BOOK REVIEWS

Recently published books and new periodicals may be submitted for review to the Book Reviews Editor:

Dr. Francis P. Mathur
Professor and Computer Science Coordinator
Mathematics Department
California State Polytechnic University
3801 West Temple Avenue
Pomona, CA 91768
Telephone: (714) 598-4421

(Note: publications reviewed in this section are not available from the IEEE Computer Society. Please order directly from the publisher.)

B79-2 *High-Level Language Computer Architecture*—Yaohan Chu, ed. (New York: Academic Press, 1975, $29.50, 273 pp.)*

A book's preface is always written first . . . and rewritten last. It's a good place to find a book's declared intention and, reliably enough, Chu's preface says the book "aims to fill a current need of tutorial material on high-level language computer architecture."

No reviewer can question the "current need" when the statement is made by an entrenched academician. Disregarding the issue of need, Chu's book falls short by several other measures but has some high points as well.

With any multi-author book it makes sense to look at the individual chapters independently. In this case there are seven of them, all about equal in length. The exception is Laliotis' chapter, about twice as long as the others.

Now, the blow by blow . . .

Chu's opening chapter, "Concepts of High-Level Language Computer Architecture," attempts to categorize computer architecture. He comes up with four types: vonNeumann, syntax oriented, indirect execution, and direct execution. The last he equates with high-level language architecture, although some may question the necessity for directness in the language-related architecture's finer points. This chapter is easy reading, provides a fair perspective and taxonomy for the remaining material, and keeps a low profile. After all, Chu *is* a sage in computer architecture!

The second chapter, also by Chu, is entitled "Design Concepts of Japanese-Language Data Processing Systems." Spock's eyebrow goes up on that one! Is the issue the architecture of the Japanese language, or the architecture of a machine that directly executes Japanese language programs? It seems that architectural and etymological issues are mixed here, perhaps without recovery.

The material deals primarily with translation problems for HLJL (high-level Japanese language) programs, mapped onto a stack machine format. There are examples of Japanese Algol programs (p. 25), too.

The third chapter returns to the book's central theme. Carl Carlson does a conventional but pleasingly careful and well-organized job in "A Survey of High-Level Language Computer Architecture." The treatment is even and well-referenced; there is a 90-plus item bibliography. He discusses the important historical thrusts of high-level language architecture research, and includes descriptions of all the following: the Burroughs' B-5500; processors for Algol, Fortran, Euler, and PL/I; the Adam processor; the Symbol project; the Hydra processor; Snobol and other list processors; and "other HLL computers." The material closes with an assessment of research issues and design problems.

The next chapter, Robert W. Doran's "Architecture of Stack Machines," concentrates on this genre and provides clearly written basic material on postfix representations (including their generalizations), the notions behind "context generation" (i.e., subroutine linkage), the ideas of block structuring and bounded context related to scoping problems, and some specifics on the B-5500 and HP-3000 implementations. All of this is very nicely done.

Theodore A. Laliotis' long chapter, "Architecture of the Symbol Computer System," is a joy to read. The presentation and organization are excellent throughout. The technical material is roughly divided into three parts: descriptions of the hardware proper, information about the language Symbol executes, and the I/O facilities. This chapter includes some excellent graphics; Figure 3 (p. 115) is typical and particularly impressive after it is explained by the text. The information is generally presented in depth and specifically includes many details that will be of importance in classroom use. There is also a good set of references.

Howard M. Bloom's chapter, "Conceptual Design of a Direct High-Level Language Processor," examines an Algol-60-like language and how programs written in it might be processed directly by hardware. Much of the material deals with the intricacies of understanding such programs; the detailed descriptions of language syntax and semantics reflect this orientation. Certainly one can't begin to understand how to execute such programs without understanding what they (are intended to) mean. There are several assumptions that may give computer architects some grief. For

*Review first published in *Computer Architecture News* (ACM SIGARCH newsletter), Vol. 5, No. 6, Feb. 1977, p. 29.

example, Bloom assumes all forward transfers are declared in advance of their occurrence.

The seventh chapter, "Architectural Design of an APL Processor," by Bernard J. Robinet, looks at what's involved in direct evaluation of APL programs. The very nature of APL makes this process difficult. The result seems to be a machine that does amazing things with pointers. Pointers, pointers, pointers... pointers to pointers to pointers...and so forth. Robinet deals only with a subset of APL and, like Bloom, concentrates more on syntax and semantics than on machine organization.

An overall judgment of this text is difficult—it is somewhat like judging the jury. Carlson's and Laliotis' chapters seem to carry the book for classroom use. Supplemental material would be needed to flesh out a semester's course.

For researchers, there is little not already available elsewhere, though perhaps not so compactly packaged as here. Carlson's chapter is an example of this.

For those interested in knowing what's going on but not in becoming instant experts, the book would seem a passable choice. The problem may lie in the relationship between a machine and the high-level language it is to process. It seems reasonable that one would have to be on one side of this question or the other: concerned first with language then machine, or concerned first with machine then language. Chu's book squats somewhere right in the middle!

Edward F. Miller, Jr.
San Francisco, California

B79-3 *An Introduction to Data Structures with Applications*—J. P. Tremblay and P. G. Sorenson (New York: McGraw-Hill, 1976, $18.50, 704 pp.)

A number of books on data structures have appeared in recent years. Their approaches differ, and their authors do not seem to concur on the subject matter itself. The nebulous character of the name "data structures" leaves room for debate.

This work firmly sidesteps the controversy. The authors make it clear in their succinct preface that the book follows the guidelines of the ACM's "Curriculum 68" report.[1] It is intended for Course I1 of the report. According to the report, this course is called "Data Structures" and is a prerequisite for the courses "Compiler Construction," "System Programming," "Information Organization and Retrieval," "Formal Languages and Syntactic Analysis," "Computer Graphics," and "Artificial Intelligence and Heuristic Programming." The report also suggests two introductory courses, "Computers and Programming" and "Introduction to Discrete Structures," as prerequisites to the data structures course. As a part of a computer science major program, a student would schedule it in his junior year.

The book's seven chapters cover linear data structures, nonlinear data structures, and applications, in that order. The first chapter introduces basic concepts of information content, representation, and storage. Among the types of information described, such as integers, character strings, and logical entities, a surprising inclusion is the "pointer" entity. This prepares the reader for the book's close relationship to PL/I.

This work treats character strings as a fundamental information type. Chapter 2 covers them in detail, defining and discussing string manipulation, matching, and storage. It suggests a simple (Markov) production as a primitive operation, a number of which form a Markov algorithm. Also interesting is the authors' inclusion of text editing among string manipulation applications.

Chapters 3 and 4 deal with linear data structures. Chapter 3 considers the sequential storage representation in detail, presenting arrays, stacks, and queues as examples. It includes Polish expressions and their conversion to code as an application of stacks. A simulation of a timesharing system demonstrates the use of queues. This last is too elaborate, however, and a simpler example would have done just as well. Chapter 4 explores linked storage representation of linear structures. It considers three special cases of linked lists—singly, circularly, and doubly linked. The linked dictionary is among the applications covered. Associative lists are presented as natural extensions of list structures.

Devoted to nonlinear data structures, Chapter 5 investigates trees, multilinked structures, and graphs. It defines basic concepts, and discusses the storage representation, manipulation, and application of these structures. Since nonlinear structures call for dynamic storage management schemes, the last section of the chapter discusses such schemes.

Today, no book on data structures is considered complete unless it includes a chapter on sorting and searching techniques. A neat and concise Chapter 6 meets this requirement. It falls short of Knuth's treatment,[2] but such a comparison, though inevitable, is quite unfair. After all, Knuth devotes a whole volume to the subject.

A lengthy chapter on file structures completes the authors' treatment of data structures. It includes a short description of storage devices (tapes, drums, and disks), and a somewhat detailed discussion of sequential, indexed sequential, and direct files. A short section defines virtual memory, paging, and segmentation schemes. Billing, records retrieval, and on-line banking systems are among the applications presented.

Throughout the book, the authors present algorithms using an excellent notation similar to Knuth's.[3] The notation allows for preciseness as well as descriptive statements. Although the statements in this notation resemble PL/I statements, a reader conversant with any programming language should have little difficulty in grasping the meaning.

This volume includes a number of extensively treated examples. These may be both an advantage and a drawback. The authors' desire for completeness has resulted in pictures of program listings. They should have forsaken these in favor of algorithmic notation.

The authors set out to write a textbook fitting the definition of a junior-level course on data structures. They have attained this admirably. The instructor will find the examples valuable, in spite of the lengthy program listings. He will be well advised, however, to study "Curriculum 68" to decide where his course will fit in his school's program.

Dr. Ashok Ingle
Rockwell International

References

1. "Curriculum 68," a report of the ACM Curriculum Committee on Computer Science, *CACM*, Vol. 11, No. 3, Mar. 1968, pp. 151-197.
2. D. E. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*," Addison-Wesley, Reading, Mass., 1973.
3. D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, Addison-Wesley, Reading, Mass. 1968.