PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

The Clifford Paterson Lecture, 1990: Progress and Research in the Computer Industry

Maurice V. Wilkes

Phil. Trans. R. Soc. Lond. A 1991 **334**, 173-184
doi: 10.1098/rsta.1991.0006

The Clifford Paterson Lecture, 1990
Progress and research in the computer industry

By Maurice V. Wilkes
Olivetti Research, 24a Trumpington Street, Cambridge CB2 1QA, U.K.

The paper surveys the technical progress which has occurred in the computer industry in the recent past and the way in which the relationship between that industry and the semiconductor industry has developed. Research in the computer industry is now dominated by software, a subject which has an intellectual basis rather than a basis in the experimental sciences. For this reason, the management of research in the computer industry presents problems of its own.

Computers and semiconductors

In 1980 it was far from clear what the future relationship would be between the semiconductor industry and the computer industry. Microprocessors, which were then coming into use, were the product of the semiconductor industry and one extreme view expressed was that that industry might take over the computer industry altogether. That was not likely to happen and in fact has not happened; instead an interface, with a firm technological basis, has been put in place and has enabled the two industries to coexist and cooperate.

Transistors are analogue devices but, if properly designed for the purpose and used in the proper manner, can behave as binary elements from which gates and flip flops can be made. The designer provides a set of design rules which lay down permissible ways in which the transistors may be used in digital circuits. If these rules are followed, a circuit designer who knows very little about semiconductor physics can successfully design digital circuits of any degree of complexity. The design rules constitute an interface between the process engineer and the designer of digital chips, whether the latter works in the semiconductor industry itself or in the computer industry.

In many industries computer models have come to play a leading role in the design process. Nowhere is this trend more marked than in the design of computer hardware. At one time, the universal practice was for experimental models – known for historical reasons as breadboards – to be constructed, and for the circuits to be checked out on a laboratory bench using an oscilloscope. These methods have now almost entirely given way to methods based on computer modelling.

The circuit is described in a hardware description language or the equivalent, and put into the computer. The tools used for checking it out are software tools; they include programs for verifying that the design rules have been followed and simulators for checking that the logic does what is intended. Other simulators enable the maximum operating speed to be determined. Similar methods are applicable

Phil. Trans. R. Soc. Lond. A (1991) **334**, 173–184
Printed in Great Britain

across the whole range of hardware design, from printed circuit boards and the silicon chips that go on them to highly complex microprocessor chips.

The fact that simulation enables a reliable estimate to be obtained of the performance of a given piece of hardware without actually constructing it, and hence enables alternative configurations to be compared, has done much to improve the efficiency with which hardware operates. This is particularly true in the case of processor design, as was demonstrated by the RISC movement for a new approach to processor design. Far-reaching claims were made for this new approach, namely, that it would enable processors to be designed that would occupy half as much silicon area as earlier processors, would take half the time to develop, and would run twice as fast. These claims were made good in the first instance by showing the results of simulation. Later, when RISC processors were built, their performance confirmed the results of the simulation.

CMOS processors

The power of CMOS microprocessors has steadily increased as the feature size has been reduced. It follows from the laws of physics that this results in a linear increase in speed. It also enables more transistors to be accommodated on a given area of silicon.

The time required to send a signal from one place to another depends on the amount of power available to charge the capacitance of the interconnecting wires. This capacitance is much greater for inter-chip wiring than for on-chip wiring. With CMOS it is not possible to provide enough power to drive inter-chip wiring at high speed. There is, therefore, a great premium on putting the whole processor on a single chip. The point at which this could first be done represented an important stage to be reached in the development of processors.

Not only have feature sizes decreased, but chip sizes have increased. The total number of transistors that can be accommodated on a single chip has therefore increased dramatically. The progress which has taken place in the past five years is well illustrated by comparing the MIPS R2000 and the Intel i860.

The MIPS R2000 processor was developed in 1986 using 2 μm technology. It is based on a RISC processor which takes up about half the available space on the silicon. The remaining half would have been insufficient to accommodate more than a very small amount of cache memory and instead the designer included the cache control circuits for off-chip instruction and data caches. This left about one third of the entire chip available and it was used for a memory management unit (MMU) with a translation look aside buffer (TLB). The importance, at that period, of RISC philosophy in making the processor as small as it was does not need to be emphasized. A processor of the same power, but of earlier design, would have occupied the entire chip with no room for anything else.

Three years later when the Intel i860 processor has developed it had become possible to accommodate on a single chip, not only the units mentioned above, but also a data cache, an instruction cache, and a highly parallel floating point coprocessor. The chip itself provided slightly more than twice the silicon area, and the amount of space taken up by each transistor was less by a factor of 2.5. The result was an increase in the effective space available by a factor of 5. The basic RISC processor itself then took up only 10% of the whole compared with 50% on the R2000. The floating point coprocessor took up 35%, the memory management unit 20% leaving 35% for cache memory.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Although the total effective space on the i860 is only five times that on the R2000, there are 10 times as many transistors – about a million. This is because much of the space is used for memory and memory is relatively more dense in transistors than logic.

The most advanced present-day CMOS processors have a feature size of a little less than 1 μm, and a transistor count of approximately a million. Progress will continue. CMOS microprocessors have already pushed up to what used to be regarded as the top end of the minicomputer range; it may be expected that they will soon go well into the main frame range. There does not seem to be any reason why a feature size of $\frac{1}{10}$ μm should not be reached before the effects of statistical noise make themselves felt. At some intermediate point, it will be necessary to reduce the operating voltage, and this will limit somewhat the gain in speed that will be obtainable.

Bipolar processors

Bipolar transistors have followed CMOS transistors in becoming smaller, although there has been a lag. This is mainly because the bipolar process is intrinsically more complex, but it is also partly because the great success of CMOS technology has led the semiconductor industry to concentrate its resources on it. It takes twice as many transistors to make a gate in bipolar technology as it does in CMOS.

Bipolar transistors are faster than CMOS transistors by a factor of perhaps five and they operate at a higher power level. The penalty that must be paid to send signals from one chip to another is not therefore as great. This has made it possible to build very fast multichip processors, using bipolar technology, frequently in the form of gate arrays. The fastest computers on the market at the present time all use bipolar processors.

Nevertheless, as bipolar transistors have become intrinsically faster, it has become necessary to develop interconnect systems that are faster than traditional printed circuit boards. This is becoming more and more difficult, and bipolar technology is, in consequence, approaching the point – reached earlier with CMOS – when in order to make further progress it will be necessary to put the whole processor on the same chip.

A pure bipolar chip, with a million transistors on it, will dissipate at least 50 W, probably twice as much. Removing the heat presents problems, but these are far from being insuperable. More severe problems are encountered in supplying the power to the chip and distributing it without a serious voltage drop or without incurring unwanted coupling. Design tools to help with these problems are lacking. There is, however, good hope that they will be overcome and that it will soon be possible, using custom bipolar technology, to implement a processor similar to the R2000 on a single chip. Such a processor may be expected to show a spectacular increase of speed compared with multichip implementations using gate arrays.

The performance of a single chip microprocessor depends to a major extent on the amount of on-chip memory it contains. As the number of transistors goes up, this may give an advantage to CMOS, since at least four times as many transistors are needed to implement a memory cell in bipolar technology as in CMOS. A CMOS chip containing several million transistors will therefore be likely to have significantly more on-chip memory that a similar bipolar chip. This advantage, combined with the benefits of having a simpler process and lower power consumption, may be sufficient to keep CMOS ahead in the race with bipolar technology.

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

Many advantages that would result from being able to put CMOS transistors and bipolar transistors on the same chip. Semiconductor companies are devoting much effort to this area, under the generic name BICMOS. There are two approaches possible. One is to implement the processor and on-chip memory in CMOS and to use bipolar transistors to drive both off-chip interconnect and long-distance wires on the chip. For this purpose, the bipolar transistors need not be of very high performance, and could be obtained by adding extra process stages to a CMOS process. The alternative is to take a high density bipolar process and add CMOS. The processor could then be implemented with bipolar transistors and the on-chip memory in CMOS.

Although at this stage it appears attractive, BICMOS will undoubtedly bring a spate of problems of its own, particularly as the noise characteristics of CMOS and bipolar circuits are very different. One of the attractions is that a BICMOS chip would dissipate very much less power than a bipolar chip of similar size; the problems of distributing power on the chip and removing the heat generated would therefore be much less.

CMOS, bipolar, and BICMOS technologies are all in a fluid state of development. There are many options open and it is by no means clear which way development will go. The preferences of research workers, the facilities available to them, and the manner in which the semiconductor industry develops as a whole will all have their influences.

Gallium arsenide

All the time, waiting in the wings, is gallium arsenide. Gallium arsenide is a semiconducting material whose properties resemble silicon, except that it has a better signal noise performance and is capable of switching speeds of up to five times as fast. For some time there was difficulty in manufacturing the wafers to the requisite standard of purity, but this now seems to have been largely overcome. Many gallium arsenide chips are available on the market, including gate arrays containing up to 16000 gates and implemented in a process which resembles CMOS, or rather NMOS, in that field effect transistors (FETs) are used.

The success, as a competitor to CMOS, of processors implemented in a gallium arsenide FET process depends entirely on the same density of integration becoming possible. With the high switching speeds of gallium arsenide transistors and the very limited power available, any necessity to send signals from one chip to another would be fatal. It is essential that the entire processor should be accommodated on the same chip.

Parallelism

For a long time people have been saying that when we run out of speed with uniprocessors we will have to turn to parallelism for further advance. We have not run out yet and, as I have indicated, we still have a long way to go. At the same time it has turned out that getting speed by parallelism is easier said than done. Those who thought that a computer with many slow processors would be an effective, lower cost, competitor to one with a single fast processor have been proved wrong. The problem lies in keeping the processors busy. There are major theoretical difficulties in the way of automatic scheduling of the work so that this is achieved. Such scheduling needs the kind of human insight into a problem and its background that goes far beyond what a compiler or operating system can achieve by statically

analysing the code. It was, in my view, the failure to appreciate this fact in the early 1980s that caused over-optimistic predictions to be made.

I do not see machines with massively parallel architectures coming to play a major role in general purpose computation. There are, however, areas in which programs are of sufficient value and remain in use for a sufficiently long period for a large investment of human labour in optimizing them for parallel computation to be justified. High energy physics was one of the first of these; others continue to emerge as time goes on.

As workstations, or rather the single-chip CMOS processors within them, are becoming more powerful, we are seeing the demise of the minicomputer as a distinct category. A minicomputer is now becoming simply a workstation repackaged with a more generous endowment of peripheral and communication equipment and possibly with more memory. We have also seen the emergence of the file-server – a version of the minicomputer or enhanced workstation dedicated to one particular function.

Mainframes are still with us and still contain the fastest processors in general use. That, however, is no longer their principal feature. They stand out because they have a very large processor I/O bandwidth, and come with very large disks and other devices for storing large quantities of data. These features make today's mainframes valuable as bulk file-servers and database repositories.

The nature of software

Software products have an established place along with other industrial products and the software engineers who produce them work alongside engineers responsible for hardware. However, software products differ fundamentally from hardware products. Software is entirely a creation of the human brain and its complexity is manmade. It has an intellectual basis, not a basis in experimental science.

Thirty years ago, when people were first coming to grips with the problem of producing software on an industrial scale, there were attempts to treat programmers as technicians and put them to work under a supervisor who might be a system analyst or a product manager. That was in the days of batch processing and primitive programming languages. Now the advance of technology, including the development of workstations and networks, has put major resources at the disposal of a single person and has allowed the programmer to achieve his potential as a fully autonomous professional. A modern programmer or software engineer can personally, from his workstation, take full executive responsibility for all aspects of a project, ranging from the strategic overview to programming details.

Not all programmers would call themselves software engineers nor, I think, is there any specific style of programming that can be called software engineering. All programmers who handle large projects face essentially the same problems. Like professionals in other disciplines, programmers have their own tastes and adopt the methods of working that they are most comfortable with. There are styles of programming as there are styles of management, and the myth that there is a particular style or discipline of programming to be preferred to all others is one that I would reject.

Programming languages

About 1960 a small group of very able people began to turn their attention to programming languages as an intellectual study. The result of their work was to

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

create not only the language ALGOL 60, but also a wholly new branch of computer studies. ALGOL 60 attracted wide interest at the theoretical level, and rapidly became the subject of much discussion and controversy. A number of compilers were successfully written. However, the language had various characteristics that rendered it unattractive to practical computer users. Perhaps the most serious limitation was that if a change were made to the body of an ALGOL program a complete recompilation was necessary. This made it difficult to work with very large programs. Moreover, the ALGOL group turned a deaf ear to any suggestions that would improve run-time efficiency or improve the effectiveness of the type-checking system. Although they would have denied it at the time, in retrospect they seem to have been more interested in programs as mathematical constructs than as tools for getting things done. ALGOL 60 was offered as something complete and final and the group opposed any attempt to set up a body that would be responsible for its evolution in the light of experience. By contrast, FORTRAN and COBOL were driven by people who stressed the merits of standardization and upward compatibility. They were preoccupied with short-term issues, and saw only the immediate limitations of what the ALGOL group had done and not its potential.

The result was that those interested in programming split into two camps and large-scale users did not profit, as they might have done, from the significant advances that were made under the ALGOL banner. However, by the 1970s, the widespread popularity of Pascal showed that there was a demand for a language constructed on scientific principles, although Pascal inherited many of the problems that I have indicated above.

There are welcome signs that a new phase is now beginning with the development of object-oriented languages. The terminology is somewhat imprecise, but I take an object to be a package containing both code and data; access to the data from outside the object is possible only by invoking the code. Objects can be used in the same way as procedures, but they are not subject to the hierarchical constraints imposed on procedures in block-structured languages implemented on a single stack, and they can be compiled separately. Moreover, it is possible to provide rigorous type checking across object boundaries. Objects make the logical structuring of a program much easier than it was with block structured languages. In fact the structured programming movement may now be seen as a brave and only partly successful attempt to convince people that logical structuring was possible in spite of a strict adherence to a block structure.

Operating systems

For twenty-five years it has been apparent that sooner or later operating systems would become machine-independent, as programming languages had already done. It often happens that developments that are inevitable but long delayed come, at last, in an unexpected way. When I first heard of Unix as an operating system written for the PDP7/9 and later moved to the PDP11, and when in 1980 I saw it being reimplemented on the VAX, I did not expect that it would be the vehicle by which the world would first come to take to its heart an operating system that ran on many different processors. Unix had one great advantage from the beginning, namely, that it carried bundled with it an extensive library of useful programs. These made it something more than a mere operating system, almost a programming

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

environment. Nevertheless, it was a system more suitable for use by software gurus than by ordinary users. It appealed to people to whom the C language in which it was written – or rather rewritten since the original version was in PDP7/9 assembly code – also appealed. Unix and C promoted each other and each has helped the other's success.

Those responsible for the development of the early workstations were all Unix enthusiasts, having had experience of it on minicomputers. It was a natural consequence that Unix should become the standard operating system for workstations. This was perhaps the most important fact in causing Unix to emerge as the first major machine-independent operating system.

One cannot help asking what would have happened if, instead of Unix, a long-established operating system from one of the major vendors had been the first operating system to emerge in machine-independent form, and had been offered with mature marketing support and good documentation. That this did not happen is, no doubt, partly due to the desire of vendors to protect their customer bases. However, that argument would not have remained decisive if there had not also been a major practical difficulty, or rather set of difficulties, that made the writing of a machine-indpendent version appear as a daunting task. These difficulties arose because there was no cleanly defined interface between these operating system and the layers of software immediately above them. Because machine independence had not been an objective, no attempt had been made to prevent the writers of software from making assumptions about the lower levels of the operating system and the way it interfaced with the hardware. This was true of software provided by third parties, as well as of software provided by the hardware vendors. The key to machine independence is having such an interface and preserving its integrity. Since enforcing an interface always involves some loss in efficiency, or is thought to do so, there is no incentive to do it unless some advantage – in the present case machine independence – is thereby attained.

Unfortunately, it is not true that all Unix systems have exactly the same interface. There are two main brands of Unix and variations within the brands. The situation will remind an old timer of the one that existed with FORTRAN before the introduction of the ANSI standard. Users complained bitterly because FORTRAN programs that they had written for one version would not run on another. Even when the standard had been introduced much further effort and self-discipline on the part of the users was necessary to make its use universal.

It is anybody's guess what the future of operating systems will be. I remarked above that Unix is as much a user environment as an operating system, and any challenger must be the same. It is possible that the Unix program library will come to be seen as more enduring than the operating system itself. X windows came at the right time to provide a better user interface than the original command interface, but X-windows on top of Unix is cumbersome. The Macintosh operating system showed, when it appeared in the early 1980s, what can be achieved with modest resources if the operating system is designed round a windowing system.

One day there must be a re-examination of what users' needs really are. Do the multitudinous features currently offered really help? What price do they entail in memory and processor cycles? Could that memory and those processor cycles be better used? Would users be happier in a simple world? Falling costs and the coming of larger memories have delayed such an assessment, but I feel that it must come. In the case of processor design a similar assessment has already taken place under the

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

aegis of the RISC movement. In that case, technical performance rather than users' convenience and preferences was the issue. It was possible to show that the adding of features one after the other could slow the processor down. It was no accident that the RISC movement came at the moment when simulation was able to provide hard facts. Quantitative assessment is not yet possible in the case of operating systems, but there is force in the view that operating systems have become too large and too complex and that this trend should be reversed.

Processor instruction set

During the whole period that the modern computer industry has been in existence, the processor instruction set has played a central role in characterizing a range of computers. It has been of importance to the prospective purchaser of a computer system as well as to the computer manufacturer. Now suddenly, in the workstation world, this is no longer the case. It is the Unix operating system and the variations to be found between the various vendors' offerings that loom large in the customer's calculations. I view this change as being a natural consequence of the adoption of an operating system capable of being implemented on any processor. If the industry moves, as I believe it will, slowly but surely towards the general adoption of machine-independent operating systems, then the processor instruction set will be permanently deposed from its former pre-eminent position.

To install Unix on a new processor, it is necessary first to equip the processor with a C compiler. If speed of compilation is not an important consideration, compilers for other languages may be designed so as to compile into C; they need pay little regard to optimization since C compilers found on Unix workstations may be expected to produce highly optimized code. Used in this way, C plays a role similar to that played by assembly languages.

There still remain some features of the processor design that show through the operating system. Among these are word length, floating-point format, and the ordering of bytes in a word. Lack of uniformity in regard to these features has been a source of inconvenience for some time and it will remain so until standards emerge and become adopted generally. This has already happened in the case of word length and we see it happening in the case of floating-point formats as a result of the introduction of the IEEE standard.

Progress in system software

Although much has happened in programming languages over a period of thirty years, progress from year to year is hard to discern. Even over five years there is not very much to see. The same is true of operating systems which started with the FORTRAN Monitor System and went on to time sharing and transaction processing.

Advances are brought about in the first place by the sustained efforts of unusually able people mostly, but not only, working in universities. They are made possible in part by the steadily increasing power of computer hardware, and in part by theoretical advances in syntax analysis, compiler design, synchronisation primitives, etc. They are introduced into industrial practice by students fresh from their first degrees and by established software engineers who have kept themselves up to date

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

by attendance at conferences and tutorial courses. There can be no industry in which the continual retraining of engineers in so widespread or in which it is so necessary.

Industrial research

Industrial research in the modern sense may be said to have its origins in the great interwar industrial research laboratories, especially the GEC Research Laboratories (now known as the Hirst Research Centre) at Wembley that Clifford Paterson made famous and the Bell Telephone Laboratories in New Jersey. The purpose of these laboratories was the application of the experimental scientific method to industrial problems. They differed greatly from the inventors' workshops of an earlier generation, particularly in regard to the scientific qualifications of the people they employed. An important outcome of their work was the acquisition of patents which could either be exploited directly or licenced to other companies. Money spent on research was seen as being a profitable investment in the purely financial sense.

My first visit to the GEC Wembley Research Laboratories was in the 1930s when I was still a student. It was at a time when the increase in motor traffic had led to a need for better road lighting and, in particular, for lighting fittings that would produce a uniform level of illumination on the road. This required improved lens design, and we were shown an experimental set-up in which the intensity of the light emitted from a fitting could be measured as a function of angle. Photocells were then high technology and measurement was regarded as a hallmark of the scientific method.

The model of industrial research just described is still valid in many industries, but must be applied with caution in the computer industry. As the computer industry has become dominated by software, hardware research has receded into the background. Some work continues in certain areas such as magnetic recording and integrated circuit packaging technology. Little of the basic physics research needed for the development of integrated circuits is now done in the computer industry. Software research makes no demands for laboratory facilities of the traditional kind or for people with qualifications in the experimental sciences. It is necessary to have people with original minds and an interest in industrial innovation, but the skills they need are essentially the same as those needed by software engineers or computer scientists generally.

Apart from computer applications, in which I include circuit simulators and other computer design tools, software research in the computer community centres on: basic programming methodology; programming languages; display software and software associated with workstations; system software, in particular operating systems and programming environments; file servers and databases.

The computer industry sees itself as advancing as a whole, with industrial research laboratories and university laboratories both contributing to innovation. There is keen competition between companies, especially in the United States, to recruit the more creative computer science and engineering graduates and to provide them with an environment in which they can develop their talents for the benefit of the company. They are encouraged to take an interest in long-range topics in cooperation with colleagues in universities and other companies. While most such work has limited relevance to the business of a particular company, it is nevertheless important in providing a vehicle by which the more productive members of the

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

research staff may establish links with their opposite numbers in other companies and in universities. Without such links, a research organization rapidly becomes inward looking, and unable to fulfil one of its most important roles, namely that of alerting the company at an early stage to incipient trends and developments that may impact its product strategy.

Product designers charged with the short-term evolution of product lines also establish external links, but these tend to be with a different sector of the computer community. Product designers need to keep themselves fully aware of what is happening to their own particular markets, and be prepared to meet competition as they see it developing. Their view of the industry is very different from the longer-term view seen by research engineers, but both are valid and one complements the other.

Transfer of technology from outside a company to within it is an important function of an industrial research organization. Many industrial research projects are aimed, not at doing innovative research, but at importing and acclimatizing technology developed elsewhere. This function is just as important as that of doing innovative research and is likely to absorb the greater part of the effort of an industrial research organization.

A characteristic of the industry is that advances in software that directly affect a user must, if they are to be successful, be adopted by the industry generally. Research is conducted with a degree of openness that it would be surprising to find in other areas. This is possible in part because the transfer of software technology is not easily brought about even to one's friends. They as well as competitors need time and experience to become comfortable with new ideas. The research ultimately benefits the industry as a whole. In the short term, the company that has invested in it, and believes in it, can hope to profit first. If a company attempts to exploit the result of the research in an exclusive manner, by being unwilling to release the relevant software, or by means of a software patent (as is now possible in the United States), it runs the risk of losing this advantage. Anyone who has observed the computer industry over a period of time will be able to point to examples of this happening.

Cooperative research

The current fashion among public bodies that sponsor research in European industry is to put much emphasis on collaboration between independent organizations. I suspect that this had its origin in a desire to promote the European spirit and to rectify what was rightly perceived to be an uneven distribution of technological expertise across Europe.

No doubt these are important objectives, but I have some reservations about the effect of collaboration in innovative research. Collaboration is useful where it enables resources to be pooled and, in some sciences such as the Earth sciences, international cooperation is essential if certain kinds of research are to be undertaken at all. However, there is no merit in cooperation for its own sake. Forced cooperation can blunt the sharp edge of innovation. At the highest level, research is aimed at changing people's mind sets. You do not do this by collaborating with them. By all means have them to work with you as disciples if they want to come and you can accommodate them.

Before engaging in a collaborative project, a company should consider carefully what its objectives are. If the transfer of technology or the promotion of good relations with other governmental or non-governmental organizations is important, then collaboration may be the correct course. But if such objectives, which are not themselves research objectives, are allowed to influence a genuine research project, the quality of the research is likely to be degraded. Unfortunately, collaboration is something that can be easily measured; the ultimate absurdity is reached if the size of the travel budget is used as a metric for assessing the health of a research programme.

In the United States, companies were, until recently, inhibited from entering into cooperative research agreements by fear of anti-trust proceedings. The situation was materially changed by the passage of two Acts of Congress, namely the National Cooperative Research Act of 1984 and the Technology Transfer Act of 1986. As a result, cooperative research organizations are now to be found in both the computer industry and the semiconductor industries. These organizations have been created by the companies themselves without government assistance, but cooperation by US Government laboratories is allowed under the second of the two acts mentioned above. In the computer industry, the first joint organization to be set up had as its object the pooling of resources to support long-term research. More recently, the Open Software Foundation has been established, and through it the participating companies hope to create what will become a common software basis for future products in the industry as a whole. This is a recognition of the fact, noted earlier, that software advances must be adopted by the industry generally if they are to achieve their full impact.

The central problem with all collaborative research undertakings is that of ensuring that the work being done remains relevant as the future unfolds. This is especially the case with innovative research. Flexibility is of the essence in research, and much money can be wasted if a research programme is set in contractual concrete at the beginning and pursued unchanged through thick and thin. Unfortunately, when major changes to a project become necessary, it may be difficult for the parties to agree on what they should be, and in the extreme case, even more difficult for them to agree on cancellation.

Long-term stability

When I was a young consultant employed by Ferranti Limited, Sir Vincent de Ferranti used to compare a company with a ship; the ship, he would say, must stay afloat at all times. He looked forward to being able to hand his own company over to his sons as he himself had received it from his own father. It is this sort of security and stability that is needed for research to flourish in a company. To achieve it a company must build up fat but, under modern conditions, if it does so it soon becomes the target for a take-over bid.

I am aware that these comments about the long-term stability of a company have relevance far beyond the welfare of industrial research organizations and those who work in them. The top management of operating companies are often blamed for taking a short-term view, but they are the helpless victims of the financial environment in which they live. Perhaps the influence of the business schools is not entirely beneficial. The managements certainly suffer from the fact that the directors of holding companies appear to be more interested in finance than in understanding

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES

the working of any particular industry. These, however, are issues on which I have little qualification to speak, nor on this occasion have I the charter to do so.

I thank the following who have provided me with information or otherwise helped me with the writing of this lecture: J. Dion, J. L. Hennessy, R. M. Needham, P. Robinson, R. W. Taylor.

Typescript received 31 October 1990; lecture delivered 15 November 1990

PHILOSOPHICAL TRANSACTIONS OF THE ROYAL SOCIETY A MATHEMATICAL, PHYSICAL & ENGINEERING SCIENCES