Introduction

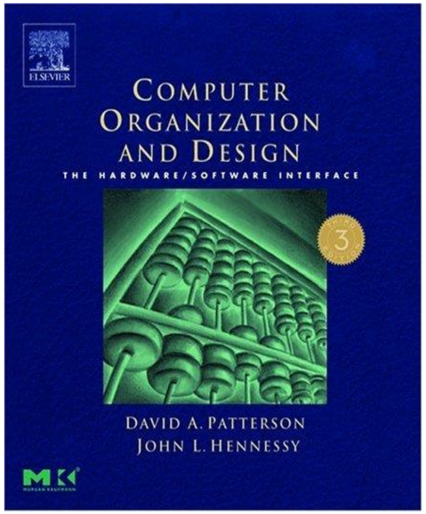
CSE 211

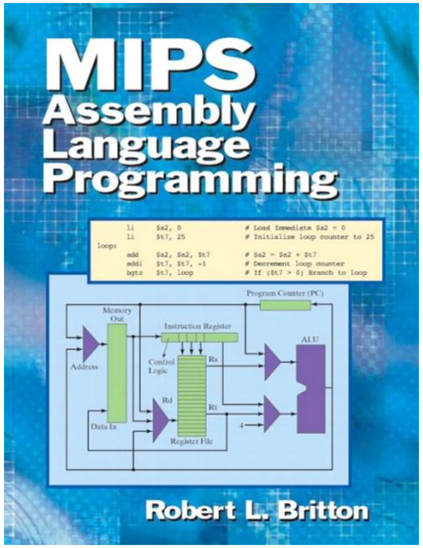
Outline

- Welcome to CSE 211
- High-Level, Assembly-, and Machine-Languages
- Components of a Computer System
- Chip Manufacturing Process
- Technology Improvements
- Programmer's View of a Computer System

Which Textbooks will be Used?

- Computer Organization & Design: The Hardware/Software Interface
  - Third Edition
  - David Patterson and John Hennessy
  - Morgan Kaufmann Publishers, 2005
- MIPS Assembly Language Programming
  - Robert Britton
  - Pearson Prentice Hall, 2004
  - Supplement for Lab
- Read the textbooks in addition to slides





Course Objectives

- Towards the end of this course, you should be able to …
    - Describe the instruction set architecture of a MIPS processor
    - Analyze, write, and test MIPS assembly language programs
    - Describe organization/operation of integer & floating-point units
    - Design the datapath and control of a single-cycle CPU
    - Design the datapath/control of a pipelined CPU & handle hazards
    - Describe the organization/operation of memory and caches
    - Analyze the performance of processors and caches

Course Learning Outcomes

- Ability to analyze, write, and test MIPS assembly language programs.
- Ability to describe the organization and operation of integer and floating-point arithmetic units.
- Ability to apply knowledge of mathematics in CPU performance analysis and in speedup computation.
- Ability to design the datapath and control unit of a processor.
- Ability to use simulator tools in the analysis of assembly language programs and in CPU design.

Required Background

- The student should already be able to program confidently in at least one high-level programming language, such as Java or C.
- Prerequisite
  - Fundamentals of computer engineering
  - Introduction to computing
- Only students with computer science or software engineering major should be registered in this course.

Software Tools

- MIPS Simulators
  - MARS: MIPS Assembly and Runtime Simulator
    - Runs MIPS-32 assembly language programs
    - Website: http://courses.missouristate.edu/KenVollmar/MARS/
  - PCSPIM
    - Also Runs MIPS-32 assembly language programs
    - Website: http://www.cs.wisc.edu/~larus/spim.html
- CPU Design and Simulation Tool
  - Logisim
    - Educational tool for designing and simulating CPUs
    - Website: http://ozark.hendrix.edu/~burch/logisim/

What is “Computer Architecture” ?

- Computer Architecture =

  Instruction Set Architecture +
  Computer Organization
- Instruction Set Architecture (ISA)
  - WHAT the computer does (logical view)
- Computer Organization
  - HOW the ISA is implemented (physical view)
- We will study both in this course

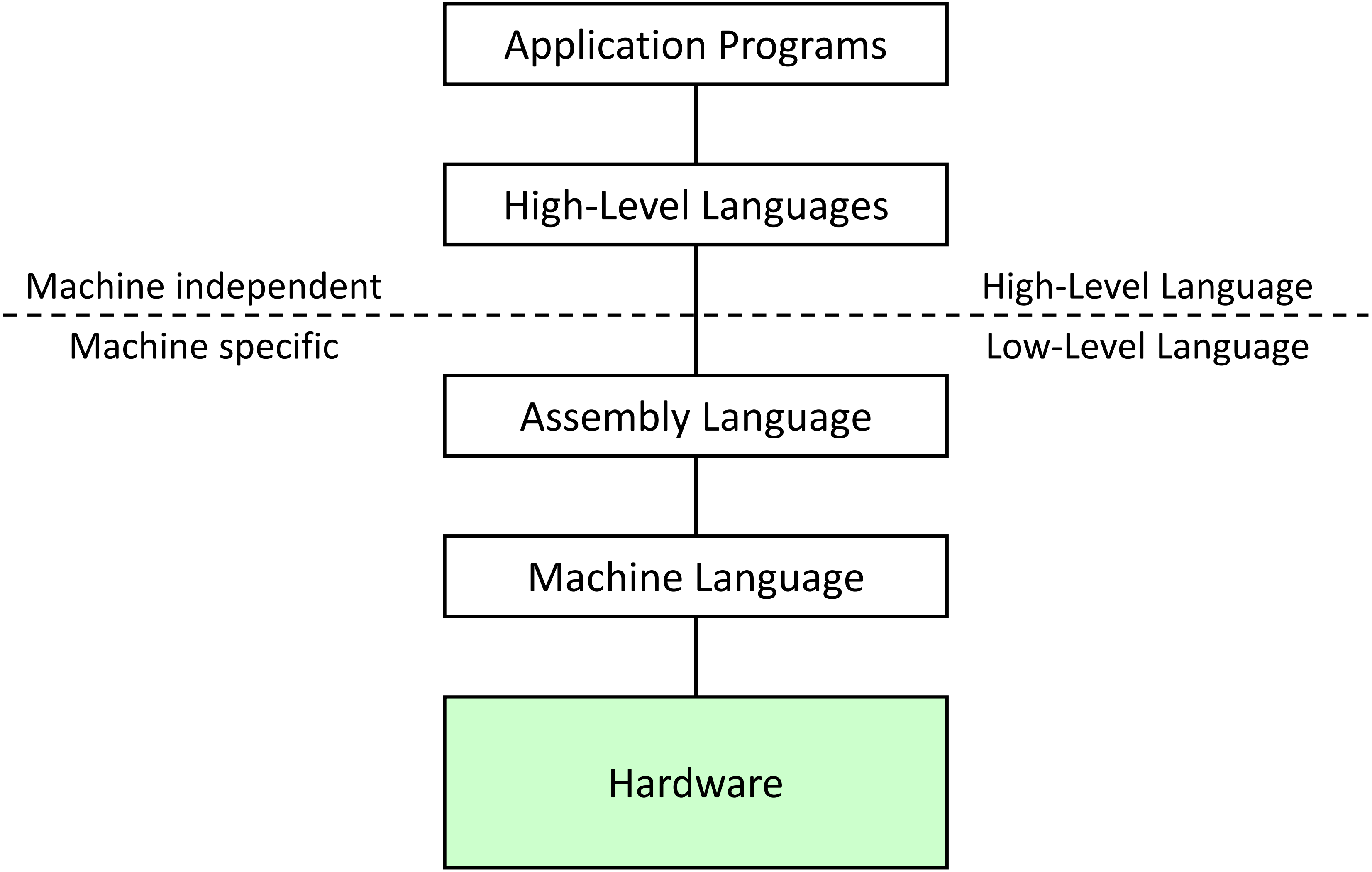
Next . . .

- Welcome to CSE 211
- High-Level, Assembly-, and Machine-Languages
- Components of a Computer System
- Chip Manufacturing Process
- Technology Improvements
- Programmer's View of a Computer System

Some Important Questions to Ask

- What is Assembly Language?
- What is Machine Language?
- How is Assembly related to a high-level language?
- Why Learn Assembly Language?
- What is an Assembler, Linker, and Debugger?

A Hierarchy of Languages



Application Programs

High-Level Languages

Machine independent

High-Level Language

Machine specific

Low-Level Language

Assembly Language

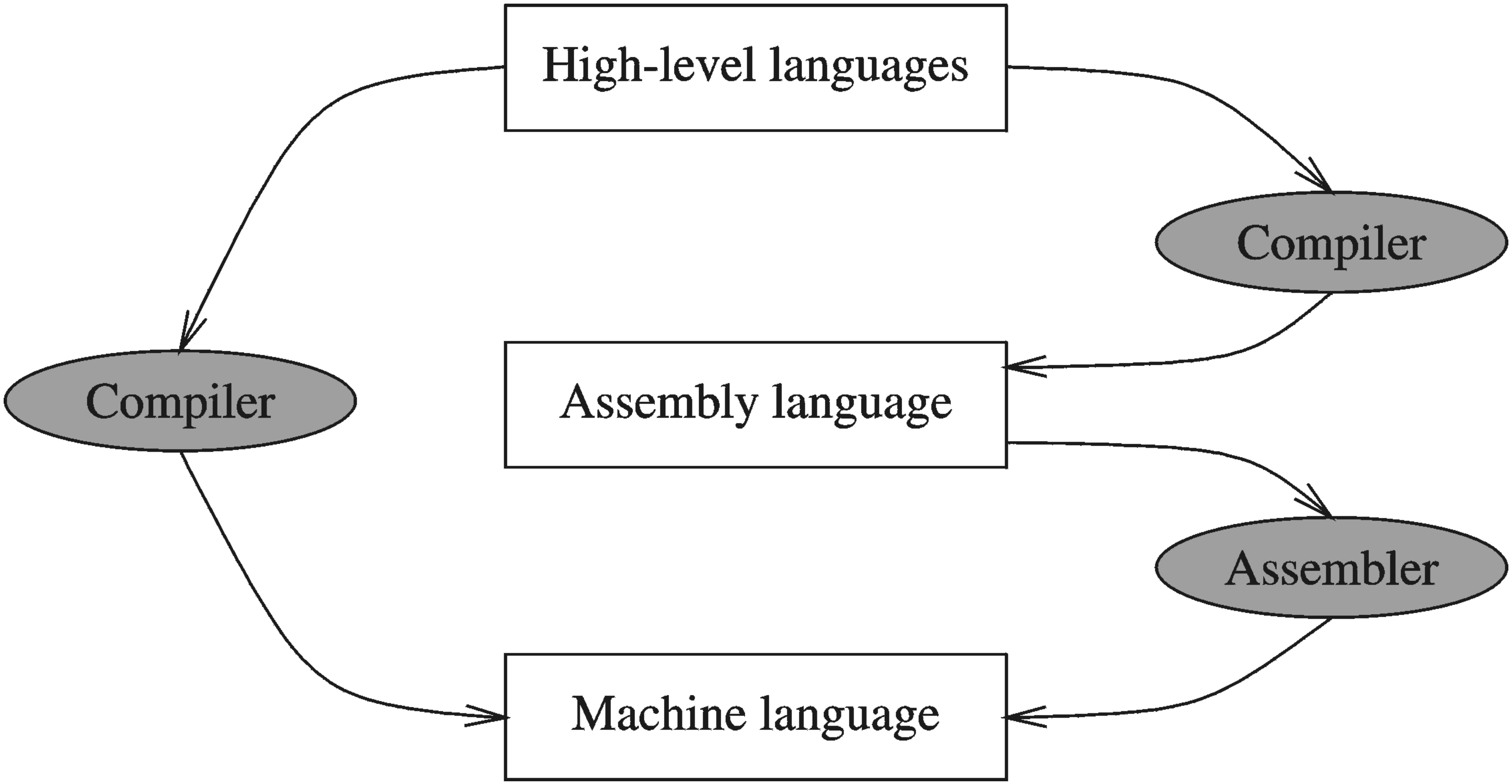
Machine Language

Hardware

Assembly and Machine Language

- Machine language
  - Native to a processor: executed directly by hardware
  - Instructions consist of binary code: 1s and 0s
- Assembly language
  - Slightly higher-level language
  - Readability of instructions is better than machine language
  - One-to-one correspondence with machine language instructions
- Assemblers translate assembly to machine code
- Compilers translate high-level programs to machine code
  - Either directly, or
  - Indirectly via an assembler

Compiler and Assembler



Instructions and Machine Language

- Each command of a program is called an instruction (it instructs the computer what to do).
- Computers only deal with binary data, hence the instructions must be in binary format (0s and 1s) .
- The set of all instructions (in binary form) makes up the computer's machine language. This is also referred to as the instruction set.

Instruction Fields

- Machine language instructions usually are made up of several fields. Each field specifies different information for the computer. The major two fields are:
- Opcode field which stands for operation code and it specifies the particular operation that is to be performed.
  - Each operation has its unique opcode.
- Operands fields which specify where to get the source and destination operands for the operation specified by the opcode.
  - The source/destination of operands can be a constant, the memory or one of the general-purpose registers.

Translating Languages

Program (C Language):

```
swap(int v[], int k) {
   int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

Compiler

MIPS Assembly Language:

```
sll $2,$5, 2
add $2,$4,$2
lw  $15,0($2)
lw  $16,4($2)
sw  $16,0($2)
sw  $15,4($2)
jr  $31
```

Assembler

MIPS Machine Language:

```
00051080
00821020
8C620000
8CF20004
ACF20000
AC620004
03E00008
```

A statement in a high-level language is translated typically into several machine-level instructions

Advantages of High-Level Languages

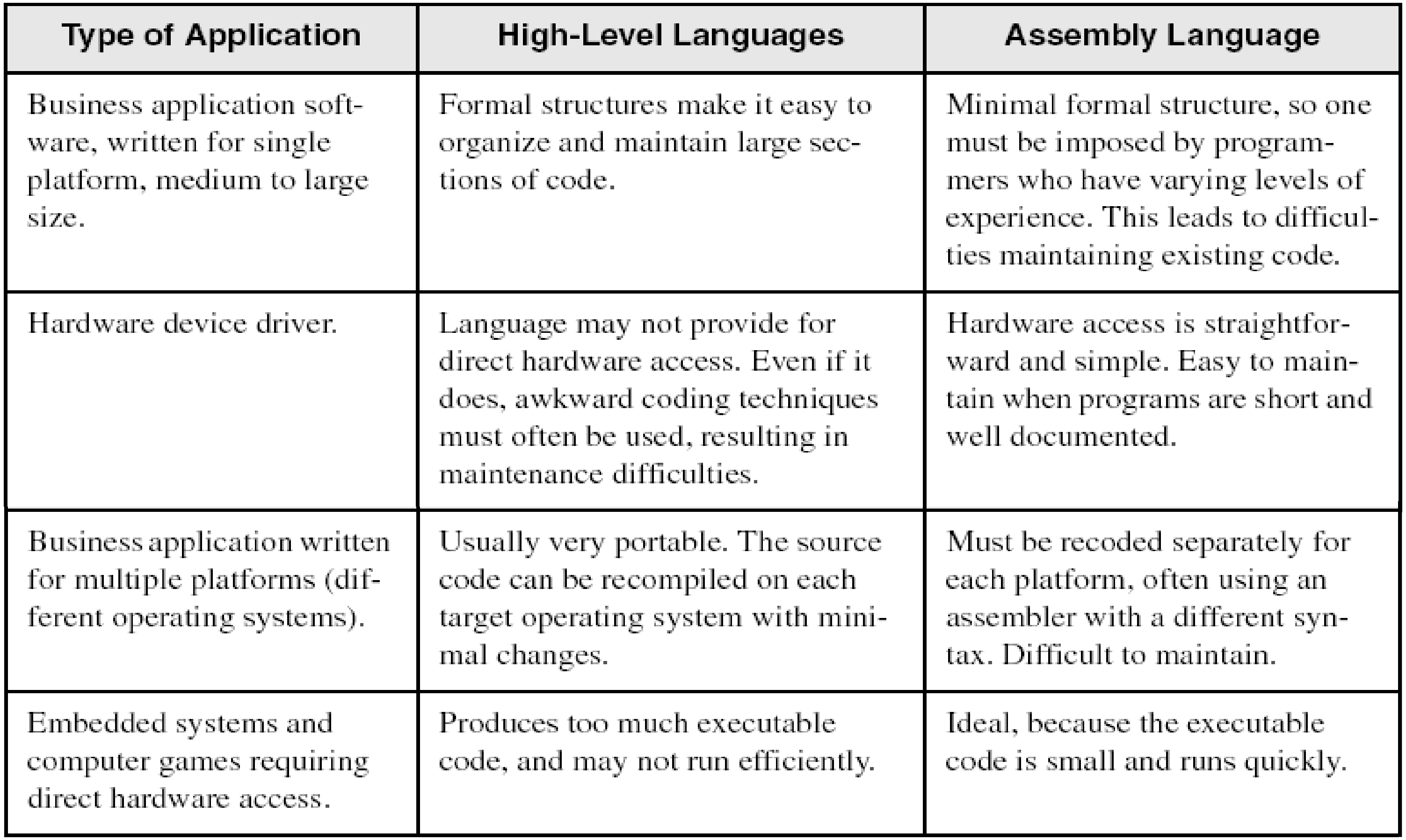
- Program development is faster
  - High-level statements: fewer instructions to code
- Program maintenance is easier
  - For the same above reasons
- Programs are portable
  - Contain few machine-dependent details
    - Can be used with little or no modifications on different machines
  - Compiler translates to the target machine language
  - However, Assembly language programs are not portable

Why Learn Assembly Language?

- Many reasons:
  - Accessibility to system hardware
  - Space and time efficiency
  - Writing a compiler for a high-level language
- Accessibility to system hardware
  - Assembly Language is useful for implementing system software
  - Also useful for small embedded system applications
- Space and Time efficiency
  - Understanding sources of program inefficiency
  - Tuning program performance
  - Writing compact code

Assembly vs. High-Level Languages

- S

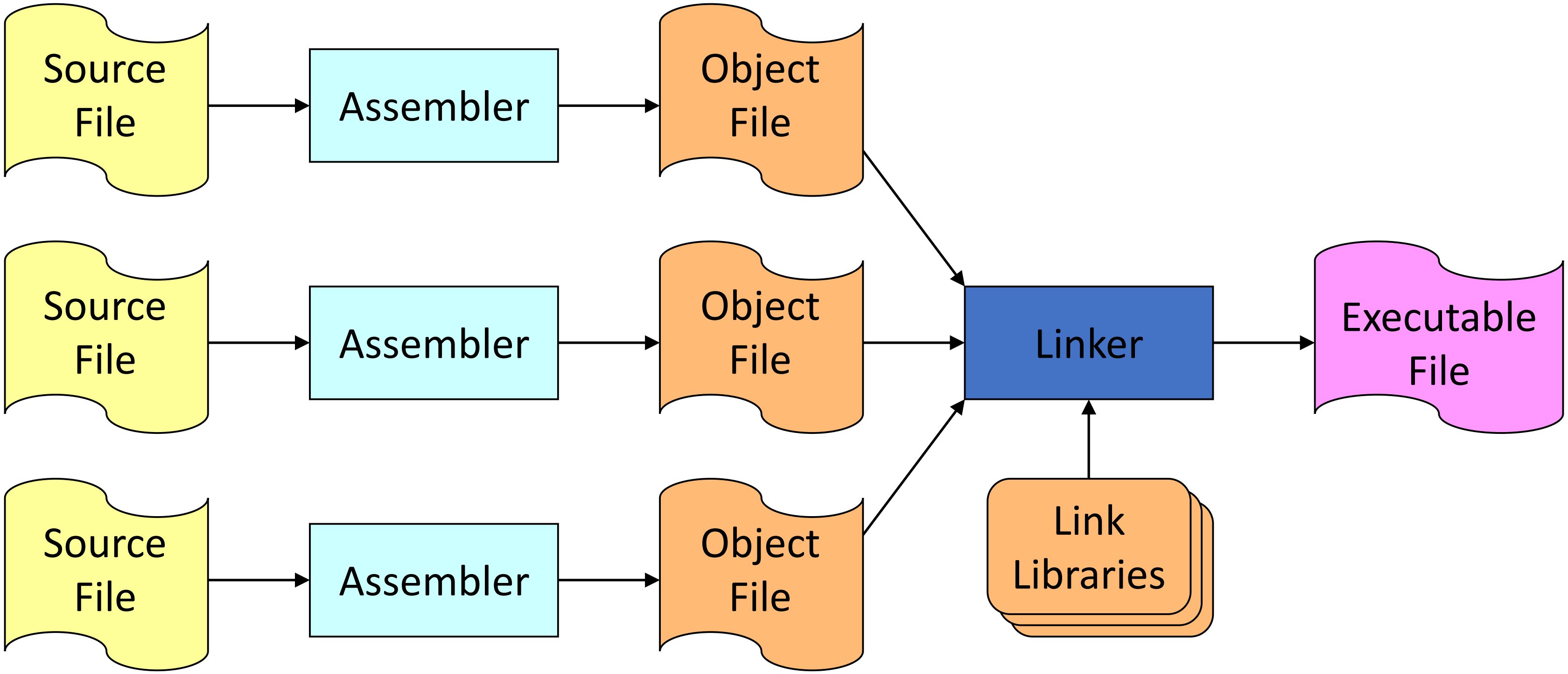


Type of Application	High-Level Languages	Assembly Language
Business application software, written for single platform, medium to large size.	Formal structures make it easy to organize and maintain large sections of code.	Minimal formal structure, so one must be imposed by programmers who have varying levels of experience. This leads to difficulties maintaining existing code.
Hardware device driver.	Language may not provide for direct hardware access. Even if it does, awkward coding techniques must often be used, resulting in maintenance difficulties.	Hardware access is straightforward and simple. Easy to maintain when programs are short and well documented.
Business application written for multiple platforms (different operating systems).	Usually very portable. The source code can be recompiled on each target operating system with minimal changes.	Must be recoded separately for each platform, often using an assembler with a different syntax. Difficult to maintain.
Embedded systems and computer games requiring direct hardware access.	Produces too much executable code, and may not run efficiently.	Ideal, because the executable code is small and runs quickly.

Assembly Language Programming Tools

- Editor
  - Allows you to create and edit assembly language source files
- Assembler
  - Converts assembly language programs into object files
  - Object files contain the machine instructions
- Linker
  - Combines object files created by the assembler with link libraries
  - Produces a single executable program
- Debugger
  - Allows you to trace the execution of a program
  - Allows you to view machine instructions, memory, and registers

Assemble and Link Process

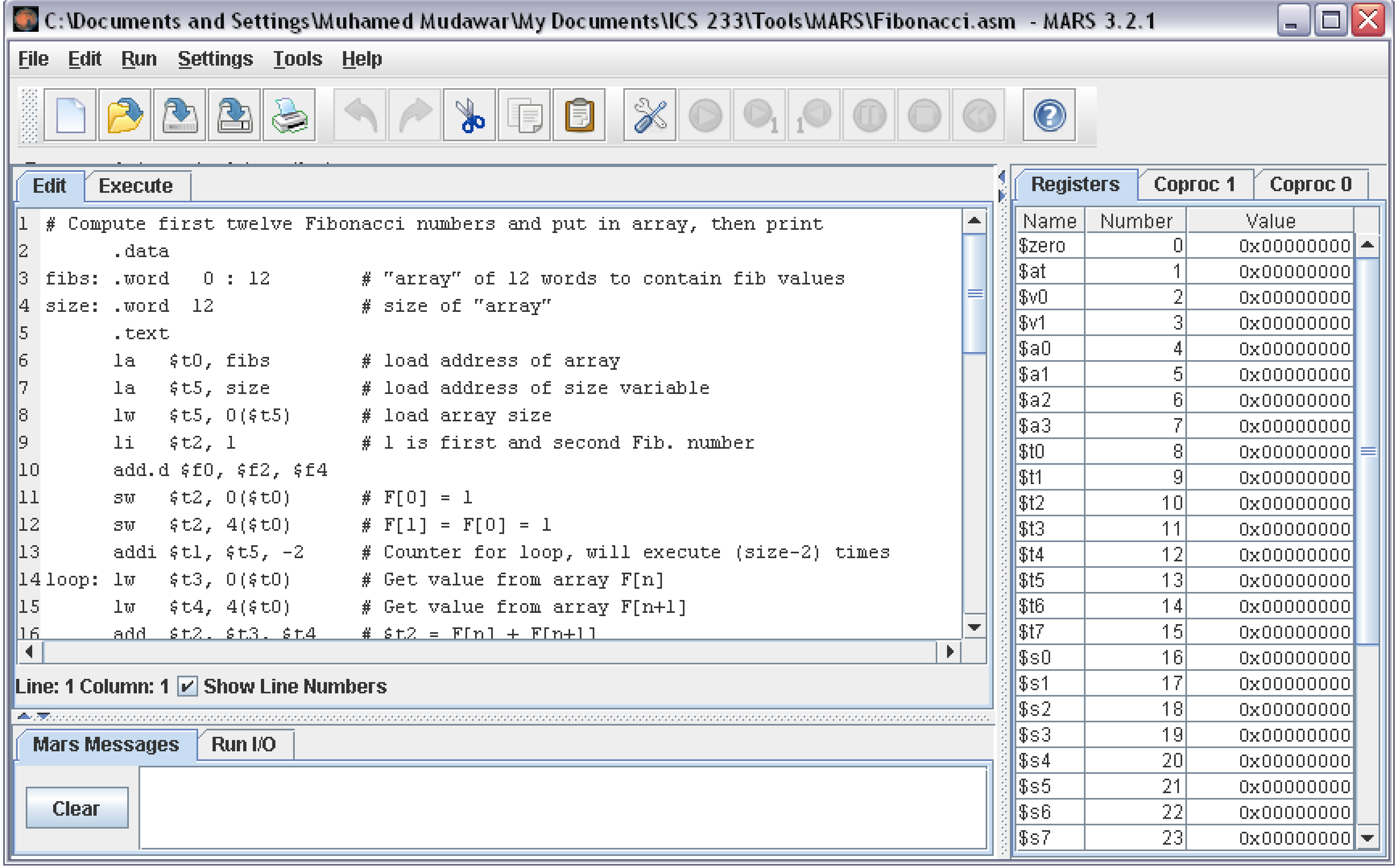


A project may consist of multiple source files

Assembler translates each source file separately into an object file

Linker links all object files together with link libraries

MARS Assembler and Simulator Tool



```
# Compute first twelve Fibonacci numbers and put in array, then print
      .data
fibs: .word   0 : 12        # "array" of 12 words to contain fib values
size: .word  12             # size of "array"
      .text
      la   $t0, fibs        # load address of array
      la   $t5, size        # load address of size variable
      lw   $t5, 0($t5)      # load array size
      li   $t2, 1           # 1 is first and second Fib. number
      add.d $f0, $f2, $f4
      sw   $t2, 0($t0)      # F[0] = 1
      sw   $t2, 4($t0)      # F[1] = F[0] = 1
      addi $t1, $t5, -2     # Counter for loop, will execute (size-2) times
loop: lw   $t3, 0($t0)      # Get value from array F[n]
      lw   $t4, 4($t0)      # Get value from array F[n+1]
      add  $t2, $t3, $t4    # $t2 = F[n] + F[n+1]
```

Line: 1 Column: 1 ☑ Show Line Numbers

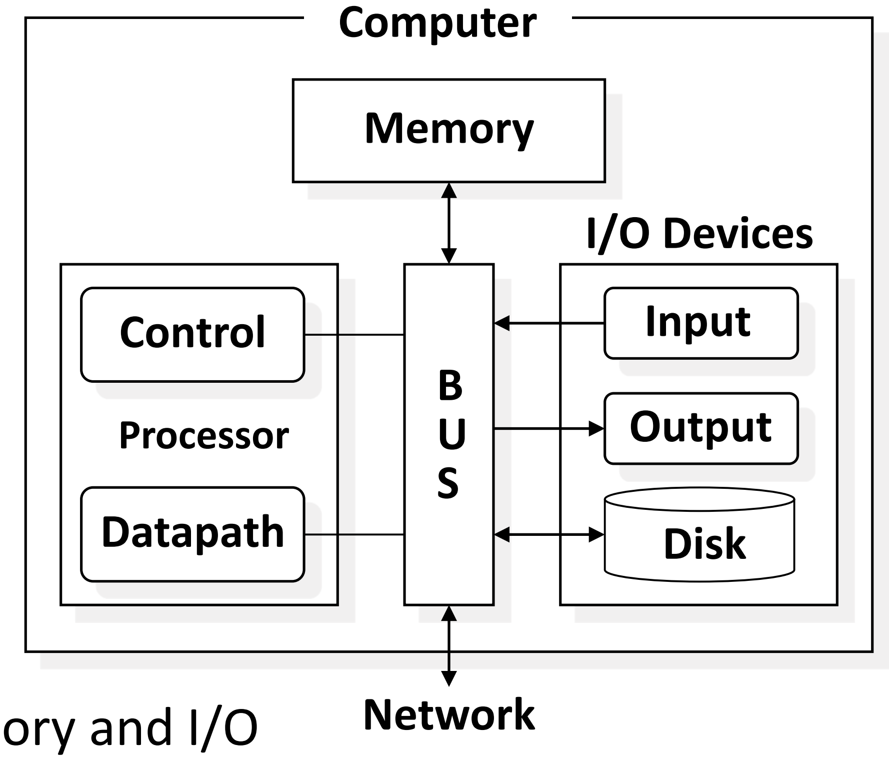
Name	Number	Value
$zero	0	0x00000000
$at	1	0x00000000
$v0	2	0x00000000
$v1	3	0x00000000
$a0	4	0x00000000
$a1	5	0x00000000
$a2	6	0x00000000
$a3	7	0x00000000
$t0	8	0x00000000
$t1	9	0x00000000
$t2	10	0x00000000
$t3	11	0x00000000
$t4	12	0x00000000
$t5	13	0x00000000
$t6	14	0x00000000
$t7	15	0x00000000
$s0	16	0x00000000
$s1	17	0x00000000
$s2	18	0x00000000
$s3	19	0x00000000
$s4	20	0x00000000
$s5	21	0x00000000
$s6	22	0x00000000
$s7	23	0x00000000

Next . . .

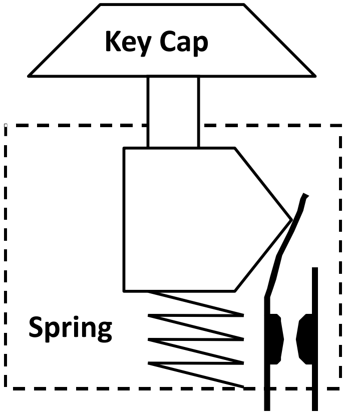
- Welcome to CSE 211
- High-Level, Assembly-, and Machine-Languages
- Components of a Computer System
- Chip Manufacturing Process
- Technology Improvements
- Programmer's View of a Computer System

Components of a Computer System

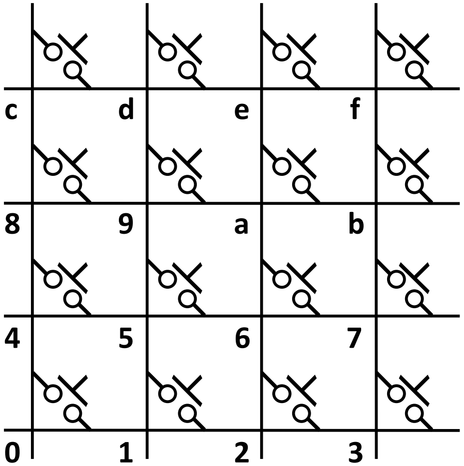
- Processor
  - Datapath
  - Control
- Memory & Storage
  - Main Memory
  - Disk Storage
- Input devices
- Output devices
- Bus: Interconnects processor to memory and I/O
- Network: newly added component for communication



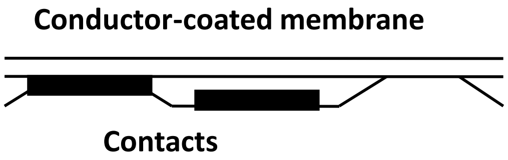
Input Devices



Mechanical switch

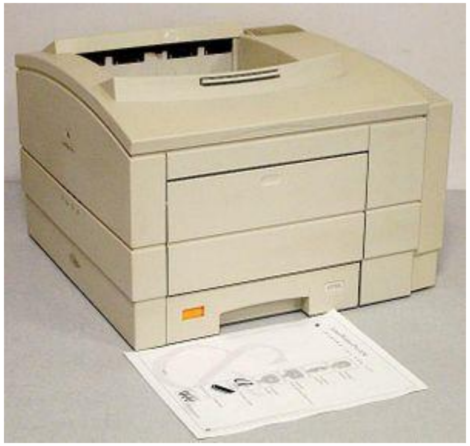


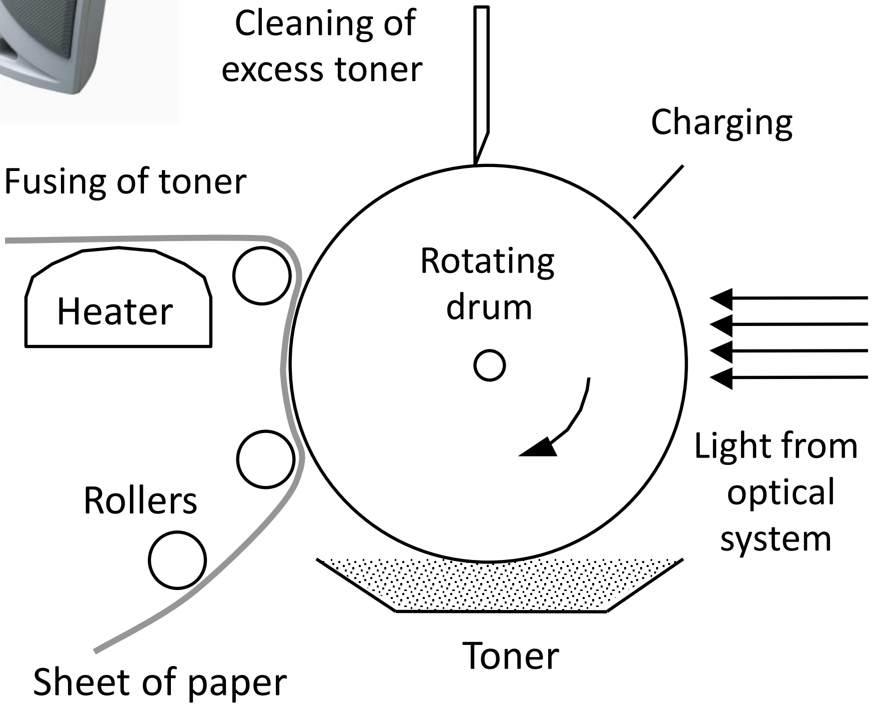
Logical arrangement of keys



Membrane switch

Output Devices



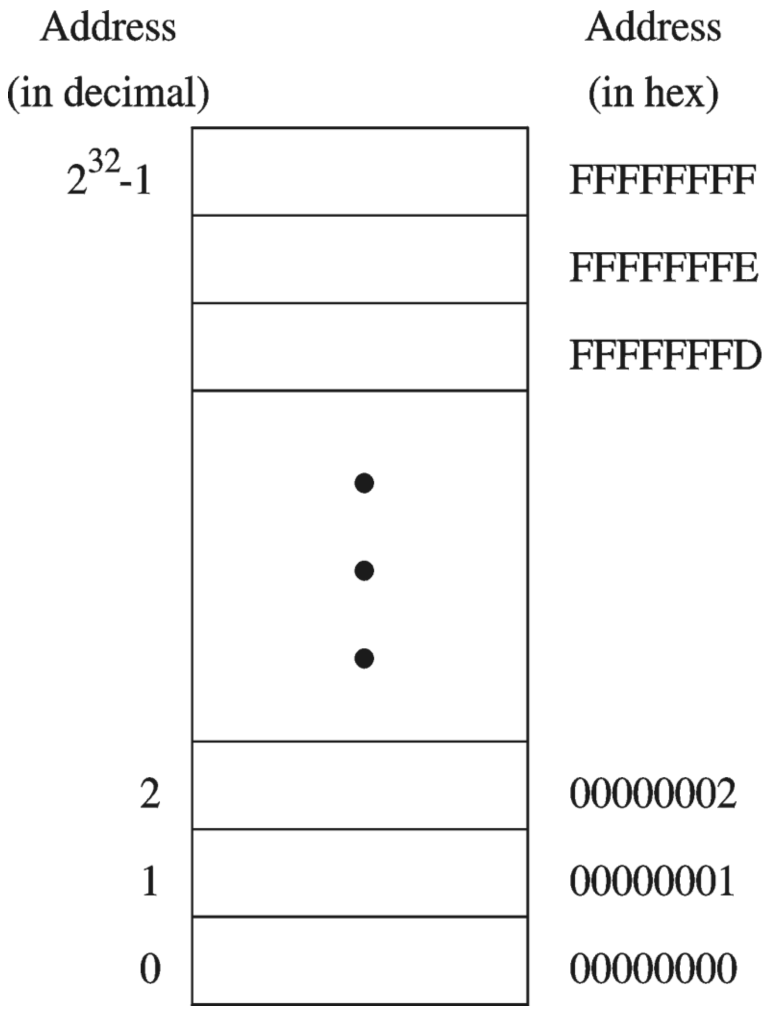


Laser printing

Memory

- Ordered sequence of bytes
  - The sequence number is called the memory address
- Byte addressable memory
  - Each byte has a unique address
  - Supported by almost all processors
- Physical address space
  - Determined by the address bus width
  - Pentium has a 32-bit address bus
    - Physical address space = **4GB = 2^{32} bytes**
  - Itanium with a 64-bit address bus can support
    - Up to **2^{64} bytes** of physical address space

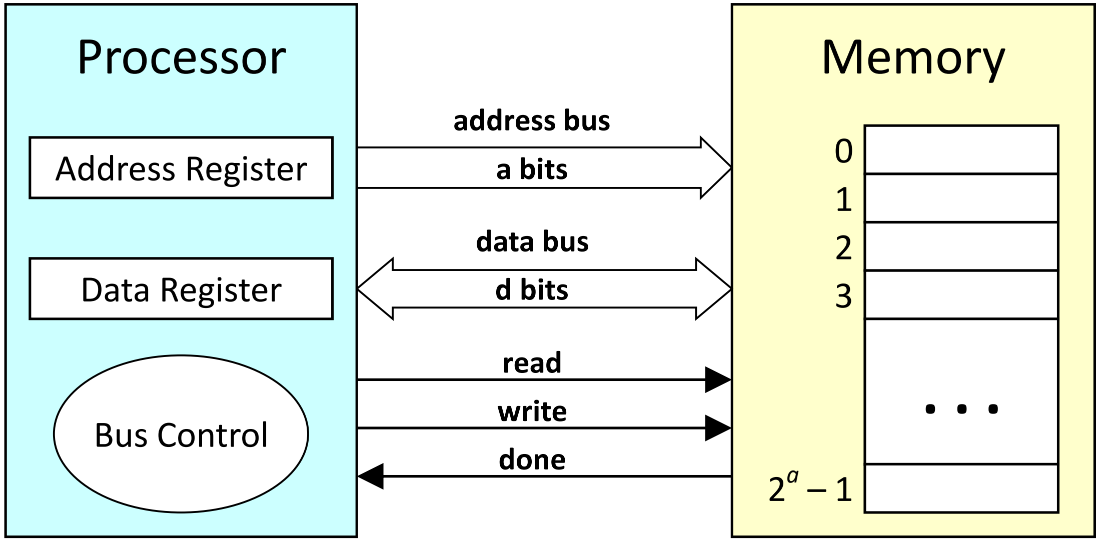
Address Space



Address Space is the set of memory locations (bytes) that can be addressed

Address, Data, and Control Bus

- Address Bus
  - Memory address is put on address bus
  - If memory address = *a* bits then 2^a locations are addressed
- Data Bus: bi-directional bus
  - Data can be transferred in both directions on the data bus
- Control Bus
  - Signals control transfer of data
  - Read request
  - Write request
  - Done transfer



Memory Devices

- Volatile Memory Devices
  - Data is lost when device is powered off
  - RAM = Random Access Memory
  - DRAM = Dynamic RAM
    - 1-Transistor cell + trench capacitor
    - Dense but slow, must be refreshed
    - Typical choice for main memory
  - SRAM: Static RAM
    - 6-Transistor cell, faster but less dense than DRAM
    - Typical choice for cache memory
- Non-Volatile Memory Devices
  - Stores information permanently
  - ROM = Read Only Memory
  - Used to store the information required to startup the computer
  - Many types: ROM, EPROM, EEPROM, and FLASH
  - FLASH memory can be erased electrically in blocks

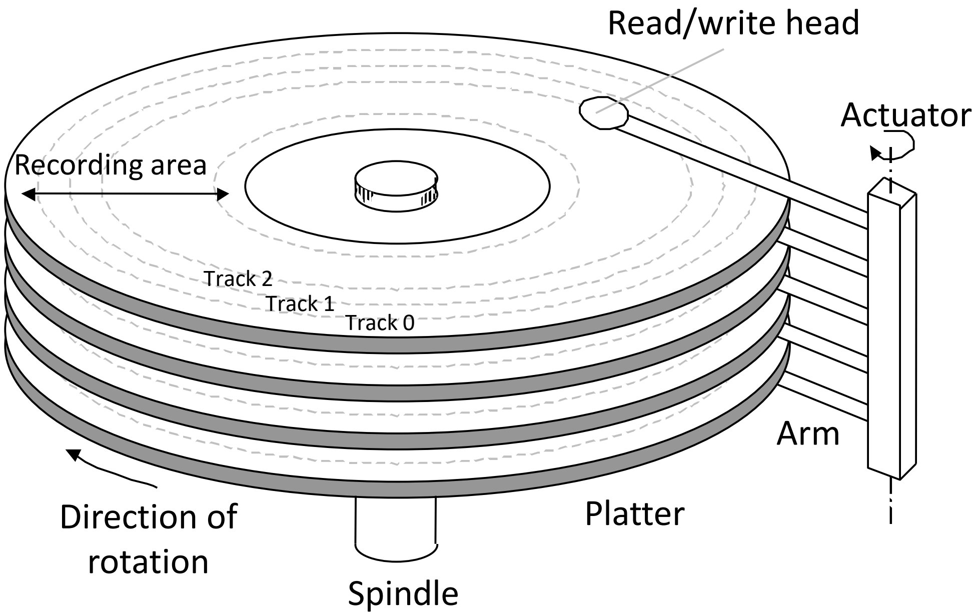
Magnetic Disk Storage

A Magnetic disk consists of a collection of platters

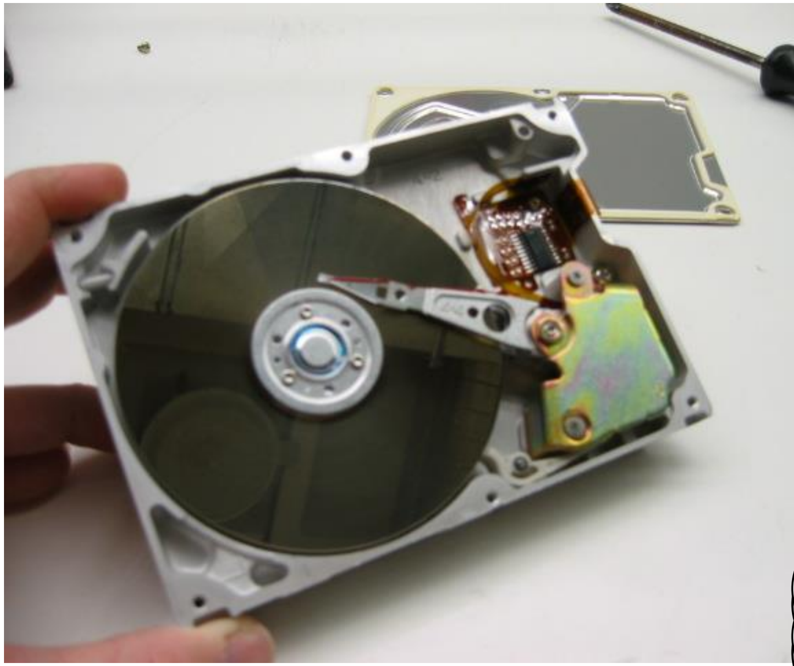
Provides a number of recording surfaces

Arm provides read/write heads for all surfaces

The disk heads are connected together and move in conjunction



Magnetic Disk Storage

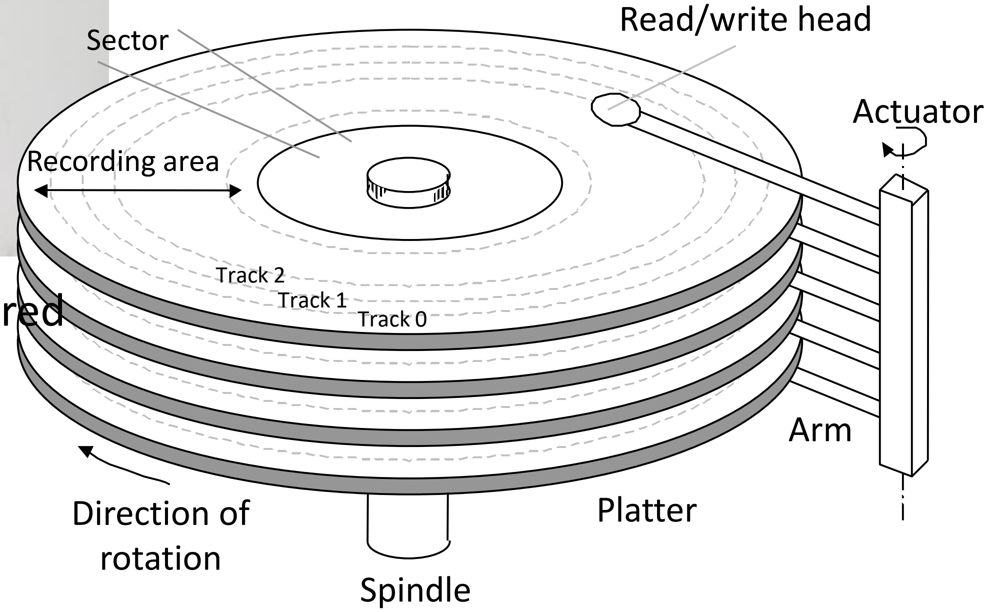


Disk Access Time =

Seek Time +

Rotation Latency +

Transfer Time



Seek Time: head movement to the desired track (milliseconds)

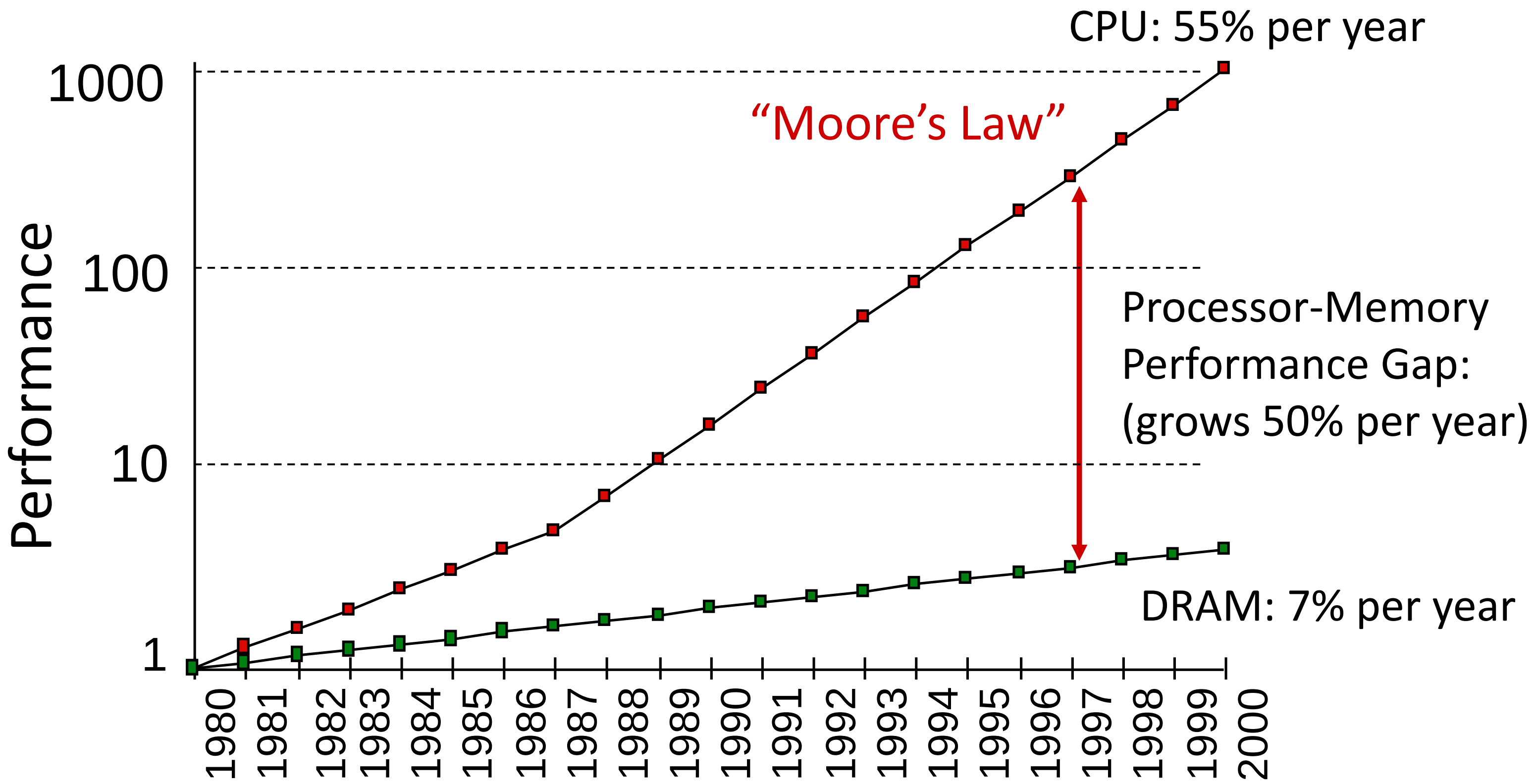
Rotation Latency: disk rotation until desired sector arrives under the head

Transfer Time: to transfer data

Example on Disk Access Time

- Given a magnetic disk with the following properties
  - Rotation speed = 7200 RPM (rotations per minute)
  - Average seek = 8 ms, Sector = 512 bytes, Track = 200 sectors
- Calculate
  - Time of one rotation (in milliseconds)
  - Average time to access a block of 32 consecutive sectors
- Answer
  - Rotations per second = 7200/60 = 120 RPS
  - Rotation time in milliseconds = 1000/120 = 8.33 ms
  - Average rotational latency = time of half rotation = 4.17 ms
  - Time to transfer 32 sectors = (32/200) * 8.33 = 1.33 ms
  - Average access time = 8 + 4.17 + 1.33 = 13.5 ms

Processor-Memory Performance Gap



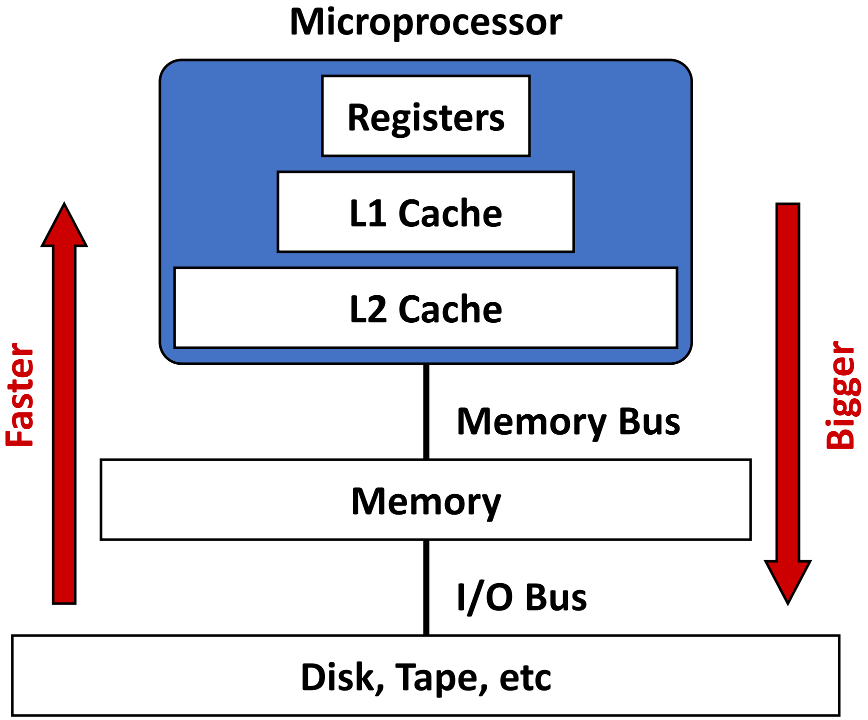
- ❖ 1980 – No cache in microprocessor
- ❖ 1995 – Two-level cache on microprocessor

The Need for a Memory Hierarchy

- Widening speed gap between CPU and main memory
  - Processor operation takes less than 1 ns
  - Main memory requires more than 50 ns to access
- Each instruction involves at least one memory access
  - One memory access to fetch the instruction
  - A second memory access for load and store instructions
- Memory bandwidth limits the instruction execution rate
- Cache memory can help bridge the CPU-memory gap
- Cache memory is small in size but fast

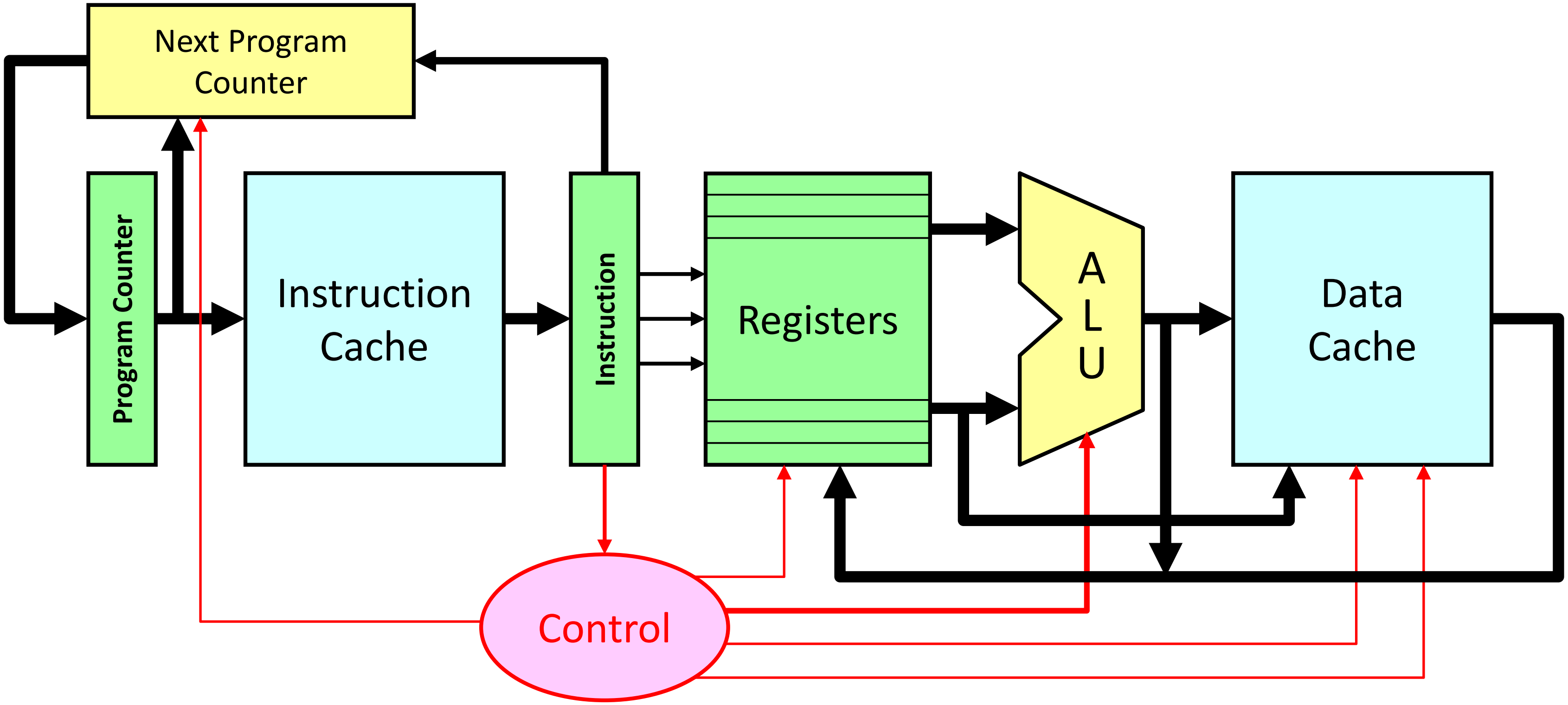
Typical Memory Hierarchy

- Registers are at the top of the hierarchy
  - Typical size < 1 KB
  - Access time < 0.5 ns
- Level 1 Cache (8 – 64 KB)
  - Access time: 0.5 – 1 ns
- L2 Cache (512KB – 8MB)
  - Access time: 2 – 10 ns
- Main Memory (1 – 2 GB)
  - Access time: 50 – 70 ns
- Disk Storage (> 200 GB)
  - Access time: milliseconds



Processor

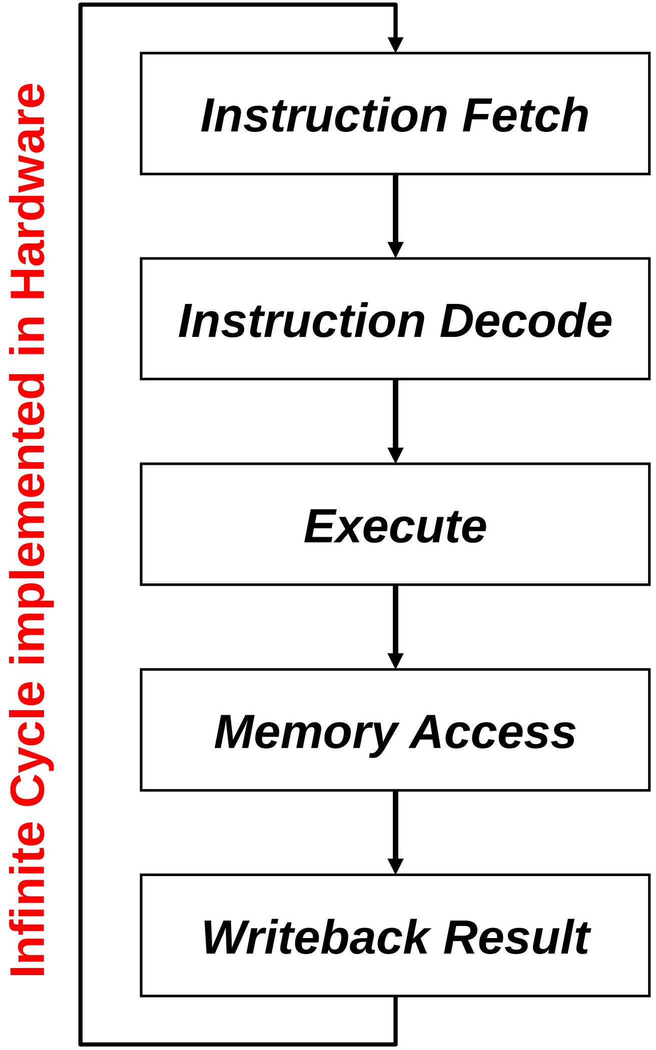
- Datapath: part of a processor that executes instructions
- Control: generates control signals for each instruction



Datapath Components

- Program Counter (PC)
  - Contains address of instruction to be fetched
  - Next Program Counter: computes address of next instruction
- Instruction Register (IR)
  - Stores the fetched instruction
- Instruction and Data Caches
  - Small and fast memory containing most recent instructions/data
- Register File
  - General-purpose registers used for intermediate computations
- ALU = Arithmetic and Logic Unit
  - Executes arithmetic and logic instructions
- Buses
  - Used to wire and interconnect the various components

Fetch - Execute Cycle



Infinite Cycle implemented in Hardware

Instruction Fetch

Fetch instruction

Compute address of next instruction

Instruction Decode

Generate control signals for instruction

Read operands from registers

Execute

Compute result value

Memory Access

Read or write memory (load/store)

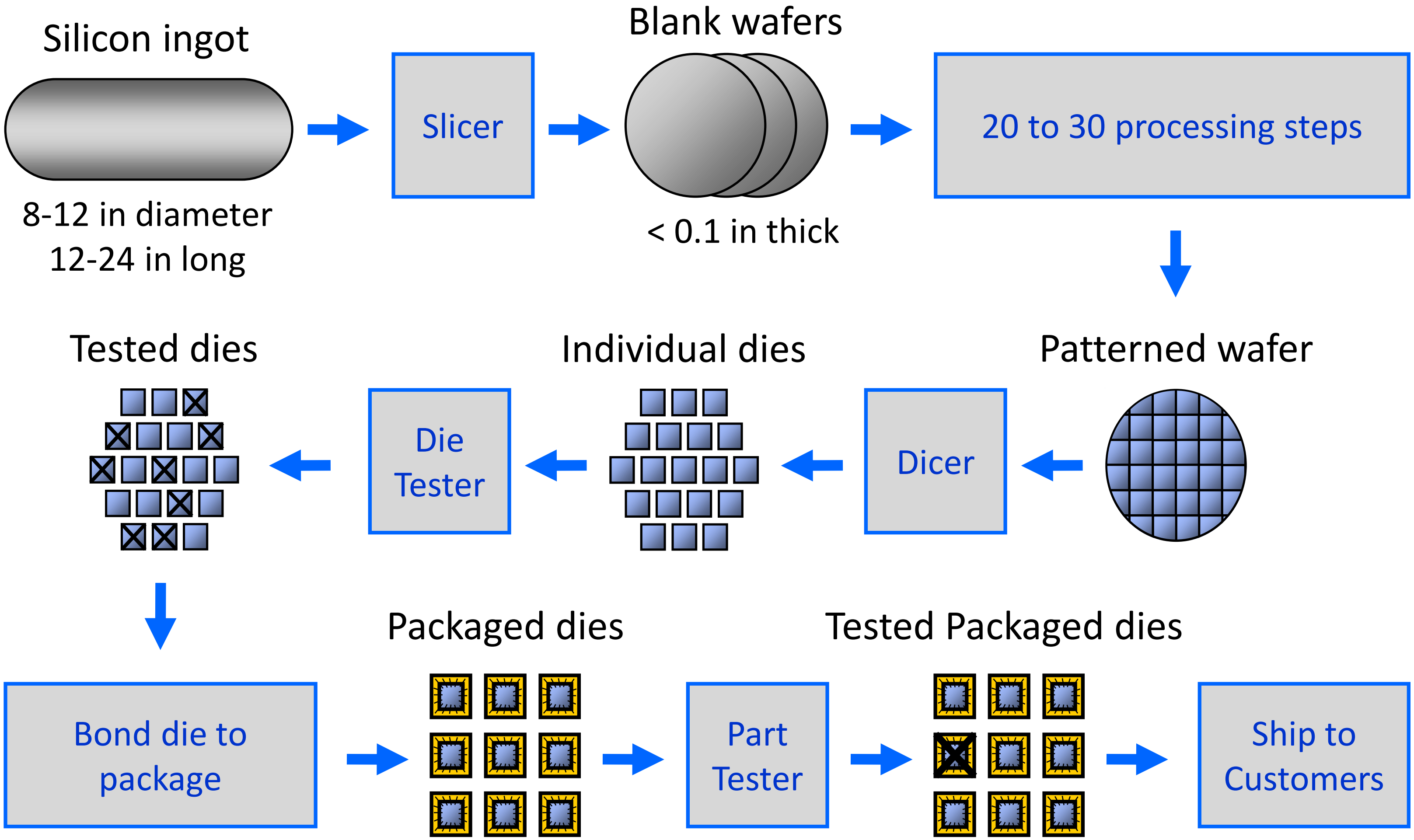
Writeback Result

Writeback result in a register

Next . . .

- Welcome to CSE 211
- Assembly-, Machine-, and High-Level Languages
- Components of a Computer System
- Chip Manufacturing Process
- Technology Improvements
- Programmer's View of a Computer System

Chip Manufacturing Process



Silicon ingot

8-12 in diameter
12-24 in long

Slicer

Blank wafers

< 0.1 in thick

20 to 30 processing steps

Patterned wafer

Dicer

Individual dies

Die Tester

Tested dies

Bond die to package

Packaged dies

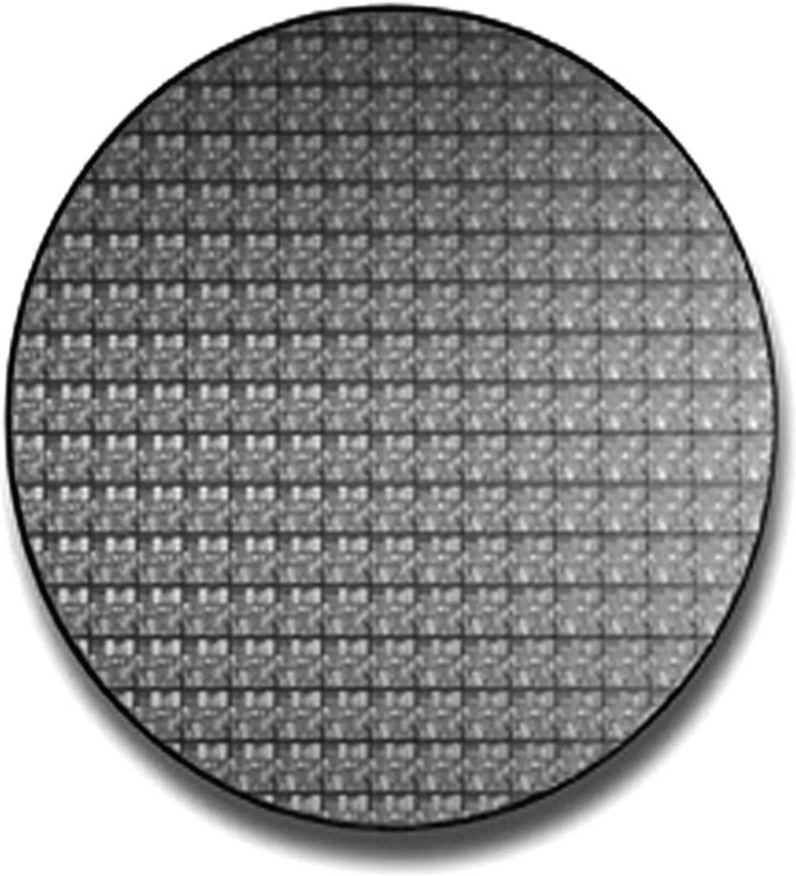
Part Tester

Tested Packaged dies

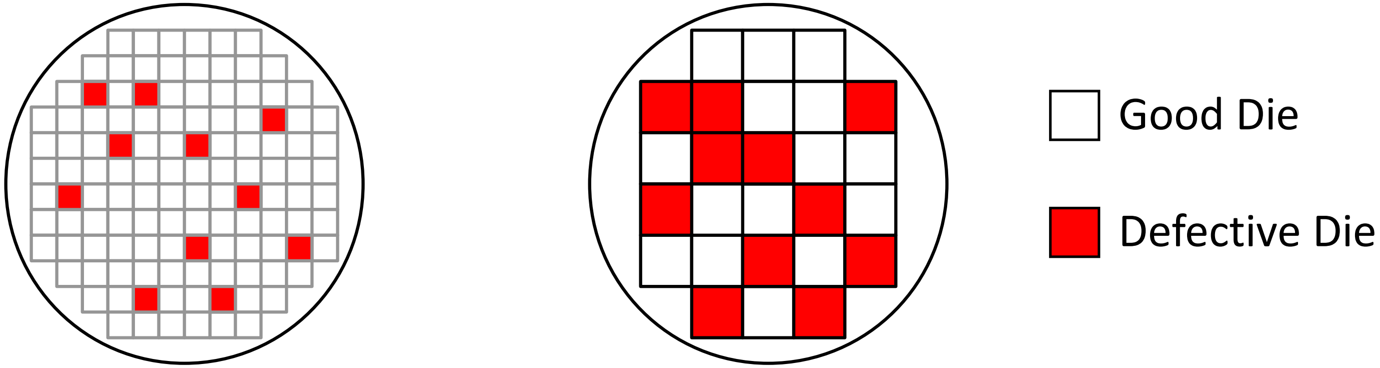
Ship to Customers

Wafer of Pentium 4 Processors

- 8 inches (20 cm) in diameter
- Die area is 250 mm^2
  - About 16 mm per side
- 55 million transistors per die
  - 0.18 µm technology
  - Size of smallest transistor
  - Improved technology uses
    - 0.13 µm and 0.09 µm
- Dies per wafer = 169
  - When yield = 100%
  - Number is reduced after testing
  - Rounded dies at boundary are useless



Effect of Die Size on Yield



120 dies, 109 good

26 dies, 15 good

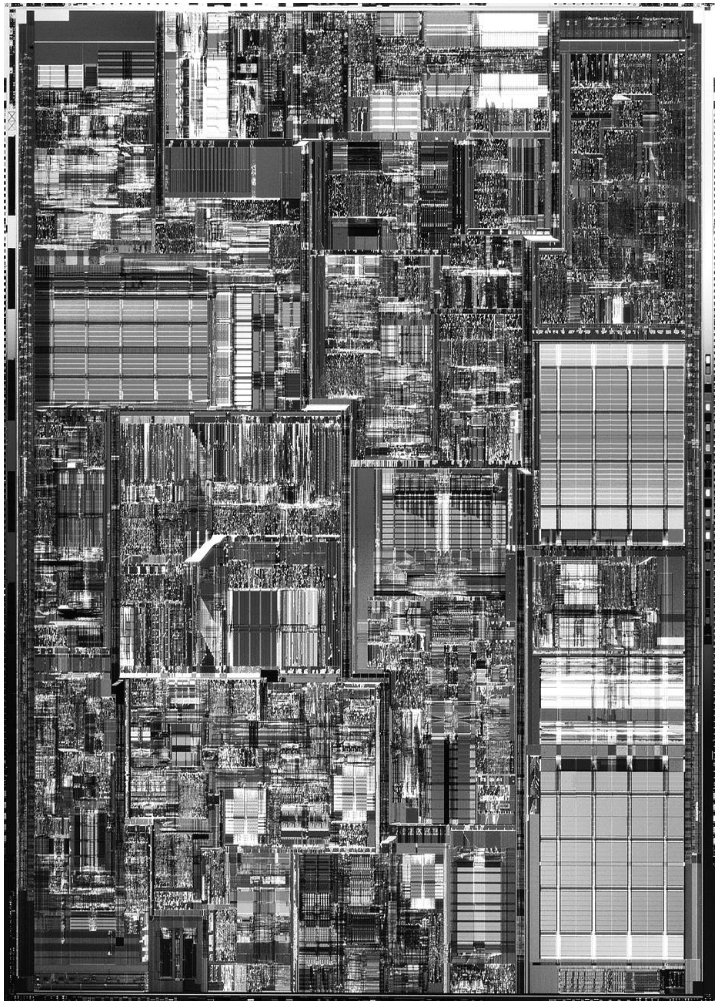
Dramatic decrease in yield with larger dies

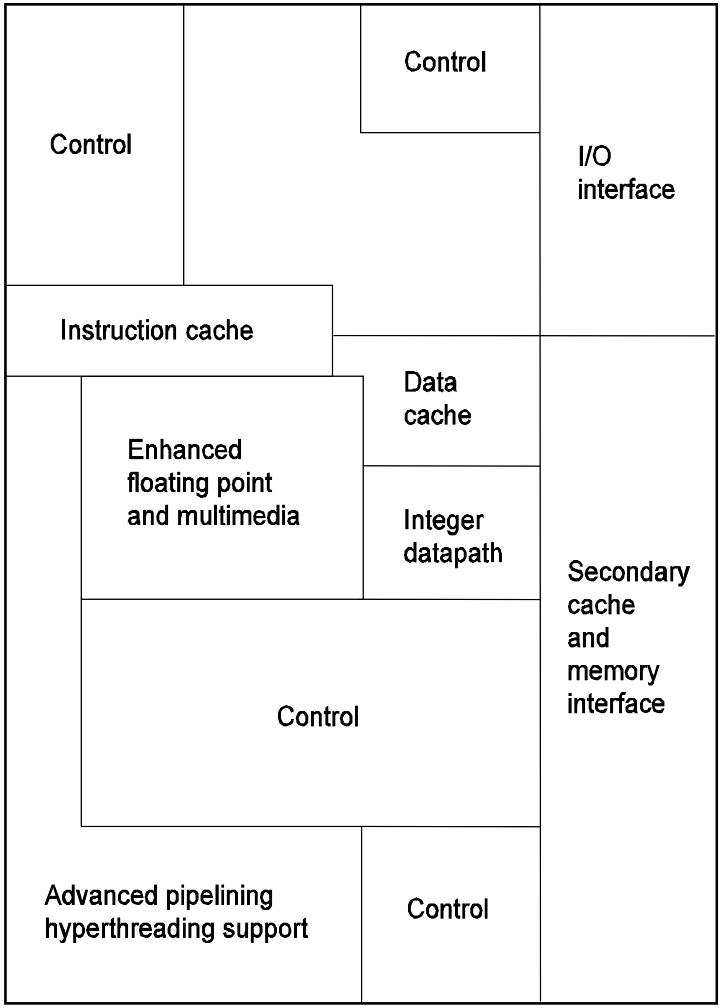
Yield = (Number of Good Dies) / (Total Number of Dies)

$$\text{Yield} = \frac{1}{(1 + (\text{Defect per area} \times \text{Die area} / 2))^2}$$

Die Cost = (Wafer Cost) / (Dies per Wafer × Yield)

Inside the Pentium 4 Processor Chip





Control

Control

I/O
interface

Instruction cache

Data
cache

Enhanced
floating point
and multimedia

Integer
datapath

Secondary
cache
and
memory
interface

Control

Advanced pipelining
hyperthreading support

Control

Next . . .

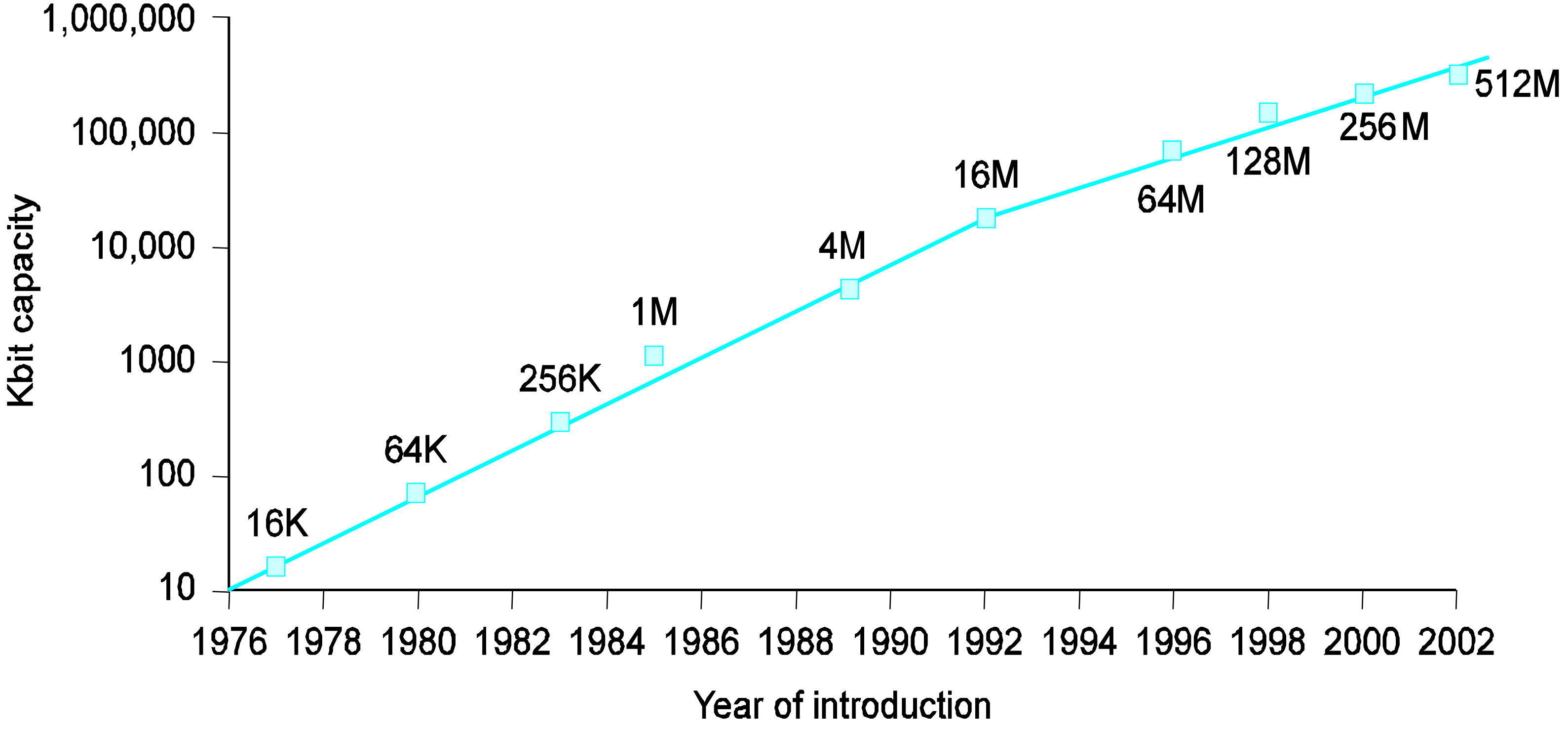
- Welcome to ICS 233
- Assembly-, Machine-, and High-Level Languages
- Components of a Computer System
- Chip Manufacturing Process
- Technology Improvements
- Programmer's View of a Computer System

Technology Improvements

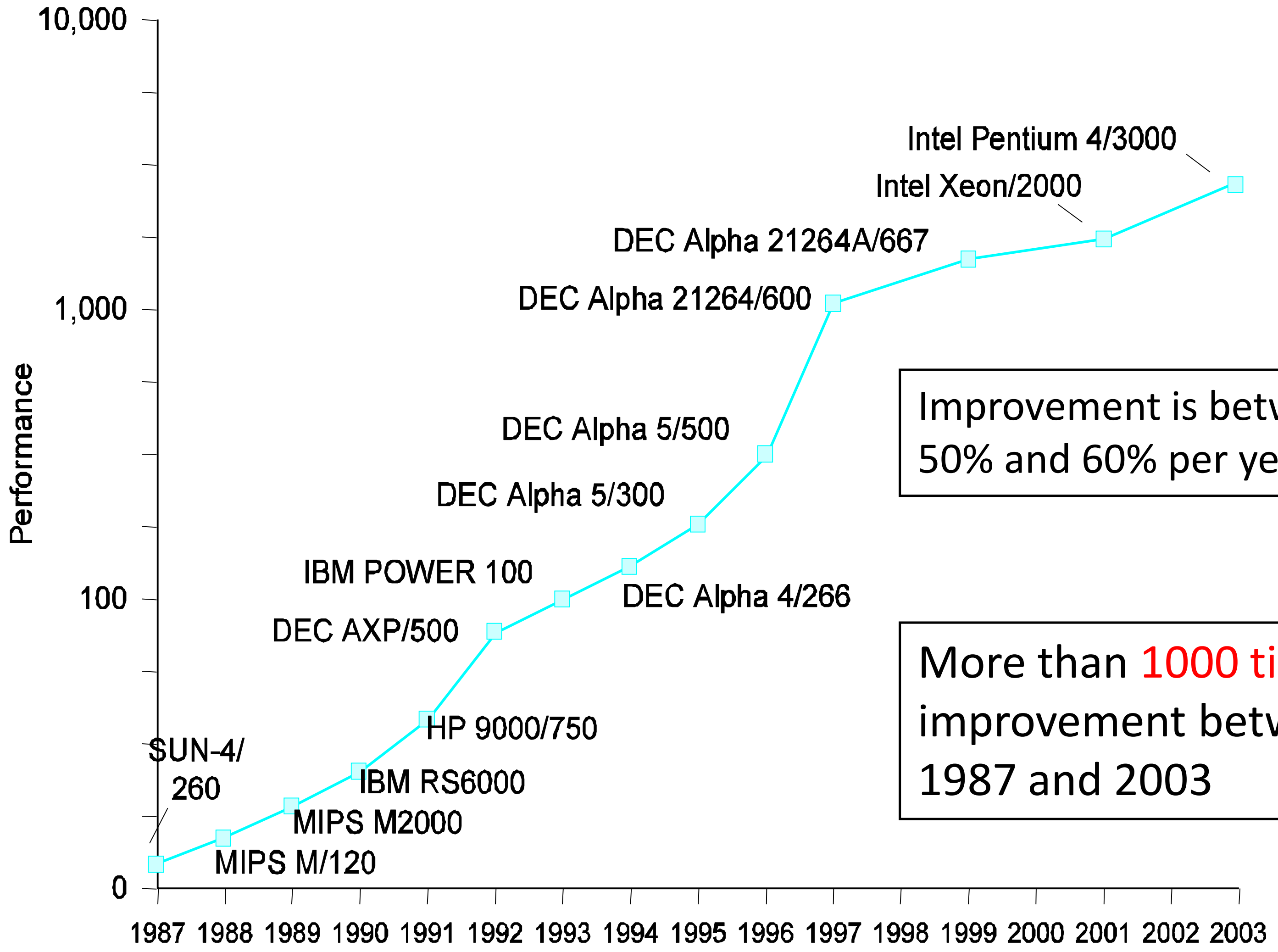
- Vacuum tube → transistor → IC → VLSI
- Processor
    - Transistor count: about 30% to 40% per year
- Memory
    - DRAM capacity: about 60% per year (4x every 3 yrs)
    - Cost per bit: decreases about 25% per year
- Disk
    - Capacity: about 60% per year
- Opportunities for new applications
- Better organizations and designs

Growth of Capacity per DRAM Chip

- DRAM capacity quadrupled almost every 3 years
  - 60% increase per year, for 20 years



Workstation Performance



Improvement is between 50% and 60% per year

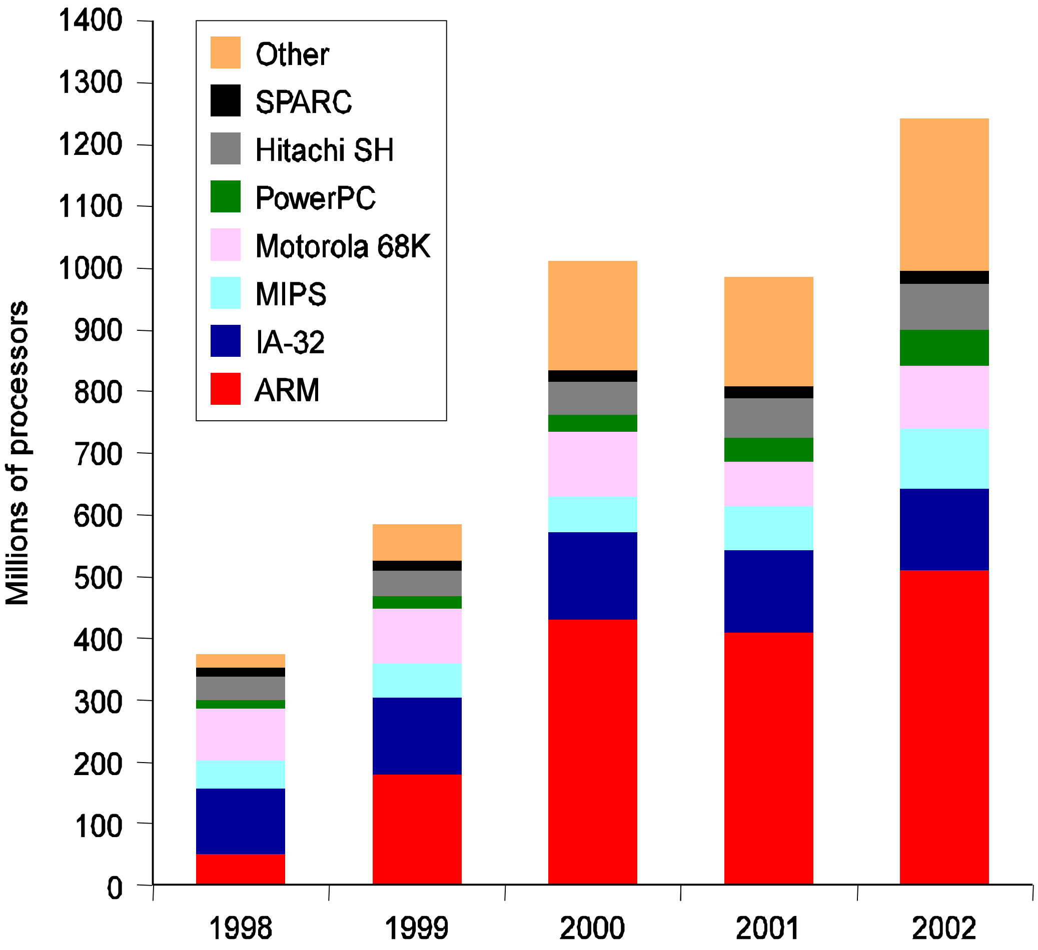
More than 1000 times improvement between 1987 and 2003

Microprocessor Sales (1998 – 2002)

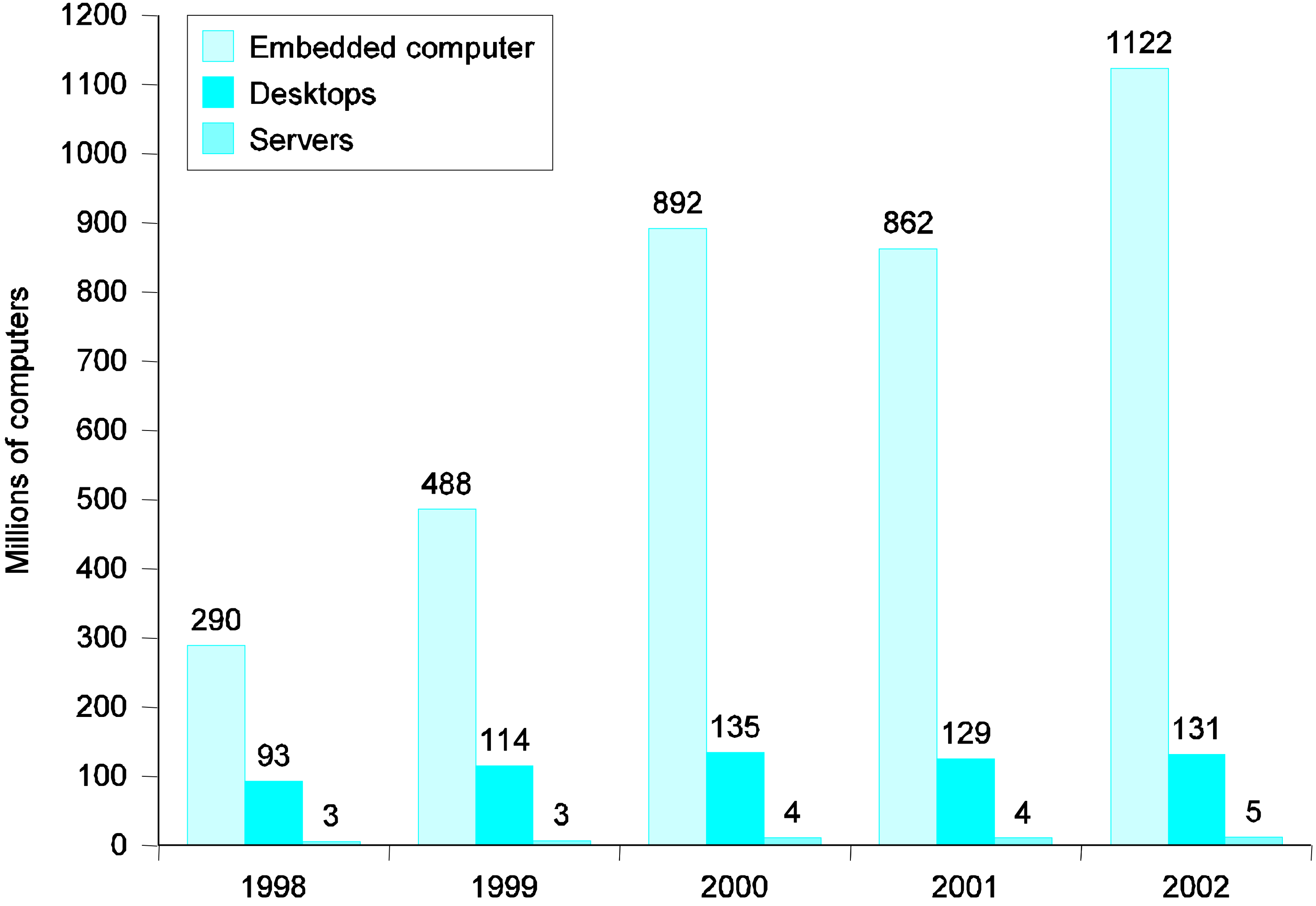
- ARM processor sales exceeded Intel IA-32 processors, which came second

- ARM processors are used mostly in cellular phones

- Most processors today are embedded in cell phones, video games, digital TVs, PDAs, and a variety of consumer devices



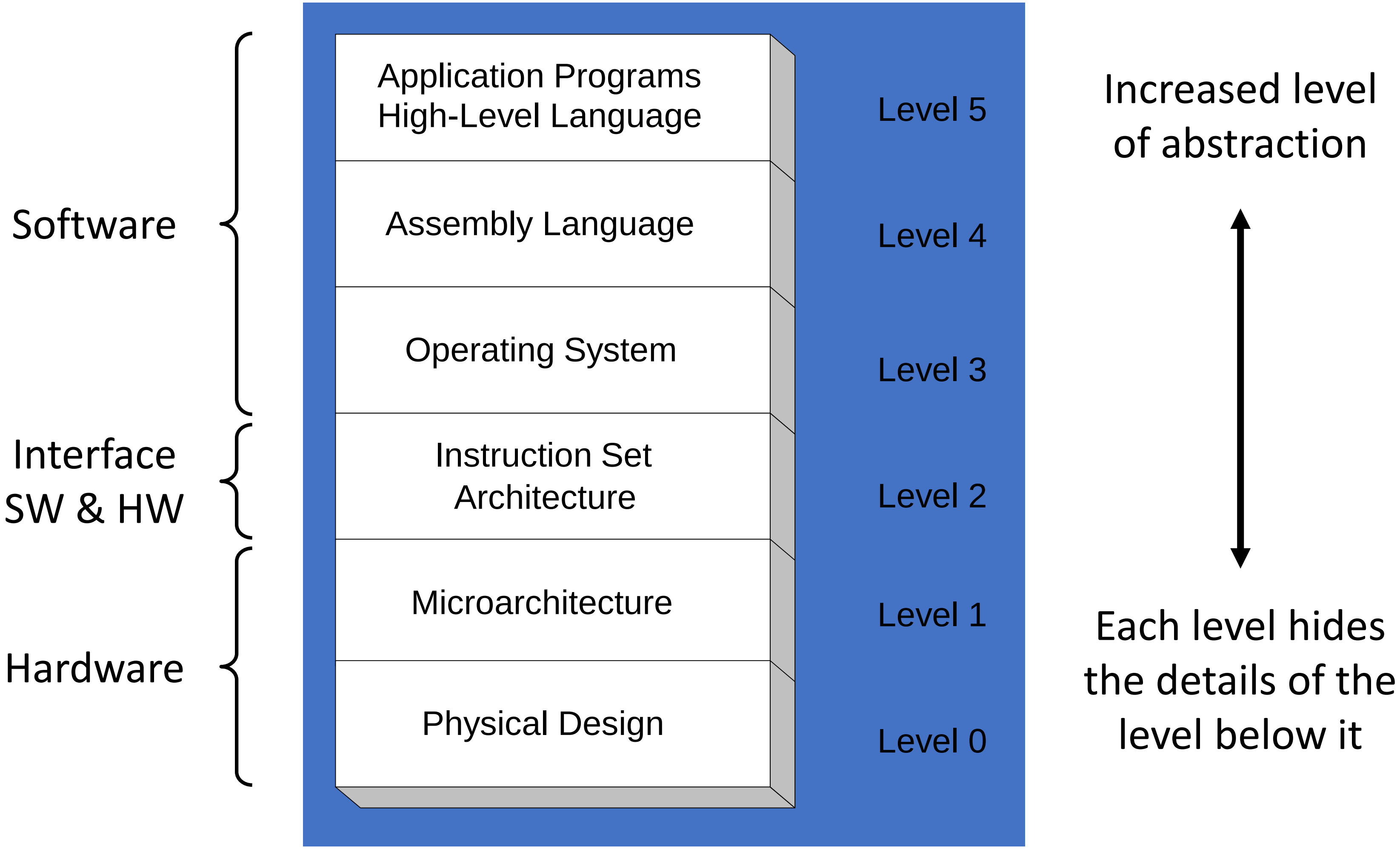
Microprocessor Sales – cont'd



Next . . .

- Welcome to ICS 233
- Assembly-, Machine-, and High-Level Languages
- Components of a Computer System
- Chip Manufacturing Process
- Technology Improvements
- Programmer's View of a Computer System

Programmer’s View of a Computer System



Software

Interface
SW & HW

Hardware

Application Programs
High-Level Language

Level 5

Assembly Language

Level 4

Operating System

Level 3

Instruction Set
Architecture

Level 2

Microarchitecture

Level 1

Physical Design

Level 0

Increased level
of abstraction

Each level hides
the details of the
level below it

Programmer's View – 2

- Application Programs (Level 5)
  - Written in high-level programming languages
  - Such as Java, C++, Pascal, Visual Basic . . .
  - Programs compile into assembly language level (Level 4)
- Assembly Language (Level 4)
  - Instruction mnemonics are used
  - Have one-to-one correspondence to machine language
  - Calls functions written at the operating system level (Level 3)
  - Programs are translated into machine language (Level 2)
- Operating System (Level 3)
  - Provides services to level 4 and 5 programs
  - Translated to run at the machine instruction level (Level 2)

Programmer's View – 3

- Instruction Set Architecture (Level 2)
  - Interface between software and hardware
  - Specifies how a processor functions
  - Machine instructions, registers, and memory are exposed
  - Machine language is executed by Level 1 (microarchitecture)
- Microarchitecture (Level 1)
  - Controls the execution of machine instructions (Level 2)
  - Implemented by digital logic
- Physical Design (Level 0)
  - Implements the microarchitecture
  - Physical layout of circuits on a chip

Course Roadmap

- Instruction set architecture (Chapter 2)
- MIPS Assembly Language Programming (Chapter 2)
- Computer arithmetic (Chapter 3)
- Performance issues (Chapter 4)
- Constructing a processor (Chapter 5)
- Pipelining to improve performance (Chapter 6)
- Memory and caches (Chapter 7)

Key to obtain a good grade: read the textbook!