TEXAS TECH UNIVERSITY
Department *of* Computer Science

CS5375 Computer Systems Organization and Architecture

Lecture 2

Instructor: Yong Chen, Ph.D.
Department of Computer Science
Texas Tech University
Yong.Chen@ttu.edu, 806-834-0284

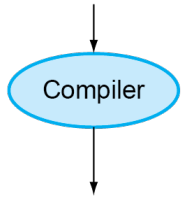
Review of Last Lecture

Level of abstraction closer to problem domain
Provides for productivity and portability

- Introduction and Classes of Computers
- How Software Works: Below Your Program
  - Application software
    - Written in high-level language
    - Go through layers of translation to instructions
  - System software
    - Compiler, Operating System (service code)
  - Hardware
    - Processor, memory, I/O controllers

High-level language program (in C)

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

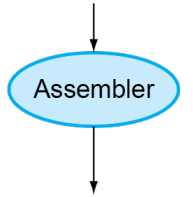


A program that translates HLL statements into assembly language statements.

Assembly language program (for RISC-V)

```
swap:
        slli x6, x11, 3
        add  x6, x10, x6
        ld   x5, 0(x6)
        ld   x7, 8(x6)
        sd   x7, 0(x6)
        sd   x5, 8(x6)
        jalr x0, 0(x1)
```

Textual representation of instructions (machine language)



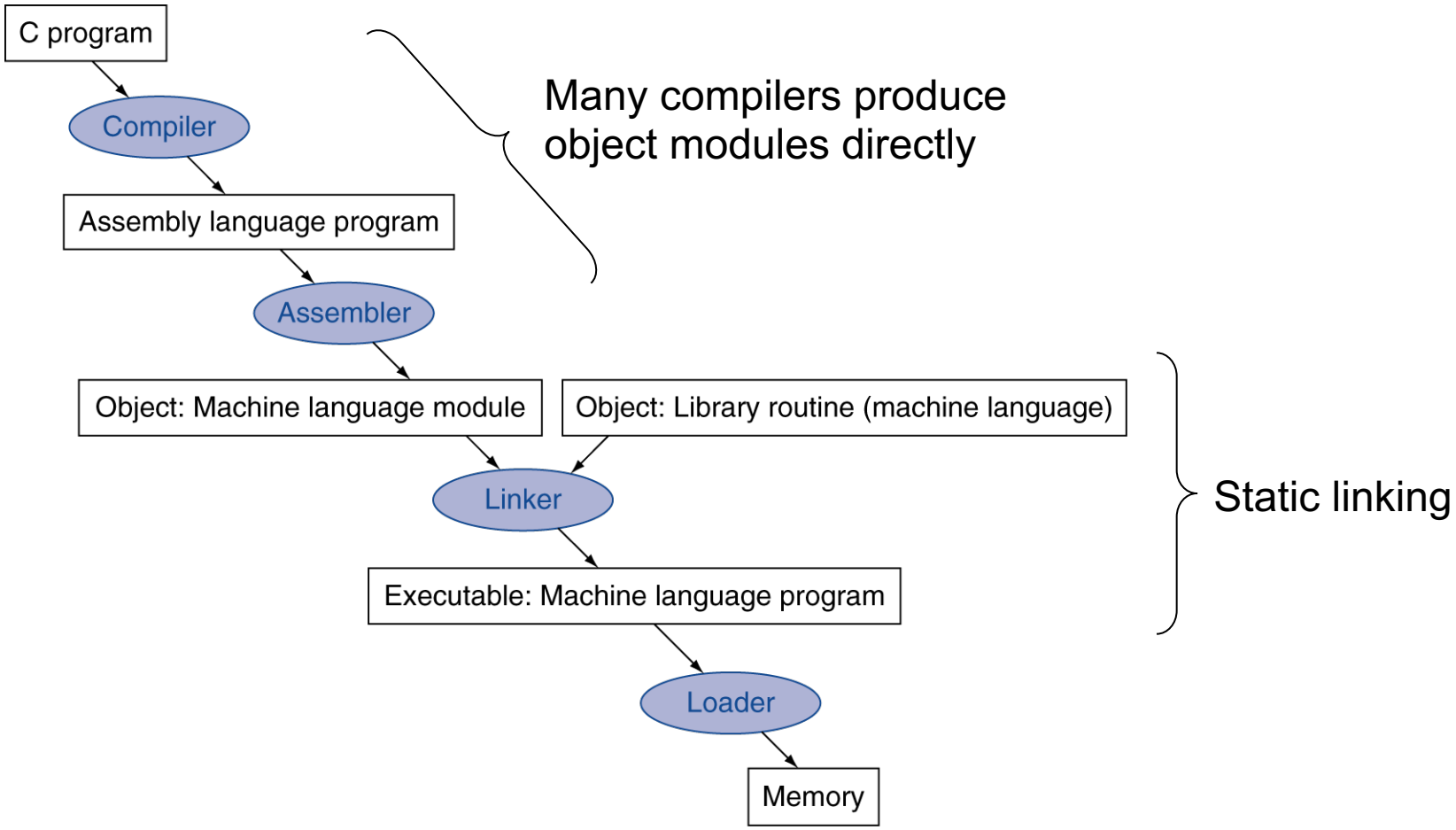
A program that translates assembly language (symbolic representation of instructions) into the binary version.

Binary machine language program (for RISC-V)

```
00000000001101011001001100010011
00000000011001010000001100110011
00000000000000011001001010000011
00000000100000011001001110000011
00000000011100110011000000100011
00000000010100110011010000100011
00000000000000001000000001100111
```

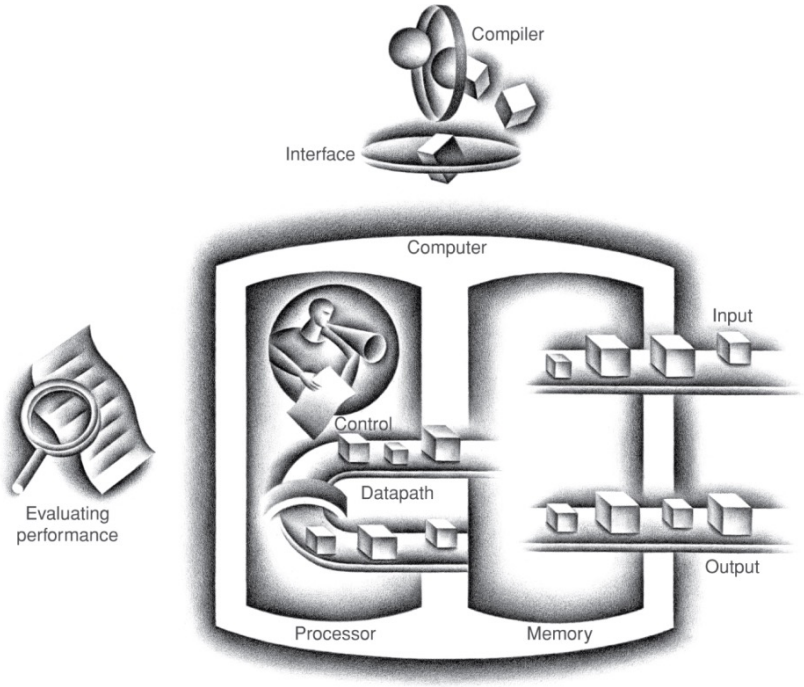
Binary digits (bits)
Encoded instructions and data

How Software Works: Translation and Startup



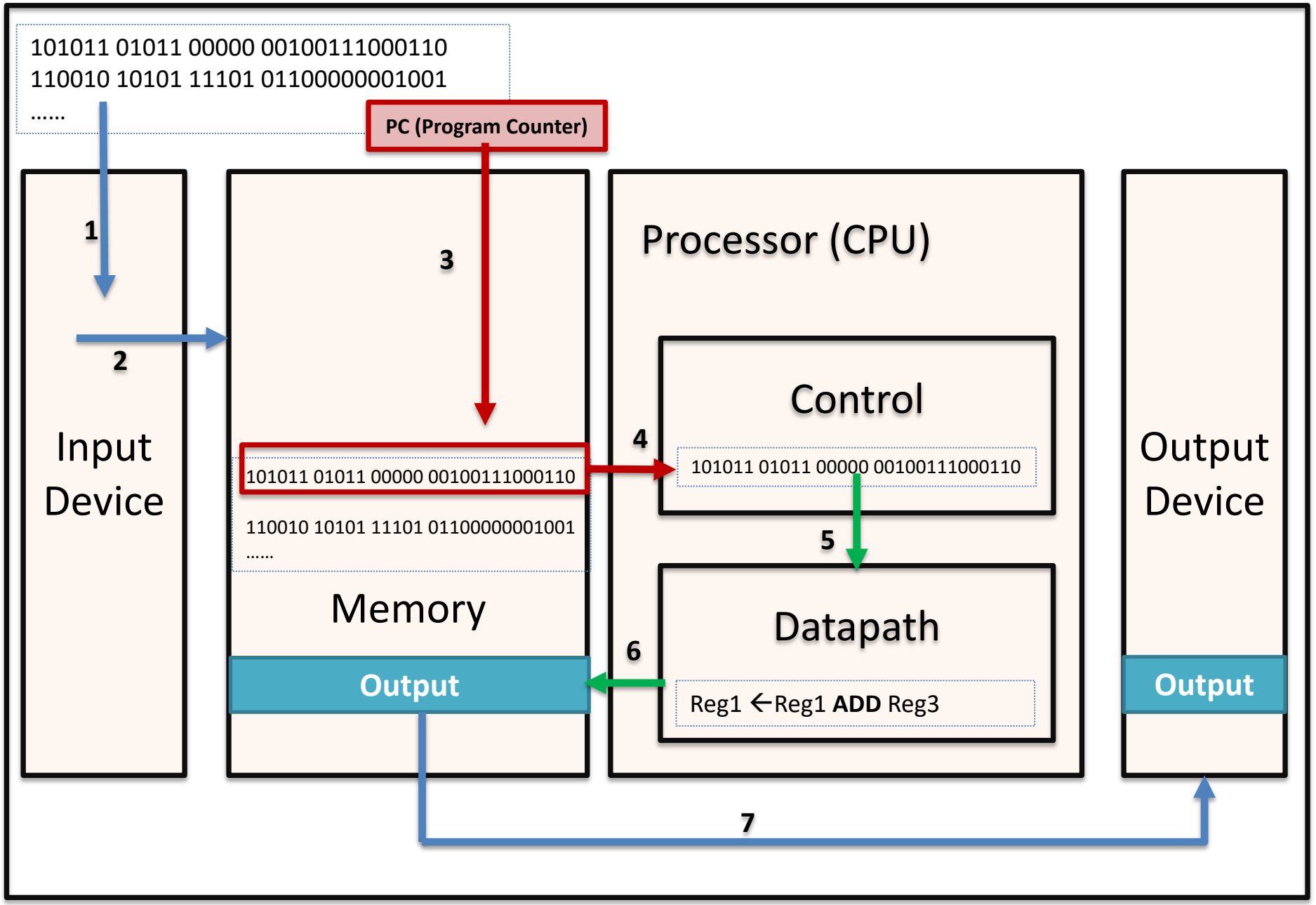
How Hardware Works: Components of a Computer

- Same components for all kinds of computer
  - Desktop, server, embedded
- Five classic components
  - Input, output, memory, datapath (processor), and control (processor)
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers



The organization of a computer, showing the five classic components. The processor gets instructions and data from memory. Input writes data to memory, and output reads data from memory. Control sends the signals that determine the operations of the datapath, memory, input, and output.

Computer (Software and Hardware Together)



Outline

- Trends in Architecture
- Defining Computer Architecture
- Trends in Technology

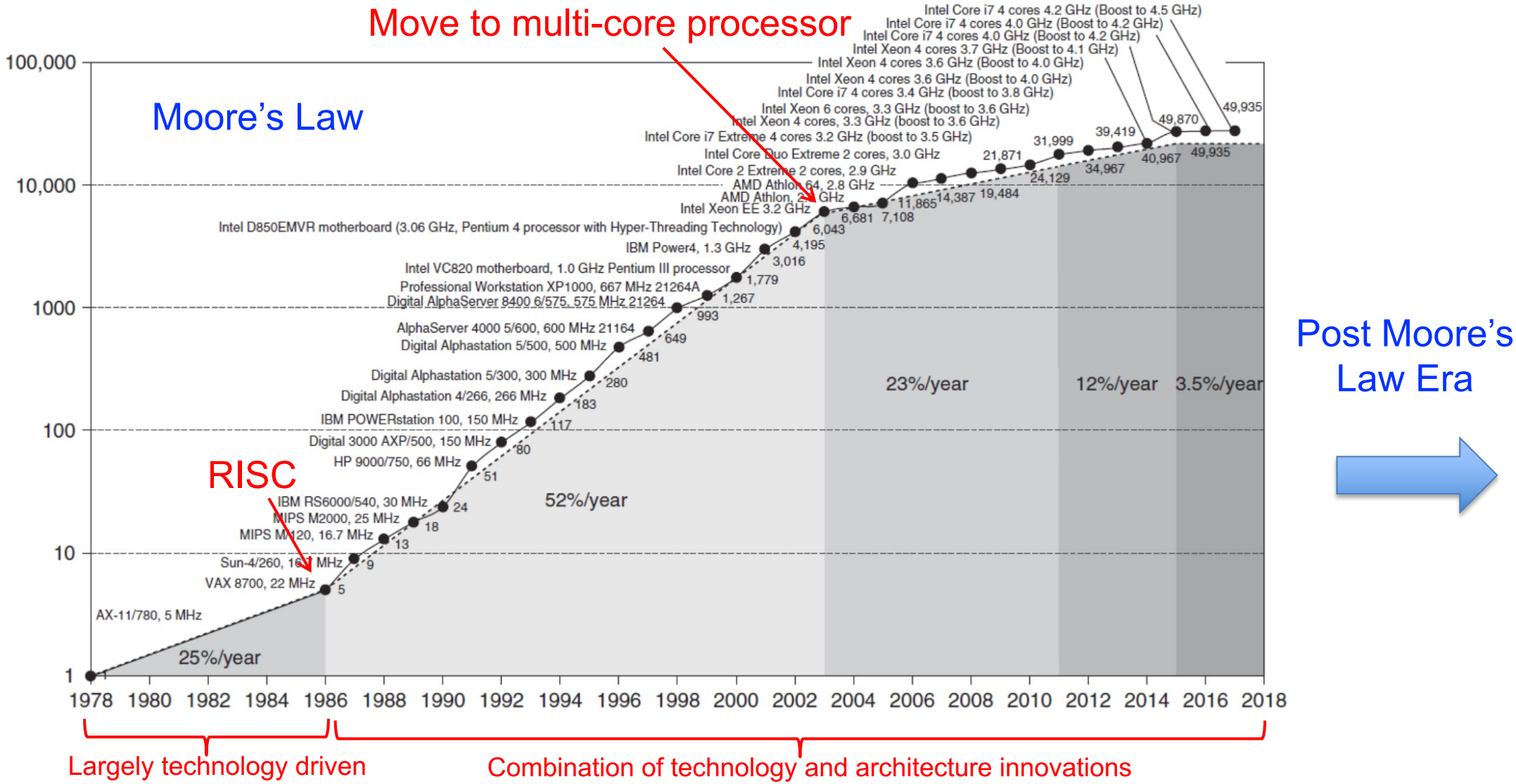
Computer Technology Improvements

- Performance improvements:
  - Improvements in semiconductor technology
    - Feature size, clock speed
    - 14nm scale in 2014, 10nm scale in 2017, 5nm in 2020, 2nm in 2024
  - Improvements in computer architectures
    - Enabled by HLL (high-level language) compilers, UNIX/Linux
    - Standardized, vendor-independent operating systems lowered the cost and risk of bringing out a new architecture
    - Lead to RISC (Reduced Instruction Set Computer) architectures with simpler instructions, v.s. CISC (Complex Instruction Set Computer)

Computer Technology Improvements (cont.)

- Performance improvements:
  - Together have enabled:
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages
    - SaaS, Virtualization, Cloud
  - Applications evolution:
    - Speech, sound, images, video, “augmented/extended reality”, “big data”

Growth in Processor Performance Over 40 Years



Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Single processor performance improvement ended in 2003
  - All major vendors moved to multi-core/many-core architecture
  - Pollack's rule
  - Free ride of performance improvements via semiconductor technology and computer architectures (sequential programming) is over!

- New models for performance (parallel programming):
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)

- These require explicit restructuring of the application
  - Parallel computing becomes universal

Parallelism

- Classes of parallelism in applications:
  - Data-Level Parallelism (DLP): identical operations operate on different data items concurrently to solve a problem

```
for (i=0; i<1000; i++)
    a[i]=b[i]+c[i];
```

  - Task-Level Parallelism (TLP): independent tasks (non-identical operations) operate on different data items concurrently to solve a problem

```
for (i=0; i<1000; i++)  /*block 1 */
    b[i+1]=b[i]+c[i]
…
for (j=0; j<5; j++)     /*block 2*/
    a[j+1]=a[j]+d[j];
```

Classes of Architectural Parallelism and Flynn's Taxonomy

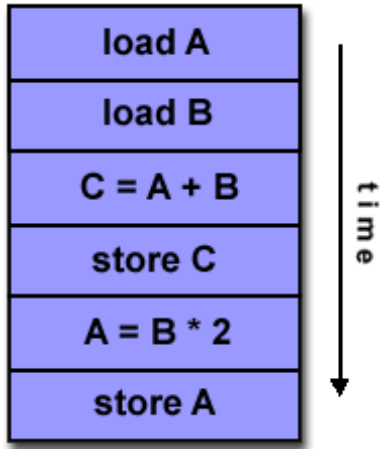
- Flynn's taxonomy distinguishes computer architectures according to how they can be classified along two independent dimensions of Instruction and Data
  - Each of these dimensions can have only one of two possible states: Single or Multiple.
  - The matrix below defines the 4 possible classifications according to Flynn:

SISD Single Instruction, Single Data	**SIMD** Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	**MIMD** Multiple Instruction, Multiple Data

Courtesy: Blaise Barney, LLNL

Single Instruction, Single Data (SISD)

- Model of serial Von Neumann machine
- Single Instruction: only one instruction stream is being executed by the CPU during any one clock cycle (single control processor)
- Single Data: only one data stream is being used as input during any one clock cycle
- This is the oldest and the most common type of computer till multicore era
- Examples: older generation mainframes, minicomputers and workstations

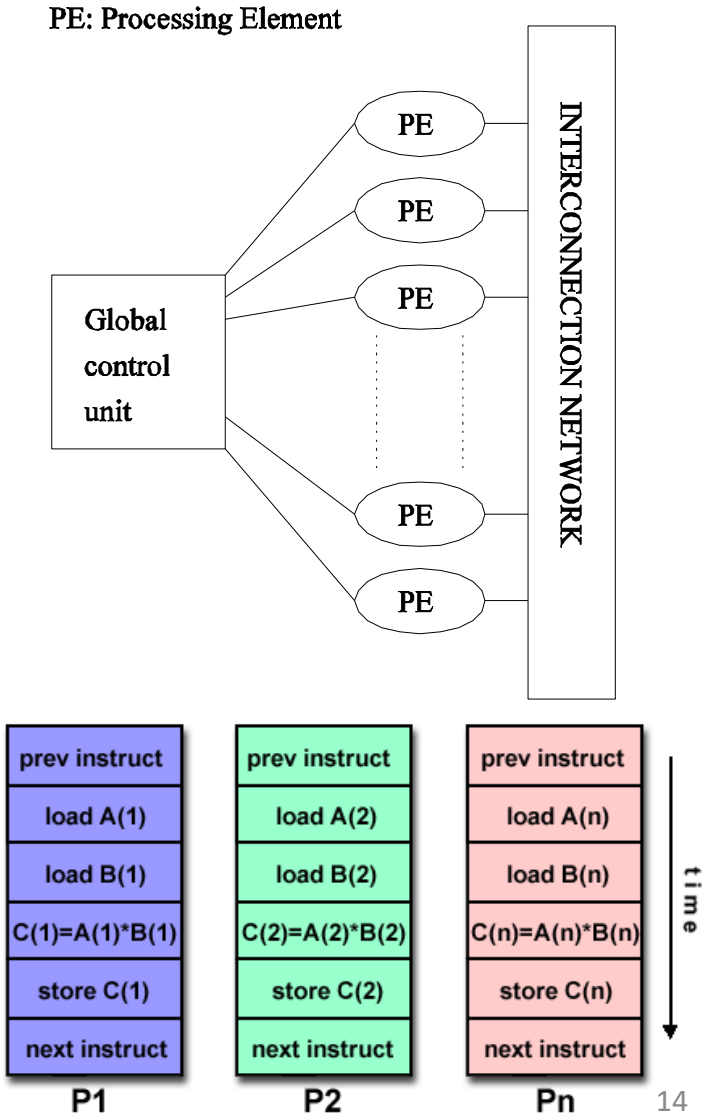


Dell laptop (uni-core)

UNIVAC1

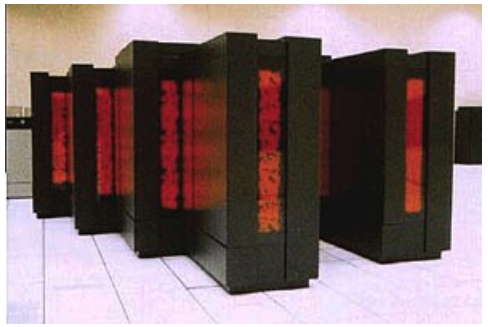
Single Instruction, Multiple Data (SIMD)

- Single Instruction: All processing units execute the same instruction at any given clock cycle
- Multiple Data: Each processing unit operate on a different data element
- It is often necessary to selectively turn off operations on certain data items. For this reason, most SIMD programming paradigms allow for an "activity mask", which determines if a processor should participate in a computation or not



Single Instruction, Multiple Data (SIMD) (cont.)

- Best suited for specialized problems with high degree of regularity, such as graphics/image processing
- Examples:
  - Connection Machine CM-2
  - Cray X-MP
- Most modern computers, particularly GPUs (Graphics Processing Unit) employ SIMD instructions and execution units



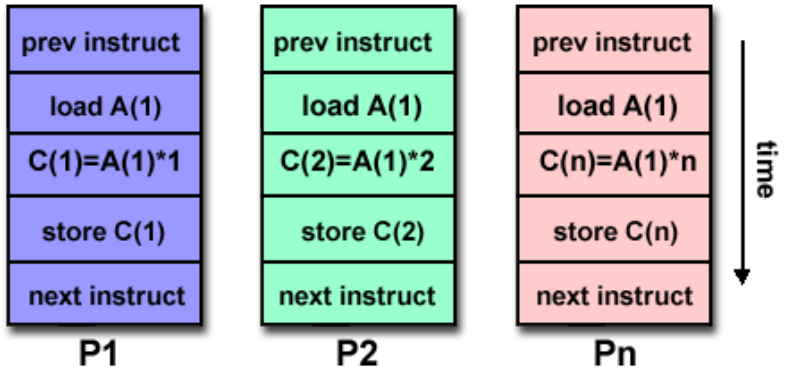
CM-2

Cray X-MP

Fermi class GTX480

Multiple Instruction, Single Data (MISD)

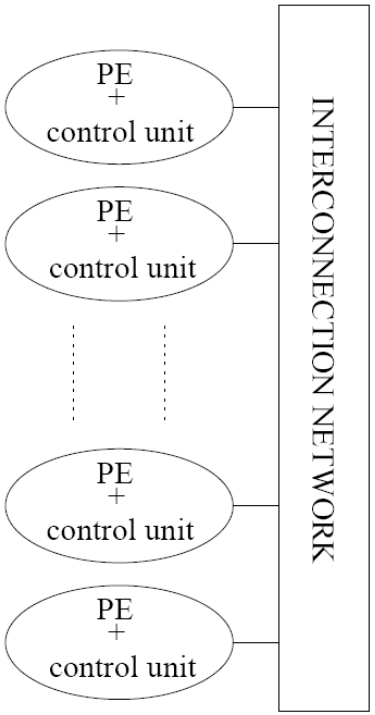
- A single data stream is fed into multiple processing units
- Each processing unit operates on the data independently via independent instruction streams
- Few actual examples of this class of parallel computer have ever existed
- Some conceivable uses might be
  - Multiple frequency filters operating on a single signal stream
  - Multiple cryptography algms attempting to crack a single coded msg

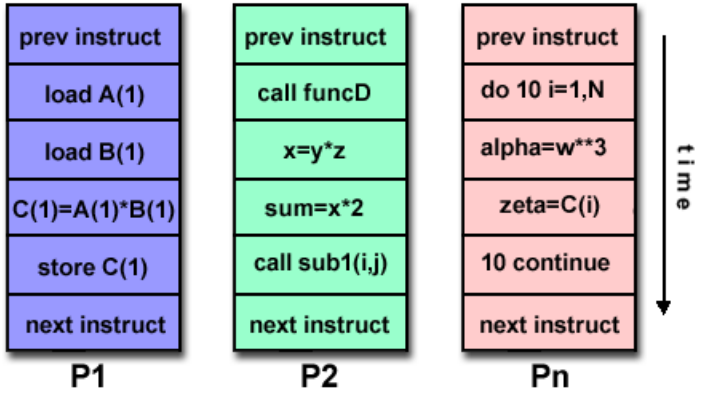


Multiple Instruction, Multiple Data (MIMD)

- Multiple Instruction: every processor may execute different instruction stream
  - May have separate clocks

- Multiple Data: every processor may be working with a different data stream

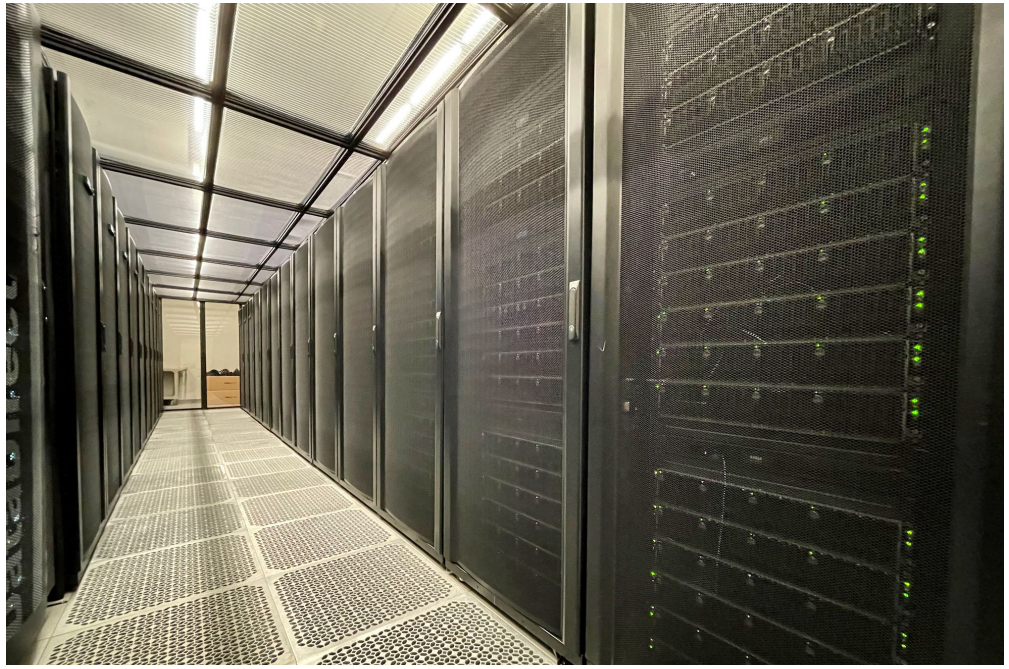




Multiple Instruction, Multiple Data (MIMD) (cont.)

- Examples
  - Multicore PCs
  - Servers
  - Clusters/warehouse-scale computers
  - Etc.

- Note: many MIMD architectures also include SIMD execution sub-components (e.g. GPUs)



RedRaider cluster at TTU

Outline

- Trends in Architecture
- Defining Computer Architecture
- Trends in Technology

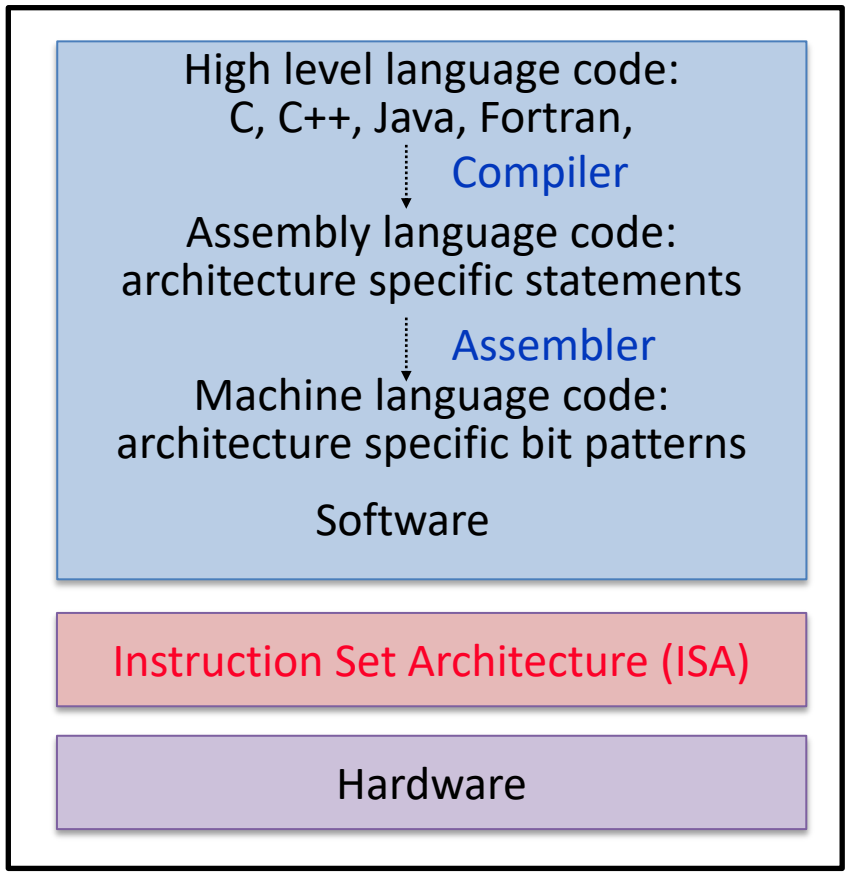
Defining Computer Architecture

- "Old" view of computer architecture:
  - Instruction Set Architecture (ISA) design
  - i.e., decisions regarding:
    - Registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding

- "Real" computer architecture:
  - Specific requirements of the target machine
    - Functional organization, power, packaging, cooling, etc.
  - Design to maximize performance within constraints: cost, power, and availability
  - Includes ISA, microarchitecture, hardware

Instruction Set Architecture (ISA)

- Instruction: computer's hardware language/commands
- Instruction set: the vocabulary of commands understood by a given architecture
- Instruction Set Architecture (ISA): defines the interface between software and hardware
- Different architectures have different instruction sets
  - But with many aspects in common



Instruction Set Architecture (ISA) (cont.)

- Instruction Set Architecture:
  - An abstract specification, can have many implementations
  - ISA creates its own software ecosystem
    - X86 ISA: more for desktops and servers
    - ARM ISA: more for mobile

- What is included in ISA?
  - Everything visible to software
  - Set of instructions and how they behave
  - Data types
  - Registers
  - Memory model
  - Protection, how I/O works, virtual memory, exceptions

RISC V.S. CISC

- RISC (Reduced Instruction Set Computer)
  - Fixed instruction lengths
  - Limited addressing modes
  - Limited number of instructions
    - E.g., only 3 jump instructions in MIPS (Microprocessor without Interlocked Pipelined Stages), a RISC ISA
  - Software-centric design
- CISC (Complex Instruction Set Computer)
  - Increased capability of each instruction
  - Lead to more addressing modes
  - Variable length instructions, variable instructions execution time
  - A large and powerful range of instructions
    - E.g., 32 jump instructions in an 8086 processor
  - Hardware-centric design

RISC V.S. CISC (cont.)

Tradeoff:

With **CISC** processors:

Q1. The assembly programs are much [shorter? or longer?]

Q2. Each line in the program takes much [shorter? or longer?] to interpret and execute

With **RISC** processors:

Q3. The programs require [more? or less?] instructions

Q4. But each instruction takes much [shorter? or longer?]

Instruction Set Architecture Examples

- Class of ISA
  - General-purpose registers
  - Register-memory vs load-store
- RISC-V registers
  - 32 g.p., 32 f.p.

Register	Name	Use	Saver
x0	zero	constant 0	n/a
x1	ra	return addr	caller
x2	sp	stack ptr	callee
x3	gp	gbl ptr	
x4	tp	thread ptr	
x5-x7	t0-t2	temporaries	caller
x8	s0/fp	saved/ frame ptr	callee

Register	Name	Use	Saver
x9	s1	saved	callee
x10-x17	a0-a7	arguments	caller
x18-x27	s2-s11	saved	callee
x28-x31	t3-t6	temporaries	caller
f0-f7	ft0-ft7	FP temps	caller
f8-f9	fs0-fs1	FP saved	callee
f10-f17	fa0-fa7	FP arguments	callee
f18-f27	fs2-fs21	FP saved	callee
f28-f31	ft8-ft11	FP temps	caller

Instruction Set Architecture Examples

- Memory addressing
  - RISC-V: byte addressed, aligned accesses faster
- Addressing modes
  - RISC-V: Register, immediate, displacement (base+offset)
  - Other examples: autoincrement, indexed, PC-relative
- Types and size of operands
  - RISC-V: 8-bit (byte), 16-bit (half-word), 32-bit (word), 64-bit (double word)

Instruction Set Architecture Examples

- Operations
  - RISC-V: data transfer, arithmetic, logical, control, floating point
  - See Fig. 1.5 in text
- Control flow instructions
  - Use content of registers (RISC-V) vs. status bits (x86, ARMv7, ARMv8)
  - Return address in register (RISC-V, ARMv7, ARMv8) vs. on stack (x86)
- Encoding
  - Fixed length (RISC-V, ARMv7/v8 except compact instruction set) vs. variable length (x86)

Top 10 Instructions for the 80x86

Rank	80x86 instruction	Integer average (% total executed)
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
Total		**96%**

Figure A.13 The top 10 instructions for the 80x86. Simple instructions dominate this list and are responsible for 96% of the instructions executed. These percentages are the average of the five SPECint92 programs.

Outline

- Trends in Architecture
- Defining Computer Architecture
- Trends in Technology

Trends in Technology

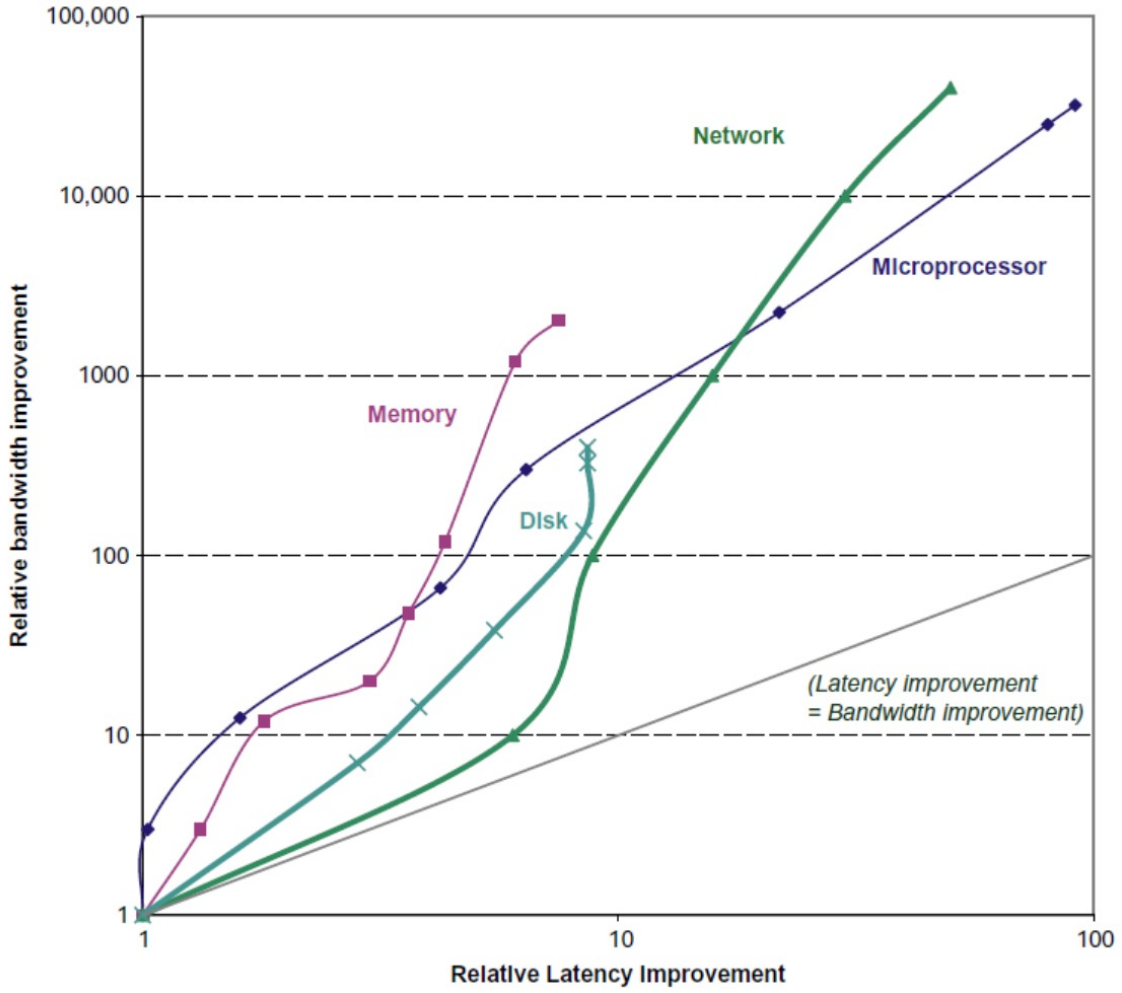
- Integrated circuit technology
  - Transistor density: 35%/year
  - Die (chip) size: 10-20%/year
  - Integration overall: 40-55%/year or doubling every 18 to 24 months (Moore's law) (we are also approaching post Moore's law era)
- DRAM capacity: 25-40%/year (slowing)
- Flash capacity: 50-60%/year
  - 8-10X cheaper/bit than DRAM
- Magnetic disk technology: 40%/year, recently slowed to 5%/year
  - Density increases may no longer be possible, maybe increase from 7 to 9 platters
  - 8-10X cheaper/bit than Flash
  - 200-300X cheaper/bit than DRAM

Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time (e.g., MB/s for disk transfer)
  - 32,000-40,000X improvement for processors
  - 300-1200X improvement for memory and disks

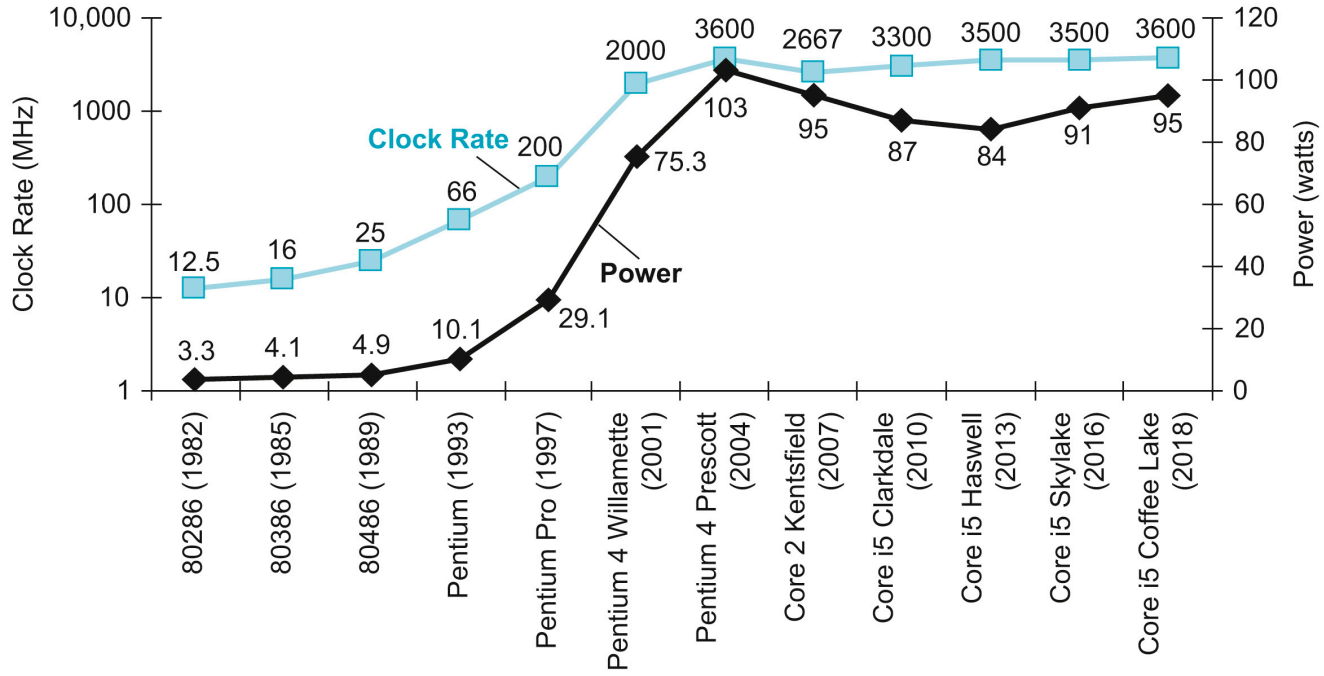
- Latency or response time
  - Time between start and completion of an event (e.g., milliseconds for a disk access)
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks

Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

Power Wall



Clock rate and power for Intel x86 microprocessors over nine generations and 36 years.

- In CMOS (Complementary Metal Oxide Semiconductor) IC technology

Power: 1/2 X Capacitive load X Voltage^2 X Frequency

×40

5V → 1V

×1000

Power Wall (cont.)

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction
  - What's the new power required?

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further (otherwise causing transistors too leaky)
  - We can't remove more heat
  - Battery life and energy bills

- How else can we improve performance?

Uniprocessor (unicore processor) ->
Multiprocessor (multicore processor)

Multiprocessors

- Multicore processors
  - More than one processor (core) per chip

- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer (free ride)
  - Hard to do (free ride is over; requires programmer's extensive efforts)
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Reducing Power

- Techniques for reducing power:
  - Turn off the clock of inactive modules (e.g., floating point units)
  - Dynamic Voltage-Frequency Scaling (DVFS)
  - Design for typical case: e.g., low power state for DRAM, disks
  - Overclocking of a single core (e.g., Intel Turbo mode) for single-threaded code while turning off other cores

Readings

- Chapter 1, 1.1-1.5