CENG 3420
Computer Organization and Design

Lecture 01: Introduction

Bei Yu

香港中文大學
The Chinese University of Hong Kong

Grading Information

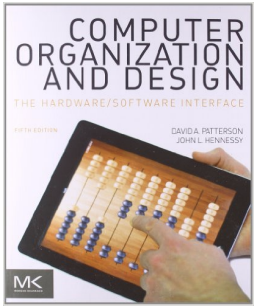
- Grade determinates
  - Attendance **5%**
  - Homework **10%**
  - Two Quizzes (Feb. 18 & Mar. 21) **15%**
  - Three Labs (Individual project) **30%**
  - Final Exam **40%**

- Late submission per day is subject to 10% of penalty.
- What's new: Q&A bonus 😀
- A student must gain at least 40% of the full marks in each part in order to pass the course.

General References

- Textbook:

Computer Organization and Design, 5th Edition ©2013



  - Soft copy, amazon.cn, or amazon.com

- Manuals:
  - LC-3 Instruction Set Architecture (ISA)
  - Lab tutorials (slides)

- Slides:
  - on the course web page before lecture
  - summary may be uploaded afterwards

Course Administration

- Instructor: **Bei** Yu byu@cse.cuhk.edu.hk
  - **Office**: SHB 914
  - **Office Hrs**: H14:00-16:00

- TA:
  - Wen Zong wzong@cse.cuhk.edu.hk
  - Yichen Wang xblwyc@163.com

Course Contents

- Introduction to the major components of a computer system, how they function together in executing a program.
- Introduction to CPU datapath and control unit design
- Introduction to techniques to improve performance and energy-efficiency of computer systems
- Introduction to multiprocessor architecture

To learn what determines the capabilities and performance of computer systems and to understand the interactions between the computer's architecture and its software so that future software designers (compiler writers, operating system designers, database programmers, application programmers, …) can achieve the best cost-performance trade-offs and so that future architects understand the effects of their design choices on software.

Why Learn This Stuff?

- You want to call yourself a “computer scientist/engineer”
- You want to build HW/SW people use (so need performance/power)
- You need to make a purchasing decision or offer “expert” advice
- Both hardware and software affect performance/power
  - Algorithm determines number of source-level statements
  - Language/compiler/architecture determine the number of machine-level instructions
  - Processor/memory determine how fast and how power-hungry machine-level instructions are executed

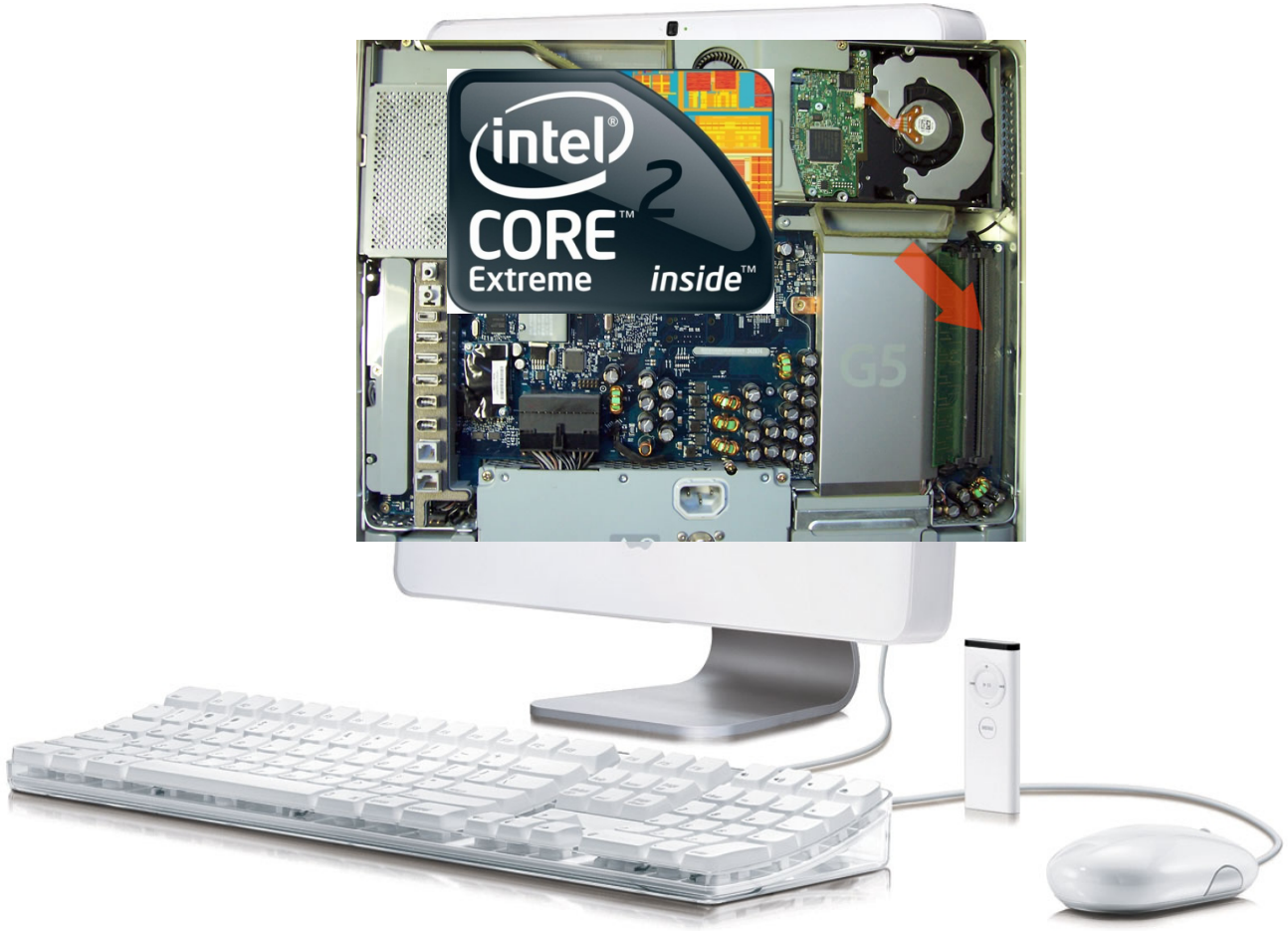
What You Should Already Know

- Basic logic design & machine organization
  - logical minimization, FSMs, component design
  - processor, memory, I/O
- Create, assemble, run, debug programs in an assembly language
  - Will be introduced in tutorial
- Create, compile, and run C (C++, Java) programs
- Create, organize, and edit files and run programs on Unix/Linux
- Create, simulate, and debug hardware structures
  - will be introduced in tutorial

Computer Organization and Design

- This course is all about how computers work
- But what do we mean by a computer?
  - Different types: embedded, laptop, desktop, server
  - Different uses: automobiles, graphics, finance, genomics…
  - Different manufacturers: Intel, Apple, IBM, Sony, Oracle…
  - Different underlying technologies and different costs !
- Analogy: Consider a course on "automotive vehicles"
  - Many similarities from vehicle to vehicle (e.g., wheels)
  - Huge differences from vehicle to vehicle (e.g., gas vs. electric)
- Best way to learn:
  - Focus on a specific instance and learn how it works
  - While learning general principles and historical perspectives

A Computer



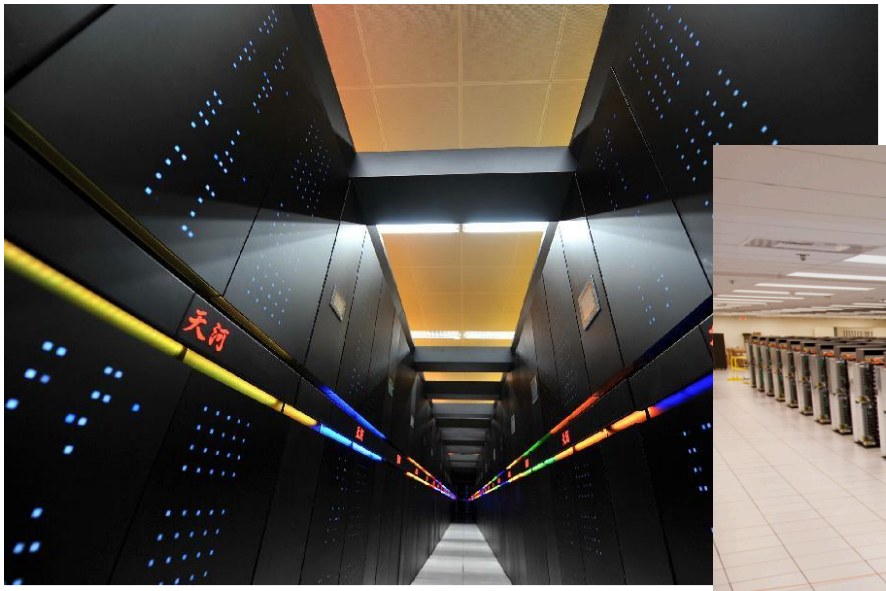
Are there other kind of computers?

Classes of Computers

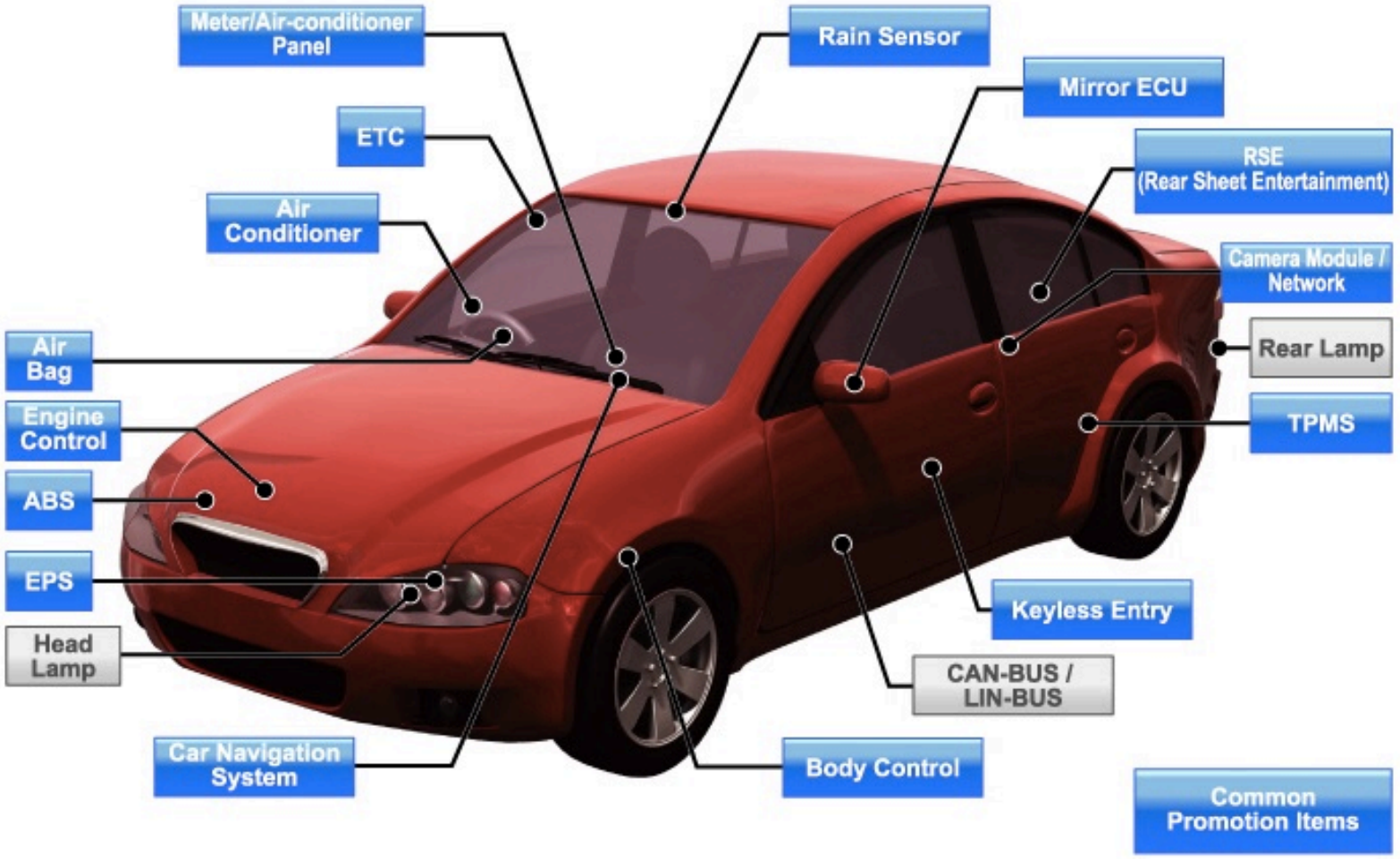
- Desktop computers
  - Designed to deliver good performance to a single user at low cost usually executing 3rd party software, usually incorporating a graphics display, a keyboard, and a mouse
- Servers
  - Used to run larger programs for multiple, simultaneous users typically accessed only via a network 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

Supercomputers

- Tianhe-2 (MilkyWay-2, 天河-2)
  - Over 3 million cores
  - Power: 17.6 MW (24 MW with cooling)
  - Speed: 33.86 PFLOPS (peta = 10^{15})

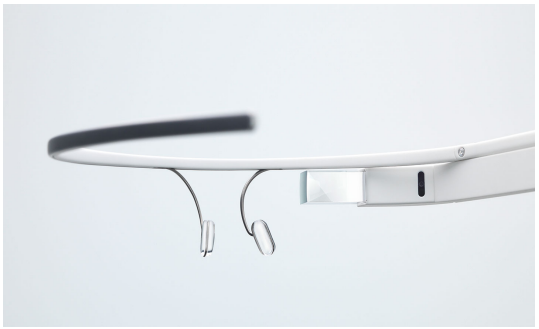


Embedded Computers in You Car



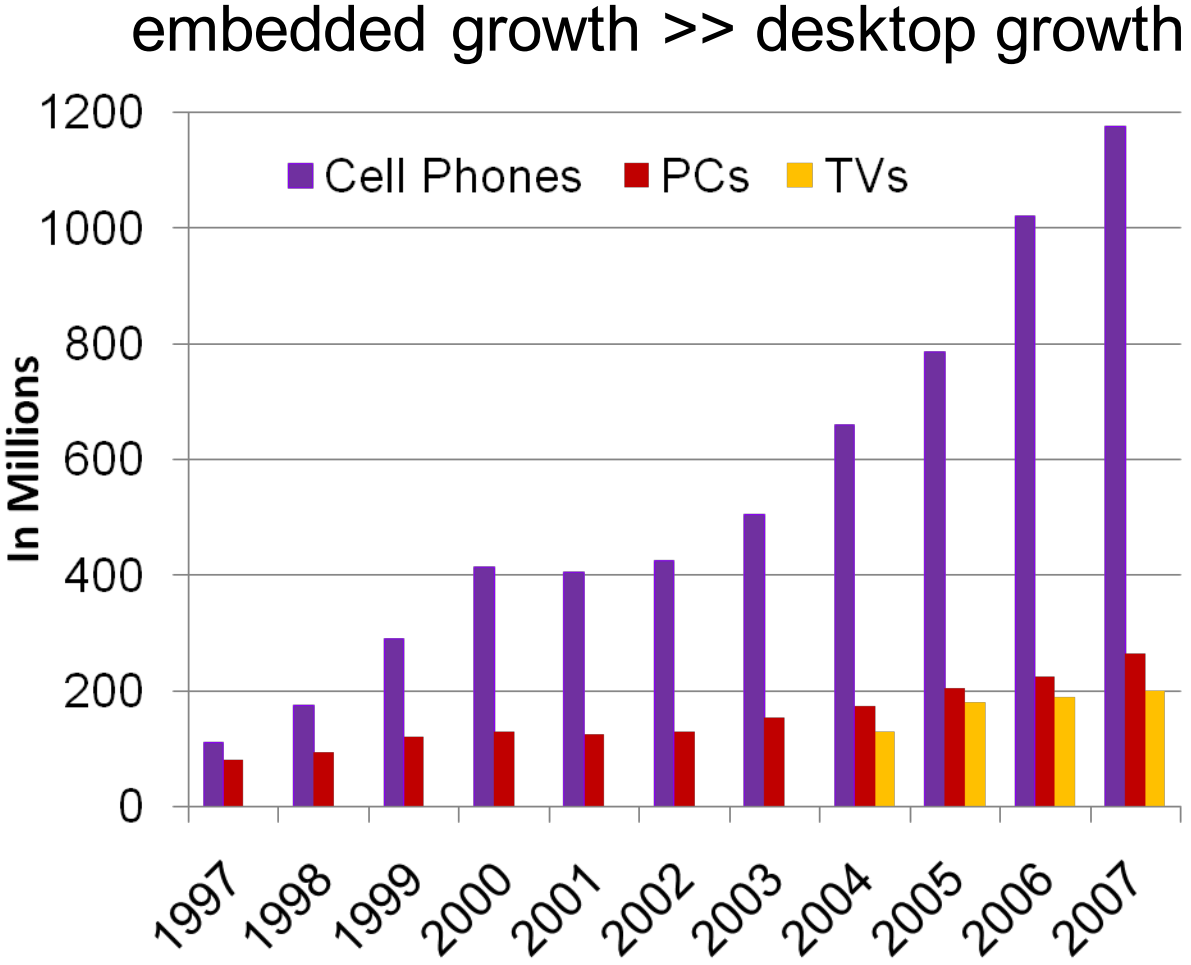
PostPC Era

- Personal Mobile Device (PMD)
  - Battery-operated device with wireless connectivity



- Warehouse Scale Computer (WSC)
  - Datacenter containing hundreds of thousands of servers providing software as a service (SaaS)

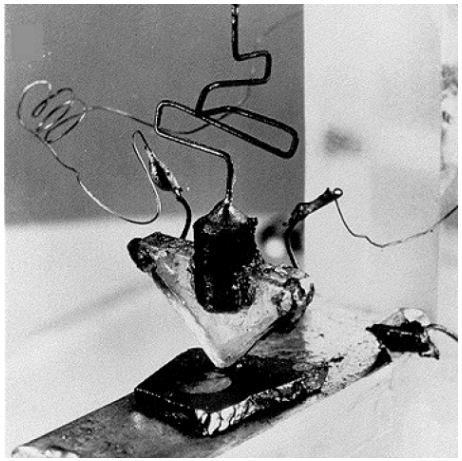
Growth in Cell Phone Sales (Embedded)



- Where else are embedded processors found?

The Evolution of Computer Hardware

- When was the first transistor invented?

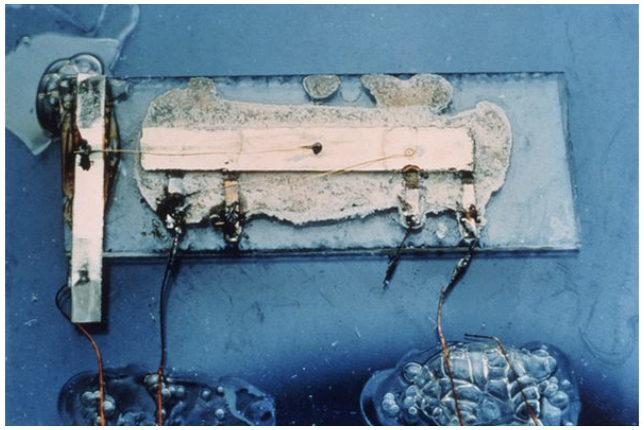


1947 - the bi-polar transistor – by Bardeen *et.al* at Bell Laboratories

UNIVAC I (Universal Automatic Computer) – the first commercial computer in USA

The Evolution of Computer Hardware

- When was the first IC (integrated circuit) invented?

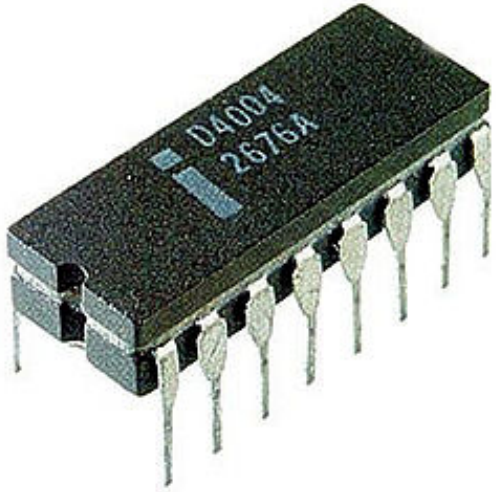


1958, by Jack Kilby@Texas Instruments, by hand, several transistors, resistors and capacitors on a single substrate

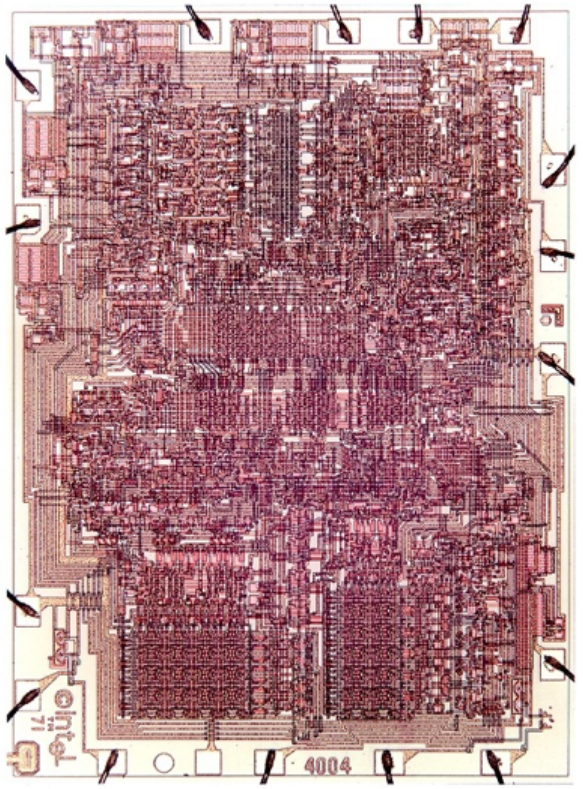
IBM System/360, 2MHz, 128KB ~ 256KB

The Evolution of Computer Hardware

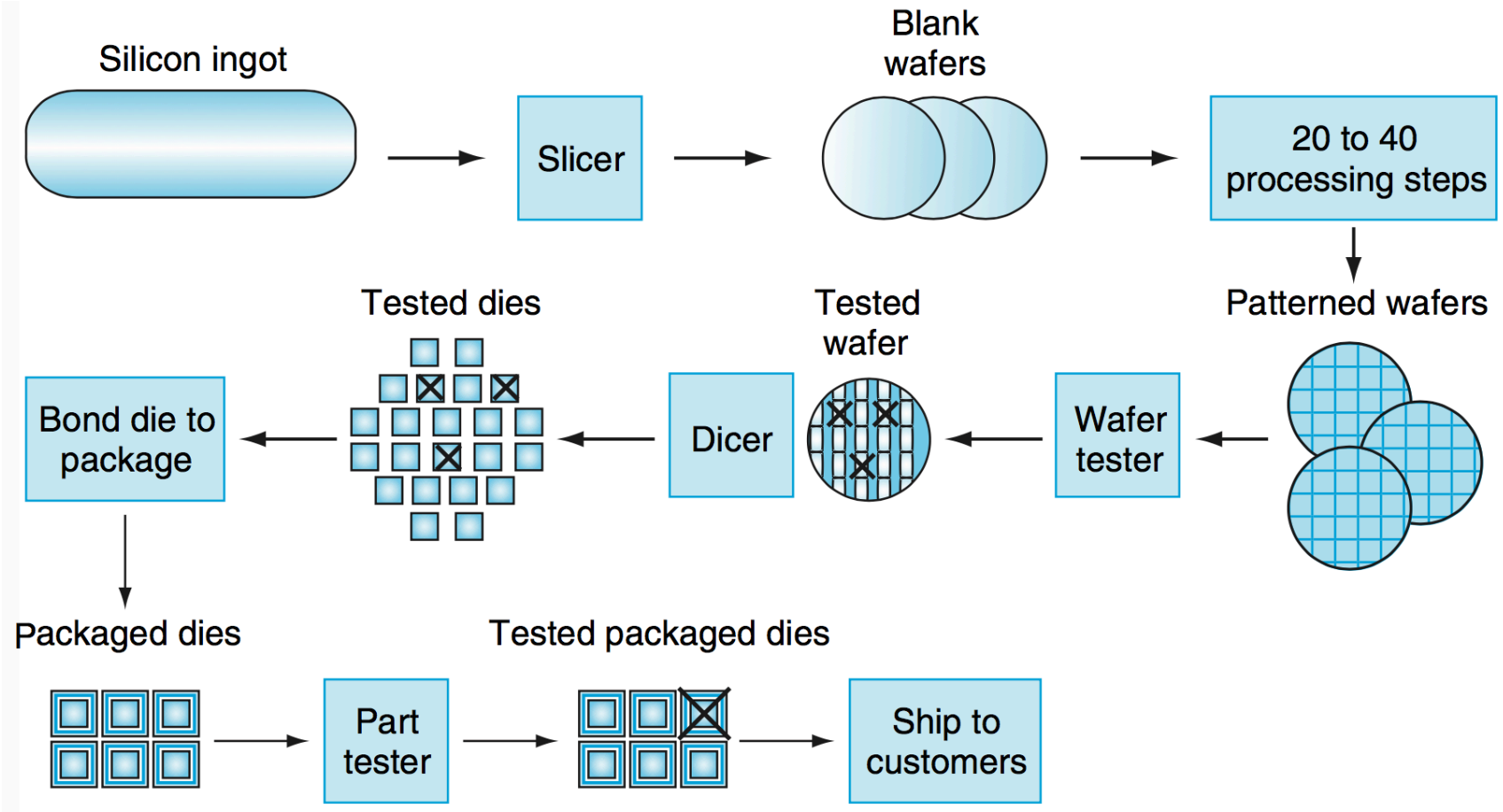
- When was the first Microprocessor?



1971, Intel 4004

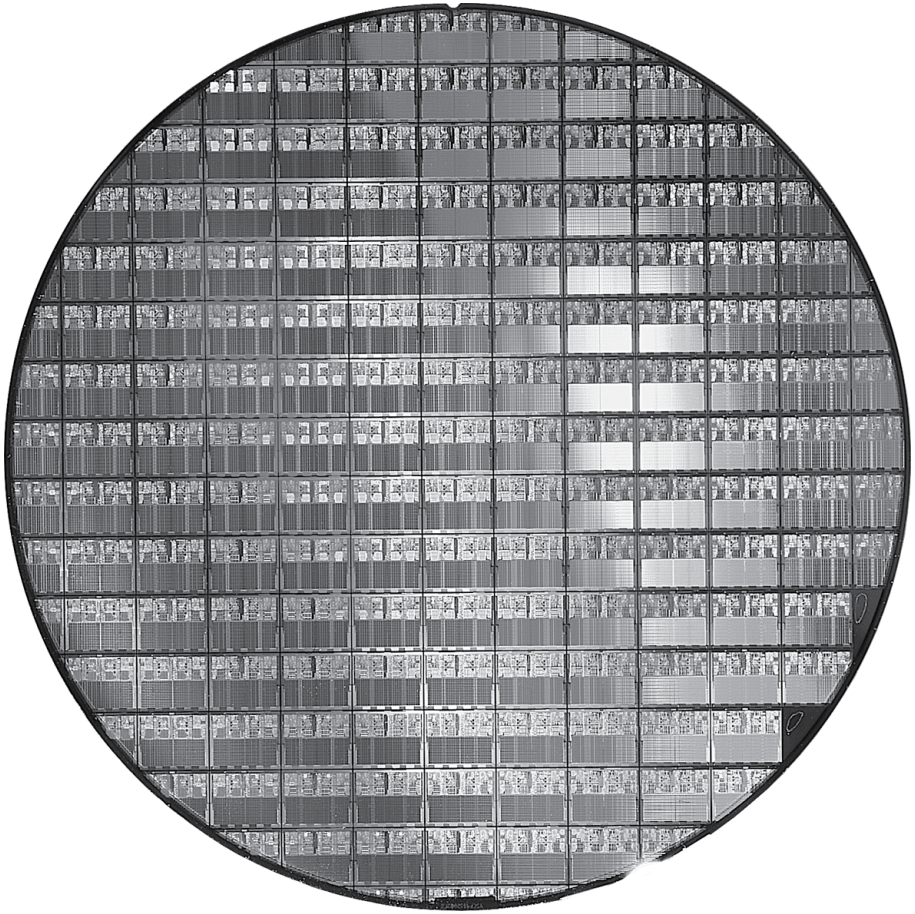


The IC Manufacturing Process



- Yield: proportion of working dies per wafer

AMD Opteron X2 Wafer



300mm wafer, 117 chips, 90nm technology

Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

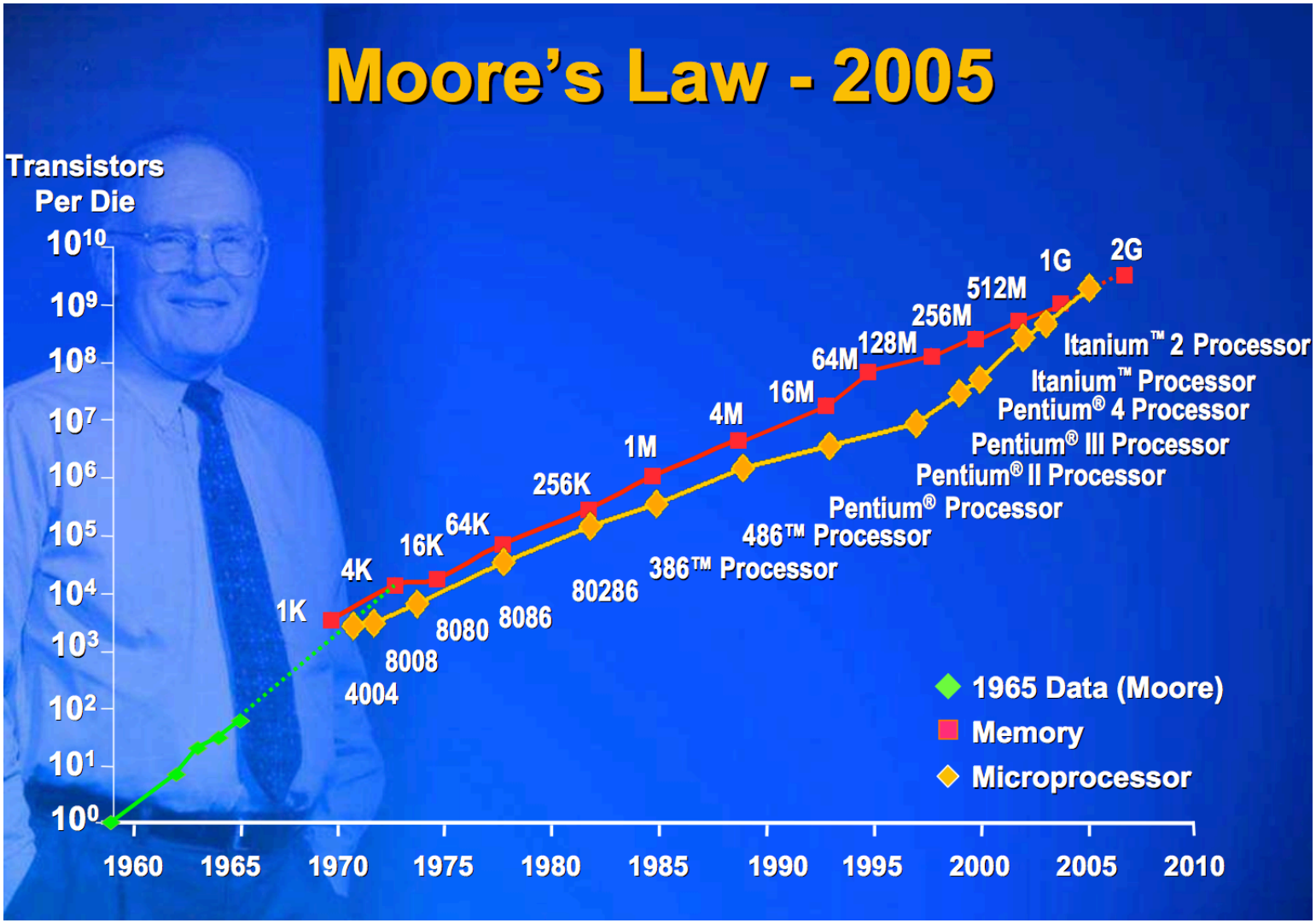
$$\text{Yield} = \frac{1}{(1+(\text{Defects per area} \times \text{Die area}/2))^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

Impacts of Advancing Technology

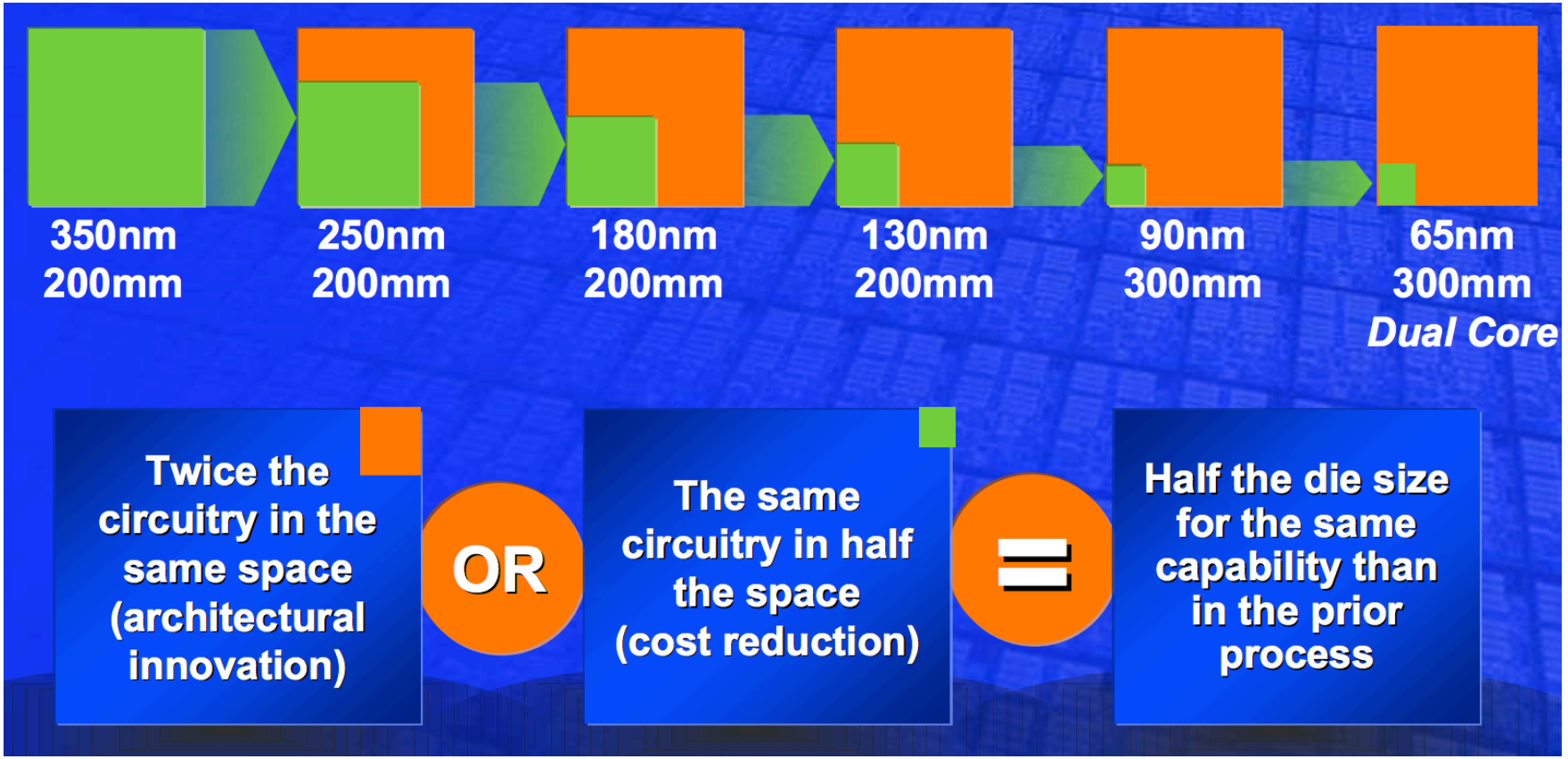
- Processor
  - logic capacity: increases about 30% per year
  - performance: 2x every 1.5 years
- Memory
  - DRAM capacity: 4x every 3 years, about 60% per year
  - memory speed: 1.5x every 10 years
  - cost per bit: decreases about 25% per year
- Disk
  - capacity: increases about 60% per year

Moore's Law for CPUs and DRAMs

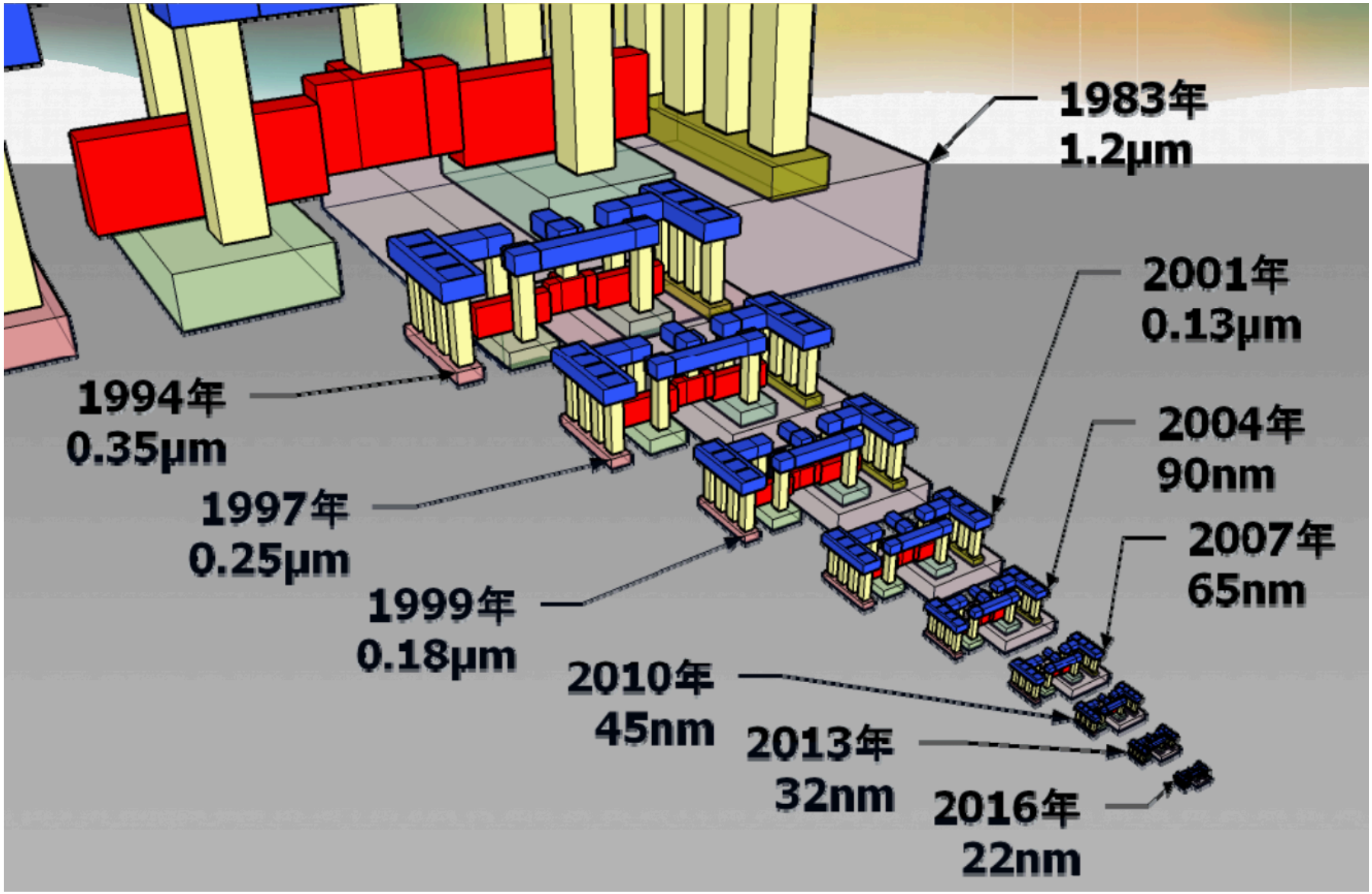


From: "Facing the Hot Chips Challenge Again", Bill Holt, Intel, presented at Hot Chips 17, 2005.

Main driver: device scaling ...



From: "Facing the Hot Chips Challenge Again", Bill Holt, Intel, presented at Hot Chips 17, 2005.

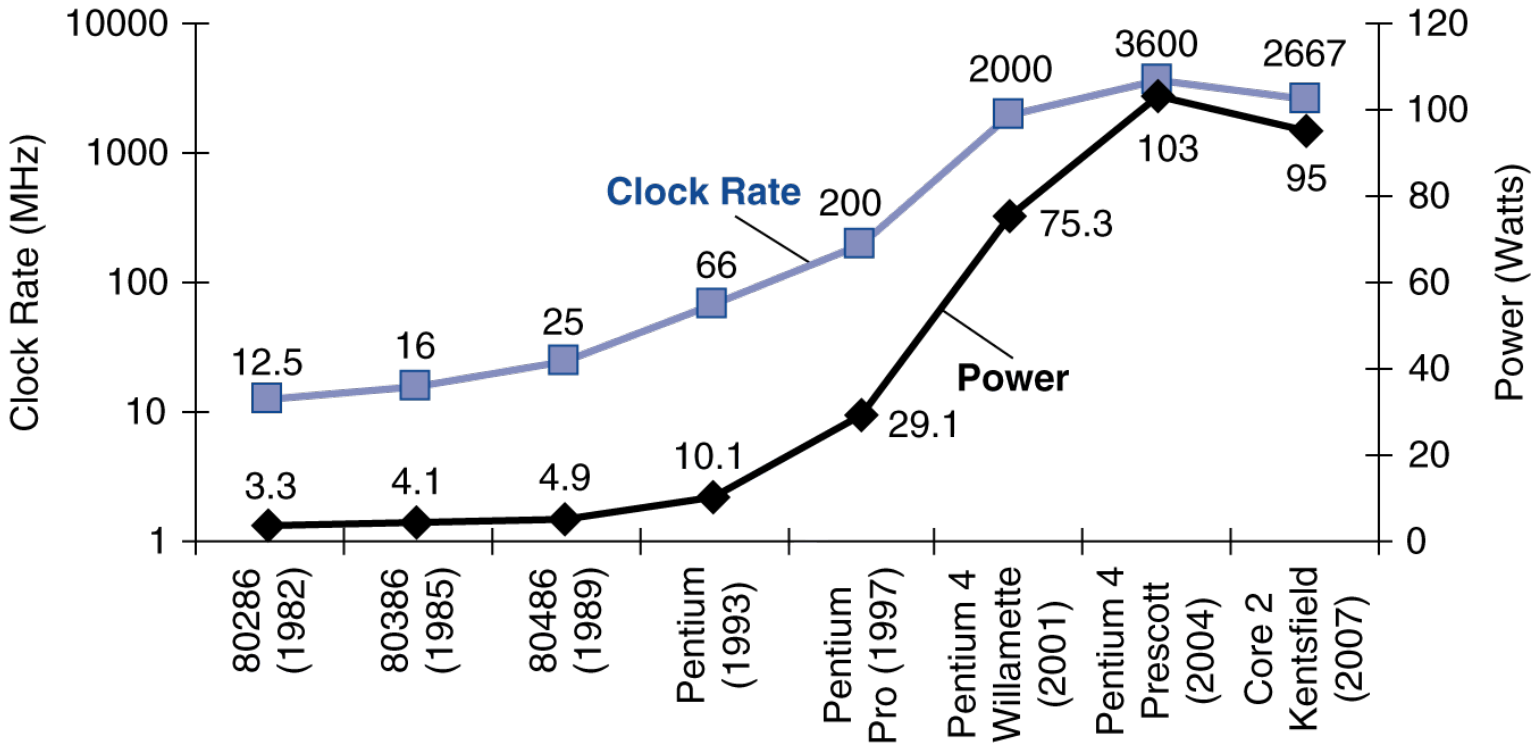

1983年
1.2μm
2001年
0.13μm
1994年
0.35μm
2004年
90nm
1997年
0.25μm
2007年
65nm
1999年
0.18μm
2010年
45nm
2013年
32nm
2016年
22nm

Technology Scaling Road Map (ITRS)

Year	2004	2006	2008	2010	2012
Feature size (nm)	90	65	45	32	22
Intg. Capacity (BT)	2	4	6	16	32

- Fun facts about 45nm transistors
  - 30 million can fit on the head of a pin
  - You could fit more than 2,000 across the width of a human hair
  - If car prices had fallen at the same rate as the price of a single transistor has since 1968, a new car today would cost about 1 cent

Highest Clock Rate of Intel Processors



- Due to process improvements
- Deeper pipeline
- Circuit design techniques

What if the exponential increase had kept up? Why not?

EX: Power Issue

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

- Example: for a simple processor, if capacitive load is reduced by 15%, voltage is reduced by 15%, maintain the same frequency, how much power consumption can be reduced?

Note: here we only consider dynamic power, but not static power

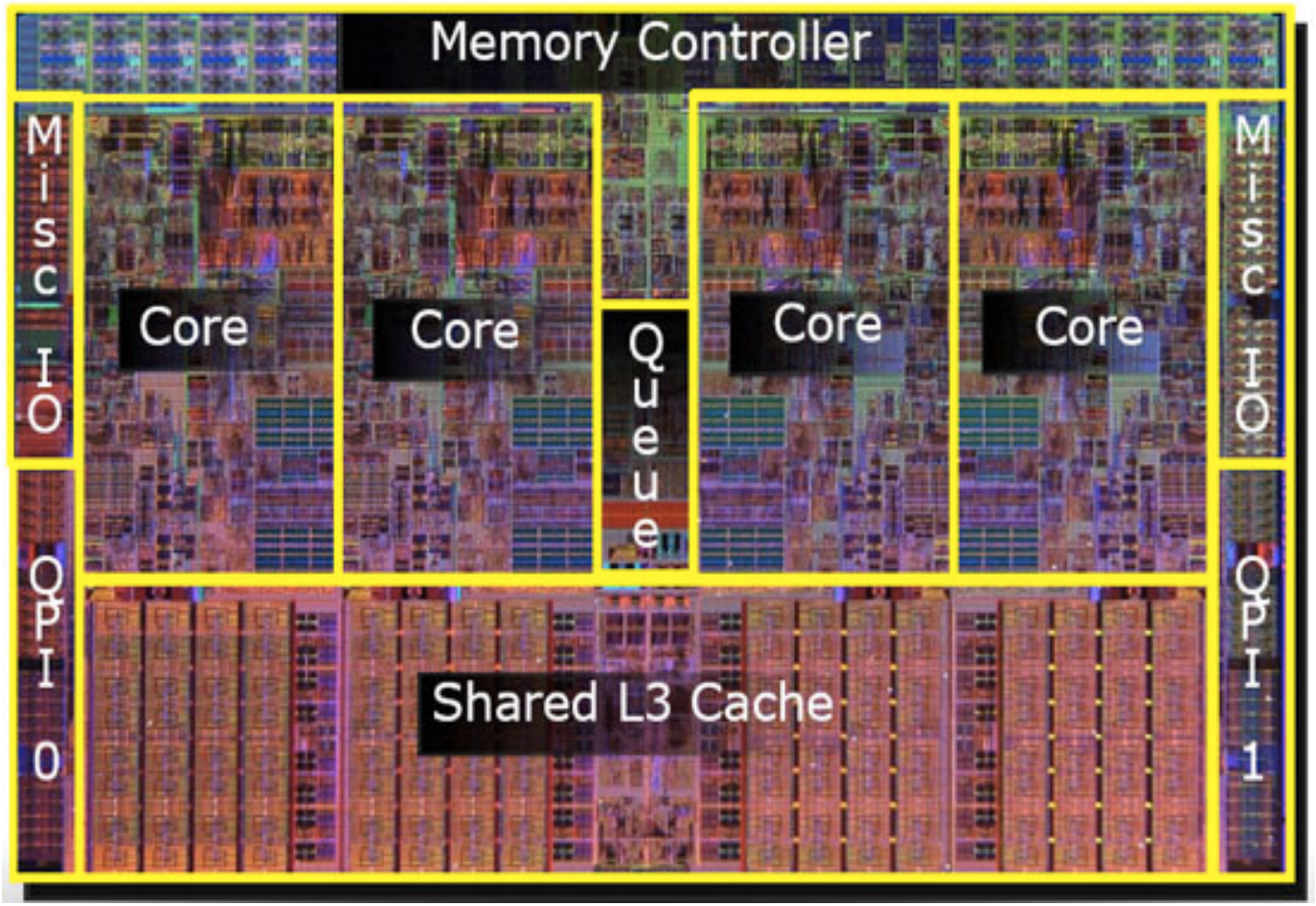
A Sea Change is at Hand

- The power challenge has forced a change in the design of microprocessors
  - Since 2002 the rate of improvement in the response time of programs on desktop computers has slowed from a factor of 1.5 per year to less than a factor of 1.2 per year
- As of 2006 all desktop and server companies are shipping microprocessors with multiple processors – cores – per chip

Product	AMD Barcelona	Intel Nehalem	IBM Power 6	Sun Niagara 2
Cores per chip	4	4	2	8
Clock rate	~2.5 GHz	~2.5 GHz	4.7 GHz	1.4 GHz
Power	120 W	~100 W	~100 W	94 W

- Plan of record is to ~~double the number of cores~~ per chip per generation (about every two years) add two cores

Intel Core i7 Processor



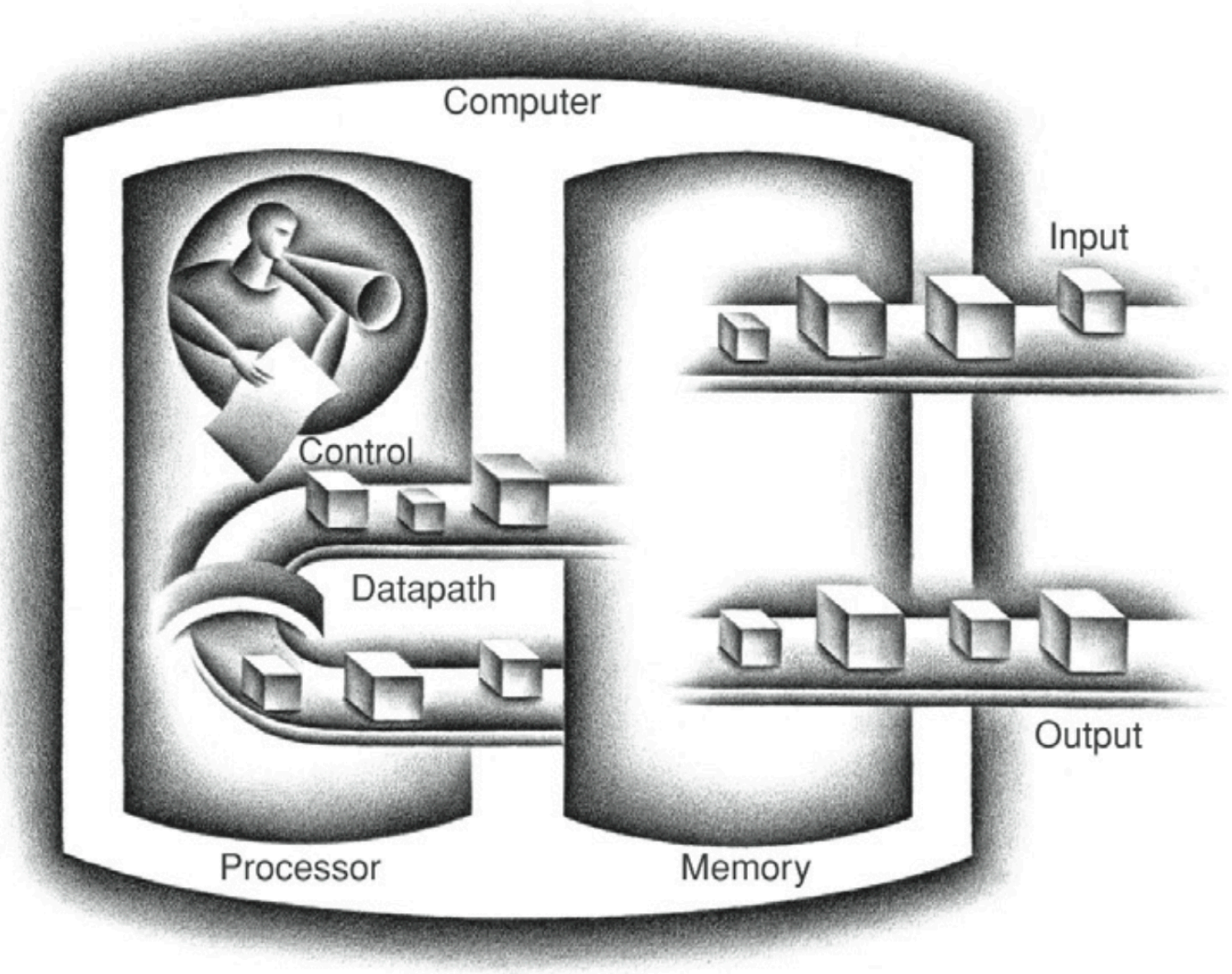
45nm technology, 18.9mm x 13.6mm, 0.73billion transistors, 2008

What is a Computer?

- Components:
  - processor (datapath, control)
  - input (mouse, keyboard)
  - output (display, printer)
  - memory (cache (SRAM), main memory (DRAM), disk drive, CD/DVD)
  - network

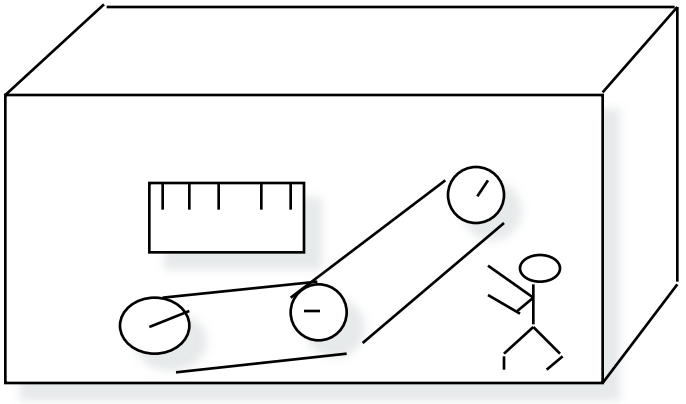
- Our primary focus: the processor (datapath and control) and its interaction with memory systems
  - Implemented using tens/hundreds of millions of transistors
  - Impossible to understand by looking at each transistor
  - We need abstraction!

Major Components of a Computer



Machine Organization

- Capabilities and performance characteristics of the principal Functional Units (FUs)
  - e.g., register file, ALU, multiplexors, memories, ...
- The ways those FUs are interconnected
  - e.g., buses
- Logic and means by which information flow between FUs is controlled

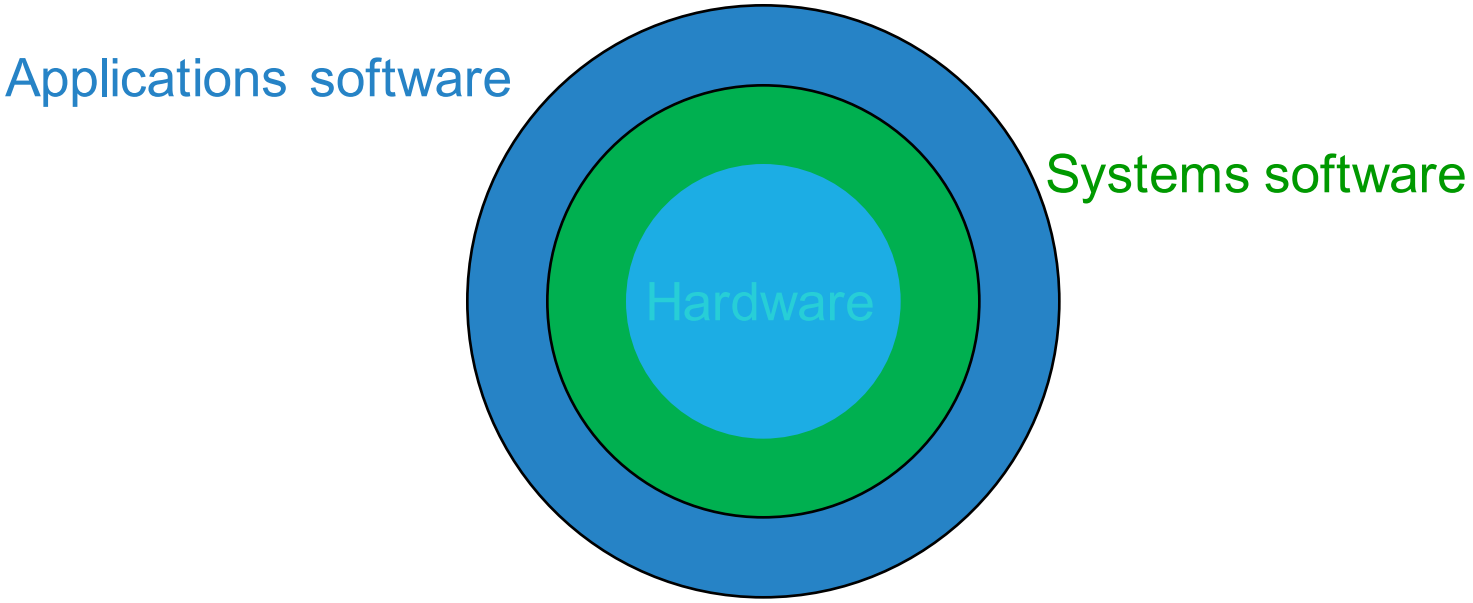


- The machine's Instruction Set Architecture (ISA)
- Register Transfer Level (RTL) machine description

Processor Organization

- Control needs to have circuitry to
  - Decide which is the next instruction and input it from memory
  - Decode the instruction
  - Issue signals that control the way information flows between datapath components
  - Control what operations the datapath's functional units perform
- Datapath needs to have circuitry to
  - Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file)
  - Interconnect the functional units so that the instructions can be executed as required
  - Load data from and store data to memory

Below the Program



- System software
  - Operating system – supervising program that interfaces the user's program with the hardware (e.g., Linux, iOS, Windows)
    - Handles basic input and output operations
    - Allocates storage and memory
    - Provides for protected sharing among multiple applications
  - Compiler – translate programs written in a high-level language (e.g., C, Java) into instructions that the hardware can execute

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

You can become programmers programming programs that program programs!

Below the Program

- High-level language program (in C)

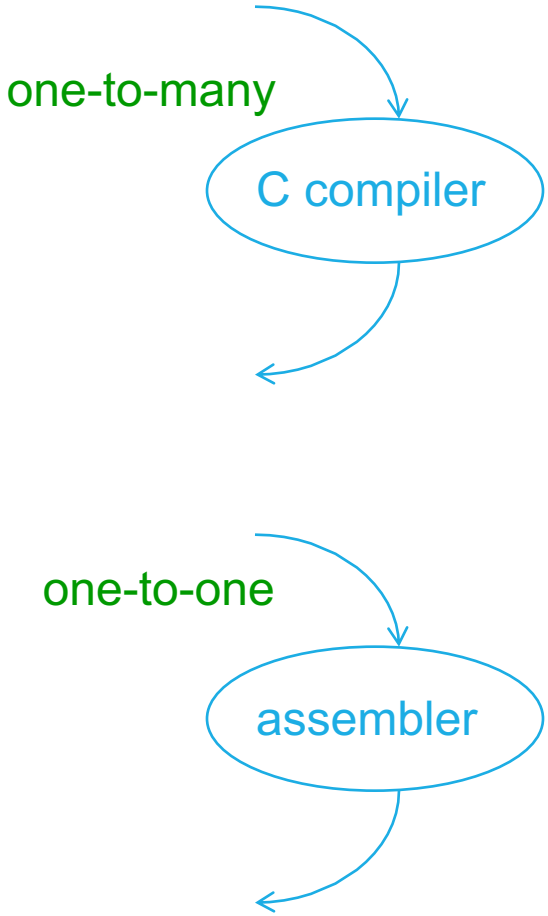
```
swap (int v[], int k)
(int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
)
```

- Assembly language program (for MIPS)

```
swap:  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
```

- Machine (object) code (for MIPS)

```
000000 00000 00101 00010000010000000
000000 00100 00010 00010000000100000
 . . .
```



Below the Program

- High-level language program (in C)

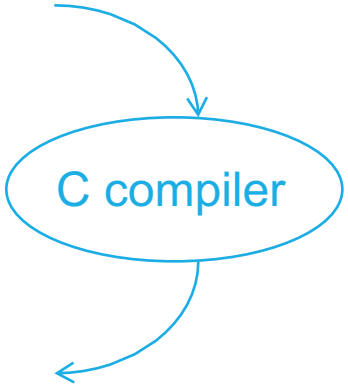
```
swap (int v[], int k)
. . .
```

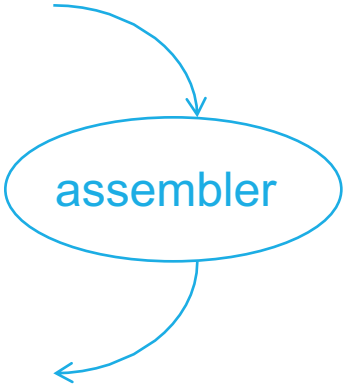
- Assembly language program (for MIPS)

```
swap:   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
```

- Machine (object) code (for MIPS)

```
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
```





Max # of operations?

Below the Program

- High-level language program (in C)

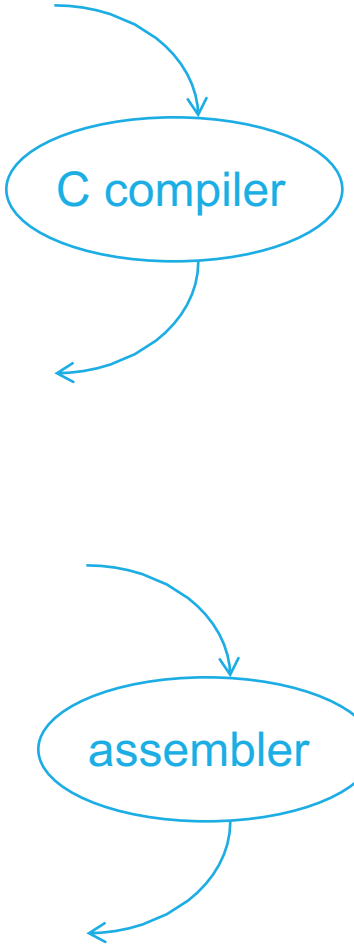
```
swap (int v[], int k)
. . .
```

- Assembly language program (for MIPS)

```
swap:   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
```

- Machine (object) code (for MIPS)

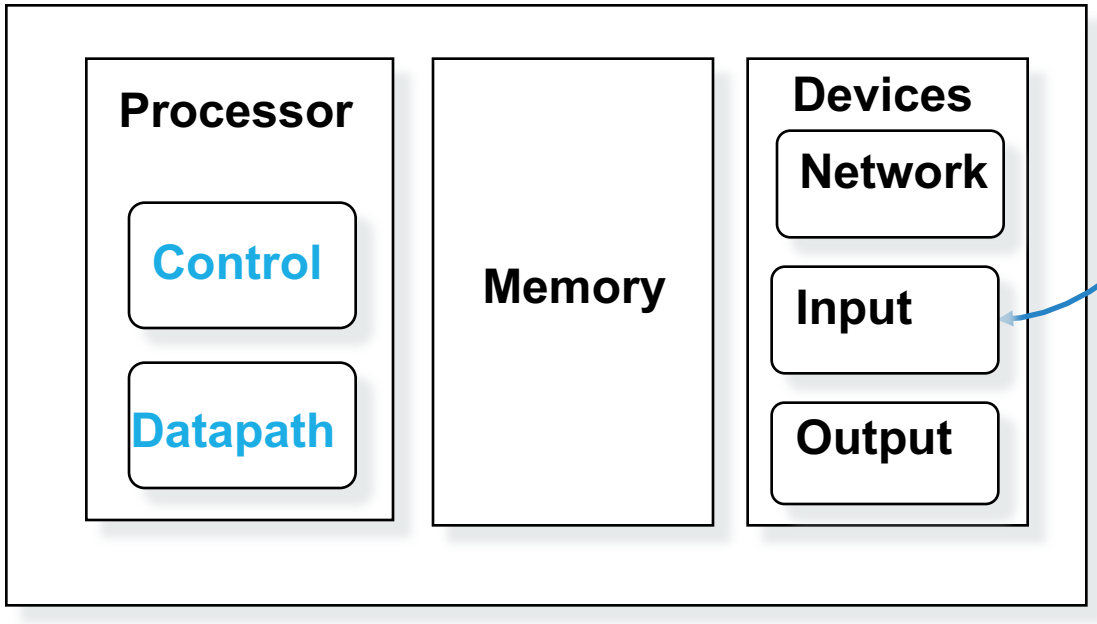
```
000000 00000 00101 00010000010000000
000000 00100 00010 00010000000100000
100011 00010 01111 00000000000000000
100011 00010 10000 00000000000000100
101011 00010 10000 00000000000000000
101011 00010 01111 00000000000000100
000000 11111 00000 00000000000001000
```



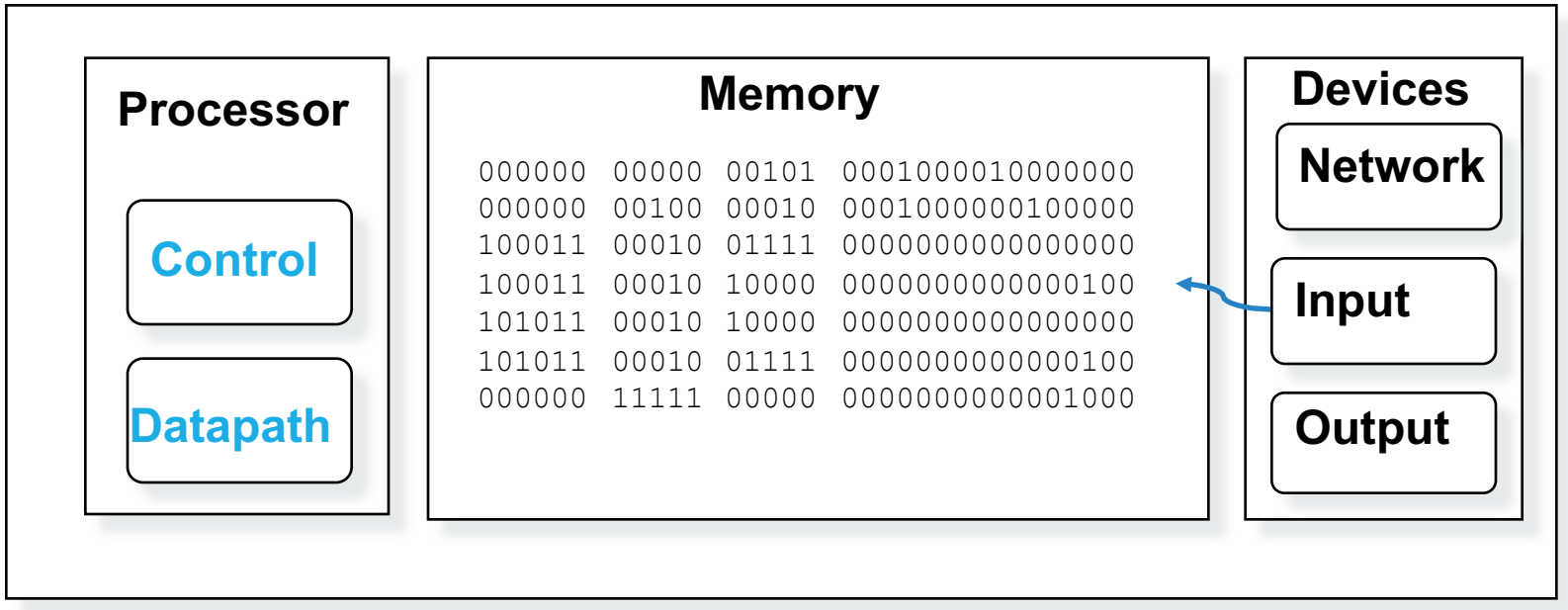
Max # of operations?

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
```

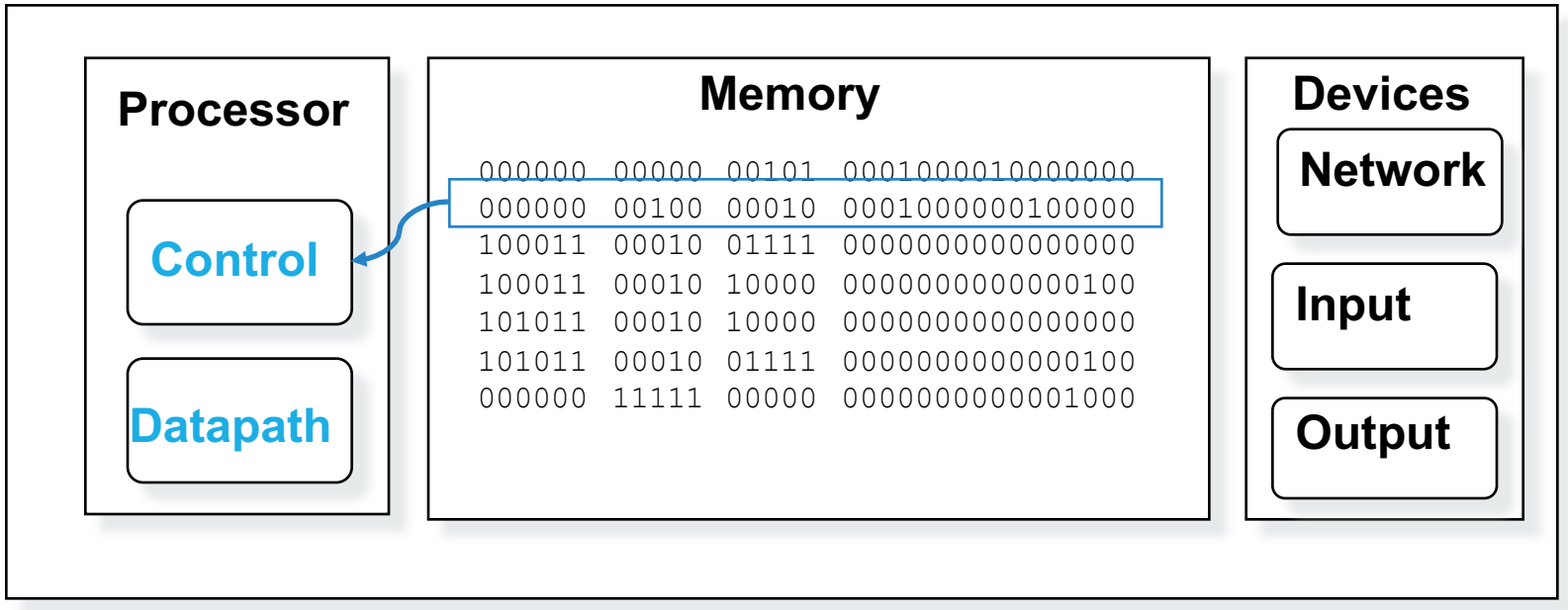


Object Code Stored in Memory



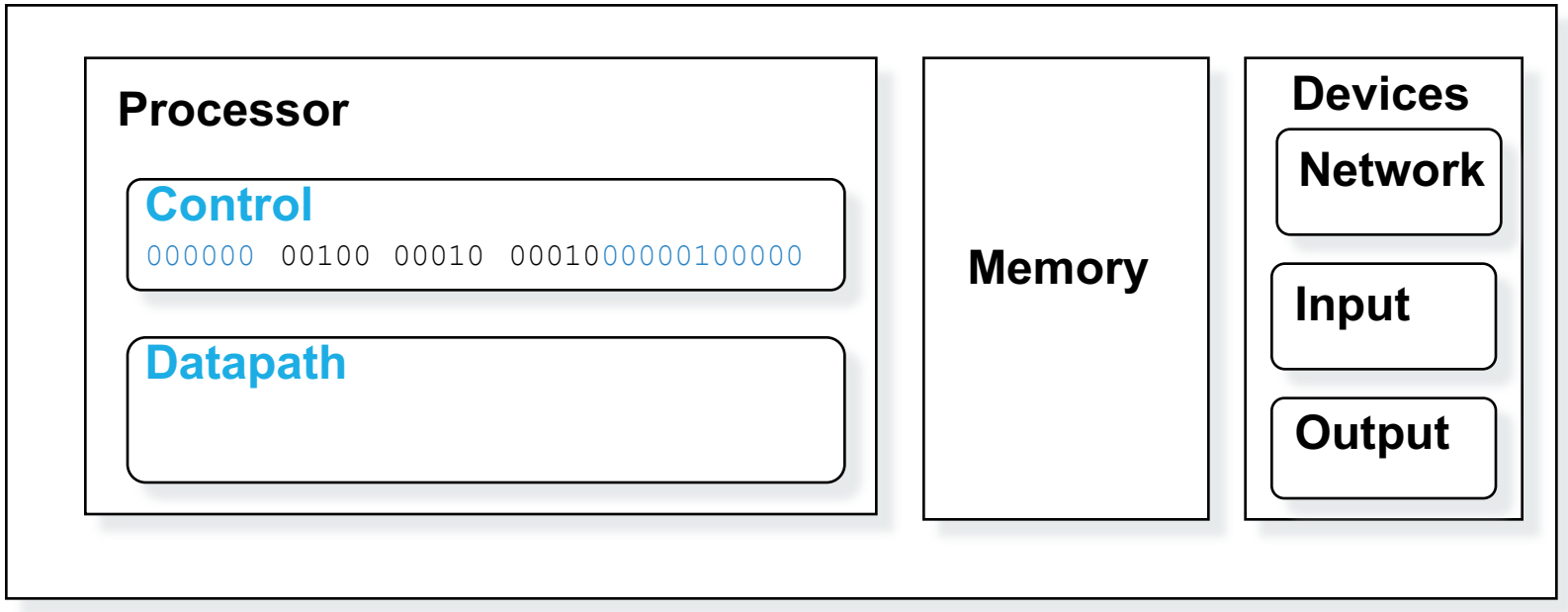
Processor Fetches an Instruction

Processor fetches an instruction from memory



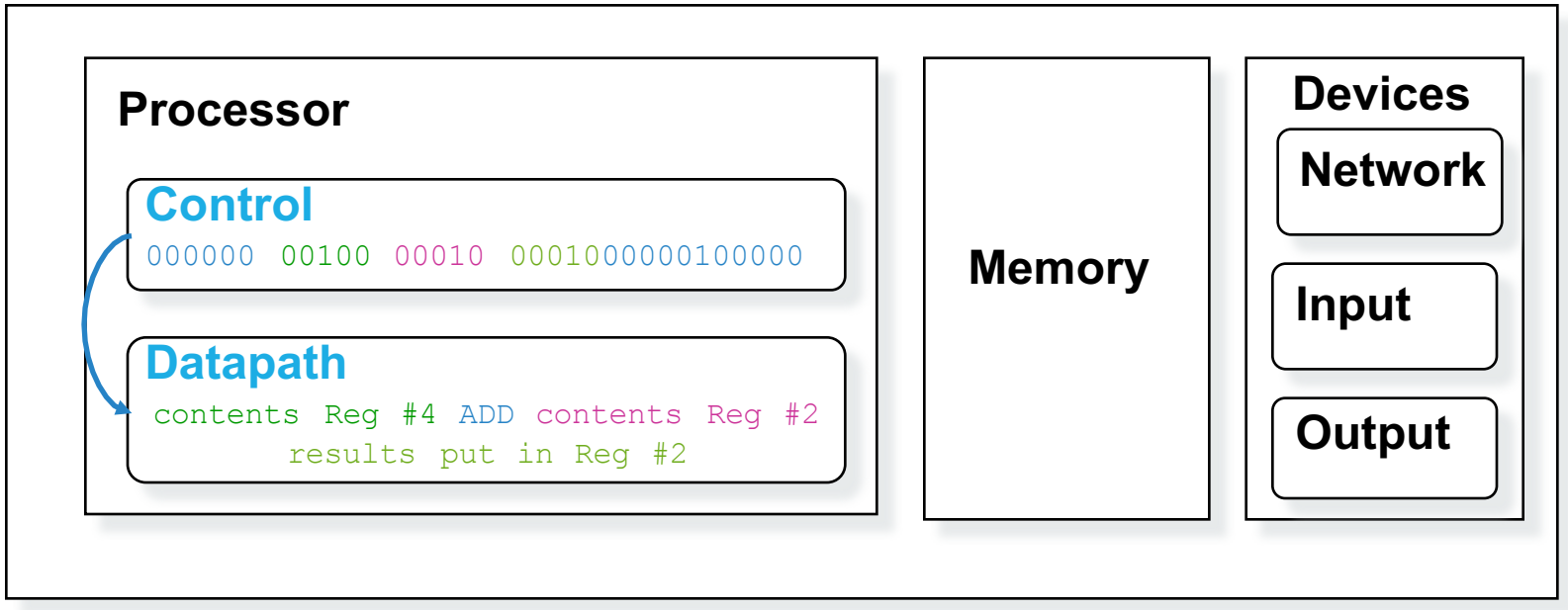
Control Decodes the Instruction

Control decodes the instruction to determine what to execute

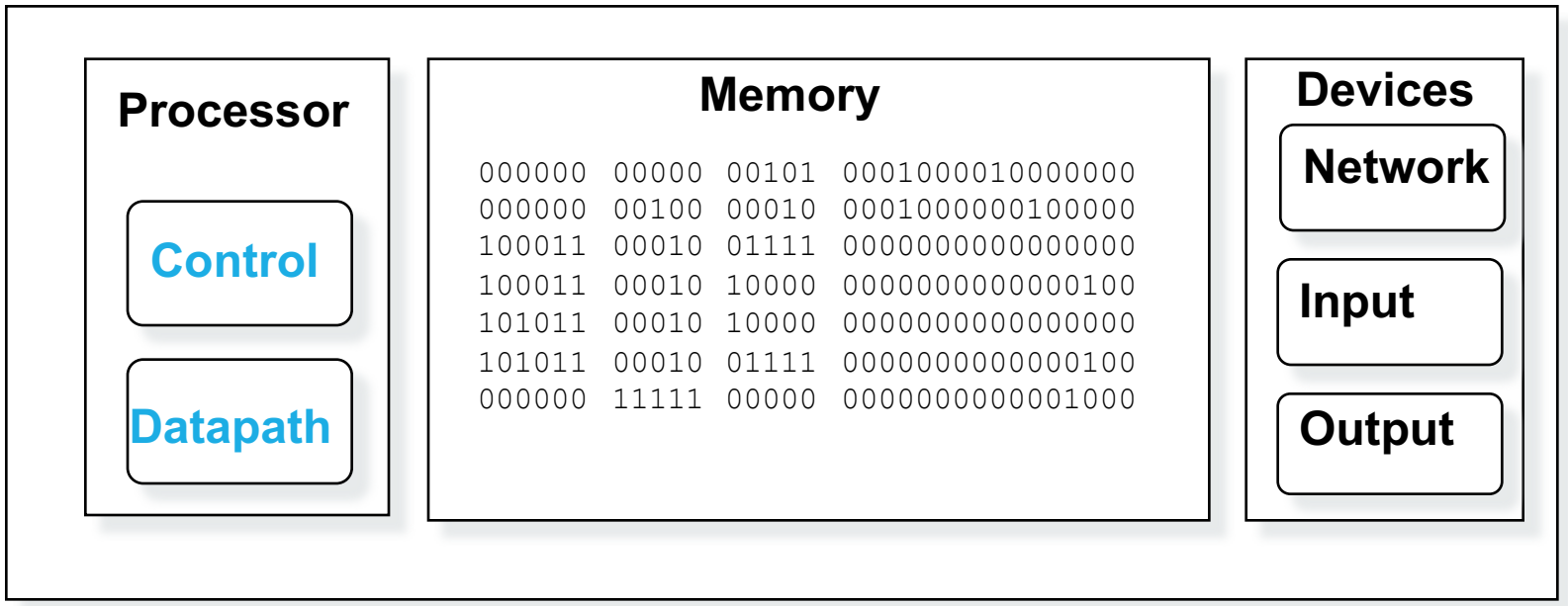


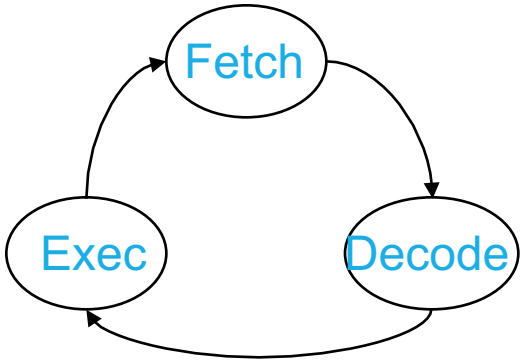
Datapath Executes the Instruction

Datapath executes the instruction as directed by control



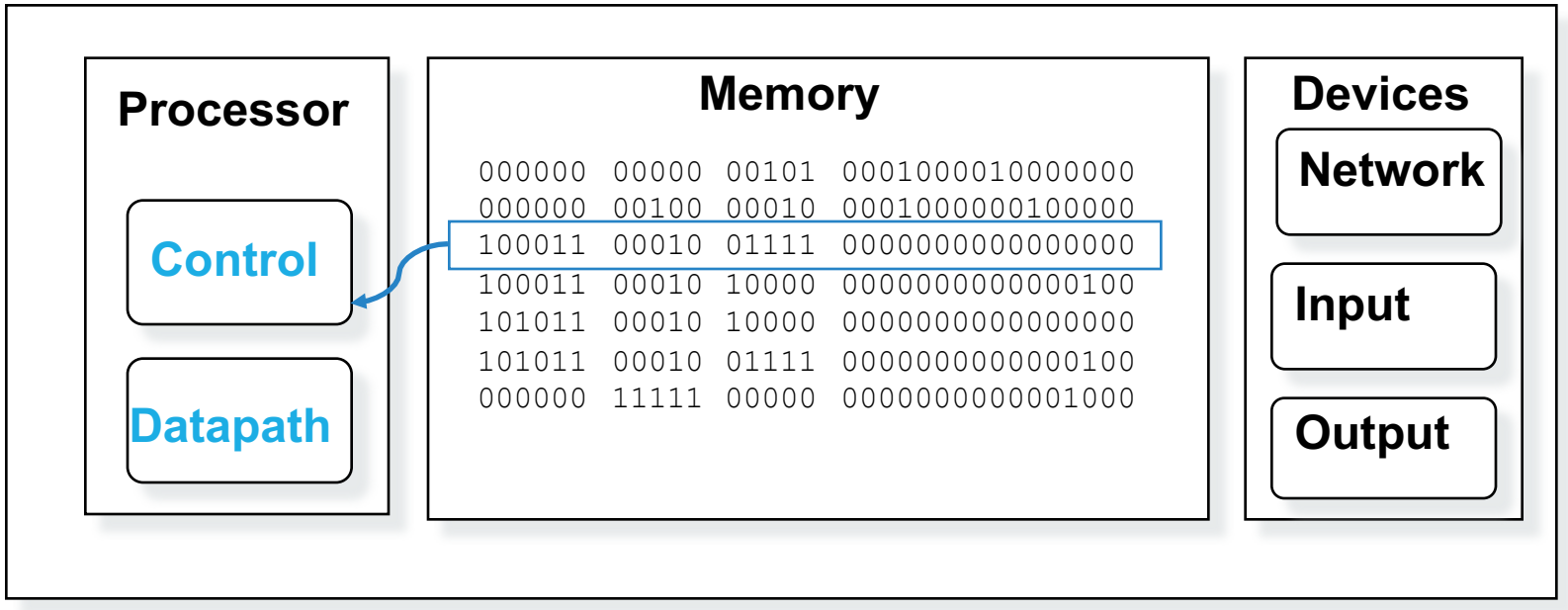
What Happens Next?





Processor Fetches the Next Instruction

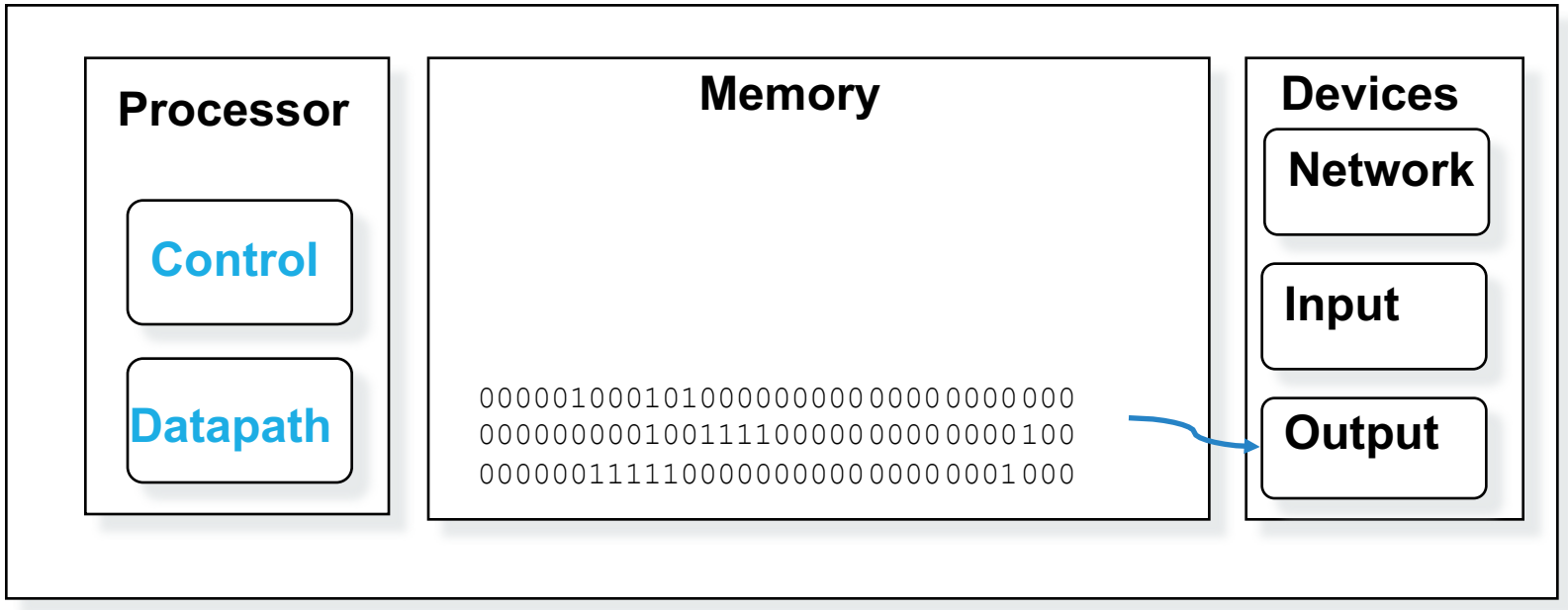
Processor fetches the *next* instruction from memory



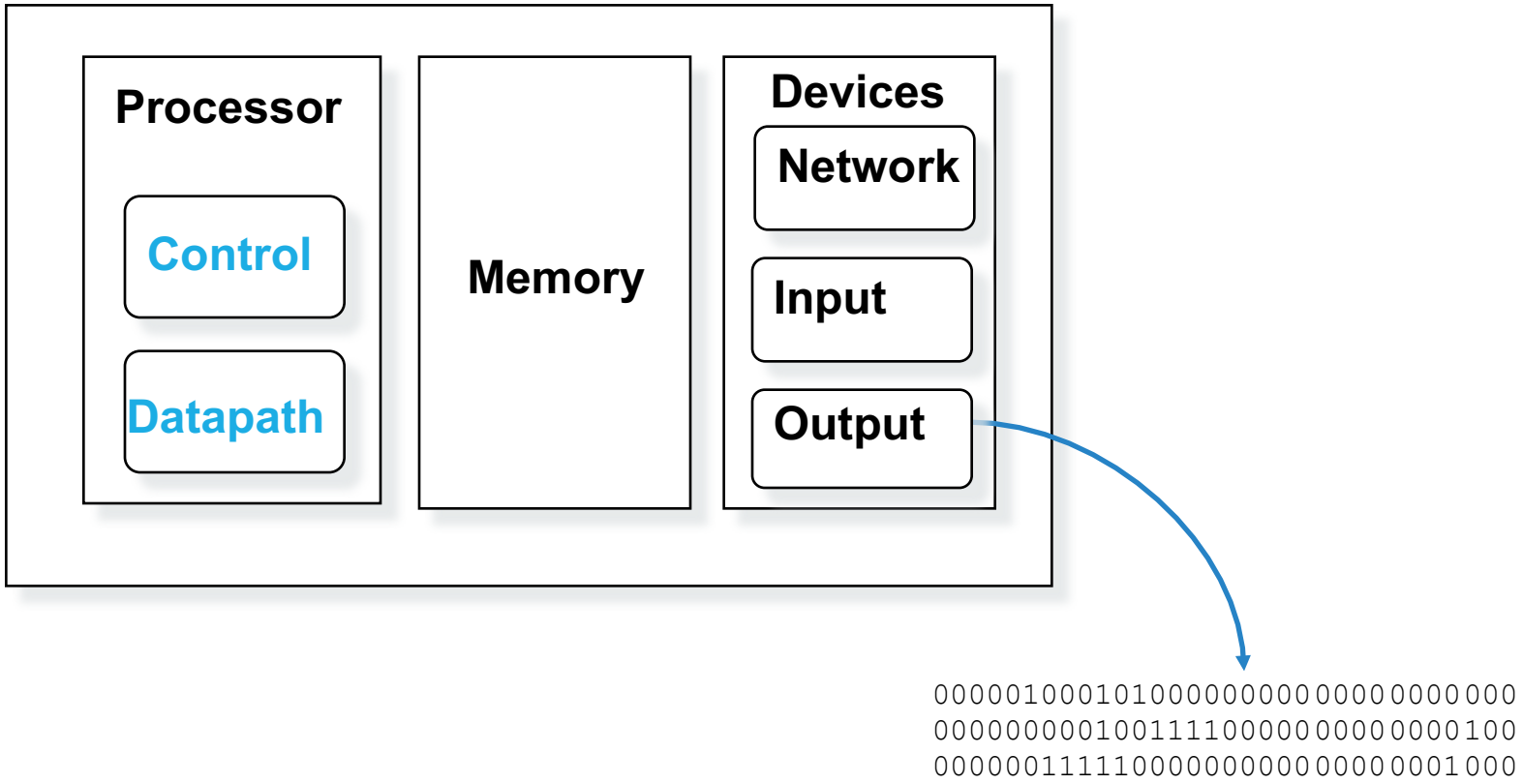
How does it know which location in memory to fetch from next?

Output Data Stored in Memory

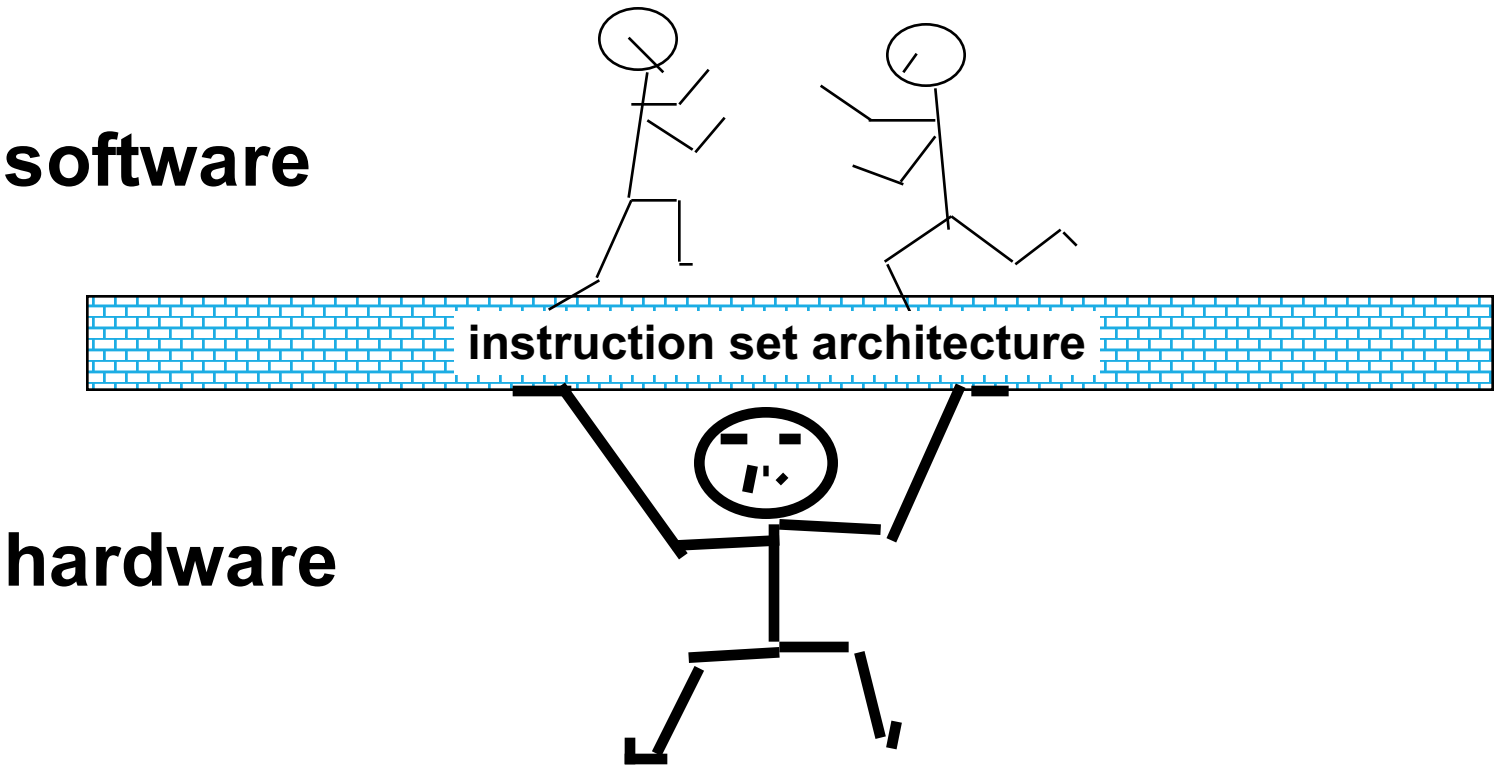
At program completion the data to be output resides in memory



Output Device Outputs Data



The Instruction Set Architecture (ISA)



The interface description separating the software and hardware

Instruction Set Architecture (ISA)

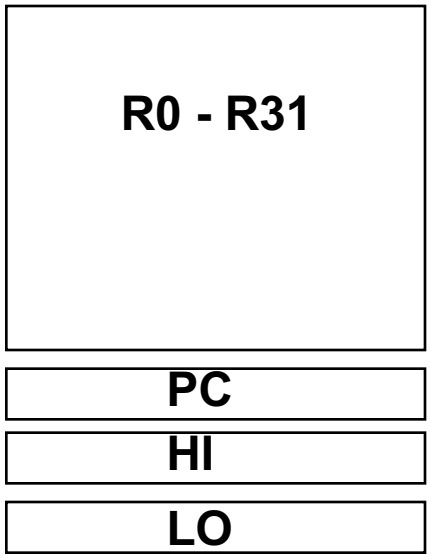
- ISA, or simply architecture – the abstract interface between the hardware and the lowest level software that includes all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, …
  - Enables implementations of varying cost and performance to run identical software

- The combination of the basic instruction set (the ISA) and the operating system interface is called the application binary interface (ABI)
  - ABI – The user portion of the instruction set plus the operating system interfaces used by application programmers. Defines a standard for binary portability across computers.

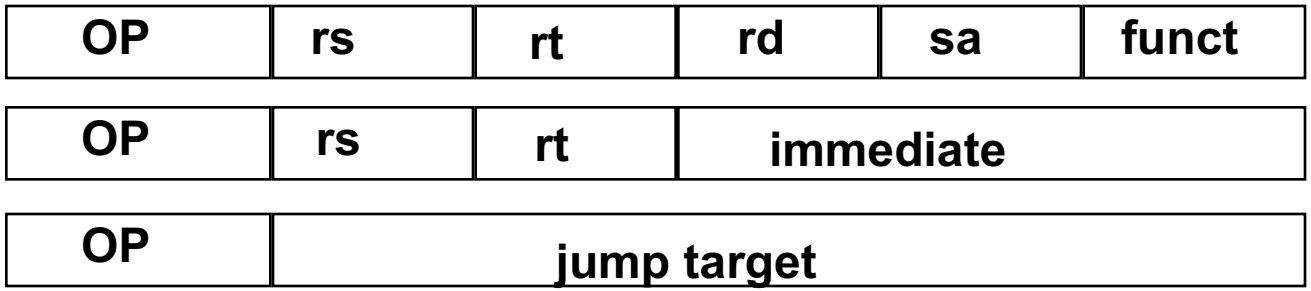
The MIPS ISA

- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

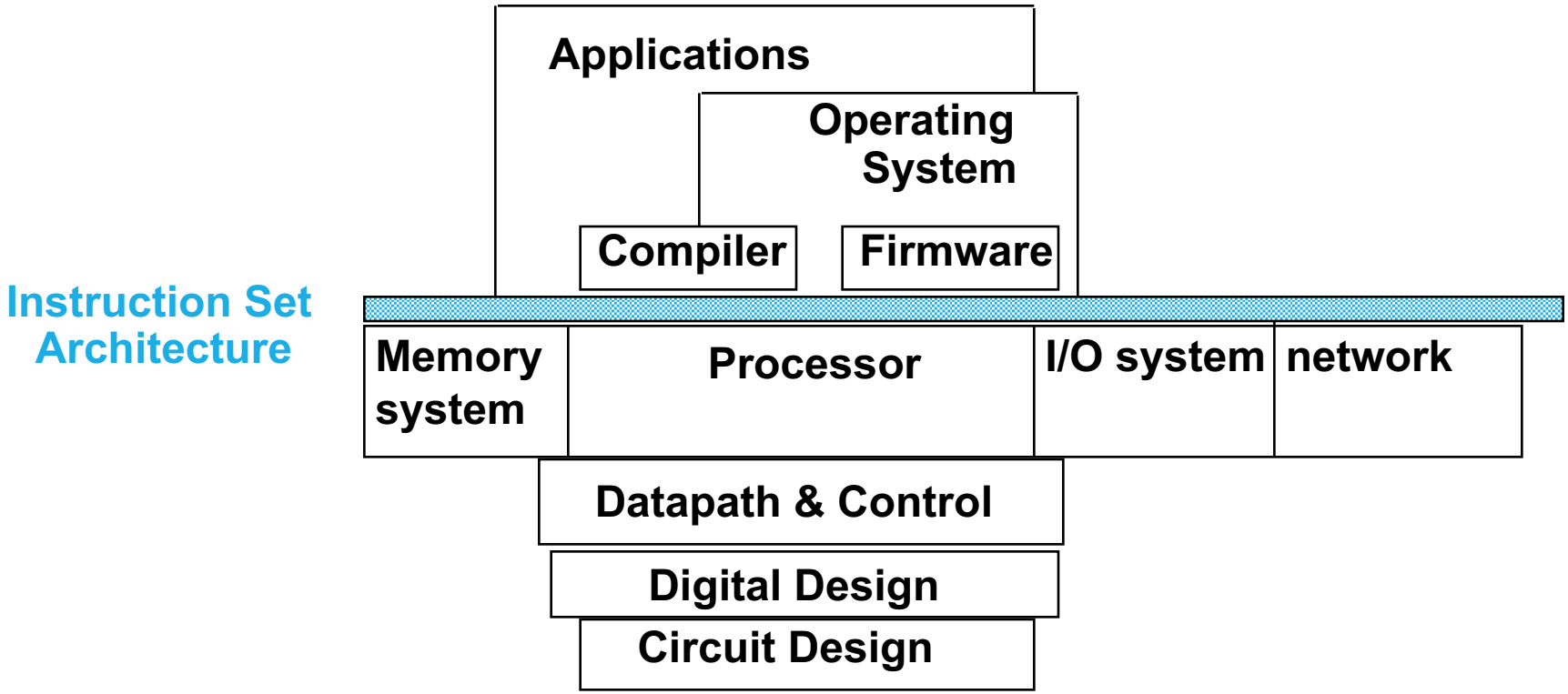
Registers



- **3 Instruction Formats: all 32 bits wide**

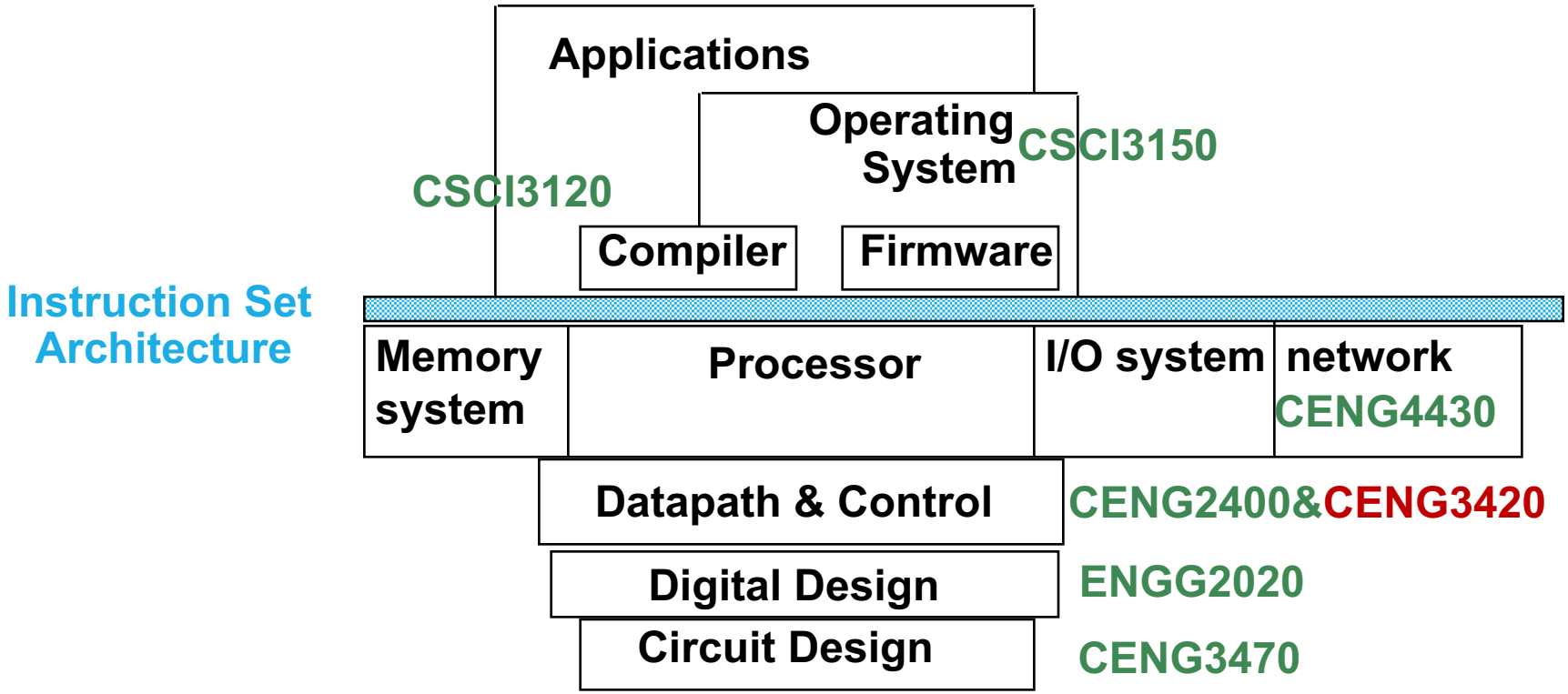


How Do the Pieces Fit Together?



- Coordination of many *levels of abstraction*
- Under a rapidly *changing* set of forces
- Design, measurement, *and* evaluation

How Do the Pieces Fit Together?



- Coordination of many *levels of abstraction*
- Under a rapidly *changing* set of forces
- Design, measurement, *and* evaluation