CS/ECE 5381/7381 Computer Architecture Spring 2023

Dr. Manikas

Computer Science

Lecture 3: Jan. 24, 2023

Fundamentals of Quantitative Design and Analysis

(Chapter 1, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Outline

- 1.1 Introduction
- 1.2 Classes of Computers
- 1.3 Defining Computer Architecture
- 1.4 Trends in Technology
- 1.5 Trends in Power and Energy in Integrated Circuits
- 1.6 Trends in Cost
- 1.7 Dependability
- 1.8 Measuring, Reporting, and Summarizing Performance
- 1.9 Quantitative Principles of Computer Design

Dependability

- Module reliability
  - Mean time to failure (MTTF)
  - Mean time to repair (MTTR)
  - Mean time between failures (MTBF) = MTTF + MTTR
  - Availability = MTTF / MTBF
- Failures in Time (FIT)
  - Rate of failures per *billion* hours
  - MTTF = 10^9/FIT

MTTF — MEAN TIME — IDEALLY
FAILURES (LARGE NUMBER)

MTTR
REPAIRS (SMALL NUMBER)

$$MTBF = MTTF + MTTR$$

$$\text{AVAILABILITY} = \frac{MTTF}{MTBF} = \frac{MTTF}{MTTF + MTTR} = \frac{\text{LARGE}}{\text{LARGE + SMALL}}$$

ideal availability = 1 (100%) $\Rightarrow$ close to 1

↓

means your device is up 100% of the time that's when it's available

obviously real devices is not going to be that.

Real availability < 1

maybe 0.9 Good.

Example 1.7.1

- Our Peruna chip has the following parameters
  - FIT = 150
  - MTTR = 2 days

a. What is the MTTF of our chip?

b. What is the availability of our chip?

PERUNA CHIP:

(a) MTTF?

GIVEN: FIT = 150

MTTR = 2 DAYS

RECALL: $MTTF = \frac{10^9}{FIT} = \frac{10^9}{150} \approx 6.6\overline{7} \times 10^6$ HOURS

$\approx 6.6\overline{7}$ MILLION HOURS.

(b) AVAILABILITY OF PERUNA CHIP?

$AVAILABILITY = \frac{MTTF}{MTTF + MTTR}$

GIVEN: MTTR = 2 Days $= 2 \text{ days} \left(\frac{24 \text{ Hours}}{\text{Days}}\right) = 48$ Hours.

MTTF? FROM PART (a), $MTTF = 6.6\overline{7} \times 10^6$ HOURS

$AVAILABILITY = \frac{6.6\overline{7} \times 10^6}{6.6\overline{7} \times 10^6 + 48} \approx \underline{0.999999}$ (6 DIGITS)

Outline

- 1.1 Introduction
- 1.2 Classes of Computers
- 1.3 Defining Computer Architecture
- 1.4 Trends in Technology
- 1.5 Trends in Power and Energy in Integrated Circuits
- 1.6 Trends in Cost
- 1.7 Dependability
- 1.8 Measuring, Reporting, and Summarizing Performance
- 1.9 Quantitative Principles of Computer Design

Measuring, Reporting, and Summarizing Performance

- Performance = "speed" of computer
- User view – how fast does my program run?
  - Execution (response) time: time between start and end of an event
- Web administrator view – how many transactions/hour?
  - Throughput: total amount of work done in a given time

Measuring Execution Time

- User view: "Wall-clock" time
  - E.g., process started at 2:00 pm, ended at 2:10 pm
- Actual execution time: CPU time
  - Time that processor is actually computing
  - Ignores waiting time for I/O, other programs

like waiting for a slow human to enter things in the keyboard.

Benchmarks

- Common set of programs to run on different computers
- SPEC (Standard Performance Evaluation Corporation)
  - Started in 1988 by group of workstation vendors.
  - Non-profit org - used as indep. testing source.
  - Goal: produce benchmarks that measure "real" performance.
  - Becoming the standard for performance measurement.

SPEC Benchmarks

- Benchmark types:
  - Open Systems: benchmarks for PCs, servers
  - High Performance: supercomputers
  - Graphics: high-end graphical workstations (CAD, simulators, games)

Reporting Performance Results

- Experiments should be *reproducible*
  - Report sufficient information such that another researcher can get the same results (assuming he/she follows your exact approach)
- SPEC benchmark reports require detailed info on computer, compiler, program parameters, etc.

Summarizing Performance Results

- We want to compare two computers (A and B)
- Run several different SPEC benchmarks on both computer A and B
- For computer j, SPECRatio for a given benchmark is

$$SPECRatio_j = \frac{Exec\_Time_{ref}}{Exec\_Time_j}$$

$$\text{where ref} = \text{a reference computer (baseline)}$$

$$\text{shorter exec time} \Rightarrow \text{larger SPECRatio}$$

↑ better.

Outline

- 1.1 Introduction
- 1.2 Classes of Computers
- 1.3 Defining Computer Architecture
- 1.4 Trends in Technology
- 1.5 Trends in Power and Energy in Integrated Circuits
- 1.6 Trends in Cost
- 1.7 Dependability
- 1.8 Measuring, Reporting, and Summarizing Performance
- 1.9 Quantitative Principles of Computer Design

Quantitative Principles of Computer Design

1. Parallelism
2. Principle of locality
3. Focus on common case
4. Amdahl's Law
5. Processor Performance Equation

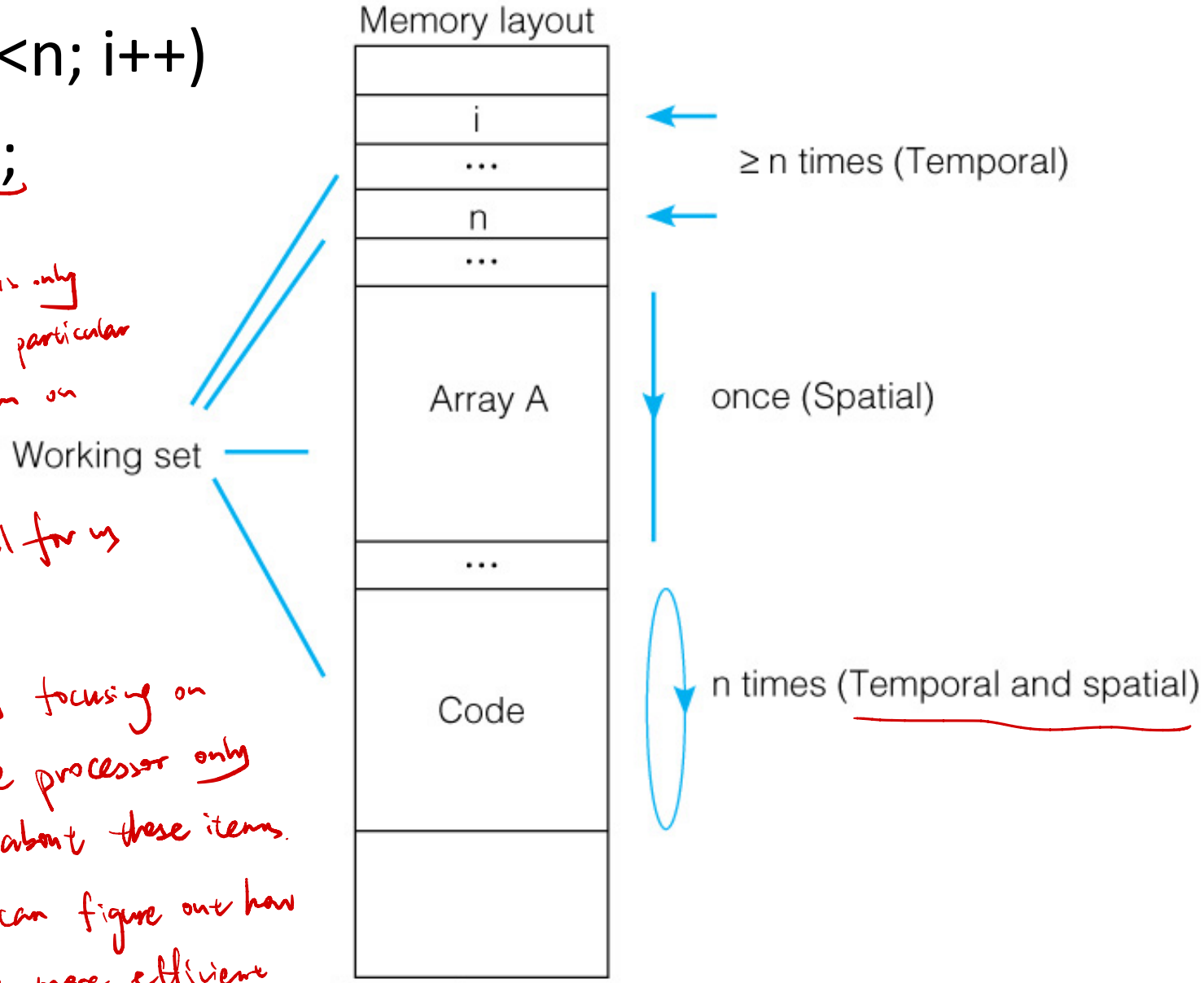
Take Advantage of Parallelism

- Parallel operations = faster execution time
- System level – multiple processors
- Processor level – can we do two (or more) tasks at the same time? (pipelining)

Principle of Locality

- Programs tend to reuse recent data and instructions
- **Temporal Locality:** items that have been recently accessed will likely be accessed again soon
- **Spatial Locality:** items that are near each other (memory address) will likely be accessed close together in time

```
for (i=0; i<n; i++)
    A[i] = 0;
```



the processor is only focusing on one particular part of a program on its time.

Why is it helpful for us in parallelism?

if we're only focusing on that means the processor only has to worry about these items. So that we can figure out how to make that more efficient for this. We don't care the rest of the program is doing

Focus on the Common Case

- Most designs have trade-offs: optimizing for one objective will degrade another objective
- For a given design, optimize for main purpose
  - Graphics workstation – optimize mathematical operation speed, at expense of power
  - Laptop – optimize battery life, at expense of operation speed

You got to think about what does the customer want and design for that and it maybe at the expense of something else.

Amdahl's Law

- Performance gain by improving a computer system component's performance

$$Speedup = \frac{Exec\_Time_{old}}{Exec\_Time_{new}}$$

Amdahl's Law Factors

- Computation time fraction that will be affected by update
  - Total original execution time T
  - Original component execution time E (E < T)
  - Fraction F = E/T
- Improvement gained by update
  - speedup

that's actually effective that's what's going to affect your speedup.

$$Speedup = \frac{Exec\_Time_{old}}{Exec\_Time_{new}}$$

$$Exec\_Time_{new} = Exec\_Time_{old}\left[\left(1-F\right)+\frac{F}{Speedup_{Enhanced}}\right]$$

$$Speedup = \frac{1}{\left(1-F\right)+\frac{F}{Speedup_{Enhanced}}}$$

Example 1.9-1

- We have a program that is 80% "parallelizable":
  - 80% of the program can be run in parallel, while 20% must be run sequentially.
- This program is currently run on a uniprocessor machine.
  - What is the speedup if we run this program on a quad core machine (4 processors)?

$$\text{SPEEDUP} = \frac{1}{(1-F) + \frac{F}{\text{SPEEDUP}_{\text{ENHANCED}}}}$$

increasing the number of processors.

GIVEN: PROGRAM IS 80%

"PARALLELIZABLE"

$\Rightarrow F = 80\% = 0.8$

RUN ON UNIPROCESSOR (ONE PROCESSOR)

→ SWITCH TO 4 PROCESSORS

why not 4.
Because we only can do 80% parallelizable

$$\text{SPEEDUP}_{\text{ENHANCED}} = \frac{4 \text{ PROC}}{1 \text{ PROC}} = 4$$

$$\text{SPEEDUP} = \frac{1}{(1-F) + \frac{F}{\text{speed}_{\text{enhanced}}}} = \frac{1}{(1-0.8) + \frac{0.8}{4}} = \frac{1}{0.2 + 0.2} = \boxed{2.5}$$

Processor Performance Equation

- Processor performance depends on
  - Clock cycle time
  - CPI (Clock cycles Per Instruction)
  - Instruction count

Clock Cycle Time

- Computers controlled by clock running at constant rate (clock rate in cycles/sec)
- A program takes several clock cycles to run
- CPU time = (clock cycles)/(clock rate)

CPI and Instruction Count

- Instruction Count (IC) = number of instructions for a program
- CPI = (clock cycles for a program)/IC
- Thus, CPU time = (IC)(CPI)/(clock rate)

CPU Time T_{exe}

$$T_{exe} = (IC)(CPI)(T_c)$$

- IC = Instruction Count
  - # of instructions in program being executed
- CPI = Cycles Per Instruction
  - Average # of cycles required to process each instruction
- T_c = clock cycle time
  - Amount of time required to execute one clock cycle

Alternate Form

$$T_{exe} = (IC)(CPI)/(R_c)$$

- IC = Instruction Count
  - # of instructions in program being executed
- CPI = Cycles Per Instruction
  - Average # of cycles required to process each instruction
- R_c = clock rate
  - Cycles/sec (Hz)
  - Reciprocal of clock cycle time: $R_c = 1/T_c$

why do we use the clock rate because guess what's how computers respect.

When you get one, it's you have a clock it tells you a clock rate of what you know maybe a gigahertz or something.

EXECUTION TIME

$$T_{EXE} = (IC)(CPI)(T_C)$$

$$= \#\,Instr \; \frac{CYCLES}{INSTR} \; \frac{SEC}{CYCLE} = \boxed{SEC}$$

$$T_{EXE} = \frac{(IC)(CPI)}{F_C}$$

$$= \frac{\#\,INSTR \; \frac{CYCLES}{INST}}{\frac{CYCLES}{SEC}} = \frac{1}{\frac{1}{SEC}} = \boxed{sec}$$

Example 1.9-2

- Your design team has designed a processor with code name of “Pony”.
- This processor has a clock cycle time of 2 ns and an average CPI of 1.
- If this processor runs a program with instruction count of 2×10^9, what is the resultant **execution time**?

nanoseconds

GIVEN : CLOCK CYCLE TIME = 2ns $= T_c$

$\hookrightarrow 2 \times 10^{-9}$ sec

CPI = 1

INSTRUCTION COUNT $= 2 \times 10^9 = IC$

EXECUTION TIME? $= T_{EXE}$

$T_{EXE} = (IC)(CPI)(T_c)$

$= (2 \times 10^9)(1)(2 \times 10^{-9} \text{ sec}) = \boxed{4 \text{ sec}}$

Example 1.9-3

- Your design team has developed a new processor with code name of “Peruna”.
- This processor has an average cycles per instruction of 2.
- When run on a SPEC benchmark program that has 10^9 instructions, the total processor execution time is 10 seconds.
- What is the **clock rate** for this processor?

GIVEN: $CPI = 2$

$IC = 10^9$

$T_{EXE} = 10$ sec

GIVEN

Clock RATE ? R_C

$$T_{EXE} = \frac{(IC)(CPI)}{R_C}$$

$$R_C = \frac{(IC)(CPI)}{T_{EXE}} = \frac{(10^9)(2)}{10} = 2 \times 10^8 \frac{\text{cycle}}{\text{sec}} \approx \boxed{200 \text{ MHz}}$$

Instruction Set Principles

(Appendix A, Hennessy and Patterson)

Note: some course slides adopted from publisher-provided material

Notes

- This section will be a review of basic instruction set principles covered in CS 4381/ECE 3382 or equivalent course

Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

Introduction

- Recall Instruction Set Architecture (ISA)
  - Instruction set - set of all possible operations in a machine's language
  - Machine's memory
  - Programmer accessible registers
  - Boundary between software and hardware
- Portion of computer visible to programmer

Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

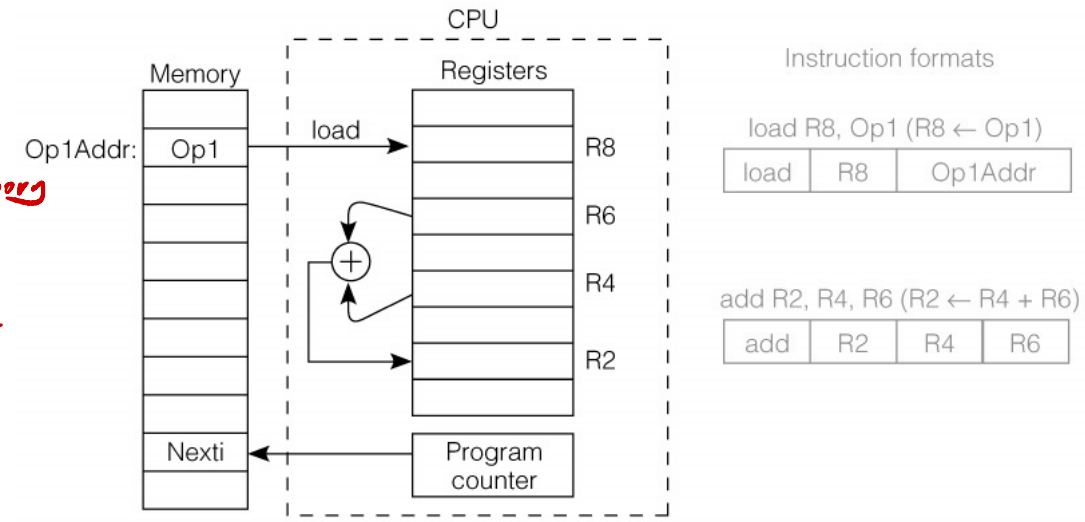
Classifying Instruction Set Architectures

- ISA's are classified by:
  - CPU registers - size and type of data (address, arith/logic, status)
  - Main memory structure
  - Method of operand access
- Taxonomy – science of classification
  - **MODERN ISA'S USE "LOAD-STORE" ARCHITECTURE**

Load-Store Architecture

the reason why we do this it's much faster to access the registers than accessing the memory

remember what are we trying to do we're trying to speed up processor performance.



Memory can only be accessed via load/store instructions

e.g., Z = A + B

```
LOAD R6,A        ;Register R6 ← A
LOAD R4,B        ;R4 ← B
ADD R2,R6,R4     ;R2 ← R6+ R4
STO R2,Z         ;Z ← R2
```

Outline

- A.1 Introduction
- A.2 Classifying Instruction Set Architectures
- A.3 Memory Addressing
- A.4 Type and Size of Operands
- A.5 Operations in the Instruction Set
- A.6 Instructions for Control Flow
- A.7 Encoding an Instruction Set
- A.9 MIPS Architecture

Memory Addressing

- For any ISA class, must define how memory is addressed
  - Address interpretation
  - Address specification

Interpreting Memory Addresses

- Memory size
  - 2^{10} = 1 K (kilo)
  - 2^{20} = 1 M (mega)
  - 2^{30} = 1 G (giga)
  - 2^{40} = 1 T (tera)
  - 2^{50} = 1 P (peta)
- 1 B (byte) = 8 b (bits)

MEMORY SIZE? Example

16 GB RAM

256 GB SSD

$16\,GB = 2^4 2^{30}\,B = 2^{34}\,B$

$256\,GB = 2^8 2^{30}\,B = 2^{38}\,B$

Word Size

- CPU word size = w
  - Typical: w = 16 or 32 bits
- Memory word size = s
  - Typical: s = 8 bits (memory is *byte-addressable*)
- Need to identify order of memory words in CPU word

Word-Ordering Schemes

- Big Endian
  - Stores the **MSB** ("the big end") at the **lower** byte address and the **LSB** at the **higher** byte address
- Little Endian
  - Stores the **LSB** ("the little end") at the **lower** byte address and the **MSB** at the **higher** byte address
- Example:
  - store ABCD1234H at word address location 0
  - AB is MSB, 34 is LSB

Memory Ordering Example

Big Endian

Byte Address	0	1	2	3
Data	AB	CD	12	34

Little Endian

Byte Address	0	1	2	3
Data	34	12	CD	AB