
COMPUTER ORGANIZATION (IS F242)

LECT 21: PERFORMANCE

CPU Time

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

- Performance improved by
 - Reducing number of clock cycles
 - Increasing clock rate
 - Hardware designer must often trade off clock rate against cycle count

Instruction Count and CPI

Clock Cycles = Instruction Count \times Cycles per Instruction

CPU Time = Instruction Count \times CPI \times Clock Cycle Time

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

- Instruction Count for a program
 - Determined by program, ISA and compiler
- Average cycles per instruction
 - Determined by CPU hardware
 - If different instructions have different CPI
 - Average CPI affected by instruction mix

The Processor Performance Equation

Clock Cycles = Instruction Count \times Cycles per Instruction

CPU Time = Instruction Count \times CPI \times Clock Cycle Time

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

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

Different instruction types having different CPIs

$$\text{CPU clock cycles} = \sum_{i=1}^n \text{IC}_i \times \text{CPI}_i$$

$$\text{CPU time} = \left(\sum_{i=1}^n \text{IC}_i \times \text{CPI}_i \right) \times \text{Clock cycle time}$$

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps} \end{aligned}$$

← A is faster...

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

— ...by this much

Calculate CPI for each sequence

	CPI for each instruction class		
	A	B	C
CPI	1	2	3

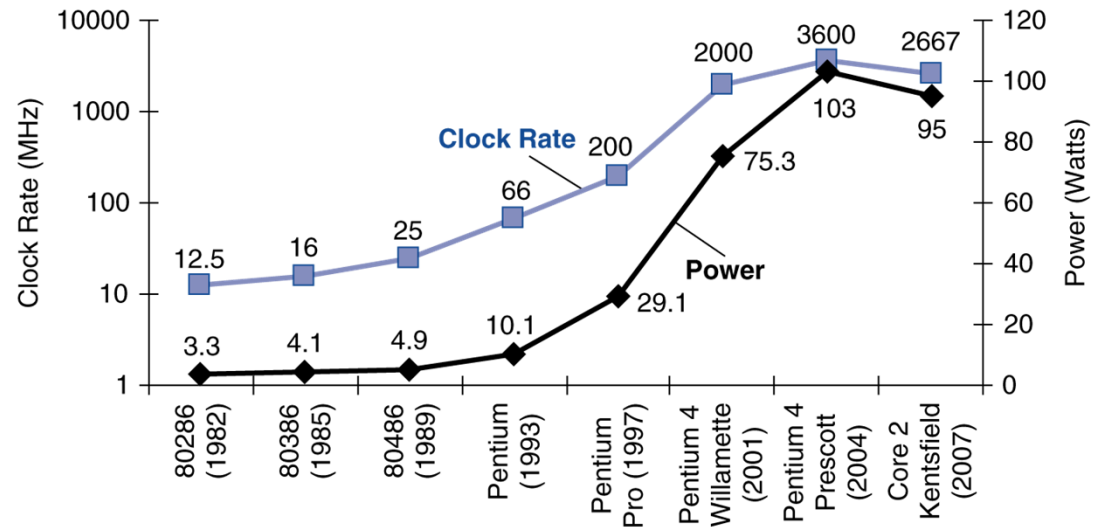
Code sequence	Instruction counts for each instruction class		
	A	B	C
1	2	1	2
2	4	1	1

$$\text{CPU clock cycles} = \sum_{i=1 \text{ to } n} (\text{CPI}_i * C_i)$$

$$\text{CPU clock cycles}_1 = 10 \text{ cycles}, \text{CPU clock cycles}_2 = 9 \text{ cycles}$$

$$\text{CPI} = \text{CPU clock cycles} / \text{Instruction count.} \quad \text{CPI}_1 = 10/5, \text{CPI}_2 = 9/6$$

Power Trends



■ In CMOS IC technology

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

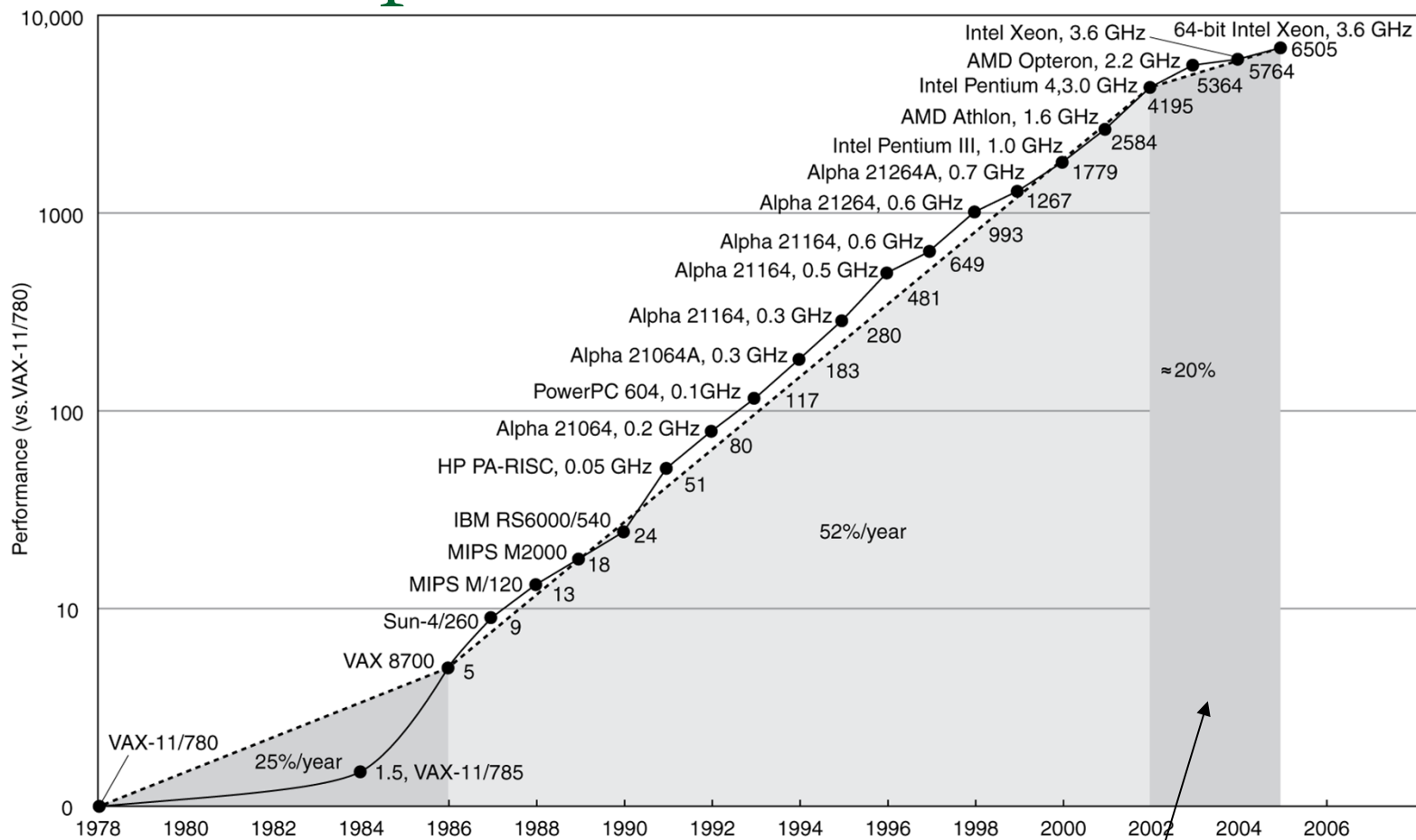
Reducing Power

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

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

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

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

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

Benchmarks

- Performance best determined by running a real application
 - use programs typical of expected workload
 - or, typical of expected class of applications
e.g., compilers/editors, scientific applications, graphics, etc.
- Small benchmarks
 - nice for architects and designers
 - easy to standardize
 - can be abused!
- Benchmark suites
 - Perfect Club: set of application codes
 - Livermore Loops: 24 loop kernels
 - Linpack: linear algebra package
 - SPEC: mix of code from industry organization

SPEC (System Performance Evaluation Corporation)

- Sponsored by industry but independent and self-managed – trusted by code developers and machine vendors
- Clear guides for testing, see www.spec.org
- Regular updates (benchmarks are dropped and new ones added periodically according to relevance)
- Specialized benchmarks for particular classes of applications
- Can still be abused..., by selective optimization!

SPEC History

- First Round: SPEC CPU89
 - 10 programs yielding a single number
- Second Round: SPEC CPU92
 - SPEC CINT92 (6 integer programs) and SPEC CFP92 (14 floating point programs)
 - compiler flags can be set differently for different programs
- Third Round: SPEC CPU95
 - new set of programs: SPEC CINT95 (8 integer programs) and SPEC CFP95 (10 floating point)
 - single flag setting for all programs
- Fourth Round: SPEC CPU2000
 - new set of programs: SPEC CINT2000 (12 integer programs) and SPEC CFP2000 (14 floating point)
 - single flag setting for all programs
 - programs in C, C++, Fortran 77, and Fortran 90

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)

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

CINT2006 for Opteron X4 2356

Name	Description	IC×10 ⁹	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.47	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.48	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates