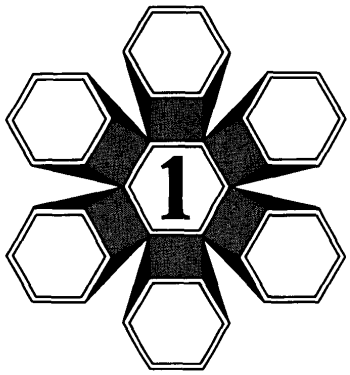
Homayoun

Reference 31



Introduction

This text is devoted to the study of the architecture of high-speed computer systems, with emphasis on design and analysis. We view a computer system as being constructed from a variety of functional modules such as processors, memories, input/output channels, and switching networks. By *architecture*, we mean the structure of the modules as they are organized in a computer system. The architectural design of a computer system involves selecting various functional modules such as processors and memories and organizing them into a system by designing the interconnections that tie them together. This is analogous to the architectural design of buildings, which involves selecting materials and fitting the pieces together to form a viable structure.

1.1 Technology and Architecture

Computer architecture is driven by technology. Every year brings new devices, new functions, and new possibilities. An imaginative and effective architecture for today could be a klunker for tomorrow, and likewise, a

ridiculous proposal for today may be ideal for tomorrow. There are no absolute rules that say that one architecture is better than another.

The key to learning about computer architecture is learning how to evaluate architecture in the context of the technology available. It is as important to know if a computer system makes effective use of processor cycles, memory capacity, and input/output bandwidth as it is to know its raw computational speed. The objective is to look at both cost and performance, not performance alone, in evaluating architectures. Because of changes in technology, relative costs among modules as well as absolute costs change dramatically every few years, so the best proportion of different types of modules in a cost-effective design changes with technology.

This text takes the approach that it is methodology, not conclusions, that needs to be taught. We present a menu of possibilities, some reasonable today and some not. We show how to construct high-performance systems by making selections from the menus, and we evaluate the systems produced in terms of technology that exists in the mid-1980s. The conclusions reached by these evaluations are probably reasonable through the end of the decade, but in no way do we claim that the architectures that look strongest today will be the best in the next decade.

The methodology, however, is timeless. From time to time the computer architect needs to construct a new menu of design choices. With that menu and the design and evaluation techniques described in this text, the architect should be able to produce high-quality systems in any decade for the technology at that time.

Performance analysis should be based on the architecture of the total system. Design and analysis of high-performance systems is very complex, however, and is best approached by breaking the large system into a hierarchy of functional blocks, each with an architecture that can be analyzed in isolation. If any single function is very complicated, it too can be further refined into a collection of more primitive functions. Processor architecture, for example, involves putting together registers, arithmetic units, and control logic to create processors—the computational elements of a computer system.

An important facet of processor architecture is the design of the instruction set for the processor, and we shall learn in the course of this text that there are controversies raging today concerning whether instruction sets should be very simple or very complex. We do not settle this controversy here; there cannot be a single answer. But we do illuminate the factors that determine the answer, and in any technology an architect can measure those factors in the course of a new design.

Computer architecture is sometimes confused with the design of computer hardware. Because computer architecture deals with modules at a functional level, not exclusively at a hardware level, computer architecture

must encompass more than hardware. We can specify, for example, that a processor performs arithmetic and logic functions, and we can be reasonably sure that these functions will be built into the hardware and not require additional programming. If we specify memory management functions in the processor, the actual implementation of those functions may be some mix of hardware and software, with the exact mix depending on performance, availability of existing hardware or software components, and costs.

When very-large-scale integration (VLSI) was in its infancy, memory-management functions were implemented in software, and the processor architecture had to support such software by providing only a collection of registers for address mapping and protection. With VLSI it becomes possible to embed a greater portion of memory management in hardware. Many systems employ sophisticated algorithms in hardware for performing memory-management functions once exclusively implemented in software.

The line between hardware and software becomes somewhat fuzzy when last year's software is embedded directly in read-only memory on a memory-management chip where it is invisibly invoked by the programs being managed. Once such a chip is packaged and is then a "black box" that does memory management, the solution becomes a hardware solution. The architect who uses the chip need not provide additional software for memory management. If a chip does most, but not all, memory-management functions internally, then the architect must look into providing the missing features by incorporating software modules.

In retrospect, computer architecture makes systems from components, and the components can be hardware, software, or a mixture of both. The skill involved in architecture is to select a good collection of components and put them together so they work effectively as a total system. Later chapters show various examples of architectures, some proven successful and some proposals that might succeed.

1.2 But Is It Art?

An article in the *New York Times* in January 1985 described a discovery of an unsigned painting by de Kooning that raised a few eyebrows among art critics. Although it does not bear his signature, there was no doubt that it is his work, and it was hung in a gallery for public viewing. The piece is a bench from the outhouse of his summer beach house that de Kooning painted abstractly to give the appearance of marble. Is this piece a great work of art by a renowned master, or is it just a painted privy seat? The point is that art appreciation is based on aesthetics, for which we have no absolute measures. We have no absolute test to conclude whether the work is a masterpiece or a piece of junk. If the art world agrees that it is a masterpiece, then it is a masterpiece.

Computer architecture, too, has an aesthetic side, but it is quite different from the arts. We can evaluate the quality of an architecture in terms of maximum number of results per cycle, program and data capacity, and cost, as well as other measures that tend to be important in various contexts. We need never debate a questions such as, "but is it fast?"

Architectures can be compared on critical measures when choices must be made. The challenge comes because technology gives us new choices each year, and the decisions from last year may not hold this year. Not only must the architect understand the best decision for today, but the architect must factor in the effects of expected changes in technology over the life of a design. Therefore, not only do evaluation techniques play a crucial role in individual decisions, but by using these techniques over a period of years, the architect gains experience in understanding the impact of technological developments on new architectures and is able to judge trends for several years in the future.

Here are the principal criteria for judging an architecture:

- Performance;
- Cost; and
- Maximum program and data size.

There are a dozen or more other criteria, such as weight, power consumption, volume, and ease of programming, that may have relatively high significance in particular cases, but the three listed here are important in all applications and critical in most of them.

1.2.1 The Cost Factor

The cost criterion deserves a bit more explanation because so many people are confused about what it means. The cost of a computer system to a user is the money that the user pays for the system, namely its price. To the designer, cost is not so clearly defined. In most cases, cost is the cost of manufacturing, including a fair amortization of the cost of development and capital tools for construction. All too often we see comparisons of architectures that compare the parts cost of System A with the purchase price of System B, where System A is a novel architecture that is being proposed as an innovation, and System B represents a model in commercial production.

Another fallacious comparison is often made when relating hardware to software. In the early years of computing, software was often bundled free of charge with hardware, but, as the industry matured, software itself became a commodity of value to be sold.

We now discover that what was once a free good now commands a significant portion of a computing budget. The trends that people quote are depicted in Fig. 1.1, where we see the cost of software steadily rising with

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.