DEPT. OF ELECTRONICS ENGINEERING & INST. OF ELECTRONICS NCTU

Computer Architecture
Lecture 1: Fundamentals of Quantitative Design and Analysis

Chih-Wei Liu 劉志尉

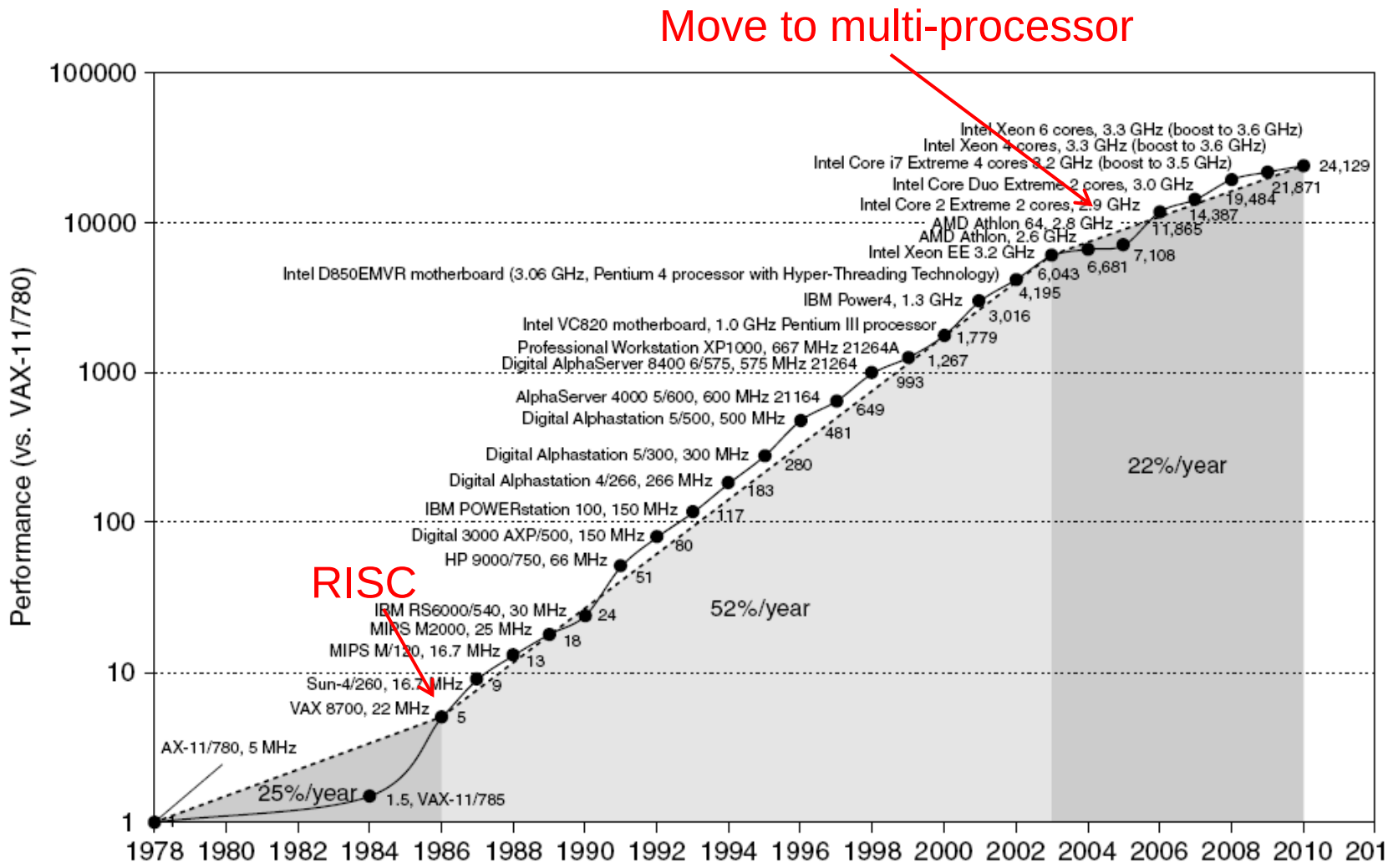
National Chiao Tung University

cwliu@twins.ee.nctu.edu.tw

Computer Technology

- Performance improvements:
  - Improvements in semiconductor technology
    - Feature size, clock speed
  - Improvements in computer architectures
    - Enabled by high-level language (HLL) compilers, UNIX
    - Lead to RISC architectures
  - Together have enabled:
    - Lightweight computers
    - Productivity-based managed/interpreted programming languages

Single Processor Performance



Current Trends in Architecture

- Cannot continue to leverage Instruction-Level parallelism (ILP)
  - Single processor performance improvement ended in 2003
- New models for performance:
  - Data-level parallelism (DLP)
  - Thread-level parallelism (TLP)
  - Request-level parallelism (RLP)
- These require explicit restructuring of the application

Classes of Computers

- Personal Mobile Device (PMD)
  - e.g. start phones, tablet computers
  - Emphasis on energy efficiency and real-time
- Desktop Computing
  - Emphasis on price-performance
- Servers
  - Emphasis on availability, scalability, throughput
- Clusters / Warehouse Scale Computers
  - Used for “Software as a Service (SaaS)”
  - Emphasis on availability and price-performance
  - Sub-class: Supercomputers, emphasis: floating-point performance and fast internal networks
- Embedded Computers
  - Emphasis: price

Parallelism

- Classes of parallelism in applications:
  - Data-Level Parallelism (DLP)
  - Task-Level Parallelism (TLP)

- Classes of architectural parallelism:
  - Instruction-Level Parallelism (ILP)
  - Vector architectures/Graphic Processor Units (GPUs)
  - Thread-Level Parallelism
  - Request-Level Parallelism

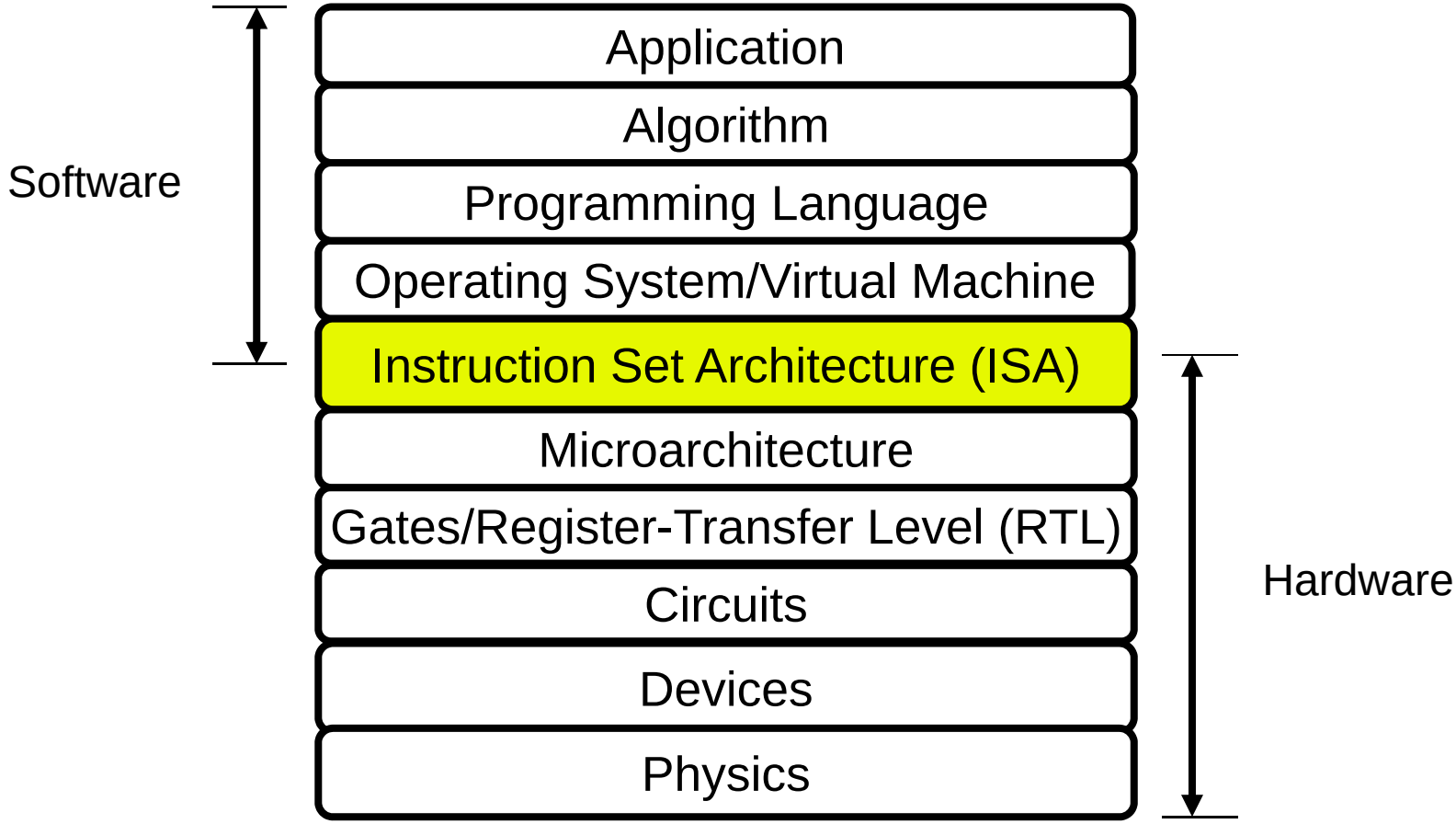
Flynn's Taxonomy

- Single instruction stream, single data stream (SISD)
- Single instruction stream, multiple data streams (SIMD)
  - Vector architectures
  - Multimedia extensions
  - Graphics processor units
- Multiple instruction streams, single data stream (MISD)
  - No commercial implementation
- Multiple instruction streams, multiple data streams (MIMD)
  - Tightly-coupled MIMD
  - Loosely-coupled MIMD

Defining Computer Architecture

- "Old" view of computer architecture:
  - Instruction Set Architecture (ISA) design
  - i.e. decisions regarding:
    - registers, memory addressing, addressing modes, instruction operands, available operations, control flow instructions, instruction encoding
- "Real" computer architecture:
  - Specific requirements of the target machine
  - Design to maximize performance within constraints: cost, power, and availability
  - Includes ISA, microarchitecture, hardware

Computer System vs. ISA

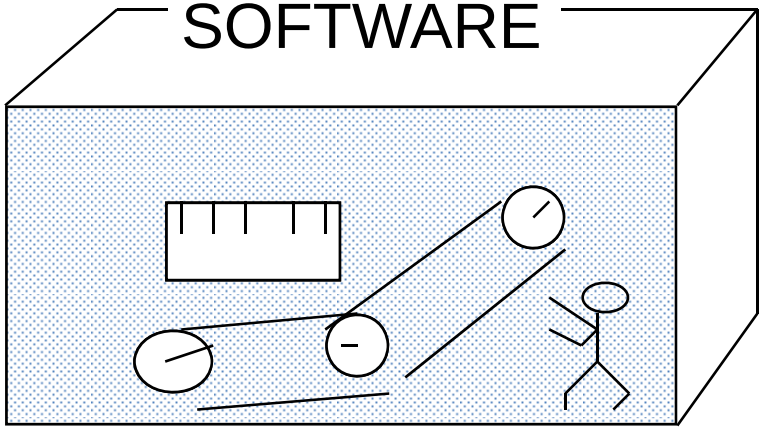


Instruction Set Architecture, ISA

"... 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."

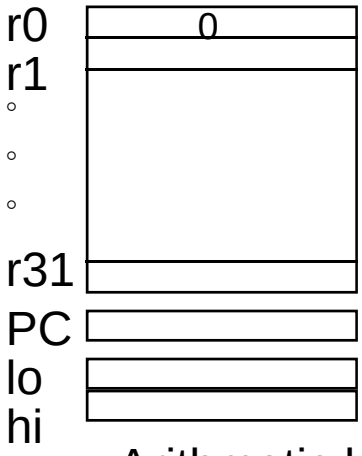
– Amdahl, Blaauw, and Brooks, 1964

- 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

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

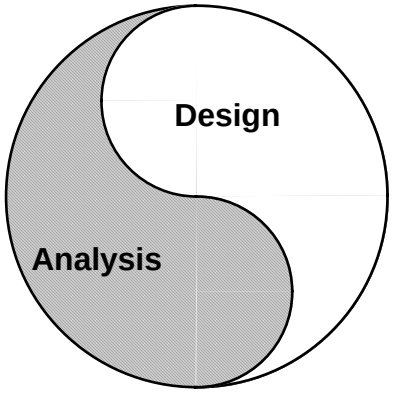
LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

Control

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

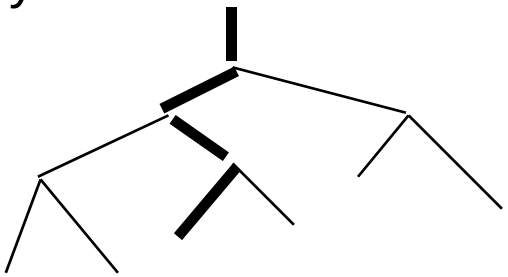
32-bit instructions on word boundary

Comp. Arch. is an Integrated Approach

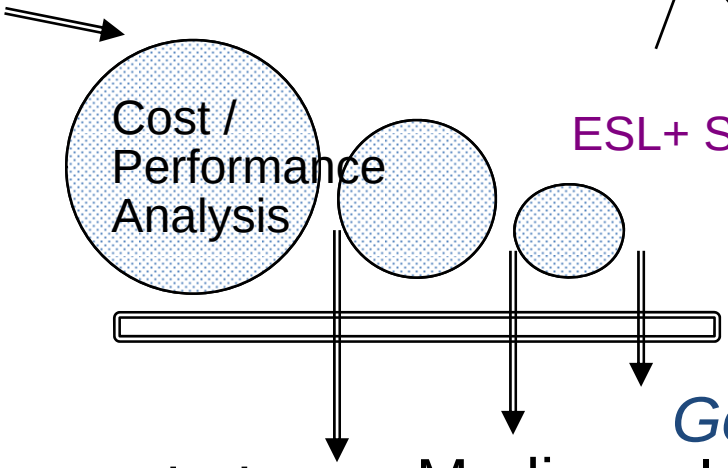


An iterative process:

- Searching the space of possible designs
- At all levels of computer systems



Creativity



ESL+ Systematic methodology

Good Ideas

Bad Ideas

Mediocre Ideas

Trends in Technology

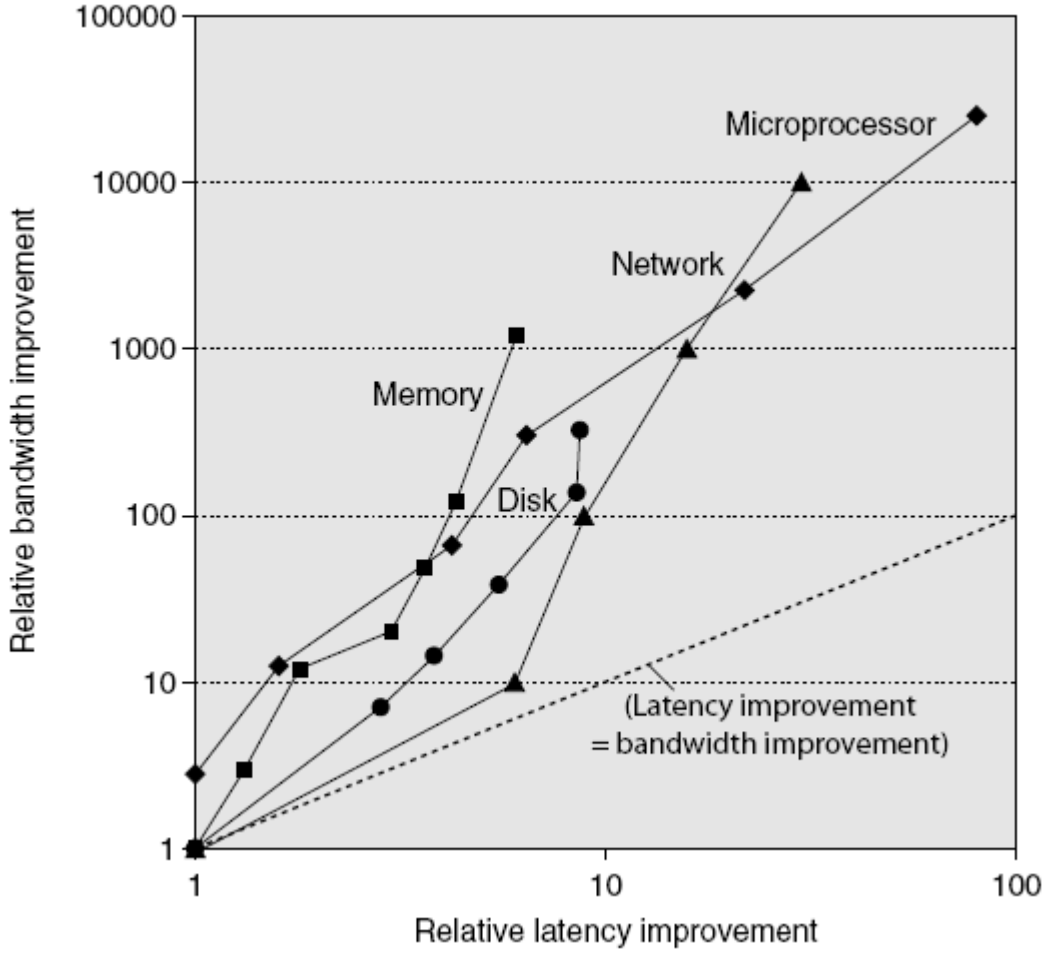
- Integrated circuit technology
  - Transistor density: 35%/year
  - Die size: 10-20%/year
  - Integration overall: 40-55%/year
- DRAM capacity: 25-40%/year (slowing)
- Flash capacity: 50-60%/year
  - 15-20X cheaper/bit than DRAM
- Magnetic disk technology: 40%/year
  - 15-25X cheaper/bit then Flash
  - 300-500X cheaper/bit than DRAM

Bandwidth and Latency

- Bandwidth or throughput
  - Total work done in a given time
  - 10,000-25,000X improvement for processors
  - 300-1200X improvement for memory and disks

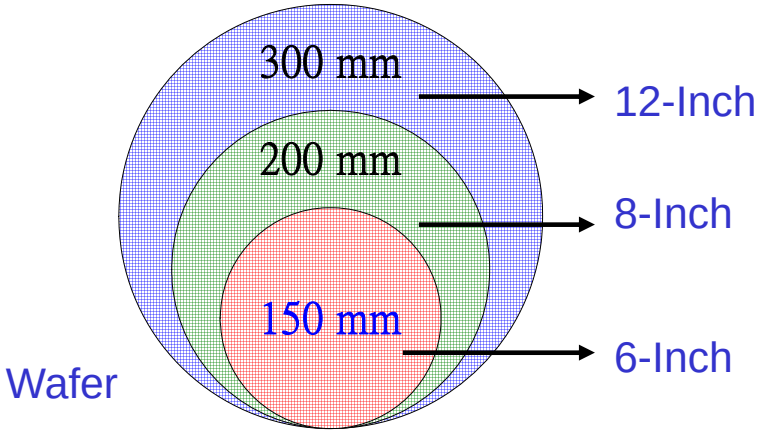
- Latency or response time
  - Time between start and completion of an event
  - 30-80X improvement for processors
  - 6-8X improvement for memory and disks

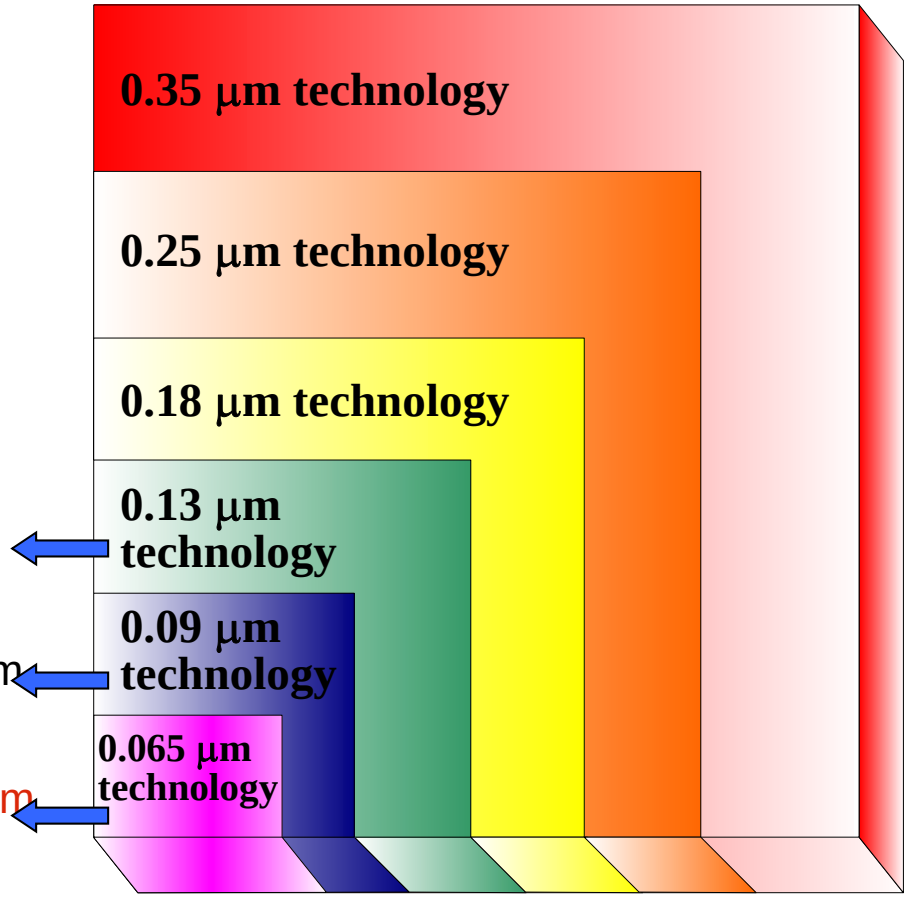
Bandwidth and Latency



Log-log plot of bandwidth and latency milestones

Feature Technology and Size





When compared to the 0.18-micron process, the new 0.13-micron process results in less than 60 percent the die size and nearly 70 percent improvement in performance

The 90-nm process will be manufactured on 300mm wafers

NEC devises low-k film for second-generation 65-nm process

Transistors and Wires

- Feature size
  - Minimum size of transistor or wire in x or y dimension
  - 10 microns in 1971 to .032 microns in 2011
  - Transistor performance scales linearly
    - Wire delay does not improve with feature size!
  - Integration density scales quadratically

Why Latency Lags Bandwidth?

- Moore's law helps BW more than latency
  - Faster transistors, more transistors, and more pin count help bandwidth
  - Smaller transistors communicate over (relatively) longer wires
- Latency helps BW, but not vice versa
  - Lower DRAM latency, i.e. more accesses per second, implies higher bandwidth
  - Higher density of disk helps BW and capacity, but not disk latency
- BW hurts latency
  - Queues or buffers help bandwidth, but hurt latency
  - Memory bank help bandwidth with widen the memory module, but not latency due to higher fan-out on address lines
- OS overhead hurts latency more than BW
  - Packet header overhead: bigger part of short message

Summary of Technology Trends

- For disk, LAN, memory, and microprocessor, bandwidth improves by square of latency improvement
  - In the time that bandwidth doubles, latency improves by no more than 1.2X to 1.4X
- Lag probably even larger in real systems, as bandwidth gains multiplied by replicated components
  - Multiple processors in a cluster or even in a chip
  - Multiple disks in a disk array
  - Multiple memory modules in a large memory
  - Simultaneous communication in switched LAN
- HW and SW developers should innovate assuming Latency Lags Bandwidth
  - If everything improves at the same rate, then nothing really changes
  - When rates vary, require real innovation

Define and Quantity Power (and Energy)

- Problem: Get power in, get power out
- Thermal Design Power (TDP)
  - Characterizes sustained power consumption
  - Used as target for power supply and cooling system
  - Lower than peak power, higher than average power consumption
- Clock rate can be reduced dynamically to limit power consumption
- Energy per task is often a better measurement

Power and Energy (1/2)

- For CMOS chips, traditional dominant power consumption has been in switching transistors, called *dynamic power*

$$P_{dynamic} = \frac{1}{2} \times C_{load} \times V^2 \times F$$

- For mobile devices, energy, instead of power, is the proper metric
  - Transistor switch from 0 -> 1 or 1 -> 0

$$E = C \times V^2$$

- For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy
- Dropping voltage helps both, so went from 5V to 1V
- As moved from one process to the next, the increase in the number of transistors switching, the frequency, dominates the decrease in load capacitance and voltage

  ➔ an overall growth in power consumption and energy
- To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)

Power and Energy (2/2)

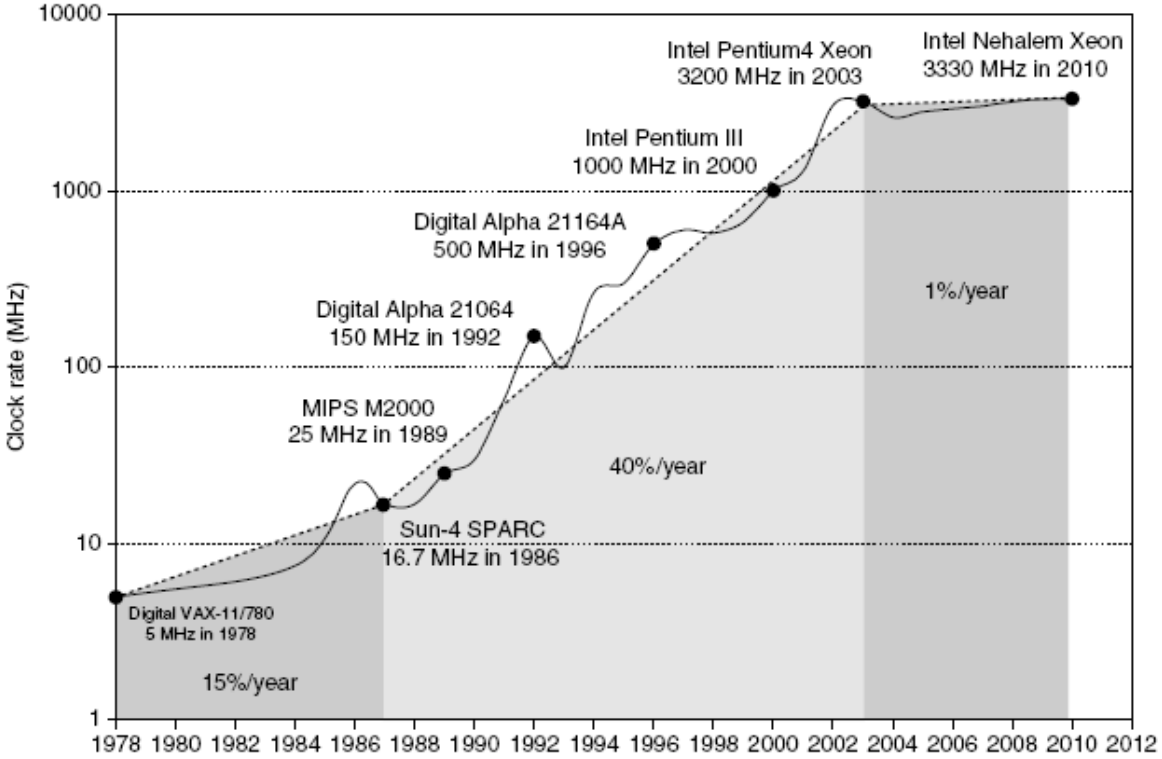
- Because leakage current flows even when a transistor is off, now *static power* is important too

$$P_{static} = I_{static} \times V$$

- Increasing the number of transistors increases power even if they are turned off
- Leakage current increases in processors with smaller transistor sizes
- In 2006, goal for leakage is 25% of total power consumption; high performance designs at 40%
- Very low power systems even gate voltage to inactive modules to control loss due to leakage

Power

- Intel 80386 consumed ~ 2 W
- 3.3 GHz Intel Core i7 consumes 130 W
- Heat must be dissipated from 1.5 x 1.5 cm chip
- This is the limit of what can be cooled by air



Trends in Cost

- Price: what you sell a finished good for
- Cost: amount spent to produce it, including overhead
- The impact of time, volume, and commodification
  - Learning curve: manufacturing costs decrease over time (max. measured by yield)
  - Volume decreases the cost
  - Commodities: sell on the grocery stores, multiple suppliers.

Cost of an IC

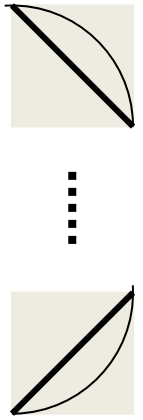
- A wafer is tested and chopped into dies

$$C_{\text{die}} = \frac{C_{\text{wafer}}}{\text{Die per wafer} \times \text{Die yield}}$$

$$\text{Die per wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2} \times \sqrt{\text{Die area}}}$$

- The die is still tested and packaged into IC

$$C_{\text{IC}} = \frac{C_{\text{die}} + C_{\text{testing die}} + C_{\text{packaging and final test}}}{\text{Final test yield}}$$



IC Yield

- Bose-Einstein formula:

$$\text{Die yield} = \text{Wafer yield} \times 1/\left(1 + \text{Defects per unit area} \times \text{Die area}\right)^{N}$$

  - Wafer yield is almost 100%
  - Defects per unit area = 0.016-0.057 defects per square cm (2010)
  - N = process-complexity factor = 11.5-15.5 (40 nm, 2010)

- Example: Find the die yield for dies that are 1.5cm on a side. Assuming a defect density of 0.031/cm^2 and *N* is 13.5.

$$\text{Die yield} = 1/\left(1 + 0.031 \times 2.25\right)^{13.5} = 0.4$$

Dependability?

- Old CW: ICs are reliable components
- New CW: On the transistor feature size down to 65nm or smaller, both transient faults and permanent faults will become more commonplace.
- Define and quantity dependability
  - 2 states of finite state machine: Failure and Restoration
  - Service accomplishment vs. Service interruption
- Module/system reliability
  - Mean time to failure (MTTF)
  - Mean time to repair (MTTR)
  - Mean time between failures (MTBF) = MTTF + MTTR
  - Availability = MTTF / MTBF

Define and Quantity Dependability

- Module reliability = measure of continuous service accomplishment (or time to failure).
  - 2 metrics
    1. Mean Time To Failure (MTTF) measures Reliability
    2. Failures In Time (FIT) = 1/MTTF, the rate of failures
       - Traditionally reported as failures per billion hours of operation
- Mean Time To Repair (MTTR) measures Service Interruption
  - Mean Time Between Failures (MTBF) = MTTF+MTTR
- Module availability = measure of the service accomplishment with respect to the alternation between the 2 states
- Module availability = MTTF / (MTTF + MTTR)

Measuring Performance

- Typical performance metrics:
  - Response time
  - Throughput
- Speedup of X relative to Y
  - Execution $time_Y$ / Execution $time_X$
- Execution time
  - Wall clock time: includes all system overheads
  - CPU time: only computation time
- Benchmarks
  - Kernels (e.g. matrix multiply)
  - Toy programs (e.g. sorting)
  - Synthetic benchmarks (e.g. Dhrystone)
  - Benchmark suites (e.g. SPEC06fp, TPC-C)

Performance Measurement

- Two different machines X and Y.

  X is *n* times faster than Y

$$\frac{\text{Execution time}_{\text{Y}}}{\text{Execution time}_{\text{X}}} = n$$

  Since execution time is the reciprocal of performance

$$\frac{\text{Execution time}_{\text{Y}}}{\text{Execution time}_{\text{X}}} = n = \frac{\text{Performance}_{\text{X}}}{\text{Performance}_{\text{Y}}}$$

- Says *n* -1 = *m*/100

  This concludes that X is m% faster than Y

Performance Equation

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

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

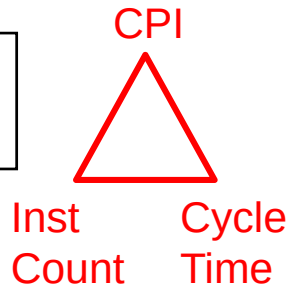
$$= \frac{\text{Seconds}}{\text{Program}}$$

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

Different instruction types having different CPIs

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

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 $\rightarrow$ 2 , 10 times improve
2. CPI of FP operations: 4 $\rightarrow$ 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 !!

Reporting Performance Results

- Which benchmark(s)?
- Performance Results should be reproducibility
  - Describe exactly the software system being measured and whether any special modifications have been made.
- Arithmetic average of execution time of all programs?
- Could add a weight per program?
  - How pick weight? Different company wants different weights for their product
- Normalized execution time with respect to a reference computer
  - Time on reference computer / time on computer being rated
  - SPECRatio

Compare/Summarize Performance

- WLOG, 2 different ways
  1. Arithmetic mean

$$\frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$$

     - Time_i is the execution time for the ith program in the workload
     - **Weighted arithmetic mean**

$$\sum_{i=1}^{n}\text{Weight}_i \times \text{Time}_i$$

     - Weight_i factors add up to 1
  2. Geometric mean

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

     - To normalize to a reference machine (e.g. SPEC)
     - Execution time ratio_i is the execution time normalized to the reference machine, for the ith program

Example

	Computers			Weightings		
	A	B	C	W(1)	W(2)	W(3)
Program P1	1.00	10.00	20.00	0.50	0.909	0.999
Program P2	1000.00	100.00	20.00	0.50	0.091	0.001

Arithmetic mean: W(1)	500.50	55.00	20.00
Arithmetic mean: W(2)	91.91	18.19	20.00
Arithmetic mean: W(3)	2.00	10.09	20.00

	Normalized to A			Normalized to B			Normalized to C		
	A	B	C	A	B	C	A	B	C
Program P1	1.00	10.00	20.00	0.10	1.00	2.00	0.05	0.50	1.00
Program P2	1.00	0.10	0.02	10.00	1.00	0.20	50.00	5.00	1.00
Arithmetic mean	1.00	5.05	10.01	5.05	1.00	1.10	25.03	2.75	1.00
Geometric mean	1.00	1.00	0.63	1.00	1.00	0.63	1.58	1.58	1.00
Total time	1.00	0.11	0.04	9.10	1.00	0.36	25.03	2.75	1.00

1. The arithmetic mean performance varies from ref. to ref.
2. The geometric mean performance is consistent

Ratio of SPECRatio

$$\text{e.g. } 1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\dfrac{ExecutionTime_{reference}}{ExecutionTime_A}}{\dfrac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

$$= \frac{ExecutionTime_B}{ExecutionTime_A} = \frac{Performance_A}{Performance_B}$$

- SPECRatio is just a ratio rather than an absolute execution time
- Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant

Geometric mean of the ratios is the ratio of geometric means

- Choice of reference computer is irrelevant.

$$\frac{Geometric\ Mean_A}{Geometric\ Mean_B} = \frac{\sqrt[n]{\prod_{i=1}^{n} SPECRatioA_i}}{\sqrt[n]{\prod_{i=1}^{n} SPECRatioB_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{SPECRatioA_i}{SPECRatioB_i}}$$

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{ExecutionTimeB_i}{ExecutionTimeA_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{PerformanceA_i}{PerformanceB_i}}$$

- Geometric mean does not predict execution time

Principles of Computer Design

1. Take Advantage of Parallelism
   - e.g. multiple processors, disks, memory banks, pipelining, multiple functional units

2. Principle of Locality
   - Reuse of data and instructions

3. Focus on the Common Case
   - Amdahl's Law

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

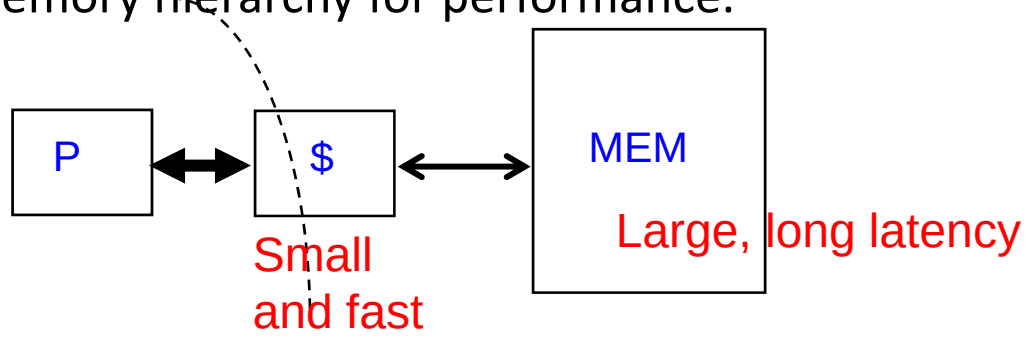
$$\text{Speedup}_{\text{overall}} = \frac{\text{Execution time}_{\text{old}}}{\text{Execution time}_{\text{new}}} = \frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

1) Taking Advantage of Parallelism

- Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand
- Set-associative caches searches in parallel by using multiple memory banks searched
- Pipelining technique overlaps instruction execution to reduce the total time to complete an instruction sequence.
- ILP, DLP, TLP, …
- Server computer increase throughput of via multiple processors or multiple tasks/disks

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)
- Cache in memory hierarchy for performance.



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

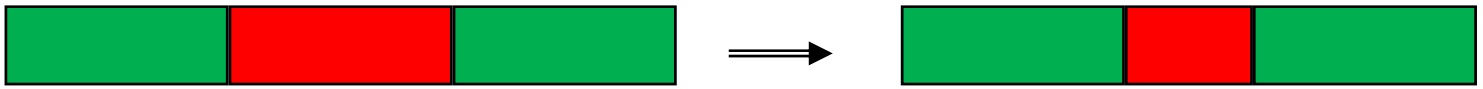
Amdahl's Law

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

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

Best you could ever hope to do:

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



old new

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

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

Reading Assignment

- Appendix L.2 The early development of Computers
- One-page report

- The Appendix L is available online at http://booksite.mkp.com/9780123838728/historical.php