

Microprocessor II

- Chapter 1: Introduction
- Chapter 2: MIPS architecture
- Chapter 3: MIPS micro-architecture
- Chapter 4: Memory Systems; virtual memory, cache memory,...
- Chapter 5: I/O interfaces, interrupts management, interfaces management

Microprocessor II

References:

- Digital Design and Computer Architecture, 2nd Edition, Sarah L. Harris David Money Harris, 2013, Elsevier
- Computer Organization And Design The Hardware / Software Interface, fifth edition, Elsevier, David A. Patterson and John L. Hennessy, 2014, Elsevier
- Computer Architecture; A quantitative Approach, sixth edition, John L. Hennessy and David A. Patterson, 2019, Elsevier
- MARS MIPS Simulator:

<http://courses.missouristate.edu/KenVollmar/MARS/download.htm>



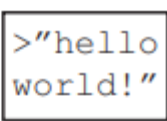


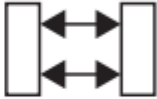
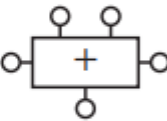

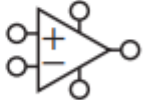


Chapter 1

INTRODUCTION

Introduction

- Architecture
- Micro-architecture

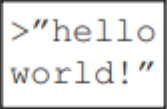


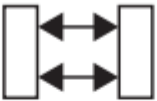
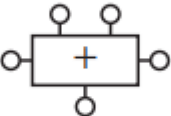

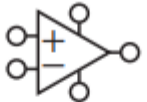


Levels of abstraction for an electronic computing system

Application Software	
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	

Introduction

- Architecture
- Micro-architecture

Levels of abstraction for an electronic computing system

Application Software		Programs
Operating Systems		Device Drivers
Architecture		Instructions Registers
Micro-architecture		Datapaths Controllers
Logic		Adders Memories
Digital Circuits		AND Gates NOT Gates
Analog Circuits		Amplifiers Filters
Devices		Transistors Diodes
Physics		Electrons

Definitions

Architecture: Programmer's view of computer

- Mainly Defined by instructions (instructions set) & operand locations (registers and memory)
- Examples of architectures: RISC-V, ARM, x86, MIPS, SPARC, and PowerPC

Definitions

Microarchitecture: How to implement an architecture in hardware

- Microarchitecture involves combining logic elements to execute the instructions defined by the architecture
- It defines the specific arrangement of registers, memories, ALUs, and other building blocks to form the microprocessor

Definitions

Microarchitecture:

- Multiple implementation exists for a single architecture : single cycle, multiple cycle, pipelined
 - Single cycle
 - Multiple cycle
 - Pipelined

Definitions

Microarchitecture:

- Single cycle
 - Each instruction executes in a one cycle
- Multiple cycle
 - Each instruction is broken into series of shorter steps
- Pipelined
 - Each instruction is broken into series of shorter steps, and multiple instructions execute at once

Definitions

Microarchitecture:

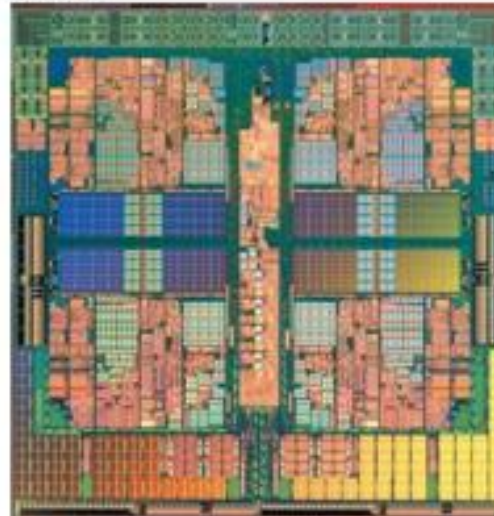
- Often, many different hardware implementations of a single architecture exist. For example, Intel and Advanced Micro Devices (AMD) both sell various microprocessors belonging to the same x86 architecture.
- Using different underlying hardware offers **trade-offs in performance, price, and power**. Some microprocessors are optimized for high-performance servers, whereas others are optimized for long battery life in laptop computers.

Example

*Same architecture but
different microarchitecture*

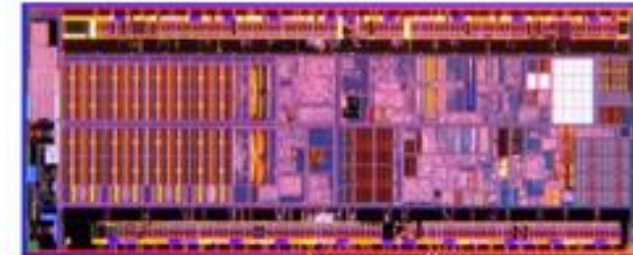
AMD Phenom X4

- X86 Instruction Set
- Quad Core
- 125W
- Decode 3 Instructions/Cycle/Core
- 64KB L1 I Cache, 64KB L1 D Cache
- 512KB L2 Cache
- 2.6GHz



Intel Atom

- X86 Instruction Set
- Single Core
- 2W
- Decode 2 Instructions/Cycle/Core
- 32KB L1 I Cache, 24KB L1 D Cache
- 512KB L2 Cache
- 1.6GHz



Classifying of Instruction Set Architecture

Operand storage in CPU	Where are they other than memory
# explicit operands named per instruction	How many? Min, Max, Average
Addressing mode	How the effective address for an operand calculated? Can all use any mode?
Operations	What are the options for the opcode?
Type & size of operands	How is typing done? How is the size specified?

These choices critically affect the number of instructions, CPI time

Classifying of Instruction Set Architecture

Most basic differentiation: internal storage in a processor

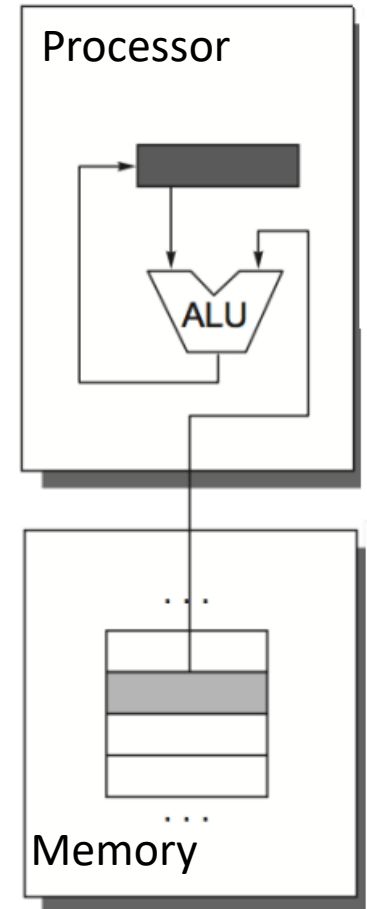
- Operands can be named implicitly or explicitly

Major choices:

- Accumulator architecture
- Stack architecture
- General purpose register architectures

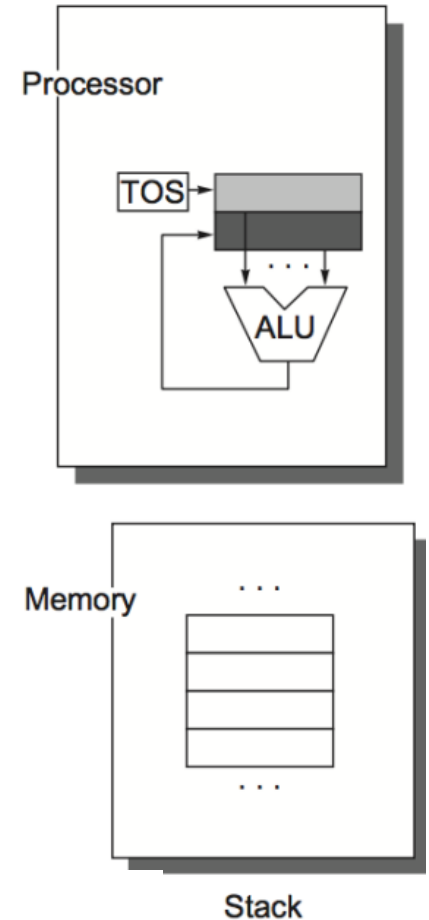
Classifying of Instruction Set Architecture

- **Accumulator architecture:** one operand is implicitly the accumulator



Classifying of Instruction Set Architecture

- **Stack architecture** : the operands are implicitly on the top of the stack

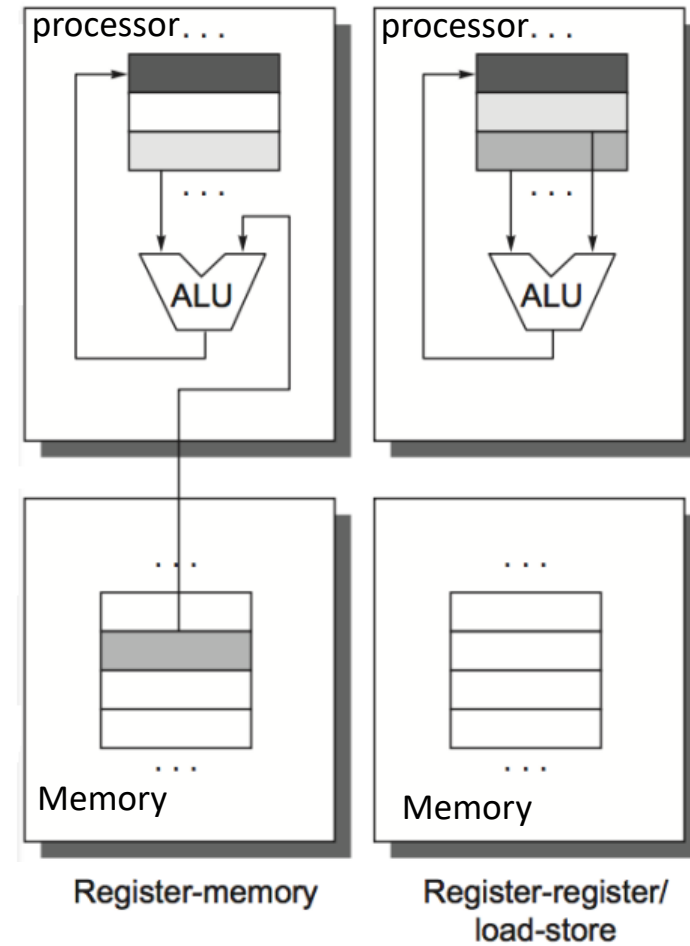


Classifying of Instruction Set Architecture

- **General purpose register architectures:** GPR architectures have only explicit operands – either registers or memory location

ISA classes

- GPR architectures:
 - *Register-Register*
 - *Register-Memory*
- Memory-Memory architecture



ISA classes

- The code sequence for $C=A+B$ using the 4 ISA classes:

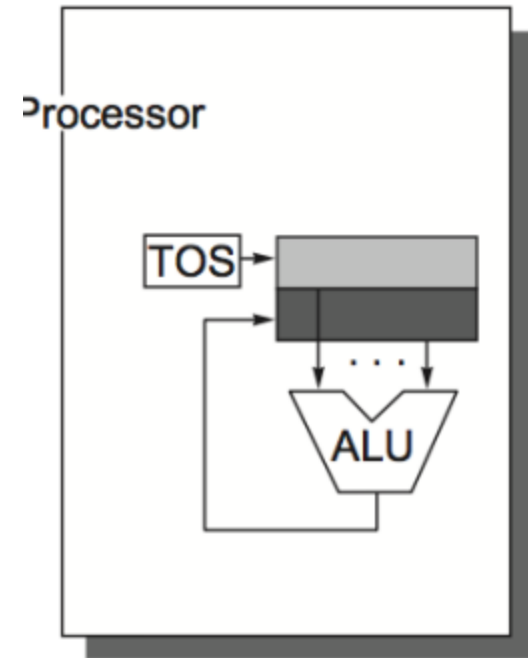
- *STACK architecture:*

Push A

Push B

ADD

POP C



ISA classes

- The code sequence for $C=A+B$ using the 4 ISA classes:

- *STACK architecture:*

Push A

Push B

ADD

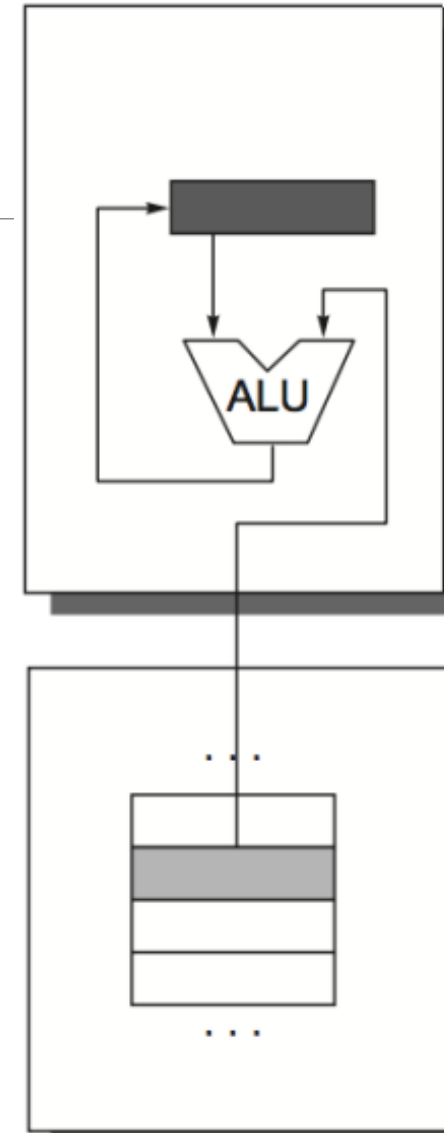
POP C

- *Accumulator architecture*

Load A

Add B

Store C



ISA classes

- The code sequence for $C=A+B$ using the 4 ISA classes:

- *STACK architecture:*

Push A

Push B

ADD

POP C

- *Accumulator architecture*

Load A

Add B

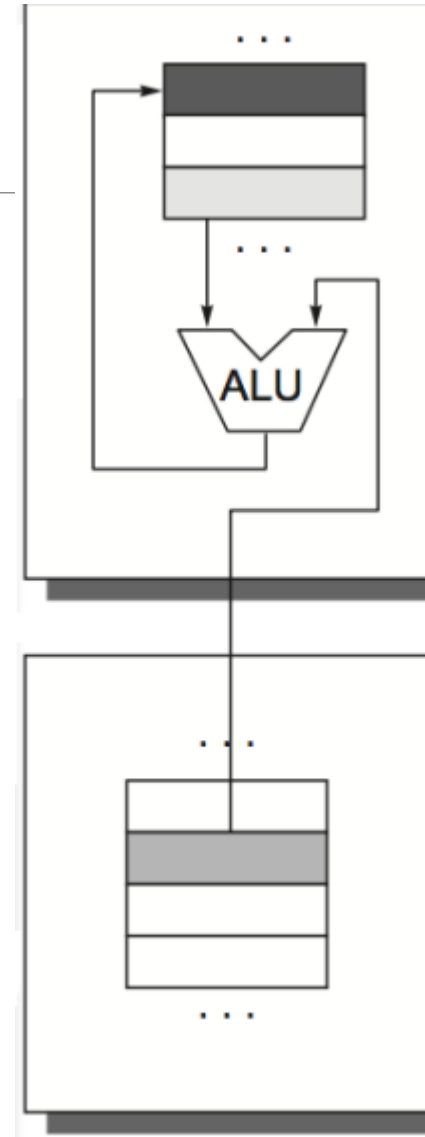
Store C

- **Register (Register–Memory)**

Load R1, A

ADD R3, R1, B

Store R3, C



■ The code sequence for $C=A+B$ using the 4 ISA classes:

– *STACK architecture:*

Push A

Push B

ADD

POP C

– *Accumulator architecture*

Load A

Add B

Store C

– *Register (Register—Memory)*

Load R1, A

ADD R3, R1, B

Store R3, C

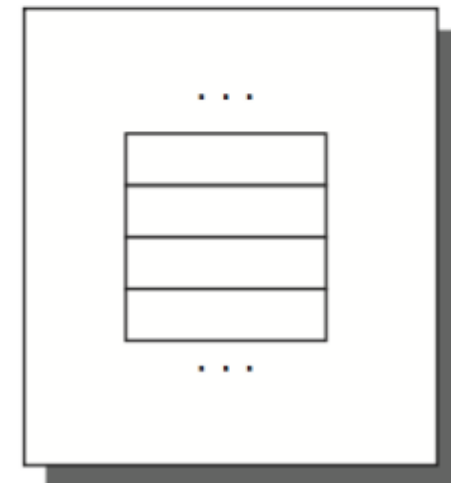
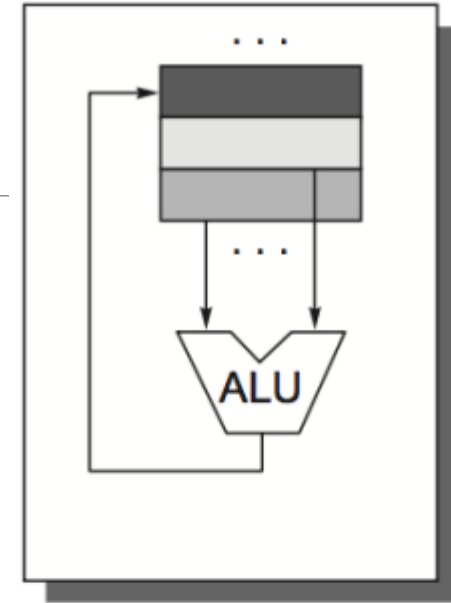
– ***Register-Register (load- store)***

Load R1, A

Load R2, B

ADD R3, R1, R2

Store R3, C



ISA classes

How many registers are sufficient?

Two major concerns for arithmetic and logical instructions (ALU)

1. Two or three operands ?

➤ $X + Y \Rightarrow X$

➤ $X + Y \Rightarrow Z$

2. How many of the operands may be memory addresses ?

➤ Number of memory addressed ?

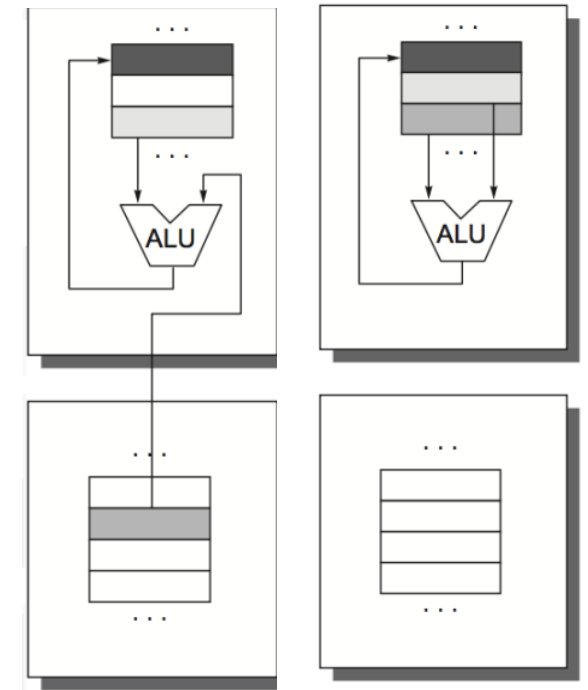
➤ Maximum number of operands allowed?

ISA classes

➤ How many of the operands may be memory addresses ?

(Number of memory addressed , Maximum number of operands allowed)

- ✓ Load-store architecture (?,?)
- ✓ Register-Memory architecture (?,?)
- ✓ Memory –Memory architecture (?,?)

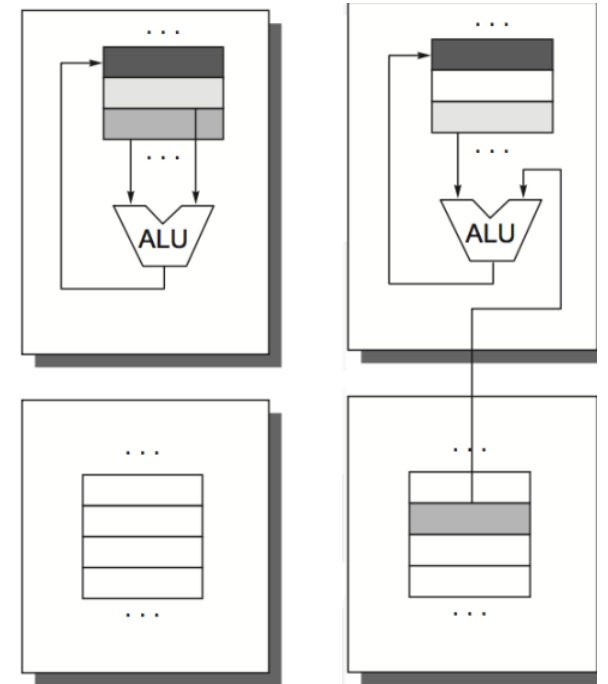


ISA classes

➤ How many of the operands may be memory addresses ?

(Number of memory addressed , Maximum number of operands allowed)

- ✓ Load-store architecture has (0-3)
- ✓ Register-Memory architecture has (1-2)
- ✓ Memory –Memory architecture has (2-2) or (3-3)



ISA classes

- **Accumulator** (before 1960, ex. 68HC11):
 - Requires 1 address of memory Add A $\text{acc} = \text{acc} + \text{mem}[A]$
- **Stack** (1960s – 1970s):
 - Requires 0 address Add $\text{tos} = \text{tos} + \text{next}$ (tos = top of stack)
- **Memory – memory** (1970s – 1980s):
 - Requires 2 addresses Add A, B $\text{mem}[A] = \text{mem}[A] + \text{mem}[B]$
 - Requires 3 addresses Add A, B, C $\text{mem}[A] = \text{mem}[B] + \text{mem}[C]$

ISA classes

- **Register – memory** (1970s – present, ex. 80x86):
 - Requires 2 addresses (1 memory and 1 register) i.e., 2 operands
 - Add R1, A $\#R1 = R1 + \text{mem}[A]$
 - Load R1, A $\#R1 = \text{mem}[A]$
 - Examples of processors (CISC architecture): Intel x86, Pentium
- **Register – register** (Load/store) (1960s – present, ex. MIPS):
 - Requires 3 addresses of registers
 - Add R1, R2, R3 $\#R1 = R2 + R3$
 - Load R1, R2 $\#R1 = \text{mem}[R2]$
 - Store R1, R2 $\# \text{mem}[R2] = R1$
 - Examples of processors (RISC architecture): MIPS, PowerPC

ISA classes

➤ Challenge is to satisfy constraints of:

- Cost
- Power
- Performance

■ Performance ?

Defining performance

- As an individual computer user, if you were running a program on two different desktop computers, you are interested in reducing ***response time / execution time*** (time between the start and completion of a task) therefore in your opinion, the faster one is the desktop computer that gets the job done first.

Defining performance

- If you were running a datacenter that had several servers running jobs submitted by many users, you'd say that the faster computer was the one that completed the most jobs during a day. Datacenter managers are often interested in increasing **throughput** or **bandwidth** (the total amount of work done in a given time).

Response Time and Throughput

Response time

- How long it takes to do a task

Throughput

- Total work done per unit time
 - e.g., tasks/transactions/ ... per hour

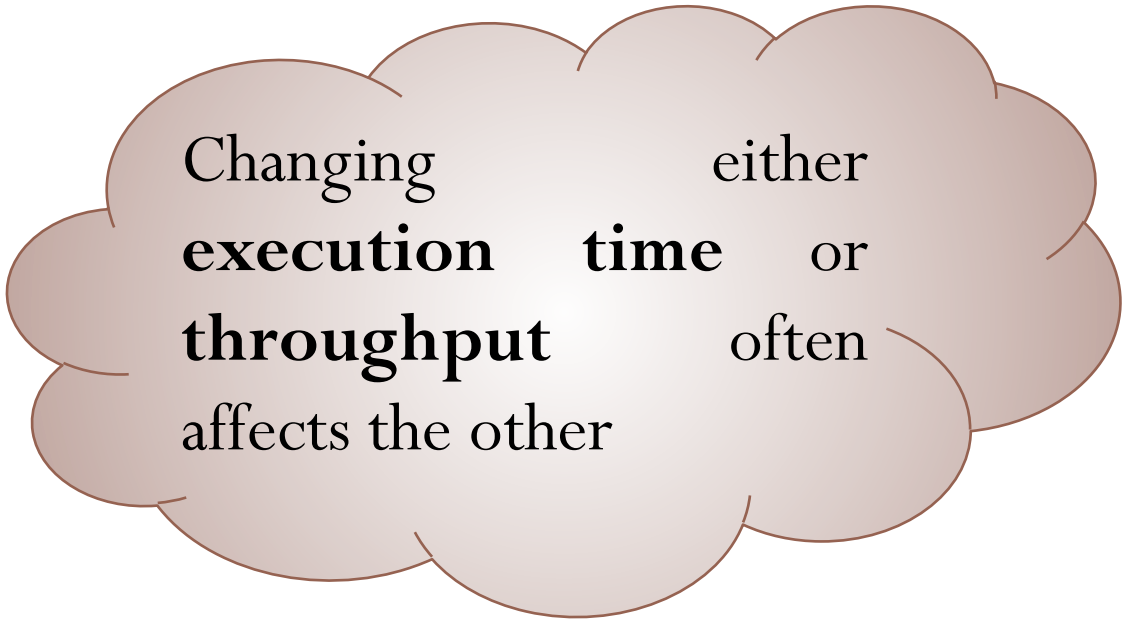
Response Time and Throughput

Response time

- How long it takes to do a task

Throughput

- Total work done per unit time
 - e.g., tasks/transactions/... per hour



Changing **execution time** either
throughput or
affects the other often

Response Time and Performance

- Decreasing response time → increasing performance
- To maximize performance, we want to minimize response time or execution time for some task
- Performance of computer x is defined as:

$$\text{Performance}_x = \frac{1}{\text{execution time}_x}$$

Exp : if “X is n time faster than Y” or equivalently “X is n times as fast as Y:

$$\frac{\text{Performance}_x}{\text{Performance}_y} = \frac{\text{Execution time}_y}{\text{Execution time}_x} = n$$

Measuring Execution Time

- **Program execution time is measured in seconds per program.** However, it can be defined in different ways, depending on what we count.
- The most straightforward definition of time is called ***wall clock time, response time, or elapsed time that*** is the total response time to complete a task, including all aspects (Processing, I/O, OS overhead, memory accesses, ...)

Measuring Execution Time

- However, 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.**
- Hence, we often want to distinguish between the elapsed time and the time over which the processor is working on our behalf. ***CPU execution time*** or simply ***CPU time***, is the time the CPU spends computing for this task and does not include time spent waiting for I/O or running other programs.

Measuring CPU execution time

- **CPU Performance: based on CPU execution time/CPU time**
- **Definitions:**
 - Clock period, clock rate
 - Clock Cycle/Clock cycle time
 - CPU clock cycles = number of clock cycles for a program

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

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

Measuring CPU execution time

Example:

Computer A: 2GHz clock, 10s CPU time

Designing Computer B:

Aim for 6s CPU time

The clock for B can be faster, but it causes an increase in the number of clock cycles ($1.2 \times \text{\#clock cycles}$)

How fast must Computer B clock be?

Measuring CPU execution time

■ Instruction Count and CPI

- **CPI:** cycle per instruction: how many cycles (clock cycles) are needed to execute one instruction
- **Instruction count:** is the number of instructions for a program

CPU Clock Cycles = ?

Measuring CPU execution time

■ Instruction Count and CPI

- **CPI:** cycle per instruction: how many cycles (clock cycles) are needed to execute one instruction
- **Instruction count:** is the number of instructions for a program, Determined by the program, ISA and compiler

$$CPU\ Clock\ Cycles = Instruction\ count \times CPI$$

Measuring CPU execution time

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

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

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

Measuring Performance

- Case of different CPIs: If different instructions have different CPI, we need to calculate average cycles per instruction.

$$\text{CPU clock cycles} = \sum_{i=1}^n (\text{CPI}_i \times C_i)$$

- average CPI:

$$\text{CPI}_{av} = \frac{\text{CPU clock cycles}}{\text{Instruction count}}$$

$$\text{CPU time} = \text{Instruction count} \times \text{CPI}_{av} \times \text{clock cycle time}$$

Measuring CPU execution time

Example: 1

- Computer A: Clock Cycle Time = 250ps, CPI = 2.0
- Computer B: Clock Cycle Time = 500ps, CPI = 1.2
- Same ISA

Which is faster, and by how much?

Example 2: Comparing Code Segments

A compiler designer is trying to decide between **two code sequences** for a particular 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?

Example 2: Comparing Code Segments

Which code sequence executes the most instructions?

Which will be faster?

What is the CPI for each sequence?

Performance Summary

CPU time = number of instructions \times $CPI_{average}$ \times clock cycle time

$$Performance = \frac{1}{CPU\ TIME}$$

RISC and CISC Architectures

Two approaches try to increase the CPU performance:

- **RISC «Reduced Instruction Set Computer»:** Reduce the cycles per instruction (CPI) at the cost of the number of instructions per program.
- **CISC «Complex Instruction Set Computer»:** The CISC approach attempts to minimize the number of instructions per program but at the cost of an increase in the number of cycles per instruction.

RISC Architecture

RISC performs **Register-Register** architecture only

Characteristics:

- Simpler instruction, hence simple instruction decoding.
- Instruction comes undersize of one word.
- Instruction takes a single clock cycle to get executed.
- More general-purpose registers.
- Simple Addressing Modes.
- Fewer Data types.
- A pipeline can be achieved.

CISC Architecture

It can perform **Register-register**, **Register-Memory** architectures

Characteristic:

- Complex instruction, hence complex instruction decoding.
- Instructions are larger than one-word size (more than one cycle for instruction fetch).
- Many instructions require multiple clock cycles to complete, due in part to the ability to access memory.
- Less number of general-purpose registers as operations get performed in memory itself.
- Complex Addressing Modes.
- More Data types.

End of chapter 1
Any questions?