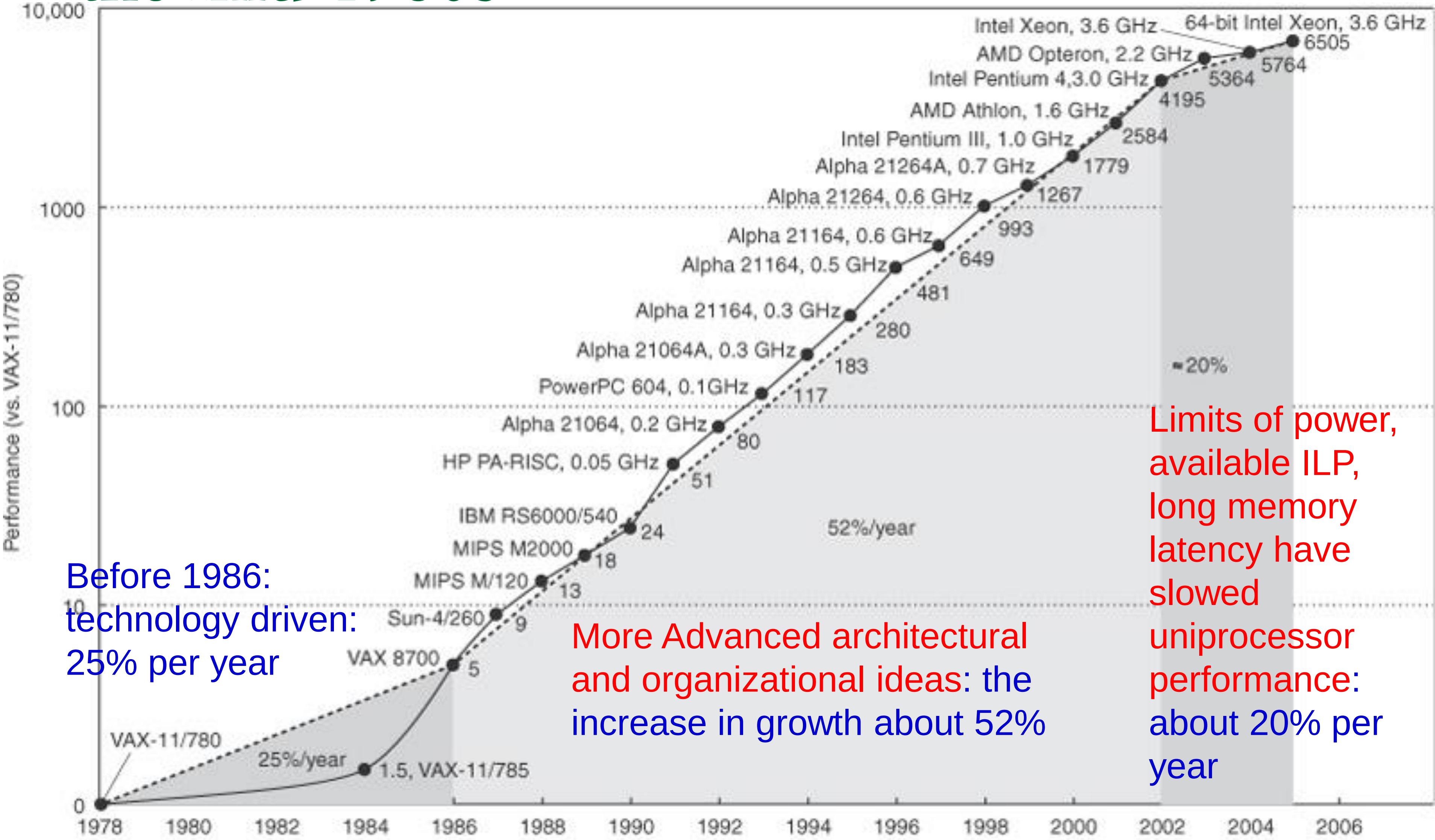
Fundamentals of Computer Design

Growth in Processor Performance since the mid-1980s



Before 1986: technology driven: 25% per year

More Advanced architectural and organizational ideas: the increase in growth about 52%

Limits of power, available ILP, long memory latency have slowed uniprocessor performance: about 20% per year

History

- Simpler instructions: Reduced Instruction Set Computer (RISC) in early 1980s
  - Focused the attention of designers on Instruction Level Parallelism and the use of caches
- RISC-based computers force prior architectures to keep up or disappear
- Intel rose to the challenge and translated x86 instructions into RISC-like instructions internally
- As transistor counts soared in late 1990s, the hardware overhead of translating the more complex x86 architecture became negligible.

RISC

- For any given level of general performance, a RISC chip will typically have far fewer transistors dedicated to the core logic which originally **allowed designers to increase the size of the register set** and increase internal parallelism.
- Uniform instruction format, using a single word with the opcode in the same bit positions in every instruction, demanding less decoding
- Identical general purpose registers, allowing any register to be used in any context, **simplifying compiler design** (although normally there are separate floating point registers)
- Simple addressing modes, with complex addressing performed via sequences of arithmetic and/or load-store operations

16 years of sustained growth in performance at an annual rate of over 50%

- It has significantly enhanced the capability available to computer users.
- For many applications, the highest-performance microprocessors of today outperform the supercomputers of less than 10 years ago.
- It has led to dominance of microprocessor-based computers.
  - PC and workstations are major products
  - Minicomputers are replaced by servers made with microprocessors
  - Even high-end supercomputers are being built with collections of microprocessors

The 16-year renaissance is over

- Performance dropped to about 20% per year
  - Maximum power dissipation of air-cooled chips
  - Little ILP (instruction-level parallelism) left to exploit efficiently
  - Almost unchanged memory latency

- The road to higher performance would be via multiple processor per chip rather than faster uni-processors

Classes of Computers

Classes of computers

- Personal Mobile Device
  - Price of System: $100-$1000
  - Price of microprocessor module: $10-$100
  - Cost, Energy, Media Performance, Responsiveness
- Desktop computing
  - Price of System: $300-$2500
  - Price of microprocessor module: $50-$500 (per processor)
  - Price performance, Graphics performance, Energy
- Servers
  - Price of System: $5000-$10,000,000
  - Price of microprocessor module: $200-$20,000
  - Throughput, Availability, Scalability , Energy

Classes of computers

- Clusters/Warehouse-Scale Computers
  - Price of System: $100,000-$200,000,000
  - Price of microprocessor module: $0.01-$100
  - Price-performance, Throughput, Energy Proportionality

- Embedded computing
  - Price of System: $10-$100,000 (include high-end network routers)
  - Price of microprocessor module: $0.01-$100
  - Price, Application-specific performance, Energy

Cost of downtime for an unavailable system

Application	Cost of downtime per hour (thousands of $)	Annual losses (millions of $) with downtime of		
		1% (87.6 hrs/yr)	0.5% (43.8 hrs/yr)	0.1% (8.8 hrs/yr)
Brokerage operations	$6450	$565	$283	$56.5
Credit card authorization	$2600	$228	$114	$22.8
Package shipping services	$150	$13	$6.6	$1.3
Home shopping channel	$113	$9.9	$4.9	$1.0
Catalog sales center	$90	$7.9	$3.9	$0.8
Airline reservation center	$89	$7.9	$3.9	$0.8
Cellular service activation	$41	$3.6	$1.8	$0.4
Online network fees	$25	$2.2	$1.1	$0.2
ATM service fees	$14	$1.2	$0.6	$0.1

Figure 1.3 The cost of an unavailable system is shown by analyzing the cost of downtime (in terms of immediately lost revenue), assuming three different levels of availability, and that downtime is distributed uniformly. These data are from Kembel [2000] and were collected and analyzed by Contingency Planning Research.

Defining Computer Architecture

Defining Computer Architecture

- In the past, computer architecture often referred to instruction set design.
  - Other aspects were called “implementation”.
- The architect’s job is much more than instruction set design.
- Task of computer designer:
  - Determine what attributes are important
  - Maximize performance while staying with cost, power, availability constraints

Issues of Instruction Set Architecture (ISA)

- Class of ISA
- Memory Addressing
- Addressing modes
- Types and sizes of operands
- Operations
- Control flow instructions
- Encoding an ISA

Class of ISA

- ISAs today: General-purpose register architecture
- 80x86
  - 16 general purpose registers, 16 floating point registers
  - Register-memory
- MIPS
  - 32 general purpose registers, 32 floating point registers
  - Load-store
- Recent ISAs are load-store

Memory addressing

- Byte addressing
- MIPS requires that objects must be aligned (accesses are faster)
- 80x86 does not require alignment

Addressing Mode

- Register
- Immediate (for constants)
- Displacement
- Register Indirect
- Indexed
- Direct/Absolute
- Memory Indirect

Types and sizes of operands

- MIPS and 80x86 support operand sizes of 8-bit (ASCII character), 16-bit (Uni-code character or half word), 32-bit (integer or word), 64-bit (double word or long integer), and IEEE 754 floating point in 32-bit (single precision) and 64-bit (double precision)
- 80x86 also supports 80-bit floating point (extended double precision)

Name	Number	Use	Preserved across a call?
$zero	0	The constant value 0	N.A.
$at	1	Assembler temporary	No
$v0–$v1	2–3	Values for function results and expression evaluation	No
$a0–$a3	4–7	Arguments	No
$t0–$t7	8–15	Temporaries	No
$s0–$s7	16–23	Saved temporaries	Yes
$t8–$t9	24–25	Temporaries	No
$k0–$k1	26–27	Reserved for OS kernel	No
$gp	28	Global pointer	Yes
$sp	29	Stack pointer	Yes
$fp	30	Frame pointer	Yes
$ra	31	Return address	Yes

Figure 1.4 MIPS registers and usage conventions. In addition to the 32 general-purpose registers (R0–R31), MIPS has 32 floating-point registers (F0–F31) that can hold either a 32-bit single-precision number or a 64-bit double-precision number.

Operations

Subset of MIPS 64

Instruction type/opcode	Instruction meaning
Data transfers	*Move data between registers and memory, or between the integer and FP or special registers; only memory address mode is 16-bit displacement + contents of a GPR*
`LB, LBU, SB`	Load byte, load byte unsigned, store byte (to/from integer registers)
`LH, LHU, SH`	Load half word, load half word unsigned, store half word (to/from integer registers)
`LW, LWU, SW`	Load word, load word unsigned, store word (to/from integer registers)
`LD, SD`	Load double word, store double word (to/from integer registers)
`L.S, L.D, S.S, S.D`	Load SP float, load DP float, store SP float, store DP float
`MFC0, MTC0`	Copy from/to GPR to/from a special register
`MOV.S, MOV.D`	Copy one SP or DP FP register to another FP register
`MFC1, MTC1`	Copy 32 bits to/from FP registers from/to integer registers
Arithmetic/logical	*Operations on integer or logical data in GPRs; signed arithmetic trap on overflow*
`DADD, DADDI, DADDU, DADDIU`	Add, add immediate (all immediates are 16 bits); signed and unsigned
`DSUB, DSUBU`	Subtract; signed and unsigned
`DMUL, DMULU, DDIV, DDIVU, MADD`	Multiply and divide, signed and unsigned; multiply-add; all operations take and yield 64-bit values
`AND, ANDI`	And, and immediate
`OR, ORI, XOR, XORI`	Or, or immediate, exclusive or, exclusive or immediate
`LUI`	Load upper immediate; loads bits 32 to 47 of register with immediate, then sign-extends
`DSLL, DSRL, DSRA, DSLLV, DSRLV, DSRAV`	Shifts: both immediate (`DS__`) and variable form (`DS__V`); shifts are shift left logical, right logical, right arithmetic
`SLT, SLTI, SLTU, SLTIU`	Set less than, set less than immediate; signed and unsigned
Control	*Conditional branches and jumps; PC-relative or through register*
`BEQZ, BNEZ`	Branch GPRs equal/not equal to zero; 16-bit offset from PC + 4
`BEQ, BNE`	Branch GPR equal/not equal; 16-bit offset from PC + 4
`BC1T, BC1F`	Test comparison bit in the FP status register and branch; 16-bit offset from PC + 4
`MOVN, MOVZ`	Copy GPR to another GPR if third GPR is negative, zero
`J, JR`	Jumps: 26-bit offset from PC + 4 (J) or target in register (JR)
`JAL, JALR`	Jump and link: save PC + 4 in R31, target is PC-relative (JAL) or a register (JALR)
`TRAP`	Transfer to operating system at a vectored address
`ERET`	Return to user code from an exception; restore user mode
Floating point	*FP operations on DP and SP formats*
`ADD.D, ADD.S, ADD.PS`	Add DP, SP numbers, and pairs of SP numbers
`SUB.D, SUB.S, SUB.PS`	Subtract DP, SP numbers, and pairs of SP numbers
`MUL.D, MUL.S, MUL.PS`	Multiply DP, SP floating point, and pairs of SP numbers
`MADD.D, MADD.S, MADD.PS`	Multiply-add DP, SP numbers, and pairs of SP numbers
`DIV.D, DIV.S, DIV.PS`	Divide DP, SP floating point, and pairs of SP numbers
`CVT._._`	Convert instructions: `CVT.x.y` converts from type `x` to type `y`, where `x` and `y` are `L` (64-bit integer), `W` (32-bit integer), `D` (DP), or `S` (SP). Both operands are FPRs.
`C.__.D, C.__.S`	DP and SP compares: "__" = `LT,GT,LE,GE,EQ,NE`; sets bit in FP status register

Figure 1.5 Subset of the instructions in MIPS64. SP = single precision; DP = double precision. Appendix B gives much more detail on MIPS64. For data, the most significant bit number is 0; least is 63.

Control flow instructions

- Conditional branches
- Unconditional jumps
- PC-relative addressing (branch address is specified by and address field that is added to Program Counter)
- Procedure Calls
- Returns

Encoding an ISA

- MIPS
  - Fixed length encoding
  - 32-bit
  - Simplifies instruction decoding
- 80x86
  - Variable length encoding
  - 1 to 18 bytes
  - Takes less space than fixed-length instructions

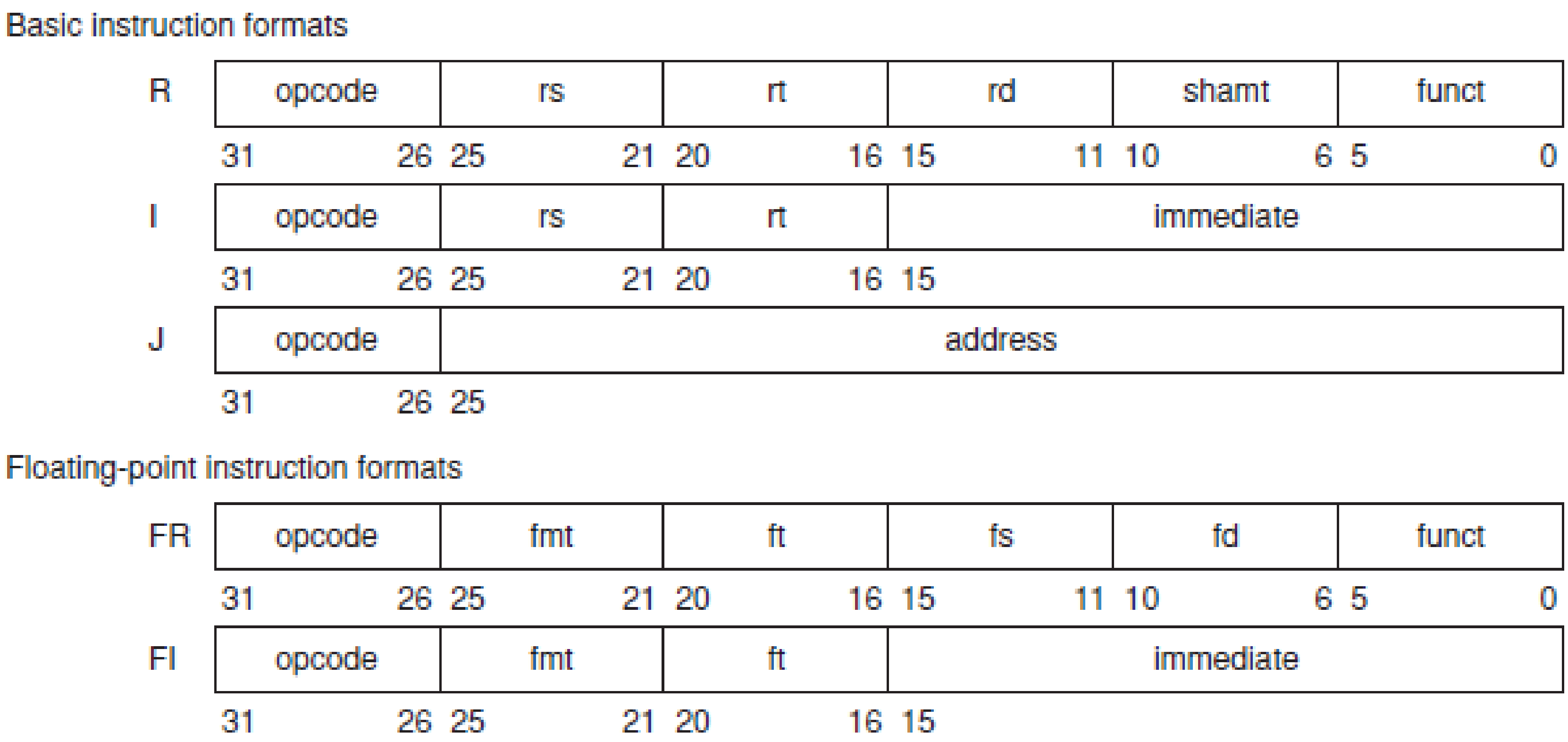


Figure 1.6 MIPS64 instruction set architecture formats. All instructions are 32 bits long. The R format is for integer register-to-register operations, such as DADDU, DSUBU, and so on. The I format is for data transfers, branches, and immediate instructions, such as LD, SD, BEQZ, and DADDIs. The J format is for jumps, the FR format for floating point operations, and the FI format for floating point branches.

Trends in Technology

Trends in power in integrated circuits

- For CMOS chips, the traditional dominant energy consumptions has been in switching transistors, also called dynamic power (watts)

$$Power_{dynamic} = \frac{1}{2} \cdot Capacitive\ load \cdot Voltage^2 \cdot Frequency\ switched$$

- Energy (joules)

$$Energy_{dynamic} = Capacitive\ load \cdot Voltage^2$$

- Voltages: dropped form 5v to 1v in 20 years
- Capacitive load: function(#transistors connected to an output, technology determining the capacitance of the wires and transistors))
- For a fixed task, slowing clock rate reduces power, not energy

Example of trends in power

- Microprocessors are designed to have adjustable voltage
- 15% reduction in voltage may result in 15% reduction in frequency
- What would be the impact on dynamic power?

$$\frac{Power_{new}}{Power_{old}} = ?$$

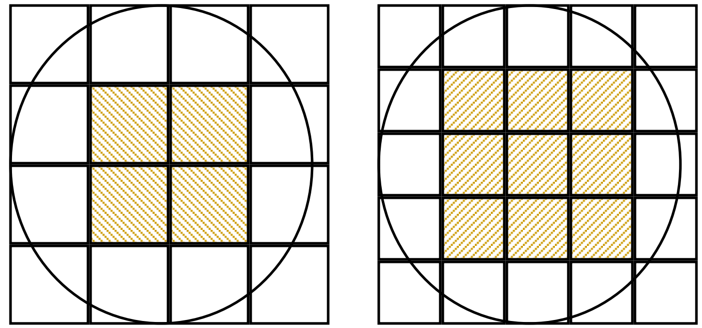
Trends in power in integrated circuits

- The increase in the number of transistors switching, the frequency of the switching leading to an overall growth in power consumption and energy
  - First microprocessor consumes tenths of a watt
  - 3.2G pentium 4 extreme edition consumes 135 watts
- The limit of what can be cooled by air (the heat must be dissipated from a chip that is about 1 cm on a side)
- Increasingly difficult challenges
  - Distributing the power, removing the heat, preventing hot spots

Trends in power in integrated circuits

- Although dynamic power is the primary source of power dissipating in CMOS, static power is becoming an important issue too because leakage current flows even when a transistor is off
- Limits of air cooling led to exploration of multiple processors on a chip running at relatively lower voltages and clock rates.

Integrated Circuits Costs



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

$$\text{Die cost} = \frac{\text{Wafer cost}}{\text{Dies per Wafer} \times \text{Die yield}}$$

$$\text{Dies per wafer} = \frac{\pi\,(\text{Wafer\_diam}/2)^2}{\text{Die\_Area}} - \frac{\pi \times \text{Wafer\_diam}}{\sqrt{2 \cdot \text{Die\_Area}}} - \text{Test\_Die}$$

- Find the number of dies per 300mm wafer for a die that is 1.5cm on a side

Die Area = 2.25 cm^2

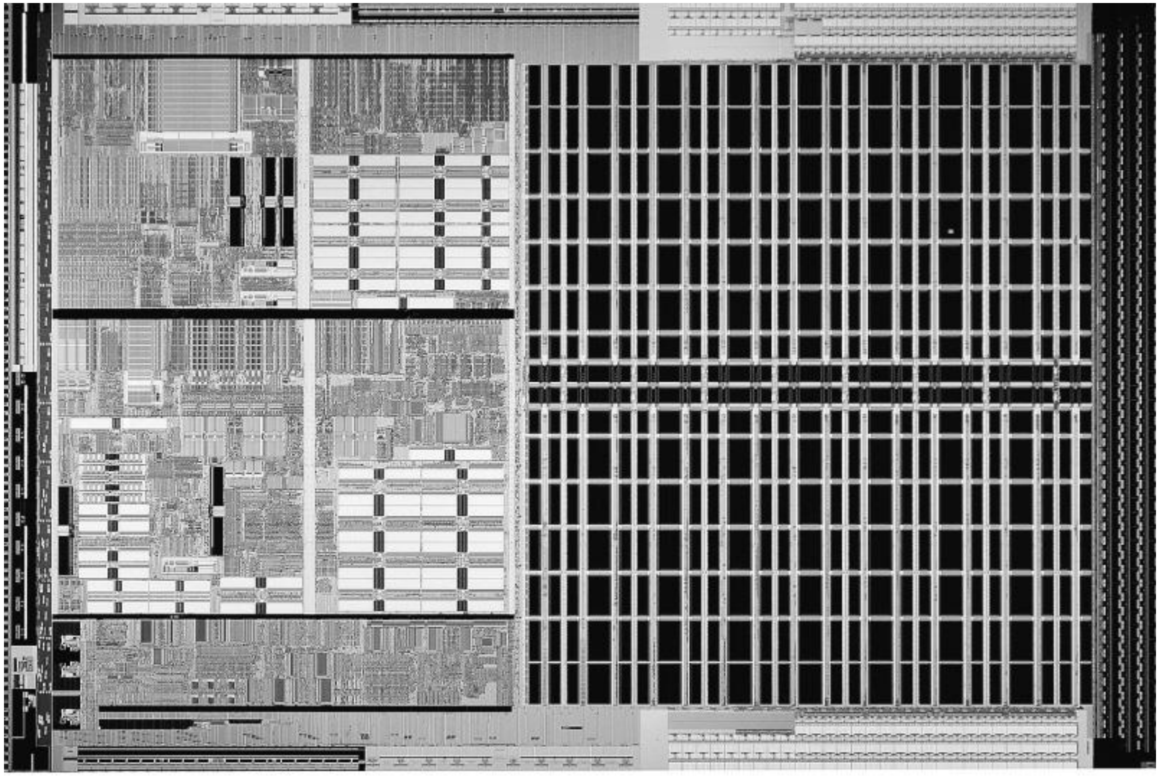
Dies per wafer = pi*$(15)^2$/2.25 - pi*30/sqrt(2)*1.5

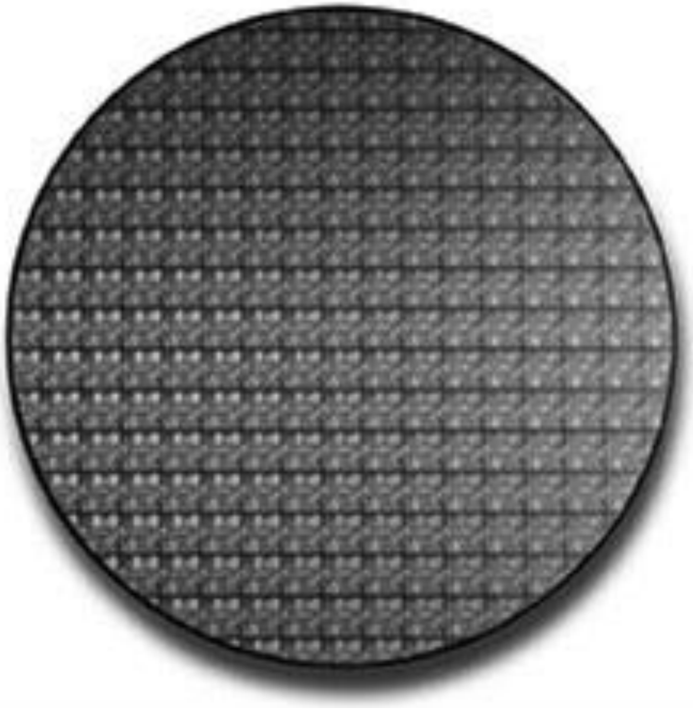
= 706.9/2.25 - 94.2/2.12

= 270

AMD Opteron microprocessor die

300mm wafer contains 117 AMD Opteron chips implemented in a 90 nm process





Integrated Circuits Costs

- The critical question is: what is the fraction of good dies on a wafer (die yield)
- Empirical model developed by looking at the yield of many manufacturing lines

$$\text{Die Yield} = \text{Wafer\_yield} \times \left\{ 1 + \left(\frac{\text{Defect\_Density} \times \text{Die\_area}}{\alpha} \right) \right\}^{-\alpha}$$

- **Wafer yield accounts for wafers that are** completely bad and need not be tested

Example of Die Yield

- Assume wafer yield 100%
- Defects per unit area: 0.4 defects per square centimeter for 90nm process
- For Multilevel metal CMOS processes in 2006 , alpha is around 4
- Find the die yield for dies that are (A) 1.5 cm on a side and (B) 1.0 cm on a side
  - DieAreaA=2.25
  - DieYieldA=$(1 + 0.4\mathrm{x}2.25/4)^{-4}$=0.44
  - DieAreaB=1.00
  - DieYieldB=$(1 + 0.4\mathrm{x}1.00/4)^{-4}$=0.68

$$\text{Die Yield} = \text{Wafer\_yield} \times \left\{ 1 + \left(\frac{\text{Defect\_Density} \times \text{Die\_area}}{\alpha} \right) \right\}^{-\alpha}$$

Why should a computer designer know about chip costs

- Manufacturing process dictates the wafer cost, wafer yield, and defects per unit area
- The computer designers can only control the die area.
- In practice
  - defects per area is small
  - the cost per die (determined by the number of good dies per wafer) roughly grows with the die area
- The computer designer affects die size and hence cost, by
  - What functions are included on or excluded from the die
  - Number of I/O pins

$$\text{Die Yield} = \text{Wafer\_yield} \times \left\{ 1 + \left(\frac{\text{Defect\_Density} \times \text{Die\_area}}{\alpha} \right) \right\}^{-\alpha}$$

Define and quantity dependability

- How to decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable (reliable)
- Systems alternate between 2 states of service with respect to an SLA:

1. Service accomplishment, where the service is delivered as specified in SLA
2. Service interruption, where the delivered service is different from the SLA

- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1

Define and quantify 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* measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- *Module availability = MTTF / (MTTF + MTTR)*

Example calculating reliability

- If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules
- Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):

$$FailureRate =$$

$$MTTF =$$

Example of Redundant Power Supplies

- Disk systems often have redundant power supplies to improve dependability.

Mean time until one power supply fails

$$MTTF_{\text{powersupplypair}} = \frac{MTTF_{\text{powersupply}} / 2}{MTTR_{\text{powersupply}} / MTTF_{\text{powersupply}}}$$

The other power supply fails before the first one is repaired

Example of Redundant Power Supplies

- Assume MTTF of power supply: 0.2M hour
- Assume it takes on average 24 hours for an operator to notice that a power supply has failed and replace it.

$$MTTF_{\text{powersupplypair}} = \frac{MTTF_{\text{powersupply}} / 2}{MTTR_{\text{powersupply}} / MTTF_{\text{powersupply}}}$$

$$= \frac{MTTF^{2}_{\text{powersupply}}}{2MTTR_{\text{powersupply}}}$$

$$= \frac{200,000 \times 200,000}{2 \times 24} = 830,000,000$$

(About 4150 times more reliable than a single power supply)

Fallacies

- Fallacy: The rated mean time to failure of disks is 1200000 hours or almost 140 years, so disks practically never fail.
  - Misleading information
  - How is the MTTF calculated? Put thousands of disks in a room, run them for a few months, and count the number that fail.
  - MTTF = total number of hours the disks worked cumulatively/the number of that failed
  - This number far exceeds the lifetime of a disk (commonly 5 years or 43800 hours)
  - For the MTTF to make sense, a user should keep replacing the disk every 5 years. (replace a disk 27 times before a failure)
  - Percentage of disks that fail in a year:

  Failed disks = Number of disks x Time period / MTTF

  = 1000 disks x 8760 hours /1000000 = 9

  i.e. 9/1000 = 0.9 % would fail per year

Performance

- **X is n times faster than Y**

$$n = \frac{\text{ExTime}_Y}{\text{ExTime}_X} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

- **The throughput of X is 1.3 times higher than Y**

The number of tasks completed per unit time on computer X is 1.3 times the number completed on Y

Evaluation of Performance - Benchmarks

- Real Programs
  - Including input, output, options in UI
- Kernels
  - Small, key pieces from real programs
- Toy Benchmarks
  - Small, easy to type, and run on almost any computer. E.g. sorting
- Synthetic Benchmarks
  - Created to match an average execution profile

Evaluation of Performance - Benchmarks

- Compiler writer and architect can conspire to make the computer appear faster on these benchmarks
- Conditions under which benchmarks are run also mater: require the vendor to use one compiler and one set of flags for all programs in the same language

Benchmarks

- SPEC (System Performance Evaluation Cooperative) has more than just CPU benchmarks. Check out http://www.spec.org
  - Graphics/Applications
  - High-performance computing (HPC) / OpenMP
  - Java Client/Server
  - Mail Servers
  - Network File System
  - Web Servers
- Other important benchmarks
  - TPC (Transaction Processing Performance Council)
    - Commercial apps
    - http://www.tpc.org
  - EEMBC (Embedded Microprocessor Benchmark Consortium)
    - Embedded apps
    - http://www.eembc.com
  - NAS Parallel Benchmark (NASA Advanced Supercomputing)
    - Parallel apps
    - http://www.nas.nasa.gov/

SPEC 2006 Programs

Re-producibility

SPEC2006 benchmark description	Benchmark name by SPEC generation				
	SPEC2006	SPEC2000	SPEC95	SPEC92	SPEC89
GNU C compiler					gcc
Interpreted string processing			perl		espresso
Combinatorial optimization		mcf			li
Block-sorting compression		bzip2		compress	eqntott
Go game (AI)	go	vortex	go	sc	
Video compression	h264avc	gzip	ijpeg		
Games/path finding	astar	eon	m88ksim		
Search gene sequence	hmmer	twolf			
Quantum computer simulation	libquantum	vortex			
Discrete event simulation library	omnetpp	vpr			
Chess game (AI)	sjeng	crafty			
XML parsing	xalancbmk	parser			
CFD/blast waves	bwaves				fpppp
Numerical relativity	cactusADM				tomcatv
Finite element code	calculix				doduc
Differential equation solver framework	dealII				nasa7
Quantum chemistry	gamess				spice
EM solver (freq/time domain)	GemsFDTD			swim	matrix300
Scalable molecular dynamics (~NAMD)	gromacs		apsi	hydro2d	
Lattice Boltzmann method (fluid/air flow)	lbm		mgrid	su2cor	
Large eddie simulation/turbulent CFD	LESlie3d	wupwise	applu	wave5	
Lattice quantum chromodynamics	milc	apply	turb3d		
Molecular dynamics	namd	galgel			
Image ray tracing	povray	mesa			
Spare linear algebra	soplex	art			
Speech recognition	sphinx3	equake			
Quantum chemistry/object oriented	tonto	facerec			
Weather research and forecasting	wrf	ammp			
Magneto hydrodynamics (astrophysics)	zeusmp	lucas			
		fma3d			
		sixtrack			

How to Summarize Suite Performance

- Arithmetic average of execution time of all programs?
  - Some SPEC programs take four times longer than others
  - So these programs would become much more important than others in arithmetic average

- Could add a weight per program, but how to select the weights?
  - Different companies want different weights for their products
  - Could be hard to reach consensus

How to Summarize Suite Performance

- Idea: use weights that make all programs execute an equal time on some reference computer
  - But we do not want the results be biased to the performance of the reference computer

- Normalize execution times to a reference computer by dividing the time on the reference computer by the time on the computer being rated SPECRatio

SPECRatio

- Normalize execution times to reference computer, yielding a ratio proportional to performance =

$$\frac{\text{time on reference computer}}{\text{time on computer being rated}}$$

How to Summarize Suite Performance

- If program SPECRatio on Computer A is 1.25 times bigger than Computer B, then

$$1.25 = \frac{SPECRatio_A}{SPECRatio_B}$$

$$= \frac{\dfrac{ExecutionTime_{reference}}{ExecutionTime_A}}{\dfrac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

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

- Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant

How to Summarize Suite Performance

- For ratios, proper mean is geometric mean (SPECRatio unit-less, so arithmetic mean meaningless)

$$Geometric\ Mean = \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$$

Ratio of geometric means
= Geometric mean of performance ratios
$\Rightarrow$ choice of reference computer is irrelevant.

- Geometric mean of ratios is chosen for summarizing performance.

The ratio of the geometric means is equal to the geometric mean of the performance ratios

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

$$= \sqrt[n]{\prod_{i=1}^{n} \frac{SPECRatioA_i}{SPECRatioB_i}} = \sqrt[n]{\prod_{i=1}^{n} \frac{ExeTime_{reference_i} / ExeTimeA_i}{ExeTime_{reference_i} / ExeTimeB_i}}$$

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

Quantitative Principles of Computer Design

- Take Advantage of Parallelism
- Principle of Locality
- Make the Common Case Fast
  - Amdahl’s Law
  - The CPU Performance Equation

Amdahl's Law

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

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

Best you could ever hope to do:

- How to distribute resources to improve cost-performance
- Comparing the overall system performance of two alternatives/ two processor design alternatives

Amdahl's Law Example

- Enhancement A
  - Floating point instructions are improved by a factor of 2
  - Suppose 50% of actual instruction execution time is contributed by FP.
- Enhancement B
  - Floating point square root (FPSQR) instructions are improved by a factor of 10.
  - Suppose FPSQR is responsible for 20% of execution time
- Which enhancement is better

Amdahl's Law Example

- Floating point instructions improved to run 2X; but only 50% of actual instruction execution time is contributed by FP. What is the overall speedup?

Amdahl's Law Example
Enhancement B

- Floating point square root (FPSQR) instructions are improved by a factor of 10.
- Suppose FPSQR is responsible for 20% of execution time. What is the overall speedup?

Amdahl's Law can be applicable beyond performance

- Example of Amdahl's law applied on reliability
- Assume a disk subsystem with the following components and MTTF

  5 disks, each 1000000-hour MTTF
  1 SCSI controller, 500000-hour MTTF
  1 power supply, and 1 redundant power supply,
  each 200000-hour MTTF
  1 fan, 500000-hour MTTF
  1 SCSI cable, 500000-hour MTTF

- Assume it takes averaged 20 hours to notice a failed power supply and repair it
- What is the overall enhancement for redundant power supply

- Method 1: Without applying Amdahl's Law
- Compute the MTTF for power supply pair first

- Failure rate_system:

- MTTF after redundant power supply

$$MTTF = 1/FailureRate =$$

- Failure rate before redundant power supply

$$FailureRate$$

$$=$$

$$MTTF = 1/FailureRate =$$

- Enhancement=?

Method 2: Applying Amdahl's Law

- power supply pair enhancement

- Power supply contributes to a fraction of 5/16

The CPU Performance Equation

- **Instruction Count (IC)**
- **Clock cycles Per Instruction (CPI)**

  CPI = CPU clock cycles for a program / Instruction Count

- **CPU time**

$$CPU\ time$$

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

$$= CPU\ clock\ cycles\ for\ a\ program / clock\ rate$$

$$= IC \times CPI \times Clock\ cycle\ time = IC \times CPI\ /\ clock\ rate$$

The processor Performance Equation

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

Instruction Count

CPI

Clock Cycle Time = 1/Clock Rate

Aspects of CPU Performance

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

Example: Calculating CPI

Op	Freq	Cycles	CPI (i)	(% Time)
ALU	50%	1	.5	(33%)
Load	20%	2	.4	(27%)
Store	10%	2	.2	(13%)
Branch	20%	2	.4	(27%)
			1.5	

Typical Mix

Example

- Frequency of FP operations = 25%
- Average CPI of FP operations = 4
- Average CPI of other instructions = 1.33
- Frequency of FPSQR = 2%
- CPI of FPSQR = 20
- Alternative 1: reduce the CPI of FPSQR to 2
- Alternative 2: reduce the average CPI of all FP to 2

- Which alternative is better?
- What is the performance speedup of each alternative?

Answer of the Example on previous page

- Before Enhancement

$$CPI_{ori} = \sum_{i=1}^{n} CPI_i \times (IC_i / InstructionCount)$$

$$= 4 \times 25\% + 1.33 \times 75\% = 2.0$$

- Alternative 1: $CPI_1 = CPI_{ori} - 2\% \times (CPI_{OldFPSQR} - CPI_{NewFPSQR})$

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

- Alternative 2: $CPI_2 = 75\% \times 1.33 + 25\% \times 2.0 = 1.5$

- Alternative 2 is better

- Speedup for alternative 1

$$\frac{CPU\_Time_{ori}}{CPU\_Time_2} = \frac{IC \times Clock\ Cycle \times CPI_{ori}}{IC \times Clock\ Cycle \times CPI_1} = \frac{2}{1.64} = 1.22$$

- Speedup for alternative 2

$$\frac{CPU\_Time_{ori}}{CPU\_Time_2} = \frac{IC \times Clock\ Cycle \times CPI_{ori}}{IC \times Clock\ Cycle \times CPI_2} = \frac{2}{1.5} = 1.33$$

Why can't we use the following equations:

Alternative 1 speedup = $$\frac{1}{(1-0.02)+\frac{0.02}{10}}=1.0183$$

Alternative 2 speedup = $$\frac{1}{(1-0.25)+\frac{0.25}{2}}=1.1428$$

We got different answers? What's wrong with it?