TEACHING OF COMPUTER DESIGN TO COMPUTER SCIENCE UNDERGRADUATES

B. W. Lampson

Rapporteurs: Mr. E. H. Satterthwaite
Mr. R. Snowdon

Teaching of Computer Design to Computer Science Undergraduates

Personal experience, both in programming and in providing liaison between programmers and engineers, suggests that the teaching of computer design is an impossible task. The subject is much vaguer and more confused than programming, but no one even knows how to teach the latter. Thus attention must currently be focused upon relatively small insights, not upon complete solutions.

Design as an Interfacing Discipline

Both computer design and programming are interfacing disciplines. A programmer realises an interface between the users of a computing system and its hardware; a computer designer, between classes of programs and a collection of components. Unfortunately, both disciplines have generally failed to achieve satisfactory matches at the 'soft' sides of the interfaces. The primary cause of this failure is that designers do not know and have not tried to learn much about what happens at the 'soft' end, i.e. about the users of the products they are designing.

The Design Process

The activity of computer design includes four phases.

1) Architecture

An abstract machine defined, for example, by a programmer's reference manual, must be designed. Such an abstract specification is what Dijkstra has called a 'pearl'; it defines the machine for those who must build upon it, so that they need not concern themselves with the ugly details of its realization.

2) Logic Design

A collection of 'perfect' components must be chosen and their interconnection must be designed for implementing the abstract machine. (In some cases, the first two phases are even reversed; the architecture is implicitly defined by the

collection of components which happen to be available).

3) Physical Realization

The idealized design must be further developed so that it will work with real devices, which are both imperfect (because of finite transmission speeds, noise, etc.) and more constrained (by the requirements for power, cooling, packaging, etc.). This phase must be concerned with the reliability of the actual device. It poses engineering problems; knowledge and techniques from electrical engineering must be applied to obtain the full potential of the hardware investment. This phase is qualitatively different from logic design, and appears to be more difficult, since bad engineering is the most common source of faults in the final product.

4) Maintenance

The final machine must be designed to be maintainable.

A striking characteristic of all phases of the computer design process is the total absence of any formal methods, or even meaningful methodologies or canons of good practice. There are a few formal schemes for minimization in logic design, but these are largely irrelevant in practice. Even in that phase, work is mainly done by hand, and decisions are made at the discretion of the designer. The lack of a relevant formal foundation presents a serious problem to the organizer of a University program in computer design. Since it is possible to learn from the mistakes of others, a study of history might be helpful. Indeed history and laboratory practice are about all that present courses in computer design have to offer.

Relation to Programming

Programming and the first two phases of computer design are quite similar. Both a computer architecture and a programming language (especially a language designed for direct interpretation) define an interface between levels of soft - or hardware. Thus there is a fundamental similarity, although computer architecture is influenced by the need for physical realization, usually appears at the micro level, and is often designed for greater generality. A tentative conclusion is that programmers might be better machine architects than engineers.

Logic design also is very similar to low-level machine language programming, such as that done in octal, and it is an observable fact that frequently the same people are good at both. There are some conceptual differences; most of these are related to the fact that parallelism is currently inexpensive in hardware but expensive in software, and do not seem to be fundamental. More significant are the practical differences between programming and logic design. Hardware design and manufacturing still use hand tool methods; there are no 'high level' tools comparable to those available for programming. A definition of hardware is 'that which cannot be changed very easily.' At the present time, there are no techniques for building hardware configurations cheaply, quickly, automatically, and reliably

The state of the art is demonstrated by the work of Seitz and of Bell. For their modular systems many of the physical and electrical problems of using real devices are resolved by the component designer and are invisible to the computer designer. Unfortunately much of the potential of the hardware is lost as a result. Furthermore, changes in the architecture or logic still require physical manipulation of the components, and as a result it is much harder to construct hardware than to construct programs, even in these systems.

None of the efforts in the field of computer-aided design of computers has yielded very satisfactory results. It is not clear why success in this area has been so elusive, and there still seem to be good reasons to believe that the use of computers to build computers could help enormously.

Microprogramming provides a useful tool, but it should clearly be considered a form of programming, not of computer design. It is basically a technique for extending the programmer's domain, where we have at least a few powerful tools to help us, at the expense of the hardware. In particular, most of the opportunities for parallelism, which provide the only significant conceptual distinction between computer design and programming, are not available to the microprogrammer. Because of the loss of this and other hardware

flexibilities, emulation by a general purpose microprocessor is slow, and satisfactory emulator performance is usually achieved by design of a special purpose microprocessor. For example, there is a nearly one-to-one correspondence between machine and micro instructions on the System 360/85, a microprogrammed computer with high performance, and a square root routine on that machine was found to be faster in 360 machine language than in microcode.

In summary, there is no meaningful distinction between programming and the phases of computer design concerned with architecture and logic design, except those associated with ease of design, construction, and change. Most of the latter difficulties arise from the problems of physical modification. Future technological developments are likely to produce an order of magnitude of change in component performance within ten years. Utilization of these advances will require more flexible thinking but will not lead to any clearer distinction between programming and computer design.

Relation to Reality

The third and fourth phases of computer design do not correspond to any form of programming but seem to have a distinctly different nature. Since the characteristics of real devices are highly dependent upon the details of a transient technology, meaningful teaching of the third design phase is especially difficult.

Machine and program maintenance, despite certain similarities, are different problems. Programs do not deteriorate; if a program works once, then it should always work when given identical conditions. In machine maintenance, time is an additional factor; a machine may work one day, break down, and fail to work the next.

Conclusions

There appears to be little difference between programming and the architectural and logic design phases of computer design, but the physical implementation and maintenance phases are completely distinct.

Studying mistakes seems to be one of the best ways of learning computer design. This is difficult for a student, since little is to be learned from a small project but large failures tend to be expensive. A book of 'horror stories' might be valuable as an inexpensive way of conveying a feeling for the problems of design to a student.

THE TEACHING OF COMPUTER DESIGN

Proceedings of the Joint
IBM University of Newcastle upon Tyne Seminar
held in the University Computing Laboratory
7th – 10th September 1971

Edited by B. Shaw

University of Newcastle upon Tyne Computing Laboratory 1972

Speakers

Professor D. Aspinall	:	Dept. of Electrical Engineering, University College of Swansea, Swansea.
Mr. P.D. Atkinson	:	Development Laboratory, I.B.M. United Kingdom, Ltd., Hursley Park, Winchester, Hampshire.
Professor R.S. Barton	:	Dept. of Computer Science, College of Engineering, The University of Utah, Salt Lake City, Utah, U.S.A.
Professor C.G. Bell	:	Dept. of Computer Science, Carnegie-Mellon University, Schenley Park, Pittsburg, Pennsylvania, U.S.A.
Mr. D.W. Davies	:	Dept. of Trade and Industry, National Physical Laboratory, Teddington, Middlesex.
Dr. B.W. Lampson	:	Xerox Data Systems, Palo Alto Research Centre, Palo Alto, U.S.A.
Dr. Z. Riesel	:	The Weizmann Institute, Israel.
Professor C.L. Seitz	:	The Dept. of Computer Science, College of Engineering, The University of Utah, Salt Lake City, Utah, U.S.A.
Professor J. Suchard	:	University of Paris, Paris, France.
Professor F.H. Sumner	:	Dept. of Computer Science, The University, Manchester.
Dr. D.J. Wheeler	:	Computer Laboratory, University of Cambridge, Cambridge.