# Computer Architecture & Assembly Language 14:332:331

## Lecture 1
## Introduction

**Naghmeh Karimi**

**Fall 16**

Adapted from *Computer Organization and Design*, *5th Edition*, Patterson & Hennessy, © 2013, Elsevier,
*and Computer Organization and Design*, *4th Edition*, Patterson & Hennessy, © 2008, Elsevier
and Mary Jane Irwin's slides from Penn State University.

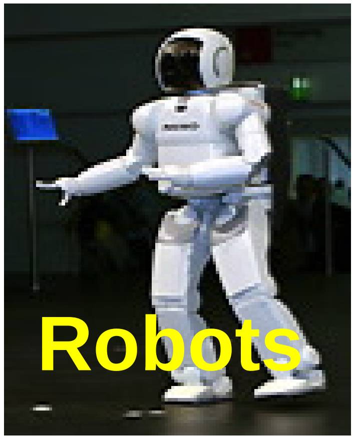
# Computing Devices Now

Sensor Nets

Laptops

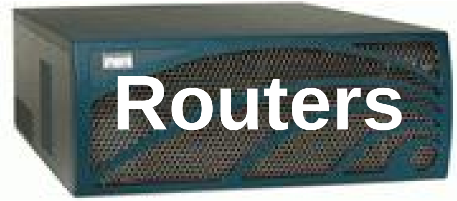
Games

Servers



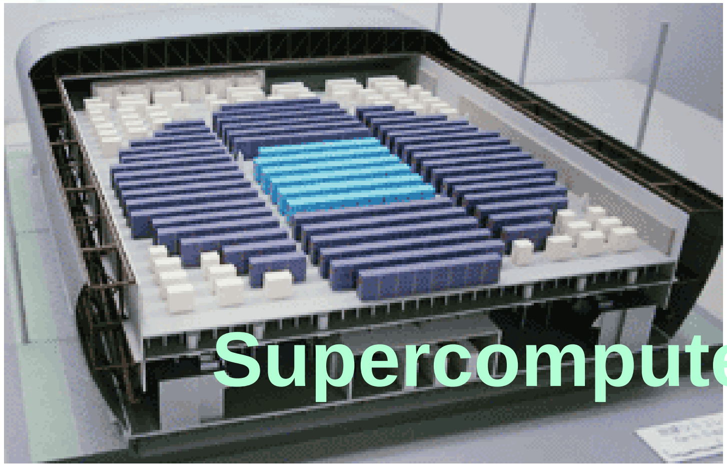
Robots

Smart phones



Routers

Automobiles



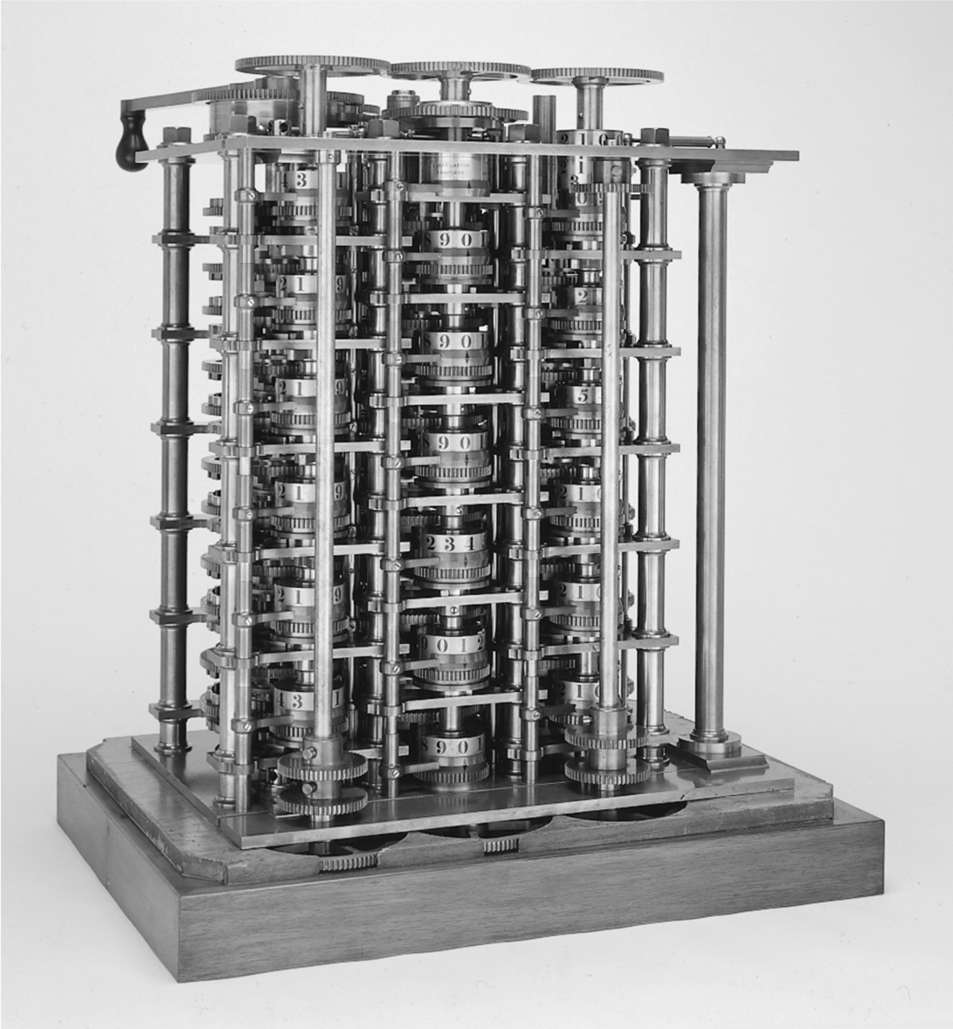
Supercomputers

Slide adapted from http://inst.eecs.berkeley.edu/~cs252

# A Brief History of IC

- 1958: First integrated circuit
  - Flip-flop using two transistors
  - Built by Jack Kilby at Texas Instruments
- 2003
  - Intel Pentium 4 mprocessor (55 million transistors)
  - 512 Mb DRAM (> 0.5 billion transistors)
- No other technology has grown so fast so long
- Driven by miniaturization of transistors
  - Smaller is cheaper, faster, lower in power!
  - Revolutionary effects on society

# The First Computer: Babbage Difference Engine



- Invented by: Charles Babbage
- Constructed in 1832
- Built from 25,000 parts
- Cost: £17,470

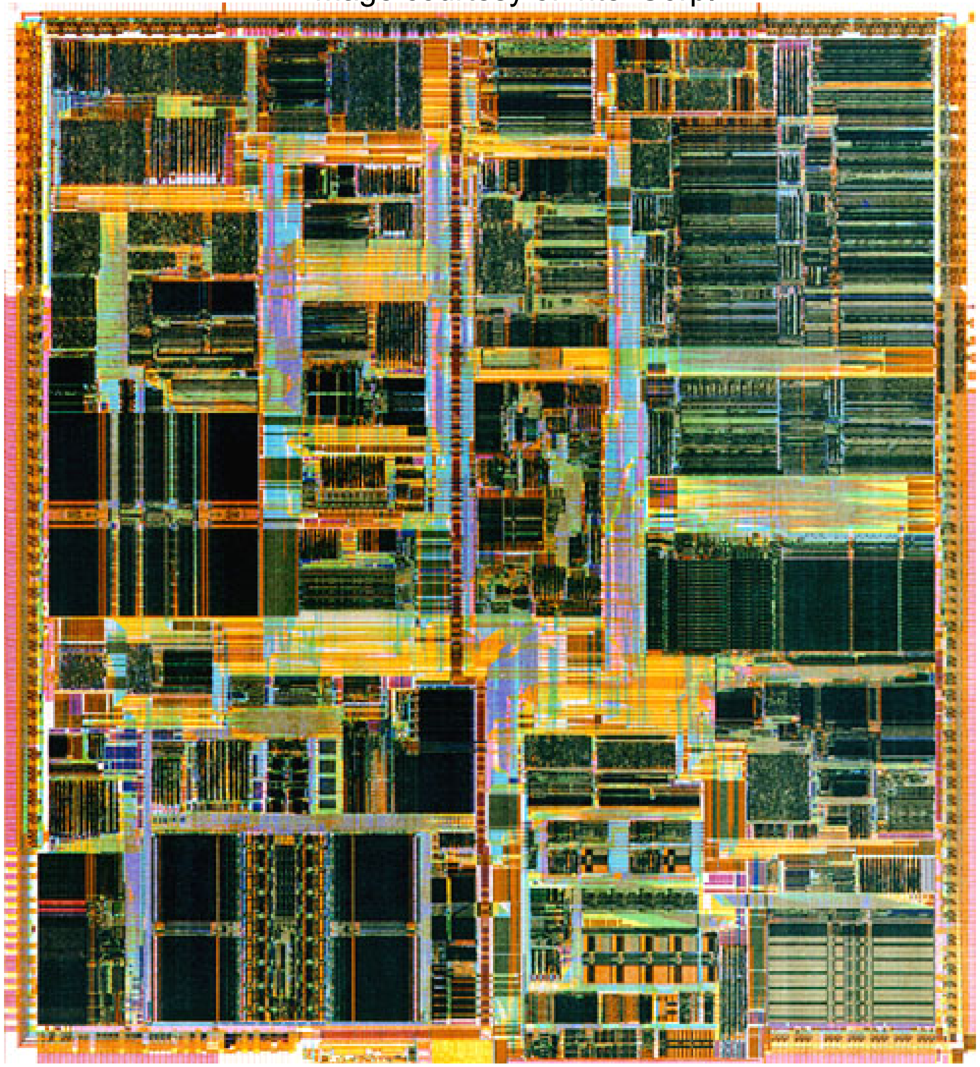
# First Integration Circuit

- Jack Kilby (Texas Instruments) invented the first integrated circuit in 1958
- Kilby joined TI just before vacation time, so while everyone else was on vacation…
- Kiby received the Nobel Prize in Physics in 2000 for the invention of the integrated circuit.

# Modern Processors

- A lot of chip area is memory (cache)
- Very little is completely custom (datapath)
- Many blocks synthesized from VHDL or Verilog

Image courtesy of Intel Corp.

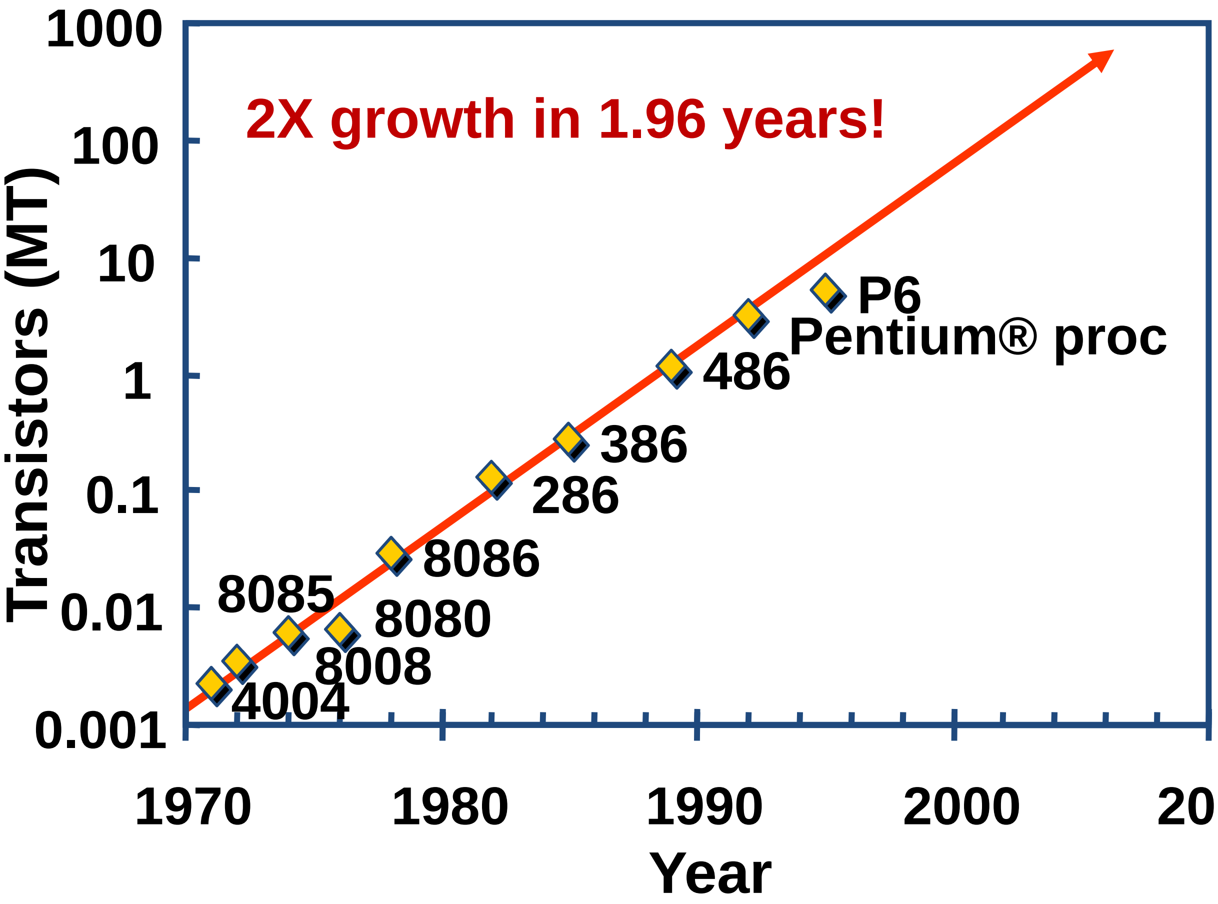


Intel Pentium V

# Moore's Law

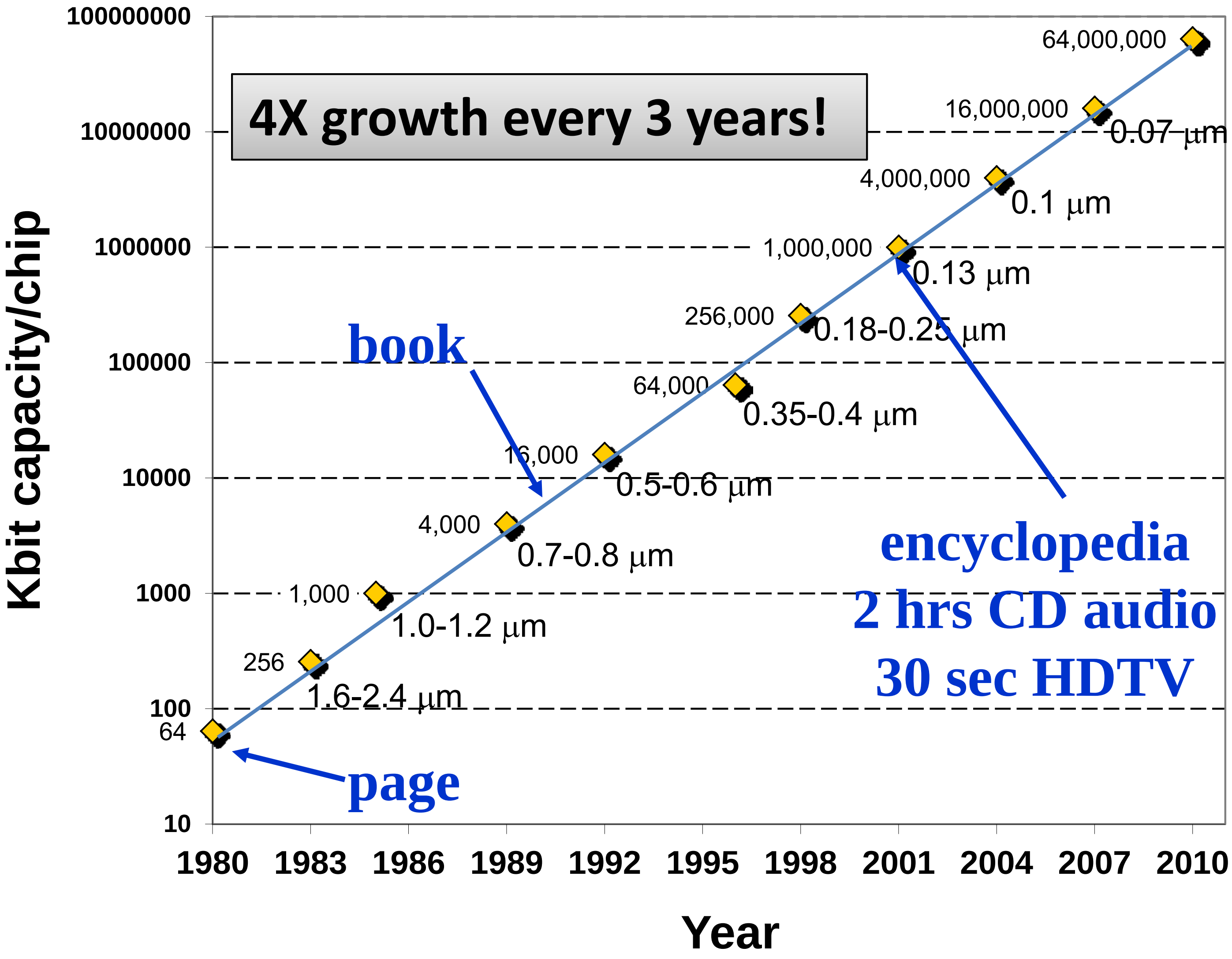
- In 1965, Gordon Moore predicted that the number of transistors that can be integrated on a die would double every 14 to 18 months (i.e., grow exponentially with time).
- Amazingly visionary – million transistor/chip barrier was crossed in the 1980's.
  - 2300 transistors, 1 MHz clock (Intel 4004) - 1971
  - 16 Million transistors (Ultra Sparc II) - 1997
  - 42 Million, 2 GHz clock (Intel P4) - 2001
  - 140 Million transistor (HP PA-8500)

# Moore's Law in Microprocessor

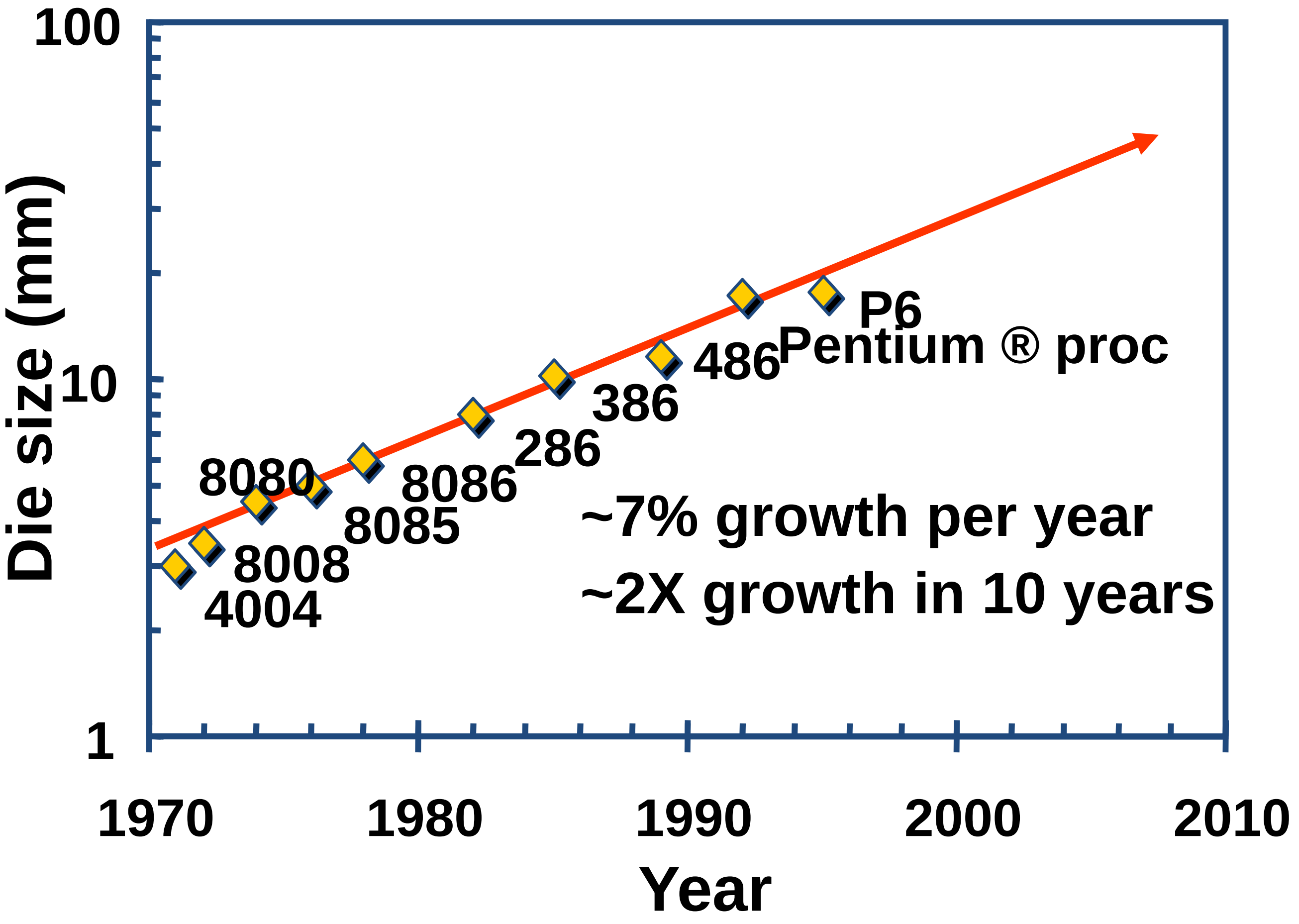


**Transistors on lead microprocessors double every 2 years**

# Evolution in DRAM Chip Capacity

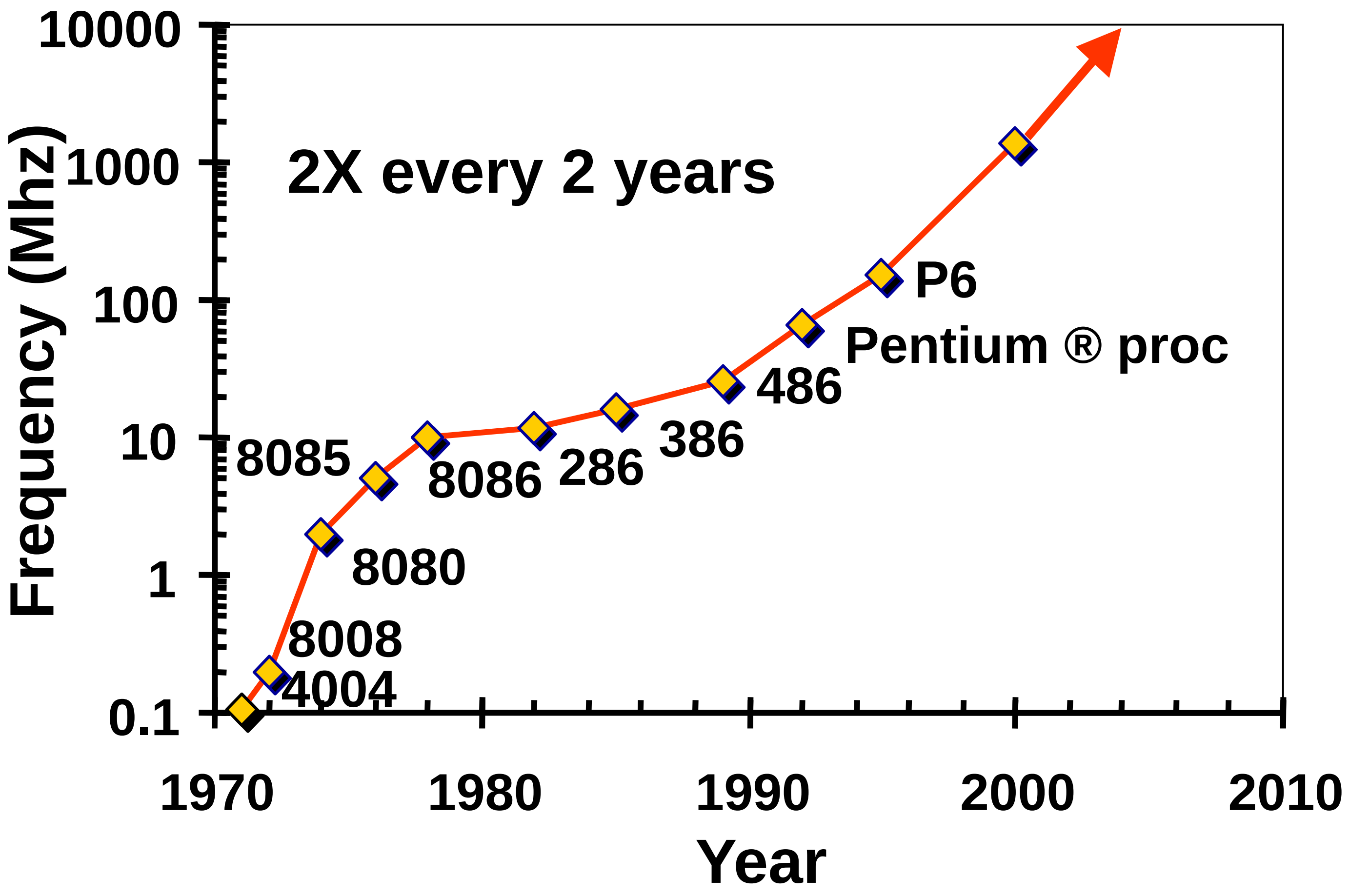


# Die Size Grows



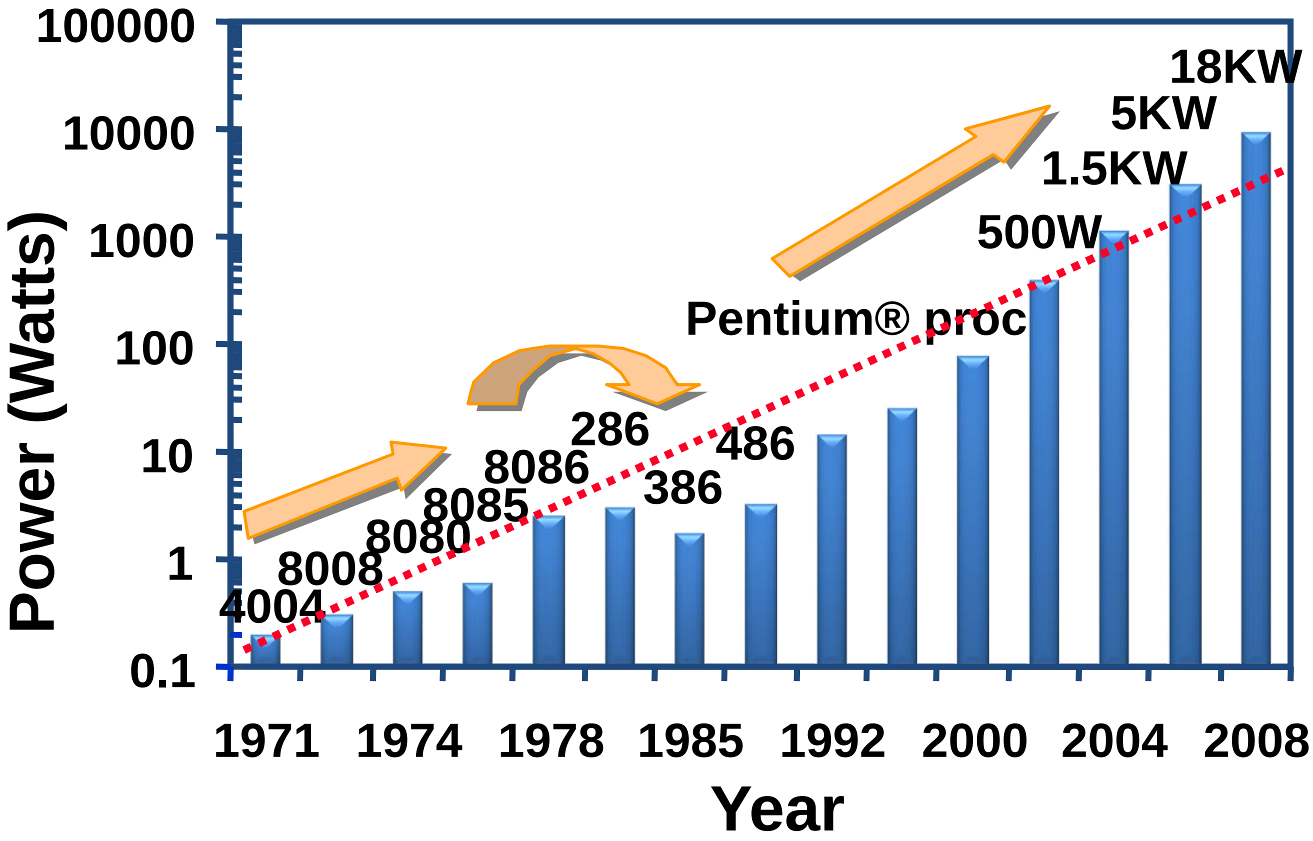
**Die size grows by 14% to satisfy Moore’s Law**

# Clock Frequency



**Lead microprocessors frequency doubles every 2 years**

# Power Dissipation



**Power delivery and dissipation will be prohibitive**

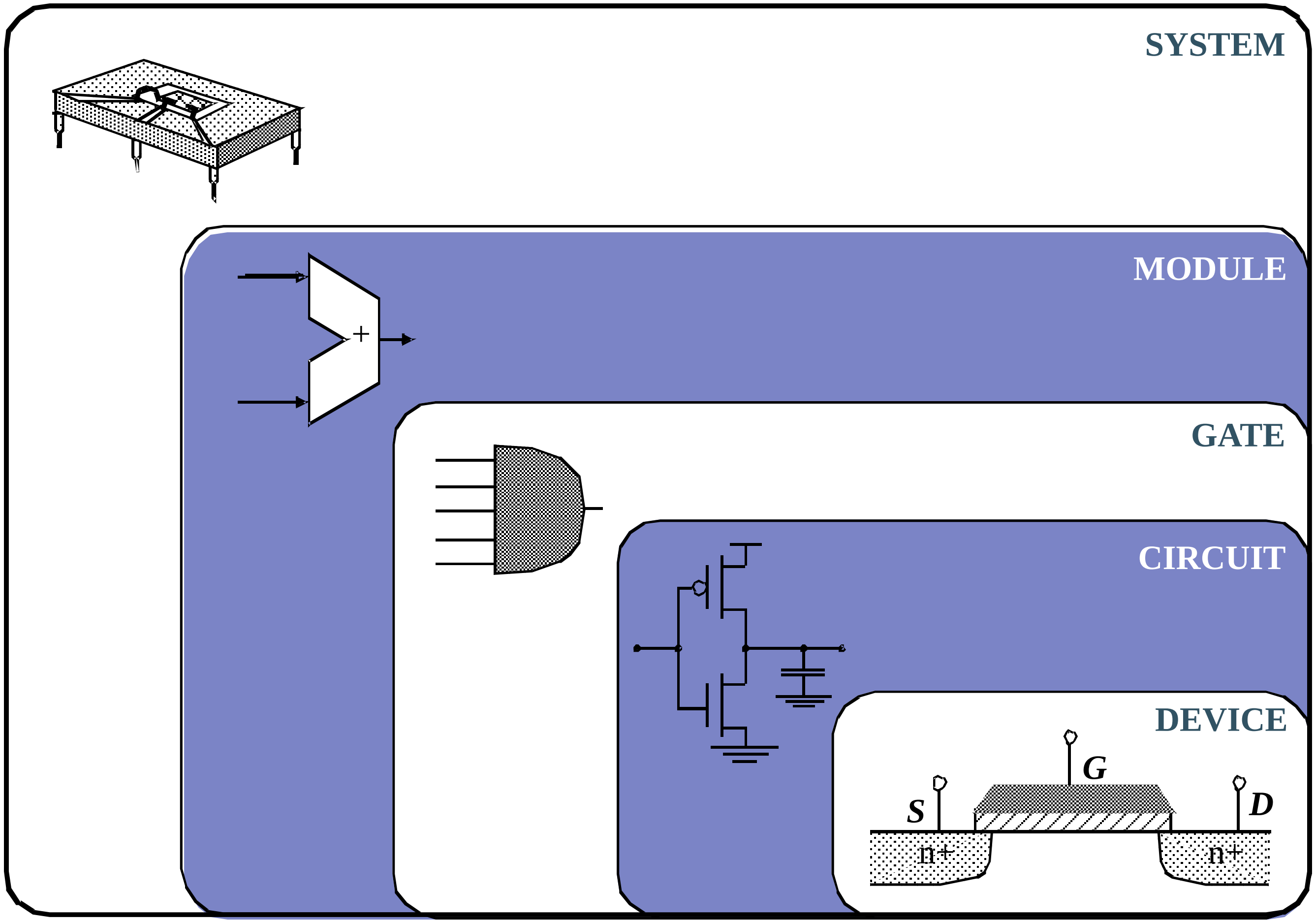
# Why Scaling?

- Technology shrinks by ~0.7 per generation
- With every generation can integrate 2x more functions on a chip; chip cost does not increase significantly
- Cost of a function decreases by 2x
- But …
  - How to design chips with more and more functions?
  - Design engineering population does not double every two years…
- Hence, a need for more efficient design methods
  - Exploit different levels of abstraction

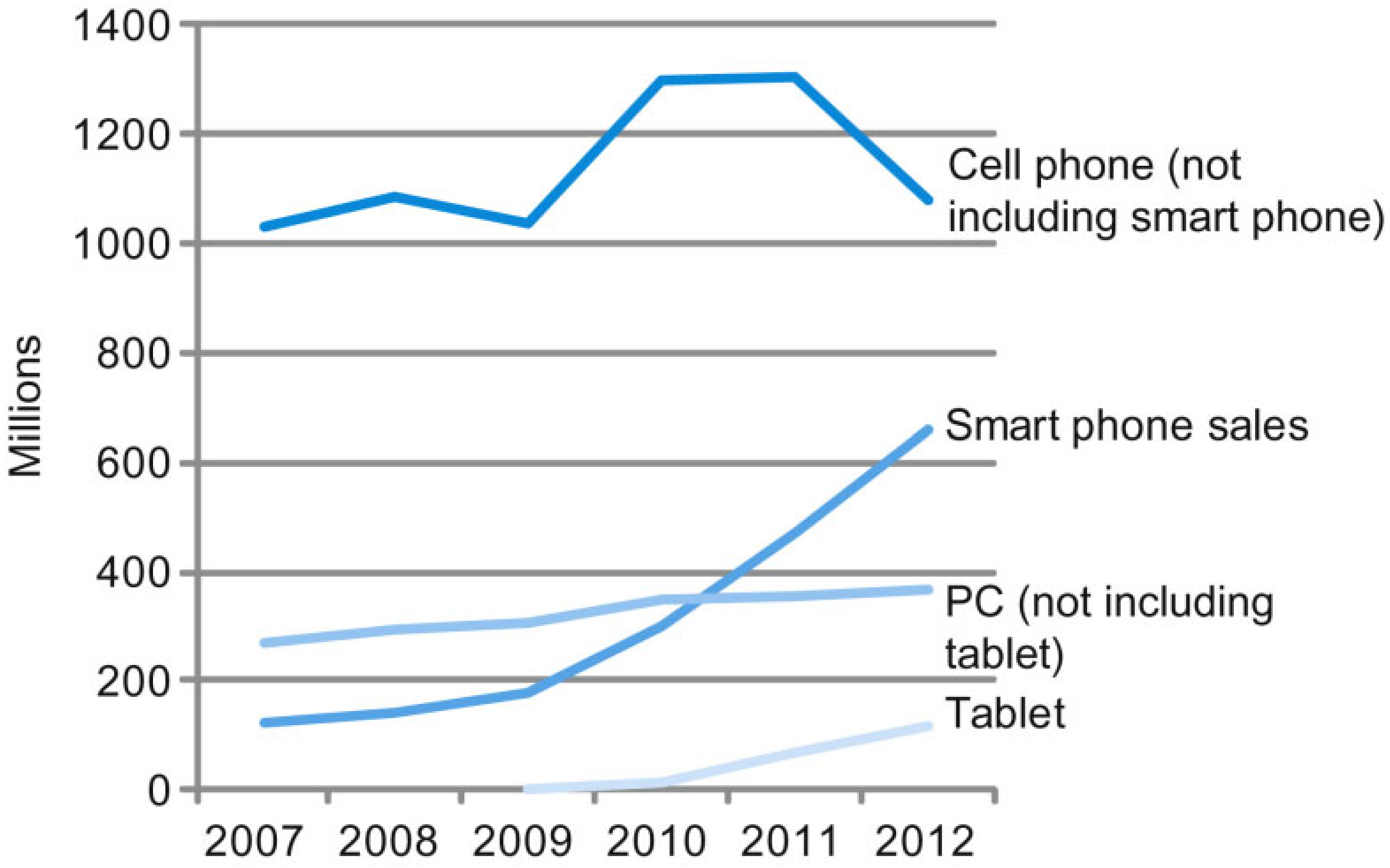
# Design Partitioning

- Architecture: User’s perspective, what does it do?
  - Instruction set, registers
  - MIPS, x86, Alpha, PIC, ARM, …
- Microarchitecture
  - Single cycle, multi-cycle, pipelined, superscalar?
- Logic: how are functional blocks constructed
  - Ripple carry, carry look-ahead, carry select adders
- Circuit:  how are transistors used
  - Complementary CMOS, pass transistors, domino
- Physical: chip layout

# Design Abstraction Levels



# The PostPC Era

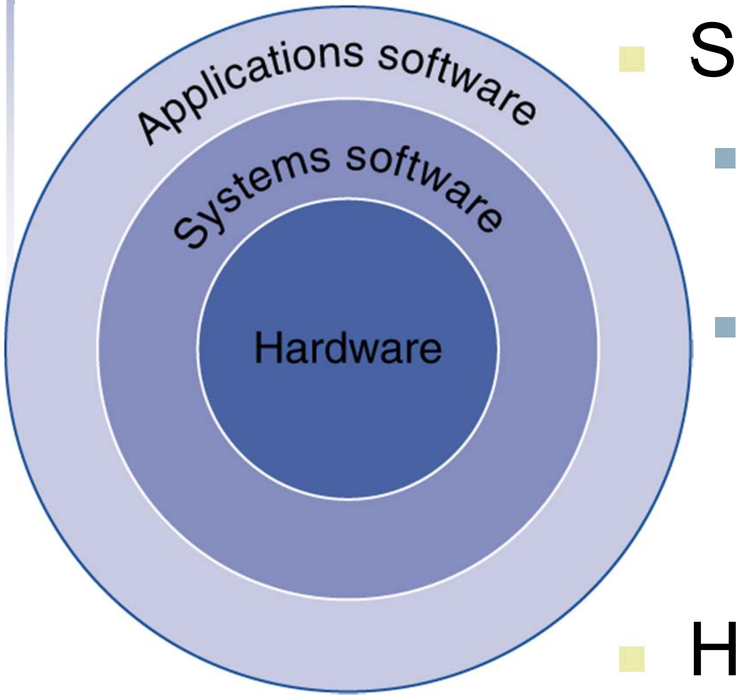


**The number manufactured of each device per year.** Tablets are the fastest growing category, nearly doubling between 2011 and 2012.

# Classes of Computers

- **Desktop computers**
  - Designed to deliver good performance to a single user at low cost usually executing 3rd party software, usually incorporating a graphics display, a keyboard, and a mouse
- **Servers**
  - Used to run larger programs for multiple, simultaneous users typically accessed only via a network and that places a greater emphasis on dependability and (often) security
- **Supercomputers**
  - A high performance, high cost class of servers with hundreds to thousands of processors, terabytes of memory and petabytes of storage that are used for high-end scientific and engineering applications
- **Embedded computers (processors)**
  - A computer inside another device used for running one predetermined application

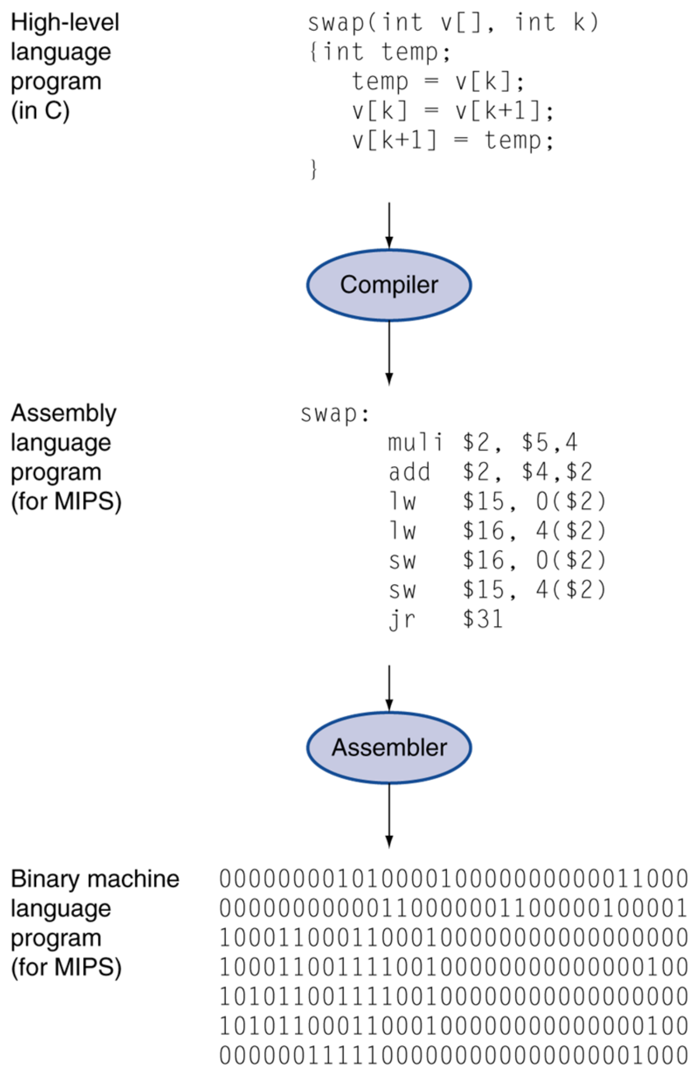
# Below Your Program



- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

# Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



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;
}
```

Assembly language program (for MIPS)

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

Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

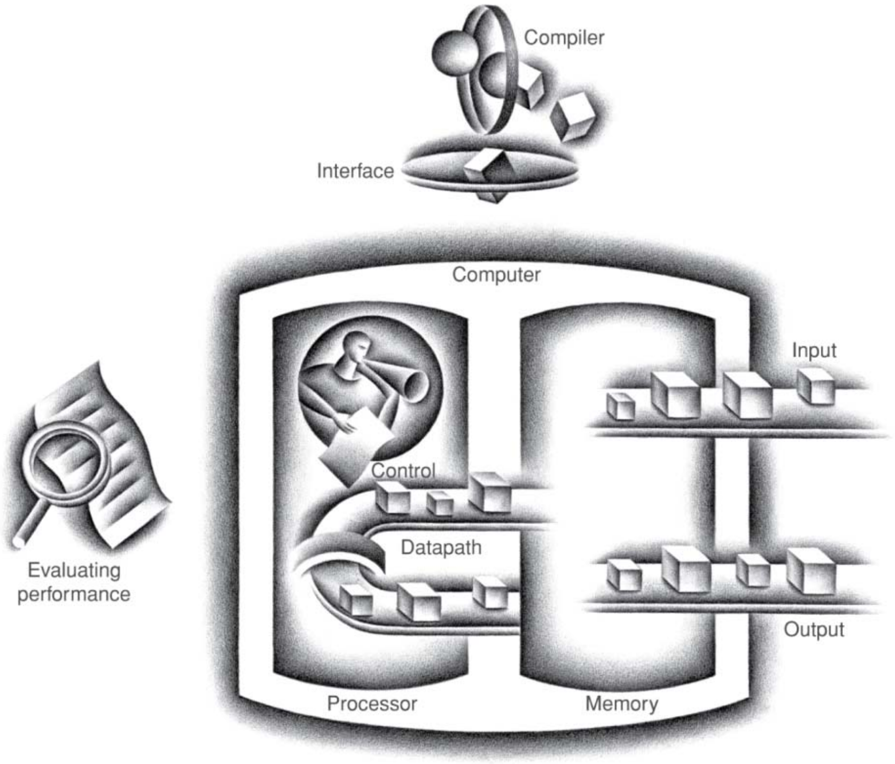
# Advantages of Higher-Level Languages?

- Higher-level languages
  - Allow the programmer to think in a more natural language and for their intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, …)
  - Improve programmer productivity – more understandable code that is easier to debug and validate
  - Improve program maintainability
  - Allow programs to be independent of the computer on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine)
  - Emergence of optimizing compilers that produce very efficient assembly code optimized for the target machine

As a result, very little programming is done today at the assembler level

# Components of a Computer

The BIG Picture

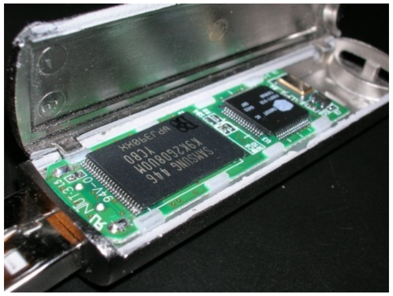


- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

§ 1.4 Under the Covers

# A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)

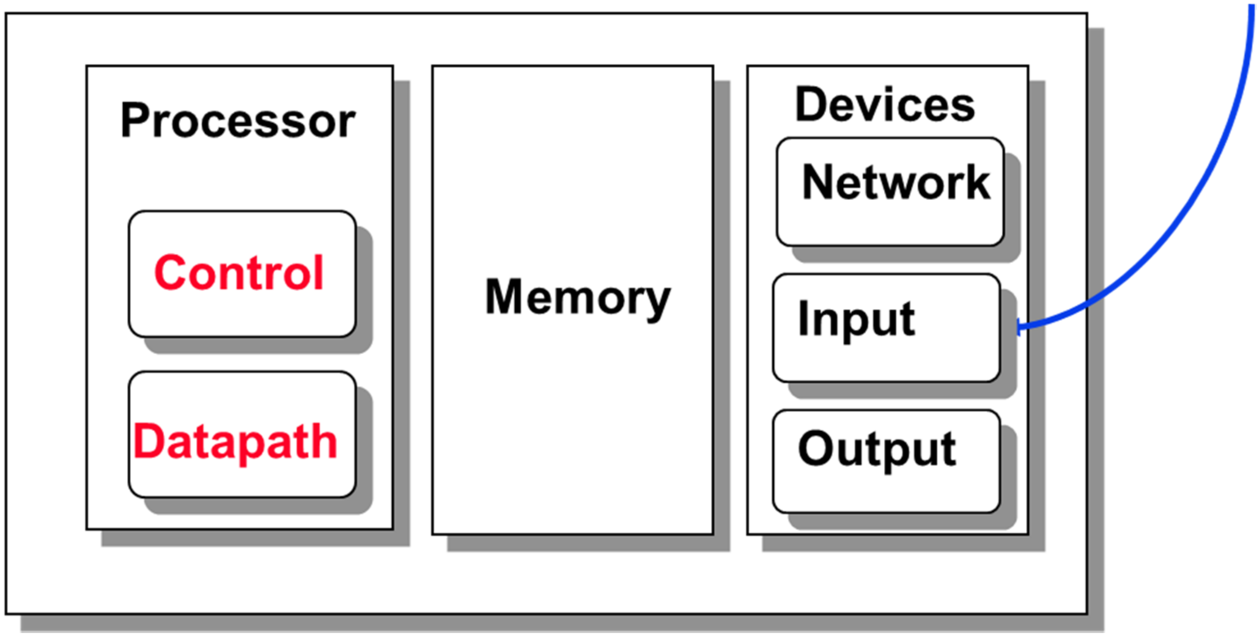


# Inside the Processor (CPU)

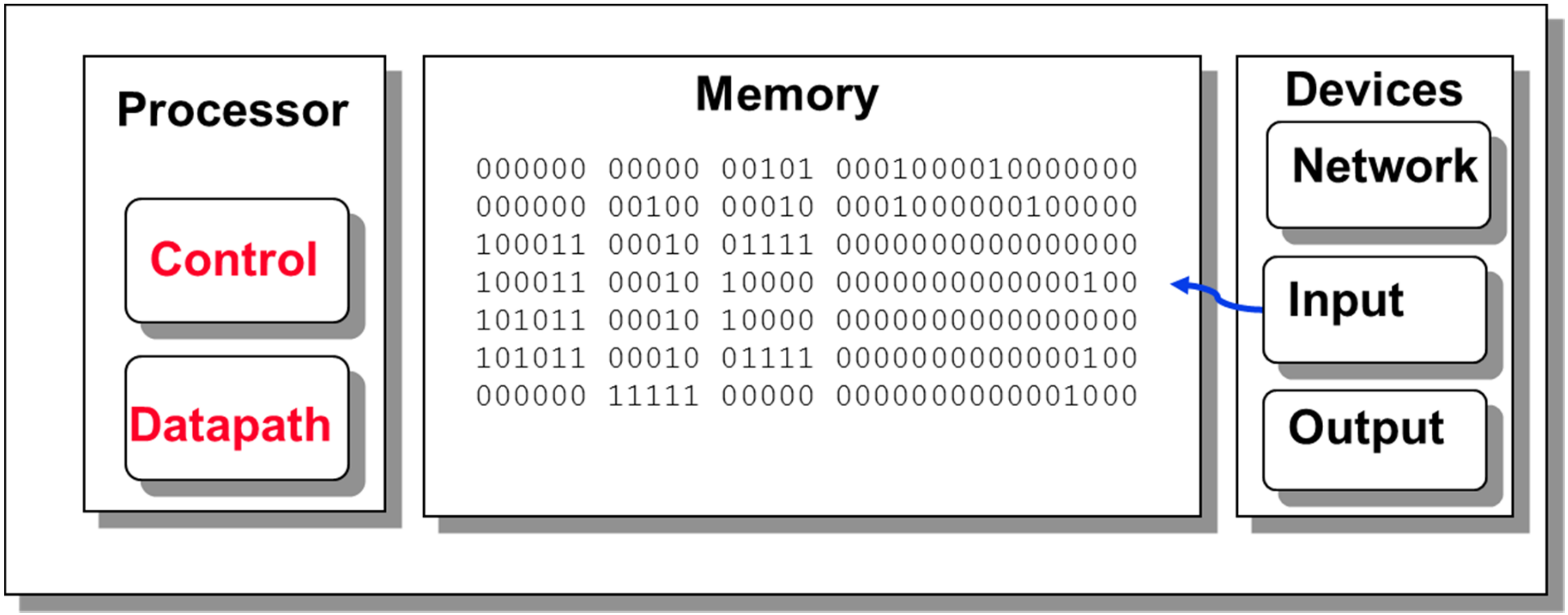
- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

# Input Device Inputs Object Code

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```

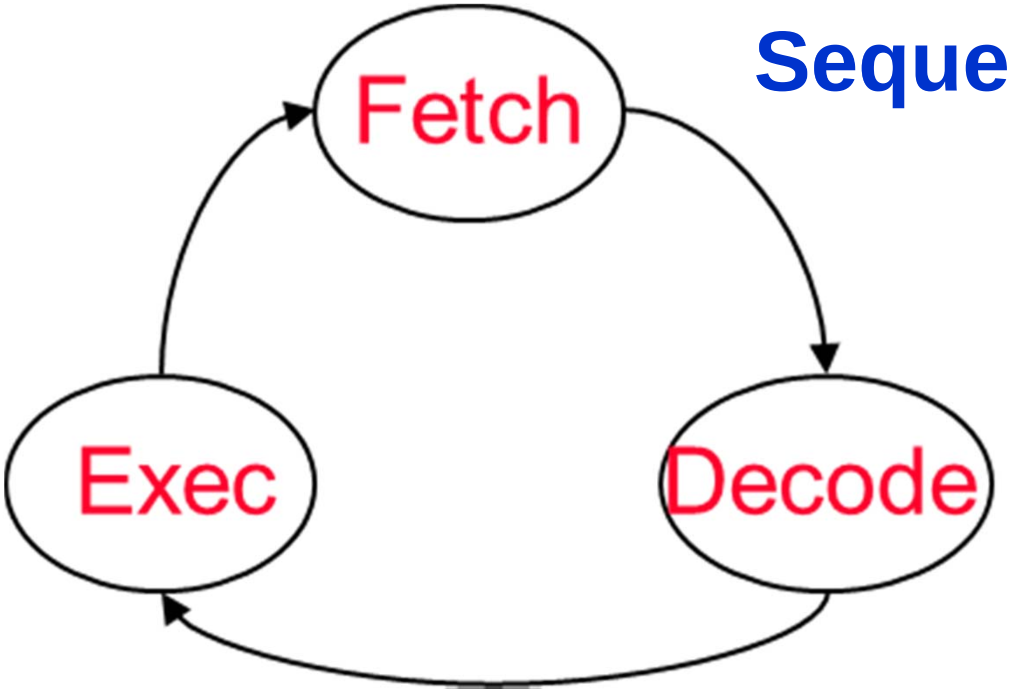


# Object Code Stored in Memory



```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```

# How to execute a program?



Sequential execution

# Processor Organization

- **Control needs to have**
  - Ability to input instructions from memory
  - Logic and means to control instruction sequencing
  - Logic and means to issue signals that control the way information flows between datapath components
  - Logic and means to control what operations the datapath' s functional units perform
- **Datapath needs to have**
  - Components - functional units (e.g., adder) and storage locations (e.g., register file) - needed to execute instructions
  - Components interconnected so that the instructions can be accomplished
  - Ability to load data from and store data to memory

# Instruction Fetch

- How do you know which instruction next?

  PC (Program Counter)

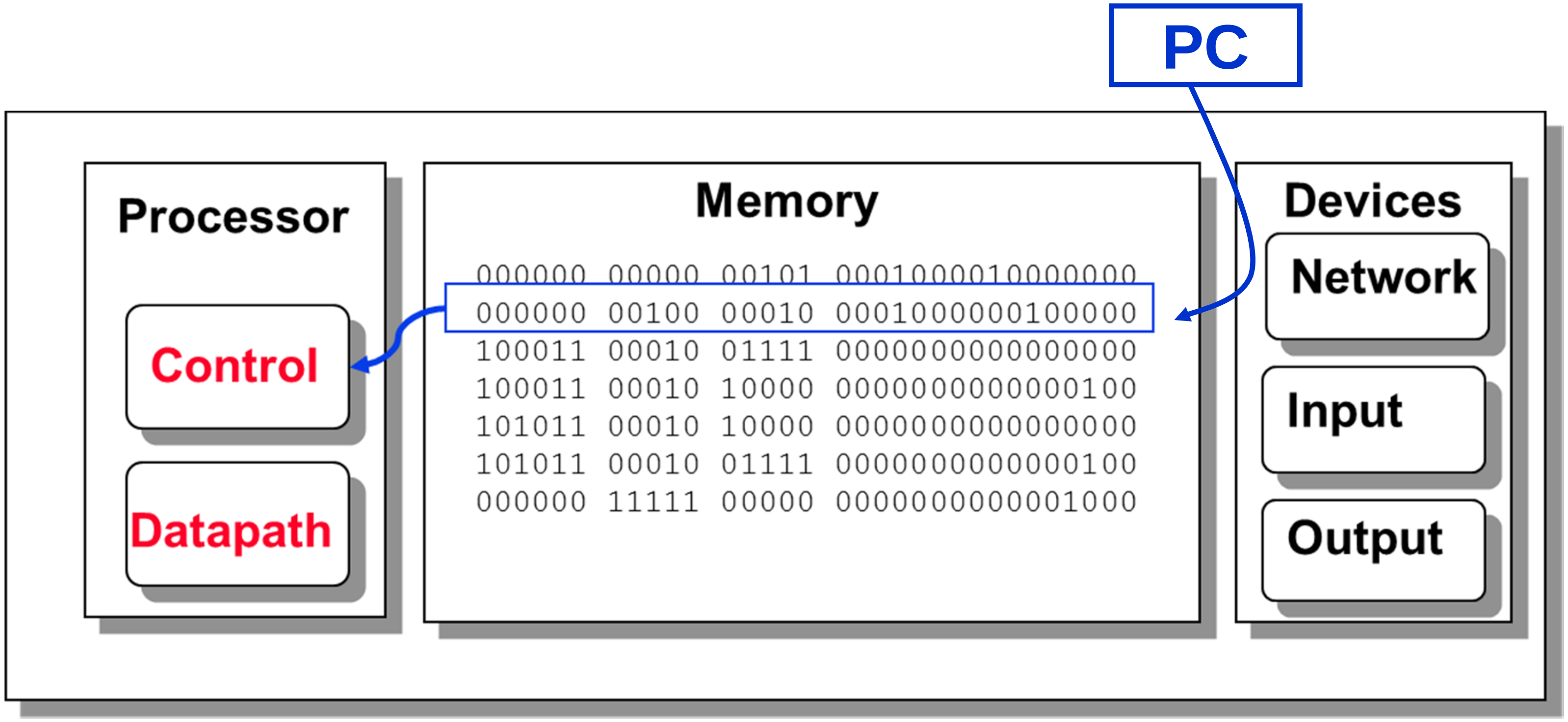
- Where to store PC?

  Disk, memory, cache, register

- How to update PC?

  Sequential, branch

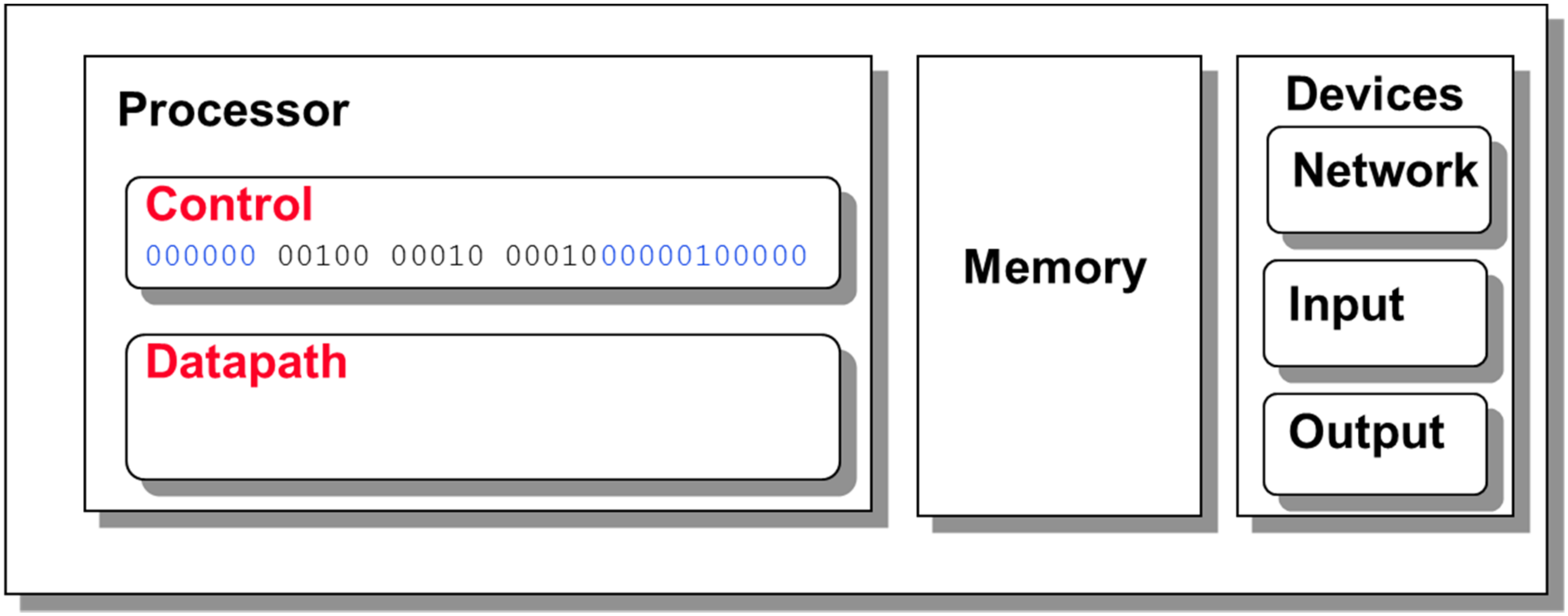
# Processor Fetches an Instruction



```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```

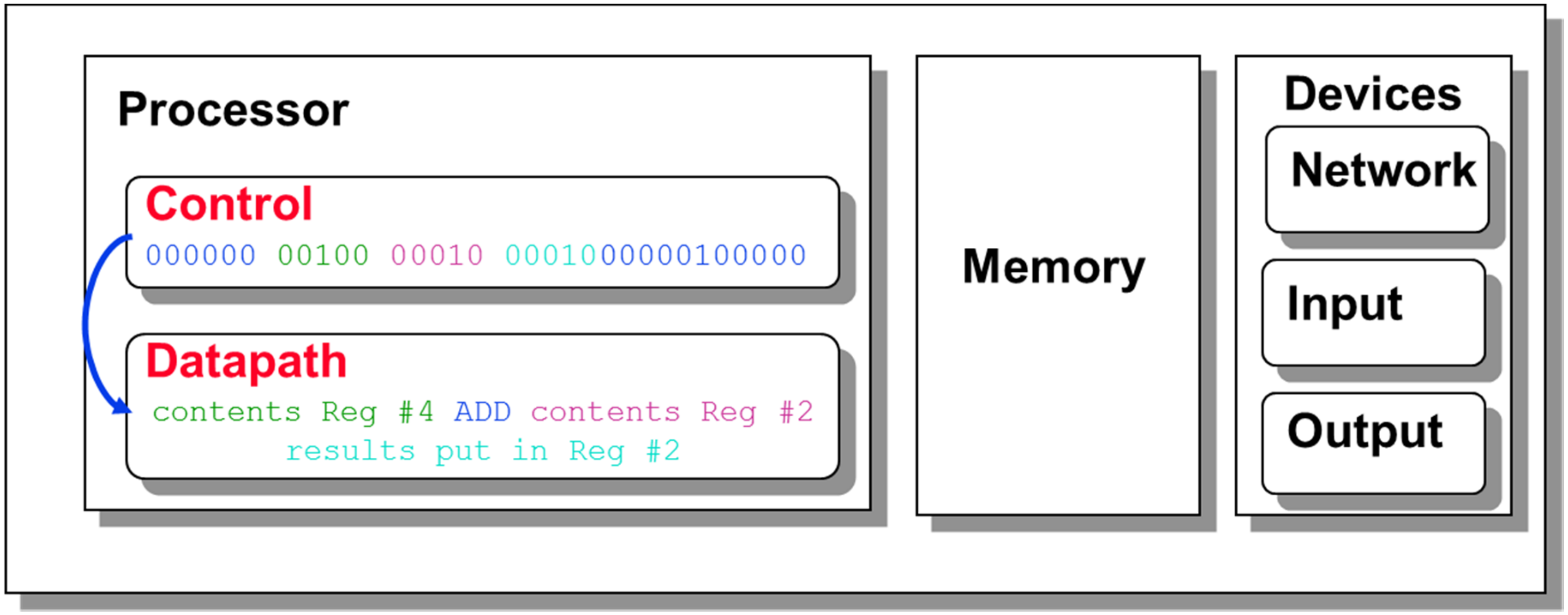
Processor fetches an instruction from memory

# Control Decodes the Instruction



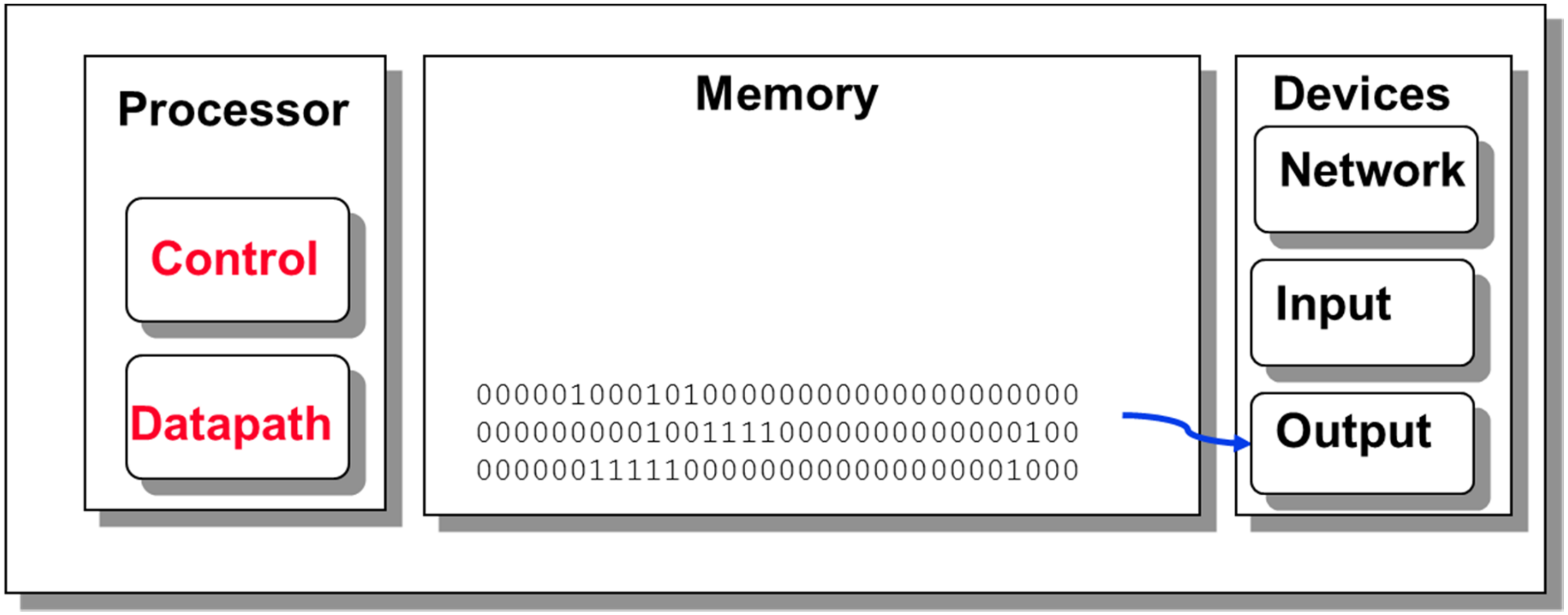
Control decodes the instruction to determine what to execute

# Datapath Executes the Instruction



Datapath executes the instruction
as directed by control

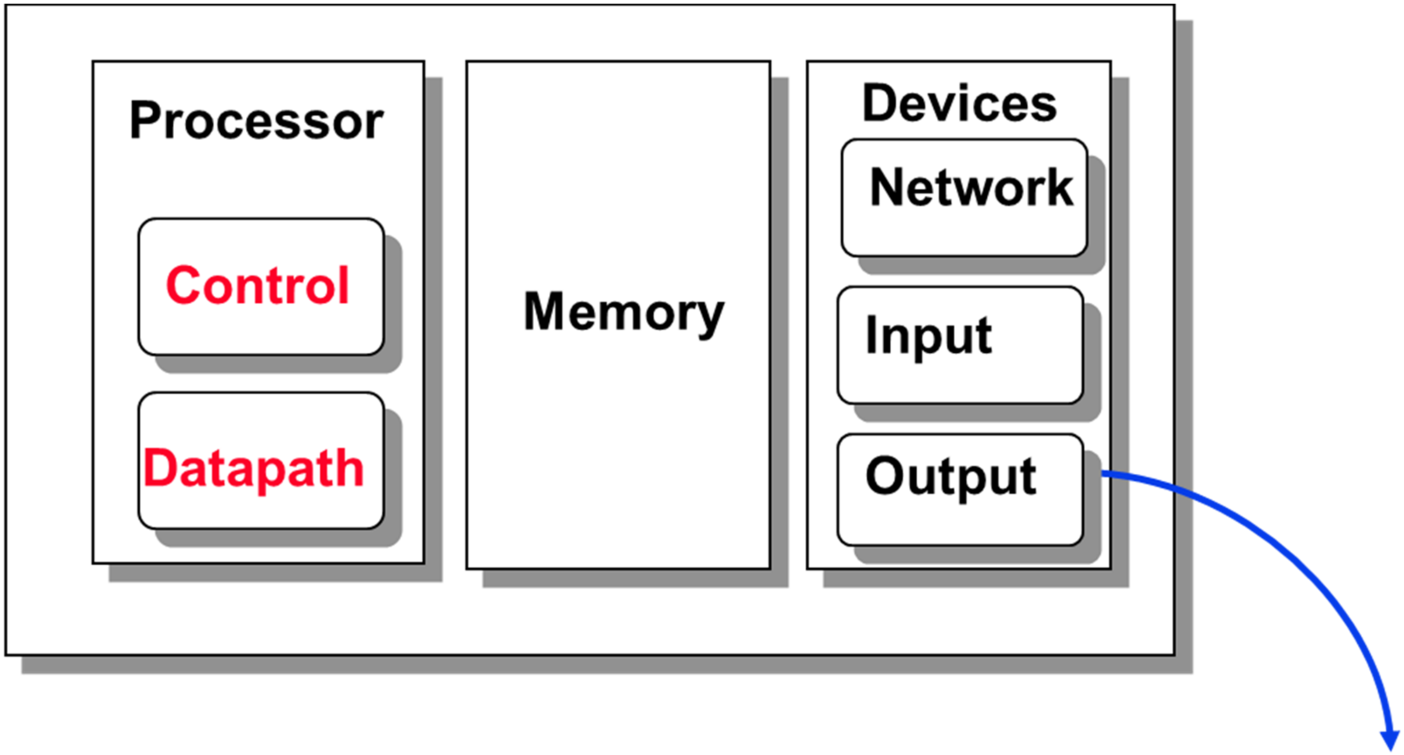
# Output Data Stored in Memory



```
00000100010100000000000000000000
00000000010011110000000000000100
00000011111000000000000000001000
```

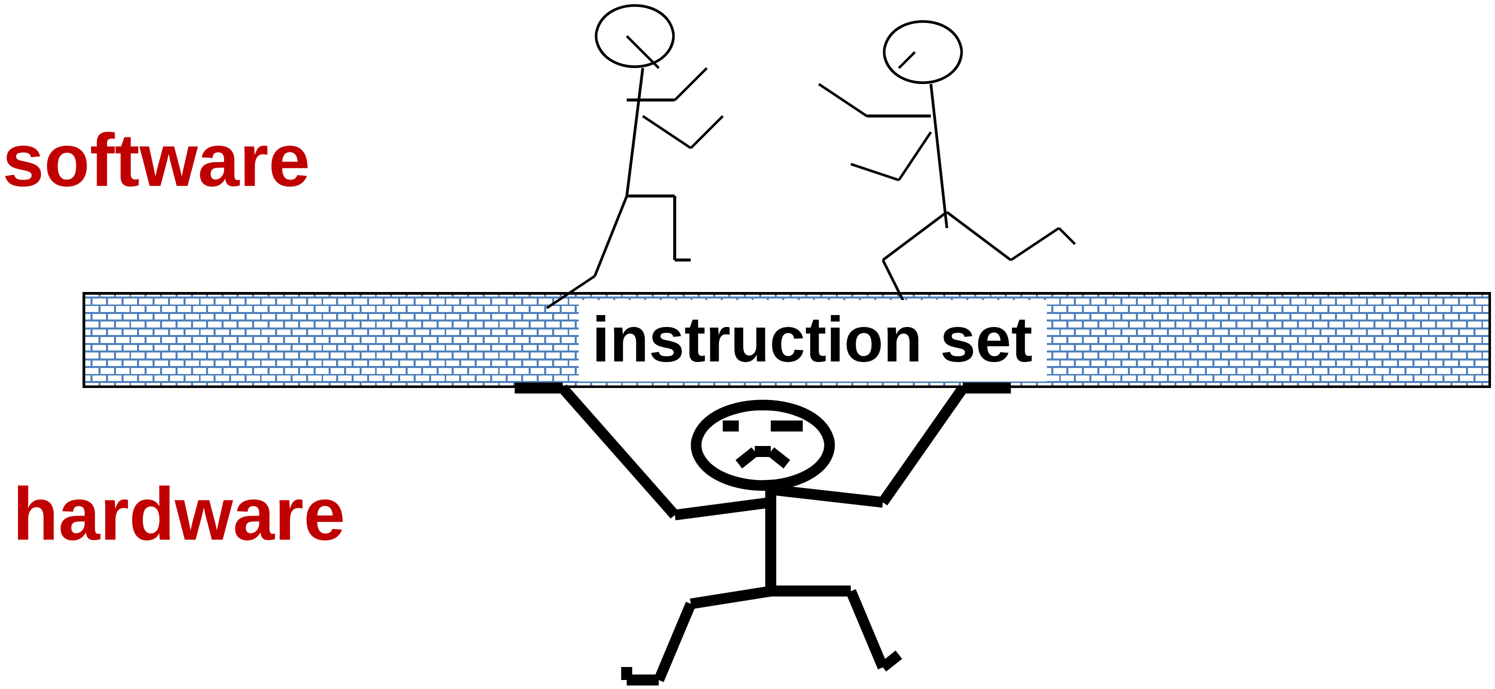
At program completion the data to be output resides in memory

# Output Device Outputs Data



```
00000100010100000000000000000000
00000000010011110000000000000100
00000011111000000000000000001000
```

# Instruction Set Architecture: Critical Interface



Slide adapted from http://inst.eecs.berkeley.edu/~cs252

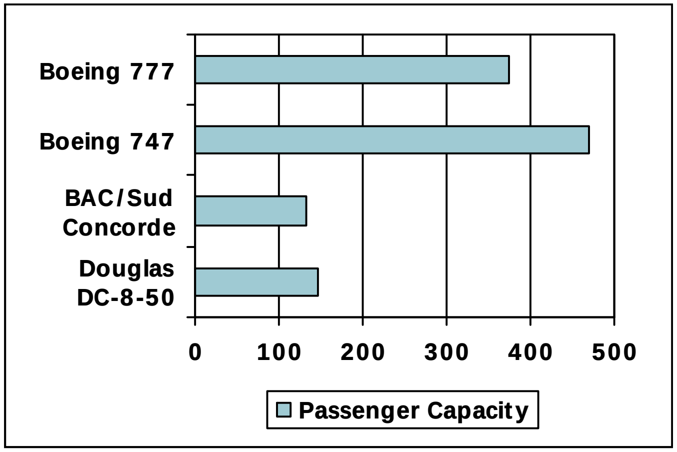
# Instruction Set Architecture (ISA)

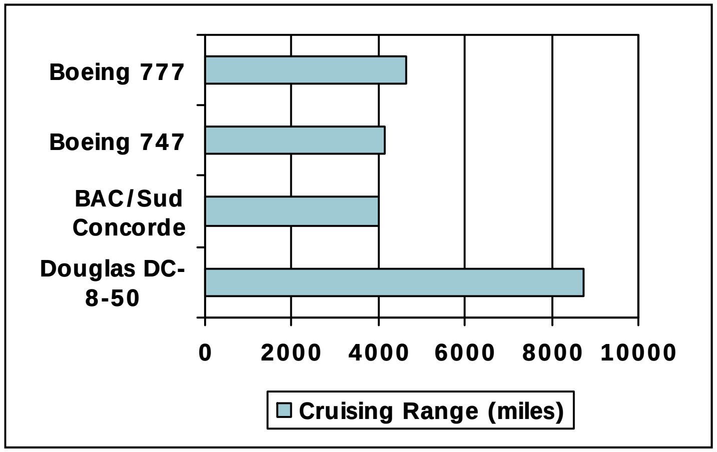
- Actual programmer-visible instruction set
- Boundary between software and hardware
- Decisions regarding:
  - Class of ISA
    - Register memory ISA (memory can be accessed by many different instructions)
    - Load-store ISA (only load/store instructions can access memory)
  - Memory addressing (byte addressable,..)
  - Addressing modes (register, immediate, displacement,…)
  - Instruction operands (size and type of operands)
  - Operations (data transfer, arithmetic/logical, control, floating point,…)
  - Control flow instructions (cond branches, uncond branches, call/return,..)
  - Instruction encoding (fixed length, variable length)

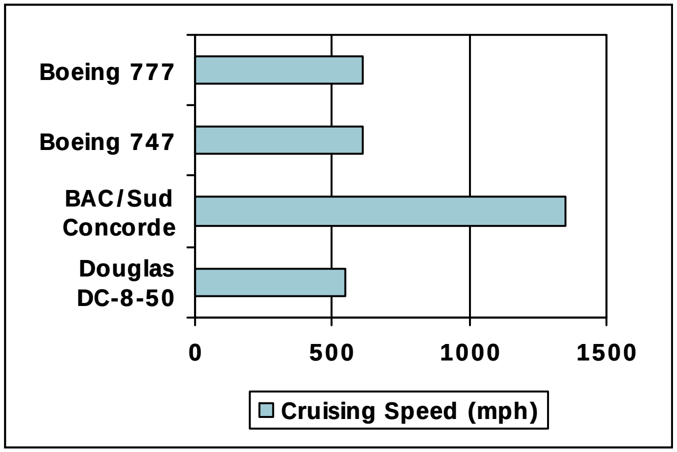
# Defining Performance

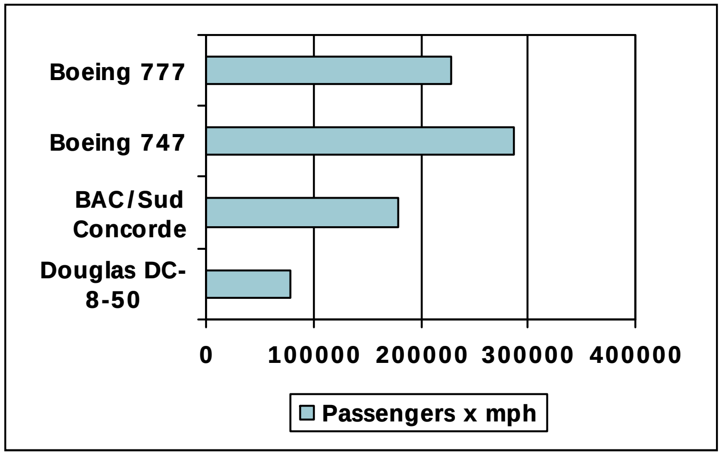
§ 1.6 Performance

- Which airplane has the best performance?









# 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
- We’ll focus on response time for now…

# Relative Performance

- To maximize performance, need to minimize execution time
  - Define Performance = 1/Execution Time
  - "X is $n$ time faster than Y"

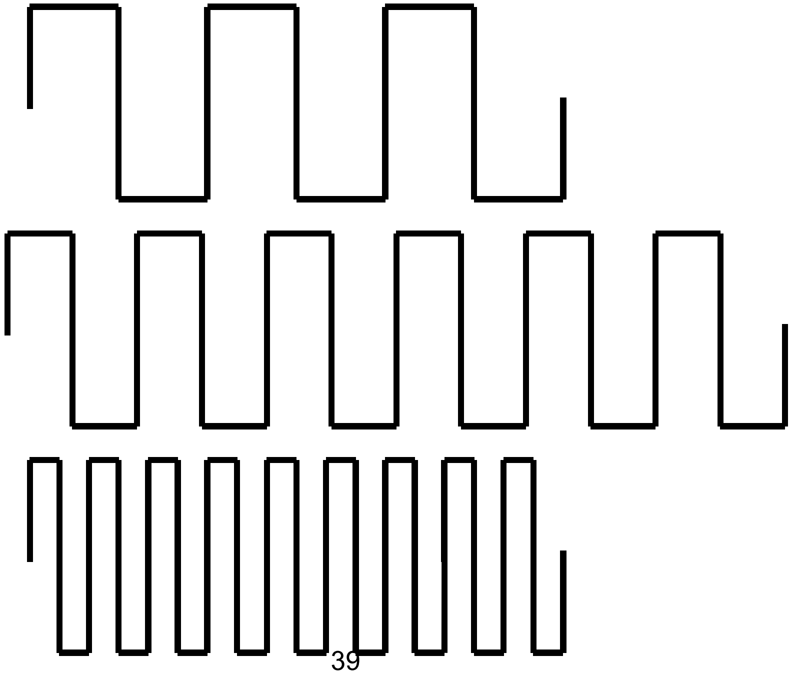
$$\begin{aligned}&\text{Performance}_\text{X}/\text{Performance}_\text{Y}\\&=\text{Execution time}_\text{Y}/\text{Execution time}_\text{X}=n\end{aligned}$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time$_\text{B}$ / Execution Time$_\text{A}$ = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

# CPU Clock (Review)

- Every action is driven by a clock in the CPU
- Clock Cycle Time (CCT) = 1/Clock_Frequency
- From CPU speed, you know time for 1 clock cycle

- Mhz clock = $10^{-6}$ seconds
- Ghz clock = $10^{-9}$ seconds



# Performance Summary

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

$$\text{CPI} = \frac{\text{CPU clock cycles for a program}}{\text{Instruction count}}$$

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Clock cycle time}$$

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI, $T_c$

# Improving Performance Example

- A program runs on computer A with a 2 GHz clock in 10 seconds. What clock rate must computer B run at to run this program in 6 seconds? Unfortunately, to accomplish this, computer B will require 1.2 times as many clock cycles as computer A to run the program.

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{clock rate}_A}$$

$$\text{CPU clock cycles}_A = 10 \text{ sec} \times 2 \times 10^9 \text{ cycles/sec}$$
$$= 20 \times 10^9 \text{ cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{\text{clock rate}_B}$$

$$\text{clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = 4 \text{ GHz}$$

# Using the Performance Equation

- Computers A and B implement the same ISA. Computer A has a clock cycle time of 250 ps and an effective CPI of 2.0 for some program and computer B has a clock cycle time of 500 ps and an effective CPI of 1.2 for the same program. Which computer is faster and by how much?

Each computer executes the same number of instructions, *I*, so

$$\text{CPU time}_A = \text{IC} \times 2.0 \times 250 \text{ ps} = 500 \times \text{IC (ps)}$$

$$\text{CPU time}_B = \text{IC} \times 1.2 \times 500 \text{ ps} = 600 \times \text{IC (ps)}$$

Clearly, A is faster ... by the ratio of execution times

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \times \text{IC}}{500 \times \text{IC}} = 1.2$$

# CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\boxed{\text{Relative frequency}}} \right)$$

# CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

| Class | A | B | C |
|---|---|---|---|
| CPI for class | 1 | 2 | 3 |
| IC in sequence 1 | 2 | 1 | 2 |
| IC in sequence 2 | 4 | 1 | 1 |

- Sequence 1: IC = 5
  - Clock Cycles<br>$= 2 \times 1 + 1 \times 2 + 2 \times 3$<br>$= 10$
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles<br>$= 4 \times 1 + 1 \times 2 + 1 \times 3$<br>$= 9$
  - Avg. CPI = 9/6 = 1.5

# A Simple Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | | | |
|---|---|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 | .5 | .25 |
| Load | 20% | 5 | 1.0 | .4 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 | .2 | .4 |
| | | | $\Sigma =$ 2.2 | **1.6** | **2.0** | **1.95** |

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

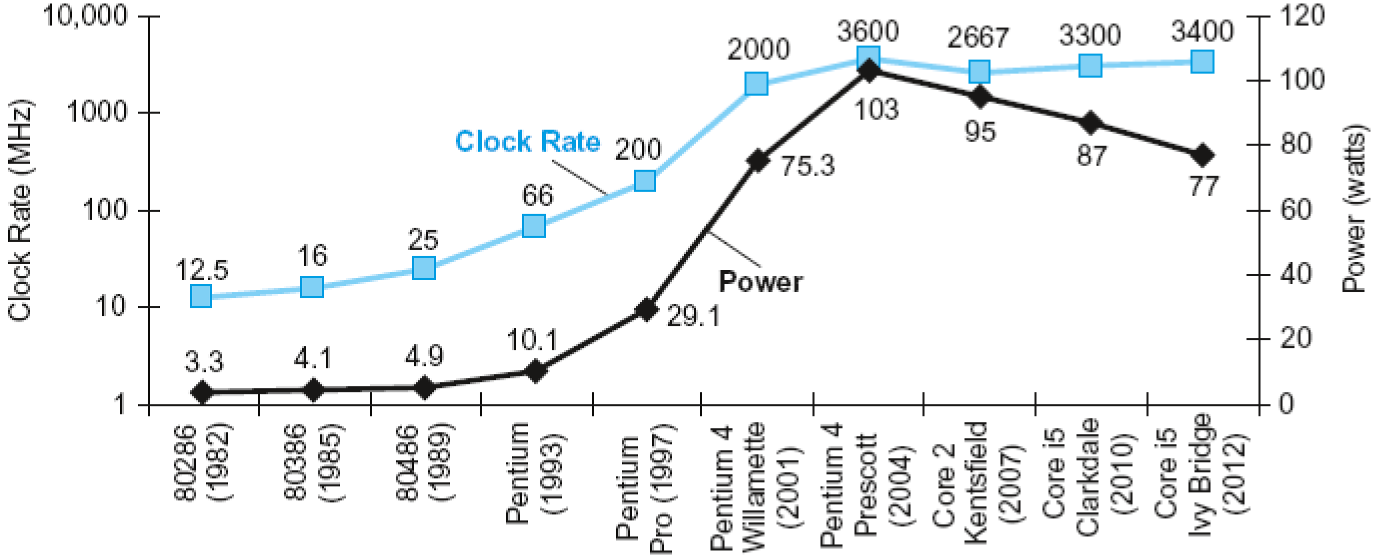
  CPU time new = 1.6 x IC x CCT so 2.2/1.6 means 37.5% faster
- How does this compare with using branch prediction to shave a cycle off the branch time?

  CPU time new = 2.0 x IC x CCT so 2.2/2.0 means 10% faster
- What if two ALU instructions could be executed at once?

  CPU time new = 1.95 x IC x CCT so 2.2/1.95 means 12.8% faster

# Power Trends

§ 1.7 The Power Wall



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

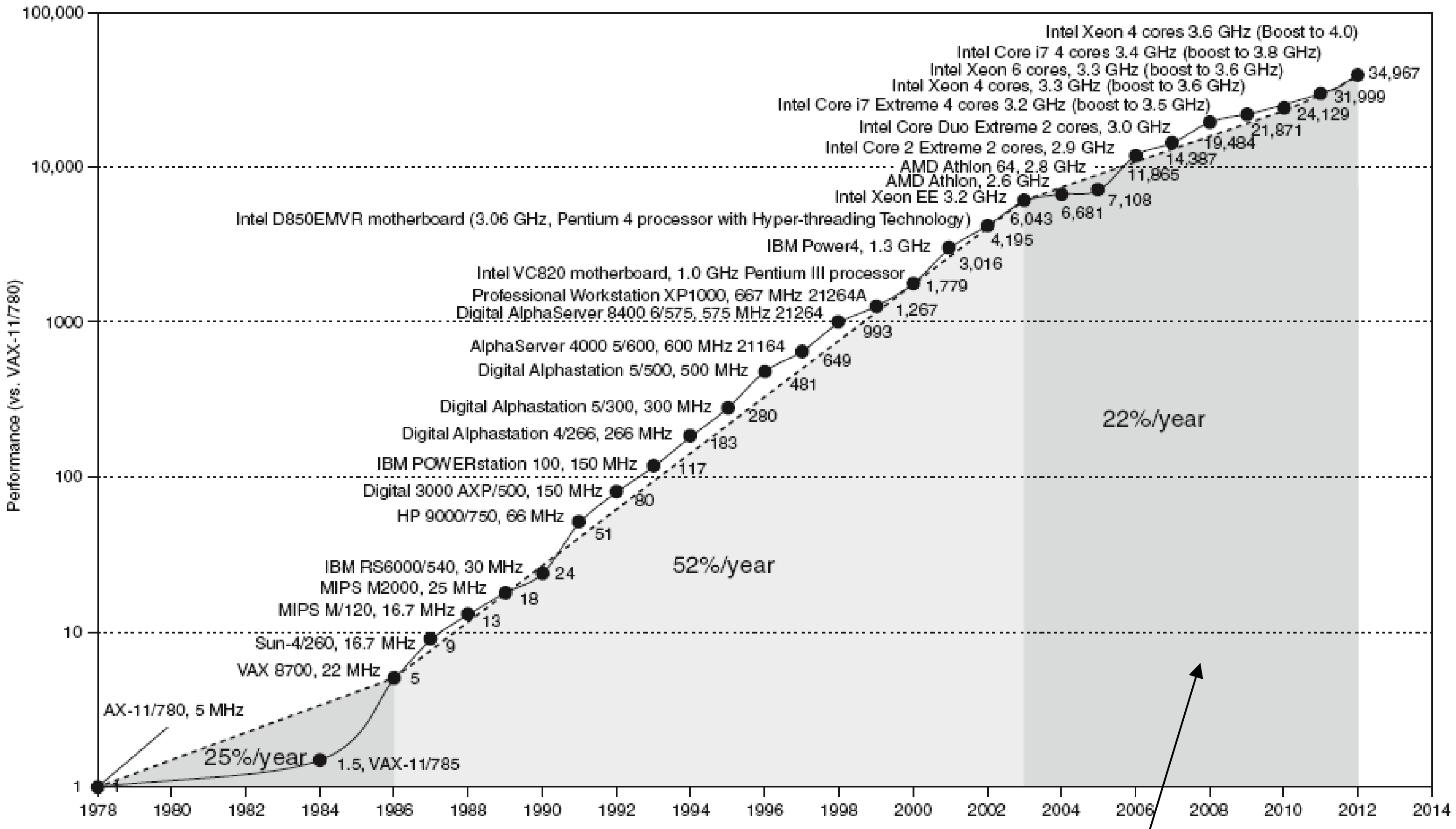
$\times 30$

$5\text{V} \rightarrow 1\text{V}$

$\times 1000$

# Uniprocessor Performance

§ 1.8 The Sea Change: The Switch to Multiproces



Constrained by power, instruction-level parallelism, memory latency

# Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

§ 1.9 Concluding Remarks