SOME REMARKS ON THE TEACHING OF COMPUTER DESIGN

Z. Riesel

Traditionally, a distinction has been made between computer engineers, who design hardware, and between programmers, who write software. The main point I want to make in these remarks is that a knowledge of hardware is not enough for computer designers and I would like to distinguish between computer builders and between computer users. The first group includes all those people who provide computing services for the end users, which means the engineers and those programmers who provide supporting software without which the end users really cannot take advantage of that marvellous machine, the electronic computer. This group of programmers is roughly covered by the name systems programmers.

In the past, there has been a cleavage, a lack of communication between computer engineers and system programmers. This lack of communication is almost as bad as the "two cultures" described by Lord Snow, the lack of understanding between the scientific and the literary communities. Many engineers have not done much programming themselves and do not know what it is that systems programmers do. Programmers are put off by the physical aspects of computers. There are also psychological differences. We engineers are considered to be serious, but somewhat dull characters, and we consider programmers to be, by and large, brilliant but unreliable. Using the analogy of the motor car industry, we have a situation where the designer of the car never takes it out for a spin himself and where the racing driver does not know how the engine is put together.

This situation has several serious disadvantages. One is that present day computers are really not too well suited for some of the work they have to do, especially in the area of compilers and operating systems (1), and computer engineers are not in a position to do anything about it. Some of them have never heard that there exists a problem.

The second disadvantage derives from the fact that computer design is becoming too difficult for engineers to do with their bare hands. Computers have to be used in the design process, and also, more and more, in the manufacturing process.

Engineers have difficulties in using the machines they have themselves designed in their own work, to help them design better machines.

Thirdly, in many situations where a computer is to be adapted to a special situation, and I am thinking in particular about real time control systems incorporating a computer, the best solution will clearly not be reached by hardware alone, nor by software alone, and the designer has to understand both, if he is to be at all effective.

Fourthly, there is the area of diagnostic testing. Such tests, when written by engineers, are invariably too simple and put no real strain on the machine.

Finally, there is control by microprogram, where the distinction between hardware and software breaks down altogether.

The fact that so few programmers understand the workings of hardware has also undesirable consequences. The most serious of these is that programmers cannot take part in and contribute to the design process because they do not know what is easy, what is difficult, and what is impossible in hardware.

I would therefore like to state my opinion that the art and the science of computer design will not make the progress it should, unless engineers become programmers and programmers become engineers. The curriculum of a school of computer design must be such as to achieve this goal, by producing all round computer builders, proficient both in hardware and in software. Of course, this statement is not an original discovery on my part. A program on the lines I am suggesting is in operation in the United Kingdom at Manchester and Swansea, and a similar approach has just been suggested by a distinguished group in the United States (2).

The reconciliation of programmers and engineers appears to be a worldwide trend. It would probably be preferable if faculties of electrical engineering were to introduce a balanced option, but if they do not, a special department of computer engineering may evolve in some places.

If the general idea is accepted, it is fairly easy now, to list the subjects to be taught in an undergraduate school for computer builders. First of all, the traditional subjects of computer engineering.

1. Circuit design
2. Combinatorial and sequential logic
3. Computer arithmetic
4. System organization
5. I/O channel organization
6. Peripheral equipment.

In particular I would like to stress peripheral equipment, a subject that has been much neglected. The computer designer will of course not himself design the mechanical parts of disc drivers, tape transports, punched card readers etc., nor even the read/write circuitry, but he must understand them. Otherwise his control units will not work.

The other five subjects lend themselves to a clean, axiomatic approach and it is therefore very important that our students get plenty of exercise in getting their designs to work in the laboratory. Laboratory work is also the only way for students to acquire the ability to use instruments, in particular oscilloscopes.

In their laboratory work our future hardware designers and system programmers will put together counters, adders and similar simple structures. More importantly, they should have at their disposal a small computing machine to which they can tie various pieces of peripheral equipment and make them work by a well chosen mixture of hardware and software. They should also be permitted and even encourage to change the control sequences, to manipulate the register structure, so as to implement new and unusual instructions,

such as square root extraction, character cocatention or table look-up.

I may add that modern technology and equipment permits laboratory work by students at a much higher level than in the era of discrete circuits (3).

I wish to stress again that I would like to see people who intend to be systems programmers, though maybe not application programmers, to take these engineering subjects, including laboratory work, much as the idea may shock them.

The second set of subjects is now taught mainly in departments of computer science, though, in my opinion, it is of crucial importance to computer engineers.

7. Programming in machine language, in assembler language, in algebraic language, in list processing language, in simulation languages.
8. Theory operating systems, assemblers, compilers and interpreters.

This, of course, must include regular access to a good sized computing machine and the completion of a good number of programming projects, of increasing complexity. This should include application programs, possibly in circuit design, programs intended for real time control of strange devices and compilers for special purpose private language, possibly invented by the students himself.

Lastly, a group of subjects that, to the best of my knowledge, is taught in very few university departments, be they electrical engineering, mathematics or computer science.

9. Use of computers in computer design, including circuit design, logic simulation, design and manufacture of printed circuits, production and updating of drawings, wiring lists and spare parts lists.

The importance of these subjects will be obvious to anybody who has recently designed a computer or part of one. In the old days, of course, engineers designed just using pencil and paper, just as programmers used only obsolute binary machine language, but these methods will not serve much longer, except for very simple projects.

To summarize, the graduates of a computer builders school will have a better understanding than mere engineers, or mere systems programmers. The introduction of such a program is desirable, urgently required, and, I believe, inevitable.

In conclusion, I would like to say something about the ordinary computer user and his need to understand something about the workings of a computing machine. I am not concerned so much about the big machine users, who are quite sophisticated and knowledgable but about small, occasional machine users, who just want to get a small job done, probably from a remote terminal. I once heard a lecture by one of the first implementors of an interactive time sharing system. The students, he said, were given an electric typewriter and taught to type in their problems and how to go about getting the correct answers. Some weeks later, when they had become experts at this, they were taken on a visit to the computing center. But, the lecturer reported proudly, they could not understand the need for all this horrible machinery, whereas the use of their typewriters for problem solving seemed perfectly obvious to them.

Now I, personally, do not like this attitude. It reminds me of the cargo cults of the South Pacific. I believe that a man who pressed a switch and turns on an electric light should know that somewhere there is an electric power station where energy conversion takes place and electricity is distributed to all users. Similarly, I believe even a very casual computer user should know about gates and registers and that a stored program is executed step by step by operating on these registers. As a matter of fact, I believe that such knowledge should by a part of popular culture in the twentieth century and be taught in the secondary schools.

References

(1) W. M. McKeenan, Language directed computer design, Fall Joint Computer Conference, 1967.

(2) C. L. Coates et al, An Undergraduate Computer Engineering Option for Electrical Engineering, Proceedings of the IEEE, June 1971.

(3) T. L. Booth, Undergraduate Digital Laboratories.