CPE 631: Introduction

Electrical and Computer Engineering
University of Alabama in Huntsville
Aleksandar Milenkovic,
milenka@ece.uah.edu
http://www.ece.uah.edu/~milenka

Lecture Outline

- Evolution of Computer Technology
- Computing Classes
- Task of Computer Designer
- Technology Trends
- Costs and Trends in Cost
- Things to Remember

Introduction

CHANGE! It is exciting. It has never been more exciting! It impacts every aspect of human life.

Eniac, 1946

(first stored-program computer)
Occupied 50x30 feet room,
weighted 30 tonnes,
contained 18000 electronic valves,
consumed 25KW of electrical power;
capable to perform 100K calc. per second

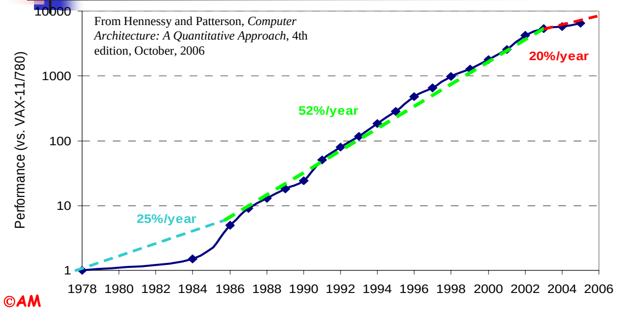
PlayStation Portable (PSP)

Approx. 170 mm (L) x 74 mm (W) x 23 mm (D)
Weight: Approx. 260 g (including battery)
CPU: PSP CPU (clock frequency 1~333MHz)
Main Memory: 32MB
Embedded DRAM: 4MB
Profile: PSP Game, UMD Audio, UMD Video

A short history of computing

- Continuous growth in performance due to ***advances in technology*** and ***innovations in computer design***
- First 25 years (1945 – 1970)
  - 25% yearly growth in performance
  - Both forces contributed to performance improvement
  - Mainframes and minicomputers dominated the industry
- Late 70s, emergence of the microprocessor
  - 35% yearly growth in performance thanks to integrated circuit technology
  - Changes in computer marketplace: elimination of assembly language programming, emergence of Unix ➔ easier to develop new architectures
- Mid 80s, emergence of RISCs (Reduced Instruction Set Computers)
  - 52% yearly growth in performance
  - Performance improvements through instruction level parallelism (pipelining, multiple instruction issue), caches
- Since ‘02, end of 16 years of renaissance
  - 20% yearly growth in performance
  - Limited by 3 hurdles: maximum power dissipation, instruction-level parallelism, and so called “memory wall”
  - Switch from ILP to TLP and DLP (Thread-, Data-level Parallelism)

Growth in processor performance



From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 4th edition, October, 2006

- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: 20%/year 2002 to present

Effect of this Dramatic Growth

- Significant enhancement of the capability available to computer user
  - Example: a today's $500 PC has more performance, more main memory, and more disk storage than a $1 million computer in 1985
- Microprocessor-based computers dominate
  - Workstations and PCs have emerged as major products
  - Minicomputers - replaced by servers
  - Mainframes - replaced by multiprocessors
  - Supercomputers - replaced by large arrays of microprocessors

Changing Face of Computing

- In the 1960s mainframes roamed the planet
  - Very expensive, operators oversaw operations
  - Applications: business data processing, large scale scientific computing
- In the 1970s, minicomputers emerged
  - Less expensive, time sharing
- In the 1990s, Internet and WWW, handheld devices (PDA), high-performance consumer electronics for video games and set-top boxes have emerged
- Dramatic changes have led to 3 different computing markets
  - Desktop computing, Servers, Embedded Computers

Computing Classes: A Summary

Feature	Desktop	Server	Embedded
Price of the system	$500-$5K	$5K-$5M	$10-$100K (including network routers at high end)
Price of the processor	$50-$500	$200-$10K	$0.01 - $100
Sold per year (estimates for 2000)	150M	4M	300M (only 32-bit and 64-bit)
Critical system design issues	Price-performance, graphics performance	Throughput, availability, scalability	Price, power consumption, application-specific performance

Desktop Computers

- Largest market in dollar terms
- Spans low-end (<$500) to high-end (≈$5K) systems
- Optimize price-performance
  - Performance measured in the number of calculations and graphic operations
  - Price is what matters to customers
- Arena where the newest, highest-performance and cost-reduced microprocessors appear
- Reasonably well characterized in terms of applications and benchmarking
- What will a PC of 2011 do?
- What will a PC of 2016 do?

Servers

- Provide more reliable file and computing services (Web servers)
- Key requirements
  - Availability – effectively provide service 24/7/365 (Yahoo!, Google, eBay)
  - Reliability – never fails
  - Scalability – server systems grow over time, so the ability to scale up the computing capacity is crucial
  - Performance – transactions per minute
- Related category: clusters / supercomputers

Embedded Computers

- Fastest growing portion of the market
- Computers as parts of other devices where their presence is not obviously visible
  - E.g., home appliances, printers, smart cards, cell phones, palmtops, set-top boxes, gaming consoles, network routers
- Wide range of processing power and cost
  - ≈$0.1 (8-bit, 16-bit processors), $10 (32-bit capable to execute 50M instructions per second), ≈$100-$200 (high-end video gaming consoles and network switches)
- Requirements
  - Real-time performance requirement (e.g., time to process a video frame is limited)
  - Minimize memory requirements, power
- SOCs (System-on-a-chip) combine processor cores and application-specific circuitry, DSP processors, network processors, ...

Task of Computer Designer

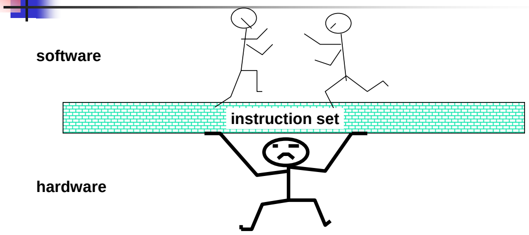
- **“Determine what attributes are important for a new machine; then design a machine to maximize performance while staying within cost, power, and availability constraints.”**
- Aspects of this task
  - Instruction set design
  - Functional organization
  - Logic design and implementation (IC design, packaging, power, cooling...)

What is Computer Architecture?

Computer Architecture covers all three aspects of computer design

- Instruction Set Architecture
  - the computer visible to the assembler language programmer or compiler writer (registers, data types, instruction set, instruction formats, addressing modes)
- Organization
  - high level aspects of computer's design such as the memory system, the bus structure, and the internal CPU (datapath + control) design
- Hardware
  - detailed logic design, interconnection and packing technology, external connections

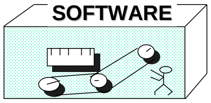
Instruction Set Architecture: Critical Interface



- 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

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 (GPRs, SPRs)
- 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: MIPS64

- Registers
  - 32 64-bit general-purpose (integer) registers (R0-R31)
  - 32 64-bit floating-point registers (F0-F31)
- Data types
  - 8-bit bytes, 16-bit half-words, 32-bit words, 64-bit double words for integer data
  - 32-bit single- or 64-bit double-precision numbers
- Addressing Modes for MIPS Data Transfers
  - Load-store architecture: Immediate, Displacement
  - Memory is byte addressable with a 64-bit address
  - Mode bit to select Big Endian or Little Endian

Example: MIPS64

- MIPS Instruction Formats (R-type, I-type, J-type)

Register-Register

31 26	25 21	20 16	15 11	10 6	5 0
Op	Rs1	Rs2	Rd		Opx

Register-Immediate

31 26	25 21	20 16	15 0
Op	Rs1	Rd	immediate

Branch

31 26	25 21	20 16	15 0
Op	Rs1	Rs2/Opx	immediate

Jump / Call

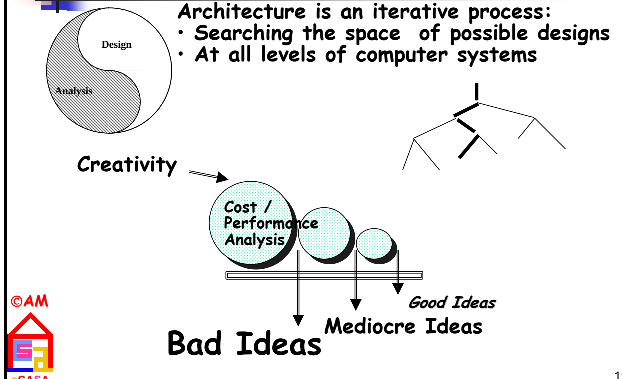
31 26	25 0
Op	target

Example: MIPS64

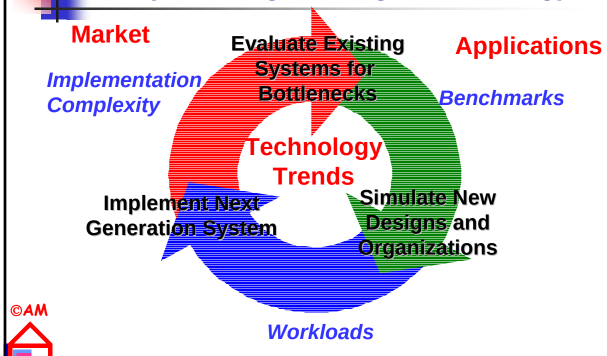
- MIPS Operations
  (See Appendix B, Figure B.26)
  - Data Transfers (LB, LBU, SB, LH, LHU, SH, LW, LWU, SW, LD, SD, L.S, L.D, S.S, S.D, MFCO, MTCO, MOV.S, MOV.D, MFC1, MTC1)
  - Arithmetic/Logical (DADD, DADDI, DADDU, DADDIU, DSUB, DSUBU, DMUL, DMULU, DDIV, DDIVU, MADD, AND, ANDI, OR, ORI, XOR, XORI, LUI, DSLL, DSRL, DSRA, DSLLV, DSRLV, DSRAV, SLT, SLTI, SLTU, SLTIU)
  - Control (BEQZ, BNEZ, BEQ, BNE, BC1T, BC1F, MOVN, MOVZ, J, JR, JAL, JALR, TRAP, ERET)
  - Floating Point (ADD.D, ADD.S, ADD.PS, SUB.D, SUB.S, SUB.PS, MUL.D, MUL.S, MUL.PS, MADD.D, MADD.S, MADD.PS, DIV.D, DIV.S, DIV.PS, CVT._._, C._.D, C._.S

Computer Architecture is Design and Analysis

Architecture is an iterative process:
- **Searching the space of possible designs**
- **At all levels of computer systems**



Computer Engineering Methodology


Market
Applications
Implementation Complexity
Benchmarks
Evaluate Existing Systems for Bottlenecks
Technology Trends
Implement Next Generation System
Simulate New Designs and Organizations
Workloads

Technology Trends

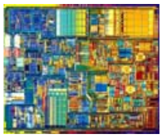
- Integrated circuit technology – 55% /year
  - Transistor density – 35% per year
  - Die size – 10-20% per year
- Semiconductor DRAM
  - Density – 40-60% per year (4x in 3-4 years)
  - Cycle time – 33% in 10 years
  - Bandwidth – 66% in 10 years
- Magnetic disk technology
  - Density – 100% per year
  - Access time – 33% in 10 years
- Network technology (depends on switches and transmission technology)
  - 10Mb-100Mb (10years), 100Mb-1Gb (5 years)
  - Bandwidth – doubles every year (for USA)

©AM

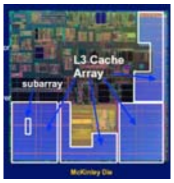
CASA

Processor Transistor Count

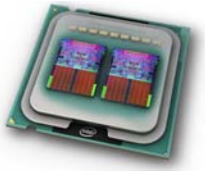
Intel 4004, 2300tr (1971)



Intel P4 – 55M tr (2001)



Intel McKinley – 221M tr. (2001)



Intel Core 2 Extreme Quad-core 2x291M tr. (2006)

©AM

CASA

Processor Transistor Count
(from http://en.wikipedia.org/wiki/Transistor_count)

Processor	Transistor count	Date of intro-duction	Manufactu-rer
Intel 4004	2300	1971	Intel
Intel 8008	2500	1972	Intel
Intel 8080	4500	1974	Intel
Intel 8088	29 000	1978	Intel
Intel 80286	134 000	1982	Intel
Intel 80386	275 000	1985	Intel
Intel 80486	1 200 000	1989	Intel
Pentium	3 100 000	1993	Intel
AMD K5	4 300 000	1996	AMD
Pentium II	7 500 000	1997	Intel
AMD K6	8 800 000	1997	AMD
Pentium III	9 500 000	1999	Intel
AMD K6-III	21 300 000	1999	AMD
AMD K7	22 000 000	1999	AMD
Pentium 4	42 000 000	2000	Intel

Processor	Transistor count	Date of introdu-ction	Manufacturer
Itanium	25 000 000	2001	Intel
Barton	54 300 000	2003	AMD
AMD K8	105 900 000	2003	AMD
Itanium 2	220 000 000	2003	Intel
Itanium 2 with 9MB cache	592 000 000	2004	Intel
Cell	241 000 000	2006	Sony/IBM/Toshiba
Core 2 Duo	291 000 000	2006	Intel
Core 2 Quadro	582 000 000	2006	Intel
Dual-Core Itanium 2	1 700 000 000	2006	Intel

©AM

CASA

Technology Directions: SIA Roadmap
(from 1999)

Year	1999	2002	2005	2008	2011	2014
Feature size (nm)	180	130	100	70	50	35
Logic trans/cm^2	6.2M	18M	39M	84M	180M	390M
Cost/trans (mc)	1.735	.580	.255	.110	.049	.022
#pads/chip	1867	2553	3492	4776	6532	8935
Clock (MHz)	1250	2100	3500	6000	10000	16900
Chip size (mm^2)	340	430	520	620	750	900
Wiring levels	6-7	7	7-8	8-9	9	10
Power supply (V)	1.8	1.5	1.2	0.9	0.6	0.5
High-perf pow (W)	90	130	160	170	175	183

©AM

CASA

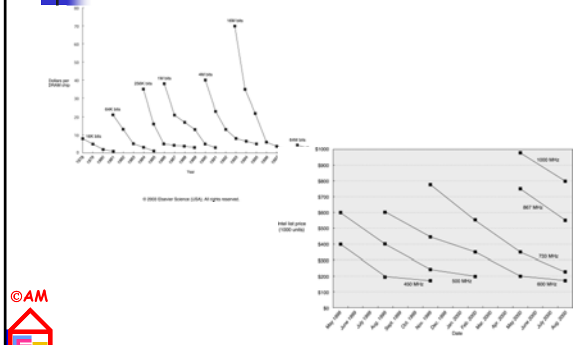
Technology Directions
(ITRS – Int. Tech. Roadmap for Semicon., 2006 ed.)

- ITRS yearly updates
- In year 2017 (10 years from now)
  - Gate length (high-performance MPUs): 13 nm (printed), 8 nm (physical)
  - Functions per chip at production (in million of transistors): 3,092
- For more info check the $HOME/docs/00_ExecSum2006Update.pdf

Cost, Price, and Their Trends

- Price – what you sell a good for
- Cost – what you spent to produce it
- Understanding cost
  - Learning curve principle – manufacturing costs decrease over time (even without major improvements in implementation technology)
    - Best measured by change in yield – the percentage of manufactured devices that survives the testing procedure
  - Volume (number of products manufactured)
    - decreases the time needed to get down the learning curve
    - decreases cost since it increases purchasing and manufacturing efficiency
  - Commodities – products sold by multiple vendors in large volumes which are essentially identical
    - Competition among suppliers lower cost

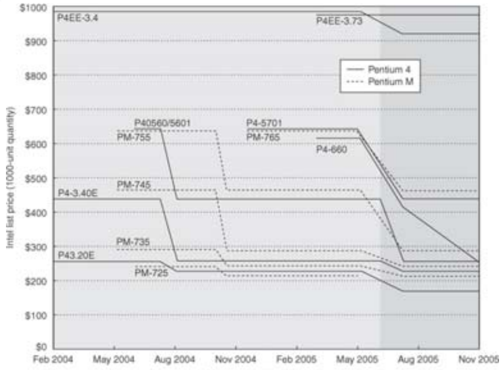
Trends in Cost:
The Price of DRAM and Intel Pentium III



© 2003 Elsevier Science (USA). All rights reserved.

© 2003 Elsevier Science (USA). All rights reserved.

Trends in Cost:
The Price of Pentium4 and PentiumM

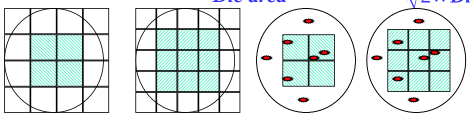


Integrated Circuits Variable Costs

$$IC\ cost = \frac{Die\ cost + Testing\ cost + Packaging\ cost}{Final\ test\ yield}$$

$$Cost\ of\ die = \frac{Cost\ of\ wafer}{Dies\ per\ wafer \times Die\ yield}$$

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



Example: Find the number of dies per 20-cm wafer for a die that is 1.5 cm on a side.
Solution: Die area = 1.5x1.5 = 2.25cm^2.
Dies per wafer = 3.14x(20/2)^2/2.25 – 3.14x20/(2x2.5)$^{0.5}$=110.

Integrated Circuits Cost (cont'd)

- What is the fraction of good dies on a wafer – die yield
- Empirical model
  - defects are randomly distributed over the wafer
  - yield is inversely proportional to the complexity of the fabrication process

$$Die\ yield = Wafer\ yield \times \left(1 + \frac{Defects\ per\ unit\ area \times Die\ area}{\alpha}\right)^{-\alpha}$$

- Wafer yield accounts for wafers that are completely bad (no need to test them); We assume the wafer yield is 100%
- Defects per unit area: typically 0.4 – 0.8 per cm^2
- α corresponds to the number of masking levels; for today's CMOS, a good estimate is α=4.0

Integrated Circuits Cost (cont'd)

- Example: Find die yield for dies with 1 cm and 0.7 cm on a side; defect density is 0.6 per square centimeter

$$Die\ yield = Wafer\ yield \times \left(1 + \frac{Defects\ per\ unit\ area \times Die\ area}{\alpha}\right)^{-\alpha}$$

  - For larger die: (1+0.6x1/4)$^{-4}$=0.57
  - For smaller die: (1+0.6x0.49/4)$^{-4}$=0.75

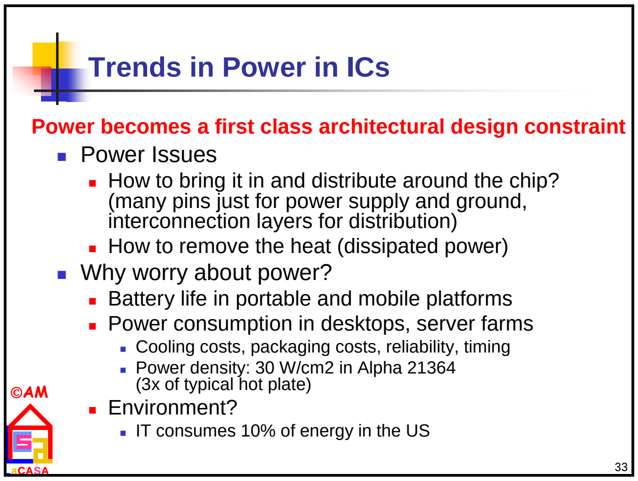
- Die costs are proportional to the fourth power of the die area $Die\ cost = f\left(Die\ area^4\right)$
- In practice $Die\ cost = f\left(Die\ area^2\right)$

Real World Examples

Chip	ML	Line width	Wafer cost	Defect [cm^2]	Area [mm^2]	Dies/ wafer	Yield	Die cost
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 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
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.70	$1500	1.5	296	40	9%	$417

From "Estimating IC Manufacturing Costs," by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

Typical in 2002:
30cm diameter wafer, 4-6 metal layers, wafer cost $5K-6K



Trends in Power in ICs

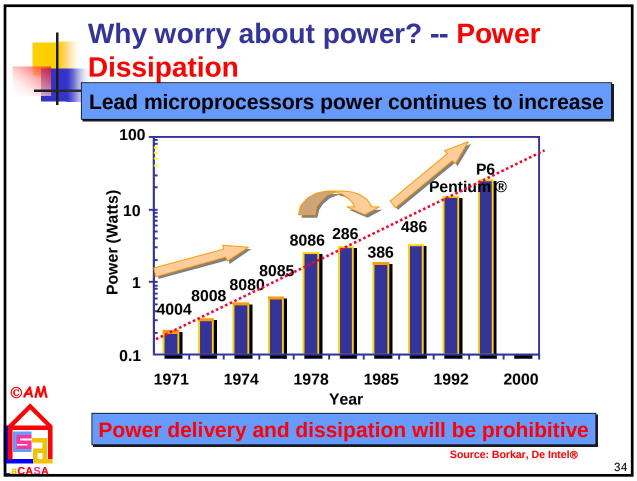
Power becomes a first class architectural design constraint

- Power Issues
  - How to bring it in and distribute around the chip? (many pins just for power supply and ground, interconnection layers for distribution)
  - How to remove the heat (dissipated power)
- Why worry about power?
  - Battery life in portable and mobile platforms
  - Power consumption in desktops, server farms
    - Cooling costs, packaging costs, reliability, timing
    - Power density: 30 W/cm2 in Alpha 21364 (3x of typical hot plate)
  - Environment?
    - IT consumes 10% of energy in the US

©AM

aCASA

33



Why worry about power? -- Power Dissipation

Lead microprocessors power continues to increase

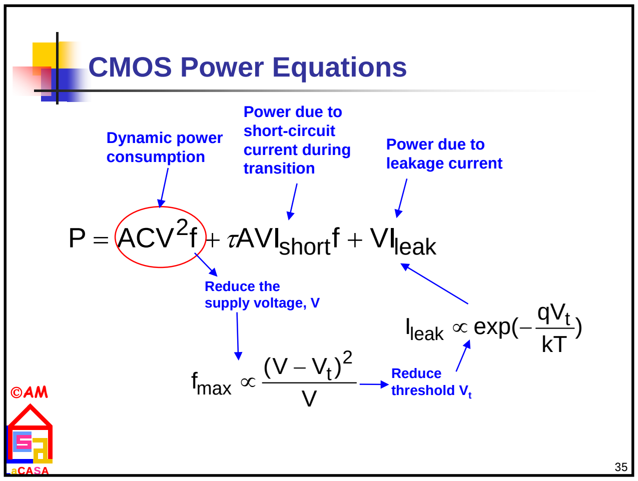
Power delivery and dissipation will be prohibitive

Source: Borkar, De Intel®

©AM

aCASA

34



CMOS Power Equations

$$P = ACV^2 f + \tau A V I_{short} f + V I_{leak}$$

Dynamic power consumption
Power due to short-circuit current during transition
Power due to leakage current
Reduce the supply voltage, V
Reduce threshold V_t

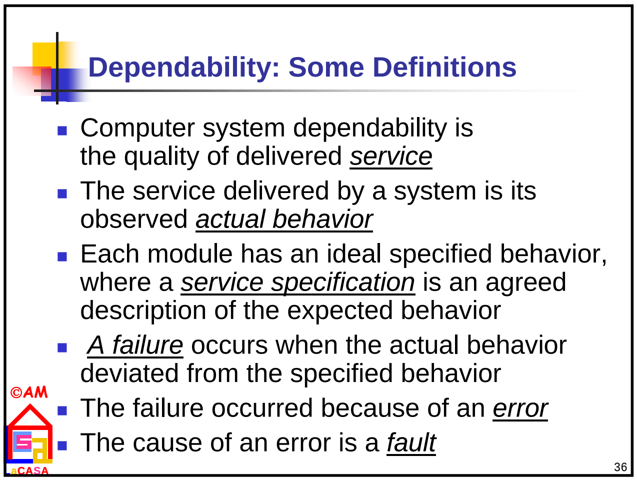
$$I_{leak} \propto \exp(-\frac{qV_t}{kT})$$

$$f_{max} \propto \frac{(V - V_t)^2}{V}$$

©AM

aCASA

35



Dependability: Some Definitions

- Computer system dependability is the quality of delivered *service*
- The service delivered by a system is its observed *actual behavior*
- Each module has an ideal specified behavior, where a *service specification* is an agreed description of the expected behavior
- *A failure* occurs when the actual behavior deviated from the specified behavior
- The failure occurred because of an *error*
- The cause of an error is a *fault*

©AM

aCASA

36

Dependability: Measures

- Service accomplishment vs. service interruption (transitions: failures vs. restorations)
- Module reliability: a measure of the continuous service accomplishment
- A measure of reliability:
  MTTF – Mean Time To Failure
  (1/[rate of failure]) reported in [failure/1billion hours of operation)
- MTTR – Mean time to repair (a measure for service interruption)
- MTBF – Mean time between failures (MTTF+MTTR)
- Module availability – a measure of the service accomplishment; = MTTF/(MTTF+MTTR)

Things to Remember

- Computing classes: desktop, server, embedd.
- Technology trends

	Capacity	Speed
Logic	4x in 3+ years	2x in 3 years
DRAM	4x in 3-4 years	33% in 10 years
Disk	4x in 3-4 years	33% in 10 years

- Cost
  - Learning curve:
    manufacturing costs decrease over time
  - Volume: the number of chips manufactured
  - Commodity

Things to Remember (cont’d)

- Cost of an integrated circuit

$$IC\ cost = \frac{Die\ cost + Testing\ cost + Packaging\ cost}{Final\ test\ yield}$$

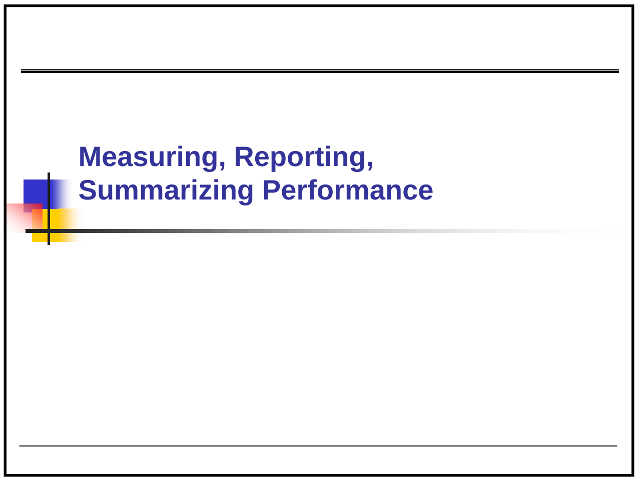
$$Cost\ of\ die = \frac{Cost\ of\ wafer}{Dies\ per\ wafer \times Die\ yield}$$

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

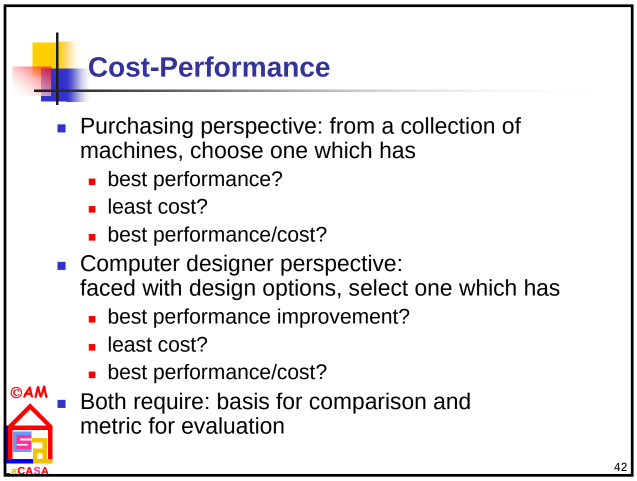
$$Die\ yield = Wafer\ yield \times \left(1 + \frac{Defects\ per\ unit\ area \times Die\ area}{\alpha}\right)^{-\alpha}$$

Design Space

- Performance
- Cost
- Power
- Dependability

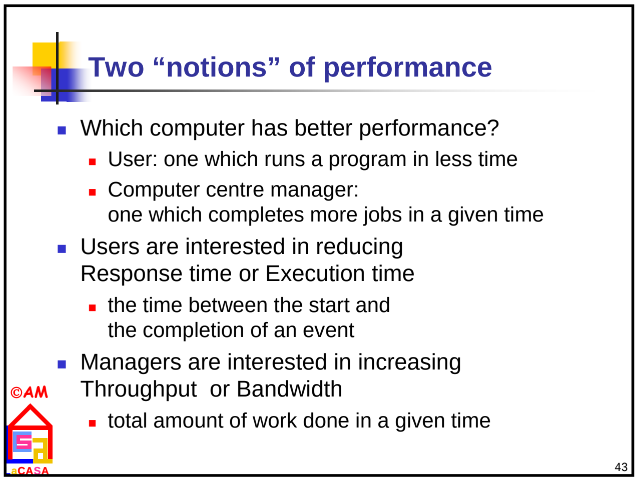


Measuring, Reporting, Summarizing Performance



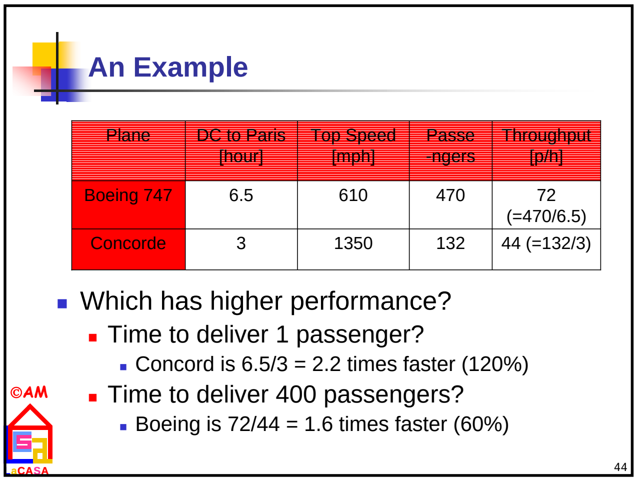
Cost-Performance

- Purchasing perspective: from a collection of machines, choose one which has
  - best performance?
  - least cost?
  - best performance/cost?
- Computer designer perspective:
  faced with design options, select one which has
  - best performance improvement?
  - least cost?
  - best performance/cost?
- Both require: basis for comparison and metric for evaluation



Two “notions” of performance

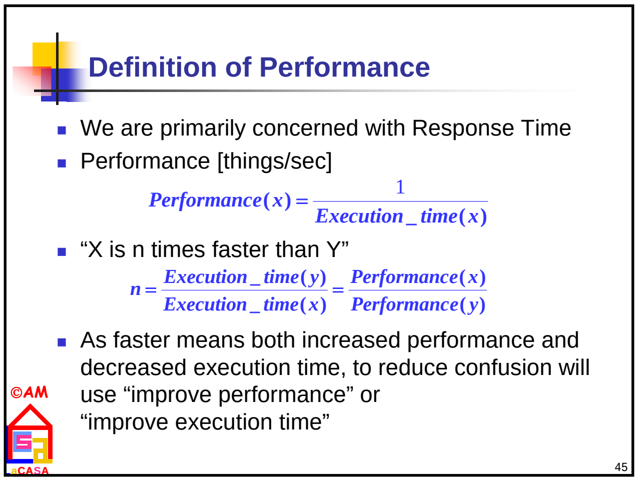
- Which computer has better performance?
  - User: one which runs a program in less time
  - Computer centre manager:
    one which completes more jobs in a given time
- Users are interested in reducing
  Response time or Execution time
  - the time between the start and
    the completion of an event
- Managers are interested in increasing
  Throughput or Bandwidth
  - total amount of work done in a given time



An Example

Plane	DC to Paris [hour]	Top Speed [mph]	Passengers	Throughput [p/h]
Boeing 747	6.5	610	470	72 (=470/6.5)
Concorde	3	1350	132	44 (=132/3)

- Which has higher performance?
  - Time to deliver 1 passenger?
    - Concord is 6.5/3 = 2.2 times faster (120%)
  - Time to deliver 400 passengers?
    - Boeing is 72/44 = 1.6 times faster (60%)



Definition of Performance

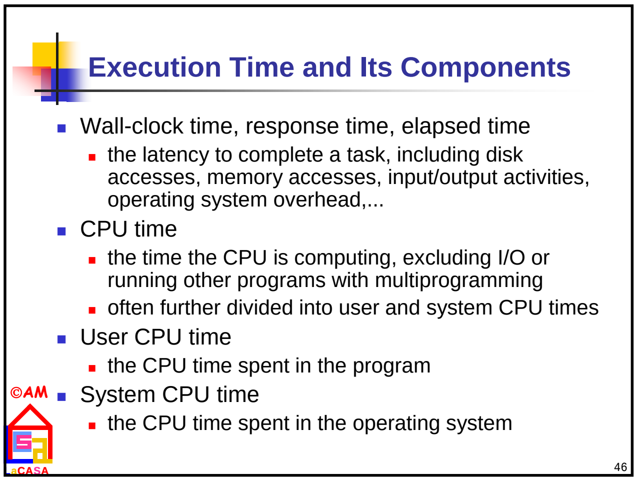
- We are primarily concerned with Response Time
- Performance [things/sec]

$$Performance(x) = \frac{1}{Execution\_time(x)}$$

- “X is n times faster than Y”

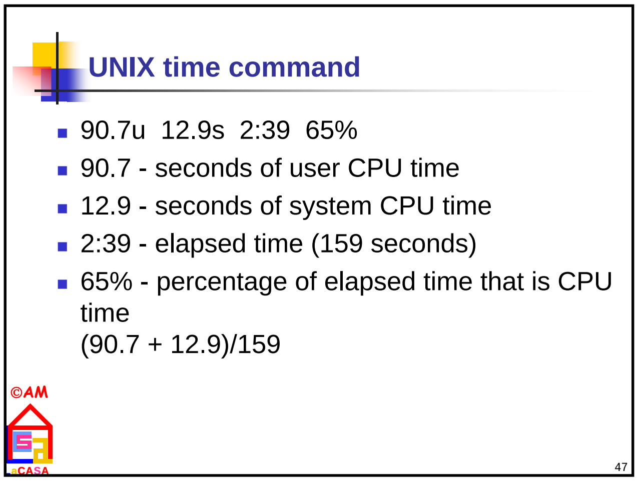
$$n = \frac{Execution\_time(y)}{Execution\_time(x)} = \frac{Performance(x)}{Performance(y)}$$

- As faster means both increased performance and decreased execution time, to reduce confusion will use “improve performance” or “improve execution time”



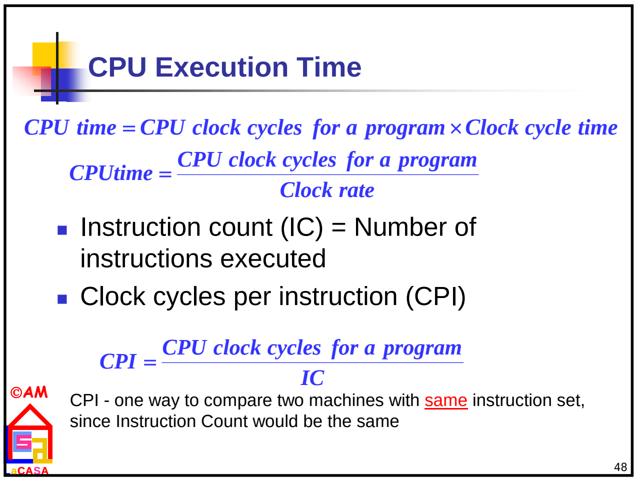
Execution Time and Its Components

- Wall-clock time, response time, elapsed time
  - the latency to complete a task, including disk accesses, memory accesses, input/output activities, operating system overhead,...
- CPU time
  - the time the CPU is computing, excluding I/O or running other programs with multiprogramming
  - often further divided into user and system CPU times
- User CPU time
  - the CPU time spent in the program
- System CPU time
  - the CPU time spent in the operating system



UNIX time command

- 90.7u 12.9s 2:39 65%
- 90.7 - seconds of user CPU time
- 12.9 - seconds of system CPU time
- 2:39 - elapsed time (159 seconds)
- 65% - percentage of elapsed time that is CPU time
  (90.7 + 12.9)/159



CPU Execution Time

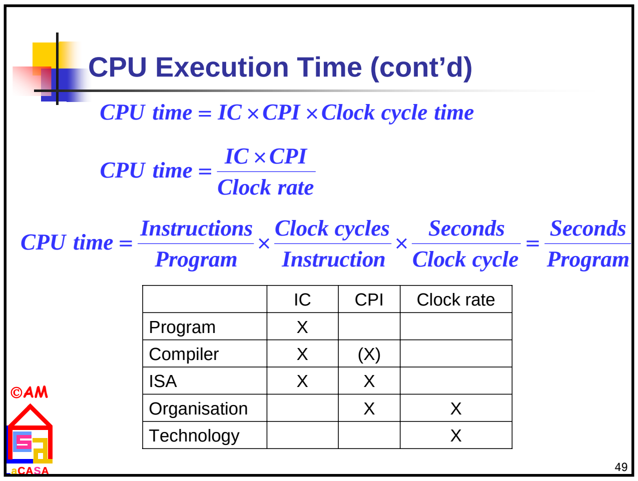
$$CPU\ time = CPU\ clock\ cycles\ \ for\ a\ program \times Clock\ cycle\ time$$

$$CPUtime = \frac{CPU\ clock\ cycles\ \ for\ a\ program}{Clock\ rate}$$

- Instruction count (IC) = Number of instructions executed
- Clock cycles per instruction (CPI)

$$CPI = \frac{CPU\ clock\ cycles\ \ for\ a\ program}{IC}$$

CPI - one way to compare two machines with same instruction set, since Instruction Count would be the same



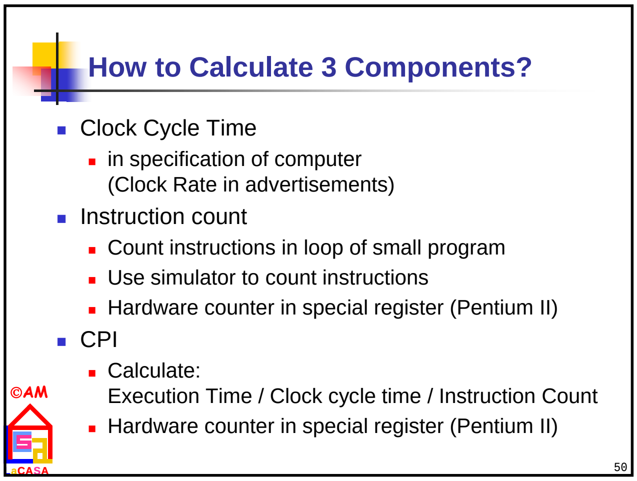
CPU Execution Time (cont'd)

$$CPU\ time = IC \times CPI \times Clock\ cycle\ time$$

$$CPU\ time = \frac{IC \times CPI}{Clock\ rate}$$

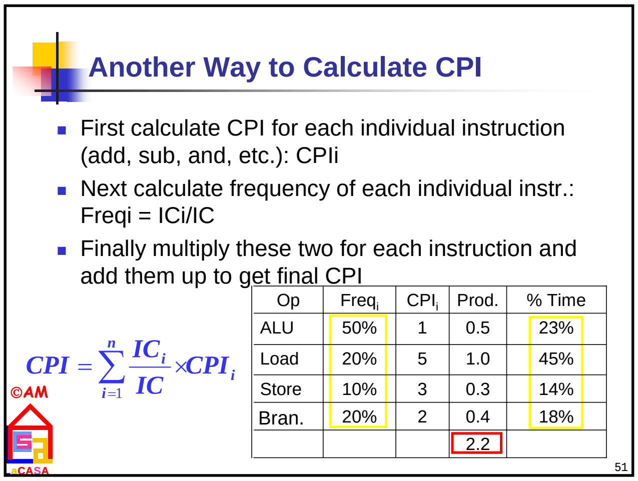
$$CPU\ time = \frac{Instructions}{Program} \times \frac{Clock\ cycles}{Instruction} \times \frac{Seconds}{Clock\ cycle} = \frac{Seconds}{Program}$$

	IC	CPI	Clock rate
Program	X		
Compiler	X	(X)	
ISA	X	X	
Organisation		X	X
Technology			X



How to Calculate 3 Components?

- Clock Cycle Time
  - in specification of computer
    (Clock Rate in advertisements)
- Instruction count
  - Count instructions in loop of small program
  - Use simulator to count instructions
  - Hardware counter in special register (Pentium II)
- CPI
  - Calculate:
    Execution Time / Clock cycle time / Instruction Count
  - Hardware counter in special register (Pentium II)

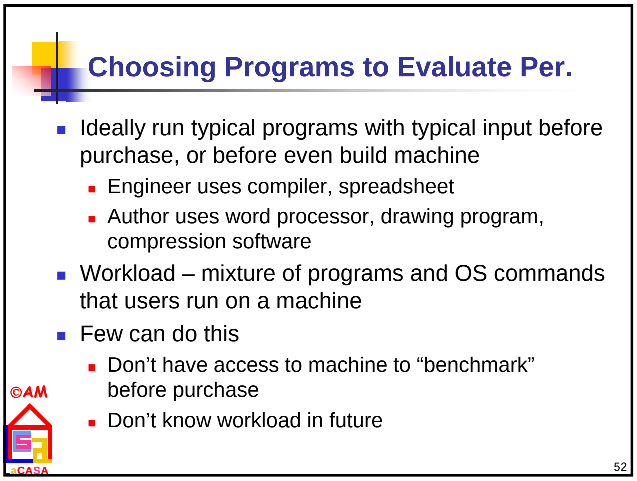


Another Way to Calculate CPI

- First calculate CPI for each individual instruction (add, sub, and, etc.): CPIi
- Next calculate frequency of each individual instr.: Freqi = ICi/IC
- Finally multiply these two for each instruction and add them up to get final CPI

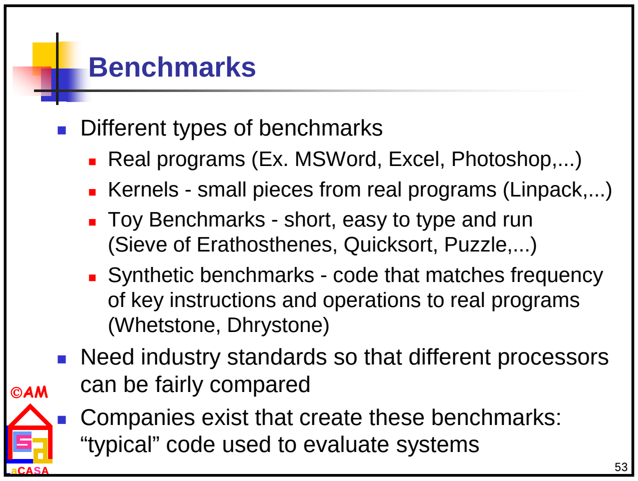
$$CPI = \sum_{i=1}^{n} \frac{IC_i}{IC} \times CPI_i$$

Op	$Freq_i$	CPI_i	Prod.	% Time
ALU	50%	1	0.5	23%
Load	20%	5	1.0	45%
Store	10%	3	0.3	14%
Bran.	20%	2	0.4	18%
			2.2	



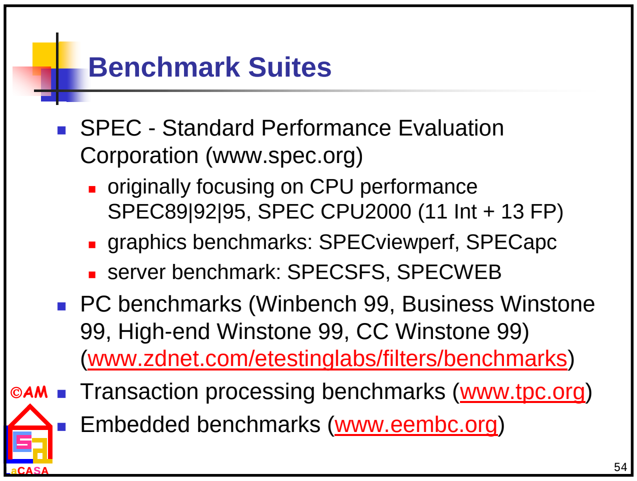
Choosing Programs to Evaluate Per.

- Ideally run typical programs with typical input before purchase, or before even build machine
  - Engineer uses compiler, spreadsheet
  - Author uses word processor, drawing program, compression software
- Workload – mixture of programs and OS commands that users run on a machine
- Few can do this
  - Don't have access to machine to "benchmark" before purchase
  - Don't know workload in future



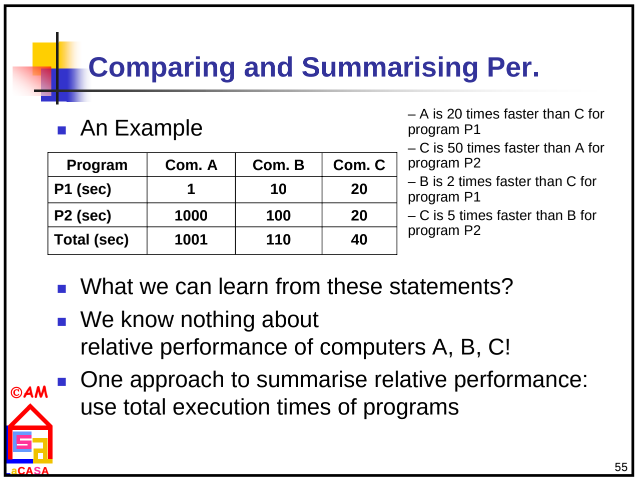
Benchmarks

- Different types of benchmarks
  - Real programs (Ex. MSWord, Excel, Photoshop,...)
  - Kernels - small pieces from real programs (Linpack,...)
  - Toy Benchmarks - short, easy to type and run (Sieve of Erathosthenes, Quicksort, Puzzle,...)
  - Synthetic benchmarks - code that matches frequency of key instructions and operations to real programs (Whetstone, Dhrystone)
- Need industry standards so that different processors can be fairly compared
- Companies exist that create these benchmarks: “typical” code used to evaluate systems



Benchmark Suites

- SPEC - Standard Performance Evaluation Corporation (www.spec.org)
  - originally focusing on CPU performance SPEC89|92|95, SPEC CPU2000 (11 Int + 13 FP)
  - graphics benchmarks: SPECviewperf, SPECapc
  - server benchmark: SPECSFS, SPECWEB
- PC benchmarks (Winbench 99, Business Winstone 99, High-end Winstone 99, CC Winstone 99) (www.zdnet.com/etestinglabs/filters/benchmarks)
- Transaction processing benchmarks (www.tpc.org)
- Embedded benchmarks (www.eembc.org)



Comparing and Summarising Per.

- An Example

Program	Com. A	Com. B	Com. C
P1 (sec)	1	10	20
P2 (sec)	1000	100	20
Total (sec)	1001	110	40

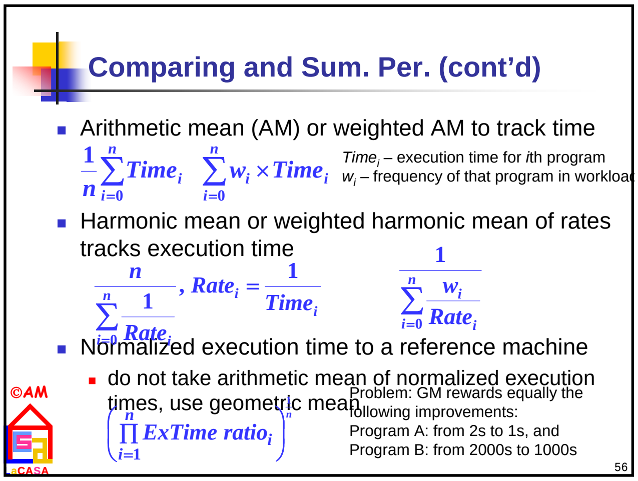
– A is 20 times faster than C for program P1

– C is 50 times faster than A for program P2

– B is 2 times faster than C for program P1

– C is 5 times faster than B for program P2

- What we can learn from these statements?
- We know nothing about relative performance of computers A, B, C!
- One approach to summarise relative performance: use total execution times of programs



Comparing and Sum. Per. (cont’d)

- Arithmetic mean (AM) or weighted AM to track time

$$\frac{1}{n}\sum_{i=0}^{n} Time_i \qquad \sum_{i=0}^{n} w_i \times Time_i$$

Time$_i$ – execution time for *i*th program
w$_i$ – frequency of that program in workloac

- Harmonic mean or weighted harmonic mean of rates tracks execution time

$$\frac{n}{\sum_{i=0}^{n}\frac{1}{Rate_i}}, Rate_i = \frac{1}{Time_i} \qquad \frac{1}{\sum_{i=0}^{n}\frac{w_i}{Rate_i}}$$

- Normalized execution time to a reference machine
  - do not take arithmetic mean of normalized execution times, use geometric mean

$$\left(\prod_{i=1}^{n} ExTime\ ratio_i\right)^{\frac{1}{n}}$$

Problem: GM rewards equally the following improvements:
Program A: from 2s to 1s, and
Program B: from 2000s to 1000s

Quantitative Principles of Design

- Where to spend time making improvements?
  ⇒ Make the Common Case Fast
  - Most important principle of computer design: Spend your time on improvements where those improvements will do the most good
  - Example
    - Instruction A represents 5% of execution
    - Instruction B represents 20% of execution
    - Even if you can drive the time for A to 0, the CPU will only be 5% faster
- Key questions
  - What the frequent case is?
  - How much performance can be improved by making that case faster?

Amdahl's Law

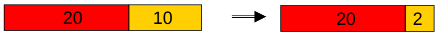
- Suppose that we make an enhancement to a machine that will improve its performance; Speedup is ratio:

$$Speedup = \frac{ExTime\ \ for\ entire\ task\ without\ enhancement}{ExTime\ \ for\ entire\ task\ using\ enhancement}$$

$$Speedup = \frac{Performance\ \ for\ entire\ task\ using\ enhancement}{Performance\ \ for\ entire\ task\ without\ enhancement}$$

- Amdahl's Law states that the performance improvement that can be gained by a particular enhancement is limited by the amount of time that enhancement can be used

Computing Speedup



- Fractionenhanced = fraction of execution time in the original machine that can be converted to take advantage of enhancement (E.g., 10/30)
- Speedupenhanced = how much faster the enhanced code will run (E.g., 10/2=5)
- Execution time of enhanced program will be sum of old execution time of the unenhanced part of program and new execution time of the enhanced part of program:

$$ExTime_{new} = ExTime_{unenhanced} + \frac{ExTime_{enhanced}}{Speedup_{enhanced}}$$

Computing Speedup (cont'd)

- Enhanced part of program is Fractionenhanced, so times are:

$$ExTime_{unenhanced} = ExTime_{old} \times \left(1 - Fraction_{enhanced}\right)$$

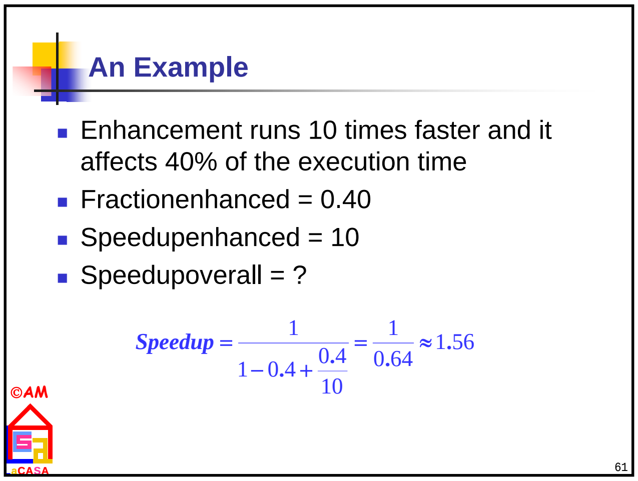
$$ExTime_{enhanced} = ExTime_{old} \times Fraction_{enhanced}$$

- Factor out Timeold and divide by Speedupenhanced:

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

- Overall speedup is ratio of Timeold to Timenew:

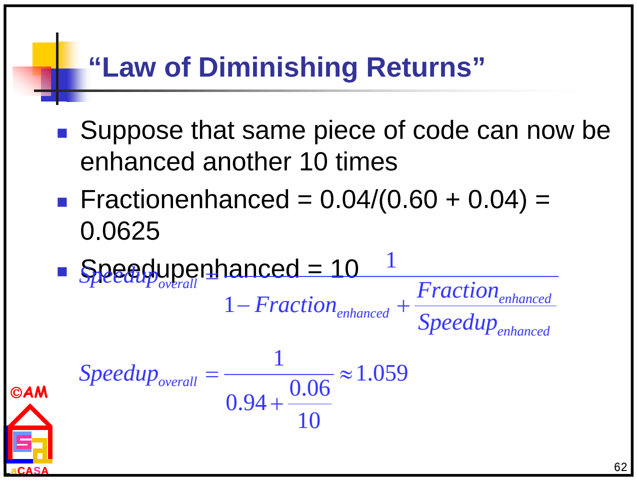
$$Speedup = \frac{1}{1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$



An Example

- Enhancement runs 10 times faster and it affects 40% of the execution time
- Fractionenhanced = 0.40
- Speedupenhanced = 10
- Speedupoverall = ?

$$Speedup = \frac{1}{1-0.4+\frac{0.4}{10}} = \frac{1}{0.64} \approx 1.56$$

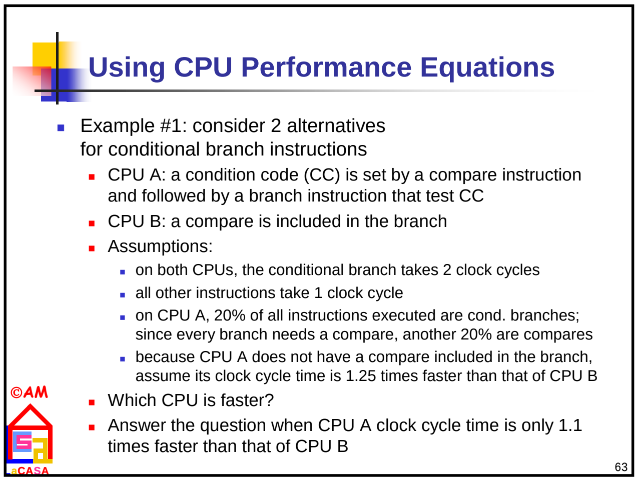


“Law of Diminishing Returns”

- Suppose that same piece of code can now be enhanced another 10 times
- Fractionenhanced = 0.04/(0.60 + 0.04) = 0.0625
- Speedupenhanced = 10

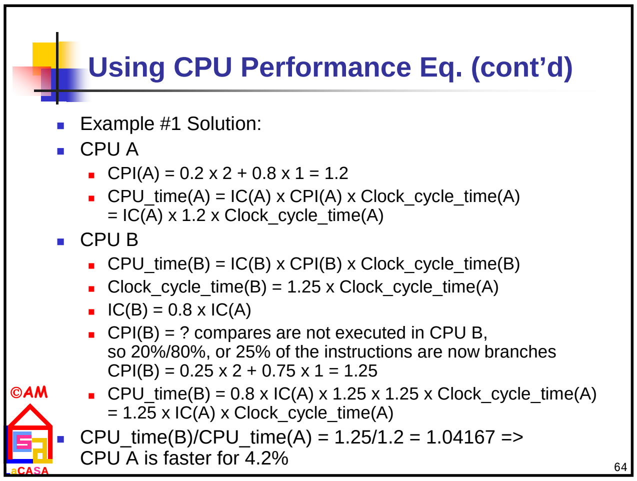
$$Speedup_{overall} = \frac{1}{1-Fraction_{enhanced}+\frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

$$Speedup_{overall} = \frac{1}{0.94+\frac{0.06}{10}} \approx 1.059$$



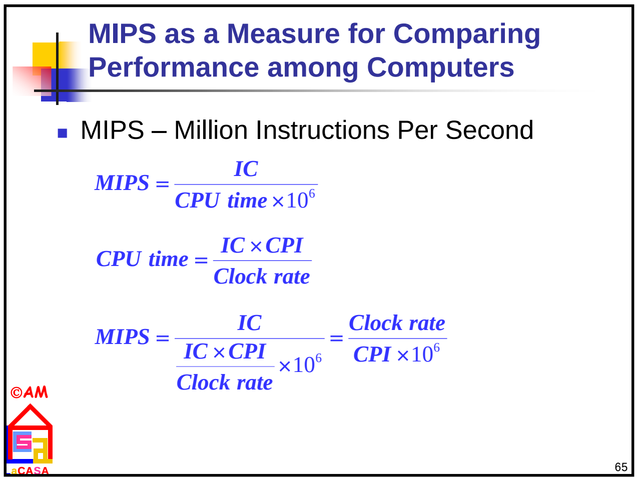
Using CPU Performance Equations

- Example #1: consider 2 alternatives for conditional branch instructions
  - CPU A: a condition code (CC) is set by a compare instruction and followed by a branch instruction that test CC
  - CPU B: a compare is included in the branch
  - Assumptions:
    - on both CPUs, the conditional branch takes 2 clock cycles
    - all other instructions take 1 clock cycle
    - on CPU A, 20% of all instructions executed are cond. branches; since every branch needs a compare, another 20% are compares
    - because CPU A does not have a compare included in the branch, assume its clock cycle time is 1.25 times faster than that of CPU B
  - Which CPU is faster?
  - Answer the question when CPU A clock cycle time is only 1.1 times faster than that of CPU B



Using CPU Performance Eq. (cont’d)

- Example #1 Solution:
- CPU A
  - CPI(A) = 0.2 x 2 + 0.8 x 1 = 1.2
  - CPU_time(A) = IC(A) x CPI(A) x Clock_cycle_time(A)
    = IC(A) x 1.2 x Clock_cycle_time(A)
- CPU B
  - CPU_time(B) = IC(B) x CPI(B) x Clock_cycle_time(B)
  - Clock_cycle_time(B) = 1.25 x Clock_cycle_time(A)
  - IC(B) = 0.8 x IC(A)
  - CPI(B) = ? compares are not executed in CPU B,
    so 20%/80%, or 25% of the instructions are now branches
    CPI(B) = 0.25 x 2 + 0.75 x 1 = 1.25
  - CPU_time(B) = 0.8 x IC(A) x 1.25 x 1.25 x Clock_cycle_time(A)
    = 1.25 x IC(A) x Clock_cycle_time(A)
- CPU_time(B)/CPU_time(A) = 1.25/1.2 = 1.04167 =>
  CPU A is faster for 4.2%



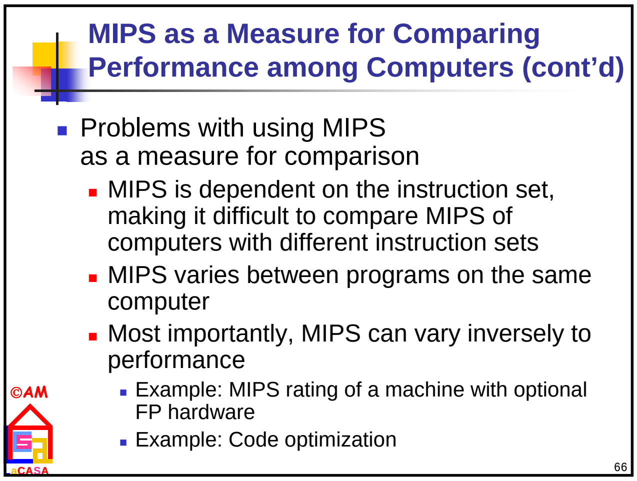
MIPS as a Measure for Comparing Performance among Computers

- MIPS – Million Instructions Per Second

$$MIPS = \frac{IC}{CPU\ time \times 10^6}$$

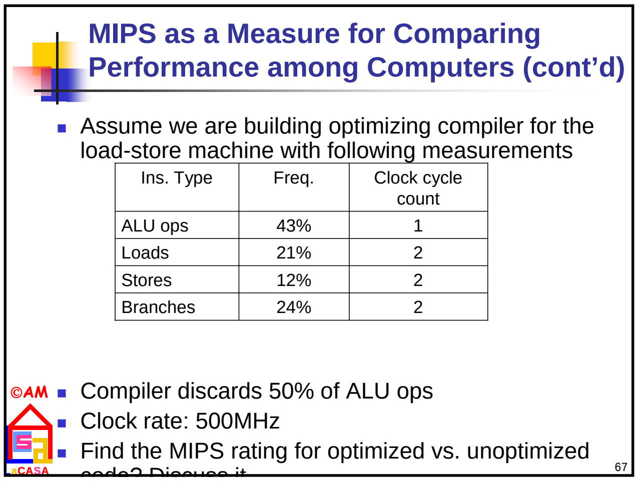
$$CPU\ time = \frac{IC \times CPI}{Clock\ rate}$$

$$MIPS = \frac{IC}{\frac{IC \times CPI}{Clock\ rate} \times 10^6} = \frac{Clock\ rate}{CPI \times 10^6}$$



MIPS as a Measure for Comparing Performance among Computers (cont'd)

- Problems with using MIPS as a measure for comparison
  - MIPS is dependent on the instruction set, making it difficult to compare MIPS of computers with different instruction sets
  - MIPS varies between programs on the same computer
  - Most importantly, MIPS can vary inversely to performance
    - Example: MIPS rating of a machine with optional FP hardware
    - Example: Code optimization

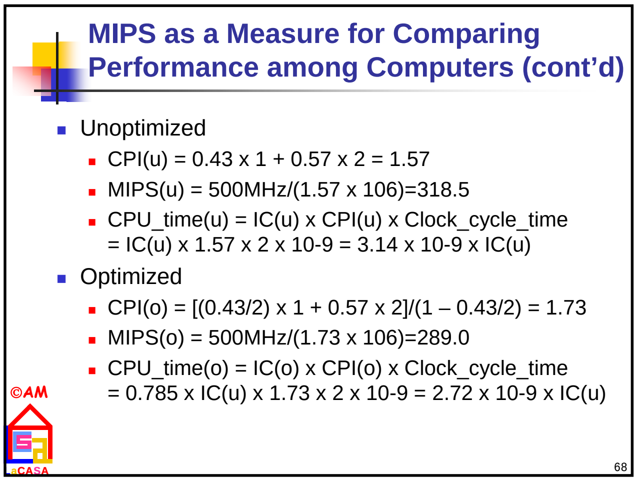


MIPS as a Measure for Comparing Performance among Computers (cont'd)

- Assume we are building optimizing compiler for the load-store machine with following measurements

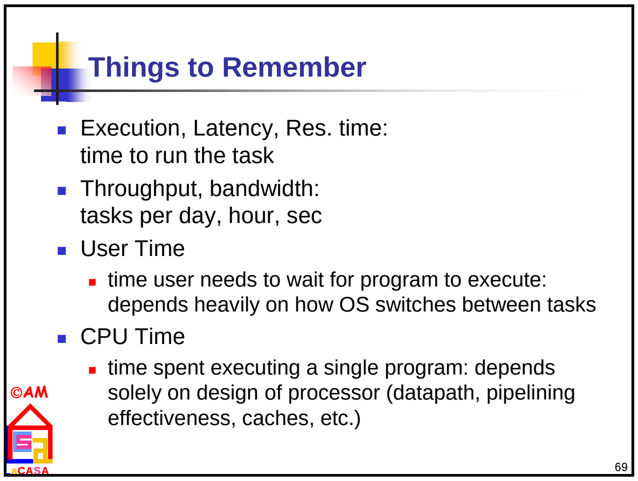
Ins. Type	Freq.	Clock cycle count
ALU ops	43%	1
Loads	21%	2
Stores	12%	2
Branches	24%	2

- Compiler discards 50% of ALU ops
- Clock rate: 500MHz
- Find the MIPS rating for optimized vs. unoptimized code? Discuss it



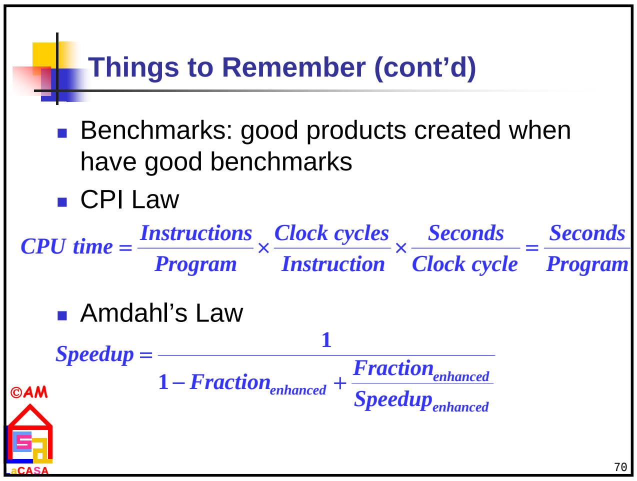
MIPS as a Measure for Comparing Performance among Computers (cont'd)

- Unoptimized
  - CPI(u) = 0.43 x 1 + 0.57 x 2 = 1.57
  - MIPS(u) = 500MHz/(1.57 x 106)=318.5
  - CPU_time(u) = IC(u) x CPI(u) x Clock_cycle_time = IC(u) x 1.57 x 2 x 10-9 = 3.14 x 10-9 x IC(u)
- Optimized
  - CPI(o) = [(0.43/2) x 1 + 0.57 x 2]/(1 – 0.43/2) = 1.73
  - MIPS(o) = 500MHz/(1.73 x 106)=289.0
  - CPU_time(o) = IC(o) x CPI(o) x Clock_cycle_time = 0.785 x IC(u) x 1.73 x 2 x 10-9 = 2.72 x 10-9 x IC(u)



Things to Remember

- Execution, Latency, Res. time: time to run the task
- Throughput, bandwidth: tasks per day, hour, sec
- User Time
  - time user needs to wait for program to execute: depends heavily on how OS switches between tasks
- CPU Time
  - time spent executing a single program: depends solely on design of processor (datapath, pipelining effectiveness, caches, etc.)



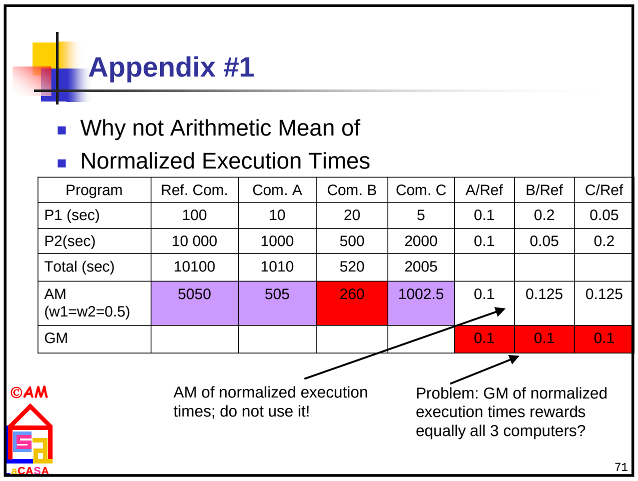
Things to Remember (cont’d)

- Benchmarks: good products created when have good benchmarks
- CPI Law

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

- Amdahl’s Law

$$Speedup = \frac{1}{1 - Fraction_{enhanced} + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$



Appendix #1

- Why not Arithmetic Mean of
- Normalized Execution Times

Program	Ref. Com.	Com. A	Com. B	Com. C	A/Ref	B/Ref	C/Ref
P1 (sec)	100	10	20	5	0.1	0.2	0.05
P2(sec)	10 000	1000	500	2000	0.1	0.05	0.2
Total (sec)	10100	1010	520	2005			
AM (w1=w2=0.5)	5050	505	260	1002.5	0.1	0.125	0.125
GM					0.1	0.1	0.1

AM of normalized execution times; do not use it!

Problem: GM of normalized execution times rewards equally all 3 computers?