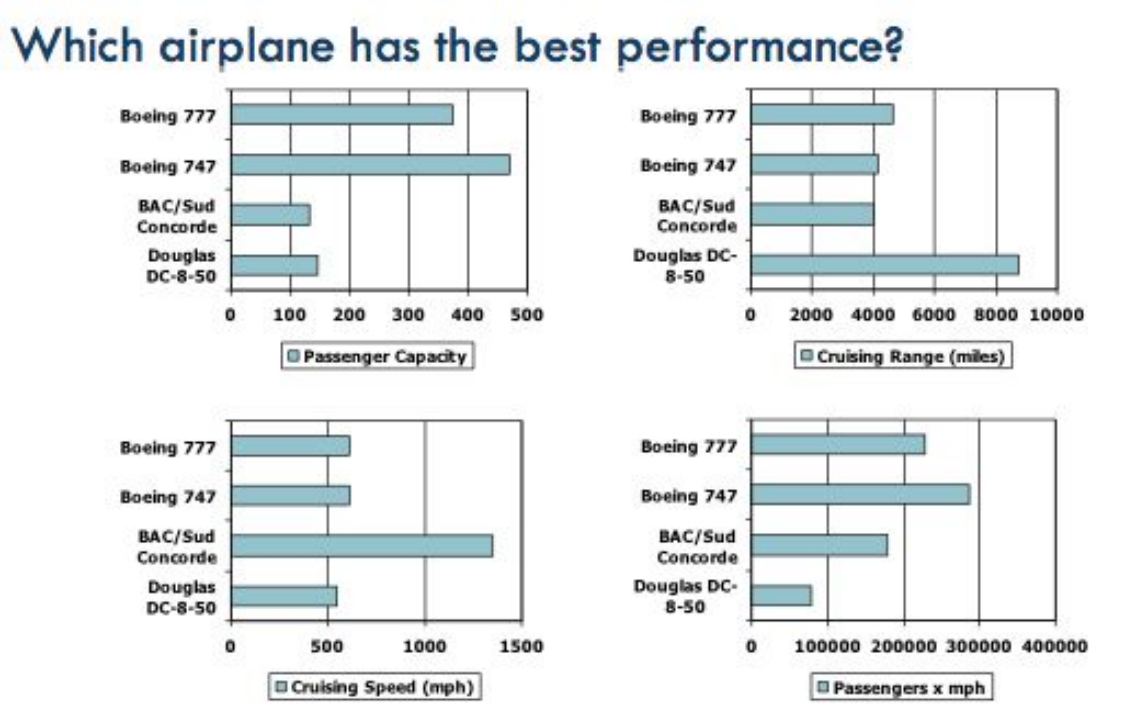
Defining performance



Assessing performance

- response time (execution time)
- throughput (bandwidth)

Comparing systems:

- relative performance

Response time and throughput

Response time (execution time)

- how long it takes to do a task
- example: one program running on a computer

Throughput (bandwidth)

- total work done per unit of time

Response time and throughput

How are response time and throughput affected by:

1. replace the processor with a faster version?
2. adding more processors?

Relative performance

- Computer A runs a program in 10s, computer B is 15s
- How much faster is A than B?
- 15/10 = 1.5
- A is 1.5 times faster than B

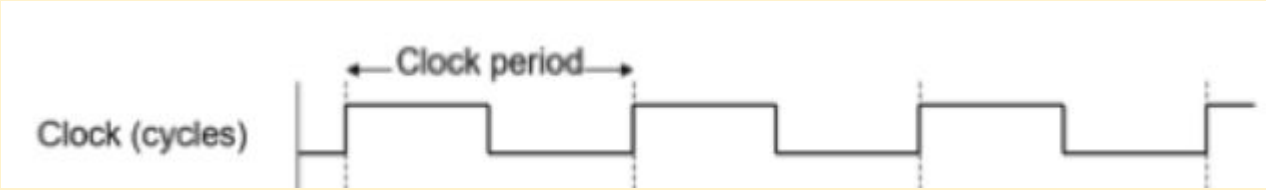
$$\frac{\text{Execution time}_B}{\text{Execution time}_A} = n$$

Measuring execution time

- What do we mean by execution time?
  - wall clock time?
  - response time?
- The CPU may be doing other things besides executing our program
  - system performance: elapsed time (includes system overhead)
  - CPU performance: one program's CPU time

Clock Cycle

- a clock cycle is the time for one clock period of the CPU, usually a constant rate
- an example CPU clock cycle or period might be 250 ps (picoseconds), and this corresponds to 4 GHz (gigahertz) clock rate
- the clock cycle and clock rate are inverses



Measures of time

Measure	Symbol	Meaning
second	s	one second
millisecond	ms	one thousandth of a second
microsecond	µs	one millionth of a second
nanosecond	ns	one billionth of a second
picosecond	ps	one trillionth of a second

Frequency

- Hertz (Hz) is a measure of frequency; it is cycles per second, the inverse of the cycle period

Frequency	Symbol	Hz	Cycles/second	Period
Hertz	Hz	1	1	1 s
Kilohertz	KHz	10^{-3}	1,000	1 ms
Megahertz	MHz	10^{-6}	1,000,000	1 µs
Gigahertz	GHz	10^{-9}	1,000,000,000	1 ns
Terahertz	THz	10^{-12}	1,000,000,000,000	1 ps

CPU time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

Performance improved by:

- reducing the number of clock cycles
- increasing the clock rate

Hardware designed must often trade-off clock rate and cycle count

CPU time

- instruction count for a program: determined by program, ISA, and compiler
- CPI is the average cycles per instruction: determined by CPU hardware

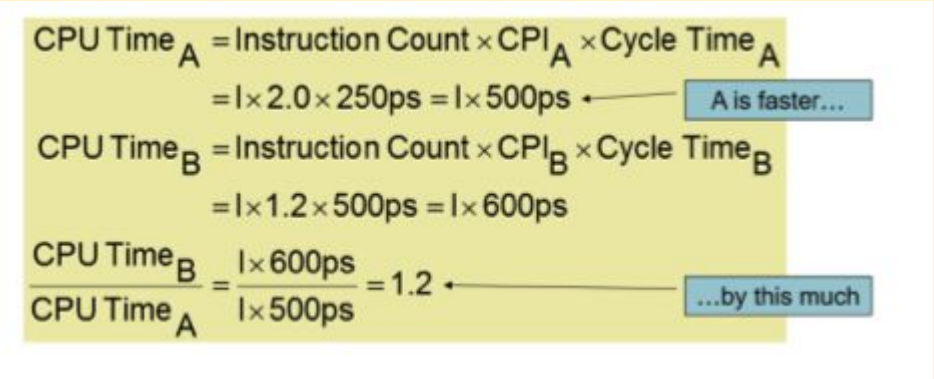
$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

CPI example

- Computer A: cycle time = 250 ps, CPI = 2.0
- Computer B: cycle time = 500 ps, CPI = 1.2
- Same ISA.
- Which is faster?



CPI in more detail

- When different instructions take different number of cycles:

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

- Weighted average CPI

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

CPU time

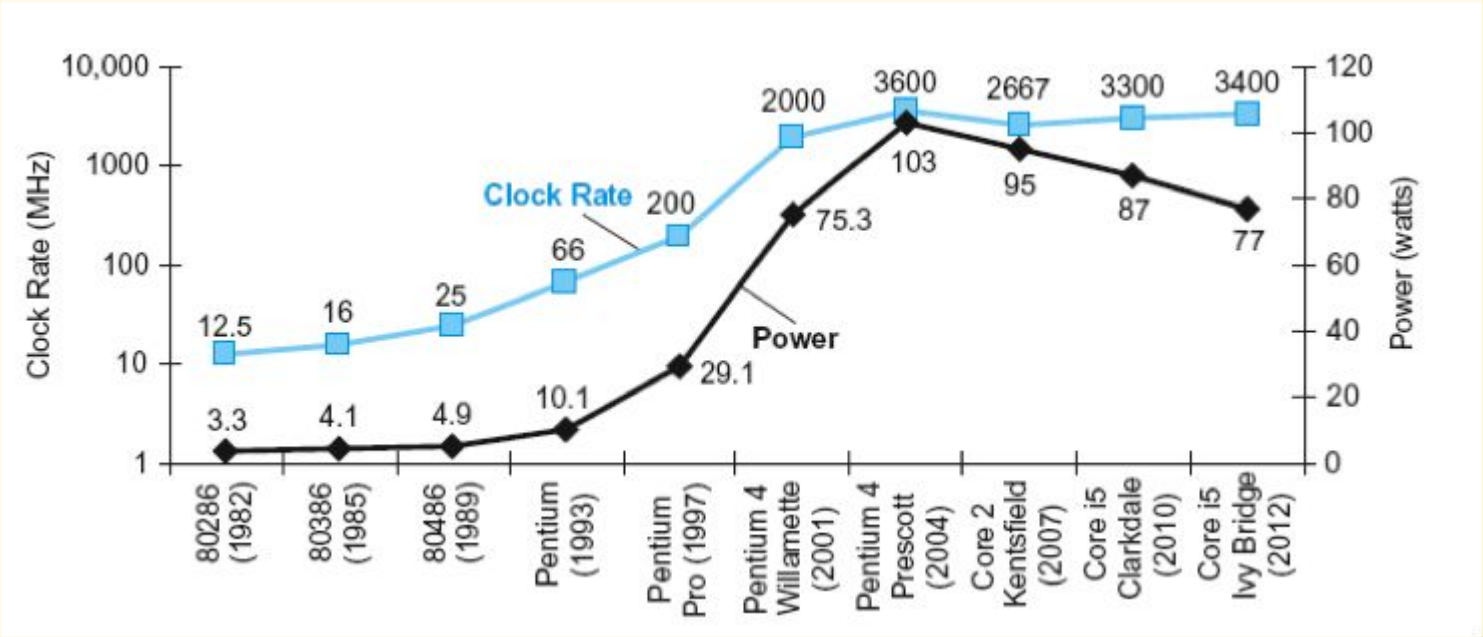
$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

Performance depends on:

- algorithm: affects IC, possibly CPI
- programming language: affects IC, CPI
- compiler: affects IC, CPI
- ISA: affects IC, CPI, clock cycle time

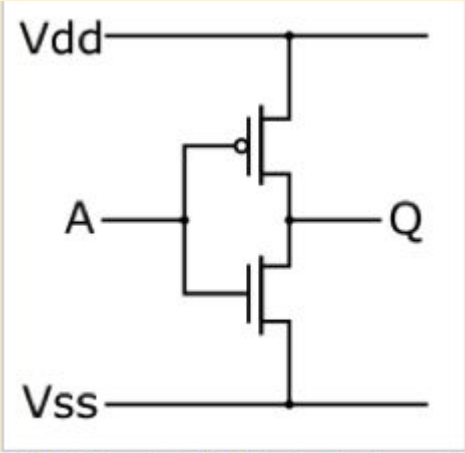
Power trends

- Power and clock rate are correlated; power wall limited clock rate



CMOS - complementary metal oxide semiconductor

- material for making low-power chips
- high noise immunity
- draws most power when switching but uses power all the time
- increasing the number of transistors increases power dissipation

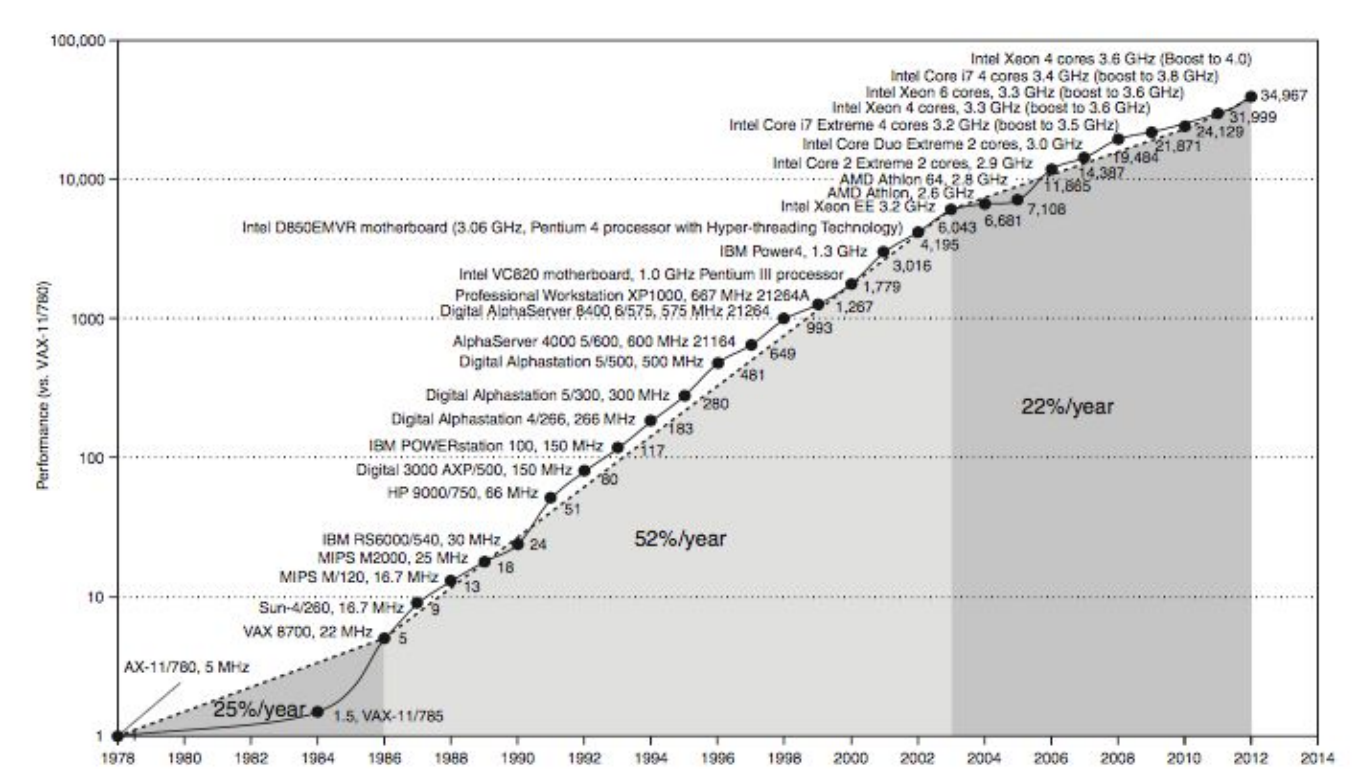


CMOS inverter (NOT logic gate)

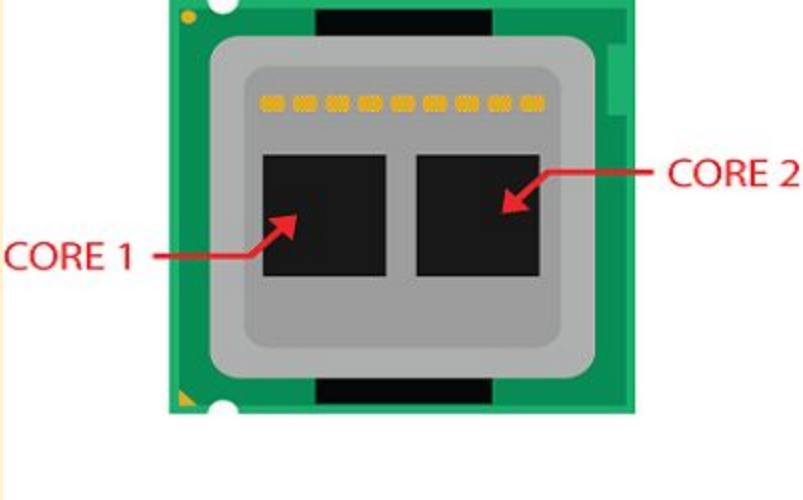
Power issues

- Joules is an energy metric
- watts is joules/second
- decreasing power consumption is a major concern for mobile devices
- power is dissipated as heat; the heat has to be cooled, causing even more power usage by fans
- the power wall limited how fast uniprocessors could get

Uniprocessor performance



Multiprocessors



- Multicore processors have more than one processor (core) per chip
- How to exploit parallel architecture:
- concurrency - like an os doing many things at once
- parallelism - dividing a problem into units that can be run at the same time

Multicore processors

- Some recent Intel CPUs:

	i7-8650U	i7-8550U	i5-8350U	i5-8250U
Max Clock	4.2GHz	4.0GHz	3.6GHz	3.4GHz
Base Clock	1.9GHz	1.8GHz	1.7GHz	1.6GHz
1-Core Turbo Boost	4.2GHz	4.0GHz	3.6GHz	3.4GHz
2-Core Turbo Boost	4.2GHz	4.0GHz	3.6GHz	3.4GHz
4-Core Turbo Boost	3.9GHz	3.7GHz	3.6GHz	3.4GHz

SPEC CPU benchmarking

- SPEC = system performance evaluation cooperative
- set of program used to compare performance
- compared to a reference machine and summarized as a geometric mean of these relative performance ratios

Intel i7

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

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

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

SPEC Power Benchmark

- Power consumption of Intel Xeon at different workload details
- ssj_ops (server side Java) application operations per second per watt
- note that power does not scale down

Target Load %	Performance (ssj_ops)	Average Power (watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1922
Σssj_ops / Σpower =		2490

Amdahl's law

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

- quantifies the overall system improvement for an improvement in one area
- Example: An application has 40% floating-point multiply instructions. If you make the multiply 5 times faster, how does this impact overall performance?
- 40/5 + 60 (unaffected) = 68
- the improvement would result in execution only 68% of the original time

MIPS (millions of instructions/second)

- not to be confused with MIPS ISA
- MIPS is not a good metric because it doesn't account for differences in ISAs and how long instructions take to execute

MIPS (millions of instructions/second)

- not to be confused with MIPS ISA
- MIPS is not a good metric because it doesn't account for differences in ISAs and how long instructions take to execute

Summary