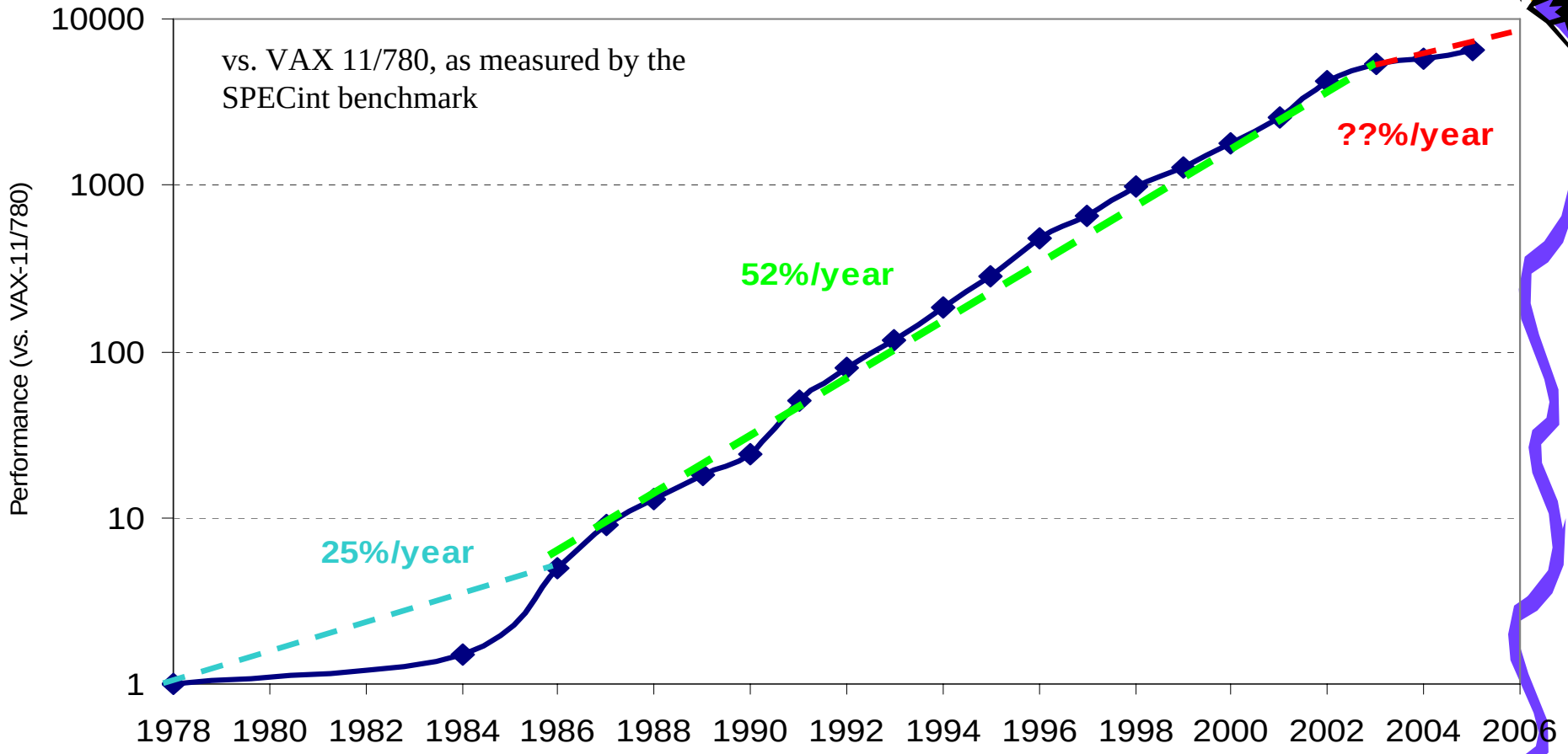
5008: Computer Architecture

Chapter 1 – Fundamentals of Computer Design

Processor Performance



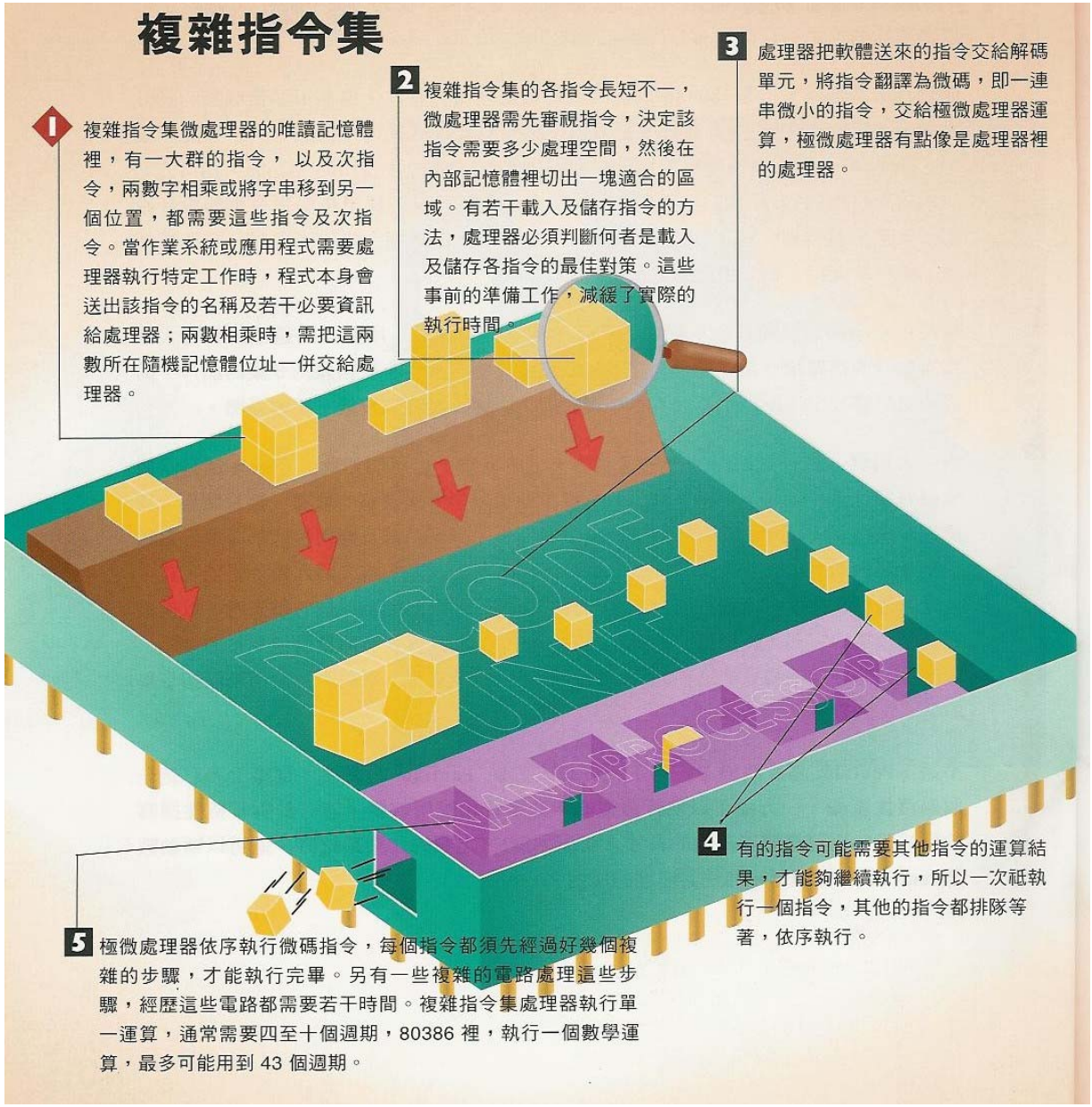
- **VAX : ~25%/year 1978 to 1986**
- **RISC + x86: ~52%/year 1986 to 2002**

UniProcessor Performance

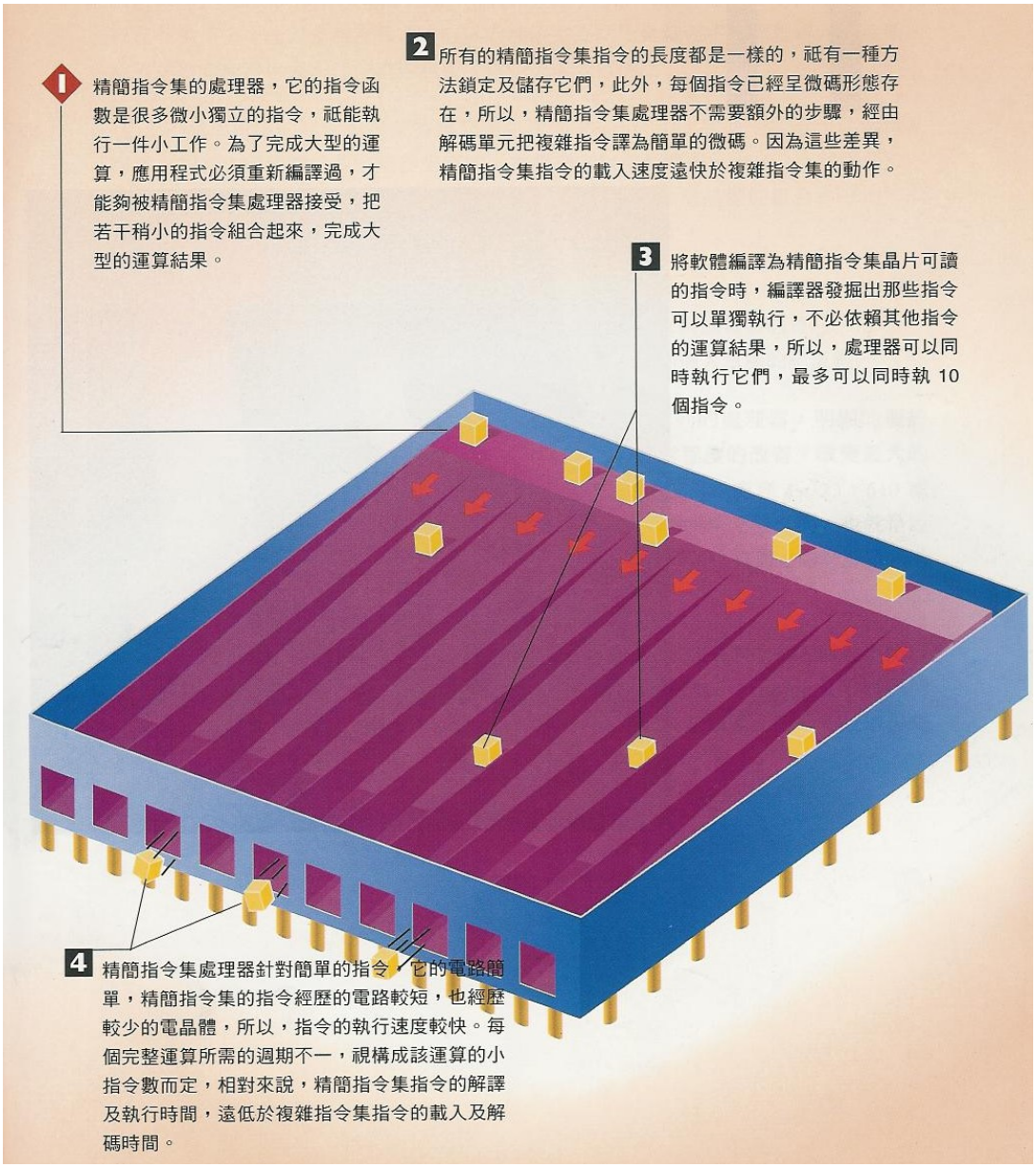
- Early 1970s, mainframes and minicomputers
  - 25%~30% growth per year in performance
- Late 1970, microprocessor
  - 35% growth per year in performance
- Early 1980s, Reduced Instruction Set Computer (RISC) architectures
  - 2 critical performance techniques
    - ILP (initially through pipelining and later through multiple instruction issue)
    - Cache
  - 50% growth per year in performance
- 1998~2000, relative performance
  - By technology: 1.35×per year
  - By technology + architecture, overall: 1.58 ×per year

    Note: 1.58≈1.35×(1+15%), the architecture improvement factor is 15%

CISC (Complex Instruction Set Computer)



RISC (Reduced Instruction Set Computer)



Preview

- Two key reasons to rapid improvement in computer performance since the mid-1980s
  - advances in the technology
  - innovation in computer design

Facts

- Since 2002, processor performance has dropped from about 50% to about 20% per year
  - High power dissipation
  - Little ILP left to exploit efficiently
  - Almost unchanged memory latency
- Faster uniprocessor or Multiple processors?

Outline

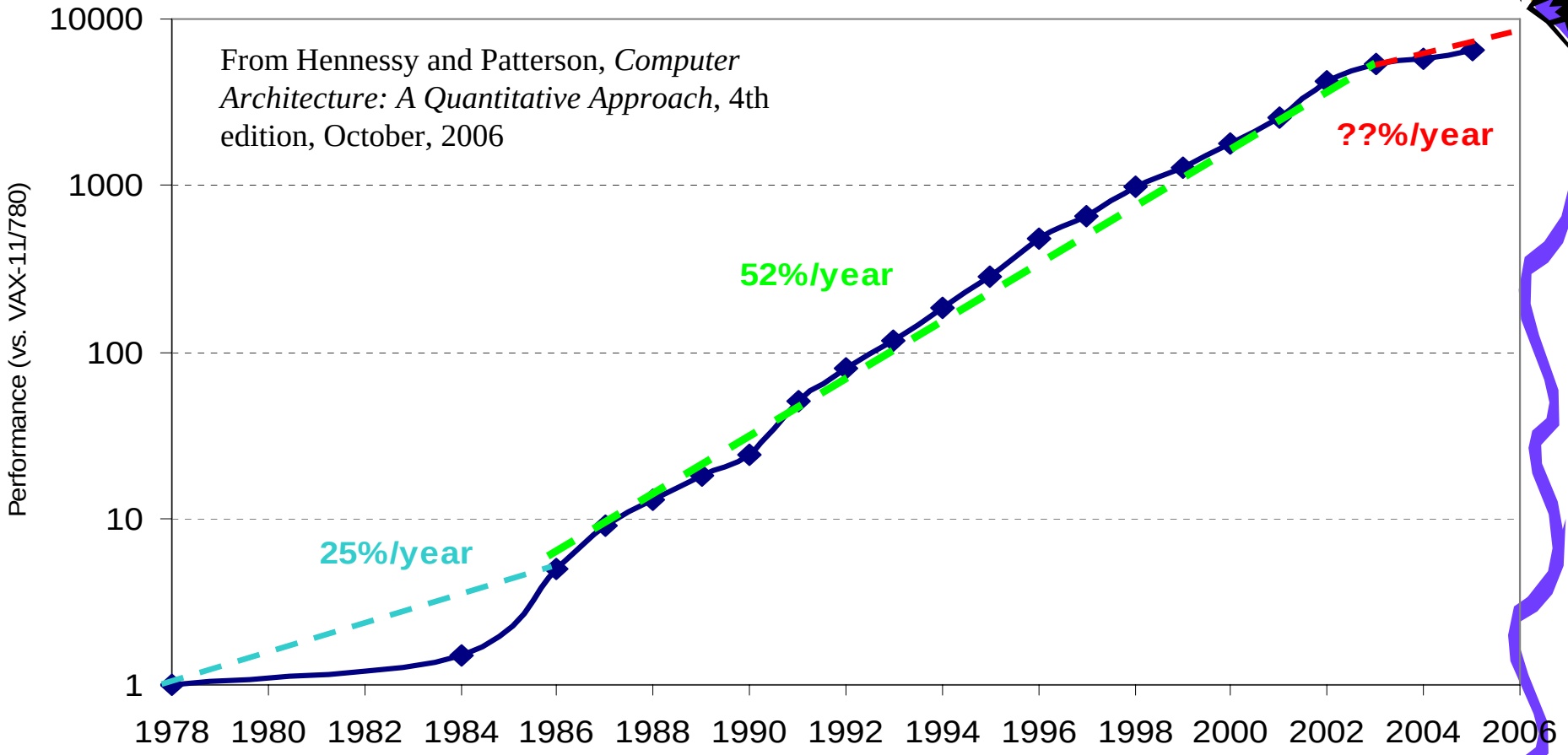
- Computer Science at a crossroads
- Computer Architecture vs. Instruction Set Arch.
- What Computer Architecture brings to table?

Crossroads: Conventional Wisdom in Comp. Arch

- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Transistors free (Can put more on chip than can afford to turn on)
- Old CW: Sufficiently increasing Instruction Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, ...)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)
- Old CW: Uniprocessor performance 2X / 1.5 yrs
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

  ⇒ Sea change in chip design: multiple "cores" (2X processors per chip / ~ 2 years)
    - More simpler processors are more power efficient

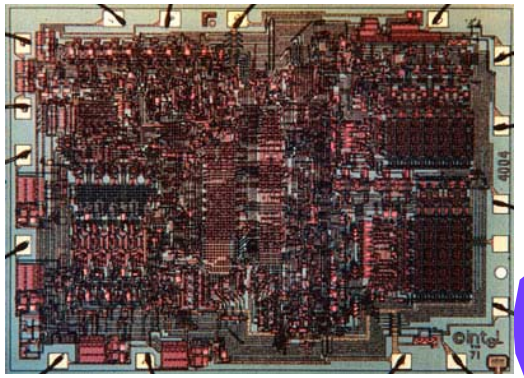
Crossroads: Uniprocessor Performance

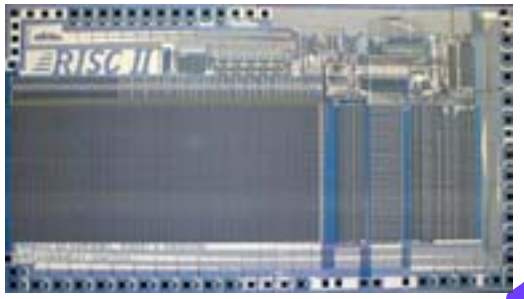


- **VAX : 25%/year 1978 to 1986**
- **RISC + x86: 52%/year 1986 to 2002**
- **RISC + x86: ??%/year 2002 to present**

Sea Change in Chip Design

- Intel 4004 (1971): 4-bit processor, 2312 transistors, 0.4 MHz, 10 micron PMOS, 11 mm^2 chip
- RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm^2 chip
- 125 mm^2 chip, 0.065 micron CMOS = 2312 RISC II+FPU+Icache+Dcache
  - RISC II shrinks to ~ 0.02 mm^2 at 65 nm
  - Caches via DRAM or 1 transistor SRAM?
  - Proximity Communication via capacitive coupling at > 1 TB/s ? (Ivan Sutherland @ Sun / Berkeley)





- **Processor is the new transistor?**

Problems with Sea Change

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, … not ready to supply Thread Level Parallelism or Data Level Parallelism for 1000 CPUs / chip,
- Architectures not ready for 1000 CPUs / chip
  - Unlike Instruction Level Parallelism, cannot be solved just by computer architects and compiler writers alone, but also cannot be solved *without* participation of computer architects

- This edition of textbook--Computer Architecture: A Quantitative Approach explores shift from Instruction Level Parallelism to Thread Level Parallelism / Data Level Parallelism

Outline

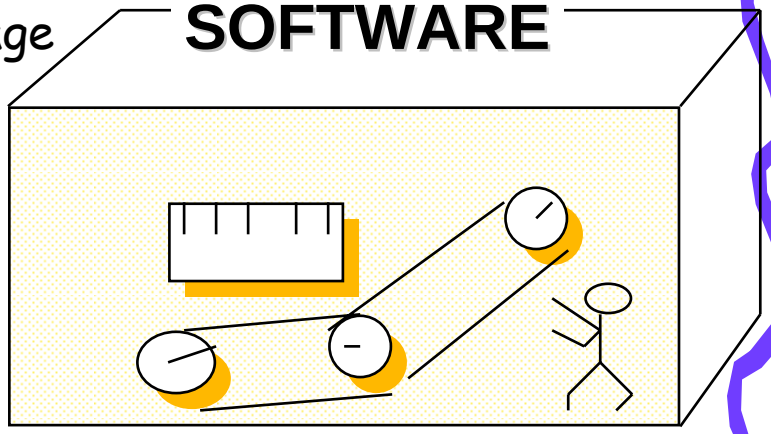
- Computer Science at a Crossroads
- Computer Architecture vs. Instruction Set Arch.
- What Computer Architecture brings to table

Instruction Set Architecture?

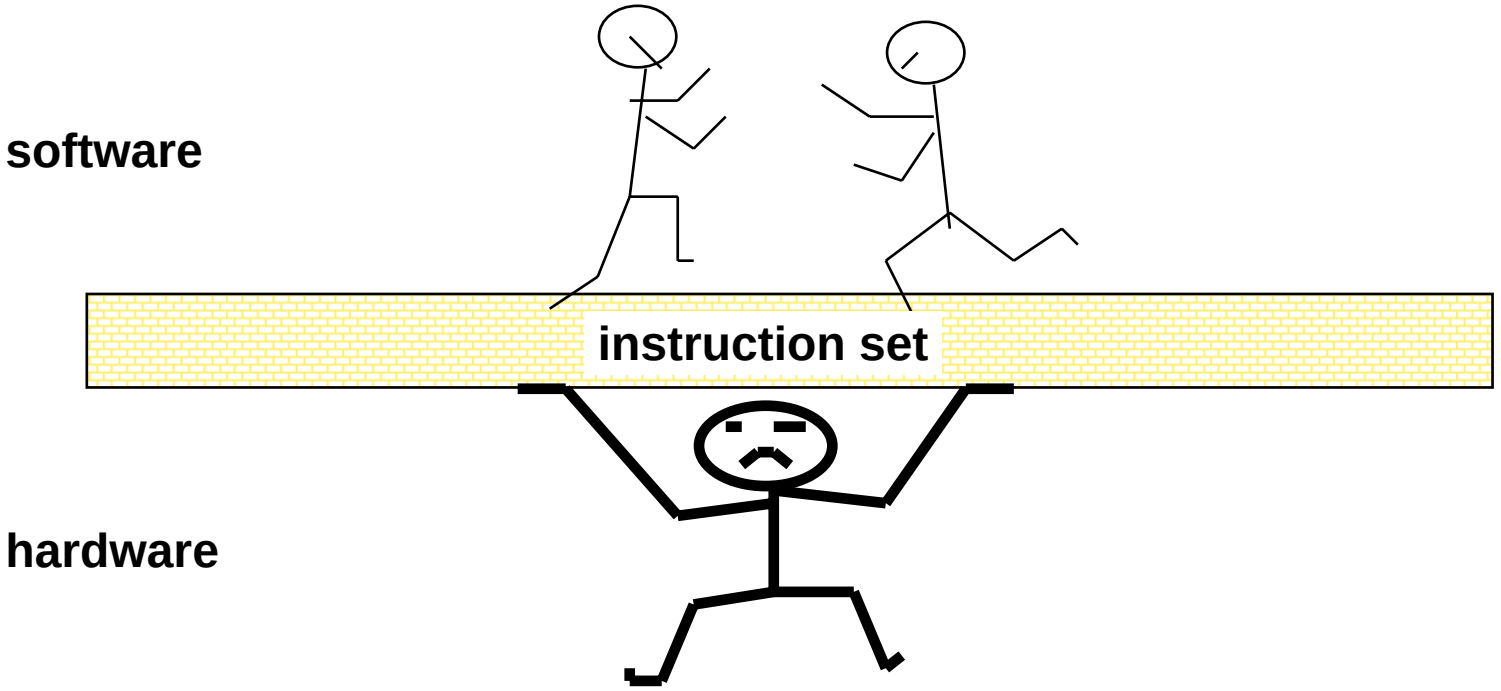
"... the attributes of a [computing] system as seen by the programmer, *i.e.* the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation."

– Amdahl, Blaauw, and Brooks, 1964

- Organization of Programmable Storage
- Data Types & Data Structures: Encodings & Representations
- Instruction Formats
- Instruction (or Operation Code) Set
- Modes of Addressing and Accessing Data Items and Instructions
- Exceptional Conditions



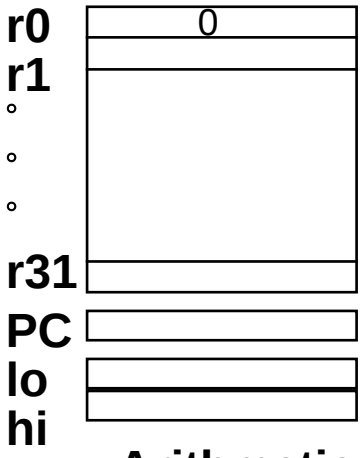
Instruction Set Architecture



ISA: Critical Interface

- Properties of a good abstraction
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

Example: MIPS



Programmable storage

- 2^32 x bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)
- HI, LO, PC

Data types ?

Format ?

Addressing Modes?

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

Control

J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

32-bit instructions on word boundary

Task of the Computer Designer

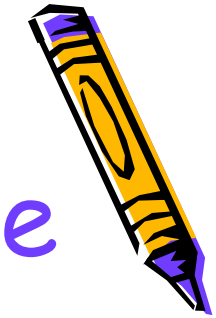
- Instruction set design
- Functional organization
- Logic design and implementation
  - To design a machine to optimize the tradeoff of the performance, while staying within cost and power constraints

- "Organization"–including the high-level aspects of computer design, such as memory system, bus structure, internal CPU
  - 2 processors with identical instruction set but very different organizations
    - NEC VR 4122 v.s. NEC VR 5432

      MIPS64 instruction set, but different pipeline and cache organization
- "Hardware" – The detailed logic design and the packaging technology
  - 2 processor with identical instruction set and nearly identical organizations, but they differ in the hardware implementation
    - Pentium II v.s. Celeron

      different clock rate and different memory system

ISA vs. Computer Architecture

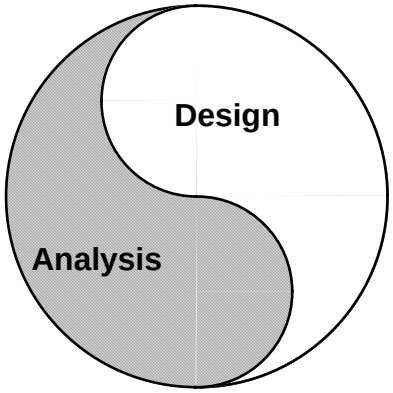


- Old definition of computer architecture = instruction set design
  - Other aspects of computer design called implementation
  - Insinuates implementation is uninteresting or less challenging
- Our view is computer architecture >> ISA
  - Architect's job much more than instruction set design; technical hurdles today *more* challenging than those in instruction set design
- Since instruction set design not where action is, some conclude computer architecture (using old definition) is not where action is
  - The differences among instruction sets are small and when there are distinct application areas

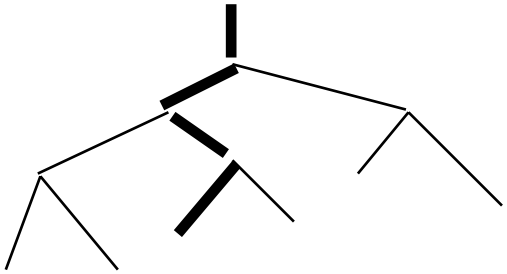
Comp. Arch. is an Integrated Approach

- What really matters is the functioning of the complete system
  - hardware, runtime system, compiler, operating system, and application
  - In networking, this is called the "End to End argument"
- Computer architecture is not just about transistors, individual instructions, or particular implementations
  - E.g., Original RISC projects replaced complex instructions with a compiler + simple instructions

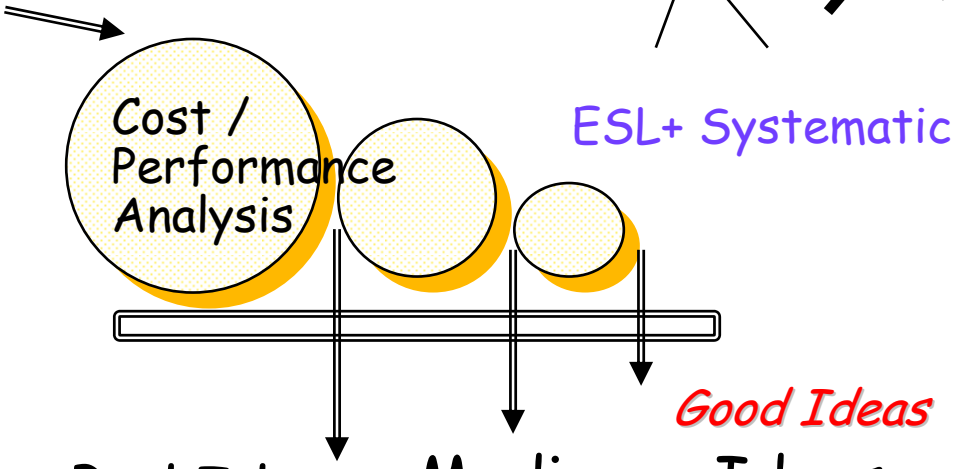
Computer Architecture is Design and Analysis



An iterative process:
- Searching the space of possible designs
- At all levels of computer systems



Creativity



ESL+ Systematic methodology

Outline

- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch.
- What Computer Architecture brings to table

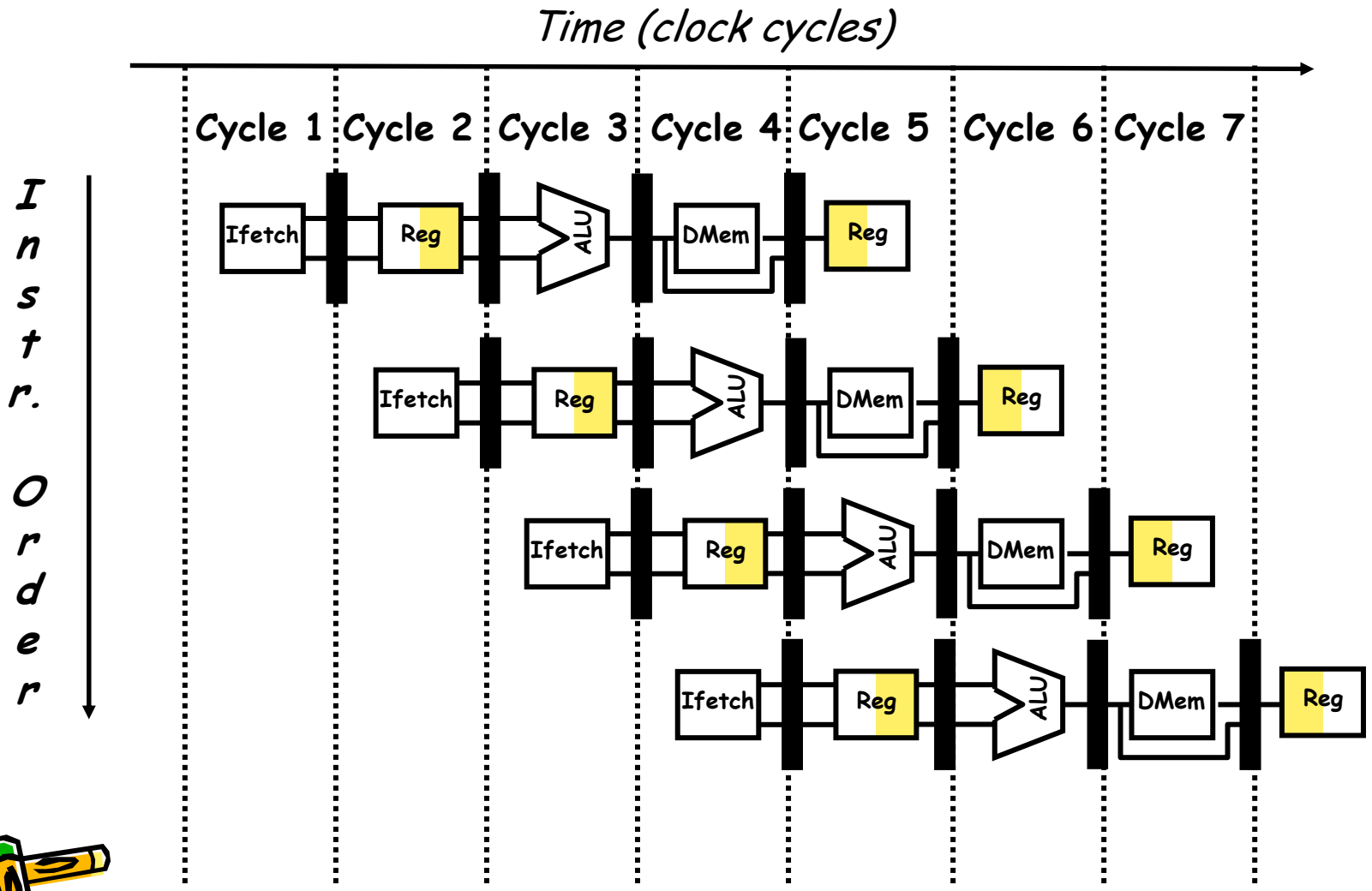
What Computer Architecture brings to Table

- Other fields often borrow ideas from architecture
- Quantitative Principles of Design
  1. Take Advantage of Parallelism
  2. Principle of Locality
  3. Focus on the Common Case
  4. Amdahl's Law
  5. The Processor Performance Equation
- Careful, quantitative comparisons
  - Define, quantity, and summarize relative performance
  - Define and quantity relative cost
  - Define and quantity dependability
  - Define and quantity power
- Culture of anticipating and exploiting advances in technology
- Culture of well-defined interfaces that are carefully implemented and thoroughly checked

1) Taking Advantage of Parallelism

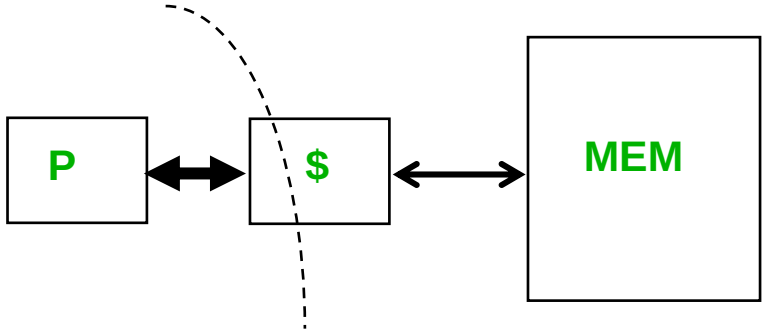
- Increasing throughput of server computer via multiple processors or multiple tasks/disks
- Detailed HW design
  - Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand
  - Multiple memory banks searched in parallel in set-associative caches
- Pipelining: overlap instruction execution to reduce the total time to complete an instruction sequence.
- ILP
- DLP, TLP

Pipelined Instruction Execution

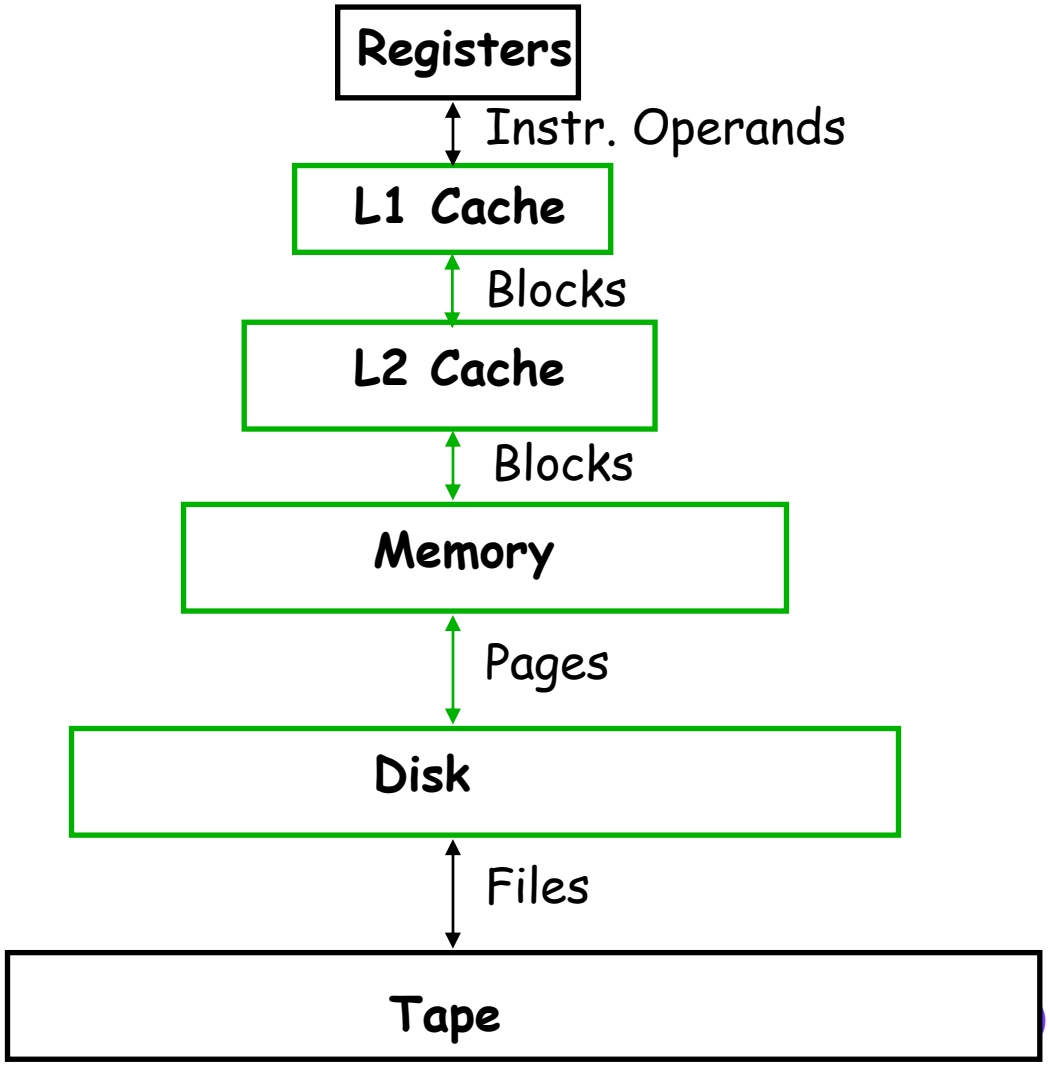


2) The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)
- Last 30 years, HW relied on locality for memory perf.



Memory Hierarchy



Upper Level

faster

Larger

Lower Level

3) Focus on the Common Case

- Common sense guides computer design
  - Since its engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

Amdahl's Law

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

$$\therefore \text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}}$$
$$= \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Example

Two design alternative for FP square root

1. Add FPSQR hardware
   - 20% of the execution time in benchmark
   - ➔ Speedup factor 10
2. Make all FP instructions faster
   - 50% of the execution time for all FP instructions
   - ➔ 1.6 times faster

Answer

$$\text{Speedup}_{\text{FPSQR}} = \frac{1}{(1-0.2)+\frac{0.2}{10}} = 1.22$$

$$\text{Speedup}_{\text{FP}} = \frac{1}{(1-0.5)+\frac{0.5}{1.6}} = 1.23$$

➔ Improving the performance of the FP operations overall is slightly better because of the higher frequency

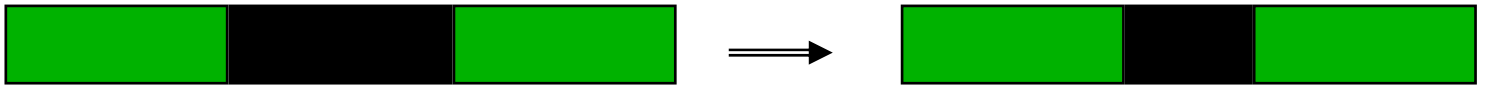
4) Amdahl's Law

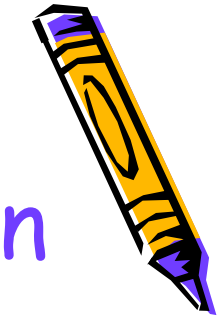
$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$





5) Processor Performance Equation

$$\text{CPU time} = \text{Instruction count} \times \text{Cycles per instruction} \times \text{Cycle time}$$

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

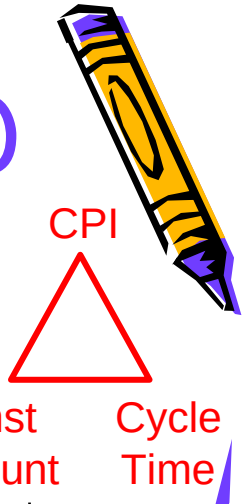
$$= \frac{\text{Seconds}}{\text{Program}}$$

It is difficult to change one parameter in complete isolation from others!!

- Instruction set architecture and compiler technology
- Organization and instruction set architecture
- Hardware technology and organization

Aspects of CPU Performance (CPU Law)

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$



	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

CPI and CPU Time

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

Throughput

CPI_i = average number of clock cycles for instruction i

IC_i = number of time the instruction i is executed in a program

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

$$\text{CPI} = \frac{\sum_{i=1}^{n} \text{CPI}_i \times \text{IC}_i}{\text{Instruction count}} = \sum_{i=1}^{n} \text{CPI}_i \times \frac{\text{IC}_i}{\text{Instruction count}}$$

Example

We have the following measurements:

- Freq. of FP operation (other than FPSQR) = 25%
- Average CPI of FP operation = 4
- Average CPI of other instructions = 1.33
- Freq. of FPSQR = 2%
- CPI of FPSQR = 20

Assume we have 2 design alternatives

1. CPI of FPSQR: 20 → 2 , 10 times improve
2. CPI of FP operations: 4 → 2.5, 1.6 times improve

Answer: (Only CPI changes, clock rate, instruction count remains identical)

$$\text{CPI}_{\text{original}} = \sum_{i=1}^{n} \text{CPI}_i \times (\text{IC}_i / \text{Instruction count}) = 4 \times 0.25 + 1.33 \times 0.75 = 2.0$$

$$\text{CPI}_{\text{new FPSQR}} = \text{CPI}_{\text{original}} - 2\% \, (\text{CPI}_{\text{old FPSQR}} - \text{CPI}_{\text{new FPSQR only}})$$

$$= 2.0 - 2\% \, (20 - 2) = 1.64$$

$$\text{CPI}_{\text{new FP}} = \sum_{i=1}^{n} \text{CPI}_i \times (\text{IC}_i / \text{Instruction count}) = 2.5 \times 0.25 + 1.33 \times 0.75 = 1.625$$

Better !!

And Some Concluding Remarks …

- Computer Architecture >> instruction sets
- Computer Architecture skill sets are different
  - 5 Quantitative principles of design
  - Quantitative approach to design
  - Solid interfaces that really work
  - Technology tracking and anticipation
- Computer Science at the crossroads from sequential to parallel computing
- Read Chapter 1, then Appendix A