

CS250: FOUNDATIONS OF COMPUTER SYSTEMS

[MACHINE LANGUAGE]

The end of the party

Have you a plodding program?

Just you wait and soon it will sprint

Miniaturization, Dennard's scaling

working in tandem good times

doubling densities constant power

Moore's Law faster clocks, magical speedups

Alas the party's come to a crashing end

In the wee hours of the millennium

Running headlong into quantum effects

Laid waste by heat

SHRIDEEP PALLICKARA

Computer Science

Colorado State University

COMPUTER SCIENCE DEPARTMENT



COLORADO STATE UNIVERSITY

1

Frequently asked questions from the previous class survey

- Memory mapped I/O
 - ▣ If I plug-in 10,000 keyboard to 1 computer, could I run out of memory



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.2

2

Topics covered in this lecture

- Assembly language
- Instruction set architectures
- Why miniaturization matters in hardware
- Moore's Law
- Dennard's scaling



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.3

3

Works of imagination should be written in very plain language; the more purely imaginative they are, the more necessary it is to be plain.
—Samuel Taylor Coleridge (1772–1834)

MACHINE LANGUAGE

4

The simplest program out there

```
/** The simplest program out there! */  
  
public class HelloWorld {  
  
    /** This does not even take an argument */  
    public static void main(String[] args) {  
  
        System.out.println("Hello World");  
    }  
  
}
```



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.5

5

What does it take to actually run this? [1/2]

- Let's look under the hood
- For starters, note that the program is nothing more than a **sequence of plain characters**, stored in a text file
- This abstraction is a complete mystery for the computer, which understands only instructions written in machine language



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.6

6

What does it take to actually run this? [2/2]

- The first thing we must do is **parse** the string of characters of which the high-level code is made, **uncover its semantics**—figure out what the program seeks to do
 - And then generate low-level code that **reexpresses this semantics** using the **machine language** of the target computer
- The result of this elaborate translation process, known as **compilation**, will be an **executable sequence of machine language instructions**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.7

7

Machine language is also an abstraction

- An *agreed upon* set of binary codes
- To make this abstraction concrete
 - It must be realized by some hardware architecture
 - And this architecture, in turn, is implemented by a certain set of chips — registers, memory units, adders, and so on
 - Now, every one of these hardware devices is constructed from lower-level, elementary logic gates
 - And these gates, in turn, can be built from primitive gates like Nand and Nor
 - These primitive gates are very low in the hierarchy, but they, too, are made of several switching devices, typically implemented by transistors



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.8

8

But this is so much easier on your computer

- On your computer, compiling and running programs is much easier
 - ▣ All you have to do is click this icon or write that command!
- Indeed, a modern computer system is like a submerged iceberg
 - ▣ Most people get to see only the top
 - ▣ Knowledge of computing systems is often sketchy and superficial



A machine language is an agreed-upon formalism designed to code machine instructions

- Using these instructions, we can instruct the computer's processor to:
 - ▣ Perform arithmetic and logical operations
 - ▣ Read and write values from and to the computer's memory
 - ▣ Test Boolean conditions
 - ▣ Decide which instruction to fetch and execute next



Design goals in high-level and machine languages differ

- Design goals in high-level languages
 - Cross-platform compatibility and power of expression
- Machine languages are designed to effect **direct execution** in, and total control of, a specific hardware platform
 - Of course, generality, elegance, and power of expression are still desired
 - But only to the extent that they support the basic requirement of direct and efficient execution in hardware



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.11

11

Machine language is the most profound interface in the computer enterprise

- The fine line **where hardware meets software**
- The point where the abstract designs of humans, as manifested in high-level programs, are finally reduced to physical operations performed in silicon



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.12

12

A machine language is both a programming artifact and an integral part of the hardware platform

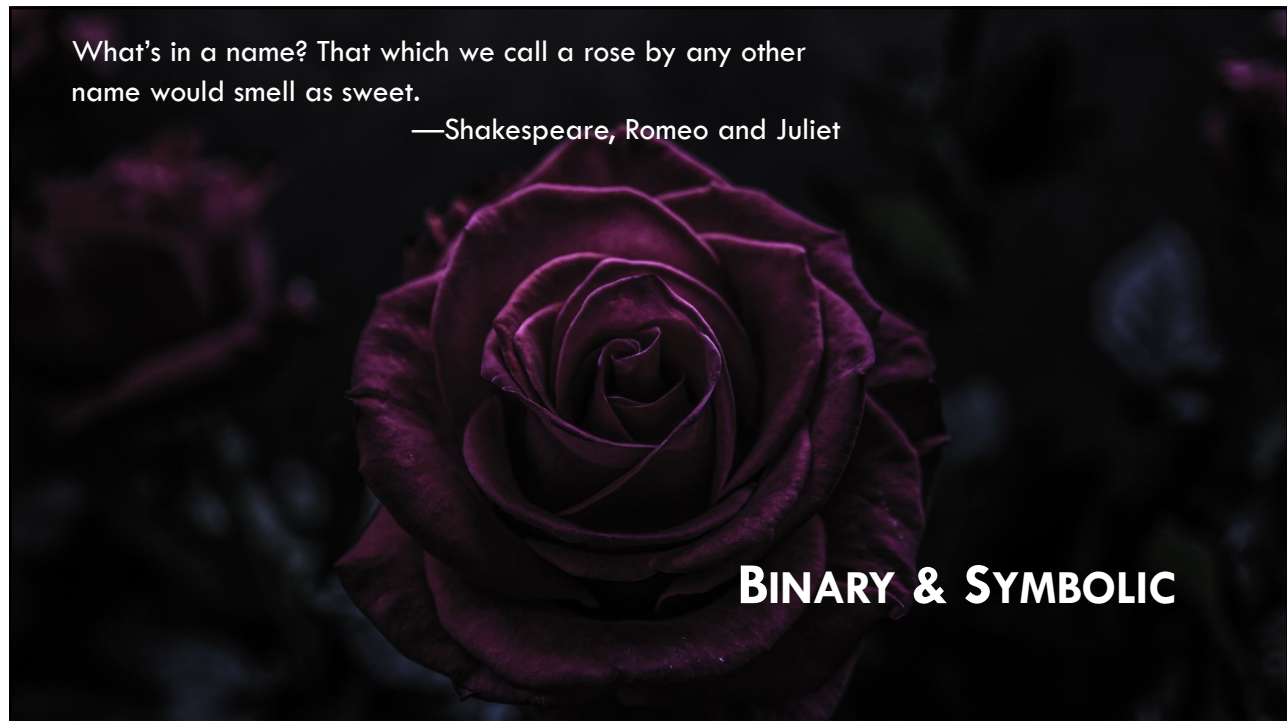
- Just as we say that the machine language is **designed to control** a particular hardware platform
- We can say that the hardware platform is **designed to execute** instructions written in a particular machine language



Who writes machine language programs?

- Even the most sophisticated software systems are, at bottom, streams of simple instructions
 - Each specifying a primitive operation on the underlying hardware
- It should be noted that machine language programs are rarely written by humans
- Rather, they are typically written by **compilers**
- And a compiler — being an automaton — can optionally bypass the symbolic instructions and generate binary machine code directly






15

Writing machine language programs

- Machine language programs can be written in two alternative, but equivalent, ways
 - Binary
 - Symbolic

 **COLORADO STATE UNIVERSITY** Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT **MACHINE LANGUAGE** **L14.16**

16

Machine Language: Binary vs Symbolic

- Consider the abstract operation “set R1 to the value of $R1 + R2$ ”
- Language designers, can decide to represent
 - ▣ The addition operation using the 6-bit code 101011,
 - ▣ Registers R1 and R2 using the codes 00001 and 00010, respectively
- Assembling these codes left to right:
 - ▣ The 16-bit instruction 1010110001000001 can be used as the binary version of “set R1 to the value of $R1 + R2$ ”



In the early days of computer systems, computers were programmed manually

- When proto-programmers wanted to issue the instruction “set R1 to the value of $R1 + R2$ ”
 - ▣ They pushed up and down **mechanical switches** that stored a binary code like 1010110001000001 in the computer’s instruction memory
- And if the program was a hundred instructions long?
 - ▣ They had to go through this ordeal a hundred times
- Of course, debugging such programs was a perfect nightmare



Symbolic codes to the rescue

- This led programmers to invent and use **symbolic codes**
 - Convenient way for documenting and debugging programs **on paper, before** entering them into the computer
- For example, the symbolic format `add R2, R1` could be chosen
 - For representing the semantics “set R1 to the value of R1 + R2” and the binary instruction `1010110001000001`



It didn't take long before several people hit on the same idea

- Symbols like R, 1, 2, and + can also be represented using agreed-upon binary codes
- **Why not use symbolic instructions for writing programs?**
 - And then use another program —a translator— for translating the symbolic instructions into executable binary code?
- This innovation liberated programmers from the tedium of writing binary code
 - Paving the way for the subsequent onslaught of high-level programming languages



Symbolic machine languages

- Symbolic machine languages are called **assembly languages**
- The programs that *translate* them into binary code are called **assemblers**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.21

21

High-level vs Assembly languages

- Syntax of high-level languages
 - ▣ Portable and **hardware independent**
- The syntax of an assembly language?
 - ▣ Tightly related to the low-level details of the target hardware
 - The available ALU operations, number and type of registers, memory size, and so on



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.22

22

But there is so much diversity in hardware

- Since different computers vary greatly in terms of any one of these parameters, there is a *Tower of Babel* of machine languages
 - Each with its obscure syntax
 - Each designed to control a particular family of CPUs
- Irrespective of this variety, though, **all machine languages are theoretically equivalent**
 - All of them support similar sets of generic tasks



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.23

23

Symbolic language & the Assembler [1/2]

- The symbolic version includes all sorts of things that humans are fond of seeing in computer programs
 - Comments, white space, indentation, symbolic instructions, and symbolic references
- None of these embellishments concern computers, which understand one thing only: bits



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.24

24

Symbolic language & the Assembler

[2/2]

- The agent that bridges the gap between the symbolic code convenient for humans and the binary code understood by the computer is the **assembler**
- The assembler takes as input a stream of assembly instructions and generates as output a **stream of translated binary instructions**
 - The resulting code can be loaded as is into the computer memory and executed



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

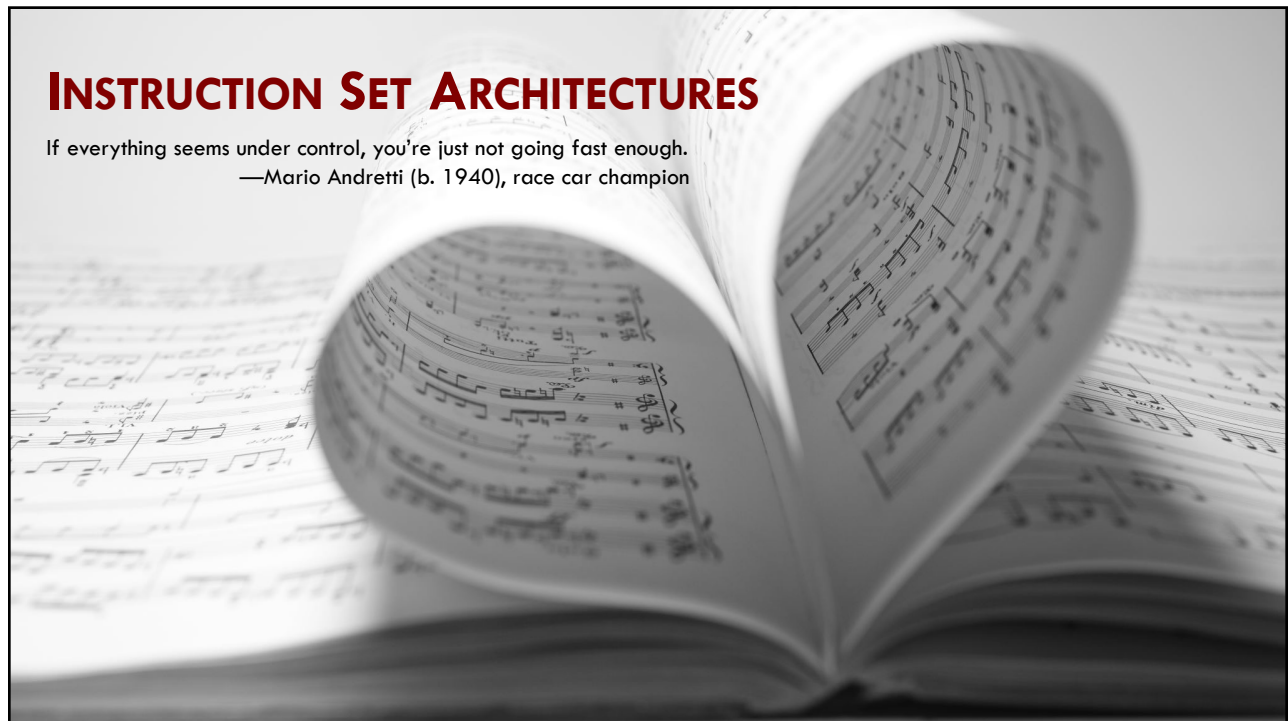
MACHINE LANGUAGE

L14.25

25

INSTRUCTION SET ARCHITECTURES

If everything seems under control, you're just not going fast enough.
—Mario Andretti (b. 1940), race car champion



26

CPU instructions

- Although all CPUs implement arithmetic instructions, the specific instructions available on different processors vary
- Some instructions that exist for one type of CPU simply don't exist on other types of CPUs
- Even instructions that **do exist** on nearly all CPUs **aren't implemented in the same way**
 - For example, the specific binary sequence used to mean "add two numbers" is not the same across processor types



A family of CPUs that use the same instructions are said to share an **instruction set architecture (ISA)**

- An ISA is also a **model** of how a CPU works
- Software that's built for a certain ISA **works on any CPU that implements that ISA**
 - It's possible for multiple processor models, even those from different manufacturers, to implement the same architecture
 - Such processors **may work very differently internally**, but by adhering to the same ISA, they can run the same software



Today, there are two prevalent instruction set architectures

- x86
 - ▣ Personal computers, desktops, and servers
- ARM
 - ▣ Hand-held devices



The majority of desktop computers, laptops, and servers use x86 CPUs

- The term **x86** refers to a set of related architectures
- The name comes from Intel Corporation's naming convention for its processors (each ending in 86)
 - ▣ Beginning with the 8086 released in 1978, and continuing with the 80186, 80286, 80386, and 80486
 - ▣ After the 80486 (or more simply the 486), Intel began branding its CPUs with names such as Pentium and Celeron
 - These processors are still x86 CPUs despite the name change
- Other companies besides Intel also produce x86 processors
 - ▣ Most notably Advanced Micro Devices, Inc. (AMD)



Over time, new instructions have been added to the x86 architecture

- But each generation has tried to **retain backward compatibility**
- This generally means that software developed for an older x86 CPU runs on a newer x86 CPU
 - But software built for a newer x86 CPU that takes advantage of new x86 instructions won't be able to run on older x86 CPUs
 - Older x86 CPUs don't understand the new instructions: **no forward compatibility**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.31

31



32

Generations of the x86 architecture

- The x86 architecture includes three major generations of processors:
 - 16-bit
 - 32-bit
 - 64-bit



What do we mean when we say that a CPU is a 16-bit, 32-bit, or 64-bit processor?

- The number of bits associated with a processor refers to the **number of bits it can deal with at a time**
 - Also known as its **bitness** or **word size**
- So, a 32-bit CPU can operate on values that are 32 bits in length
 - More specifically, this means that the computer architecture has 32-bit registers, a 32-bit address bus, or a 32-bit data bus
 - Or all three may be 32-bit



Bitness

- The original 8086 processor, released in 1978, was a 16-bit processor
 - ▣ Intel's subsequent x86 processors were also 16-bit until the 80386 processor
- The 80386 released in 1985, brought with it a new 32-bit version of the x86 architecture
 - ▣ This 32-bit version of x86 is sometimes called **IA-32**
- Thanks to backward compatibility, modern x86 processors still fully support IA-32



Interestingly, it was AMD, and not Intel, that brought x86 into the 64-bit era

- In the late 1990s, Intel's 64-bit focus was on a new CPU architecture called IA-64 or Itanium
 - ▣ This was not an x86 ISA, and ended up as a niche product for servers
- With Intel focused on Itanium, AMD seized the opportunity to extend the x86 architecture
 - ▣ In 2003, AMD released the Opteron processor, the first 64-bit x86 CPU



AMD's architecture was originally known as AMD64

- Later Intel adopted this architecture and called its implementation Intel 64
- The two implementations are mostly **functionally identical**
- Today 64-bit x86 is generally referred to as x64 or **x86-64**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.37

37

Although x86 rules the PC/server world, ARM processors command the realm of mobile devices

- Multiple companies manufacture ARM processors
- A company called ARM Holdings develops the ARM architecture and licenses their designs to other companies to implement



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.38

38

It's common for ARM CPUs to be used in **system-on-chip (SoC)** designs

- Where a single integrated circuit contains not only a CPU, but also memory and other hardware
- The ARM architecture originated in the 1980s as a 32-bit ISA
- A 64-bit version of the ARM architecture was introduced in 2011



ARM processors are favored in mobile devices

- Due to their **reduced power consumption** and lower cost as compared to x86 processors
- ARM processors can be used in PCs as well
 - But that market largely remains focused on x86, to retain backward compatibility with existing x86 PC software



ARMs foray into desktops and laptops

- Started in earnest in 2020 when Apple announced their intention to move macOS computers from x86 to ARM CPUs
- The first line of computers, MacBook Pros were released with this ARM-based SoC design in 2021
- Includes mechanisms to work with executables that use legacy x86-64 and x86-32 codes
 - ▣ **Rosetta** software to use apps built for a Mac with an Intel processor



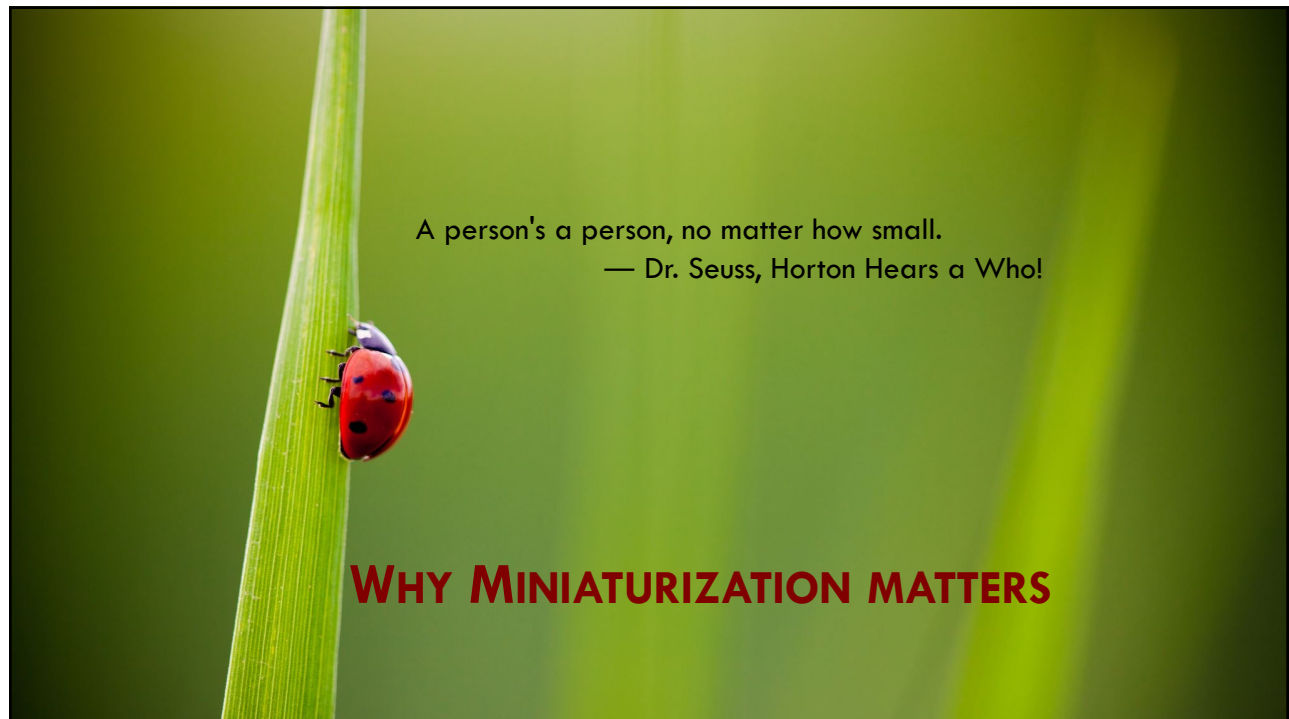
COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.41

41



42

Why Miniaturization Matters in Hardware: An analogy

- Imagine you have to drive your kids to and from school, which is 10 miles away, at an average speed of 40 miles per hour
- The combination of distance and speed means that only two round trips per hour are possible
- You can't complete the trip more quickly
 - ▣ Without either driving faster or moving closer to school



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.43

43

Modern computers drive electrons around instead of kids

- Electricity travels at the **speed of light**, which is about 300 million meters per second
 - ▣ Except in the US, where it goes about ~186,000 miles/second or a billion feet per second
- Because we haven't yet discovered a way around this physical limitation
 - ▣ The only way we can minimize travel time in computers is to **have the parts close together**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.44

44

Miniaturization and clock speeds

- Computers today can have clock speeds around 4 GHz, which means they can do four billion things per second
- Electricity only travels about **75 millimeters in a four-billionth** of a second
- A typical CPU that measures about 18 millimeters on each side
 - ▣ There's just enough time to make **two complete round trips** across this CPU in four-billionths of a second
 - ▣ It follows that making things small permits higher performance



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.45

45

Miniaturization: The energy angle

[1/2]

- When driving kids to and from school coffee alone is insufficient
 - ▣ It takes energy to travel!
- Making things small reduces the amount of travel needed, which **reduces the amount of energy** needed



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.46

46

Miniaturization: The energy angle

[2/2]

- **Reduced energy requirements** translates into **lower power consumption** and **less heat** generation
 - Reduced heat keeps your phone from burning a hole in your pocket!
- This is one of the reasons why the history of computing devices has been characterized by efforts to make hardware smaller
 - But making things very small introduces other problems



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.47

47



48

Moore's Law

[1/2]

- **Empirical observation** linked to gains from experience in production
- The number of transistors in a dense integrated circuit (IC) doubles about every two years
 - For most of the last fifty years ... equivalent to saying that computers did get **twice as fast**
- Not a law of physics
 - An observation and projection of a historical trend



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.49

49

Moore's Law

[2/2]

- Nearly two decades since what has been called “the breakdown of Moore's law”
 - And the switch to multicore processors instead of ever faster single chips
- But again, this is wrong!
 - Moore's law has not broken down at all – transistor numbers are continuing to increase
 - What has happened is that it is **no longer possible to keep running these transistors at ever faster speeds**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.50

50



51

Dennard scaling

- Named after Robert Dennard
 - ▣ Led the IBM research team that first described this effect in a 1974 paper
- As transistors got smaller the **power density was constant**
 - ▣ Power use stays in proportion with area; both voltage and current scale (downward) with length
 - ▣ If a transistor's linear size reduced by 2?
 - The power it used fell by 4!
 - With voltage and current both halving



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.52

52

Dennard's Observations

[1/3]

- With every technology generation, transistor dimensions could be scaled by -30% ($0.7\times$)
- This has the following effects **simultaneously**:
 - Chip area reduces by 50%
 - To keep the electric field constant, the voltage, V , is reduced by 30% ($0.7\times$)
 - Voltage is field times length
 - Circuit delays reduce by 30% ($0.7\times$)
 - Because time is length over velocity



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.53

53

These in turn results in changes to capacitance and chosen frequency

- The 30% reduction in delay allows an **increase in operating frequency** by about 40% ($1.4\times$)
 - Frequency varies with one over delay $1/0.7 \sim 1.4$
- The 30% reduction in all distances and related 50% drop in area leads to a decrease in capacitance, C , by 30% ($0.7\times$)
- **Power consumption decreases** by 50%, because active power is CV^2f
 - $0.7 \times 0.7^2 \times 1.4 \sim 0.5$



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.54

54

Dennard's Observations: Takeaway

- In every technology generation:
 - ▣ Area halves and ...
 - ▣ Power consumption halves!
- In other words, if the transistor density doubles?
 - ▣ **Power consumption stays the same with twice the number of transistors**



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.55

55

What has broken down is not the ability to etch smaller transistors

- We are **unable to drop the voltage and the current** needed to operate reliably
- In the run-up to hitting this wall, the reduction in the size of the transistors ran slightly ahead
 - ▣ We ended up getting to 3GHz a little faster than expected



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.56

56

So what's the problem?

- **Static power losses have increased** every more rapidly as a proportion of overall power supplied as voltages have dropped
- Notably, **static power losses heat the chip**
 - *Further* increasing static power loss and threatening **thermal runaway** – and complete breakdown
 - This is a vicious cycle not a virtuous one



Reasons for increased static power loss

- Complex **quantum effects** due to
 - Component sizes being reduced
 - Modifications to chemical composition of chips to handle miniaturizations
- There seems to be no way out!
 - “Moore’s law” ... in the sense of ever faster chips, is dead



Consequences of the breakdown on Dennard scaling

- **Unable to increase clock frequencies** significantly
- Most CPU manufacturers focus on multicore processors to improve performance
 - An increased core count benefits many workloads
 - The increase in active switching elements from having multiple cores *still results in increased overall power consumption*
 - Worsens CPU power dissipation issues



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.59

59

The problems of dissipation persist to some extent in multicores as well

- **Only a fraction** of an integrated circuit can actually be active at any given point in time without violating power constraints
- The remaining (inactive) area is referred to as **dark silicon**



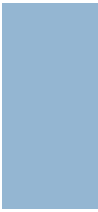
COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE


L14.60

60



ANOTHER ARCHITECTURE

COMPUTER SCIENCE DEPARTMENT




COLORADO STATE UNIVERSITY

61

The Harvard architecture

- Named after the Harvard Mark I computer a joint effort between IBM and Harvard



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALICKARA
COMPUTER SCIENCE DEPARTMENT

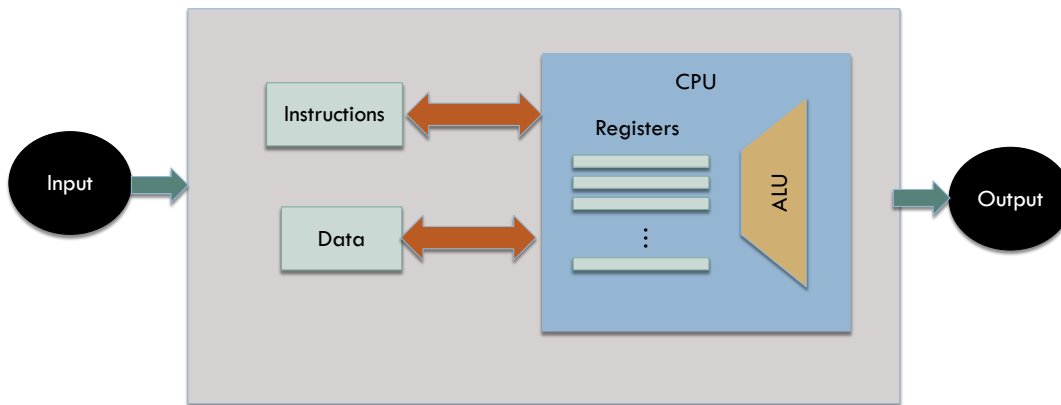
MACHINE LANGUAGE

L14.62

62

Depiction of the architecture

- The only difference between them is the way the memory is arranged



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.63

63

Some comparison points

- All else being equal, the von Neumann architecture is slightly slower
 - ▣ Because it can't access instructions and data at the same time, since there's only one memory bus
- The Harvard architecture gets around that but requires additional hardware for the second memory bus



COLORADO STATE UNIVERSITY

Professor: SHRIDEEP PALLICKARA
COMPUTER SCIENCE DEPARTMENT

MACHINE LANGUAGE

L14.64

64

The contents of this slide-set are based on the following references

- Noam Nisan and Shimon Schocken. *The Elements of Computing Systems: Building a Modern Computer from First Principles*. 2nd Edition. ISBN-10/ ISBN-13: 0262539802 / 978-0262539807. MIT Press. [Preface, Chapter 4-5]
- Jonathan E. Steinhart. *The Secret Life of Programs: Understand Computers -- Craft Better Code*. ISBN-10/ ISBN-13 : 1593279701/ 978-1593279707. No Starch Press. [Chapter 5]
- Randall Hyde. *Write Great Code, Volume 1, 2nd Edition: Understanding the Machine* 2nd Edition. ASIN: B07VSC1K8Z. No Starch Press. 2020. [Chapter 11]
- Matthew Justice. *How Computers Really Work: A Hands-On Guide to the Inner Workings of the Machine*. ISBN-10/ISBN-13 : 1718500661/ 978-1718500662. No Starch Press. 2020. [Chapter 7]
- Adrian McMenamin: <https://cartesianproduct.wordpress.com/2013/04/15/the-end-of-dennard-scaling/>
- https://en.wikipedia.org/wiki/Dennard_scaling
- https://en.wikipedia.org/wiki/Moore%27s_law

