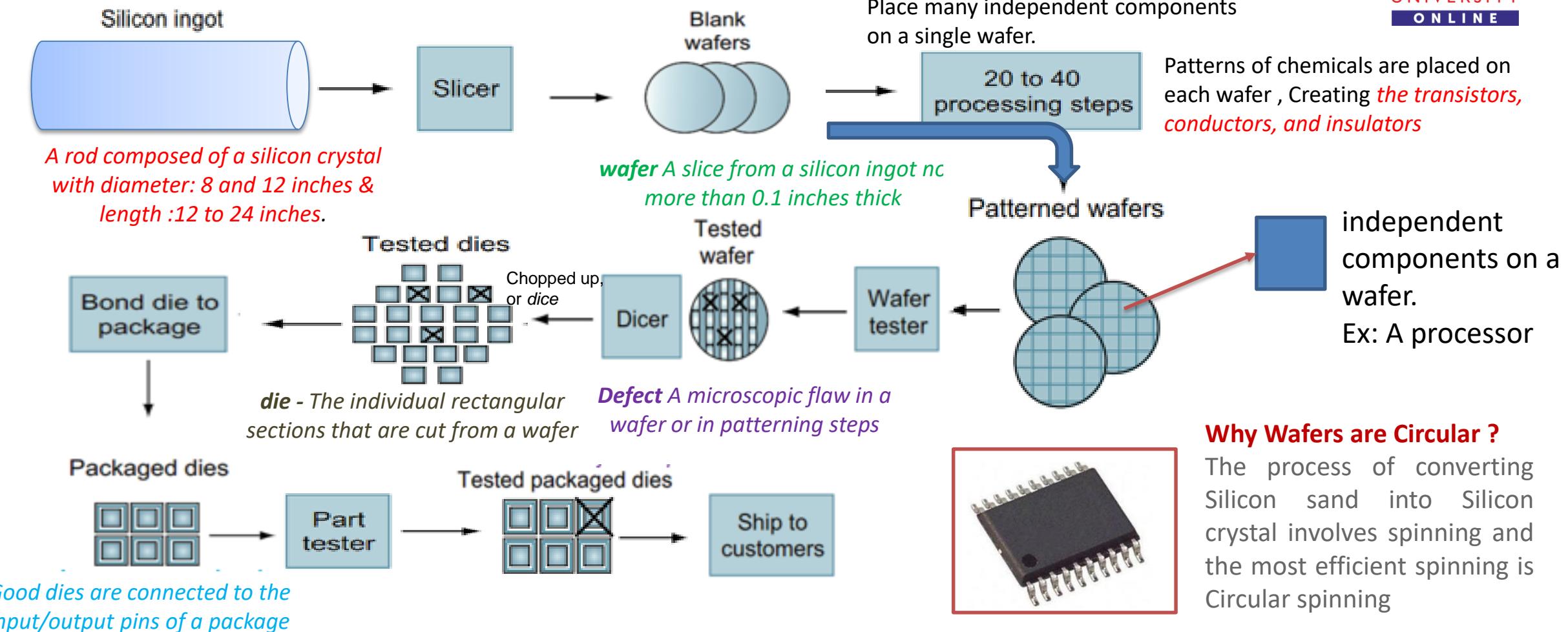
- A **transistor** is simply an on/off switch controlled by electricity.
- The *integrated circuit* (IC) combined dozens to hundreds of transistors into a single chip.
- When Gordon Moore predicted the continuous doubling of resources, he was forecasting the growth rate of the number of transistors per chip.

The Chip manufacturing process

- The manufacture of a chip begins with **silicon**, a substance found in sand. Because silicon does not conduct electricity well, it is called a **semiconductor**.
- With a special chemical process, it is possible to add materials to silicon that allow tiny areas to transform into one of three devices:
 1. **Excellent conductors of electricity** (using either microscopic copper or aluminum wire)
 2. **Excellent insulators from electricity** (like plastic sheathing or glass)
 3. **Areas that can conduct or insulate under specific conditions** (as a switch)
- Transistors fall into the last category.
- **A VLSI circuit**, then, is just **billions of combinations of conductors, insulators, and switches** manufactured in a single small package.



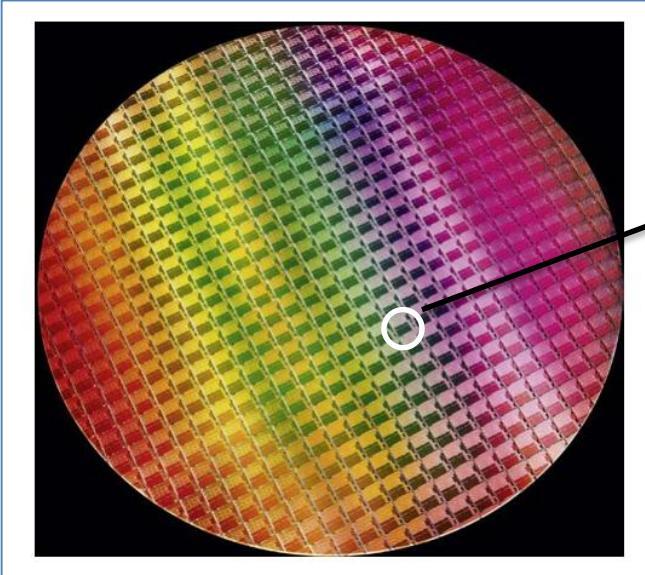
The chip manufacturing process



Good dies are connected to the input/output pins of a package

The chip manufacturing process

A photograph of a wafer containing microprocessors before they have been diced



A 12-inch (300mm) wafer this 10nm wafer contains **10th Gen Intel® Core™ processors**, code-named “Ice Lake” (Courtesy Intel).

An individual microprocessor die.



The processor integrated circuit inside the A12 package. The size of chip is 8.4 by 9.91 mm, and it was manufactured originally in a 7-nm process.

It has

- two identical ARM processors
- four small cores,
- a Graphics Processing Unit (GPU)
- a domain-specific accelerator for neural networks called the NPU
- Second-level cache memory (L2) banks for the big and small cores
- Interfaces to the main memory (DDR DRAM).

(CourtesyTechInsights,
www.techinsights.com)

Cost and Prize of the Chip

Yield : The percentage of good dies from the total number of dies on the wafer.

The cost of an integrated circuit can be **expressed in three simple equations:**

$$\text{Dies per wafer} = \frac{\text{Wafer area}}{\text{Die area}} \quad \dots \dots \dots \quad (1)$$

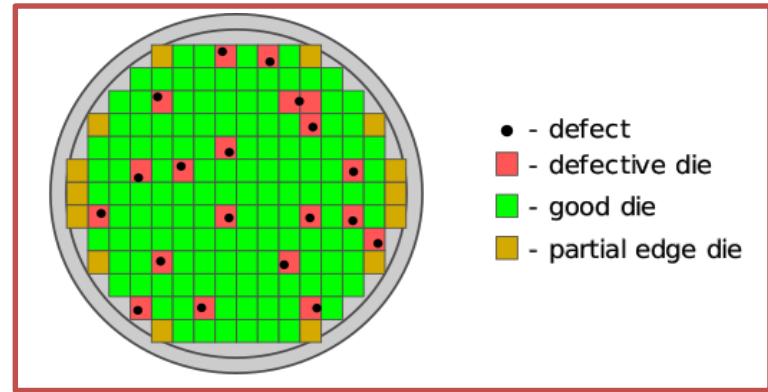
Equation (10) is an approximation, **since it does not subtract the area near the border of the round wafer** that cannot accommodate the rectangular dies

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{yield}} \quad \dots \dots \dots \quad (2)$$

Equation (3) **Empirical observations** of Die-yield (Assuming wafer yield is 100%) at IC factories, with the exponent related to the number of critical processing steps.

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}))^N} \quad \dots \dots \dots \quad (3)$$

N is the process-complexity factor.



Computer Abstractions and Technology

Technologies for Building Processors and Memory

Cost and Prize of the Chip

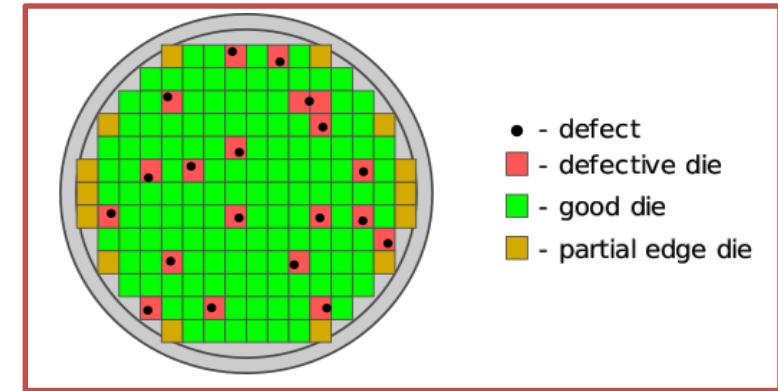
Yield : The percentage of good dies from the total number of dies on the wafer.

N in eqn(3) is the process-complexity factor & depends on technology used.

For **130nm** processes , **$N=4$**

For **28 nm** processes in 2017, **N is 7.5–9.5.**

For **16 nm** process, N ranges from **10 to 14.**



Note: 1. Factor N should be known to calculate the yield. In case factor N is not given use the following empirical formula i.e., eqn(4).

$$\text{Yield} = \frac{1}{\left(1 + \text{defects per area} \times \frac{\text{die area}}{2}\right)^N} \quad \text{----- (4)}$$

If wafer yield is not 100%, then use the following formula

$$\text{Die yield} = \text{Wafer yield} \times 1 / \left(1 + \text{Defects per unit area} \times \text{Die area}\right)^N \quad \text{----- (5)}$$



Computer Abstractions and Technology

Technologies for Building Processors and Memory

Cost and Prize of the Chip

Defects per Area can be derived from eqn (4) or eqn(5). From eqn (4)

$$\text{Yield} \left(1 + \text{defects per area} \times \frac{\text{diearea}}{2} \right)^2 = 1$$

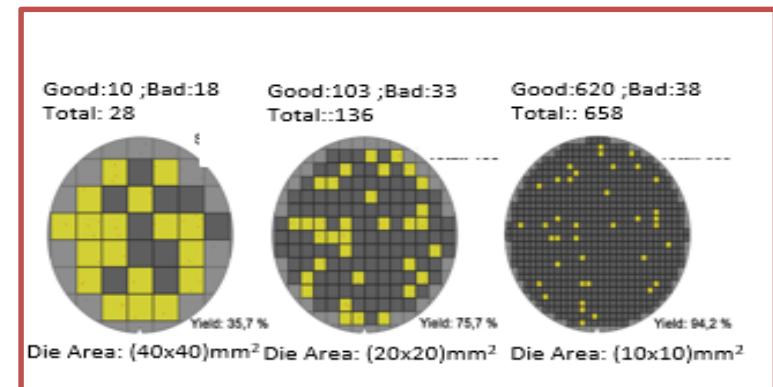
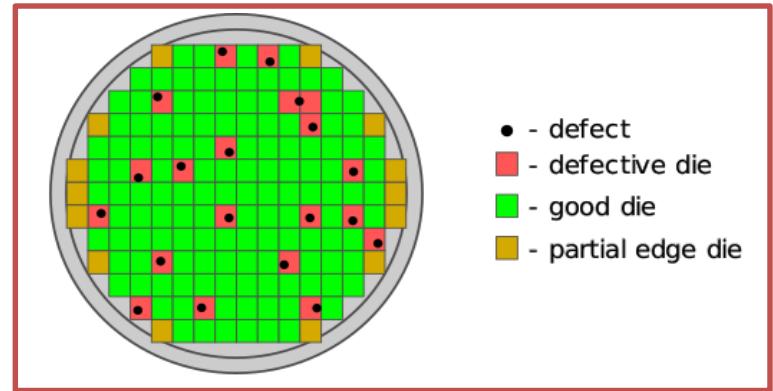
By taking square root

$$\sqrt{\text{Yield}} \left(1 + \text{defects per area} \times \frac{\text{diearea}}{2} \right) = 1$$

$$\left(\sqrt{\text{Yield}} + \sqrt{\text{Yield}} \times \text{defects per area} \times \frac{\text{diearea}}{2} \right) = 1$$

$$\sqrt{\text{Yield}} \times \text{defects per area} \times \frac{\text{diearea}}{2} = 1 - \sqrt{\text{Yield}}$$

$$\text{defects per area} = \left(\frac{1 - \sqrt{\text{Yield}}}{\sqrt{\text{Yield}} \times \frac{\text{diearea}}{2}} \right) \quad \text{--- (6)}$$



Computer Abstractions and Technology

Technologies for Building Processors and Memory

Numerical

Question: Find the number of dies per 300mm (30 cm) wafer for a die that is 1.5 cm on a side and for a die that is 1.0cm on a side.

Case 1:

$$\text{Number of dies} \approx \left(\frac{\text{Wafer Area}}{\text{Die Area}} \right) \approx \left(\frac{\pi r^2}{1.5 \times 1.5} \right) = \left(\frac{3.14 \times (15)^2}{1.5 \times 1.5} \right) = 314$$

Case 2:

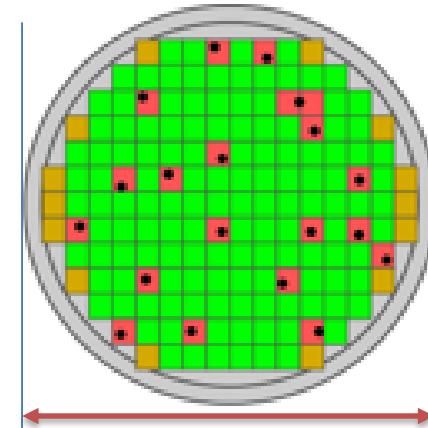
$$\text{Number of dies} \approx \left(\frac{\text{Wafer Area}}{\text{Die Area}} \right) \approx \left(\frac{\pi r^2}{1.0 \times 1.0} \right) = \left(\frac{3.14 \times (15)^2}{1.0 \times 1.0} \right) = 706$$

Die per Wafer by subtracting the area near the border

$$\# \text{Dies per wafer} = \frac{\pi \times (\text{Wafer radius})^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2} \times \text{Die area}}$$

(sensitive to die size)

(# of dies along the edge)



| | |
|-------------------|-------|
| Diameter of Wafer | 300mm |
| Square Die width | 1.5cm |
| Square Die width | 1cm |

Computer Abstractions and Technology

Technologies for Building Processors and Memory

Numerical

Question: For wafer 300mm diameter, Find the die yield for dies that are 1.5 cm on a side and 1.0 cm on a side, assuming a defect density of 0.047 per cm² and N=12.

Case 1: The total Die Area = (1.5 x 1.5) cm² = 2.25 cm²

As N is given,

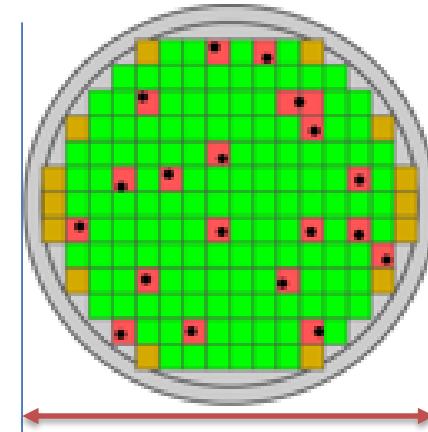
$$\text{Yield} = \left(\frac{1}{(1 + \text{defects per area} \times \text{die area})^N} \right) = \left(\frac{1}{(1 + 0.047 \times 2.25)^{12}} \right) = 0.2993$$

Out of 314 dies 94 dies will be good

Case 2: The total Die Area = (1.0 x 1.0) cm² = 1.00 cm²

$$\text{Yield} = \left(\frac{1}{(1 + \text{defects per area} \times \text{die area})^N} \right) = \left(\frac{1}{(1 + 0.047 \times 1)^{12}} \right) = 0.05762$$

Out of 706 dies 407 dies will be good



| | |
|-------------------|-------|
| Diameter of Wafer | 300mm |
| Square Die width | 1.5cm |
| Square Die width | 1cm |

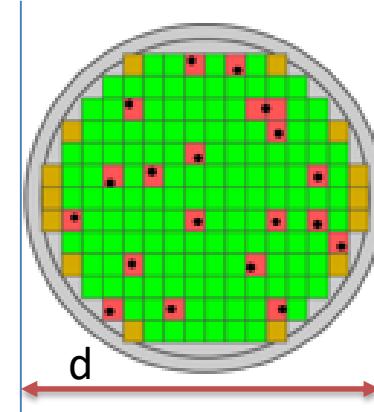
Computer Abstractions and Technology

Technologies for Building Processors and Memory

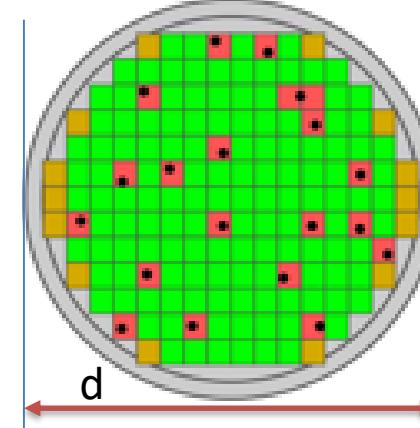
Numerical

Question: Assume a 15 cm diameter wafer has a cost of 12, contains 84 dies, and has 0.020 defects/cm². Assume a 20 cm diameter wafer has a cost of 15, contains 100 dies, and has 0.031 defects/cm².

- Find the yield for both wafers.
- Find the cost per die for both wafers.
- If the number of dies per wafer is increased by 10% and the defects per area unit increases by 15%, find the die area and yield.
- Assume a fabrication process improves the yield from 0.92 to 0.95. Find the defects per area unit for each version of the technology given a die area of 200 mm².



| | |
|-------------------------|-------|
| Diameter of Wafer | 15cm |
| Cost of wafer | 12 |
| Number of Die | 84 |
| Defects/cm ² | 0.020 |



| | |
|-------------------------|-------|
| Diameter of Wafer | 20cm |
| Cost of wafer | 15 |
| Number of Die | 100 |
| Defects/cm ² | 0.031 |

Computer Abstractions and Technology

Technologies for Building Processors and Memory



PES
UNIVERSITY
ONLINE

Solution

$$a) \text{ Die Area}(15\text{cm}) = \left(\frac{\text{Wafer Area}}{\text{Dies per wafer}} \right) = \left(\frac{\pi r^2}{84} \right) = \left(\frac{3.14 \times 7.5^2}{84} \right) = 2.10 \text{ cm}^2$$

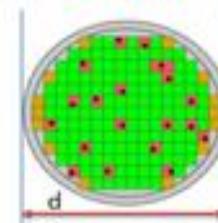
$$\text{Die Area}(20\text{cm}) = 3.14 \text{ cm}^2$$

$$\text{Yield}(15\text{cm}) = \left(\frac{1}{\left(1 + \text{Defects per area} \times \frac{\text{Die Area}}{2} \right)^2} \right) = \left(\frac{1}{\left(1 + 0.020 \times \frac{2.10}{2} \right)^2} \right) = 0.9593$$

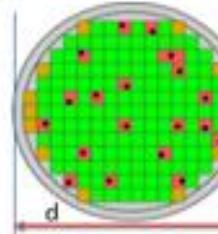
$$\text{Yield}(20\text{cm}) = 0.9093$$

$$b) \text{ Cost per Die (15cm)} = \left(\frac{\text{Cost per Wafer}}{\text{Dies per wafer} \times \text{Yield}} \right) = \left(\frac{12}{84 \times 0.9593} \right) = 0.1489$$

$$\text{Cost per Die (20cm)} = \left(\frac{15}{100 \times 0.9093} \right) = 0.1650$$



| | |
|-------------------------|-------|
| Diameter of Wafer | 15cm |
| Cost of wafer | 12 |
| Number of Die | 84 |
| Defects/cm ² | 0.020 |
| N | 12 |



| | |
|-------------------------|-------|
| Diameter of Wafer | 20cm |
| Cost of wafer | 125 |
| Number of Die | 100 |
| Defects/cm ² | 0.031 |
| N | 12 |

Computer Abstractions and Technology

Technologies for Building Processors and Memory

c) Die Area(15cm) = $\left(\frac{\text{Wafer Area}}{\text{Dies per wafer}}\right) = \left(\frac{\Pi r^2}{84 \times 1.1}\right) = \left(\frac{3.14 \times 7.5^2}{84 \times 1.1}\right) = 1.91 \text{ cm}^2$

Die Are(20cm) = $\left(\frac{3.14 \times 7.52}{100 \times 1.1}\right) = 2.86 \text{ cm}^2$

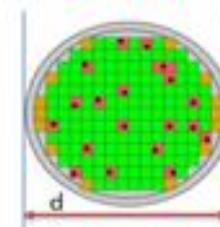
Yield(15cm) = $\left(\frac{1}{\left(1 + \text{Defects per area} \times \frac{\text{Die Area}}{2}\right)^2}\right) = \left(\frac{1}{\left(1 + 0.020 \times 1.15 \times \frac{1.91}{2}\right)^2}\right) = 0.9575$

Yield(20cm) = $\left(\frac{1}{\left(1 + 0.031 \times 1.15 \times \frac{2.86}{2}\right)^2}\right) = 0.9082$

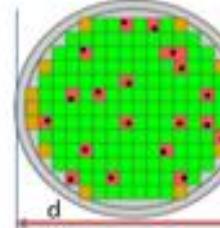
d) defects per area ($y = 0.92$) = $\left(\frac{1 - \sqrt{\text{Yield}}}{\sqrt{\text{Yield}} \times \frac{\text{diearea}}{2}}\right) = \left(\frac{1 - \sqrt{0.92}}{\sqrt{0.92} \times \frac{2\text{cm}}{2}}\right) = 0.043 \text{ defects/cm}^2$

defects per area ($y = 0.95$) = $\left(\frac{1 - \sqrt{\text{Yield}}}{\sqrt{\text{Yield}} \times \frac{\text{diearea}}{2}}\right) = \left(\frac{1 - \sqrt{0.95}}{\sqrt{0.95} \times \frac{2\text{cm}}{2}}\right) = 0.0259 \text{ defects/cm}^2$

As the Yield increases , the defects per unit area decreases



| | |
|-------------------------|-------|
| Diameter of Wafer | 15cm |
| Cost of wafer | 12 |
| Number of Die | 84 |
| Defects/cm ² | 0.020 |
| N | 12 |

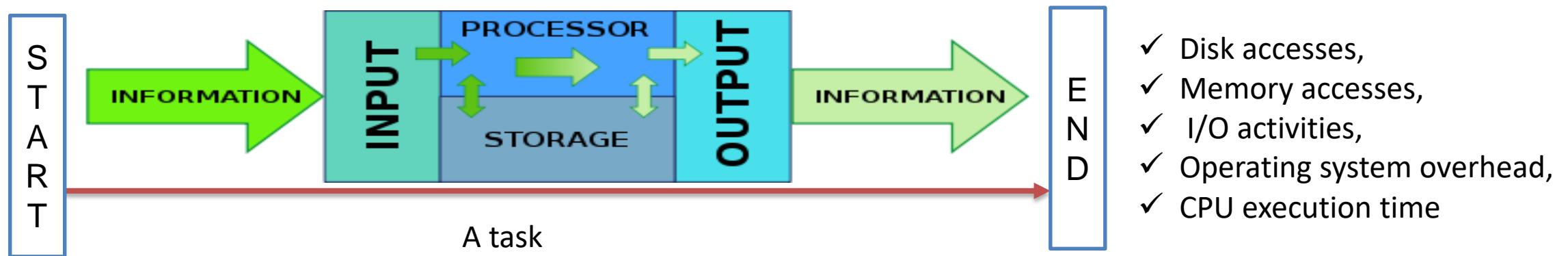


| | |
|-------------------------|-------|
| Diameter of Wafer | 20cm |
| Cost of wafer | 125 |
| Number of Die | 100 |
| Defects/cm ² | 0.031 |
| N | 12 |

Basic Definitions

Response time/ Execution time: The total time required for the computer to complete a task, including disk accesses, memory accesses, I/O activities, operating system overhead, CPU execution time, and so on.

Important to Individual users



Throughput / bandwidth : it is the number of tasks completed per unit time.

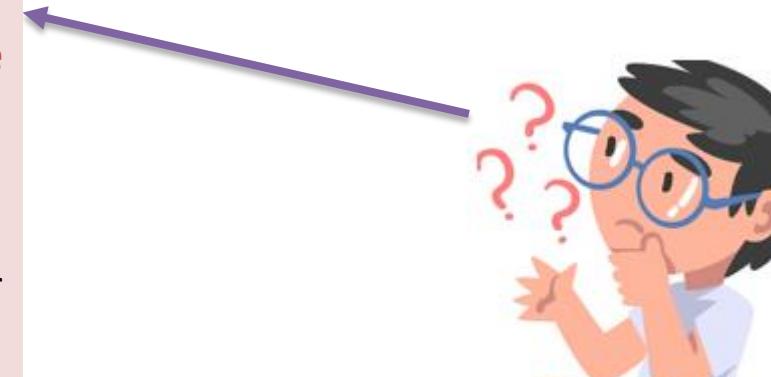
Basic Definitions

Task : Throughput and Response Time

Do the following changes to a computer system **increase throughput, decrease response time, or both?**

Case 1. Replacing the processor in a computer with a faster version

Case 2. Adding additional processors to a system that uses multiple processors for separate tasks.

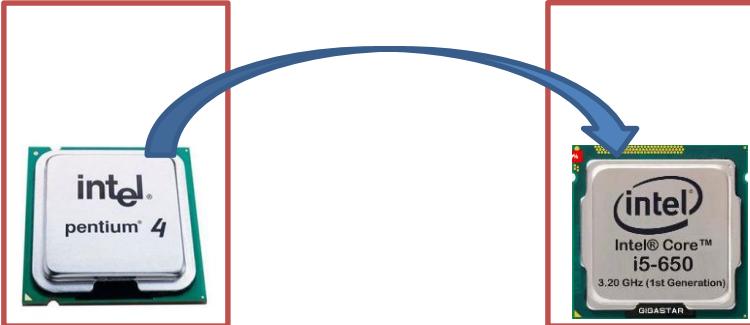


Mr. Think

Basic Definitions

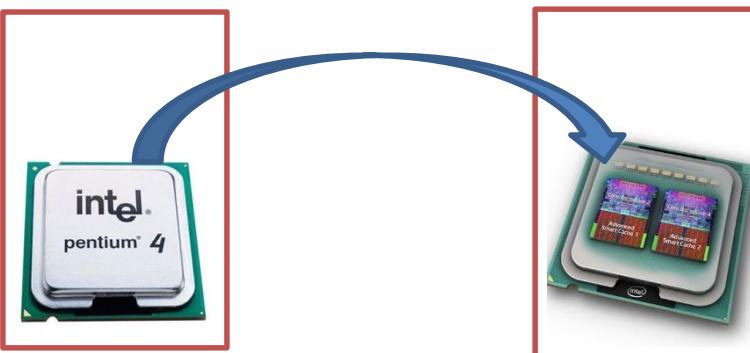
Task : Throughput and Response Time

Case 1 : Replacing the processor in a computer with a faster version



Decreasing response time almost always improves throughput. Hence, in this case, both response time and throughput are improved.

Case 2: Adding additional processors to a system that uses multiple processors for separate tasks.



In this case, No one task gets work done faster, so only throughput increases.

Performance of a Computer

- To maximize performance, we want to minimize response time or execution time for some task. i.e., Lesser Response time Maximum is the performance (Performance and Response or Execution time are Inversely proportional).
- Thus, we can relate performance and execution time for a computer X:

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

- For two computers X and Y, if the performance of X is greater than the performance of Y, we have

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

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



Computer X



Computer Y

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

Performance of a Computer

How to relate the performance of two different computers quantitatively

$$\frac{\text{Performance}_X}{\text{Performance}_Y} = n$$

- If X is n times as fast as Y, then the execution time on Y is n times as long as it is on X:

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

Task: Run same program (C program) on two different laptops and check the time taken by two laptops and identify the reason

Relative Performance



Computer X runs a program in 10 seconds



Computer Y runs the same program in 15 seconds

How much faster is A than B?

Performance of a Computer

Relative Performance



Computer X runs a program in 10 seconds



Computer Y runs the same program in 15 seconds

How much faster is A than B?

We know that A is n times as fast as B if

$$\frac{\text{Performance}_A}{\text{Performance}_B} = \frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Thus the performance ratio is

$$\frac{15}{10} = 1.5$$

and A is therefore 1.5 times as fast as B.

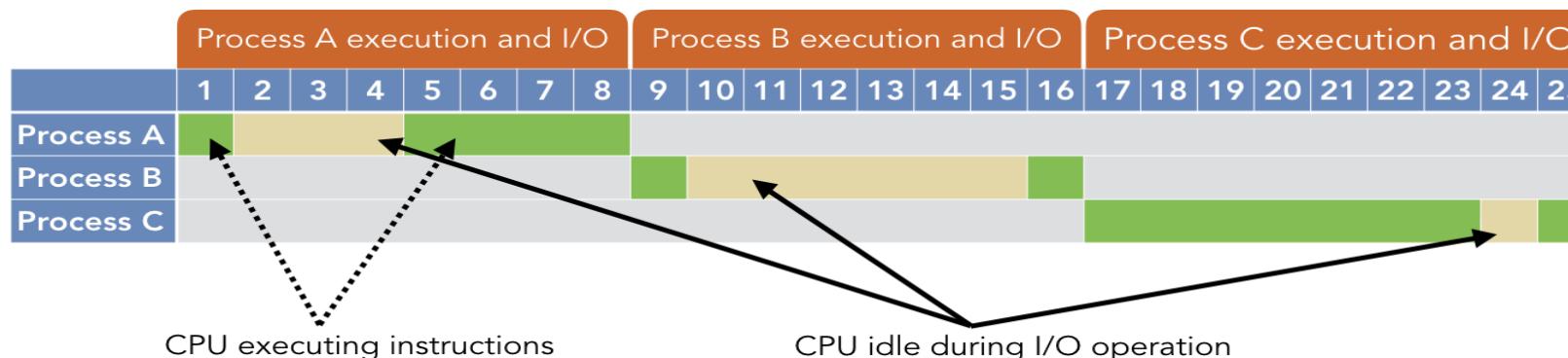
Measuring Performance

“The computer that performs the same amount of work in the least time is the fastest.”

Wall clock time, Response time, or Elapsed time: The total time to complete a task, including

- ✓ disk accesses,
- ✓ memory accesses,
- ✓ input/output (I/O) activities,
- ✓ operating system overhead—everything.

Three sequential uni-programming processes.

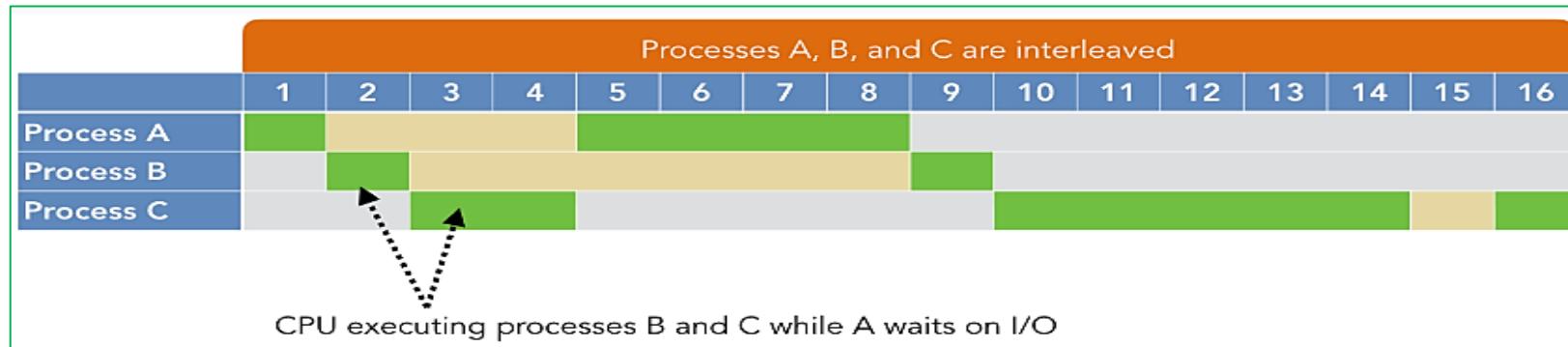


Measuring Performance

Computers are often **shared**, however, and a processor may work on several programs simultaneously.

In such cases, **the system may try to optimize throughput rather than attempt to minimize the elapsed time for one program**.

Multiprogramming environment.



The **green regions** indicate **CPU execution** and the **yellow indicates I/O operations**. However, note that processes B and C can run while A is waiting on its I/O operation. Similarly, A and C execute while B is waiting on I/O operations. As a result, the CPU is only completely idle while C's I/O operation is performed at time 15, because A and B have already run to completion

Optimize throughput

In such cases, the system may try to **optimize throughput** rather than attempt to **minimize the elapsed time for one program**.

Measuring Performance

CPU execution time (CPU time) : It is the time the CPU spends on computing a particular task and does not include time spent waiting for I/O or running other programs.

CPU time can be further divided into -

User CPU time: the amount of time the processor spends in running your application code

System CPU time: the time spent running code in the operating system kernel on behalf of your program

We will use **CPU performance to refer to User CPU time** and concentrate on the same.

CPU Performance and Its Factors

Different applications are sensitive to Different aspects of the performance of a computer system

Example: Applications running on servers, depend as much on I/O performance, which, in turn, relies on both hardware and software.

CPU Performance and Its Factors

CPU performance - the bottom-line performance measure is CPU execution time.

A simple formula relates the most basic metrics (clock cycles and clock cycle time) to CPU time:

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock cycle time}}$$

Alternatively, because clock rate and clock cycle time are inverses,

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

Who is responsible for improving the Number of clocks required for a program or the Length of the clock cycle

Hardware designer

CPU Performance and Its Factors

Computer A



Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.

The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

Computer B



Computer designer build a computer, B, which will run this program in 6 seconds

First find the CPU time for A

$$\text{CPU execution time} = \frac{\text{CPU clock cycles for a program}}{\text{Clockrate}}$$

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{Clock rate}_A}$$

$$10 \text{ seconds} = \frac{\text{CPU clock cycles}_A}{2 \times 10^9 \text{ cycles}} \text{ second}$$

$$\text{CPU clock cycles}_A = 10 \text{ seconds} \times 2 \times 10^9 \frac{\text{cycles}}{\text{second}} = 20 \times 10^9 \text{ cycles}$$

CPU Performance and Its Factors

Computer A



Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock.

The designer has determined that a substantial increase in the clock rate is possible, but this increase will affect the rest of the CPU design, **causing computer B to require 1.2 times as many clock cycles as computer A for this program**. What clock rate should we tell the designer to target?

Answer: 4 GHz

Computer B



Computer designer build a computer, B, which will run this program in 6 seconds

CPU time for B

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clockrate}}$$

$$\text{CPU time}_B = \frac{1.2 \times \text{CPU clock cycles}_A}{\text{Clock rate}_B}$$

$$6 \text{ seconds} = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{Clock rate}_B}$$

$$\text{Clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = \frac{0.2 \times 20 \times 10^9 \text{ cycles}}{\text{second}} = \frac{4 \times 10^9 \text{ cycles}}{\text{second}} = 4 \text{ GHz}$$

Instruction Performance

- The performance equations above did not include any reference to the number of instructions needed for the program
- Program is basically set of Instructions.
- The computer had to execute the instructions to run the program, therefore execution time must depend on the number of instructions in a program.
- Therefore, the number of clock cycles required for a program can be written as

$$\text{CPU execution time for a program} = \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

$$\text{CPU execution time for a program} = \text{CPU clock cycles for a program} \times \text{Clock cycle time}$$

$$\text{CPU clock cycles} = \text{Instructions for a program} \times \frac{\text{Average clock cycles per instruction}}$$

If Instructions retired =2000 and Avg clock cycles per instruction is 0.75, then
 $\text{CPU clock cycles} = 2000 \times 0.75 = 1500 \text{ clock cycles}$

Computer Abstractions and Technology

Performance



Instruction Performance

Clock cycles per instruction (CPI):One way of comparing two different implementations

- The **average number of clock cycles each instruction takes to execute.**
- Since different instructions may take different amounts of time depending on what they do, CPI is an average of all the instructions executed in the program.
- **CPI provides one way of comparing two different implementations** of the identical instruction set architecture, since the number of instructions executed for a program will, of course, be the same.

Instruction Performance

Suppose we have two implementations of the same instruction set architecture.

Computer A



Computer A has a clock cycle time of 250ps and a CPI of 2.0 for some program

Computer B



computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program

Which computer is faster for this program and by how much?

First, find the number of processor clock cycles for each computer:

$$\text{CPU clock cycles}_A = I \times 2.0$$

$$\text{CPU clock cycles}_B = I \times 1.2$$

Now we can compute the CPU time for each computer:

$$\begin{aligned}\text{CPU time}_A &= \text{CPU clock cycles}_A \times \text{Clock cycle time} \\ &= I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}\end{aligned}$$

Computer Abstractions and Technology

Performance

Instruction Performance

Suppose we have two implementations of the same instruction set architecture.

Computer A



Computer B



Computer A has a clock cycle time of 250ps and a CPI of 2.0 for some program

computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program

Which computer is faster for this program and by how much?

Now we can compute the CPU time for each computer:

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

Computer Abstractions and Technology

Performance

Instruction Performance

Suppose we have two implementations of the same instruction set architecture.

Computer A



Computer A has a clock cycle time of 250ps and a CPI of 2.0 for some program

Computer B



computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program

Which computer is faster for this program and by how much?

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

Conclusion:

Computer A is 1.2 times as fast as computer B for this program.

The Classic CPU Performance Equation

Basic performance equation in terms of **instruction count** (the number of instructions executed by the program), **CPI**, and **clock cycle time**:

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

- The three key factors that affect performance Instruction Count, CPI and Clock rate.
- We can use these formulas to
 - ✓ Compare two different implementations or
 - ✓ To evaluate a design alternative if we know its impact on these three parameters.

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a computer. The hardware designers have supplied the following facts:

| | CPI for each instruction class | | |
|-----|--------------------------------|---|---|
| | A | B | C |
| CPI | 1 | 2 | 3 |

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

| Code sequence | Instruction counts for each instruction class | | |
|---------------|---|---|---|
| | A | B | C |
| 1 | 2 | 1 | 2 |
| 2 | 4 | 1 | 1 |

Which code sequence executes the most instructions? Which will be faster?
What is the CPI for each sequence?

- Number of Instructions in Code Sequence
 $1 : 2+1+2=5$
- Number of Instructions in Code Sequence
 $2 : 4+1+2=6$

Code Sequence 1 executes fewer Instructions

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a computer. The hardware designers have supplied the following facts:

| | CPI for each instruction class | | |
|-----|--------------------------------|---|---|
| | A | B | C |
| CPI | 1 | 2 | 3 |

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

| Code sequence | Instruction counts for each instruction class | | |
|---------------|---|---|---|
| | A | B | C |
| 1 | 2 | 1 | 2 |
| 2 | 4 | 1 | 1 |

Which code sequence executes the most instructions? Which will be faster?
What is the CPI for each sequence?

- Total number of clock cycles for sequence 1:
 $= 2 \times 1 + 1 \times 2 + 2 \times 3 = 10 \text{ clock cycles}$
- Total number of clock cycles for sequence 1:
 $= 4 \times 1 + 1 \times 2 + 1 \times 3 = 9 \text{ clock cycles}$

So code sequence 2 is faster, even though it executes one extra instruction.

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a computer. The hardware designers have supplied the following facts:

| | CPI for each instruction class | | |
|-----|--------------------------------|---|---|
| | A | B | C |
| CPI | 1 | 2 | 3 |

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

| Code sequence | Instruction counts for each instruction class | | |
|---------------|---|---|---|
| | A | B | C |
| 1 | 2 | 1 | 2 |
| 2 | 4 | 1 | 1 |

Which code sequence executes the most instructions? Which will be faster?
What is the CPI for each sequence?

- CPI of Sequence 1 and 2 are

$$CPI_1 = \frac{\text{CPU clock cycles}_1}{\text{Instruction count}_1} = \frac{10}{5} = 2.0$$

$$CPI_2 = \frac{\text{CPU clock cycles}_2}{\text{Instruction count}_2} = \frac{9}{6} = 1.5$$

1.6 [4] <§1.6> Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

- a. Which processor has the highest performance expressed in instructions per second?
- b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.
- c. We are trying to reduce the execution time by 30%, but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

1.6

- a. performance of P1 (instructions/sec) = $3 \times 10^9 / 1.5 = 2 \times 10^9$
performance of P2 (instructions/sec) = $2.5 \times 10^9 / 1.0 = 2.5 \times 10^9$
performance of P3 (instructions/sec) = $4 \times 10^9 / 2.2 = 1.8 \times 10^9$
- b. cycles(P1) = $10 \times 3 \times 10^9 = 30 \times 10^9$ s
cycles(P2) = $10 \times 2.5 \times 10^9 = 25 \times 10^9$ s
cycles(P3) = $10 \times 4 \times 10^9 = 40 \times 10^9$ s
- c. No. instructions(P1) = $30 \times 10^9 / 1.5 = 20 \times 10^9$
No. instructions(P2) = $25 \times 10^9 / 1 = 25 \times 10^9$
No. instructions(P3) = $40 \times 10^9 / 2.2 = 18.18 \times 10^9$
 $CPI_{new} = CPI_{old} \times 1.2$, then CPI(P1) = 1.8, CPI(P2) = 1.2, CPI(P3) = 2.6
 $f = \text{No. instr.} \times CPI / \text{time}$, then
 $f(P1) = 20 \times 10^9 \times 1.8 / 7 = 5.14 \text{ GHz}$
 $f(P2) = 25 \times 10^9 \times 1.2 / 7 = 4.28 \text{ GHz}$
 $f(P3) = 18.18 \times 10^9 \times 2.6 / 7 = 6.75 \text{ GHz}$

The basic components of performance and how each is measured.

| Components of performance | Units of measure |
|------------------------------------|--|
| CPU execution time for a program | Seconds for the program |
| Instruction count | Instructions executed for the program |
| Clock cycles per instruction (CPI) | Average number of clock cycles per instruction |
| Clock cycle time | Seconds per clock cycle |

FIGURE 1.15 The basic components of performance and how each is measured.

IPC (instruction per Clock Cycle):

- Although you might expect that the minimum CPI is 1.0, some processors fetch and execute multiple instructions per clock cycle.
- To reflect that approach, some designers invert CPI to talk about **IPC**, or *instructions per clock cycle*.
- If a processor executes on average two instructions per clock cycle, then it has an IPC of 2 and hence a CPI of 0.5.

CISC Architecture – Minimum CPI is 1

Single Cycle Processor–CPI is always 1

Pipelined Processor–CPI is greater than 1

In some Processor–CPI is less than 1 which means multiple Instructions per clock cycle.

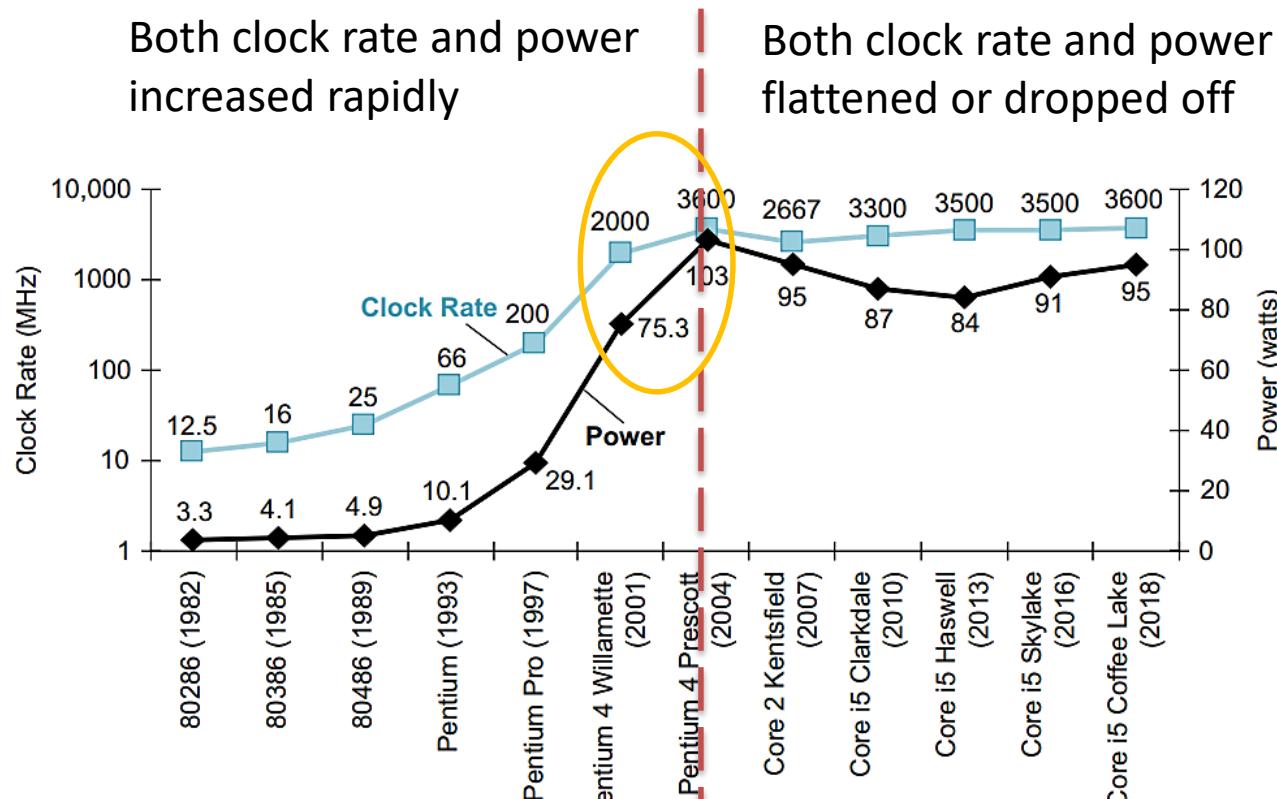
Computer Abstractions and Technology

The Power Wall

The Clock rate and Power for Intel x86 microprocessors over nine generations and 36 years

Both clock rate and power increased rapidly

Both clock rate and power flattened or dropped off



How could clock rates grow by a factor of 1000 while power increased by only a factor of 30?



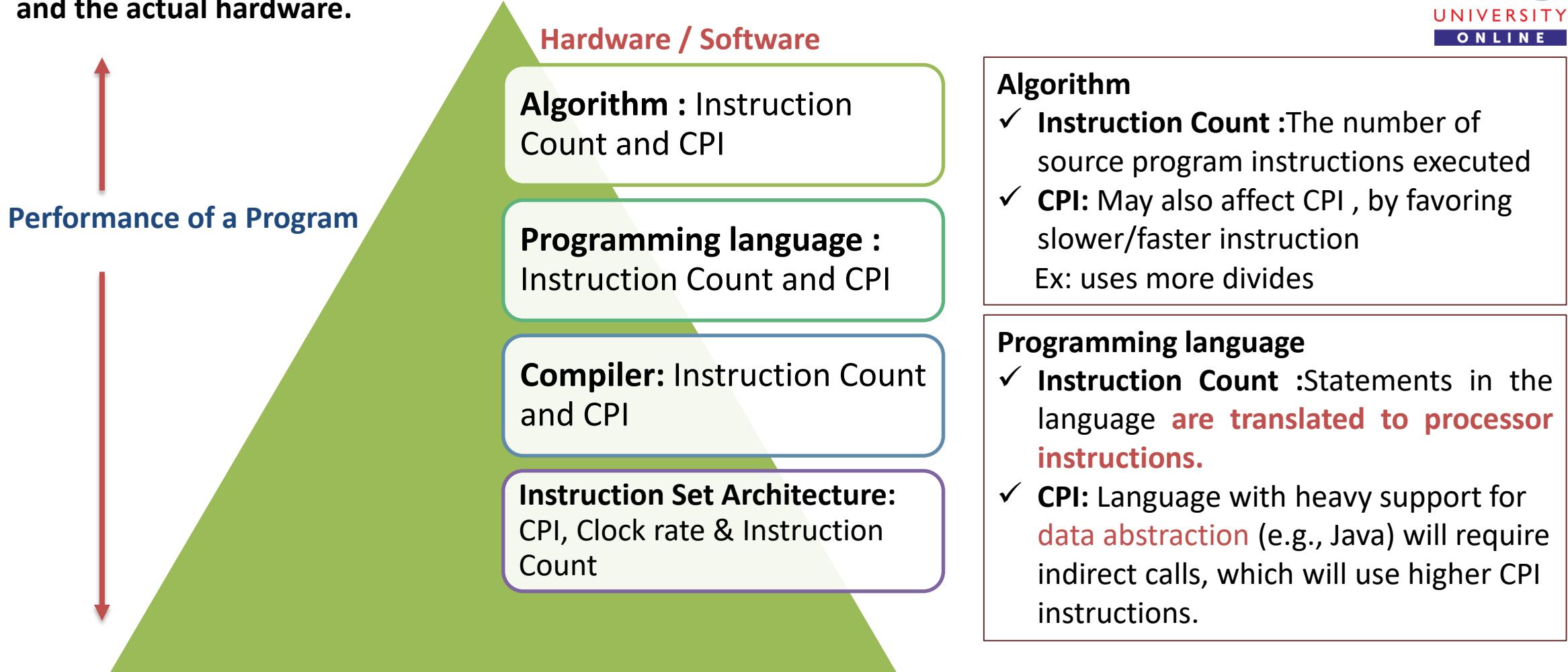
Energy and thus power can be reduced by lowering the voltage, which occurred with each new generation of technology, and power is a function of the voltage².

$$\text{Power} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

FIGURE 1.16 Clock rate and power for Intel x86 microprocessors over nine generations

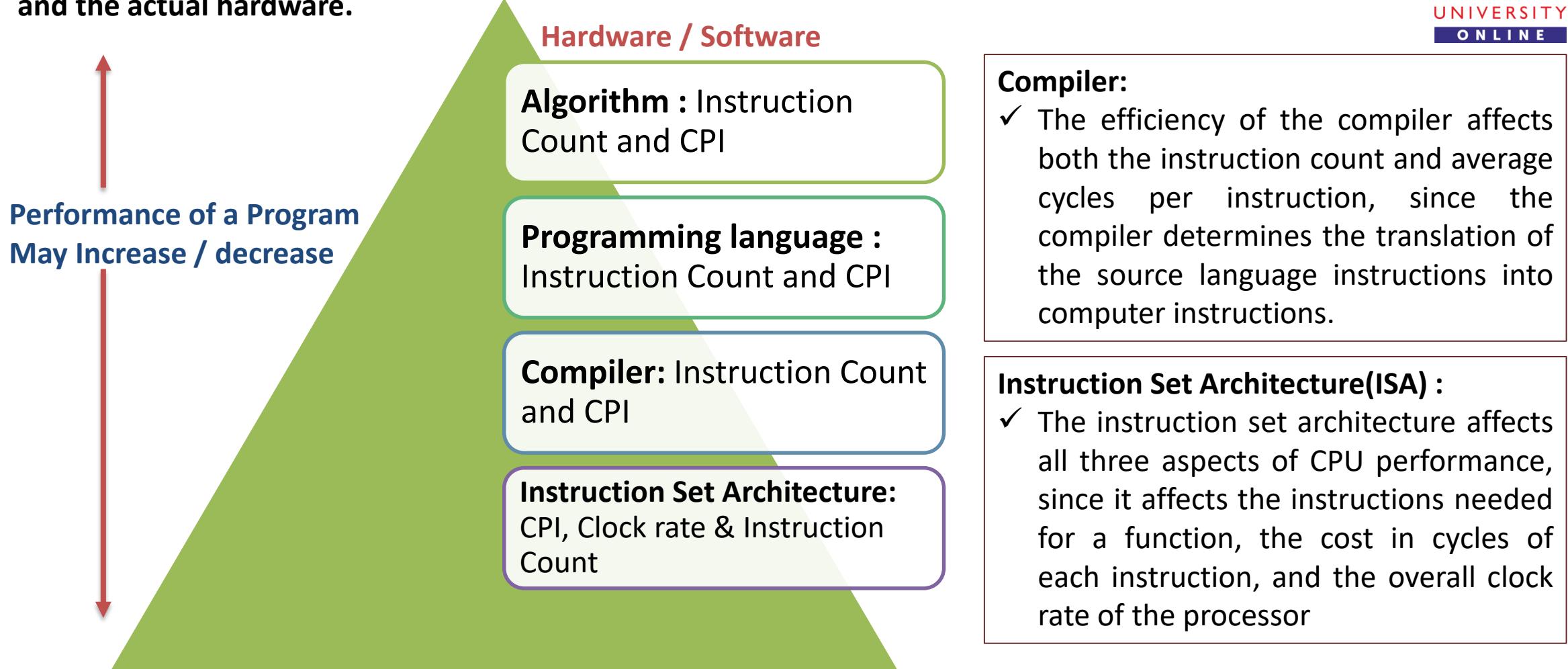
Parameters affecting Performance of a Program

The performance of a program depends on the **algorithm, the language, the compiler, the architecture, and the actual hardware**.



Parameters affecting Performance of a Program

The performance of a program depends on the **algorithm, the language, the compiler, the architecture, and the actual hardware**.



Turbo mode:

- Although clock cycle time has traditionally been fixed, to save energy or temporarily boost performance, **today's processors can vary their clock rates, so we would need to use the average clock rate for a program.**
- For example, the Intel Core i7 will temporarily increase clock rate by **about 10% until the chip gets too warm.**
- **Intel calls this Turbo mode.**

| Processor Number | Cores/Threads | Base Freq (GHz) | Intel Turbo Boost Technology 2.0 | | |
|---|---------------|-----------------|----------------------------------|---------------------------|---------------------------|
| | | | Max Single Core Turbo (GHz) | Max Dual Core Turbo (GHz) | Max Quad Core Turbo (GHz) |
| Intel® Core™ i7 Processors (S-Processor Line) | | | | | |
| i7-6700 | 4/8 | 3.4 | 4.0 | 3.9 | 3.7 |

The Clock rate and Power for Intel x86 microprocessors over nine generations and 36 years

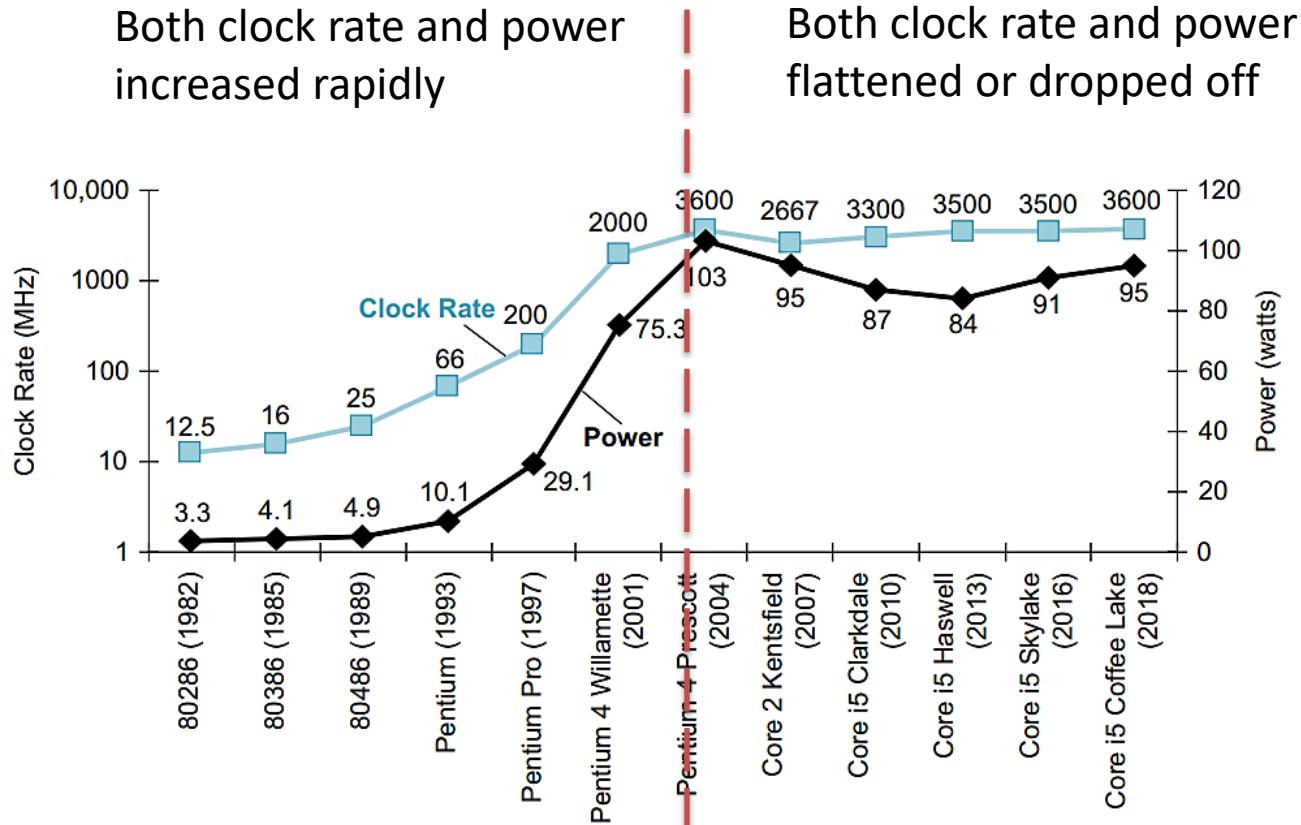


FIGURE 1.16 Clock rate and power for Intel x86 microprocessors over nine generations

- ✓ The reason they grew together is that **they are correlated**, and
- ✓ the reason for their recent slowing is that we have **run into the practical power limit** for cooling commodity microprocessors

- The dominant technology for **integrated circuits** is called **CMOS**
- For **CMOS**, the primary source of energy consumption is **dynamic energy** i.e., the **energy that is consumed when transistors switch states**
- The dynamic energy depends on the **capacitive loading** of each transistor and the **voltage applied**

$$\text{Energy} \propto \text{Capacitive load} \times \text{Voltage}^2$$

This equation is the energy of a pulse during the logic transition of $0 \rightarrow 1 \rightarrow 0$ or $1 \rightarrow 0 \rightarrow 1$. The energy of a single transition is then

$$\text{Energy} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2$$

The power required per transistor is just the product of energy of a transition and the frequency of transitions:

$$\text{Power} \propto 1/2 \times \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}$$

Example Some microprocessors today are designed to have adjustable voltage, so a 15% reduction in voltage may result in a 15% reduction in frequency. What would be the impact on dynamic energy and on dynamic power?

Answer Because the capacitance is unchanged, the answer for energy is the ratio of the voltages

$$\frac{\text{Energy}_{\text{new}}}{\text{Energy}_{\text{old}}} = \frac{(\text{Voltage} \times 0.85)^2}{\text{Voltage}^2} = 0.85^2 = 0.72$$

which reduces energy to about 72% of the original. For power, we add the ratio of the frequencies

$$\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = 0.72 \times \frac{(\text{Frequency switched} \times 0.85)}{\text{Frequency switched}} = 0.61$$

shrinking power to about 61% of the original.

Relative Power

- ✓ Suppose we developed a new, simpler processor that has 85% of the capacitive load of the more complex older processor.
- ✓ Further, assume that it can adjust voltage so that it can reduce voltage 15% compared to processor B, which results in a 15% shrink in frequency. What is the impact on dynamic power?

$$\frac{\text{Power}_{\text{new}}}{\text{Power}_{\text{old}}} = \frac{\text{Capacitive load} \times 0.85 \times \text{Voltage}^2 \times 0.85^2 \times \text{Frequency switched} \times 0.85}{\text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency switched}}$$

Thus the power ratio is

$$0.85^4 = 0.52$$

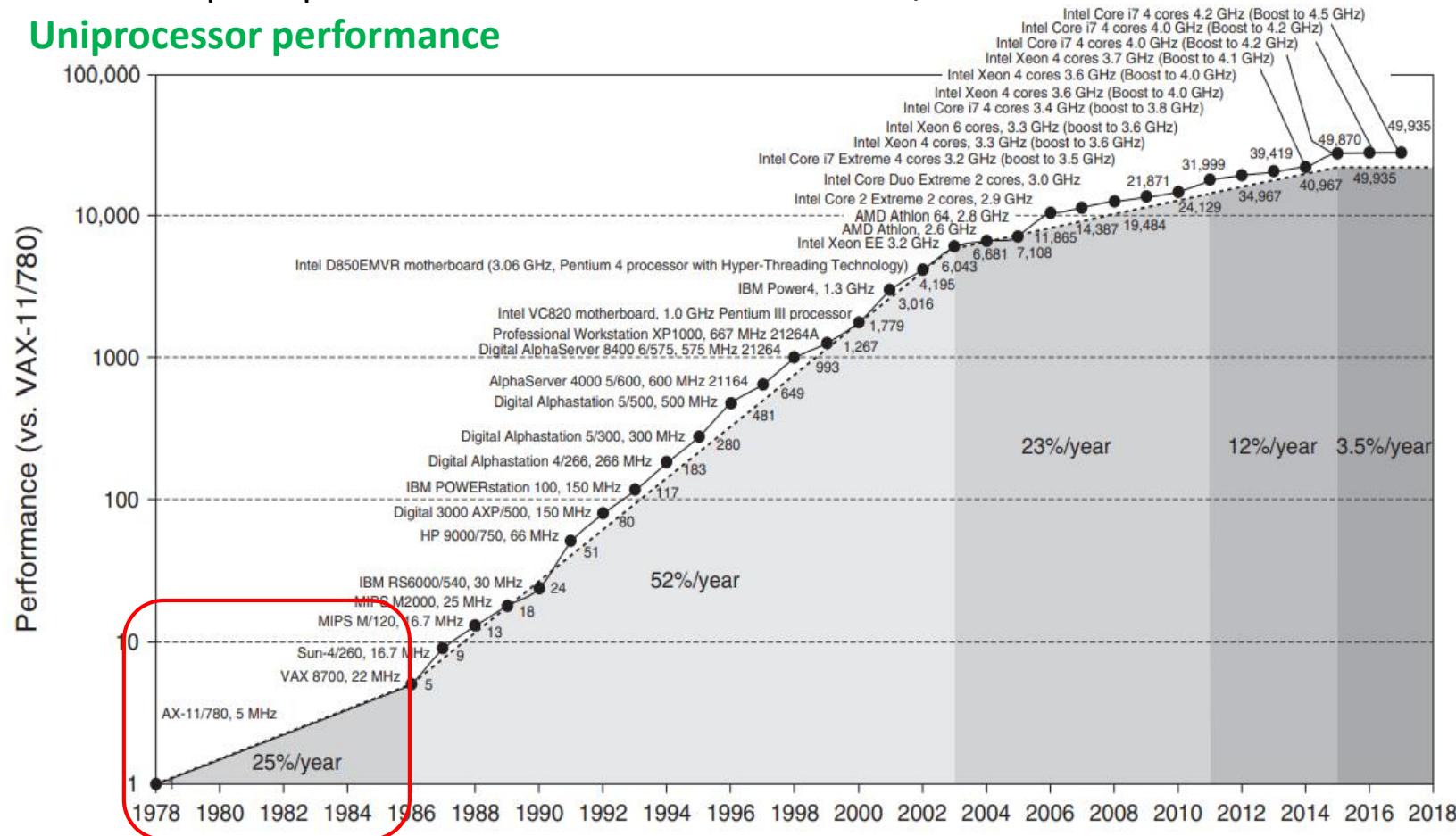
Hence, the new processor uses about half the power of the old processor.

The Sea Change: The Switch from Uniprocessors to Multiprocessors

Growth in Processor Performance since the mid-1980s

This chart plots performance relative to the VAX 11/780

Uniprocessor performance



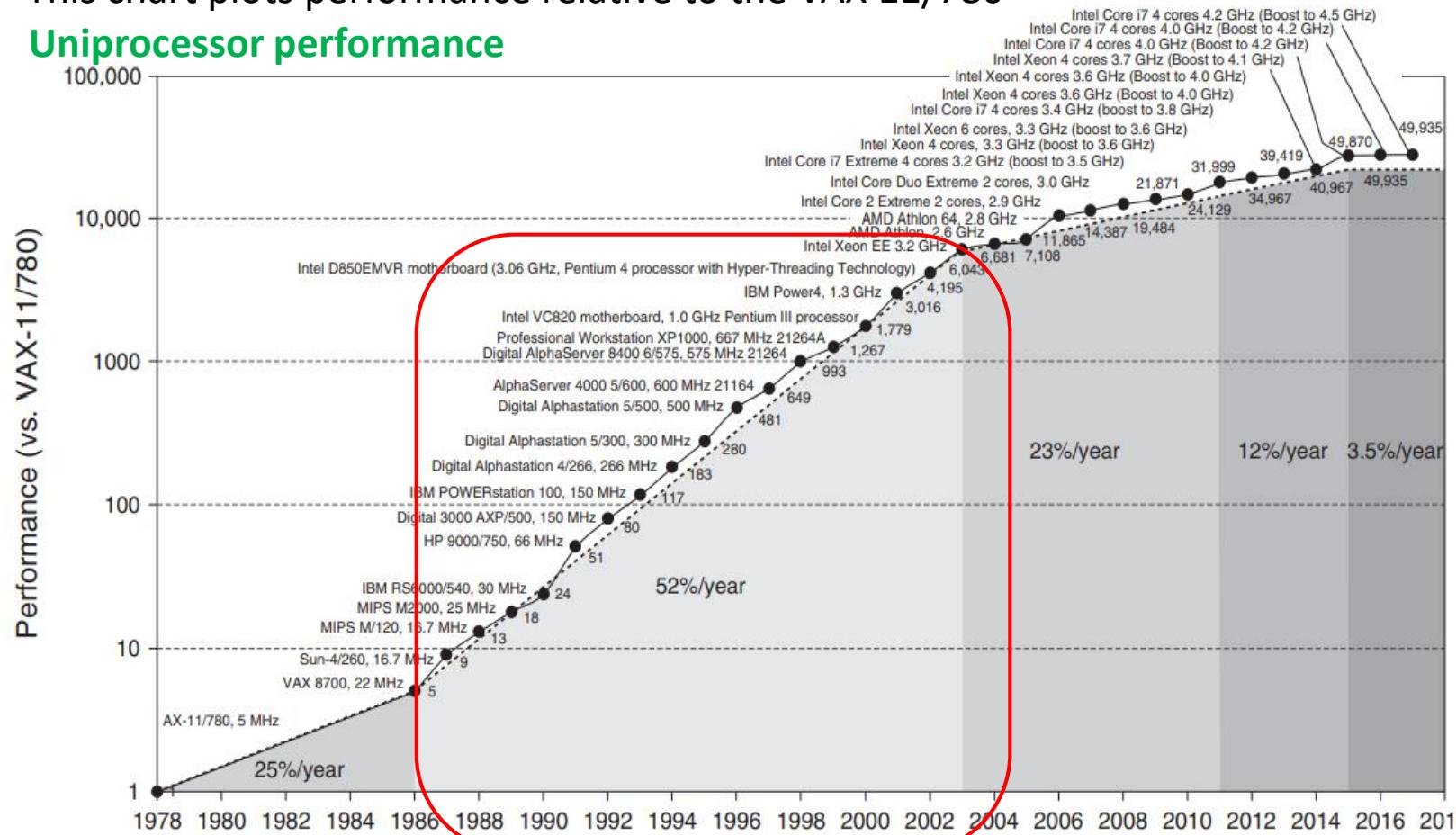
Prior to the mid-1980s, processor performance growth was largely technology driven and averaged about 25% per year.

The Sea Change: The Switch from Uniprocessors to Multiprocessors

Growth in Processor Performance since the mid-1980s

This chart plots performance relative to the VAX 11/780

Uniprocessor performance



The increase in growth to about 52% since then is attributable to more advanced architectural and organizational ideas

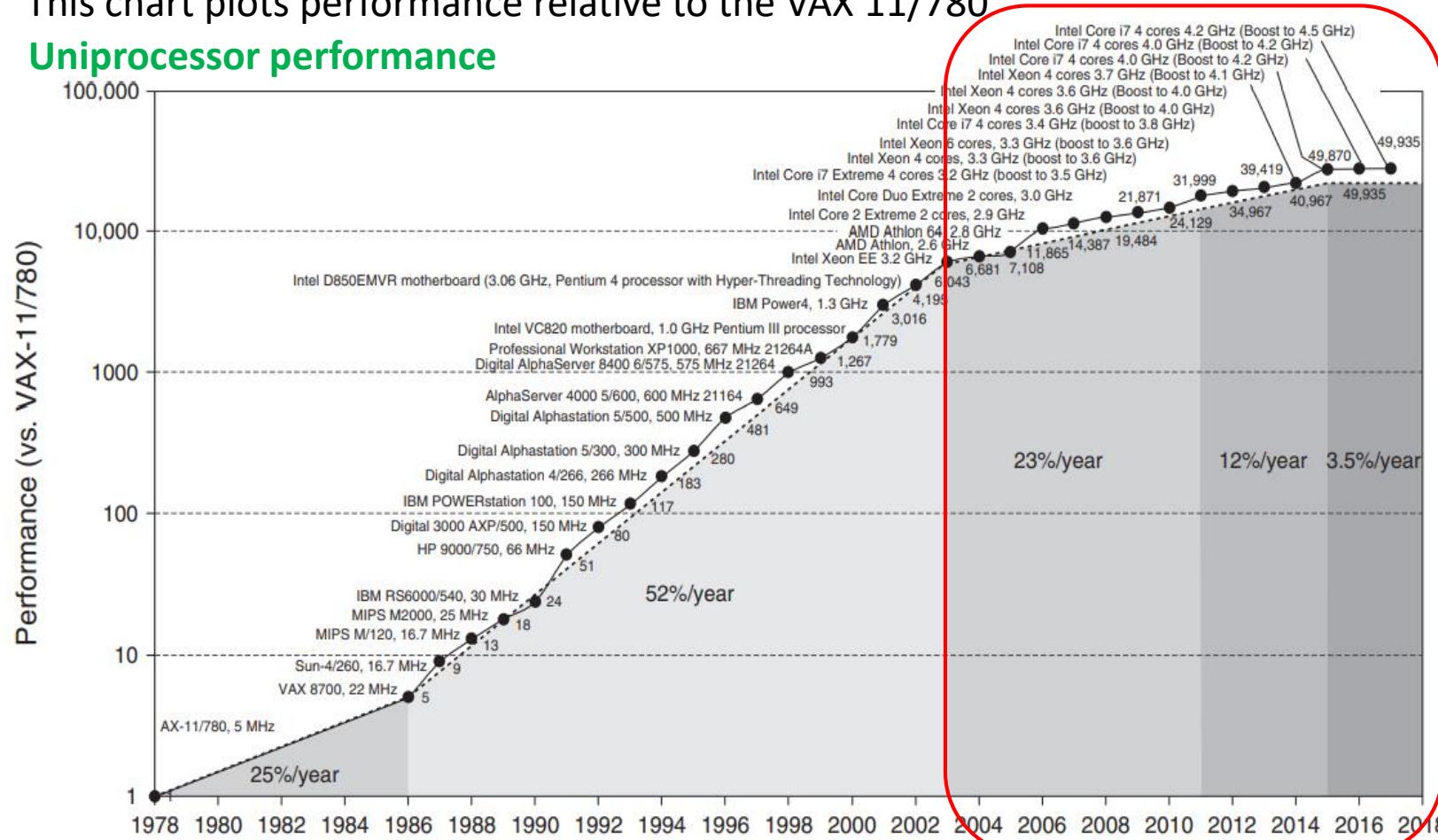
First of a range of popular and influential computers implementing the VAX ISA.

The Sea Change: The Switch from Uniprocessors to Multiprocessors

Growth in Processor Performance since the mid-1980s

This chart plots performance relative to the VAX 11/780

Uniprocessor performance



Since 2002, the limits of power, available instruction-level parallelism, and long memory latency have slowed uniprocessor performance recently, to about 3.5% per year.

The Sea Change: The Switch from Uniprocessors to Multiprocessors

What do we do??

From mid 1980s

Continuing to decrease the response time of one program running on the single processor

As of 2006

All desktop and server companies are shipping microprocessors with multiple processors per chip

Challenges for Programmers

- Rewrite their programs to take advantage of multiple processors.
- Programmers will have to continue to improve the performance of their code as the number of cores increases

Benefit is often more on **throughput** than on response time

Companies refer to processors as “cores,” and such microprocessors are generically called multicore microprocessors.

Hence, a “quadcore” microprocessor is a chip that contains four processors or four cores.

Challenges for Programmers

Why has it been so hard for programmers to write explicitly parallel programs?

- Parallel programming is by **definition performance programming**, which increases the difficulty of programming.
- Fast for parallel hardware means that the programmer must **divide an application** so that each processor has roughly the same amount to do at the same time, and that the overhead of scheduling and coordination doesn't fritter away the potential performance benefits of parallelism.

The Sea Change: The Switch from Uniprocessors to Multiprocessors

To reflect this sea change in the industry -

- Parallelism and Instructions: Synchronization.
- Parallelism and Computer Arithmetic: Subword Parallelism.
- Parallelism via Instructions.
- Parallelism and Memory Hierarchies: Cache Coherence.
- Parallelism and Memory Hierarchy: Redundant Arrays of Inexpensive Disks.

Computer Abstractions and Technology

Benchmarking the Intel Core i7

System Performance Evaluation Cooperative (SPEC) CPU Benchmark

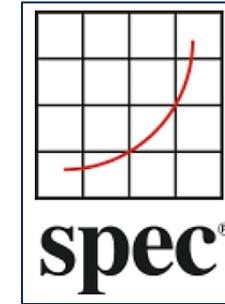
Workload: A set of programs run on a computer

To evaluate two computer systems, a user would simply compare the execution time of the workload on the two computers.

Benchmarks: Programs specifically chosen to measure performance

The benchmarks form a workload that the user hopes will predict the performance of the actual workload.

SPEC: (System Performance Evaluation Cooperative) is an effort funded and supported by a number of computer vendors to create standard sets of benchmarks for modern computer systems.



SPEC CPU Benchmark

- In 1989, SPEC originally created a benchmark set focusing on processor performance (now called SPEC89).
- The latest is SPEC CPU2017, which consists of a set of **10 integer benchmarks** (SPECspeed 2017 Integer) and **13 floating-point benchmarks** (CFP2006).
- The **integer benchmarks** vary from part of a C compiler to a chess program to a quantum computer simulation.
- The **floating-point benchmarks** include structured grid codes for finite element modeling, particle method codes for molecular dynamics, and sparse linear algebra codes for fluid dynamics.

Computer Abstractions and Technology

Benchmarking the Intel Core i7

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

| 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 |

Table describes the SPEC integer benchmarks and their execution time on the Intel Core i7 and shows the factors that explain execution time:

- Instruction count
- CPI
- Clock cycle time

Computer Abstractions and Technology

Benchmarking the Intel Core i7

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

| 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 |

Execution Time:

Execution Time = Instruction Count X CPI X Clock cycle time

Ex:

Perlbench Execution Time : 2684 X 10⁹X0.42X 0.556 X 10⁻⁹ : 627

Computer Abstractions and Technology

Benchmarking the Intel Core i7

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

| 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 |

Reference Time:

Reference Time = Supplied by SPEC

Ex:

Perlbench Reference Time :
1774

Computer Abstractions and Technology

Benchmarking the Intel Core i7

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

| 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 |

SPECratio:

$$\text{SPECratio} = \frac{\text{Reference Time}}{\text{Execution Time}}$$

Ex:

Perlbench SPECratio : 1774/627
: 2.83

Computer Abstractions and Technology

Benchmarking the Intel Core i7

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

| 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 |

Geometric Mean:

The single number quoted as **SPECspeed 2017 Integer** is the geometric mean of the SPECratios.

SPEC Power Benchmark

- Given the increasing importance of energy and power, SPEC added a benchmark to measure power.
- SPECpower_ssj2008 is the first industry-standard benchmark for measuring both the performance and the power consumption of servers.
- SSJ - Server Side Java

http://www.spec.org/power/docs/SPECpower_ssj2008-Design_ssj.pdf

Computer Abstractions and Technology

Benchmarking the Intel Core i7



SPEC Power Benchmark

| Target Load % | Performance (ssj_ops) | Average Power (watts) |
|--|-----------------------|-----------------------|
| 100% | 4,864,136 | 347 |
| 90% | 4,389,196 | 312 |
| 80% | 3,905,724 | 278 |
| 70% | 3,418,737 | 241 |
| 60% | 2,925,811 | 212 |
| 50% | 2,439,017 | 183 |
| 40% | 1,951,394 | 160 |
| 30% | 1,461,411 | 141 |
| 20% | 974,045 | 128 |
| 10% | 485,973 | 115 |
| 0% | 0 | 48 |
| Overall Sum | 26,815,444 | 2,165 |
| $\sum \text{ssj_ops} / \sum \text{power} =$ | | 12,385 |

FIGURE 1.19 SPECpower_ssj2008 running on a dual socket 2.2 GHz Intel Xeon Platinum 8276L with 192 GiB of DRAM and one 80 GB SSD disk.

SPECpower reports power consumption of servers at different workload levels, divided into 10% increments, over a period of time.

Computer Abstractions and Technology

Benchmarking the Intel Core i7

SPEC Power Benchmark

| Target Load % | Performance (ssj_ops) | Average Power (watts) |
|--|-----------------------|-----------------------|
| 100% | 4,864,136 | 347 |
| 90% | 4,389,196 | 312 |
| 80% | 3,905,724 | 278 |
| 70% | 3,418,737 | 241 |
| 60% | 2,925,811 | 212 |
| 50% | 2,439,017 | 183 |
| 40% | 1,951,394 | 160 |
| 30% | 1,461,411 | 141 |
| 20% | 974,045 | 128 |
| 10% | 485,973 | 115 |
| 0% | 0 | 48 |
| Overall Sum | 26,815,444 | 2,165 |
| $\sum \text{ssj_ops} / \sum \text{power} =$ | | 12,385 |

SPEC boils these numbers down to one number, called “**overall ssj_ops per watt**.”

The formula for this single summarizing metric is

$$\text{overall ssj_ops per watt} = \left(\sum_{i=0}^{10} \text{ssj_ops}_i \right) / \left(\sum_{i=0}^{10} \text{power}_i \right)$$

ssj_ops_i is performance at each 10% increment

power_i is power consumed at each performance level

FIGURE 1.19 SPECpower_ssj2008 running on a dual socket 2.2 GHz Intel Xeon Platinum 8276L with 192 GiB of DRAM and one 80 GB SSD disk.

Python Program to Multiply Two Matrices

Programiz
Python Online Compiler

Google מודעת למשתמש מהו המודעה זו מוצגת? ⓘ אם לא רצית לראות את המודעה זו

main.py

```
1 # Program to multiply two matrices using nested loops
2
3 # 3x3 matrix
4 X = [[12,7,3],
5     [4 ,5,6],
6     [7 ,8,9]]
7 # 3x4 matrix
8 Y = [[5,8,1,2],
9     [6,7,3,0],
10    [4,5,9,1]]
11 # result is 3x4
12 result = [[0,0,0,0],
13             [0,0,0,0],
14             [0,0,0,0]]
15
16 # iterate through rows of X
17 for i in range(len(X)):
18     # iterate through columns of Y
19     for j in range(len(Y[0])):
20         # iterate through rows of Y
21         for k in range(len(Y)):
22             result[i][j] += X[i][k] * Y[k][j]
23
24 for r in result:
25     print(r)
26
```

Run

Shell

```
[114, 160, 60, 27]
[74, 97, 73, 14]
[119, 157, 112, 23]
> |
```

A simple python program is written to multiply two matrices

- I used a Python online Compiler
- I ran it on my personal laptop

Computer Abstractions and Technology

Going Faster: Matrix Multiply in Python

```
for i in xrange(n):
    for j in xrange(n):
        for k in xrange(n):
            C[i][j] += A[i][k] * B[k][j]
```



| | | |
|----------------|--|-------------|
| Machine used | n1-standard-96 server in Google Cloud Engine, which has two Intel Skylake Xeon chips, and each chip has 24 processors or cores and running Python version 3.1. | |
| Size of Matrix | 960 x 960 | 4096 x 4096 |
| Time taken | 5 minutes | 6 hours |

- ✓ If we used the Numpy library(python) instead, a 960 x 960 matrix multiply would take much less than 1 second instead of 5 minutes.



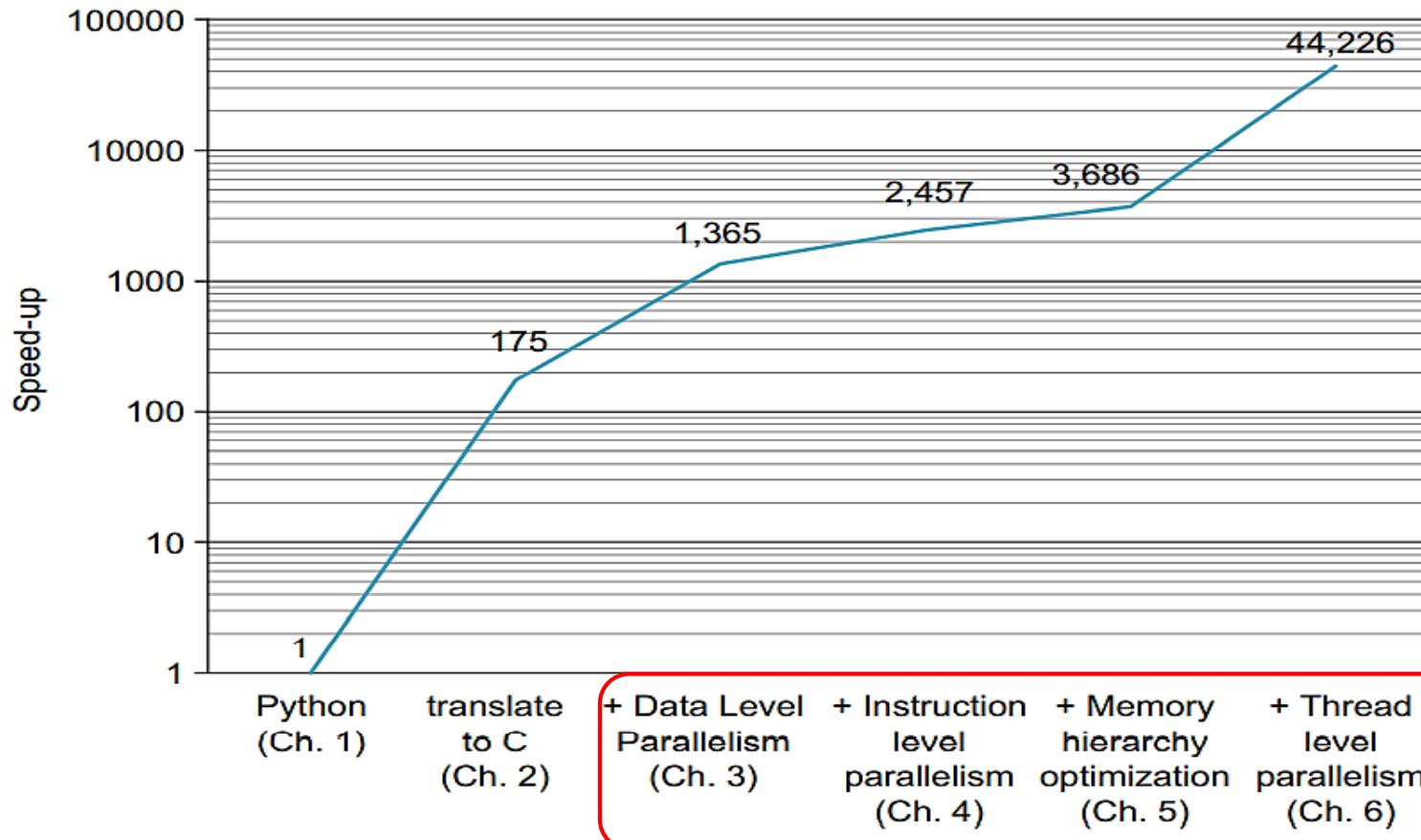
While it is quick to write the matrix multiply in Python, who wants to wait that long to get the answer?



Python Program to Multiply Two Matrices - Optimizations

| Text Book Chapter | Techniques used | Increase in performance by a factor of |
|-------------------|--|--|
| Chapter 2 | Convert the Python version of matrix multiply to a C version | 200 |
| Chapter 3 | Use subword parallelism via C intrinsics | 8 |
| Chapter 4 | Use loop unrolling to exploit multiple instruction issue and out-of-order execution hardware | 2 |
| Chapter 5 | Use cache blocking to increase performance on large matrices | 1.5 |
| Chapter 6 | Use parallel for loops in OpenMP to exploit multicore hardware | 12 to 17 |

Python Program to Multiply Two Matrices - Optimizations



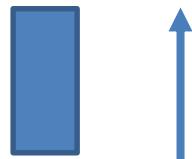
The last four steps leverage our understanding how the underlying hardware really works in a modern microprocessor and collectively only requires 21 lines of C code.

FIGURE 1.20 Optimizations of matrix multiply program in Python in the next five chapters of this book.



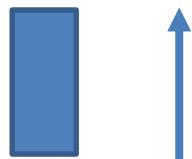
Pitfall

Expecting the improvement of one aspect of a computer to increase overall performance by an amount proportional to the size of the improvement.



Improvement in x by a factor f

=



Improvement in overall performance by a factor f

Suppose a program runs in 100 seconds on a computer, with multiply operations responsible for 80 seconds of this time.

How much do I have to improve the speed of multiplication if I want my program to run five times faster?



Fallacies and Pitfalls

Suppose a program runs in 100 seconds on a computer, with multiply operations responsible for 80 seconds of this time.

How much do I have to improve the speed of multiplication if I want my program to run five times faster?

Multiplication – 80 seconds

$$\frac{\text{Execution time after improvement}}{\text{Execution time affected by improvement}} = \frac{\text{Execution time unaffected}}{\text{Amount of improvement}}$$

For this problem:

$$\text{Execution time after improvement} = \frac{80 \text{ seconds}}{n} + (100 - 80 \text{ seconds})$$

Since we want the performance to be five times faster, the new execution time should be 20 seconds, giving

$$20 \text{ seconds} = \frac{80 \text{ seconds}}{n} + 20 \text{ seconds}$$
$$0 = \frac{80 \text{ seconds}}{n}$$

That is, there is no amount by which we can enhance-multiply to achieve a fivefold increase in performance, if multiply accounts for only 80% of the workload.



Computers at low utilization use little power.

- Power efficiency matters at low utilizations because server workloads vary.
- Utilization of servers in Google's warehouse scale computer, for example, is between 10% and 50% most of the time and at 100% less than 1% of the time.
- Specially configured computer with the best results in 2020 still uses 33% of the peak power at 10% of the load.
- Systems in the field that are not configured for the SPECpower benchmark are surely worse.



Designing for performance and designing for energy efficiency are unrelated goals.

Since energy is power over time, it is often the case that hardware or software optimizations that take **less time save energy** overall even if the optimization takes a bit more energy when it is used.



Using a subset of the performance equation as a performance metric

- We have already warned about the danger of predicting performance based on simply one of the clock rate, instruction count, or CPI.
- Another common mistake is to use only two of the three factors to compare performance.

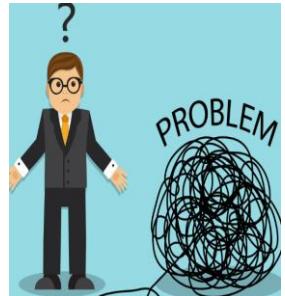
One alternative to time is MIPS (million instructions per second).

For a given program, MIPS is simply

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

Since MIPS is an instruction execution rate, MIPS specifies performance inversely to execution time; **faster computers have a higher MIPS rating.**

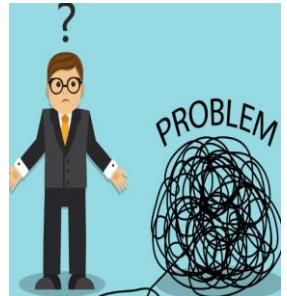
There are three problems with using MIPS as a measure for comparing computers.



MIPS specifies the instruction execution rate but does not take into account the capabilities of the instructions.

We cannot compare computers with different instruction sets using MIPS, since the instruction counts will certainly differ.

There are three problems with using MIPS as a measure for comparing computers.



MIPS varies between programs on the same computer;
thus, a computer cannot have a single MIPS rating.

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

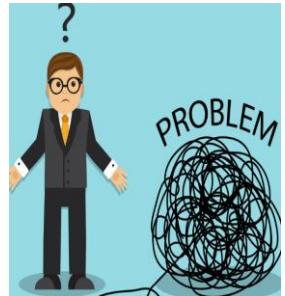
$$\text{MIPS} = \frac{\frac{\text{Instruction count}}{\text{Instruction count} \times \text{CPI}} \times 10^6}{\text{Clock rate}} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

Change in CPI changes the MIPS rating

Computer Abstractions and Technology

Fallacies and Pitfalls

There are three problems with using MIPS as a measure for comparing computers.



If a new program executes more instructions but each instruction is faster, MIPS can vary independently from performance!

Consider the following performance measurements for a program:

| Measurement | Computer A | Computer B |
|-------------------|------------|------------|
| Instruction count | 10 billion | 8 billion |
| Clock rate | 4 GHz | 4 GHz |
| CPI | 1.0 | 1.1 |

- a. Which computer has the higher MIPS rating?
- b. Which computer is faster?





THANK YOU

Mahesh Awati & Vinay Reddy

Department of Electronics and Communication

mahesha@pes.edu

+91 9741172822