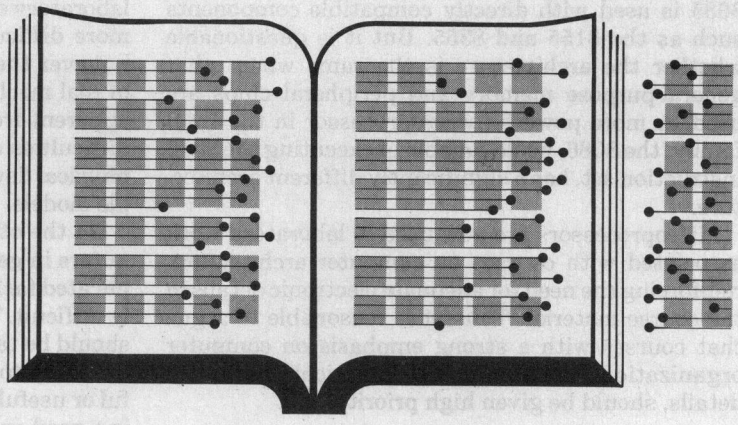
The proliferation of microprocessors has intensified the demand for graduates who have had strong course work in the practical hardware details of computer organization.

TEACHING COMPUTER STRUCTURES



Zvonko G. Vranesic
University of Toronto

Kenneth J. Thurber
Sperry Univac
University of Minnesota

Courses in the structural principles of computer hardware are an essential part of modern university computer science and electrical engineering programs. While few people would argue against the importance of this subject, there is a lack of consensus about what material should be taught, the appropriate relative emphasis of programming to electronic aspects, and the type of presentation best suited for the subject matter.

Strongly motivated by practical considerations, in this article we discuss an approach to courses on digital hardware. First, we deal with issues pertinent to courses in which the main objective is to introduce students to fundamental material. Second, we consider a more specialized level of hardware treatment, where interconnection and classification of more complex systems are highlighted.

Our opinions are largely influenced by our experience teaching students in computer science and electrical engineering at the University of Toronto and the University of Minnesota, and they reflect feedback from the industrial community that ultimately absorbs university graduates.

Computer architecture vs computer organization

The study of computer architecture is concerned with the functional structure of hardware as seen through the eyes of a programmer.[1] It involves only those aspects of a computer that have a direct impact on the logical execution of a program, not the particulars of electronic implementation that affect parameters such as the execution speed of a given machine instruction. Under the umbrella of computer architecture it is customary to consider characteristics such as word length, bus structure, number and type of registers, addressing modes, instruction set, input/output mechanism, and memory access techniques. Thus, one can speak of the IBM 370 architecture, the PDP-11 architecture, etc. In contrast, a specific implementation of a particular architecture, for example one model in the PDP-11 line, is discussed under the heading of computer organization, which involves hardware details essentially transparent to the programmer. These include the control mechanism and associated signaling, bus interfaces, and the internal structure of storage units. Moreover, computer organization involves interfacing details of related peripheral devices.

It has been a fairly common academic practice to place greater emphasis on computer architecture than on computer organization.[2-7] This unfortunate situation exists because insufficient importance is placed on teaching computer hardware. Since, in the past, few people in the computer business came into direct contact with hardware, it was easy to argue that computer specialists would benefit most from hardware exposure at the architectural level, concentrating their efforts on software aspects. Adequate treatment of computer organization tended to be presented only in engineering-oriented programs, which, on the other hand, often underplayed the importance of software.

Then microprocessors came along. Quickly it became apparent that the most useful people for microprocessor system development were neither software nor hardware specialists, but those with reasonable knowledge of both areas. At the microprocessor chip level, it is just as important to know the interfacing details as to understand the architecture.

0018-9162/80/0600-0019$00.75

Thus, the microprocessor environment warrants a shift toward computer organization in the computer curriculum.

A further point: it is hardly appropriate to talk about significant microprocessor architectures that are reflected in a number of distinct implementation models. For example, the Intel 8080 and 8085 are supposed to have the same architecture. This is so if the 8085 is used with directly compatible components such as the 8155 and 8355. But it is questionable whether the architecture is the same when other general-purpose memory and peripheral chips are used. A more powerful microprocessor in the Intel family, the 8086, while capable of executing the 8080 instruction set, has a significantly different architecture.

Microprocessors are now used in laboratory work associated with courses on computer architecture, reinforcing the need for adequate electronic details in the course material. Thus, it is reasonable to argue that courses with a strong emphasis on computer organization, incorporating practical hardware details, should be given high priority.

What should be taught as fundamental material

Much has been said and written about what should be taught in a computer hardware course.[8,9] While it is easy enough to say that the most relevant material must be covered, there is considerable disagreement as to what this material is and what is indeed fundamental. In fact, available textbooks show that, aside from a few basic topics, each author has his own ideas as to what is important and interesting.[10-15]

In the typical course, much time is spent on well-defined concepts, particularly those where some formalism can be introduced. A good example is computer arithmetic. It lends itself to neat presentation; circuitry needed to implement the desired algorithms can be described in concise and easily understood terms. Moreover, some of the useful algorithms are rather ingenious and fun to talk about, and suitable illustrative examples are readily available. Another popular topic is the internal structure of memories. There is general agreement on the relative merits of the few schemes for constructing primary memories and they provide nice material for classroom discussion.

On the other hand, it is much more difficult to adequately treat input/output techniques, interfacing problems, communications and control signaling, and the associated timing and signal level constraints. Yet, these are the main areas of concern in digital design, particularly in small machine environments. Few graduates are likely to spend much time either designing arithmetic units or having to understand their peculiarities. Rather, since the majority of students will work in software development, most of their hardware efforts will be focused on interfacing and signaling problems. Thus, although these problems often have no clear-cut best solutions, students should be presented enough examples to illustrate the alternatives. Design practitioners are the best sources of the required information.

Implementation (and use) of digital hardware can be a frustrating process, and the student should learn something about its difficulties. He can best do so in a suitable laboratory. He should also consider the problem of testing, which is often covered in the course of laboratory experiments. Instruction in these areas is more difficult in courses without laboratory work, but even then the student should learn that the world of real machines is full of problems that may not be apparent from the pages of a textbook. Since many difficulties are caused by the imperfect behavior of physical devices, it's not wise to overemphasize simple models.

On the other hand, computer hardware, and computers in general, are often treated in a far too complicated fashion. No computer concepts are inherently difficult. There are a few very useful concepts that should be taught as simply as possible, avoiding the tendency to complicate them, no matter how powerful or useful they may be. Microprogrammed control is a good example.

The descriptive approach

There are several different approaches to teaching computer organization. Instructors tend to be strongly opinionated about which is the best approach; their opinions are often influenced by the textbooks available for the course in question. In the following discussion we consider the relative merits of one approach, namely, the descriptive approach, which relies on examples closely related to real, commercially available machines. The method has a number of advantages but also some drawbacks.

Advantages. It is impossible to do justice to the subject of computer organization without the aid of good examples. They should be sufficiently detailed to ensure that students will appreciate the intricacies of function and implementation. The instructor should avoid falling into the trap of never getting past the level of grossly oversimplified block diagrams and generalities. Students are prone to assume that structures represented by block diagrams are intuitively obvious, and general comments often mean little to them. One can best avoid this undesirable situation by using meaningful examples from the real world.[16]

A major advantage of using examples derived from specific manufactured computer architectures (e.g., PDP-11, IBM 370, HP-3000) is that students become familiar with some commercially available products, clearly desirable now that practical aspects of university education are being emphasized.

Digital system design, specifically computer design, is still not a science in the traditional sense—it is an art—so optimal solutions rarely exist. The design process is not unlike most engineering design endeavors: many design alternatives are

usually possible, each presenting some positive and some negative factors. How one goes about making sensible decisions in this process is ideally what students should learn. They can do this well only by looking at real examples that incorporate typical constraints. A study of real designs leads to a better appreciation of the trade-offs involved. In addition to understanding the architectural structures, students can gain some feeling for electronic and economic aspects as well. Functionality, speed of operation (or performance), maintainability, and cost are basic criteria that can be properly discussed within the framework of a real machine.

If laboratory work is associated with the course (as it ideally should be), then it makes sense to use examples based on the computers in the laboratory. Thus, students not only become acquainted with the structural concepts but also gain some hands-on experience, which typically leads to insights into related side issues.

The alternative to using real machines as examples is to define a hypothetical machine. It is quite easy and tempting to come up with a paper design of a computer to illustrate most concepts taught in the course. Indeed, many textbooks follow this approach.[3,4,14,17,18] These hypothetical computers often have fairly extensive assembly languages. The trouble is that such machines usually have features that do not exist in the same combination in any commercial computer. The features included are often incompatible and difficult to implement, and sometimes their implementation is not economically feasible. Occasionally, these hypothetical models contain outright design errors. Similar difficulties, though less severe, may be found in hypothetical designs patterned after real computers.

Another problem is that it is impossible to get any hands-on experience with hypothetical machines. While it is possible to simulate them, this is hardly an adequate alternative.

Disadvantages. The descriptive approach presents two serious difficulties: no single computer illustrates all the desired concepts, and just about any computer has some unappealing features. In addition, if a single machine is used, students can easily leave with the impression that no other viable machines exist. If asked (as an exercise) to suggest their own designs for a simple computer, their proposals will probably resemble the example discussed in the class.

These difficulties can be overcome by drawing examples from more than one computer architecture. For instance, one might use the PDP-11 architecture as the basic framework of discussion, illustrate input/output processor schemes through the IBM 370 channels or the peripheral processing units in Control Data machines, support the presentation of stack computers with the HP-3000, and so on. If the material is not clouded with unnecessary details, students will benefit from exposure to machines they will run into after graduation. This is the approach taken in a recently published textbook.[15]

Discussion of commercially available architectures can be carried to a dangerous extreme, whereby general principles are totally masked by the details of particular examples. This typically occurs in courses based exclusively on microprocessors, especially if the same ones are also used in laboratory work. One should not forget that microprocessors and microcomputers are really just ordinary computers that happen to be manufactured on one or more LSI chips. Thus, it is much more useful to talk about general computer principles, while relying upon the laboratory to place the characteristics of the microprocessor environment into proper perspective.

Computer design is essentially an engineering endeavor. It is subject to many practical constraints, and scientifically pure goals are often economically unfeasible dreams.

Finally, we should consider some of the objections raised by academic purists. One often hears that a descriptive approach, or a case study approach as it is sometimes inappropriately called, is not scientific in nature. Indeed, some prominent computer scientists say that computer hardware design is immature as a scientific discipline. They claim that this immaturity is characterized not only by recurring errors in new designs (caused by insufficient understanding of fundamental principles and past mistakes), but also by a preference for pictorial representation over symbolic and verbal descriptions of hardware. We find it difficult to get excited about such arguments—at times they seem quite frivolous. Computer design is essentially an engineering endeavor. As such it is subject to many practical constraints, and scientifically pure goals are often economically unfeasible dreams. It is probably best to accept this reality, at least for the time being, and make sure that we teach our students what present computers are really like, since ideal machines are still a long way off.

An illustrative example. We mentioned that the descriptive approach to teaching computer organization is synonymous with doing a case study. Of course, when a real machine is used in examples, its features should not be distorted in any way. This is particularly important where characteristics that have a direct impact on software are concerned. For example, tinkering with the instruction set is not advisable since it is heartening for students to know that whatever programs they are asked to look at or write can be executed on the machine they are learning about. On the other hand, a case study, covering all the implementation details of specific hardware, can be rather cumbersome. Suitably simplified or generalized schemes derived from the chosen architecture lead to a more purposeful presentation.

Consider the diagram in Figure 1. It depicts an important facet of microprogrammed control: the way

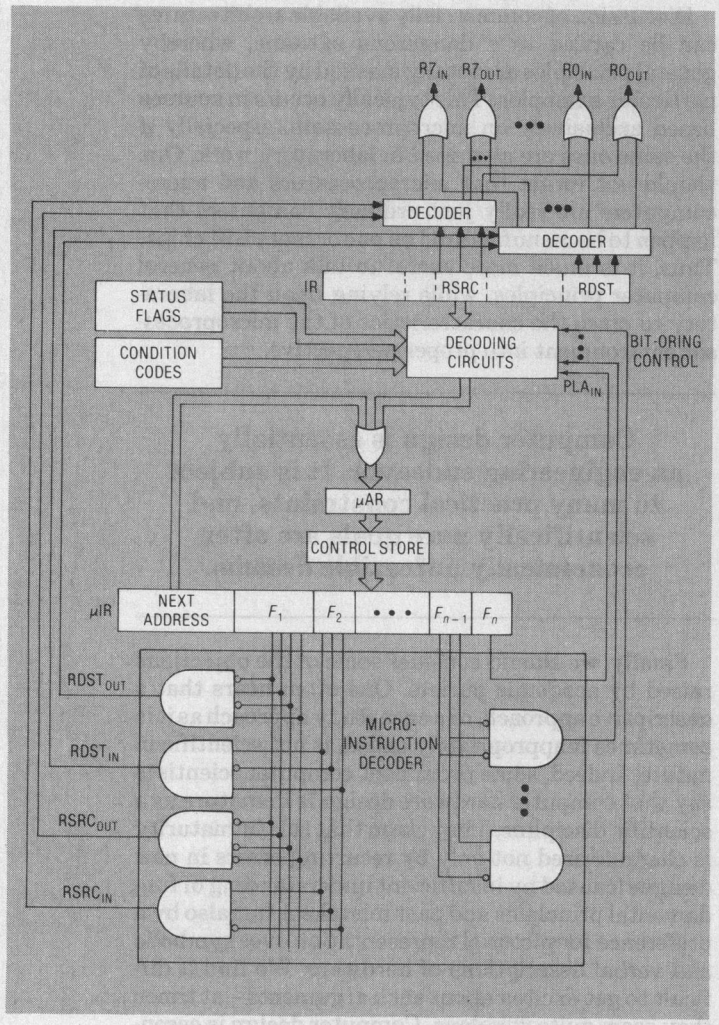


Figure 1. An illustration of the generation of control signals.

the required control signals can be generated. The scheme is patterned after the PDP-11 structure, but it is sufficiently simplified (and different from actual PDP-11 implementations) to serve as a general example. It shows how signals that control the gating of registers are obtained from the information in a microinstruction and a machine instruction held in the instruction register. It also shows a widely used method of microprogram sequencing control. A field in the microinstruction specifies the address of the next microinstruction, which may be modified through bit-ORing circuitry.

Figure 1 shows the flavor and level of presentation appropriate for instructional examples. Actually, it contains sufficient information to allow a fairly intensive general discussion of some key aspects of microprogrammed control. Such examples can be developed to demonstrate most of the relevant concepts in computer organization. The effort involved may be considerable, but it is worthwhile, judging from the positive reaction of the students who have been exposed to such material.

The second-level hardware course

Overall trends in integrated circuits and the foreseeable continuation of pragmatic engineering in the computer industry dictate our choice for the second level of discussion. Integrated circuit technology will be a primary factor in the problems students can be expected to face in the future. Clearly, the majority of students will be building systems based on microcomputers and minicomputers. Many of these systems will contain more than one processor or will have to interface with another system containing at least one processor. Thus, rather than a course exposing students to timeworn concepts such as computer arithmetic, we suggest a course (immediately after the introductory course), which discusses system concepts built upon a multiplicity of processors and then details the pragmatics of intercomputer communication. Further, we suggest that this course primarily concentrate on the difficult trade-off issues of making computers communicate. This is the approach taken by two recently published architecture books.[19,20]

System concepts. At the system level, we suggest an examination of system concepts derived from various processors. The discussion of system concepts can be based on the idea that processors with arbitrary capability levels can be used to develop the systems. The discussion of design trade-offs can concentrate on the interconnection capabilities required to make the specific systems operate. Typical system concepts that should be introduced are multiprocessors,[21] distributed computer systems,[22] and networks.[23] In particular, this course must present design trade-offs, relationships, and concrete implementation examples that show the usefulness of bus structure technology, circuit (crossbar) switch technology, and packet switch technology. Students should also learn how a processor uses any of these computer interconnection techniques for communications.

There are good abstractions of systems at this level. The Anderson/Jensen model (Figure 2) provides a comprehensive overview of abstract system types based on interconnections. We suggest that the student be given this model. Each system concept he studies should be backed up with a discussion of one of the many actual implementation examples.

Interconnection schemes. Following the study of systems, we suggest a detailed examination of how the interconnection structures can be built. Bus structures, packet switches, and circuit switches should be considered.

Students' understanding of packet switching should include the concepts of message, communication subnetwork, channel, circuit, protocol, datagram, virtual circuit, and packet. The presentation might proceed along the lines of the following discussion.

A computer communications network (Figure 3) is a collection of host computers (that provide services

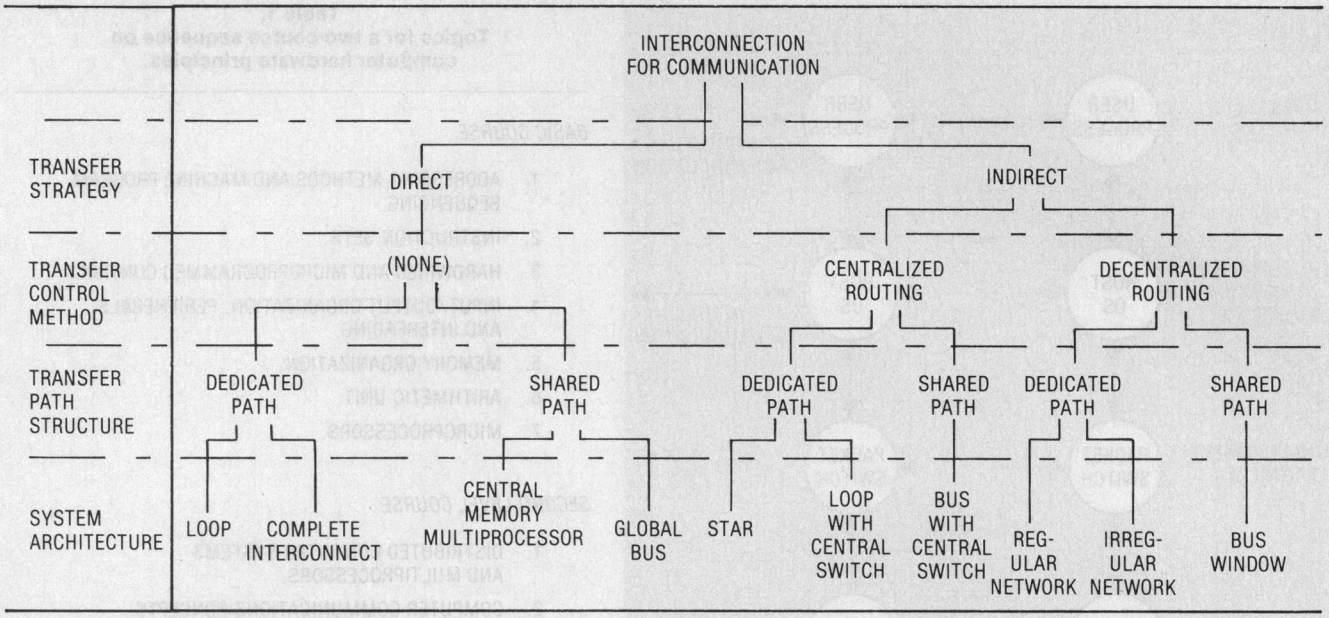


Figure 2. The Anderson/Jensen taxonomy of systems.

to either another host or an end user) and a communications subnetwork that provides communication links between hosts, users, or both.

There are three main switching techniques used in communication subnetworks:

(1) circuit switching,
(2) message switching, and
(3) packet switching.

Circuit switching will be discussed at a later time. A *message* may be defined as a logical unit of information, e.g., a telegram or data record from a data file. In message switching techniques, a message is placed in the communications subnetwork. The subnetwork establishes a channel, the message is transmitted, and the channel is released.

Packet switching concepts attempt to solve problems caused by the variable length property of a message by dividing it into packets. A *packet* can be viewed as a fixed-length subdivision of a message, prefaced with an identifier containing suitable address information and information which will allow the message to be reconstructed. Packets can be viewed as independent message subdivisions, which can make their own way through a datagram network; i.e., datagram packets can travel independently over different channels. In a virtual circuit system, packets must be delivered in order of transmission. Packet switching is important because it promises to provide effective support to many different types of message traffic simultaneously.

The communication subsystems of the packet switch with specific responsibility for handling transmission problems are usually termed *communications protocols.* For reasons very similar to

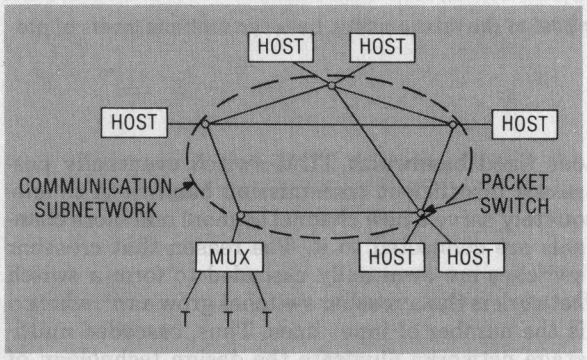


Figure 3. A model of a resource-sharing computer network.

those raised within the context of structured programming, communication protocols are almost always multilayered, and each lower layer is transparent to all higher layers. Figure 4 is a general organizational diagram of the relationships between levels of protocol. There are a number of more detailed models of protocol layers (such as the ISO reference model).[24] The course should consider various protocol and design issues at each level before going on to circuit switching.

In covering circuit switches, the instructor should describe how to build elementary circuit switches using time division multiplexing, or TDM, techniques. Second, he should describe how to build large networks composed of stages of cascaded elementary circuit switches. These elementary circuit switches could be conventional crossbar switches or TDM circuit switches. TDM switches are cascaded because

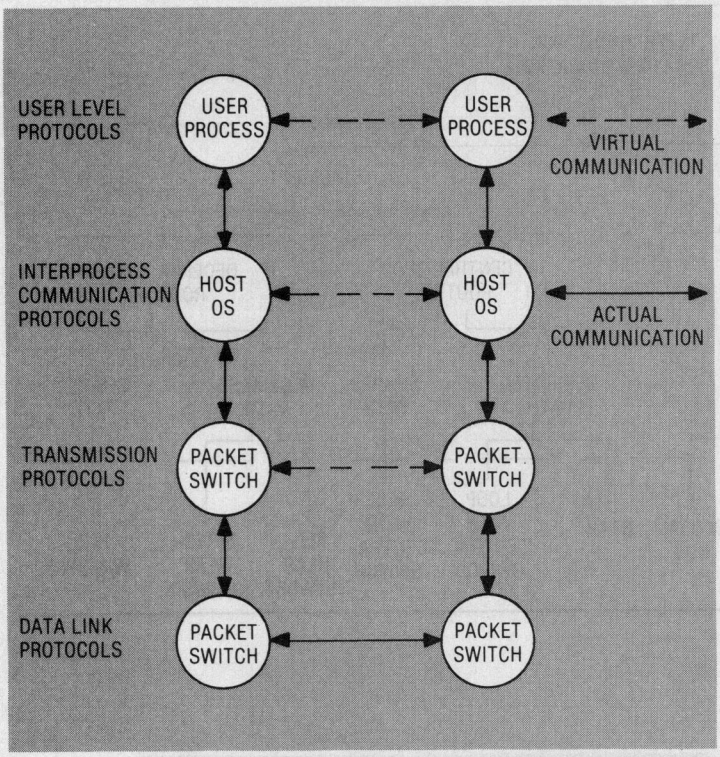


Figure 4. A model of the relationships between multiple levels of protocols.

Table 1.
Topics for a two-course sequence on computer hardware principles.

BASIC COURSE

1. ADDRESSING METHODS AND MACHINE PROGRAM SEQUENCING.
2. INSTRUCTION SETS.
3. HARDWIRED AND MICROPROGRAMMED CONTROL.
4. INPUT/OUTPUT ORGANIZATION, PERIPHERALS, AND INTERFACING.
5. MEMORY ORGANIZATION.
6. ARITHMETIC UNIT.
7. MICROPROCESSORS.

SECOND-LEVEL COURSE

1. DISTRIBUTED COMPUTER SYSTEMS AND MULTIPROCESSORS.
2. COMPUTER COMMUNICATIONS CONCEPTS.
3. MESSAGE AND PACKET SWITCHING.
4. INTERCONNECTION NETWORKS AND CIRCUIT SWITCHING TECHNIQUES.
5. TRANSMISSION TECHNIQUES AND PROTOCOLS.
6. EXAMPLES OF SYSTEMS.

one fixed bandwidth TDM switch eventually possesses insufficient transmission bandwidth to adequately serve a new channel as more and more channels are connected to it. The reason that crossbar switches are eventually cascaded to form a switch network is that crossbar switches grow as n^2, where n is the number of input lines. Thus, cascaded multistage networks illustrate the design technology of circuit switch communication.

After circuit switching, bus structures should be considered. This section of the course should address the problem of bus control and information transfer techniques on the bus. The instructor must note that the bus control techniques described could be used in other areas of the computer system, e.g., in the design of channel priority logic. Further, bus communication techniques would also be useful in I/O design, e.g., fully interlocked request/acknowledge interfaces to peripheral devices.

These two courses would leave students with a good appreciation of the fundamentals of digital hardware, while emphasizing the computer structures that are presently of greatest interest. Table 1 lists the specific topics we would include in each course. The course work should include laboratory exercises. It is most appropriate to base the laboratory on microprocessors, perhaps to the extent of providing a local network of microprocessor systems,[25] but at least providing hands-on hardware experience emphasizing detailed system interfacing design issues.[26]

The descriptive approach is by far the most practical way to teach computer architecture and organization. The structure of computers can be handled as a collection of concepts and organizational schemes illustrated with examples derived from widely accepted commercial products.

An excessively formal introduction is not desirable at this stage of students' development. Hardware description languages are suspect in their educational effectiveness. Students easily find themselves more concerned about the syntax of the language than about the structures the languages represent. Languages are more effective in documentation and simulation tasks.

Perhaps the most important question is whether or not it is possible to break down general computer structures into a set of basic primitives. So far, few results have emerged to indicate an affirmative answer. The level of primitives ought to be considerably above the level used in basic logic design and switching theory. In attempting to define this level, we should keep in mind the fate of switching theory, which in just a few years went from a field of great activity and relevance to one of little practical interest. Formal structures may find it difficult to keep up with the developments in a rapidly changing technological area.

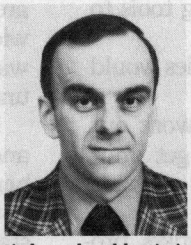
At the systems level, we must also take an innovative approach. The student should be introduced to system and intercomputer communication concepts at an early stage (senior undergraduate). Today

good taxonomies are available, allowing a comprehensive description of system concepts and communication organization. Thus, there is every reason to pursue an aggressive, forward-looking approach in computer organization courses. ■

References

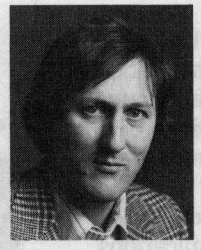
1. C. G. Bell and A. Newell, *Computer Structures: Readings and Examples,* McGraw-Hill, New York, 1971, p. 562.
2. H. Hellerman, *Digital Computer System Principles,* 2nd ed., McGraw-Hill, New York, 1974.
3. Y. Chu, *Computer Organization and Microprogramming,* Prentice-Hall, Englewood Cliffs, N.J., 1972.
4. C. C. Foster, *Computer Architecture,* Prentice-Hall, Englewood Cliffs, N.J., 1969.
5. A. S. Tanenbaum, *Structured Computer Organization,* Prentice-Hall, Englewood Cliffs, N.J., 1976.
6. H. Katzan, Jr., *Computer Systems Organization and Programming,* Science Research Associates, Chicago, 1976.
7. C. W. Gear, *Computer Organization and Programming,* 2nd ed., McGraw-Hill, New York, 1974.
8. G. E. Rossman et al., "A Course of Study in Computer Hardware Architecture," *Computer,* Vol. 8, No. 12, Dec. 1975, pp. 44-63.
9. O. Garcia, "Computer Organization and Architecture and the Laboratory Sequence," *Computer,* Vol. 10, No. 12, Dec. 1977, pp. 91-96.
10. H. W. Gschwind and E. J. McCluskey, *Design of Digital Computers,* Springer-Verlag, New York, 1975.
11. H. S. Stone and D. P. Siewiorek, *Introduction to Computer Organization and Data Structures: PDP-11 Edition,* McGraw-Hill, New York, 1975.
12. M. E. Sloan, *Computer Hardware and Organization,* Science Research Associates, Chicago, 1976.
13. M. M. Mano, *Computer System Architecture,* Prentice-Hall, Englewood Cliffs, N.J., 1976.
14. F. J. Hill and G. R. Peterson, *Digital Systems: Hardware Organization and Design,* 2nd ed., Wiley, New York, 1978.
15. V. C. Hamacher, Z. G. Vranesic, and S. G. Zaky, *Computer Organization,* McGraw-Hill, New York, 1978.
16. S. G. Zaky, Z. G. Vranesic, and V. C. Hamacher, "On the Teaching of Computer Organization to Engineering Undergraduates," *IEEE Trans. Education,* Vol. E-20, No. 1, Feb. 1977, pp. 27-30.
17. D. Lewin, *Theory and Design of Digital Computers,* Nelson, London, 1972.
18. A. M. Abd-alla and A. C Meltzer, *Principles of Digital Computer Design,* Prentice-Hall, Englewood Cliffs, N.J., 1976.
19. K. J. Thurber and G. M. Masson, *Distributed-Processor Communication Architecture,* Lexington Books (D. C. Heath), Lexington, Mass., 1979.
20. K. J. Thurber, *Tutorial: Distributed Processor Communication Architecture,* IEEE Computer Society, Long Beach, Calif., 1979.*
21. P. H. Enslow, "Multiprocessor Organization—A Survey," *Computing Surveys,* Vol. 9, No. 1, Mar. 1977, pp. 103-129.
22. G. A. Anderson and E. D. Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," *Computing Surveys,* Vol. 7, No. 4, Dec. 1975, pp. 197-213.
23. L. Kleinrock, "Principles and Lessons in Packet Communications," *Proc. IEEE,* Vol. 66, No. 11, Nov. 1978, pp. 1320-1329.
24. L. Pouzin and H. Zimmermann, "A Tutorial on Protocols," *Proc. IEEE,* Vol. 66, No. 11, Nov. 1978, pp. 1346-1370.
25. J. D. Nicoud, A. Droz, R. Forster, and D. Roux, "Software Facilities for Microprocessor Development and Teaching," in *Microprocessors and Their Applications* (J. Tiberghien et al., eds.), North-Holland Publishing Co., 1979, p. 347.
26. G. J. Lipovski, *Microcomputer Interfacing: Principles and Practices,* Lexington Books (D. C. Heath), Lexington, Mass., 1980.

*This tutorial is available from the IEEE Computer Society Publications Office, 5855 Naples Plaza, Suite 301, Long Beach, CA 90803; also available on videotape from the author.



Zvonko G. Vranesic is a professor in the Departments of Electrical Engineering and Computer Science at the University of Toronto, with research interests in computer architecture, microprocessor systems, fault-tolerant computing, and many-valued switching systems. He is coauthor of *Computer Organization* and served as guest editor of the December 1977 special section on multiple-valued logic in the *IEEE Transactions on Computers.* During the 1977-78 academic year, Vranesic was a senior visitor at the Computer Laboratory, University of Cambridge, England.

He received the BASc degree in 1963, the MASc degree in 1966, and the PhD degree in electrical engineering in 1968, all from the University of Toronto.



Kenneth J. Thurber is a senior staff scientist at Sperry Univac's Defense Systems Division. His technical interests include all areas of computer systems architecture. He is currently the chairman of the IEEE Computer Society's Technical Committee on Computer Communications.