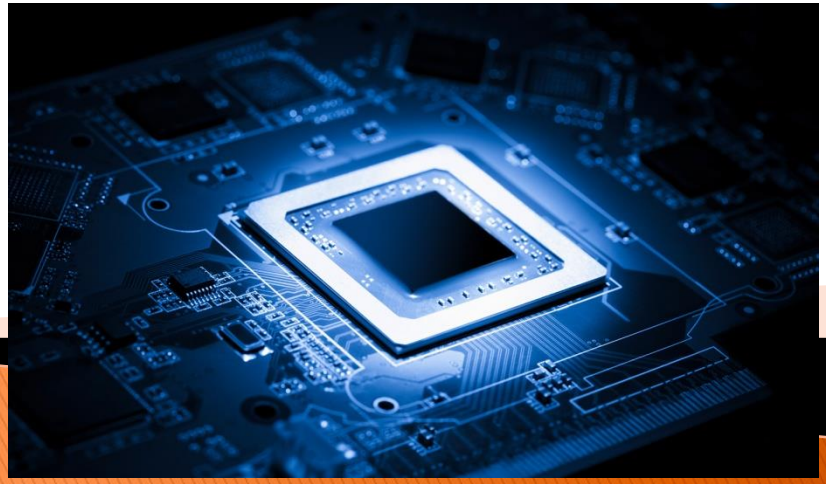
CSE 331
Computer Organization
Lecture 1

Computer Abstractions and Technology



The Computer Revolution

Computer

2013
Samsung

1973
Motorola

1896
İsveç

Electrical

Computer

2016
Raikkonen

1989

Mechanical

1946

IoT


Recipe Send to Oven
220° 20:00
-PIZZA START
Recipes
Recipes by Stored Food
Oven settings transferred to Oven.
OK
Recipe Send to Oven
Recipes
Sweet Potato Pizza
Cancel


SMART Saving
PUSH


Smart Lock
Unlocked
Back

The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

Fun Facts

The original transistor built by Bell Labs in 1947 was large enough that it was pieced together by hand. By contrast, more than 100 million 22nm tri-gate transistors could fit onto the head of a pin.[1]

More than 6 million 22nm tri-gate transistors could fit in the period at the end of this sentence.[2]

A 22nm tri-gate transistor's gates are so small you could fit more than 4,000 of them across the width of a human hair.[3]

If a typical house shrunk as transistors have, you would not be able to see a house without a microscope. To see a 22nm feature with the naked eye, you would have to enlarge a chip to be larger than a house.[4]

Compared to Intel's first microprocessor, the 4004, introduced in 1971, a 22nm CPU runs over 4,000 times as fast and each transistor uses about 5,000 times less energy. The price per transistor has dropped by a factor of about 50,000.

A 22nm transistor can switch on and off well over 100 billion times in one second. It would take you around 2,000 years to flick a light switch on and off that many times.[5]

It's one thing to design a tri-gate transistor but quite another to get it into high volume manufacturing. Intel's factories produce over 5 billion transistors every second. That's 150,000,000,000,000,000 transistors per year, the equivalent of over 20 million transistors for every man, woman and child on earth.

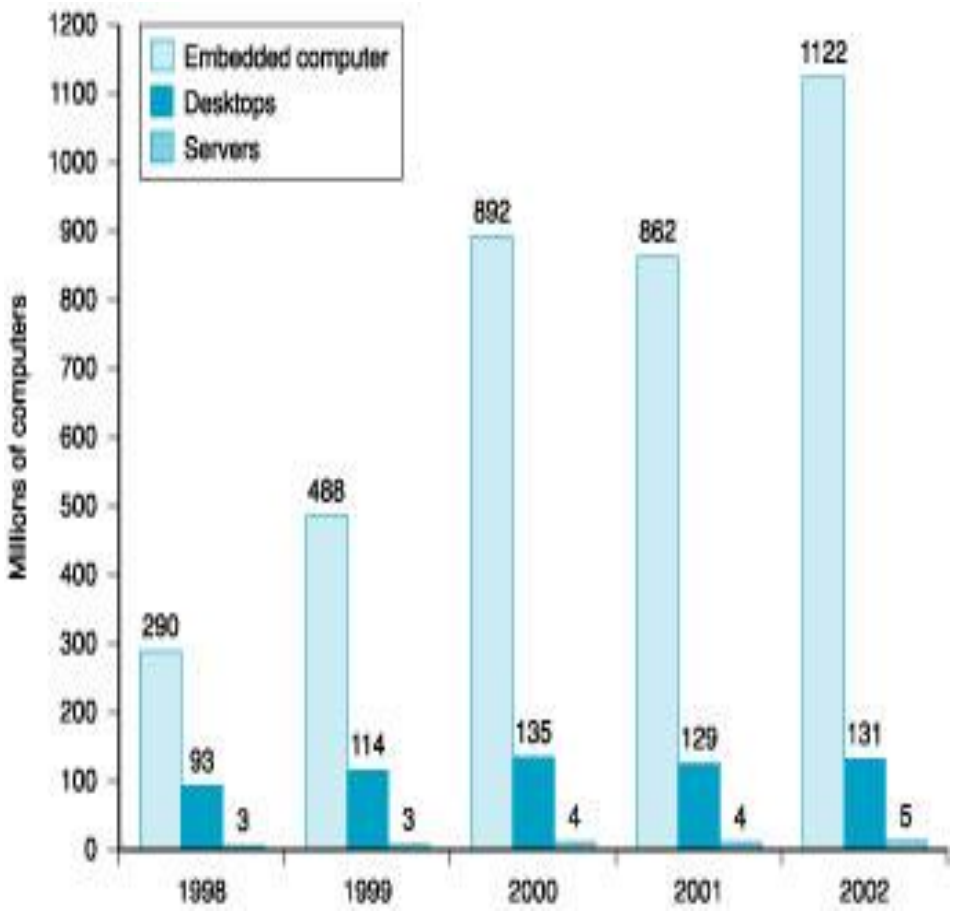
The 3rd Generation Intel® Core™ processor — quad core, contains 1.48 billion transistors. If transistors were people, Intel's chip has more transistors than the population of China at approximately 1.3 billion people.

Introduction

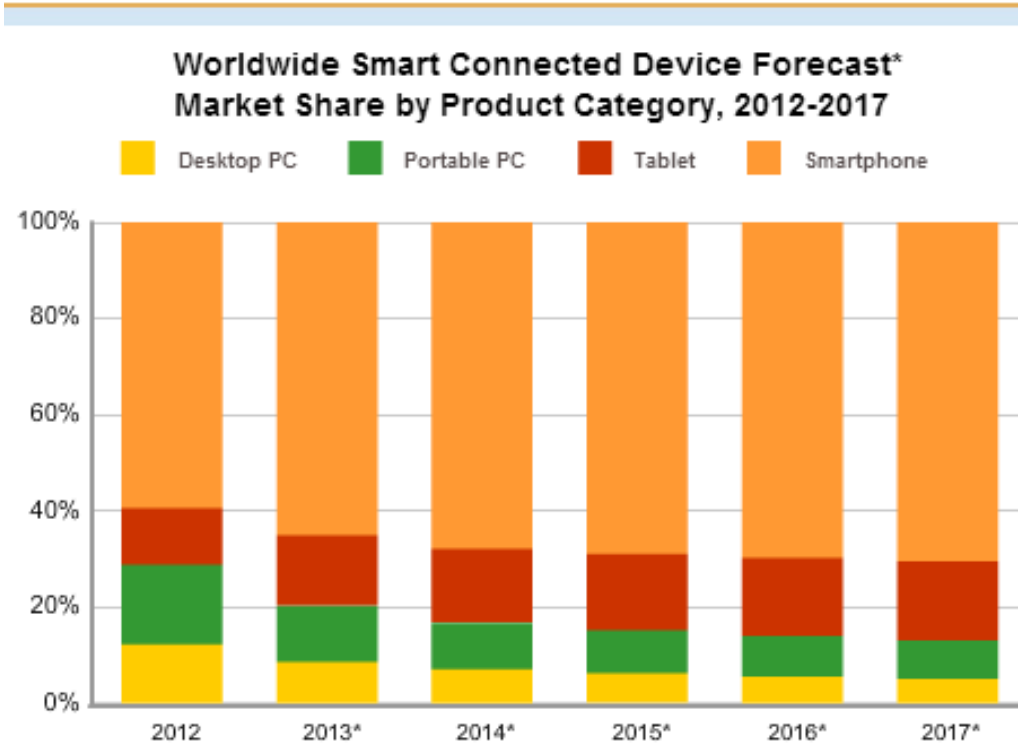
- This course is all about how computers work
- But what do we mean by a computer?
  - Different types: desktop, servers, embedded devices
  - Different uses: automobiles, graphics, finance, genomics…
  - Different manufacturers: Intel, IBM, AMD, Sun…
  - Different underlying technologies and different costs!

Classes of Computers

- Desktop computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
  - In 2010, 2.5% of energy consumption in the US. A further 2.5% required to cool the servers.
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints



International Data Corporation



POST-PC Era: Personal Mobile Device (PMD)

- Battery operated
- Connects to the Internet
- Hundreds of dollars
- Smart phones, tablets, electronic glasses

What to learn

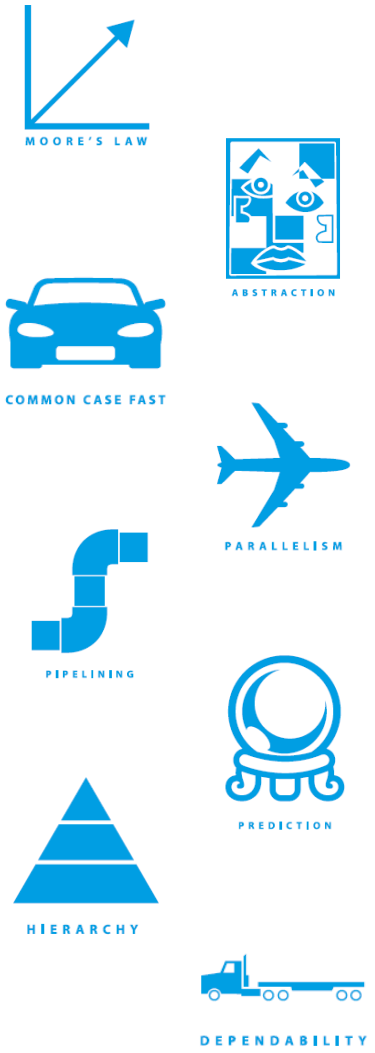
- How are programs written in a high-level language, translated into the language of the hardware
- What is **the interface between the software and the hardware**, and how does software instruct the hardware to perform needed function
- What determines the **performance of a program**
- What techniques can be used by **hardware designers to improve performance**

Understanding Performance

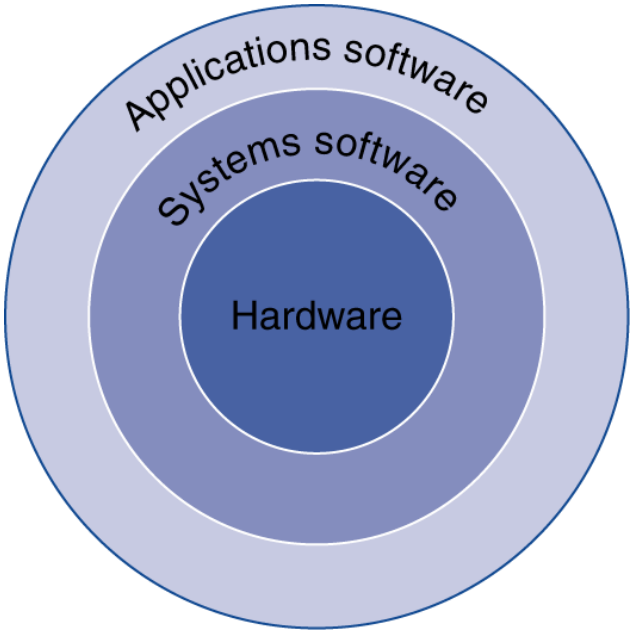
- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

Eight Great Ideas

- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



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

Abstraction

- 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;
}
```

Compiler

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
```

Assembler

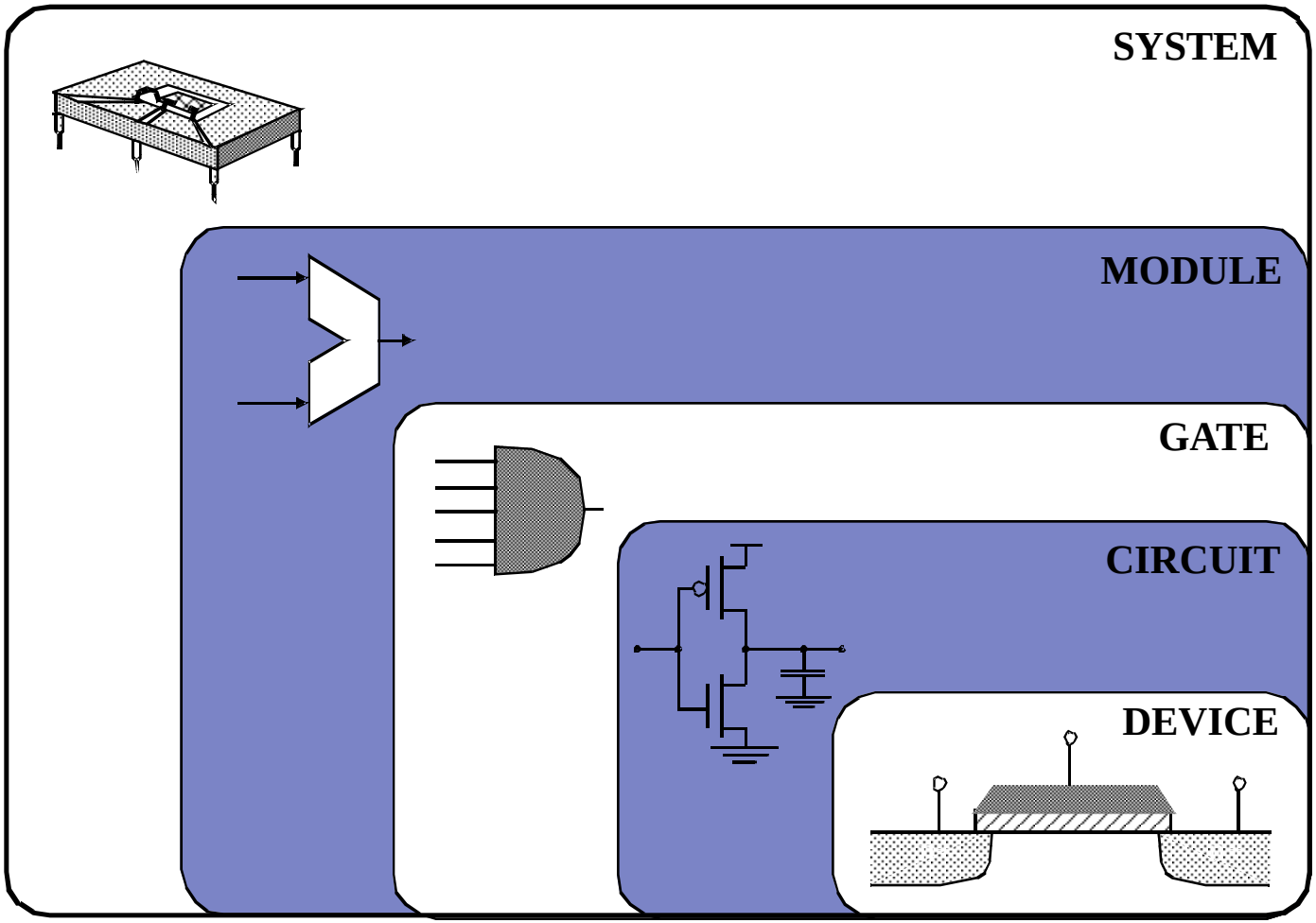
Binary machine language program (for MIPS)

```
00000000101000010000000000011000
00000000000011000000110000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Instruction Set Architecture

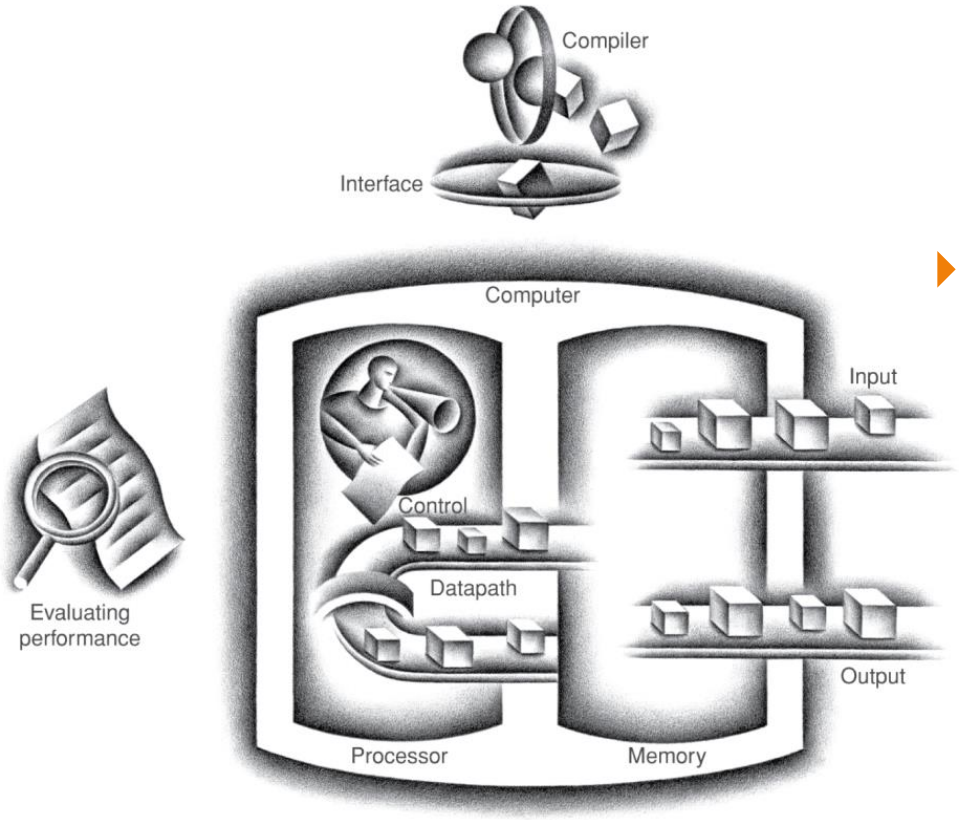
- A very important abstraction
  - interface between hardware and low-level software
  - standardizes instructions, machine language bit patterns, etc.
  - advantage: *different implementations of the same architecture*
  - disadvantage: *sometimes prevents using new innovations*
- Modern instruction set architectures:
  - IA-32, PowerPC, MIPS, SPARC, ARM, and others

Design Abstraction Levels



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

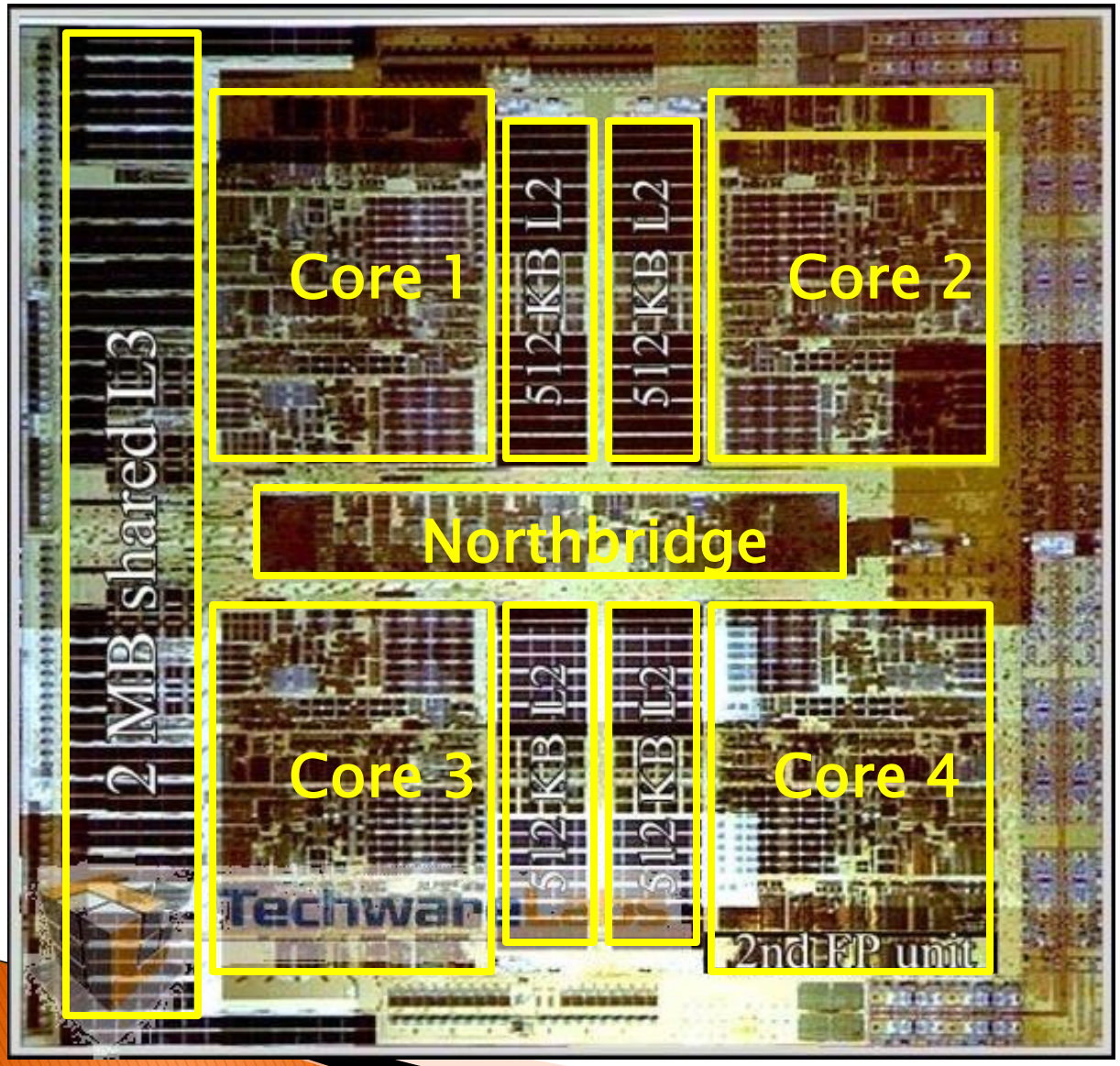
Opening the Box

Capacitive multitouch LCD screen

3.8 V battery

Computer board

Inside the Processor: AMD's Barcelona

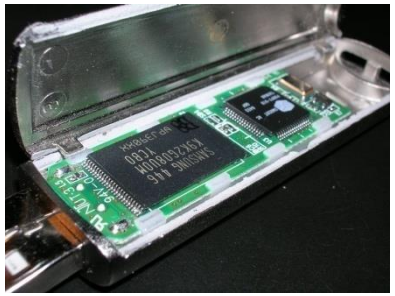


- Four out-of-order cores on one chip
- 1.9 GHz clock rate
- 65nm technology
- Three levels of caches (L1, L2, L3) on chip
- Integrated Northbridge

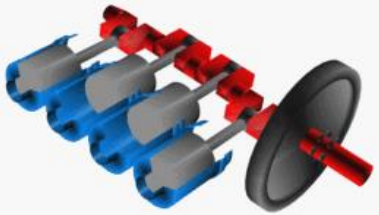
http://www.techwarelabs.com/reviews/processors/barcelona/

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)



Historical Perspective



Al-Jazari

Elephant Clock

The "castle clock", an astronomical clock invented by Al-Jazari in 1206, is considered to be the earliest programmable analog computer.

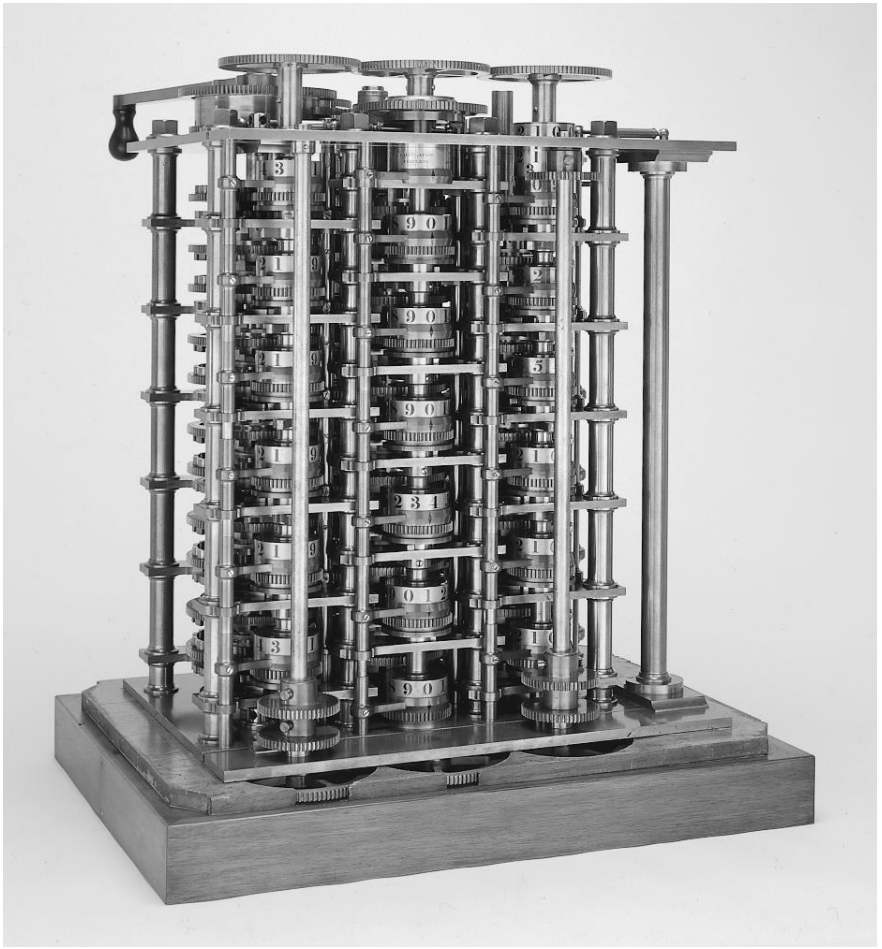
-The zodiac, the solar and lunar orbits

- Crescent moon-shaped pointer travelling across a gateway causing automatic doors to open every hour.

- Five robotic musicians who play music when struck by levers operated by a camshaft attached to a water wheel.

- The length of day and night could be re-programmed every day in order to account for the changing lengths of day and night throughout the year.

Historical Perspective



The First Computer

The Babbage Difference Engine (1832)

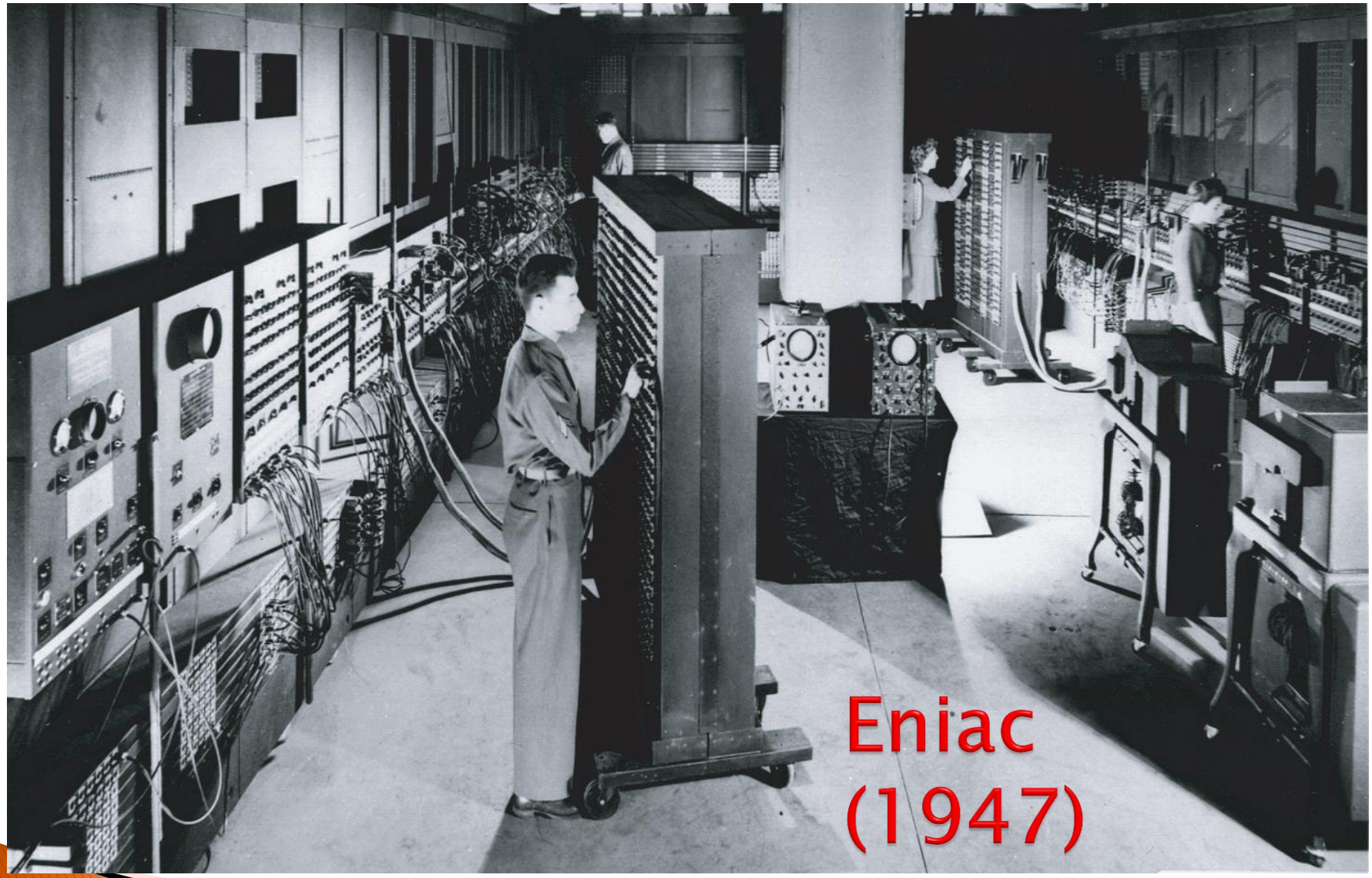
25,000 parts

cost: £17,470

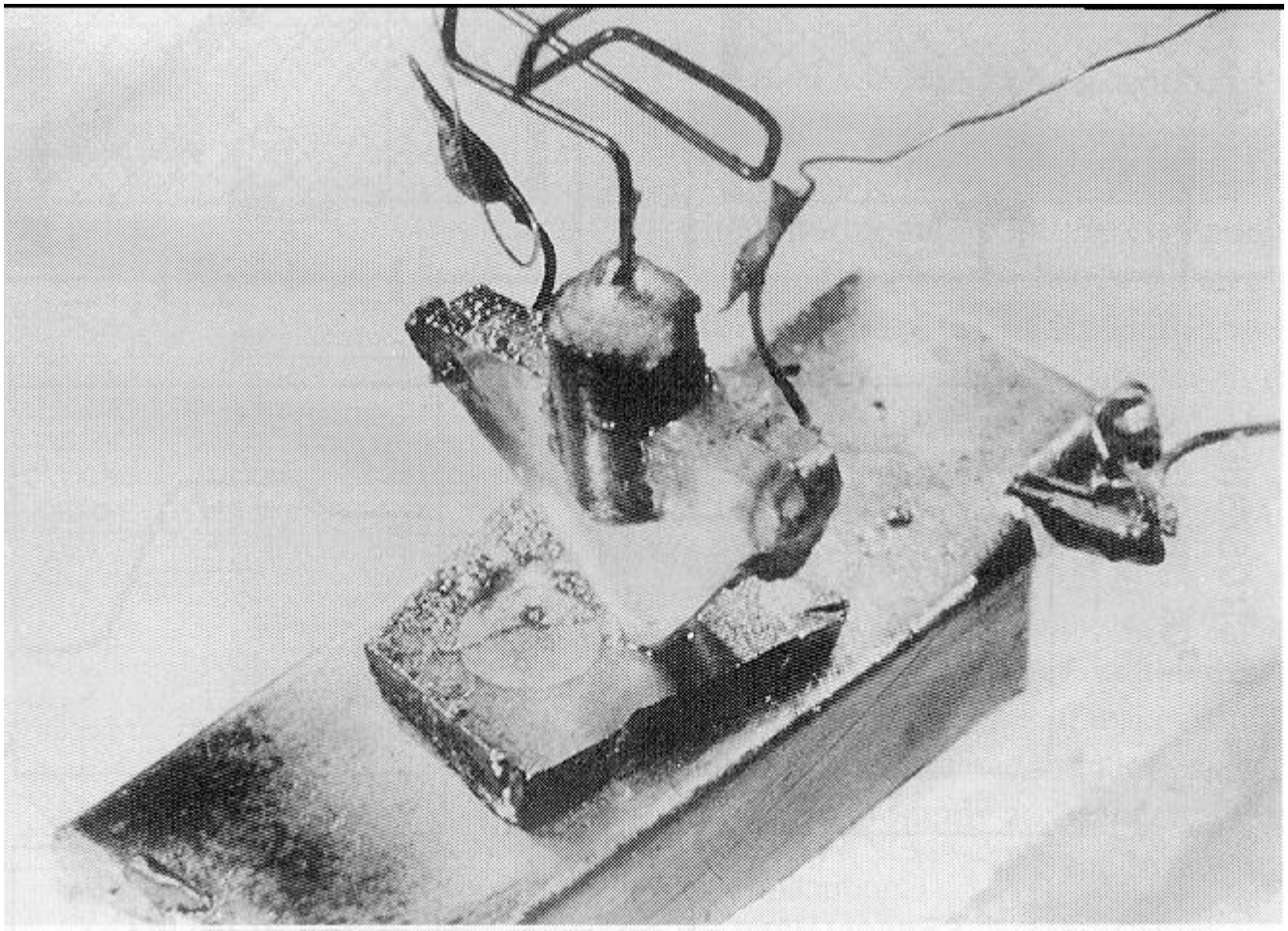
Historical Perspective

- ENIAC built in World War II was the first general purpose computer
  - Used for computing artillery firing tables
  - 30 tonnes
  - 25 m long by 3 m high
  - Each of the twenty 10 digit registers was half meter long
  - Used 18,000 vacuum tubes
  - Performed 1900 additions per second

Historical Perspective



The Transistor Revolution



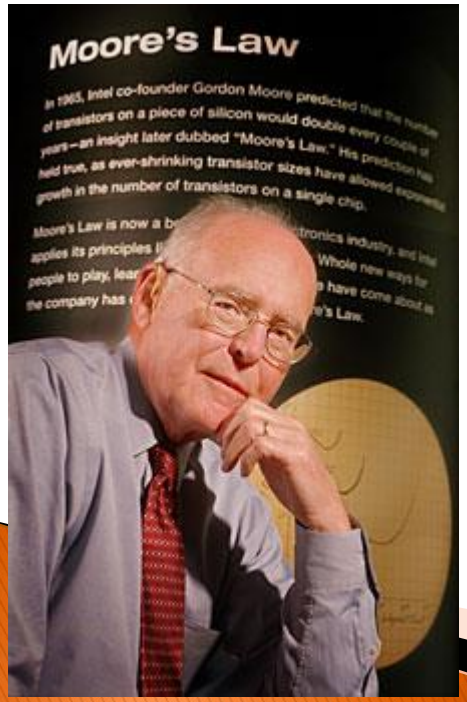
First transistor
Bell Labs, 1948

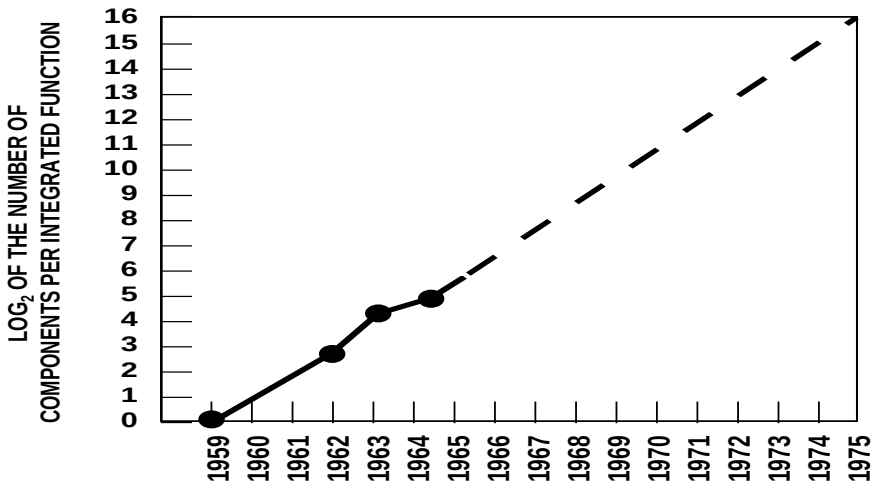
$1 million First Commercial Computer: UNIVAC I (1951)

Moore's Law

- In 1965, Gordon Moore noted that the number of transistors on a chip doubled every 18 to 24 months.

- He made a prediction that semiconductor technology will double its effectiveness every 18 months

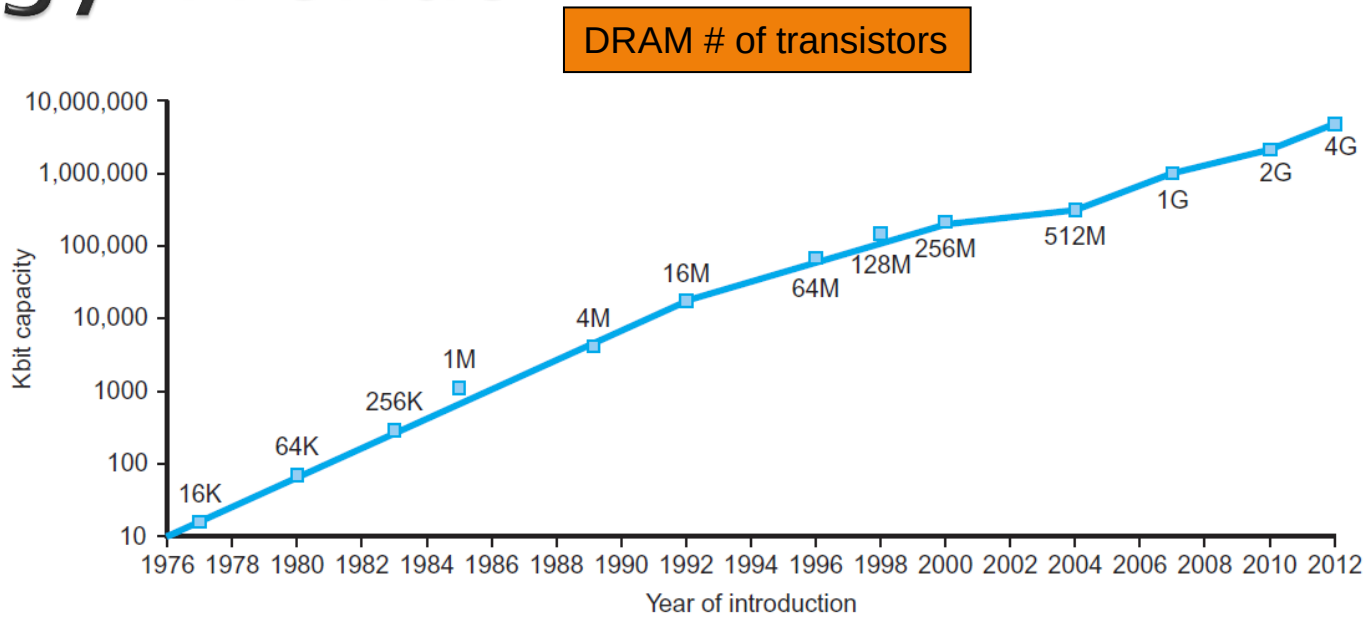




Electronics, April 19, 1965.

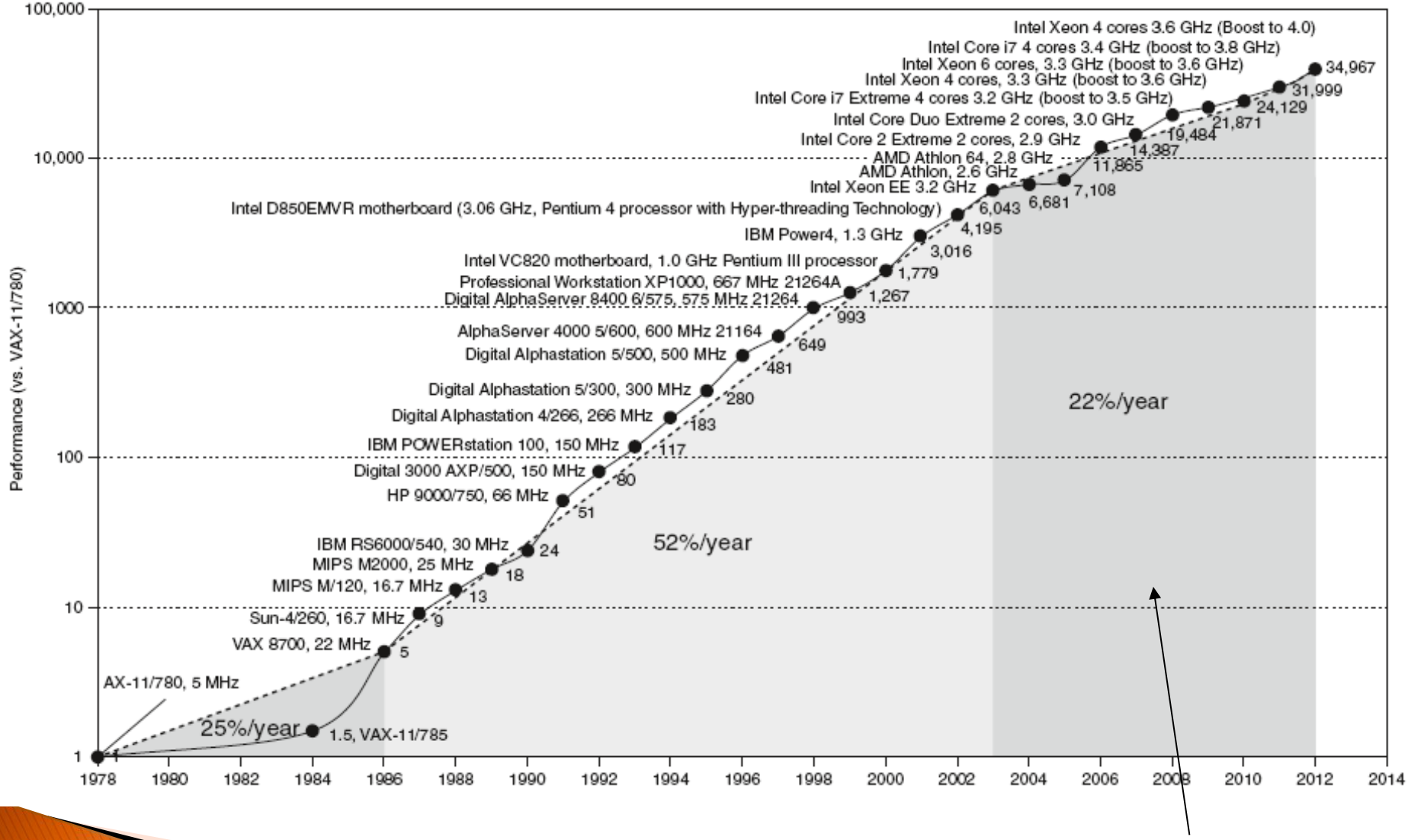
Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



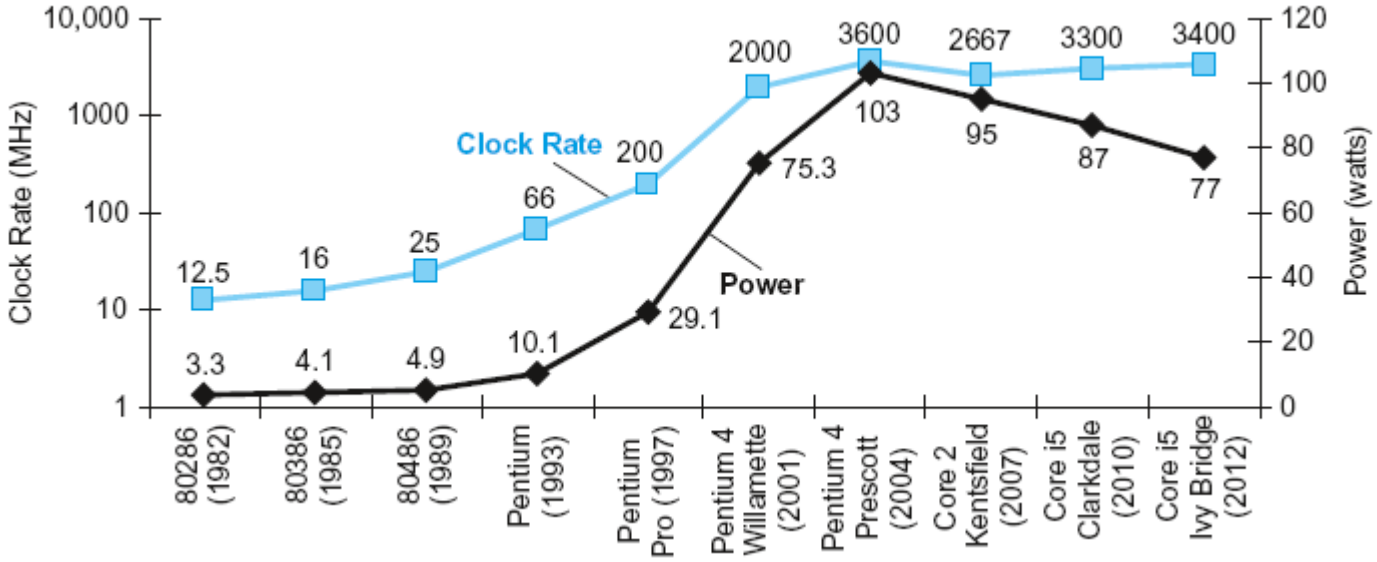
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

Power Trends



- In CMOS IC technology

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

Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

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

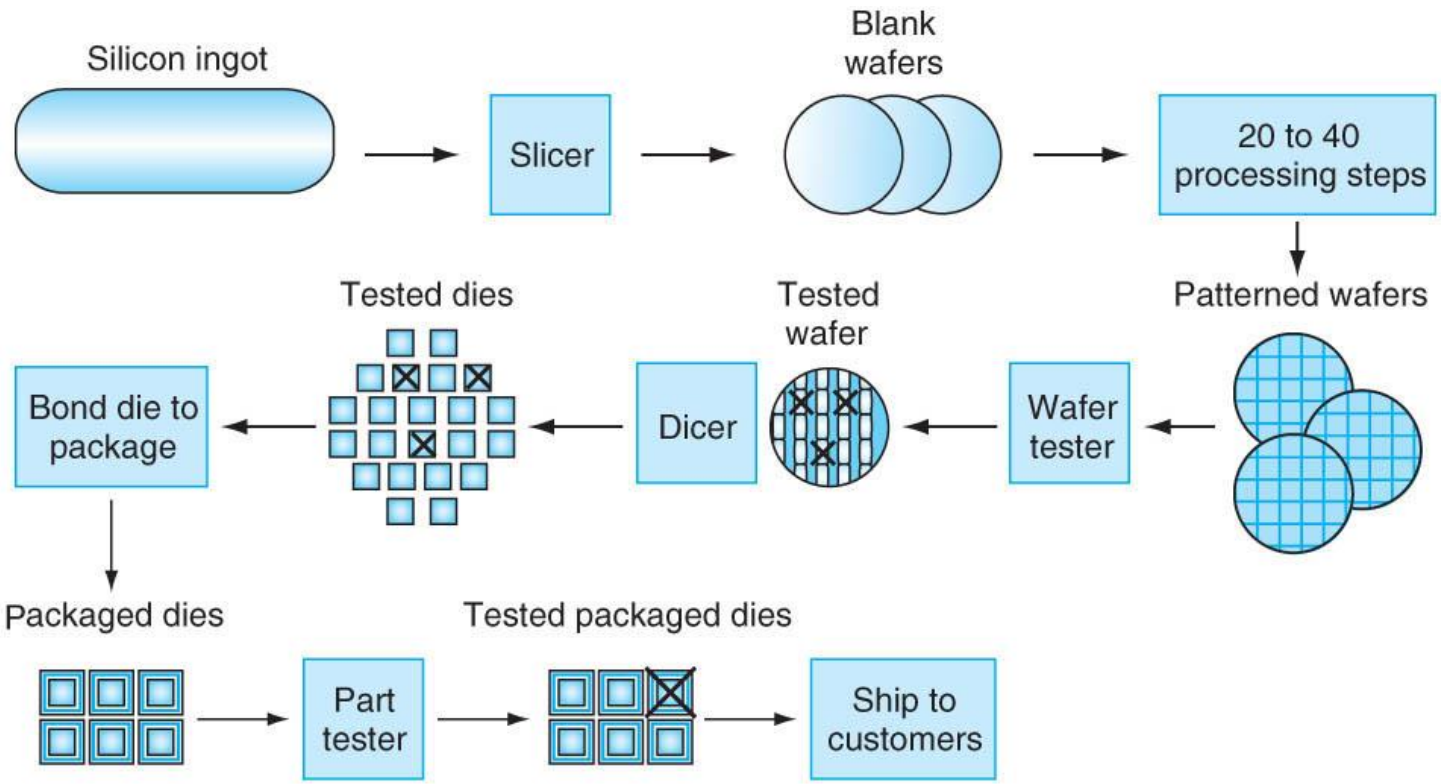
- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip

- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

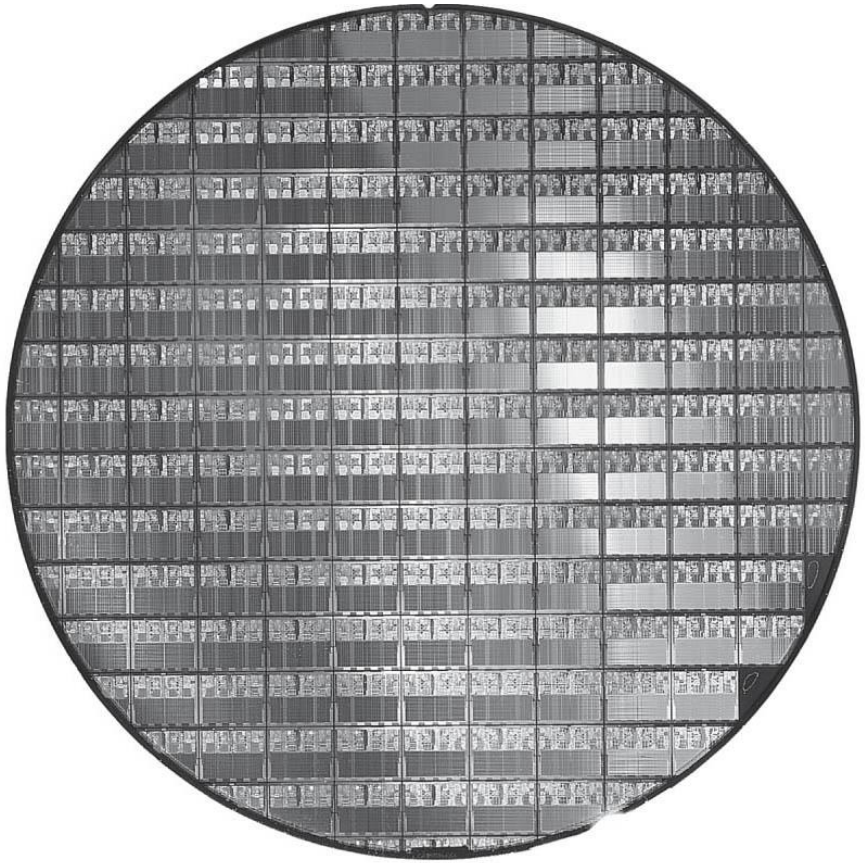
Manufacturing



https://www.youtube.com/watch?v=qm67wbB5GmI

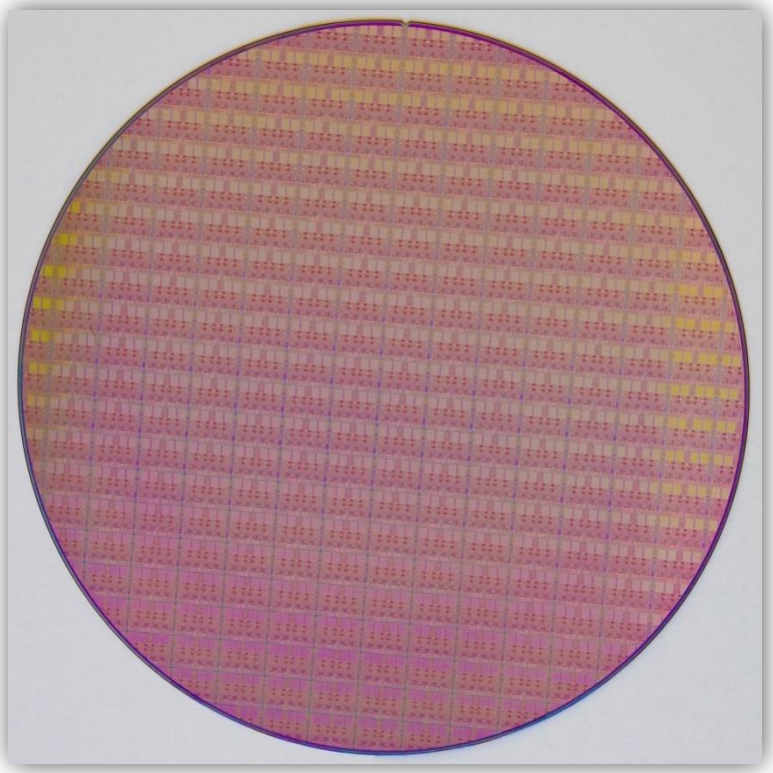
Manufacturing

Single Crystal Silicon Ingot



- X2: 300mm wafer, 117 chips, 90nm technology

Intel Core i7 Wafer



- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

$$Yield = \frac{\text{No. of good chips per wafer}}{\text{Total number of chips per wafer}} \times 100\%$$

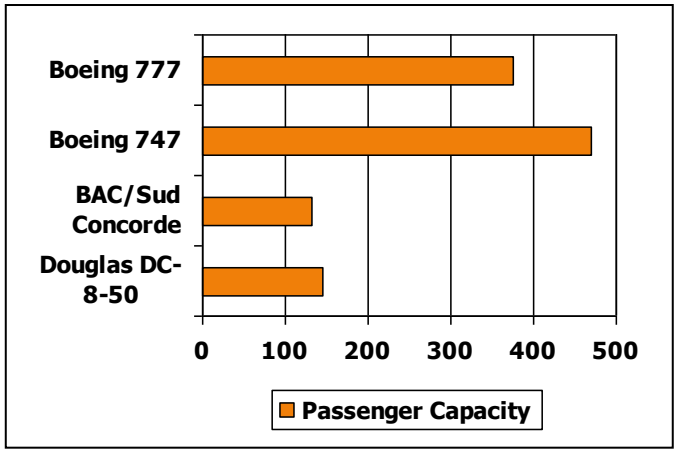
Chip	Metal layers	Line width	Wafer cost	Def./ cm^2	Area mm^2	Dies/ wafer	Yield	Die cost
386DX	2	0.90	$900	1.0	43	360	71%	$4
486 DX2	3	0.80	$1200	1.0	81	181	54%	$12
Power PC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
Super Sparc	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

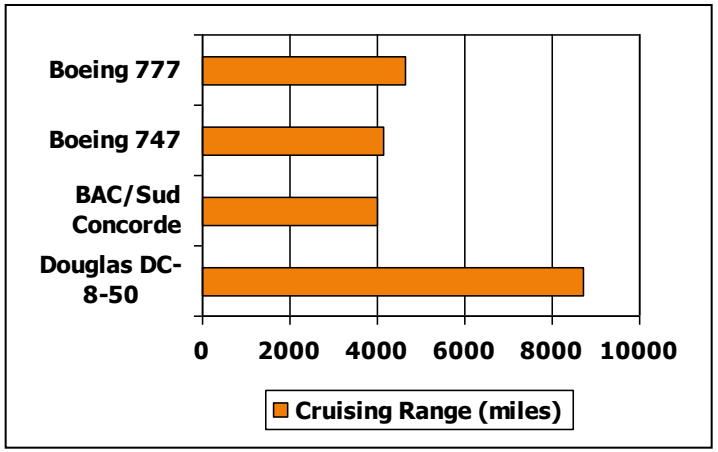
Understanding Performance

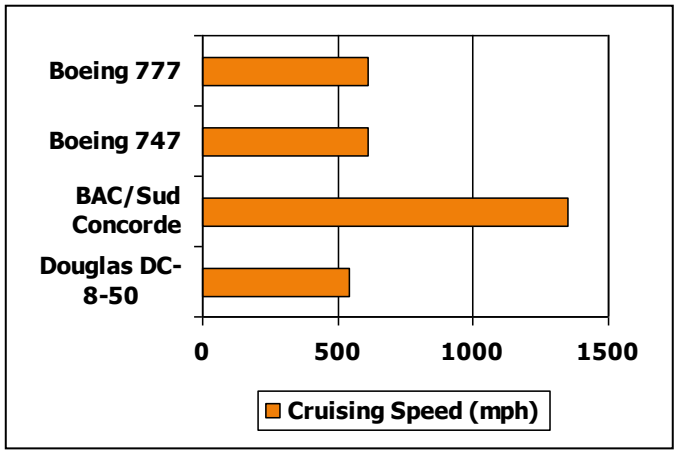
- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

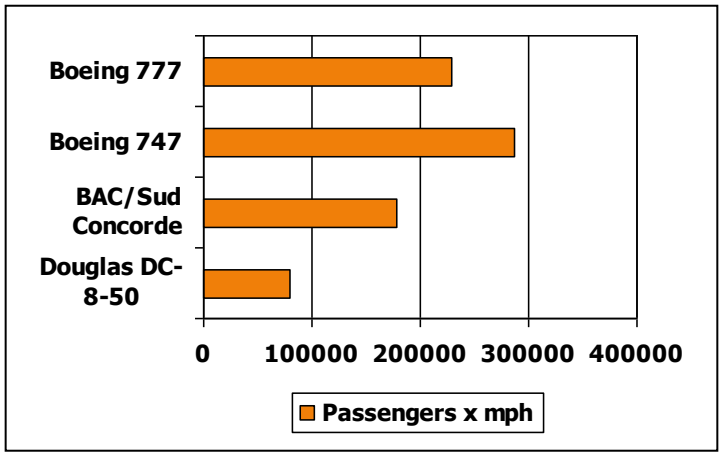
Defining Performance

- 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
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y = \text{Execution time}_Y / \text{Execution time}_X = n$$

- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

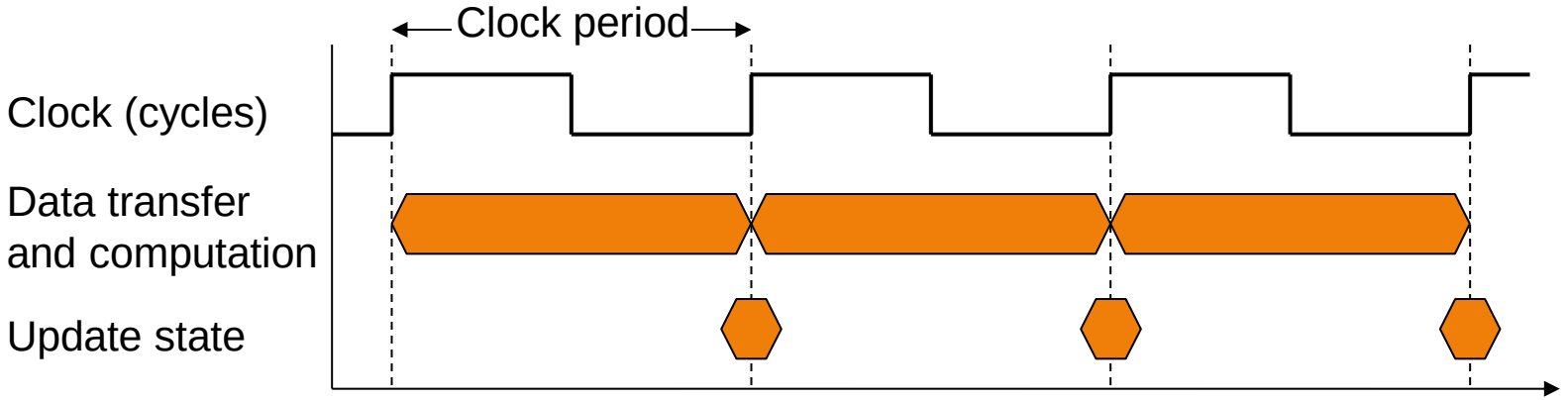
- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

Basic Measurement Metrics

- Comparing Machines
  - Metrics
    - Execution time
    - Throughput
    - CPU time
    - MIPS – millions of instructions per second
    - MFLOPS – millions of floating point operations per second
- Comparing Machines Using Sets of Programs
  - Arithmetic mean, weighted arithmetic mean
  - Benchmarks

CPU Clocking

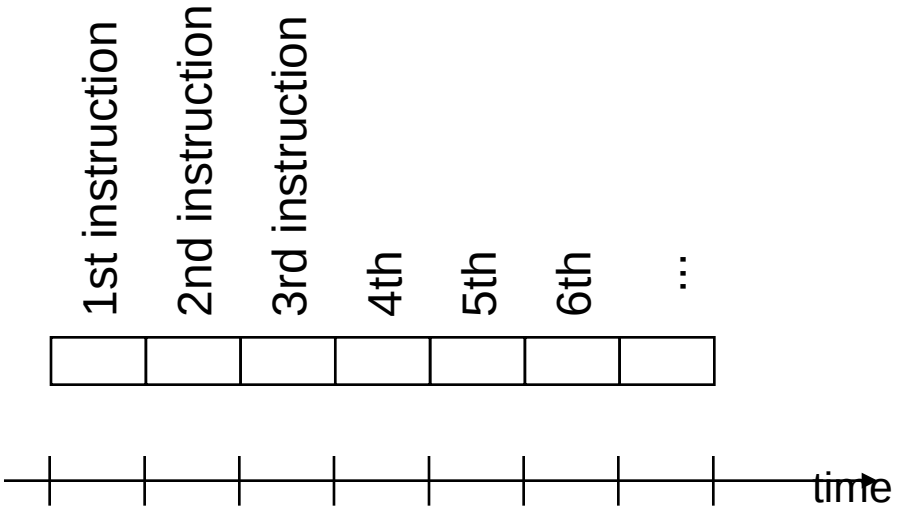
- Operation of digital hardware governed by a constant-rate clock



  - Clock period: duration of a clock cycle
    - e.g., 250ps = 0.25ns = 250×10^{-12}s
  - Clock frequency (rate): cycles per second
    - e.g., 4.0GHz = 4000MHz = 4.0×10^{9}Hz

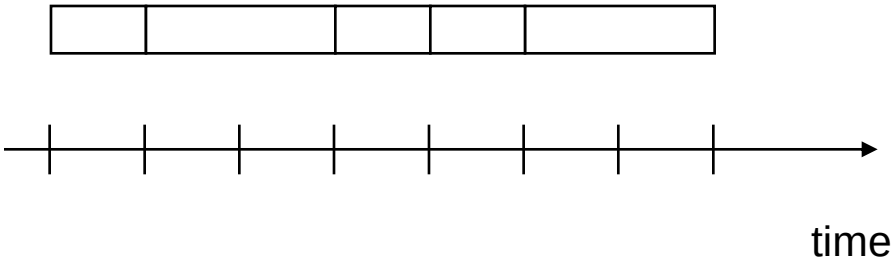
How Many Cycles are Required for a Program?

- Could assume that # of cycles = # of instructions



- This assumption is incorrect, different instructions take different amounts of time on different machines.

Different Numbers of Cycles for Different Instructions

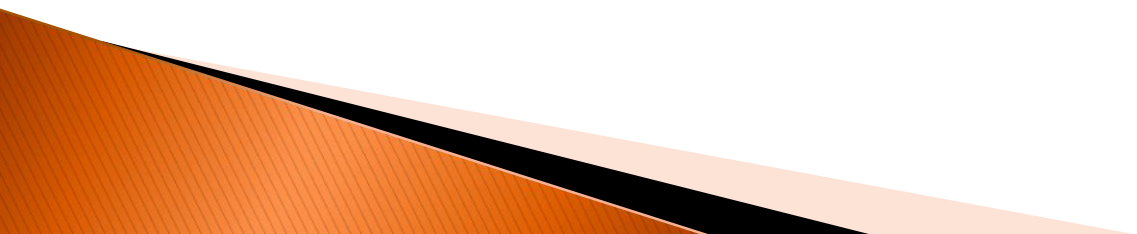


- Division takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers

Now That We Understand Cycles

- A given program will require
  - some number of instructions (machine instructions)
  - some number of clock cycles
  - some number of seconds

- We have a vocabulary that relates these quantities:
  - clock cycle time (seconds per cycle)
  - clock rate (cycles per second)
  - CPI (cycles per instruction)



Computing CPU Time

- The time to execute a given program can be computed as
  CPU time = CPU clock cycles x clock cycle time

  - Since clock cycle time and clock rate are reciprocals
    CPU time = CPU clock cycles / clock rate
  - The number of CPU clock cycles can be determined by
    CPU clock cycles = (instructions/program) x (clock cycles/instruction)
    = Instruction count x CPI
    which gives
    CPU time = Instruction count x CPI x clock cycle time
    CPU time = Instruction count x CPI / clock rate
  - The units for CPU time are

$$\text{CPU time} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{clock cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{clock cycle}}$$

CPU Time Example

- **Example 1:**
  - CPU clock rate is 1 MHz
  - Program takes 45 million cycles to execute
  - What's the CPU time?

45,000,000 * (1 / 1,000,000) = 45 seconds

- **Example 2:**
  - CPU clock rate is 500 MHz
  - Program takes 45 million cycles to execute
  - What's the CPU time

45,000,000 * (1 / 500,000,000) = 0.09 seconds

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
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

Computing CPI

- The CPI is the average number of cycles per instruction.
- If for each instruction type, we know its frequency and number of cycles need to execute it, we can compute the overall CPI as follows:

$$\text{CPI} = \sum \text{CPI} \times \text{F}$$

- For example

Op	F	CPI	CPI x F	% Time
ALU	50%	1	.5	23%
Load	20%	5	1.0	45%
Store	10%	3	.3	14%
Branch	20%	2	.4	18%
Total	100%		2.2	100%

Performance

- Performance is determined by execution time
- Do you think any of the variables is sufficient enough to determine computer performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second

- It is not true to think that one of the variables is indicative of performance.

Number of Instruction Example

- 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 one, two, and three 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?

# of cycles for first code = (2 * 1) + (1 * 2) + (2 * 3) = 10 cycles

# of cycles for second code = (4 * 1) + (1 * 2) + (1 * 3) = 9 cycles

10 / 9 = 1.11 times

CPI for first code = 10 / 5 = 2

CPI for second code = 9 / 6 = 1.5

Performance Example

- Computers M1 and M2 are two implementations of the same instruction set with the same instruction count.

M1 has a clock rate of 50 MHz and M2 has a clock rate of 100 MHz.

M1 has a CPI of 2.8 and M2 has a CPI of 3.2 for a given program.

How many times faster is M2 than M1 for this program?

$$\frac{\text{ExTime}_{M1}}{\text{ExTime}_{M2}} = \frac{\text{IC}_{M1} \times \text{CPI}_{M1} / \text{Clock Rate}_{M1}}{\text{IC}_{M2} \times \text{CPI}_{M2} / \text{Clock Rate}_{M2}} = \frac{2.8/50}{3.2/100} = 1.75$$

- What would the clock rate of M1 have to be for them to have the same execution time?

$$2.8 / \text{Clock Rate}_{M1} = 3.2 / 100 \Longrightarrow \text{Clock Rate}_{M1} = 87.5 \text{ MHz}$$

Problems with Arithmetic Mean

- Applications do not have the same probability of being run
- For example, two machines timed on two benchmarks:

	Machine A	Machine B
Program 1	2 seconds (%20)	6 seconds (20%)
Program 2	12 seconds (%80)	10 seconds (%80)

Average execution $time_A$ = (2 + 12) / 2 = 7 seconds

Average execution $time_B$ = (6 + 10) / 2 = 8 seconds

Weighted average execution $time_A$ = 2*0.2 + 12*0.8 = 10 seconds

Weighted average execution $time_B$ = 6*0.2 + 10*0.8 = 9.2 seconds

Poor Performance Metrics

- Marketing metrics for computer performance included MIPS and MFLOPS

- MIPS : millions of instructions per second
  - MIPS = instruction count / (execution time x 10^6)
  - For example, a program that executes 3 million instructions in 2 seconds has a MIPS rating of 1.5
  - Advantage : Easy to understand and measure
  - Disadvantages : May not reflect actual performance, since simple instructions do better.

- MFLOPS : millions of floating point operations per second

MIPS Example

- Two different compilers are being tested for a 500 MHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software.

  The first compiler's code uses 5 billions Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.

  The second compiler's code uses 10 billions Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.

- Which sequence will be faster according to MIPS?

- Which sequence will be faster according to execution time?

MIPS Example (Con't)

	Instruction counts (in billions) for each instruction class		
Code from	A (1 cy)	B (2 cy)	C (3 cy)
Compiler 1	5	1	1
Compiler 2	10	1	1

CPU Clock cycles$_1$ = (5 x 1 + 1 x 2 + 1 x 3) x 10^9 = 10 x 10^9

CPU Clock cycles$_2$ = (10 x 1 + 1 x 2 + 1 x 3) x 10^9 = 15 x 10^9

CPU time$_1$ = 10 x 10^9 / 500 x 10^6 = 20 seconds

CPU time$_2$ = 15 x 10^9 / 500 x 10^6 = 30 seconds

MIPS$_1$ = (5 + 1 + 1) x 10^9 / 20 x 10^6 = 350

MIPS$_2$ = (10 + 1 + 1) x 10^9 / 30 x 10^6 = 400

Performance Summary

- The two main measure of performance are
  - execution time : time to do the task
  - throughput : number of tasks completed per unit time

- Performance and execution time are reciprocals. Increasing performance, decreases execution time.

- The time to execute a given program can be computed as:

  CPU time = Instruction count x CPI x clock cycle time

  CPU time = Instruction count x CPI / clock rate

- These factors are affected by compiler technology, the instruction set architecture, the machine organization, and the underlying technology.

- When trying to improve performance, look at what occurs frequently => make the common case fast.

Example

- Assume that a program runs in 100 seconds on a machine, with multiply operations responsible for 80 seconds. How much do I have to improve the speed of multiplication if I want my program to run 2 times faster.

Execution time after improvement =

$$\frac{\text{Execution time affected by improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

$$50 \text{ seconds} = \frac{80 \text{ seconds}}{n} + (100 - 80 \text{ seconds})$$

$$n = \frac{80 \text{ seconds}}{30 \text{ seconds}} = 2.67$$

Amdahl's Law

- **Pitfall:** Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- Corollary: make the common case fast

Amdahl's Law

- Speedup due to an enhancement is defined as:

$$\text{Speedup} = \frac{\text{ExTime old}}{\text{ExTime new}} = \frac{\text{Performance new}}{\text{Performance old}}$$

- Suppose that an enhancement accelerates a fraction

- $\textbf{Fraction}_{\textbf{enhanced}}$ of the task by a factor $\textbf{Speedup}_{\textbf{enhanced}}$

$$\textbf{ExTime}_{\textbf{new}} = \textbf{ExTime}_{\textbf{old}} \times \left[(1 - \textbf{Fraction}_{\textbf{enhanced}}) + \frac{\textbf{Fraction}_{\textbf{enhanced}}}{\textbf{Speedup}_{\textbf{enhanced}}} \right]$$

$$\textbf{Speedup} = \frac{\textbf{ExTime}_{\textbf{old}}}{\textbf{ExTime}_{\textbf{new}}} = \frac{1}{(1 - \textbf{Fraction}_{\textbf{enhanced}}) + \frac{\textbf{Fraction}_{\textbf{enhanced}}}{\textbf{Speedup}_{\textbf{enhanced}}}}$$

Example of Amdahl's Law

- Floating point instructions are improved to run twice as fast, but only 10% of the time was spent on these instructions originally. How much faster is the new machine?

$$\text{Speedup} = \frac{\text{ExTime}_{old}}{\text{ExTime}_{new}} = \frac{1}{(1 - \text{Fraction}_{enhanced}) + \frac{\text{Fraction}_{enhanced}}{\text{Speedup}_{enhanced}}}$$

$$\text{Speedup} = \frac{1}{(1 - 0.1) + 0.1/2} = 1.053$$

  - The new machine is 1.053 times as fast, or 5.3% faster.

  - How much faster would the new machine be if floating point instructions become 100 times faster?

$$\text{Speedup} = \frac{1}{(1 - 0.1) + 0.1/100} = 1.109$$

Estimating Performance Improvements

- Assume a processor currently requires 10 seconds to execute a program and processor performance improves by 50 percent per year.

- By what factor does processor performance improve in 5 years?

  $(1 + 0.5)^5 = 7.59$

- How long will it take a processor to execute the program after 5 years?

  $\text{ExTime}_{\text{new}} = 10/7.59 = 1.32$ seconds

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

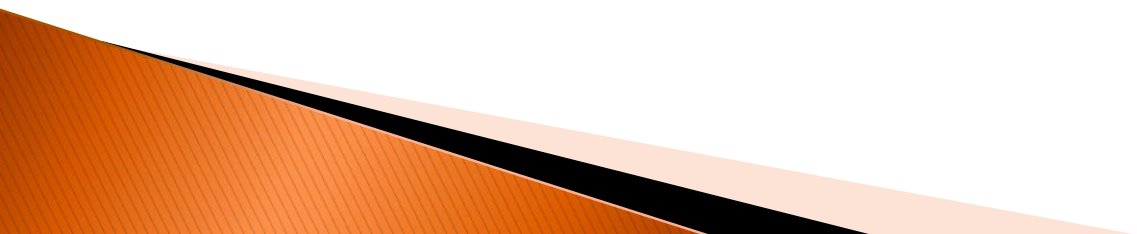
CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

Summary of Performance Evaluation

- Response Time and Throughput
- CPU time = # instr x CPI x cycle time

- MIPS and MFLOPS poor performance metrics
- Amdahl's law provides an efficient method for determining speedup due to an enhancement.
- SPEC benchmarks to measure CPU performance



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