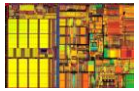
Computer Organization and Architecture CEC 470

Module 01 : Introduction to computer technology & ISA

Introduction

- Rapidly changing field:
  - vacuum tube  -> transistor  ->IC  -> VLSI 
  - doubling every 1.5 years:
    - memory capacity
    - processor speed (*due to advances in technology and hardware organization*)

3

3

Computer

3

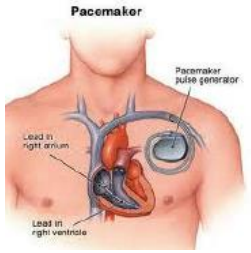
Computer

Computer

Computer






Pacemaker
Pacemaker pulse generator
Lead in right atrium
Lead in right ventricle




Canon


Computer

ALL are enabled by embedded microprocessor or Central Processing Unit (CPU)!

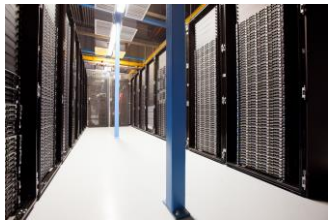
Classes of computers

- **PERSONAL/DESKTOP COMPUTERS**

  deliver good performance to a single user at low cost, includes graphic display, a keyboard, and a mouse

- **Servers**

  execute larger programs, multiple users served simultaneously, accessed via network,...

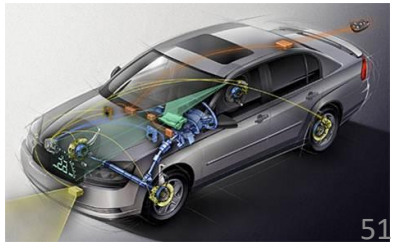


- **Supercomputers**

  complex computations, expensive, terabytes of memory, petabyte of storage, high end scientific applications,...

- **Embedded computers**

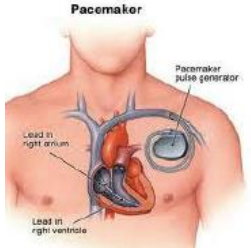
  inside other devices and perform one predetermined operation



Computer







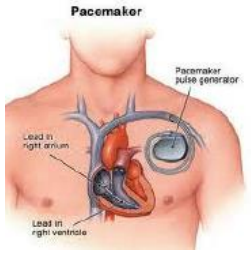



ALL are enabled by embedded microprocessor !

Computer







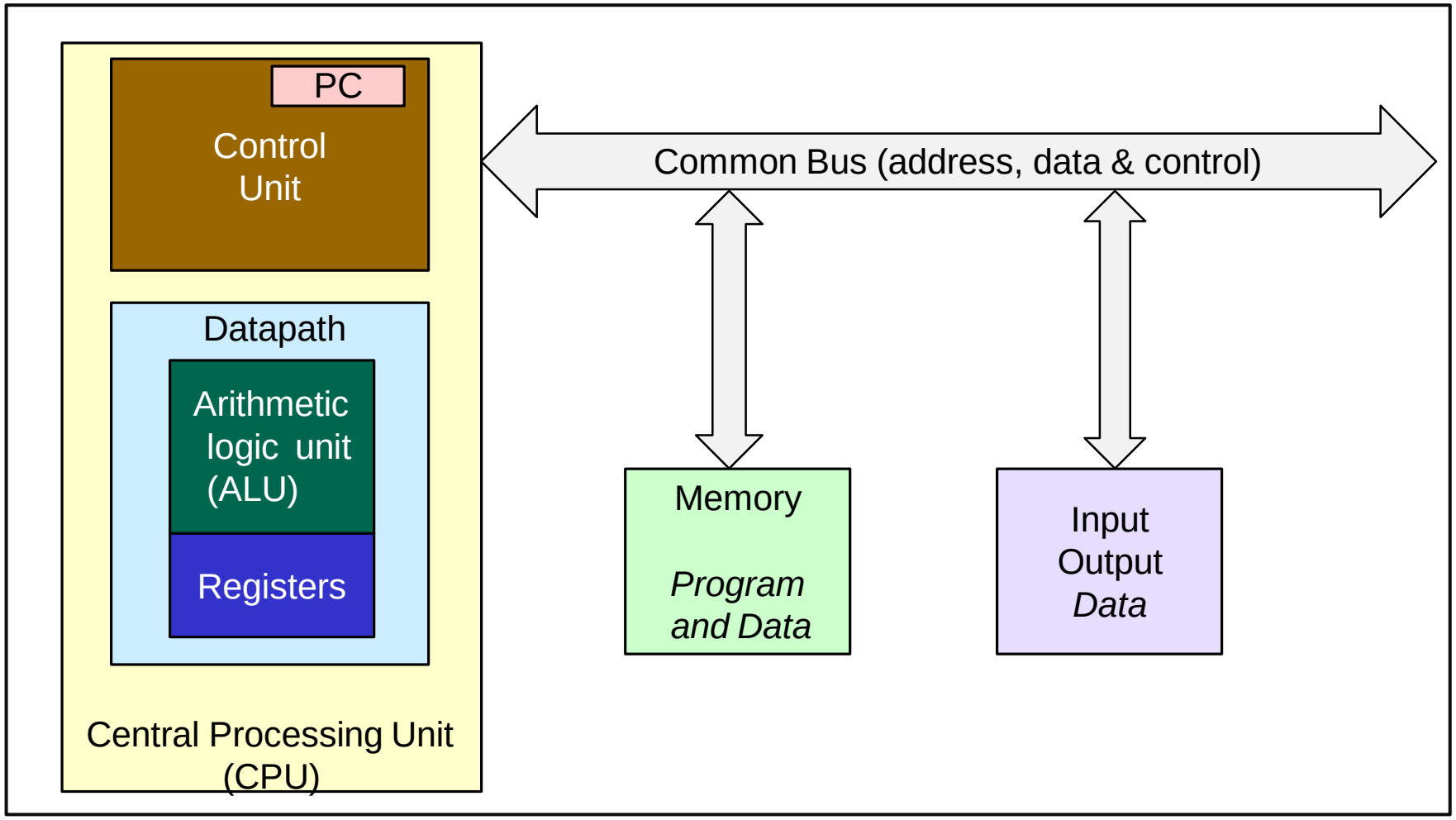



ALL are enabled by embedded microprocessor !

What's does the computers do?

- EXECUTE PROGRAMS

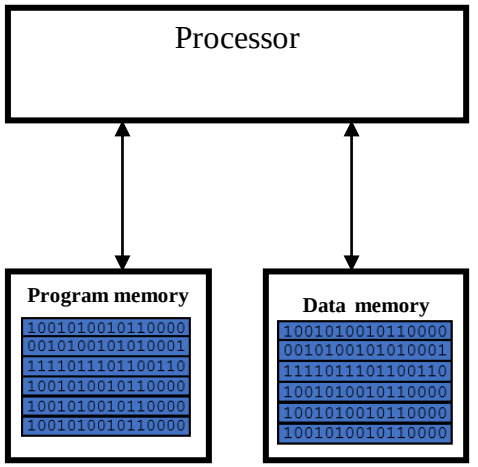
Five classical components of a computer



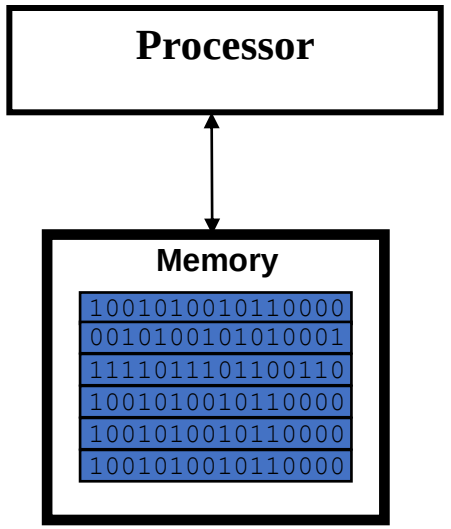
Computer System

Von Neumann Machine /Stored Program Computer

- Program(instructions) are ***bit sequences***, just like data
- Programs are stored in memory and are called **machine code instructions**
  - *To be read and written just like data*



- Each machine code instruction consists of a pattern of "1" and "0s" which determine operation/action to be performed
- Mapping of machine code instructions to CPU operations is sometimes called as **instruction set architecture (ISA)**



Stores both program& data

Fetch, decode and execute cycle

- Instructions are fetched and put into special registers inside CPU
- Bits in the register control the subsequent actions (=execution)
- Fetch the next instruction and repeat

What is computer architecture? Easy answer

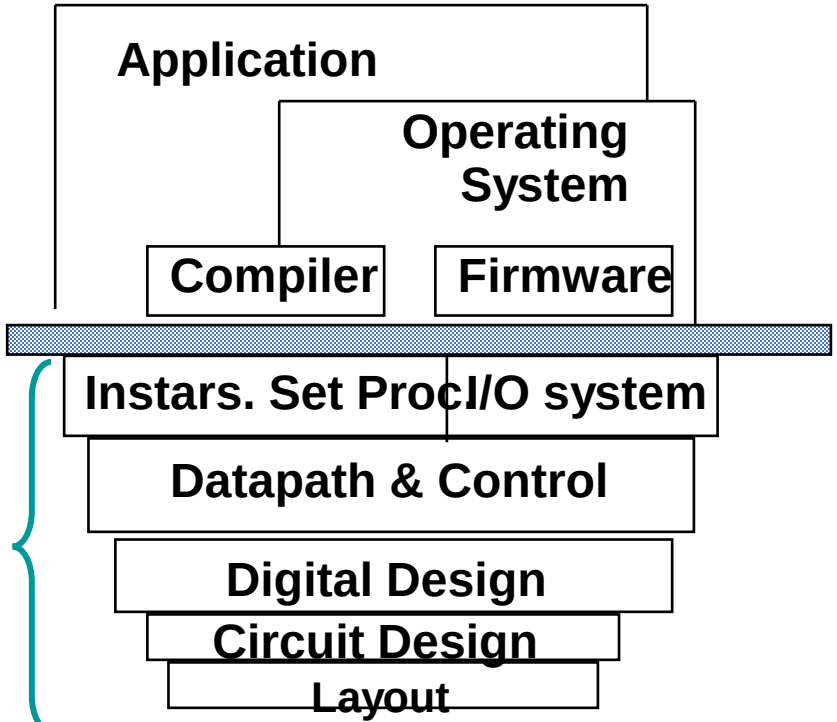
Instruction set architecture (ISA)

+

machine organization

What is computer architecture? Detailed answer

- Several ***levels of abstraction*** involved in communicating with a computer

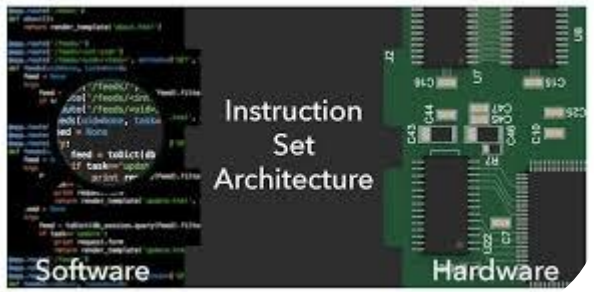


Notice how abstraction hides the detail of lower levels, yet gives a useful view for a given purpose

Instruction set architecture (ISA)

- *ISA, or simply architecture: the abstract interface between hardware and the lowest level of software that encompasses all the information necessary to write a machine language program, including instructions, registers, memory access, IO, …*

- ISA Includes
  - Organization of storage
  - Data types
  - Encoding and representing instructions
  - Instruction Set (i.e., opcodes)
  - Modes of addressing data items/instructions
  - Program visible exception handling

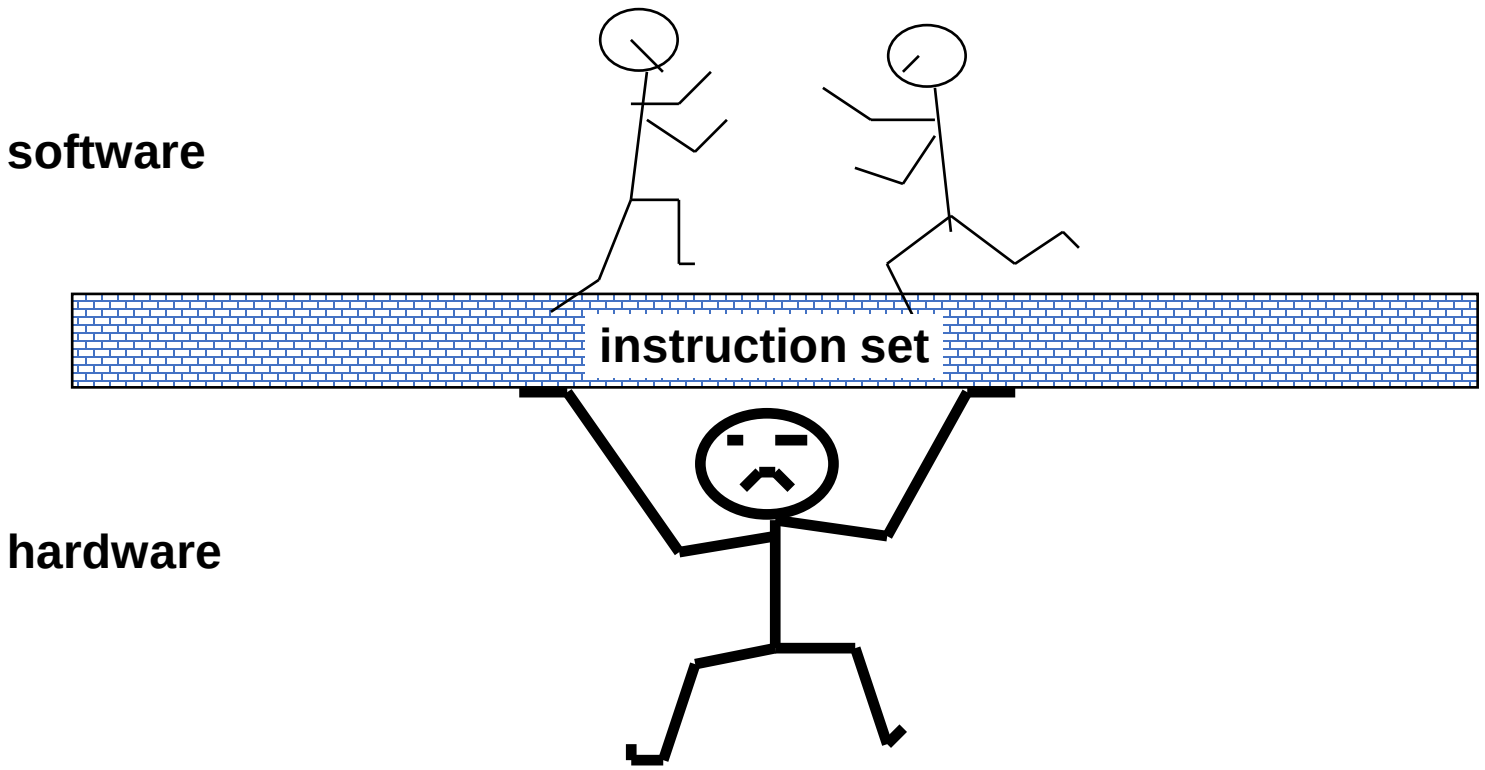


Instruction set architecture (ISA)

- ***A very important abstraction***
  - interface between hardware and low-level software
  - standardizes instructions, machine language bit patterns, etc.
  - **advantage**: *different implementations of the same architecture*
  - **disadvantage**: *sometimes prevents using new innovations*

- **Common instruction set architectures:**
  - IA-64, IA-32, PowerPC, MIPS, SPARC, ARM, and others
  - All are multi-sourced, with different implementations for the same ISA

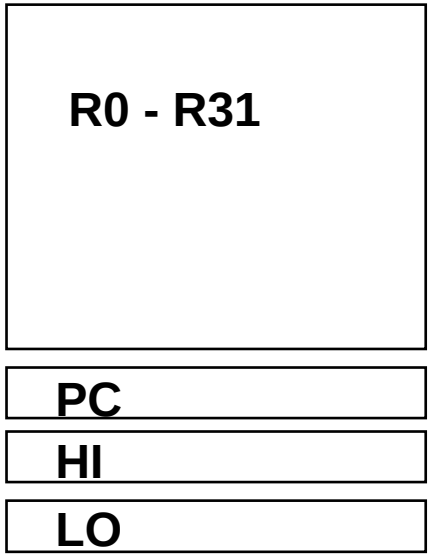
Instruction set architecture (ISA)



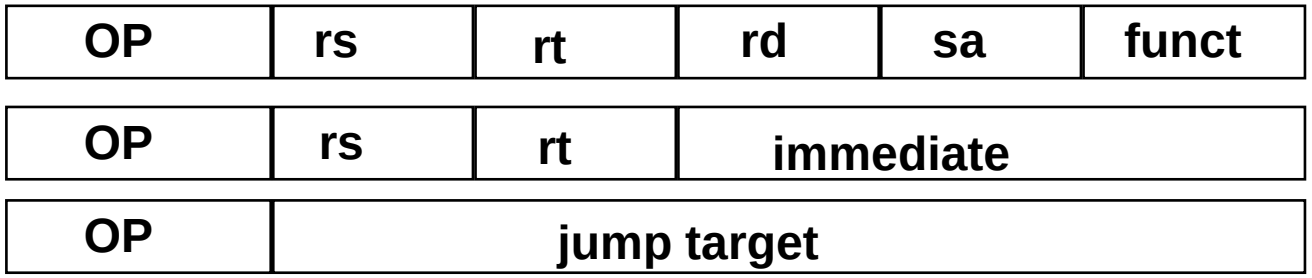
MIPS ISA

Microprocessor without Interlocked Pipelined Stages (MIPS) is a Reduced Instruction Set Computer (RISC)

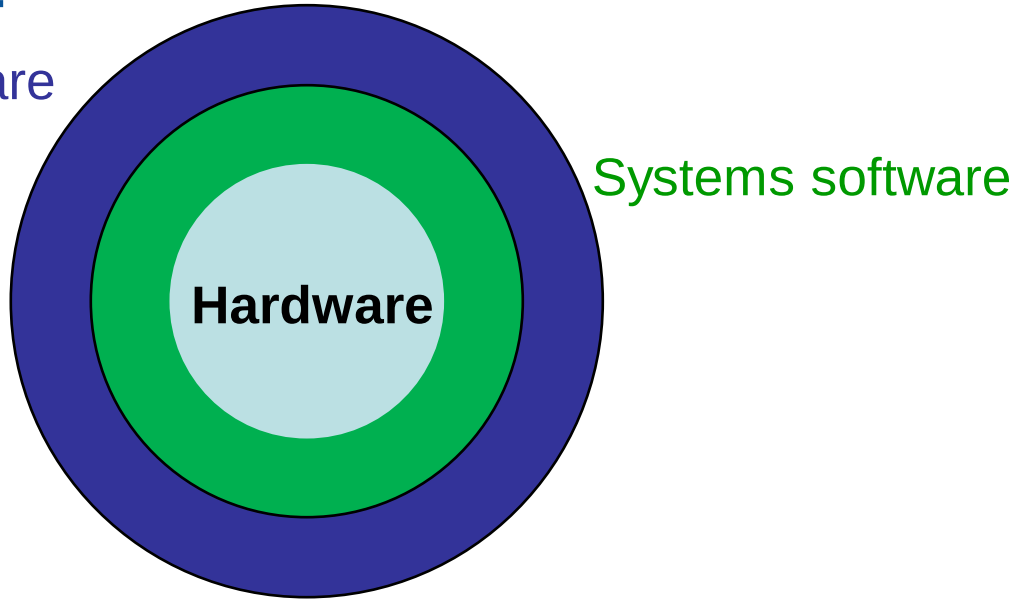
- Instruction Categories
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
  - Memory Management
  - Special



3 Instruction Formats, 32 bits wide



Below the program



System software:

Operating system: interface between the user program and the hardware (e.g., Linux, MacOS, Windows)

- handles basic input and output,
- allocates storage and memory,
- protected sharing among multiple applications

Compiler: translator (high-level language to instructions that the hardware can execute)

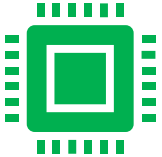
What language does a computer speak?

High Level Language Program

Compiler

Assembly Language Program

Assembler



Machine Language Program

Machine Interpretation

Control Signal Specification

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw   $15,   0($2)
lw   $16,   4($2)
sw   $16,   0($2)
sw   $15,   4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

ALUOP[0:3] <= InstReg[9:11] & MASK
[i.e.high/low on control lines]

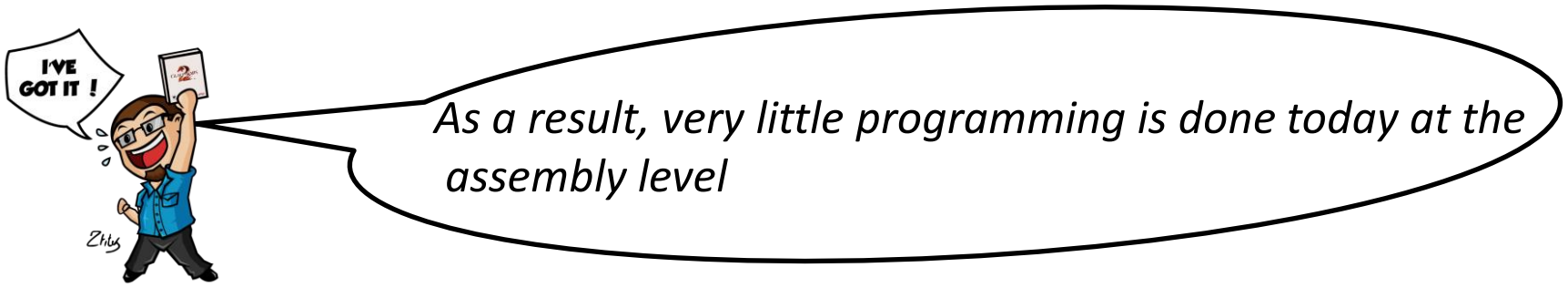
Advantages of high-level language

Advantages of high-level language

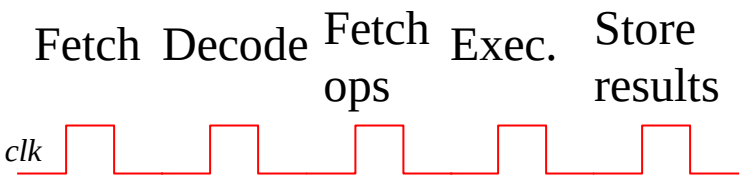
- **Higher-level languages (HLLS)**
  - Allow the programmer to **think in a *more natural language*** and tailored for the intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, …)
  - **Improve programmer productivity & maintainability** – more understandable code that is easier to debug and validate
  - Allow programs to be **machine independent** of the computer on which they are developed (compilers and assemblers can translate HLL programs to the binary instructions of any machine)

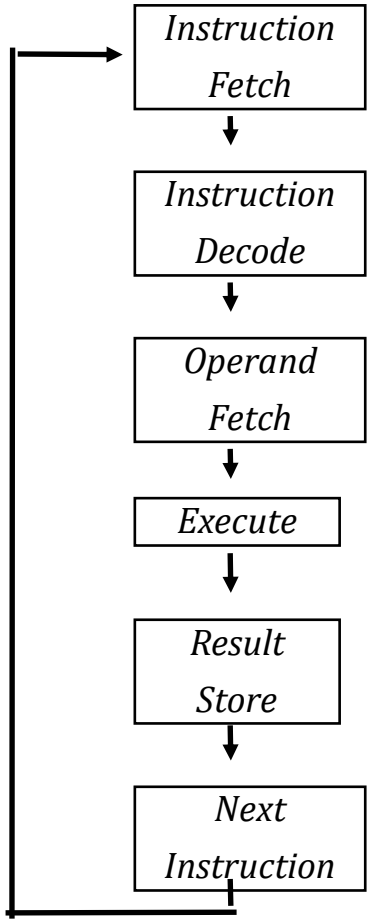
Advantages of high-level language

- **Higher-level languages (HLLS)**
  - Allow the programmer to **think in a *more natural language*** and tailored for the intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, …)
  - **Improve programmer productivity & maintainability** – more understandable code that is easier to debug and validate
  - Allow programs to be **machine independent** of the computer on which they are developed (compilers and assemblers can translate HLL programs to the binary instructions of any machine)



Execution Cycle (Sequential Model)





Fetch instruction from memory

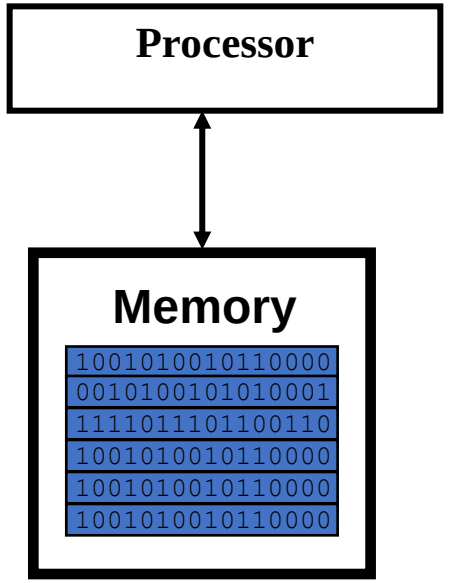
Determine required actions and instruction size

Locate and obtain operand data

Compute result value or status

Deposit results in storage for later use

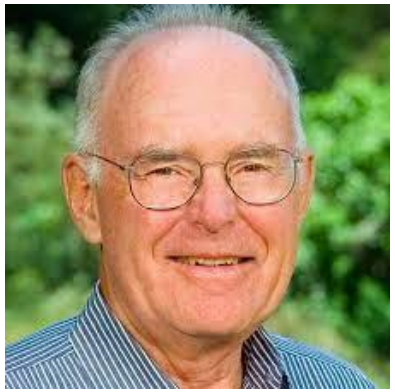
Fetch next instruction



Stores both program& data

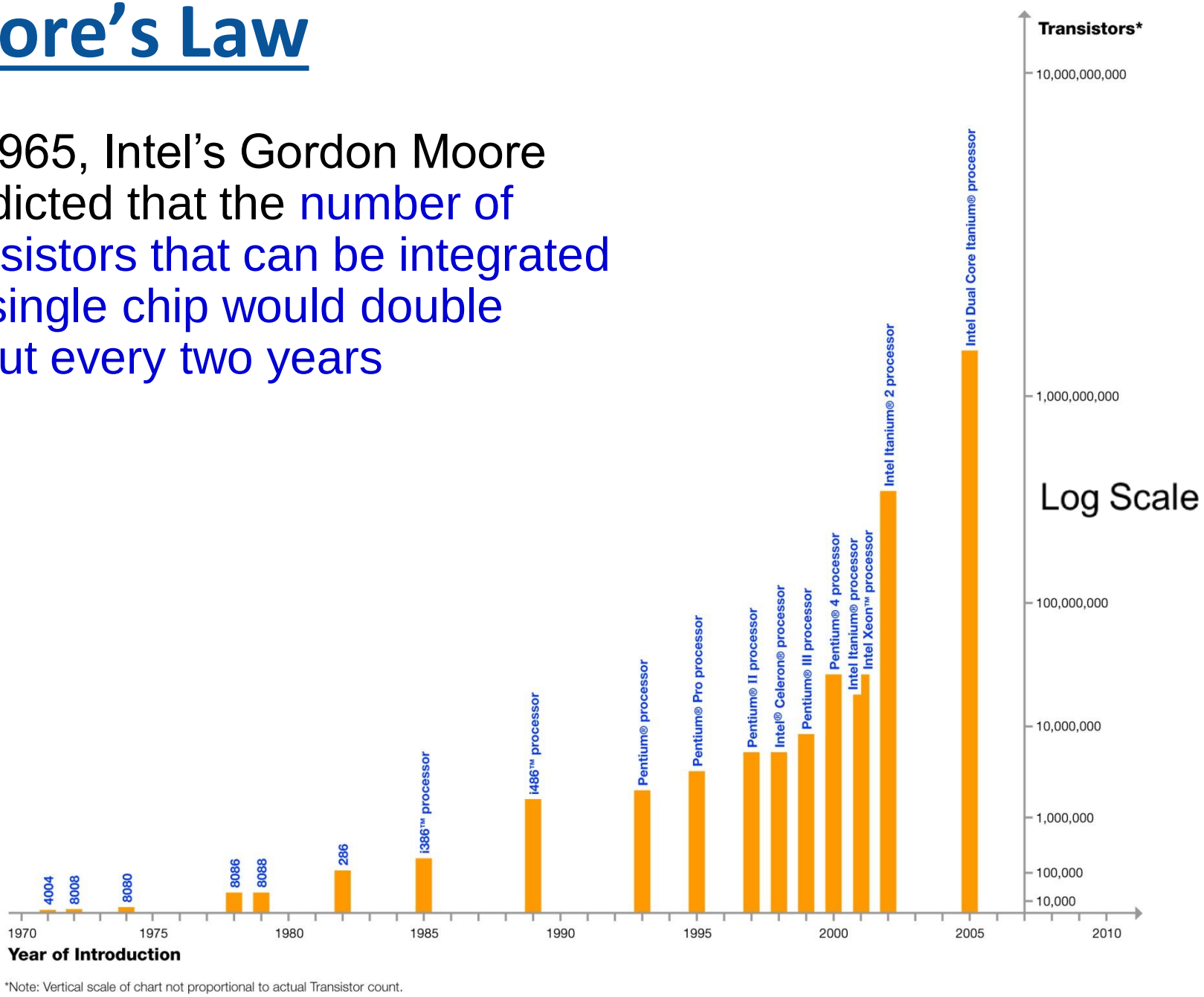
Moore's Law

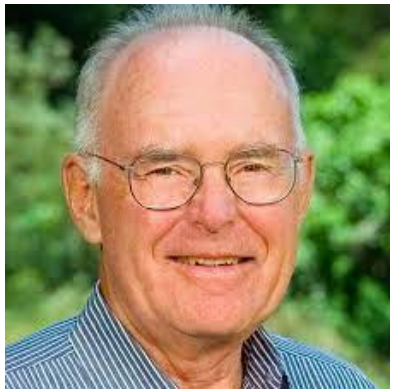
In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years



Moore's Law

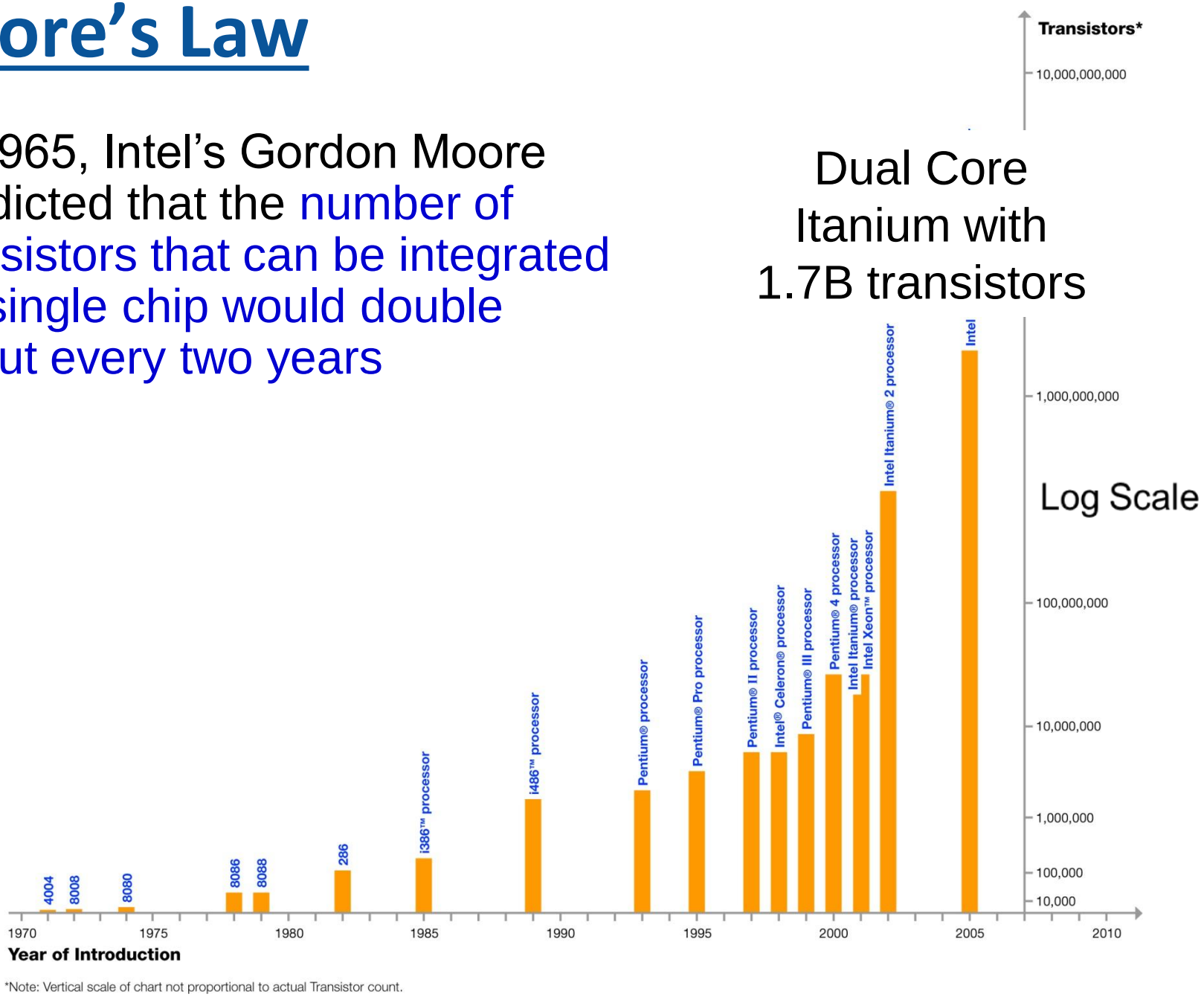
In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years



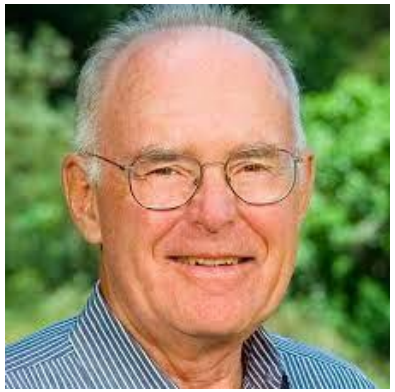


Moore's Law

In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years



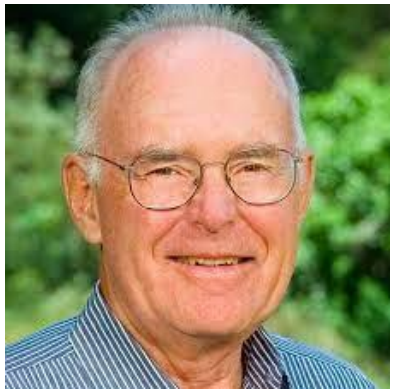
Dual Core Itanium with 1.7B transistors



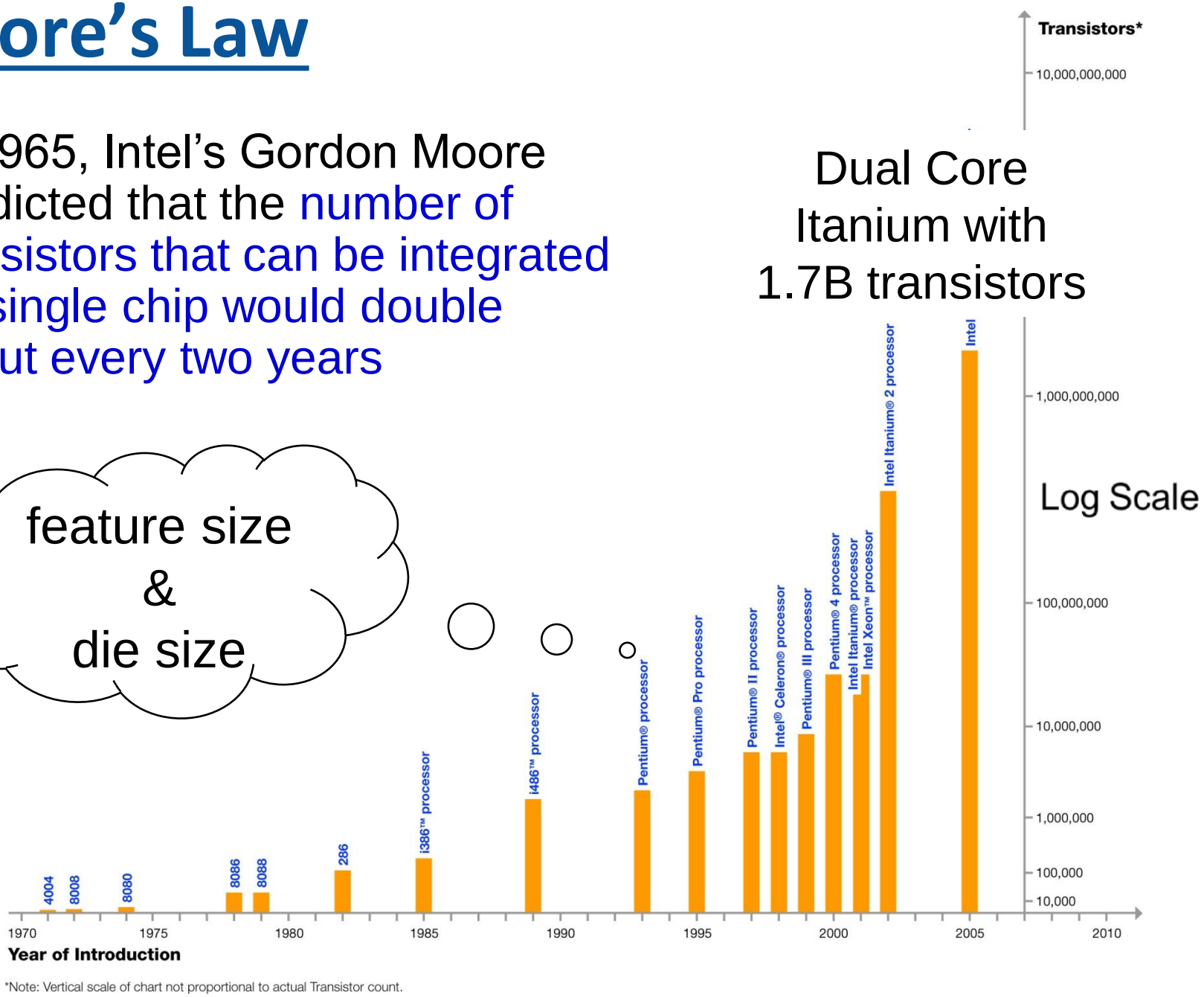
Moore's Law

In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years

Dual Core Itanium with 1.7B transistors



feature size
&
die size



Technology scaling road map

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

Technology scaling road map

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

Technology scaling road map

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

Technology scaling road map

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

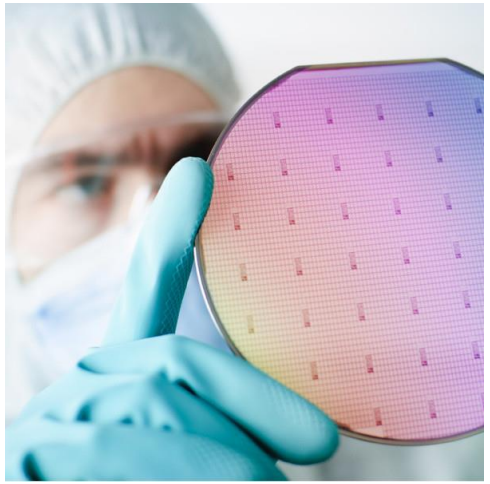
- Fun facts about 45nm transistors
  - 30 million can fit on the head of a pin
  - You could fit more than 2,000 across the width of a human hair
  - If car prices had fallen at the same rate as the price of a single transistor has since 1968, a new car today would cost about 1 cent

Technology scaling road map

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

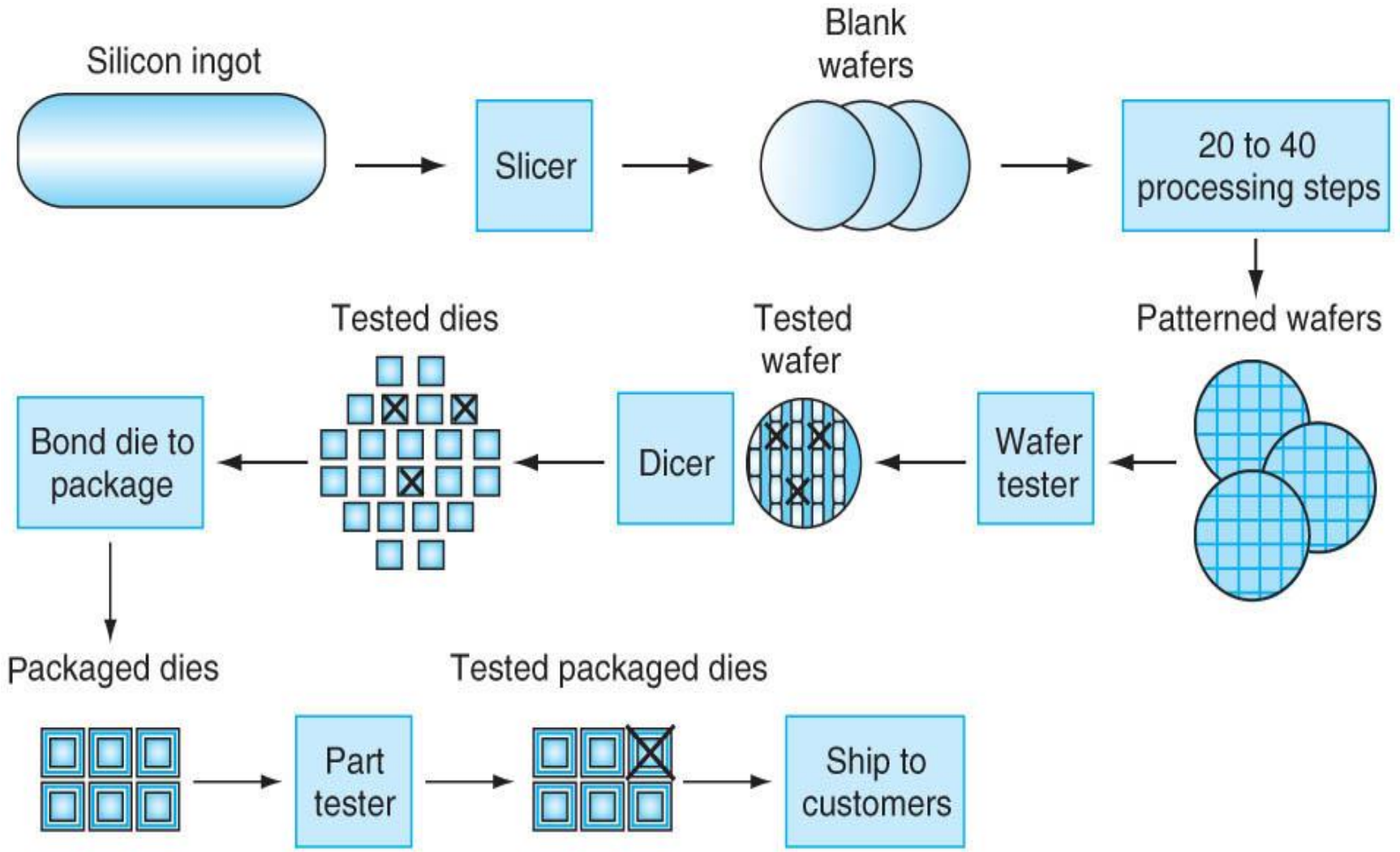
- Fun facts about 45nm transistors
  - 30 million can fit on the head of a pin
  - You could fit more than 2,000 across the width of a human hair
  - If car prices had fallen at the same rate as the price of a single transistor has since 1968, a new car today would cost about 1 cent

Semiconductors



- 50-year-old industry
  - Still has continuous improvements
  - New generation every 2-3 years
    - 30% reduction in dimension, 50% in area
    - 30% reduction in delay, 50% speed increase
    - Current generation: Reduce cost and increases performance
    - Processors are fabricated on ingots cut into wafers which are then etched to create transistors
    - Wafers are then diced to form chips, some of which have defect
    - Yield is the measurement of the good chips
    - Next generation: Larger with more functions
- Each generation is an incremental improvement

Chip manufacturing process



Hitting the power wall

"For the P6, success criteria included performance above a certain level and failure criteria included power dissipation above some threshold."

Bob Colwell, Pentium Chronicles

Performance

- **Performance** is the key to understanding underlying motivation for the hardware and its organization

- Measure, report, and summarize performance to enable users to
  - make intelligent choices
  - see through the marketing hype! Why is some hardware better than others for different programs?

- *What factors of system performance are hardware related? (e.g., do we need a new machine, or a new operating system?)*

- *How does the machine's instruction set affect performance?*

Computer performance: time,time,time !!

- ***Response Time*** (*elapsed time*, *latency*):
  - how fast will my program run?
  - how long does it take to execute (start to finish) *my* job?
  - how long must wait for the database query?

Individual user concerns…

***Throughput*:**

- how *many* jobs can the machine run at once?
- what is the *average* execution rate?
- how *much* work is getting done?

Systems manager concerns…

If we upgrade a machine with a new processor, what do we increase?

If we add a new machine to the lab what do, we increase?

Execution time

- ***Elapsed Time***
  - counts everything (*disk and memory accesses, waiting for I/O, running other programs, etc.*) from start to finish
  - **elapsed time = CPU time + wait time (I/O, other programs, etc.)**

- ***CPU time***
  - doesn't count waiting for I/O or time spent running other programs
  - can be divided into *user CPU time* and *system CPU time* (OS calls)

CPU time = user CPU time + system CPU time

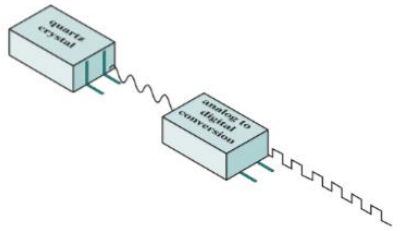
⇒ elapsed time = user CPU time + system CPU time + wait time

Our focus: *user CPU time* (*CPU execution time* or, simply, *execution time*) time spent executing the lines of code that are *in our program*

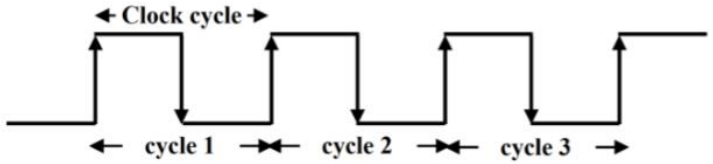
Clock Cycles

- Execution time is reported in *cycles*. In modern computer hardware each event, e.g., multiplication, addition, etc., is a sequence of *cycles*

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$



- *Clock ticks* indicate start and end of cycles:



- *Cycle time* = time between ticks = seconds per cycle
- *Clock rate* (*frequency*) = cycles per second (1 Hz = 1 cycle/sec, 1 MHz = 10^6 cycles/sec)
- *Example*: A 200 Mhz. clock has a $\frac{1}{200 \times 10^6}$ $= 5$ nanoseconds

Performance equation I

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

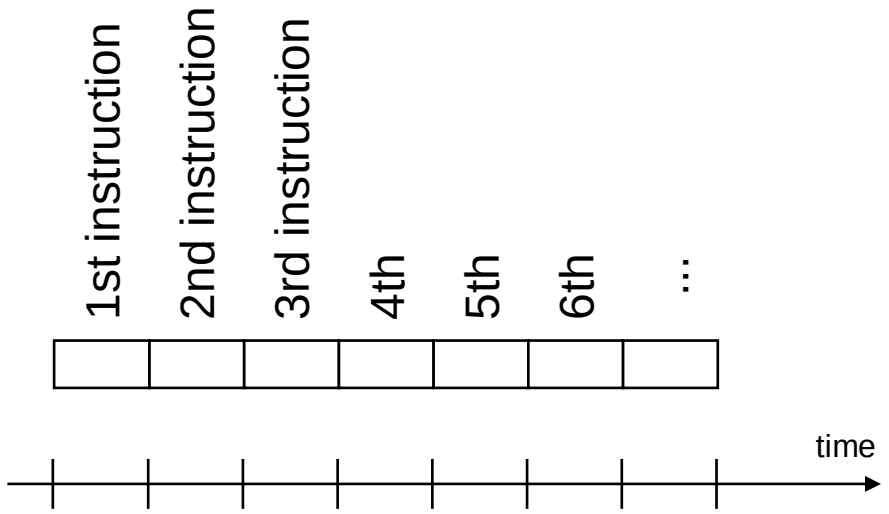
equivalently

CPU execution time for a program = CPU clock cycles for a program X Clock cycle time

So, to improve performance one can either: reduce the number of cycles for a program, or reduce the clock cycle time, or, equivalently, increase the clock rate

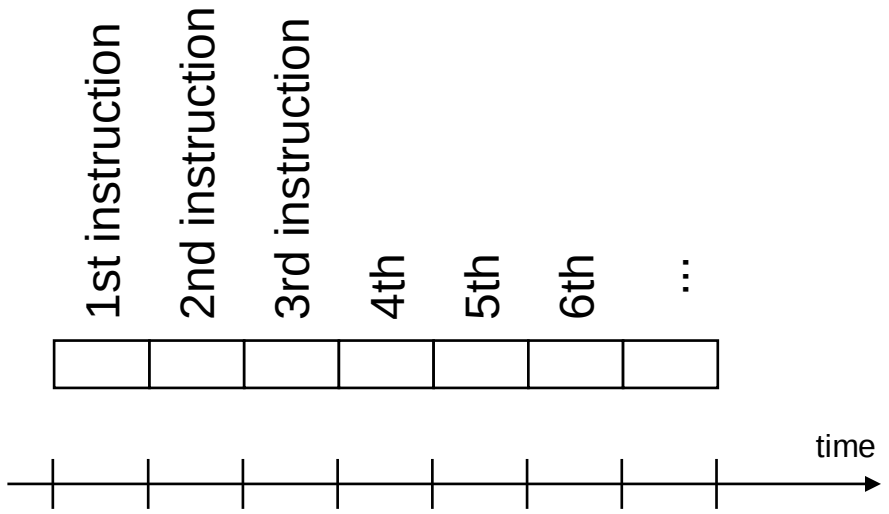
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



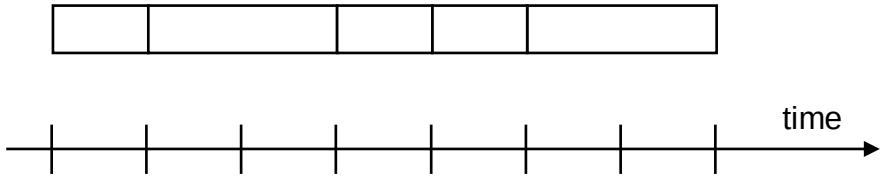
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



- *This assumption is incorrect!* Because:
  - Different instructions take different amounts of time (cycles)
  - Why…?

How many cycles are required for a program?



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- ***Important point:*** changing the cycle time often changes the number of cycles required for various instructions because it means changing the hardware design. More later…

Example

- Our favorite program runs in 10 seconds on computer A, which has a 400Mhz. clock.

- We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program.

- *What clock rate should we tell the designer to target?*

Example Solution

Computer A: 10 sec and frequency = 400 MHz

Computer B: 6 sec and frequency =?

If computer A takes X cycles, the computer B will take 1.2X cycles.

$$10/6 = X/400\ \text{MHz} / 1.2X/Y$$

$$10/6 = Y/480$$

$$4800/6 = Y$$

$$800\ \text{MHz} = Y$$

Performance equation II

$$\text{CPU execution time for a program} = \text{Instruction count for a program} \times \text{average CPI} \times \text{Clock cycle time}$$

CPI Example I

- Suppose we have two implementations of the same instruction set architecture (ISA). For some program:
  - machine A has a clock cycle time of 10 ns. and a CPI of 2.0
  - machine B has a clock cycle time of 20 ns. and a CPI of 1.2

- *Which machine is faster for this program, and by how much?*
- *If two machines have the same ISA, which of our quantities (e.g., clock rate, CPI, execution time, # of instructions, MIPS) will always be identical?*

CPI Example I solution

CPU time of A = 2 * 10 nsec = 20 nsec

CPU time of B = 1.2* 20 nsec = 24 nsec

Hence computer A is faster by :

$$\frac{\text{CPU performance of B}}{\text{CPU performance of A}} = 24/20 = 1.2$$

CPI Example II

- A compiler designer is trying to decide between two code sequences for a particular machine.
- Based on the hardware implementation, there are three different classes of instructions: Class A, Class B, and Class C, and they require 1, 2 and 3 cycles (respectively).
- The first code sequence has 5 instructions:

  2 of A, 1 of B, and 2 of C
  The second sequence has 6 instructions:

  4 of A, 1 of B, and 1 of C.

- *Which sequence will be faster? How much? What is the CPI for each sequence?*

CPI Example II solution

Class A: 1 cycle

Class B: 2 cycle

Class C: 3 cycle

Sequence 1: 2 of A, 1 of B, and 2 of C

Sequence 2: 4 of A, 1 of B, and 1 of C

CPI seq1 : 2*1+1*2+2*3= 10 CPI

Average CPI= 10/5= 2

CPI seq2 : 4*1+1*2+1*3= 9 CPI

Average CPI=9/6= 1.5

Terminology

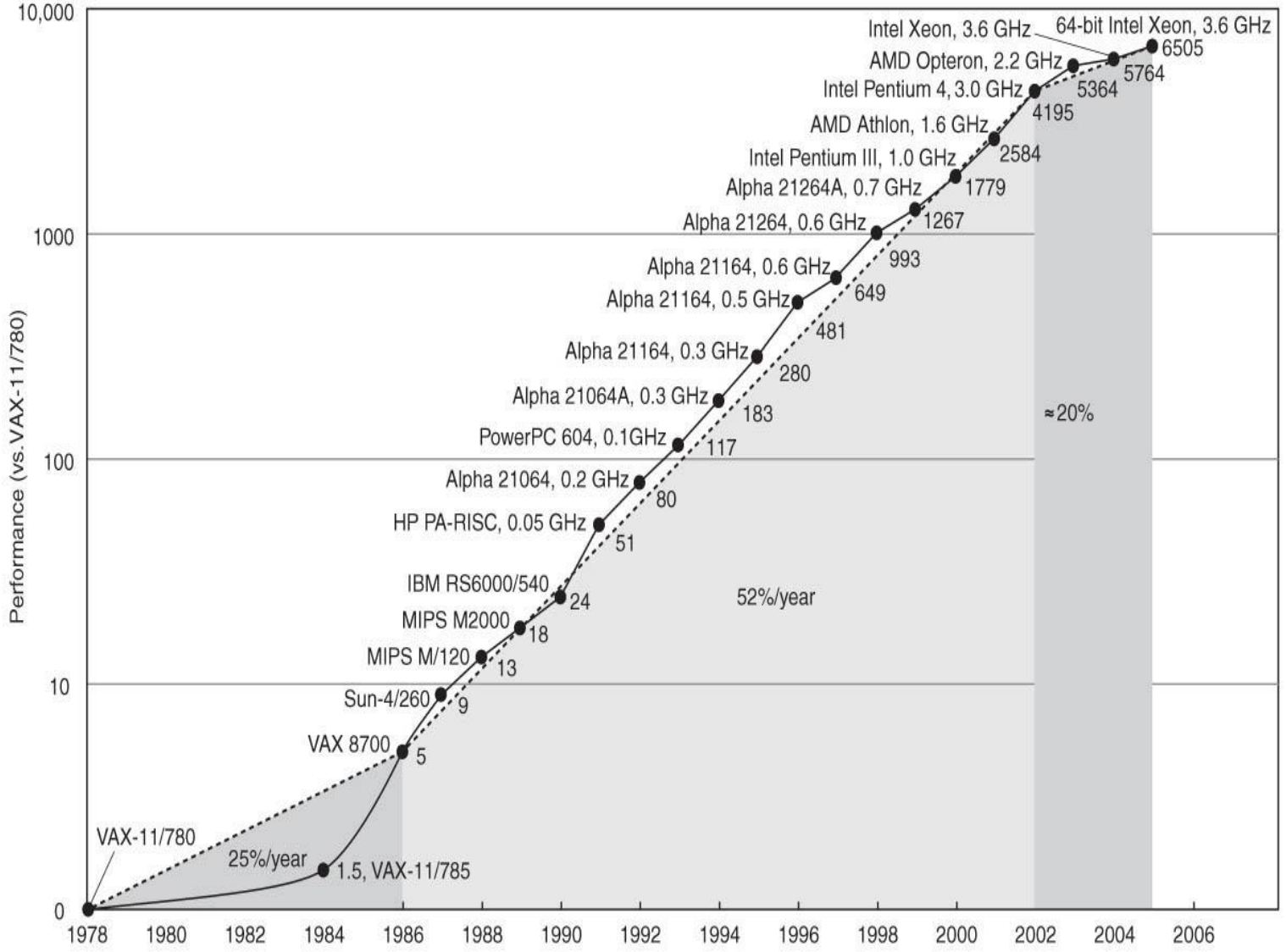
- A given program will require:
  - some number of instructions (machine instructions), some number of cycles, some number of seconds
- We have a vocabulary that relates these quantities:
  - *cycle time* (seconds per cycle)
  - *clock rate* (cycles per second)
  - (*average*) *CPI* (cycles per instruction)
    - a floating-point intensive application might have a higher average CPI
  - *MIPS* (millions of instructions per second)
    - this would be higher for a program using simple instructions

Performance measure

- Performance is determined by execution time
- Do any of these other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

- *Common pitfall* : thinking one of the variables is indicative of performance when it really isn't

Processor performance growth flattens!



Power
RC delay
Memory Latency

The latest revolution: multicores

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
- In 2011 all desktop and server companies are shipping microprocessors with multiple processors – cores – per chip

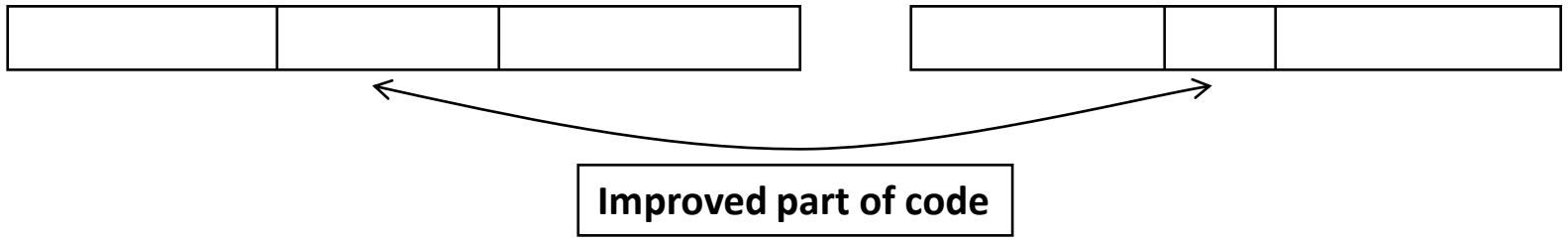
Product	AMD Barcelona	Intel Nehalem	IBM Power 6	Sun Niagara 2
Cores per chip	4	4	2	8
Clock rate	2.5 GHz	~2.5 GHz?	4.7 GHz	1.4 GHz
Power	120 W	~100 W?	~100 W?	94 W

The plan is to double the number of cores per chip per generation (about every two years)

Amdahl's law

- Speed up overall=

$$\frac{1}{\text{Execution Time Unaffected} + (\text{Execution Time Affected} / \text{Rate of Improvement})}$$



- *Example*:
  - Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time.
  - *How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?*
  - *How about making it 5 times faster?*

Design Principle: *Make the common case fast*

Amdahl's law

- *Example*:
  - Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time.
  - *How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?*
  - *How about making it 5 times faster?*

20+(80/n))= 25 ; n=16

(20+(80/n))=20; n=?
So its not possible to make 5 times faster.