ADVANCED COMPUTER ARCHITECTURE

1Q. Explain the fundamentals of computer Design

1.1 Fundamentals of Computer Design

Computer technology has made incredible progress in the roughly from last 55 years. This rapid rate of improvement has come both from advances in the technology used to build computers and from innovation in computer design. During the first 25 years of electronic computers, both forces made a major contribution; but beginning in about 1970, computer designers became largely dependent upon integrated circuit technology. During the 1970s, performance continued to improve at about 25% to 30% per year for the mainframes and minicomputers that dominated the industry.

The late 1970s after invention of microprocessor the growth roughly increased 35% per year in performance. This growth rate, combined with the cost advantages of a mass-produced microprocessor, led to an increasing fraction of the computer business. In addition, two significant changes are observed in computer industry.

- First, the virtual elimination of assembly language programming reduced the need for object-code compatibility.
- Second, the creation of standardized, vendor-independent operating systems, such as UNIX and its clone, Linux, lowered the cost and risk of bringing out a new architecture.

These changes made it possible to successfully develop a new set of architectures, called RISC (Reduced Instruction Set Computer) architectures. In the early 1980s. The RISC-based machines focused the attention of designers on two critical performance techniques, the exploitation of instruction-level parallelism and the use of caches. The combination of architectural and organizational enhancements has led to 20 years of sustained growth in performance at an annual rate of over 50%. Figure 1.1 shows the effect of this difference in performance growth rates.

The effect of this dramatic growth rate has been twofold.

- First, it has significantly enhanced the capability available to computer users. For many applications, the highest performance microprocessors of today outperform the supercomputer of less than 10 years ago.
- Second, this dramatic rate of improvement has led to the dominance of micro-

processor-based computers across the entire range of the computer design.

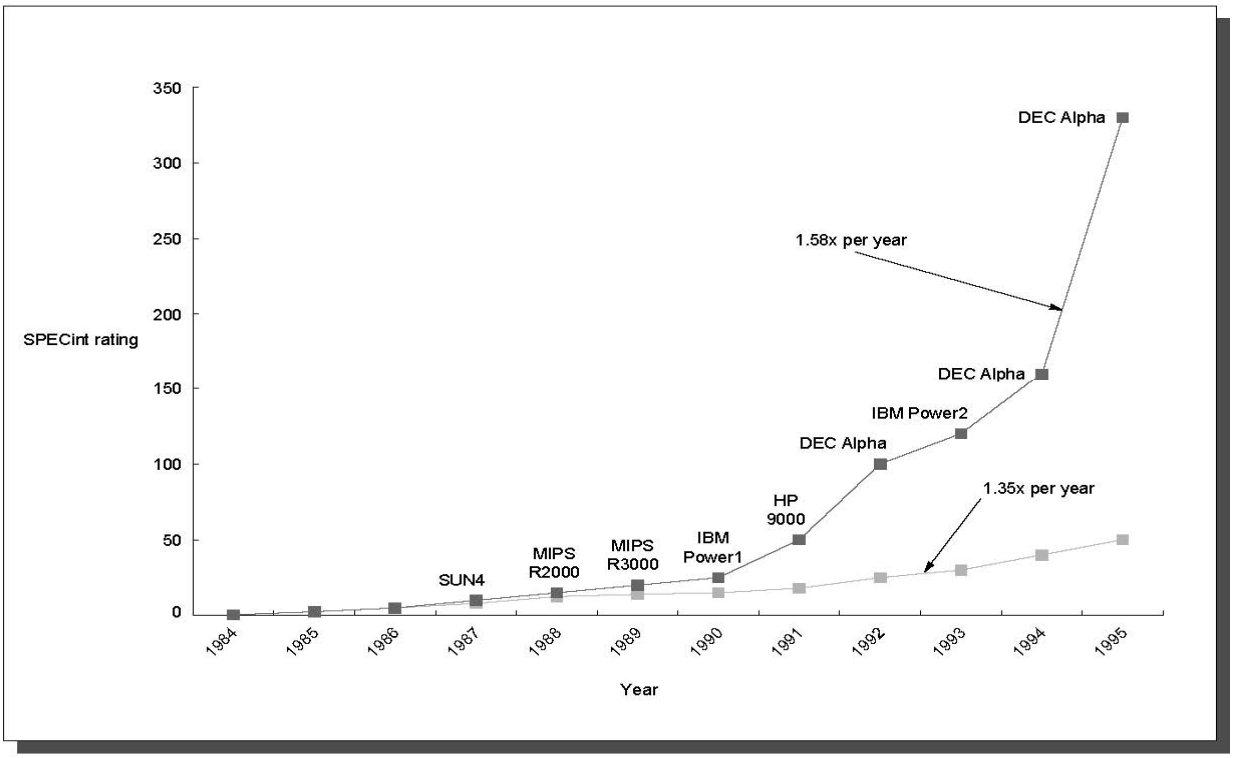


FIGURE 1.1 Growth in microprocessor performance since the mid 1980s has been substantially higher than in earlier years as shown by plotting SPECint performance.

2Q. Explain various Technology Trends in Computer Industry
1.3 Technology Trends

The changes in the computer applications space over the last decade have dramatically changed the metrics. Desktop computers remain focused on optimizing cost-performance as measured by a single user, servers focus on availability, scalability, and throughput cost-performance, and embedded computers are driven by price and often power issues.
If an instruction set architecture is to be successful, it must be designed to survive rapid changes in computer technology. An architect must plan for technology changes that can increase the lifetime of a computer.

The following Four implementation technologies changed the computer industry:

Integrated circuit logic technology—Transistor density increases by about 35% per year, and die size increases 10% to 20% per year. The combined effect is a growth rate in transistor count on a chip of about 55% per year.

Semiconductor DRAM : Density increases by between 40% and 60% per year and Cycle time has improved very slowly, decreasing by about one-third in 10 years. Bandwidth per chip increases about twice as fast as latency decreases. In addition, changes to the DRAM interface have also improved the bandwidth.

Magnetic disk technology: it is improving more than 100% per year. Prior to 1990, density increased by about 30% per year, doubling in three years. It appears that disk technology will continue the faster density growth rate for some time to come. Access time has improved by one-third in 10 years.

Network technology—Network performance depends both on the performance of switches and on the performance of the transmission system, both latency and bandwidth can be improved, though recently bandwidth has been the primary focus. For many years, networking technology appeared to improve slowly: for example, it took about 10 years for Ethernet technology to move from 10 Mb to 100 Mb. The increased importance of networking has led to a faster rate of progress with 1 Gb Ethernet becoming available about five years after 100 Mb.

These rapidly changing technologies impact the design of a microprocessor that may, with speed and technology enhancements, have a lifetime of five or more years.

Scaling of Transistor Performance, Wires, and Power in Integrated Circuits

Integrated circuit processes are characterized by the *feature size*, which is decreased from 10 microns in 1971 to 0.18 microns in 2001. Since a transistor is a 2-dimensional object, the density of transistors increases quadratically with a linear decrease in feature size. The increase in transistor performance, this combination of scaling factors leads to a complex interrelationship between transistor performance and process feature size.

First approximation, transistor performance improves linearly with decreasing feature size.

In the early days of microprocessors, the higher rate of improvement in density was used to quickly move from 4-bit, to 8bit, to 16-bit, to 32-bit microprocessors. More recently, density improvements have supported the introduction of 64-bit microprocessors as well as many of the innovations in pipelining and caches.

The signal delay for a wire increases in proportion to the product of its resistance and capacitance. As feature size shrinks wires get shorter, but the resistance and capacitance per unit length gets worse. Since both resistance and capacitance depend on detailed aspects of the process, the geometry of a wire, the loading on a wire, and even the adjacency to other structures. In the past few years, wire delay has become a major design limitation for large integrated circuits and is often more critical than transistor switching delay. Larger and larger fractions of the clock cycle have been consumed by the propagation delay of signals on wires. In 2001, the Pentium 4 broke new ground by allocating two stages of its 20+ stage pipeline just for propagating signals across the chip.

Power also provides challenges as devices are scaled. For modern CMOS microprocessors, the dominant energy consumption is in switching transistors. The energy required per transistor is proportional to the product of the load capacitance of the transistor, the frequency of switching, and the square of the voltage. As we move from one process to the next, the increase in the number of transistors switching and the frequency with which they switch, dominates the decrease in load capacitance and voltage, leading to an overall growth in power consumption.

3Q. What is Cost and Price. Explain the impact of Time, volume and commodification on Cost and Price

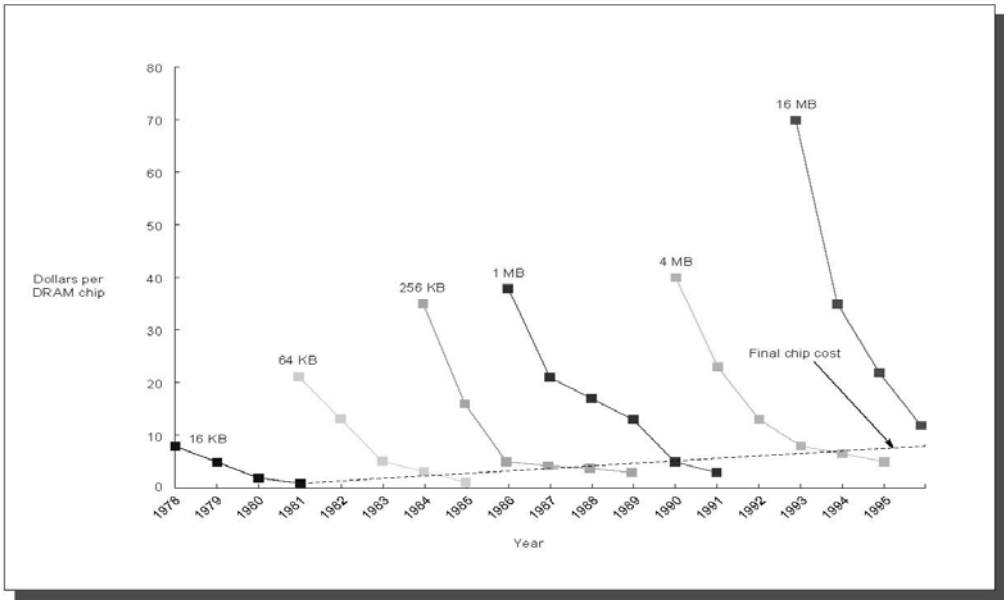
1.4 Cost, Price and their Trends

In the past 15 years, the use of technology improvements to achieve lower cost, as well as increased performance, has been a major theme in the computer industry.

- Price is what you sell a finished good for,
- Cost is the amount spent to produce it, including overhead.

The Impact of Time, Volume, Commodification, and Packaging

The cost of a manufactured computer component decreases over time even without major improvements in the basic implementation technology. The underlying principle that drives costs down is the *learning curve* manufacturing costs decrease over time. As an example of the learning curve in action, the price per megabyte of DRAM drops over the long term by 40% per year. Figure 1.5 plots the price of a new DRAM chip over its lifetime.



The Microprocessor prices also drop over time, but because they are less standardized than DRAMs, the relationship between price and cost is more complex. In a period of significant competition, price tends to track cost closely. Figure 1.6 shows processor price trends for the Pentium III.

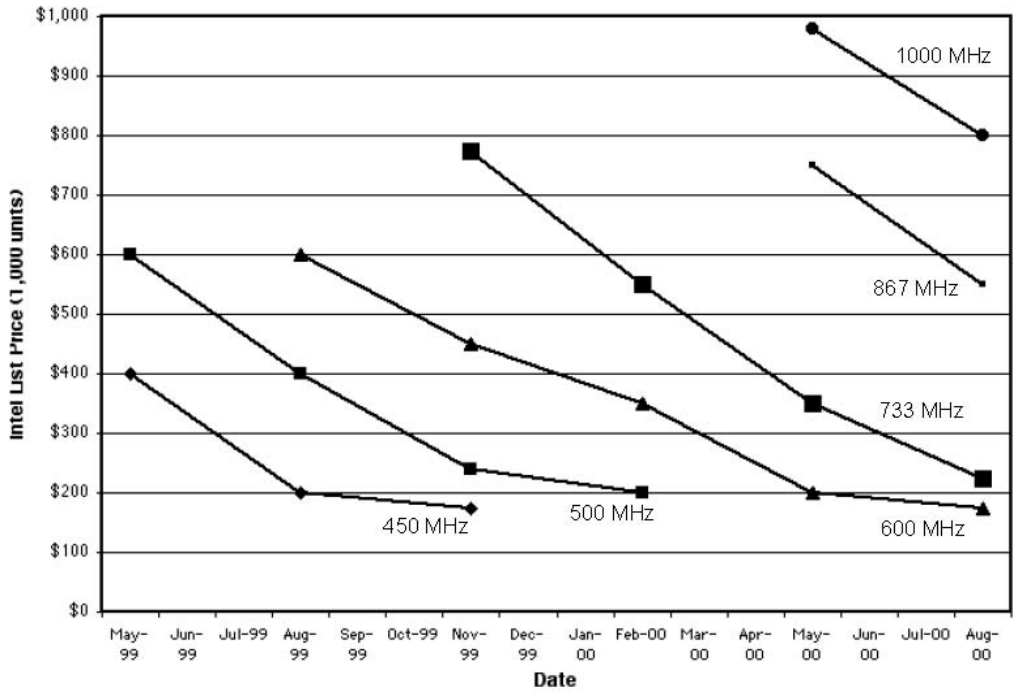


FIGURE 1.6 The price of an Intel Pentium III at a given frequency decreases over time as yield enhancements decrease the cost of good die and competition forces price reductions. Data courtesy of Microprocessor Report, May 2000 issue. The most recent introductions will continue to decrease until they reach similar prices to the lowest cost parts available today ($100-$200). Such price decreases assume a competitive environment where price decreases track cost decreases closely.

The Volume is a second key factor in determining cost. Increasing volumes affect cost in several ways.

- First, they decrease the time needed to get down the learning curve, which is partly proportional to the number of systems (or chips) manufactured.
- Second, volume decreases cost, since it increases purchasing and manufacturing efficiency.

As a rule of thumb, some designers have estimated that cost decreases about 10% for each doubling of volume.

The Commodities are products that are sold by multiple vendors in large volumes and are essentially identical. Virtually all the products sold on the shelves of grocery stores are commodities, as are standard DRAMs, disks, monitors, and keyboards. In the past 10 years, much of the low end of the computer business has become a commodity business focused on building IBM-compatible PCs. There are a variety of vendors that ship virtually identical products and are highly competitive. Of course, this competition

decreases the gap between cost and selling price, but it also decreases cost.

4. How to calculate cost of an Integrated Circuit and explain how cost becomes price by taking an example.

Cost of an Integrated Circuit:

The cost of packaged integrated circuit is

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

The number of good chips per wafer requires first learning how many dies fit on a wafer and then learning how to predict the percentage of those that will work. From there it is simple to predict cost:

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

The number of dies per wafer is basically the area of the wafer divided by the area of the die. It can be more accurately estimated by

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

The first term is the ratio of wafer area (πr^2) to die area. The second compensates for the "square peg in a round hole" problem rectangular dies near the periphery of round wafers. Dividing the circumference (πd) by the diagonal of a square die is approximately the number of dies along the edge. For example, a wafer 30 cm (≈ 12 inch) in diameter produces $\pi \times 225 - (\pi \times 30 / 1.41) = 640$ 1-cm dies.

Distribution of Cost in a System: An Example

Figure 1.9 shows the approximate cost breakdown for a $1,000 PC in 2001. Although the costs of some parts of this machine can be expected to drop over time, other components, such as the packaging and power supply, have little room for improvement.

System	Subsystem	Fraction of total
Cabinet	Sheet metal, plastic	2%
	Power supply, fans	2%
	Cables, nuts, bolts	1%
	Shipping box, manuals	1%
	Subtotal	**6%**
Processor board	Processor	23%
	DRAM (128 MB)	5%
	Video card	5%
	Motherboard with basic I/O support, and networking	5%
	Subtotal	**38%**
I/O devices	Keyboard and mouse Monitor	3% 20%
	Hard disk (20 GB) DVD drive	9% 6%
	Subtotal	**37%**
Software	OS + Basic Office Suite	**20%**

Cost Versus Price—Why They Differ and By How Much

Cost goes through a number of changes before it becomes price, and the computer designer should understand how a design decision will affect the potential selling price. For example, changing cost by $1000 may change price by $3000 to $4000.

The relationship between price and volume can increase the impact of changes in cost, especially at the low end of the market. Typically, fewer computers are sold as the price increases. Furthermore, as volume decreases, costs rise, leading to further increases in price.

Direct costs refer to the costs directly related to making a product. These include labor costs, purchasing components, scrap (the leftover from yield), and warranty. Direct cost typically adds 10% to 30% to component cost.

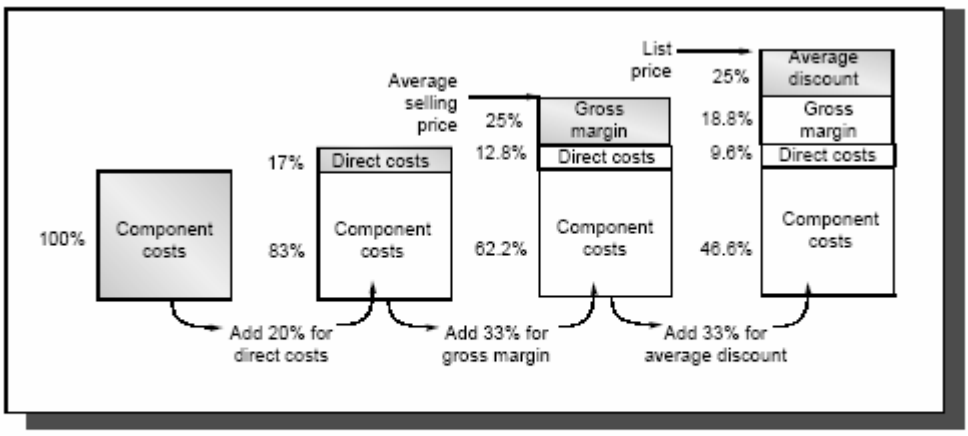


FIGURE 1.10 The components of price for a $1,000 PC. Each increase is shown along the bottom as a tax on the prior price. The percentages of the new price for all elements are shown on the left of each column.

The next addition is called the *gross margin*, the company's overhead that cannot be billed directly to one product. This can be thought of as indirect cost. It includes the company's research and development (R&D), marketing, sales, manufacturing equipment maintenance, building rental, cost of financing, pretax profits, and taxes. When the component costs are added to the direct cost and gross margin,

Average selling price is the money that comes directly to the company for each product sold. The gross margin is typically 10% to 45% of the average selling price, depending on the uniqueness of the product. Manufacturers of low-end PCs have lower gross margins for several reasons. First, their R&D expenses are lower. Second, their cost of sales is lower, since they use indirect distribution by mail, the Internet, phone order, or

retail store) rather than salespeople. Third, because their products are less unique, competition is more intense, thus forcing lower prices and often lower profits, which in turn lead to a lower gross margin.

List price and average selling price are not the same. One reason for this is that companies offer volume discounts, lowering the average selling price. As personal computers became commodity products, the retail mark-ups have dropped significantly, so list price and average selling price have closed.

5Q. How to measure and report the performance of the systems

1.5 Measuring and Reporting Performance

The computer user is interested in reducing *response time(* the time between the start and the completion of an event) also referred to as *execution time*. The manager of a large data processing center may be interested in increasing *throughput(* the total amount of work done in a given time).

In comparing design alternatives, we often want to relate the performance of two different machines, say X and Y. The phrase "X is faster than Y" is used here to mean that the response time or execution time is lower on X than on Y for the given task. In particular, "X is *n* times faster than Y" will mean

$$\frac{ExecutionTime_y}{ExecutionTime_x} = n$$

Since execution time is the reciprocal of performance, the following relationship holds:

$$n = \frac{ExecutionTime_y}{ExecutionTime_x} = \frac{\frac{1}{Performance_y}}{\frac{1}{Performance_x}} = \frac{Performance_x}{Performance_y}$$

The phrase "the throughput of X is 1.3 times higher than Y" signifies here that the number of tasks completed per unit time on machine X is 1.3 times the number completed on Y.

Even execution time can be defined in different ways depending on what we count. The most straightforward definition of time is called *wall-clock time*, *response time*, or *elapsed time*, which is the latency to complete a task, including disk accesses, memory accesses, input/output activities, operating system overhead

Choosing Programs to Evaluate Performance

A computer user who runs the same programs day in and day out would be the perfect candidate to evaluate a new computer. To evaluate a new system the user would simply compare the execution time of her *workload*—the mixture of programs and operating system commands that users run on a machine.

There are five levels of programs used in such circumstances, listed below in decreasing order of accuracy of prediction.

1. *Real applications*— Although the buyer may not know what fraction of time is spent on these programs, she knows that some users will run them to solve real problems. Examples are compilers for C, text-processing software like Word, and other applications like Photoshop. Real applications have input, output, and options that a user can select when running the program. There is one major downside to using real applications as benchmarks: Real applications often encounter portability problems arising from dependences on the operating system or compiler. Enhancing portability often means modifying the source and sometimes eliminating some important activity, such as interactive graphics, which tends to be more system-dependent.

2. *Modified (or scripted) applications*—In many cases, real applications are used as the building block for a benchmark either with modifications to the application or with a script that acts as stimulus to the application. Applications are modified for two primary reasons: to enhance portability or to focus on one particular aspect of system performance. For example, to create a CPU-oriented benchmark, I/O may be removed or restructured to minimize its impact on execution time. Scripts are used to reproduce interactive behavior, which might occur on a desktop system, or to simulate complex multiuser interaction, which occurs in a server system.

3. *Kernels*—Several attempts have been made to extract small, key pieces from real programs and use them to evaluate performance. Livermore Loops and Linpack are the best known examples. Unlike real programs, no user would run kernel programs, for they exist solely to evaluate performance. Kernels are best used to isolate performance of individual features of a machine to explain the reasons for differences in performance of real programs.

4. *Toy benchmarks*—Toy benchmarks are typically between 10 and 100 lines of code and produce a result the user already knows before running the toy program. Programs like Sieve of Eratosthenes, Puzzle, and Quicksort are popular because they are small, easy to type, and run on almost any computer. The best use of such programs is beginning programming assignments.

5. *Synthetic benchmarks*—Similar in philosophy to kernels, synthetic benchmarks try to match the average frequency of operations and operands of a large set of programs. Whetstone and Dhrystone are the most popular synthetic benchmarks.

6Q. What is Benchmark. Explain various Benchmark suites.

Benchmark Suites

Recently, it has become popular to put together collections of benchmarks to try to measure the performance of processors with a variety of applications. One of the most successful attempts to create standardized benchmark application suites has been the SPEC (Standard Performance Evaluation Corporation), which had its roots in the late 1980s efforts to deliver better benchmarks for workstations. Just as the computer industry has evolved over time, so has the need for different benchmark suites, and there are now SPEC benchmarks to cover different application classes, as well as other suites based on the SPEC model. Which is shown in figure 1.11

Benchmark Name	Benchmark description
Business Winstone 99	Runs a script consisting of Netscape Navigator, and several office suite products (Microsoft, Corel, WordPerfect). The script simulates a user switching among and running different applications.
High-end Winstone 99	Also simulates multiple applications running simultaneously, but focuses on compute intensive applications such as Adobe Photoshop.
CC Winstone 99	Simulates multiple applications focused on content creation, such as Photoshop, Premiere, Navigator, and various audio editing programs.
Winbench 99	Runs a variety of scripts that test CPU performance, video system performance, disk performance using kernels focused on each subsystem.

Desktop Benchmarks

Desktop benchmarks divide into two broad classes: CPU intensive benchmarks and graphics intensive benchmarks intensive CPU activity). SPEC originally created a benchmark set focusing on CPU performance (initially called SPEC89), which has evolved into its fourth generation: SPEC CPU2000, which follows SPEC95, and SPEC92. (Figure 1.30 on page 64 discusses the evolution of the benchmarks.) SPEC CPU2000, summarized in Figure 1.12, consists of a set of eleven integer benchmarks (CINT2000) and fourteen floating point benchmarks (CFP2000).

Benchmark	Type	Source	Description
gzip	Integer	C	Compression using the Lempel-Ziv algorithm
vpr	Integer	C	FPGA circuit placement and routing
gcc	Integer	C	Consists of the GNU C compiler generating optimized machine code.
mcf	Integer	C	Combinatorial optimization of public transit scheduling
crafty	Integer	C	Chess playing program.
parser	Integer	C	Syntactic English language parser
eon	Integer	C++	Graphics visualization using probabilistic ray tracing
perlbmk	Integer	C	Perl (an interpreted string processing language) with four input scripts
gap	Integer	C	A group theory application package
vortex	Integer	C	An object-oriented database system
bzip2	Integer	C	A block sorting compression algorithm.
twolf	Integer	C	Timberwolf: a simulated annealing algorithm for VLSI place and route
wupwise	FP	F77	Lattice gauge theory model of quantum chromodynamics.
swim	FP	F77	Solves shallow water equations using finite difference equations.
mgrid	FP	F77	Multigrid solver over 3-dimensional field.
apply	FP	F77	Parabolic and elliptic partial differential equation solver
mesa	FP	C	Three dimensional graphics library.
galgel	FP	F90	Computational fluid dynamics.
art	FP	C	Image recognition of a thermal image using neural networks
equake	FP	C	Simulation of seismic wave propagation.
facerec	FP	C	Face recognition using wavelets and graph matching.
ammp	FP	C	molecular dynamics simulation of a protein in water
lucas	FP	F90	Performs primality testing for Mersenne primes
fma3d	FP	F90	Finite element modeling of crash simulation
sixtrack	FP	F77	High energy physics accelerator design simulation.
apsi	FP	F77	A meteorological simulation of pollution distribution.

Although SPEC CPU2000 is aimed at CPU performance, two different types of graphics benchmarks were created by SPEC: SPEC viewperf is used for benchmarking systems supporting the OpenGL graphics library, while SPECapc consists of applications that make extensive use of graphics. SPECviewperf measures the 3D rendering performance of systems running under OpenGL using a 3-D model and a series of OpenGL calls that transform the model. SPECapc consists of runs of three large applications:

1. Pro/Engineer: a solid modeling application that does extensive 3-D rendering. The input script is a model of a photocopying machine consisting of 370,000 triangles.
2. SolidWorks 99: a 3-D CAD/CAM design tool running a series of five tests varying from I/O intensive to CPU intensive. The largetest input is a model of an assembly line consisting of 276,000 triangles.
3. Unigraphics V15: The benchmark is based on an aircraft model and covers a wide spectrum of Unigraphics functionality, including assembly, drafting, numeric control machining, solid modeling, and optimization. The inputs are all part of an aircraft design.

Server Benchmarks

Just as servers have multiple functions, so there are multiple types of benchmarks. The simplest benchmark is perhaps a CPU throughput oriented benchmark. SPEC CPU2000 uses the SPEC CPU benchmarks to construct a simple throughput benchmark where the processing rate of a multiprocessor can be measured by running multiple copies (usually as many as there are CPUs) of each SPEC CPU benchmark and converting the CPU time into a rate. This leads to a measurement called the SPECRate. Other than SPECRate, most server applications and benchmarks have significant I/O activity arising from either disk or network traffic, including benchmarks for file server systems, for web servers, and for database and transaction processing systems. SPEC offers both a file server benchmark (SPECSFS) and a web server benchmark (SPECWeb). SPECSFS (see http://www.spec.org/osg/sfs93/) is a benchmark for measuring NFS (Network File System) performance using a script of file server requests; it tests the performance of the I/O system (both disk and network I/O) as well as the CPU. SPECSFS is a throughput oriented benchmark but with important response time requirements.

Transaction processing benchmarks measure the ability of a system to handle transactions, which consist of database accesses and updates. All the TPC benchmarks measure performance in transactions per second. In addition, they include a response-time requirement, so that throughput performance is measured only when the response time limit is met. To model real-world systems, higher transaction rates are also associated with larger systems, both in terms of users and the data base that the transactions are applied to. Finally, the system cost for a benchmark system must also be included, allowing accurate comparisons of cost-performance.

Embedded Benchmarks

Benchmarks for embedded computing systems are in a far more nascent state than those for either desktop or server environments. In fact, many manufacturers quote Dhrystone performance, a benchmark that was criticized and given up by desktop systems more than 10 years ago! As mentioned earlier, the enormous variety in embedded applications, as well as differences in performance requirements (hard real-time, soft real-time, and overall cost-performance), make the use of a single set of benchmarks unrealistic. In practice, many designers of embedded systems devise benchmarks that reflect their application, either as kernels or as stand-alone versions of the entire application. For those embedded applications that can be characterized well by kernel performance, the best standardized set of benchmarks appears to be a new benchmark set: the EDN Embedded Microprocessor Benchmark Consortium (or EEMBC–pronounced embassy). The EEMBC benchmarks fall into five classes: automotive/industrial, consumer, networking, office automation, and telecommunications Figure 1.13 shows the five different application classes, which include 34 benchmarks.

Benchmark Type	# of this type	Example benchmarks
Automotive/industrial	16	6 microbenchmarks (arithmetic operations, pointer chasing, memory performance, matrix arithmetic, table lookup, bit manipulation), 5 automobile control benchmarks, and 5 filter or FFT benchmarks.
Consumer	5	5 multimedia benchmarks (JPEG compress/decompress, filtering, and RGB conversions).
Networking	3	Shortest path calculation, IP routing, and packet flow operations.
Office automation	4	Graphics and text benchmarks (Bezier curve calculation, dithering, image rotation, text processing).
Telecommunications	6	Filtering and DSP benchmarks (autocorrelation, FFT, decoder, and encoder)

FIGURE 1.13 The EEMBC benchmark suite, consisting of 34 kernels in five different classes. See www.eembc.org for more information on the benchmarks and for scores.

7Q. What is Amdahl's Law. Explain with example.

1.6. Quantitative Principles of Computer Design

The most important and pervasive principle of computer design is to make the common case fast In applying this simple principle, we have to decide what the frequent case is and how much performance can be improved by making that case faster. A fundamental law, called *Amdahl's Law*, can be used to quantify this principle.

Amdahl's Law

The performance gain that can be obtained by improving some portion of a computer can be calculated using Amdahl's Law. Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used. Amdahl's Law defines the *speedup* that can be gained by using a particular feature.

Speedup is the Ratio

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

Alternatively,

$$\text{Speedup} = \frac{\text{Execution Time for entire task without using enhancement}}{\text{Execution Time for entire task using enhancement when possible}}$$

Speedup tells us how much faster a task will run using the machine with the enhancement as opposed to the original machine. Amdahl's Law gives us a quick way to find the speedup from some enhancement, which depends on two factors:

1. *The fraction of the computation time in the original machine that can be converted to take advantage of the enhancement*—For example, if 20 seconds of the execution time of a program that takes 60 seconds in total can use an enhancement, the fraction is 20/60. This value, which we will call Fractionenhanced, is always less than or equal to 1.

2. *The improvement gained by the enhanced execution mode; that is, how much faster the task would run if the enhanced mode were used for the entire program*— This value is

the time of the original mode over the time of the enhanced mode: If the enhanced mode takes 2 seconds for some portion of the program that can completely use the mode, while the original mode took 5 seconds for the same portion, the improvement is 5/2. We will call this value, which is always greater than 1, Speedupenhanced.

The execution time using the original machine with the enhanced mode will be the time spent using the unenhanced portion of the machine plus the time spent using the enhancement:

$$\text{Execution time}_{\text{New}} = \text{Execution time}_{\text{old}} \left(\left(1 - Fraction_{Enhanced}\right) + \frac{Fraction_{Enhanced}}{Speedup_{Enhanced}} \right)$$

The overall speedup is the ratio of the execution times:

$$Speedup_{Overall} = \frac{Exection_{old}}{Exection_{New}} = \frac{1}{\left(\left(1 - Fraction_{Enhanced}\right) + \frac{Fraction_{Enhanced}}{Speedup_{Enhanced}} \right)}$$

Amdahl's Law can serve as a guide to how much an enhancement will improve performance and how to distribute resources to improve cost/performance.

The CPU Performance Equation

Essentially all computers are constructed using a clock running at a constant rate. These discrete time events are called *ticks*, *clock ticks, clock periods, clocks, cycles*, or *clock cycles*. Computer designers refer to the time of a clock period by its duration (e.g., 1 ns) or by its rate (e.g., 1 GHz). CPU time for a program can then be expressed two ways:

CPU Time = CPU Clock Cycles Per a Program X Clock Cycle Time

Or

$$CPU\ Time = \frac{CPU\ Clock\ Cycles\ Per\ \text{Pr}ogram}{Clock\ Rate}$$

In addition to the number of clock cycles needed to execute a program, we can also count the number of instructions executed—the *instruction path length* or *instruction count* (IC). If we know the number of clock cycles and the instruction count we can calculate the average number of *clock cycles per instruction* (CPI).

CPI is computed as

$$CPI = \frac{\text{CPU Clock Cycles Per a Program}}{\text{Instruction Count}}$$

This allows us to use CPI in the execution time formula:

CPU time = Instruction count X Clock Cycle Time X Cycles per Instruction

Principle of Locality

locality of reference means: Programs tend to reuse data and instructions they have used recently. A widely held rule of thumb is that a program spends 90% of its execution time in only 10% of the code. An implication of locality is that we can predict with reasonable accuracy what instructions and data a program will use in the near future based on its accesses in the recent past.

Locality of reference also applies to data accesses, though not as strongly as to code accesses. Two different types of locality have been observed. *Temporal locality* states that recently accessed items are likely to be accessed in the near future. *Spatial locality* says that items whose addresses are near one another tend to be referenced close together in time.

Advantage of Parallelism

Advantage of parallelism is one of the most important methods for improving performance. We give three brief examples, which are expounded on in later chapters. Our first example is the use of parallelism at the system level. To improve the throughput performance on a typical server benchmark, such as SPECWeb or TPC, multiple processors and multiple disks can be used. The workload of handling requests can then be

spread among the CPUs or disks resulting in improved throughput. This is the reason that scalability is viewed as a valuable asset for server applications. At the level of an individual processor, taking advantage of parallelism among instructions is critical to achieving high performance. This can be done to do this is through pipelining. The basic idea behind pipelining is to overlap the execution of instructions, so as to reduce the total time to complete a sequence of instructions. Viewed from the perspective of the CPU performance equation, we can think of pipelining as reducing the CPI by allowing instructions that take multiple cycles to overlap. A key insight that allows pipelining to work is that not every instruction depends on its immediate predecessor, and thus, executing the instructions completely or partially in parallel may be possible.

Parallelism can also be exploited at the level of detailed digital design. For example set associative caches use multiple banks of memory that are typical searched in parallel to find a desired item. Modern ALUs use carry-lookahead, which uses parallelism to speed the process of computing sums from linear in the number of bits in the operands to logarithmic.