The papers are organized into parts by topic covering such subjects as the basic concepts in program testing, the design of programs and languages to facilitate testing, testing mathematical software, testing large software systems, models of program behavior, and standards and measures of program quality.

The first part dealing with the basic concepts in program testing includes the excellent introduction by Hetzel mentioned previously. The parts dealing with the design of programs and languages for testing contain papers restating the obvious need to structure programs so that they may be validated—a need, unfortunately, which is seldom, if ever, heeded in practice: of special interest in this part is the article by Snowdon on the PEARL system which is designed to facilitate the construction and proof of well-structured programs. The part dealing with testing mathematical software is interesting in that it deals with aspects of validation generally ignored in discussions of "program correctness"; for example, the effects of finite precision arithmetic and truncation, and such human factors as ease of use and documentation.

The part dealing with the testing of large software systems was of the greatest personal interest to this reviewer and leaves him with a feeling of awe. The part contains three papers. A. Scherr of IBM describes the process used to design, develop, and test the time sharing option for OS/360. J. Brown of TRW describes an automated software test tools used, among other things, to test orbital determination programs. E. Youngberg of UNIVAC proposes another system for software testing. This reviewer highly recommends these papers for anyone who lacks an appreciation of the true enormity of the problem.

The reviewer’s major regret is that the questions and discussions which must surely have followed the presentation of the papers were not included. I suspect they were lively and would have been enlightening.

W. A. Wulf
Carnegie-Mellon Univ.
Pittsburgh, Pa.


This book is intended (author’s preface) as a suitable text for course B2 of Association for Computing Machinery Curriculum 1968 Recommendations. In general, the level of presentation is consistent with that intent. However, the book does not contain adequate material on all of the topics suggested in B2. For instance, micro-programming is very briefly discussed in three pages, and only two descriptive pages are provided on macros. There is also not enough discussion of selected programming techniques such as sorting, searching, scanning, text editing, arithmetic expression recognition, syntactic recognition, etc.

In contrast to the shortcomings listed above, there is a good 45 page chapter on assembly language and facilities with numerous examples, including special emphasis (15 pages) on subprogram linkages and parameter passing protocols. Character and number systems and representations are clearly described in a 25 page chapter. The book goes much farther than required by B2 in the area of conventional switching theory through to the level of Karnaugh map techniques (35 pages), and the area of basic electronics (20 pages) and gates and flip-flops (20 pages). Thus, somewhat more than a quarter of the text pages is devoted to the area of basic switching theory and digital systems. This may be a questionable choice in a book with its particular title.

The above features of the book by no means constitute a complete description; and a summary of the book’s structure is now provided. The authors have chosen to organize the text into four sections: Section 1—Introduction to the Design and Function of Computer Systems (5 chapters, 56 pages, 100 questions); Section 2—How the Computer Computes (2 chapters, 56 pages, 51 questions); Section 3—Hardware Design (5 chapters, 102 pages, 102 questions); Section 4—Software Design (2 chapters, 40 pages, 40 questions). There is a 12-page appendix on number system details and logic diagram conventions, and a 500 term index. As well as the numerous thoughtful problems at the end of each chapter, there is a brief list of further reading references. The references are almost all to more advanced text treatments of the pertinent material.

The topics treated, by section, are as follows.

Section 1: The hardware/software relationship, system design, flow-charting of algorithms, equipment, binary numbers, logic, flip-flops, registers, main memory, control unit, arithmetic and logic unit, central processing unit (CPU), I/O, programming criteria, translations, source and object code, assembly language and techniques, supervisor, continuous and discrete value systems, information representation, storage addressing, binary numbers, fixed and floating point representations, codes, system designer’s job, hardware/software tradeoffs, traditional computer design, hardware design alternatives in operand addressing formats and I/O, and software design alternatives.

Section 2: Instruction decoding, memory reference instructions, nonmemory reference instructions, input/output, generalized assembly language, register reference instructions, integer arithmetic instructions, floating-point arithmetic instructions, program counter manipulation (jumps), subroutines, I/O instructions, logical instructions, shift instructions, and pseudo-instructions.

Section 3: Boolean algebra, truth tables, negation, OR, AND, EXOR, NAND, NOR, expressions, properties, duality, standard forms, gate implementations, completeness of NAND and NOR, algebraic minimization, Karnaugh maps, hardware circuits, diode gates, transistors, transistor logic gates, flip-flops, ferromagnetics, cores, addressing organization, semiconductor memories, other memories, tapes, disks, drums.

Section 4: Translators, assemblers, compilers, high-level versus low-level languages, problem-oriented languages, interpreters, conversational translation, microprogramming, supervisory command language, loaders, libraries, multiprogramming.

Sections 2–4 give increasing levels of detail on the topics briefly introduced in Section 1. This is why the topics from early sections seem to be repeated in later sections. It is a deliberate pedagogical style used by the authors. It would seem to make the book usable by a variety of people with different backgrounds. But it has a drawback when used as a text in a highly organized curriculum such as Curriculum 1968. Most of the material in Section 1 is actually included in course B1 (therefore students already have it) and some of the details in Section 3 are better placed in I6 (therefore the students will get it elsewhere). These faults would not be serious in themselves in a B2 text, except that in the case of this book, some material that is in B2 has been covered too lightly. In general, the book is well written, and the diagrams, tables, and program segments are easily understood.

Prof. V. C. Harnacher
Univ. of Toronto
Toronto, Ont., Canada


This book surveys five areas of current interest to computer scientists: formal language theory, computational complexity, program schemas, tree automata, and compiler design theory. Several of the papers in this monograph were given at a session entitled, “An Overview of Theoretical Computer Science,” at the Fourth Annual Princeton Conference on Information Sciences and Systems. The contributing authors and the titles of their articles are:

Ronald V. Book, “Topics in Formal Language Theory,”
Allan Borodin, “Computational Complexity: Theory and Practice,”
Zohar Manna, “Program Schemas,”