Computer Conservation Society

Aims and objectives

The Computer Conservation Society (CCS) is a co-operative venture between the British Computer Society, the Science Museum of London and the Museum of Science and Industry in Manchester.

The CCS was constituted in September 1989 as a Specialist Group of the British Computer Society (BCS). It is thus covered by the Royal Charter and charitable status of the BCS.

The aims of the CCS are to

- ◇ Promote the conservation of historic computers and to identify existing computers which may need to be archived in the future
- ◇ Develop awareness of the importance of historic computers
- ◇ Encourage research on historic computers and their impact on society

Membership is open to anyone interested in computer conservation and the history of computing.

The CCS is funded and supported by voluntary subscriptions from members, a grant from the BCS, fees from corporate membership, donations, and by the free use of Science Museum facilities. Some charges may be made for publications and attendance at seminars and conferences.

There are a number of active Working Parties on specific computer restorations and early computer technologies and software. Younger people are especially encouraged to take part in order to achieve skills transfer.

Resurrection

The Bulletin of the Computer Conservation Society

ISSN 0958 - 7403

Number 26

Autumn 2001

Contents

News Round-Up **2**

Assertions: A Personal Perspective
Tony Hoare **4**

Keeping Old Warhorses in Action
Stuart Fyfe **13**

Punched Cards Are Not Dead Yet
Hamish Carmichael **21**

CCS Web site information **22**

Zuse Z3 replica is operational
Martin Campbell-Kelly **23**

In Memoriam ICL
Hamish Carmichael **24**

CCS Collection Policy **25**

Society Activity **26**

Letters to the Editor **29**

Forthcoming Events **31**

News Round-Up

A life size bronze statue of Alan Turing was unveiled in Manchester's Sackville Park on 23 June. The monument, designed by Glyn Hughes, depicts Turing sitting on a bench holding an apple.

- 101010101 -

The Treasurer reported at the AGM in May that the Society's finances had improved significantly over the year. This was thanks mainly to donations of over £3500 from members. Thank you all for your support.

- 101010101 -

Our archivist, Harold Gearing, has announced his decision to relinquish this responsibility on grounds of age, handing his functions over to the Secretary over a period of time. The AGM passed with acclamation a vote of thanks to Harold for his long and valuable service to the Society.

- 101010101 -

The origins of software were the subject of a Society seminar at the Science Museum in June, under the title *Program Verification and Semantics: the Early Work*. Among the illustrious speakers at this event was Sir Tony Hoare, and an edited version of his presentation appears on pages 4-12 of this issue. Other speakers included Professors Peter Landin and Robin Milner, while Professors Jonathan Bowen and Cliff Jones co-chaired the meeting. A full report will appear in the next issue: readers can go now to <vmoc.museophile.sbu.ac.uk/pvs01/> for further details.

- 101010101 -

There was a conference to celebrate the 50th anniversary of the first business application of a computer at London's Guildhall in early November. Sponsors included the BBC World Service, BT Ignite, the Corporation of London, ICL, KPMG, the NCC and Yahoo.

The Society's Pegasus now sits in its own purpose-built section of the Science Museum, which was formally opened in May in the presence of many contemporary computer specialists, some of whom worked with Pegasus computers during their active life. They included John Crawley, formerly of the National Research Development Corporation (NRDC) which played a major role in instigating the Pegasus project; George Felton, a former Ferranti engineer who wrote many programs for Pegasus; and other former Ferranti executives including Sir John Fairclough, who went on to achieve renown as a Government scientific adviser.

- 101010101 -

The Charles Babbage Institute is launching an electronic journal under the title *Iterations: An Interdisciplinary Journal of Software History*. Editor Philip Farna tells us that *Iterations* "provides an outlet for scholarly articles on software history, a forum for first hand accounts of significant events and developments on software, reviews, and feedback from readers". For more information see <www.cbi.umn.edu/iterations/faq>.

- 101010101 -

The Association for a Conservatory of Information Technology (Aconit), the Association for History of Telecommunications and Information Technology (AHTTI) and the Committee for the History of Armements [sic] (Charme) are jointly organising an international conference on the History of Computing and Networks for autumn 2002, in collaboration with the Institute of Applied Mathematics in Grenoble (Imag). More details can be found at <www.aconit.org/colloq2002>.

- 101010101 -

Max Burnet, Secretary of the Australian Computer Museum Society, reports that there are two Ferranti Sirius computers still surviving Down Under — a remarkable survival rate as only three were ever shipped there. One is at Monash University, the other in the Museum of Victoria, and both are in excellent condition. Max would be interested in hearing from anyone who has further information on the Sirius, at <mburnet@bigpond.net.au>.

Assertions: A Personal Perspective

Tony Hoare

The author describes how an early interest in preventing programs from crashing has led to a lifelong study into methods of proving the correctness of programs. He tells how this led to an interest in assertions and their role in program proofs, and how that in turn pointed towards a methodology for the specification and design of programs.

An assertion is a Boolean formula written in the text of a program, at a place where its evaluation will always be true (at least, that is the intention of the programmer). In the absence of jumps, it specifies the internal interface between the part of the program that comes before it and the part that comes after. The interface between a procedure declaration and its call is defined by assertions known as preconditions and post-conditions.

If the assertions are strong enough, they express everything that the programmers on either side of the interface need to know about the program on the other side, even before the code is written. Indeed, such strong assertions can serve as the basis of a formal proof of the correctness of a complete program.

In this article, I will describe how my early experience in industry triggered my interest in assertions and their role in program proofs; and how my subsequent research at university extended the idea into a methodology for the specification and design of programs.

Now that I have returned to work in industry, I have had the opportunity to investigate the current role of assertions in industrial program development. My personal perspective illustrates the complementary roles of pure research, aimed at academic ideals of excellence, and the unexpected ways in which the results of such research contribute to the gradual improvement of engineering practice.

Programs Will Crash

My first job was as a programmer for Elliott Brothers of London at Borehamwood. My task was to write library programs in decimal machine code for the company's new 803 computer. After a preliminary exercise which gave my boss confidence in my skill, I was entrusted with the task of implementing a new sorting method.

I really enjoyed optimizing the inner loops of my program to take advantage of the most ingenious instructions of the machine code. I also enjoyed documenting the code according to the standards laid down for programs to be delivered to customers as part of our library. Even testing the program was fun; tracing the errors was like solving mathematical puzzles. How wonderful that programmers get paid for that too! Surely programmers should pay the cost for removal of their own mistakes.

But not such fun was the kind of error that caused my test programs to run wild, that is, crash; quite often, they even over-wrote the data needed to diagnose the cause of the error. Was the crash due perhaps to a jump into the data space, or to an instruction over-written by a number?

The only way to find out was to add extra output instructions to the program, tracing its behaviour up to the moment of the crash. But the sheer volume of the output only added to the confusion. Remember, in those days the lucky programmer was one who had access to the computer just once a day. Even 40 years later, the problem of crashing programs is not altogether solved.

Role of Syntax

When I had been in my job for six months, an even more important task was given me, that of designing a new high level programming language for the projected new and faster members of the company's range of computers. By great good fortune, there came into my hands a copy of Peter Naur's report on the algorithmic language Algol 60, which had recently been designed by an international committee of experts. We decided to implement a subset of that language, which I selected with the goal of efficient implementation on the Elliott computers. In the end, I thought of an efficient way of implementing nearly the whole language.

An outstanding merit of Peter Naur's report was that it was only 21 pages long. Yet it gave enough accurate information for an implementer to compile the language without any communication with the language designers. Furthermore, a user could program in the language without any communication either with the implementers or with the designers. Even so the program worked on the very first time it was submitted to the compiler. Apart from a small error in the character codes, this is what actually happened one day at an exhibition of an Elliott 803 computer in Eastern Europe. Few languages designed since then have matched such an achievement.

Part of the credit for this success was the very compact yet precise notation for defining the grammar or syntax of the language, the class of texts that are worthy of consideration as meaningful programs. This notation was due originally to the great linguist, psychologist and philosopher Noam Chomsky.

It was first applied to programming languages by John Backus, in a famous article on the Syntax and the Semantics of the proposed International Algorithmic Language of the Zurich ACM-GAMM Conference, Paris, 1959. After dealing with the syntax, the author looked forward to writing a continuation article on the semantics. It never appeared: in fact the article laid down a challenge of finding a precise and elegant formal definition of the meaning of programs, which inspires good research in computer science right up to the present day.

The syntactic definition of the language served as a pattern for the structure of the whole of our Algol compiler, which used a method now known as recursive descent. As a result, it was logically impossible (almost) for any error in the syntax of a submitted program to escape detection by the compiler. If a successfully compiled program went wrong, the programmer had complete confidence that this was not the result of a misprint that made the program meaningless.

Chomsky's syntactic definition method was soon more widely applied both to earlier and to later programming languages, with results that were rarely as attractive as for Algol 60. I thought that this failure reflected the intrinsic irregularity and ugliness of the syntax of these other languages. One purpose of a good formal definition method is to guide the designer to improve the quality of the language it is used to define.

In designing the machine code to be output by the Elliott Algol compiler, I took it as an over-riding principle that no program compiled from the high level language could ever run wild. Our customers had to accept a significant performance penalty, because every subscripted array access had to be checked at run time against both upper and lower array bounds. They knew how often such a check fails in a production run, and they told me later that they did not want even the option to remove the check.

As a result, programs written in Algol would never run wild, and debugging was relatively simple, because the effect of every program could be inferred from the source text of the program itself, without knowing anything about the compiler or about the machine on which it was running. If only we had a formal semantics to complement the formal syntax of the language, perhaps the compiler would be able to help in detecting and averting other kinds of programming error as well.

Role of Semantics

Interest in semantics at the time was widespread. In 1964, a conference took place in Vienna on Formal Language Description Languages for Computer Programming. It was attended by 51 scientists from 12 nations. One of the papers was entitled *The definition of programming languages by their compilers*: it was written by Jan Garwick, pioneer of computing science in Norway.

The title appalled me, because it suggested that the meaning of any program is determined by selecting a standard implementation of that language on a particular machine. So if you wanted to know the meaning of a Fortran program, for example, you would run it on an IBM 709, and see what happened. Such a proposal seemed to me grossly unfair to all computer manufacturers other than IBM, at that time the world-dominant computing company.

It would be impossibly expensive and counter-productive on an Elliott 803, with a word length of 39 bits, to give the same numerical answers as the IBM machine, which had only 36 bits in a word — we could more efficiently give greater accuracy and range. Even more unfair was the consequence that the IBM compiler was by definition correct, but any other manufacturer would be compelled to reproduce all of its errors (they would have to be called just anomalies, because errors would be logically impossible).

Since then, I have always avoided operational approaches to programming language semantics. The principle that 'a program is what a program does' is not a good basis for exploration of the concept of program correctness.

I did not make a presentation at the Vienna conference, but I did make one comment: I thought that the most important attribute of a formal definition of semantics should be to leave certain aspects of the language carefully undefined. As a result, each implementation would have carefully circumscribed freedom to make efficient choices in the interests of its users and in the light of the characteristics of a particular machine architecture.

I was very encouraged that this comment was applauded, and even Garwick expressed his agreement. In fact, I had misinterpreted his title: his paper called for an abstract compiler for an abstract machine, rather than selection of an actual commercial product as standard.

The Axiomatic Approach

The inspiration of my remark in Vienna dates back to 1952, when I went to Oxford as an undergraduate student. Some of my neighbours in college were mathematicians, and I joined them in a small unofficial night-time reading party to study mathematical logic from the text book by Quine. Later, a course in the philosophy of mathematics pursued more deeply this interest in axioms and proofs, as an explanation of the unreasonable degree of certainty which accompanies the contemplation of mathematical truth.

It was this background that led me to propose the axiomatic method for defining the semantics of a programming language, while preserving a carefully controlled vagueness in certain aspects. I drew the analogy with the foundations of the various branches of mathematics, like projective geometry or group theory; each branch is in effect defined by the set of axioms that are used without further justification in all proofs of the theorems of that branch.

The axioms are written in the common notations of mathematics, but they also contain a number of undefined terms, like lines and points in projective geometry, or units and products in group theory; these constitute the conceptual framework of that branch. I was convinced that an axiomatic presentation of the basic concepts of programming would be much simpler than any compiler of any language for any computer, however abstract.

I still believe that axioms provide an excellent interface between the roles of the pure mathematician and the applied mathematician. The pure mathematician deliberately gives no explicit meaning to the undefined terms appearing in the axioms, theorems and proofs. It is the task of the applied mathematician and the experimental scientist to find in the real world a possible meaning for the terms, and check by carefully designed experiment that this meaning satisfies the axioms.

The engineer is even allowed to take the axioms as a specification which must be met in the design of a product, for example, the compiler for a programming language. Then all the theorems for that branch of pure mathematics can be validly applied to the product, or to the relevant real world domain.

Surprisingly often, the more abstract approach of the pure mathematician is rewarded by the discovery that there are many different applications of the same axiom set. By analogy, there could be many different implementations of the axiom set which defines a standard programming language. That was exactly the carefully circumscribed freedom that I wanted for the compiler writer, who has to take normal engineer's responsibility that the implementation satisfies the axioms, as well as efficiently running its users' programs.

My first proposal for such an axiom set took the form of equations, as encountered in school texts on algebra, but with fragments of program on the left and right hand sides of the equation instead of numbers and numeric expressions. The same idea was explored earlier and more thoroughly in a doctoral dissertation by Shigeru Igarashi at the University of Tokyo.

I showed my first pencilled draft of a paper on the axiomatic approach to Peter Lucas; he was leading a project at the IBM Research Laboratory in Vienna to give a formal definition to IBM's new programming language, known as PL/I. He was attracted by the proposal, but he rapidly abandoned the attempt to apply it to PL/I.

The designers of PL/I had a very operational view of what each construct of the language would do, and they had no inclination to support a level of abstraction necessary for an attractive or helpful axiomatic presentation of the semantics. I was not disappointed: in the arrogance of idealism, I was confirmed in my view that a good formal definition method would be one that clearly reveals the quality of a programming language, whether bad or good; and the axiomatic method had shown its capability of at least of revealing badness. Other evidence for the badness of PL/I was its propensity for crashing programs.

Role of Assertions

By 1968, it was evident that research into programming language semantics was going to take a long time before it found application in industry, and in those days it was accepted that long-term research should take place in universities. I therefore welcomed the opportunity to take up a post as Professor of Computer Science at the Queen's University in Belfast.

By a happy coincidence, as I was moving house, I came across a preprint of Robert Floyd's paper *Assigning Meanings to Programs*. Floyd adopted the same philosophy as I had, that the meaning of a programming language is defined by the rules that can be used for reasoning about programs in the language. These could include not only equations, but also rules of inference. By this means, he presented an effective method of proving the total correctness of programs, not just their equality to other programs.

I saw this as the achievement of the ultimate goal of a good formal semantics for a good programming language, namely, the complete avoidance of programming error. Furthermore, the quality of the language was now the subject of objective scientific assessment, based on the simplicity of the axioms and the guidance they give for program construction. The axiomatic method is a way to avoid the dogmatism and controversy that so often accompanies programming language design, particularly by committees.

For a general purpose programming language, correctness can be defined only relative to the intention of a particular program. In many cases, the intention can be expressed as a post-condition of the program, that is an assertion about the values of the variables of the program that is intended to be true when the program terminates.

The proof of this fact usually depends on annotating the program with additional assertions in the middle of the program text; these are expected to be true whenever execution of the program reaches the point where the assertion is written. At least one assertion, called an invariant, is needed in each loop: it is intended to be true before and after every execution of the body of the loop. Often, the correct working of a program depends on the assumption of some precondition, which must be true before the program starts.

Floyd gave the proof rules whose application could guarantee the validity of all the assertions except the precondition, which had to be assumed. He even looked forward to the day when a verifying compiler could actually check the validity of all the assertions automatically before allowing the program to be run. This would be the ultimate solution to the problem of programming error, making it logically impossible in a running program. I correctly predicted this achievement would be some time after I had retired from academic life, which would be in 30 years time.

So I started my life-long project by first extending the set of axioms and rules to cover all the familiar constructions of a conventional high-level programming language. These included iterations, procedures and parameters, recursion, functions, and even jumps. Eventually, there were enough proof rules to cover almost all of a reasonable programming language, like Pascal, for which I developed a proof calculus in collaboration with Niklaus Wirth.

Since then, the axiomatic method has been explicitly used to guide the design of languages like Euclid and Eiffel. These languages were prepared to accept the restrictions on the generality of expression that are necessary to make the axioms consistent with efficient program execution. For example, the body of an iteration (*for* statement) should not assign a new value to the controlled variable; the parameters of a procedure should all be distinct from each other (no aliases); and all jumps should be forward rather than backward.

I recommended that these restrictions should be incorporated in the design of any future programming language; they were all of a kind that could be enforced by a compiler, so as to avert the risk of programming error. Restrictions that contribute to provability, I claimed, are what make a programming language good.

I was even worried that my axiomatic method was too powerful, because it could deal with jumps, which Dijkstra had pointed out to be a bad feature of the conventional programming of the day. My consolation was that the proof rule for jumps relies on a subsidiary hypothesis, and is inherently more complicated than the rules for structured programming constructs.

Subsequent wide adoption of structured programming confirmed my view that simplicity of the relevant proof rule is an objective measure of quality in a programming language feature. Further confirmation is now provided by program analysis tools like Lint and PREfix, applied to less disciplined languages such as C. They identify just those constructions that would invalidate the simple and obvious proof methods, and warn against their use.

A common objection to Floyd's method of program proving was the need to supply additional assertions at intermediate points in the program. It is very difficult to look at an existing program and guess what these assertions should be. I thought this was an entirely mistaken objection.

It was not sensible to try to prove the correctness of existing programs, partly because they were mostly going to be incorrect anyway. I followed Dijkstra's constructive approach to the task of programming: the obligation of ultimate correctness should be the driving force in designing programs that were going to be correct by construction.

In this top-down approach, the starting point for a software project should always be the specification, and the proof of the program should be developed along with the program itself. Thus the most effective proofs are those constructed before the program is written. This philosophy has been beautifully illustrated in Dijkstra's own book *A Discipline of Programming*, and in many subsequent textbooks on formal approaches to software engineering.

Assertions in Use

Assertions are widely used in the software industry today, primarily to detect, diagnose and classify programming errors during test. They are sometimes kept in product code to forestall the danger of crashes, and to analyse them when they occur. They are beginning to be used by compilers as hints to improve optimisation. They are also beginning to be recognised by program analysis tools, to inhibit false complaints of potential error.

The one purpose for which they are hardly ever used is for proof of programs. Nevertheless, assertions provide the fundamental tool for scientific investigation of professional disciplines of programming; and they show the direction for future advance in the development both of programming tools and of programming languages.

There are still many issues that remain as a challenge for future research. And there are encouraging signs that the research is beginning to spread a beneficial influence on practices, tools and languages coming into use; and that this will lead to significant improvements in the quality of software products, for the benefit of their rapidly increasing numbers of users.

Editor's note: This is an edited version of a talk given by the author to the Society at the London Science Museum on 5 June 2001. Sir Tony Hoare is Senior Researcher with Microsoft Research Ltd at Cambridge.

Keeping Old Warhorses in Action

Stuart Fyfe

The Computer Conservation Society was not the first body to rescue and restore computers that were no longer wanted. A group of students in Kingston, Surrey had the same idea a decade or so earlier. They acquired, dismantled, reassembled and recommissioned an ICT 1301, and went on to build a successful bureau business around this spare time activity. One of them tells here the story of Galdor Computing.

In the early seventies, the "Amateur Computer Club" was among the few national groups pursuing what then seemed a very strange hobby. It published a design for a hardwired 4-bit thing called the weeny-bitter. I had already had a go at an 8-bit design published in *Wireless World*, and had made an adding machine using old Strowger telephone parts, but I wanted something better — something with enough power to be actually used for something.

The chance came while I was an engineering student at what is now Kingston University (at the time it was called Kingston College of Technology, until the Minister of Education, one Margaret Thatcher, designated it Kingston Polytechnic in a grand ceremony).

I was one of three or four students who shared a flat over a shop in nearby Surbiton. We thought it would be fun to buy a scrap computer to play with, and let schools, researchers and ordinary people come and use it for their own projects. We wanted to link it by telephone to an amateur astronomical observatory with a big telescope near Croydon. We did a load of weird stuff — like the electric toilet flush that signalled to the world when it was used, and could be operated remotely for giving surprises; and the Meccano rover that ran up and down the pavement of the main road. Later, there was the model railway that ran all round the flat.

We looked at an Elliott 802 with mercury delay line storage, and a PDP whatsit, and got to know the surplus electronics trade and the world of 5-track paper tape, but nothing was quite right — we could not find a proper machine. It was a surprise when I got a phone call and a visit from an ICL engineer, Rodney Brown, who said that London University would be scrapping an ICT 1300 to replace it with an ICL 1904A. It was in their administration building, Senate House. Rodney said it was a bit big, but has everything necessary, and there was scope for adding attachments of our own. Just the job!

We visited Senate House and saw "Flossie" in action. It was serial number 6, the first one out of the factory in 1962, and it boasted the first proper commercial printer. I think my GCE pass-slips were printed by it, as perhaps were some of yours. It was, in fact, the programmable tabulator that Babbage had wanted to build.

In 1970 we had nowhere to put such a thing, and had to find a temporary store while we looked for permanent premises. Apparently we were only competing with scrap dealers for the price of the metal, and the Operations Manager, Mr Hutt, was pleased to see it going for educational use at a price of £200.

We spent a few days camping on his basement computer room floor while we dismantled it. Engineering students seem to enjoy this sort of thing; very odd. We'd studied the manuals and knew more or less what we were doing by then, with help from friendly ICL engineers. The units came apart very easily, after cutting the wire wraps between the bays and unbolting the frames. And it was all on wheels, which was a very civilised arrangement. There was a lot of ironmongery for the kickplates and covers, so we took a few photographs to help the process of re-assembly. We didn't know how long it might be before it could be turned on again: in the event it was a couple of years.

There were a dozen friends and students helping. We hired an articulated lorry, and after a delay while the driver got a new battery for the tail-lift, we trundled the frames along what seemed like miles of corridors in the basement of Senate House, and strapped the units into the lorry.

We unloaded into a double garage in Sutton, where my colleague Trevor Jarrett was living (the last I heard of him, he was selling computers for ICL in Australia). It was a tight squeeze in the garage, but it all went in, and there was a small heater to discourage condensation. That house and garage have been demolished now — they were just across the road from where Harry Secombe lived.

I won't describe the tortuous search for permanent premises, but we had noticed that estates of wartime prefab houses were being dismantled. You could get one of these for £200, delivered in sections, from advertisers in *Exchange and Mart.* It suddenly dawned on us that there was room for one of these prefabs in the back garden of our student flat if we bought the lease; pulled down the workshop of a semi-retired old-fashioned tailor; pulled down another workshop occupied by a printer; got planning consent for light industrial use; and barrowed a couple of lorry loads of concrete down the narrow passage that was the street access.

It was a good day when the building arrived and we tried to work out how to reassemble it. It was a better day when we held a party for all our helpers and friends in the new, empty room. The council had made us add an internal brick wall for sound-proofing, and there was a lot of fuss about whether there really was three-phase mains power in the street outside or not.

By now we had formed the Galdor Electronics Company Limited, trading as Galdor Computing. I and my colleagues Derek Odell and Andrew Keene were still studying electrical and electronic engineering, which included light current and heavy current studies, which was appropriate. Others came to our flat or lived nearby, notably Dirk Koopman, Don Hedges, Chris Hewitt, Adrian Barnes, and later Peter Singleton and John Skeene.

The name *Galdor* is a character in a book that I was reading — “The Lord of the Rings” by JRR Tolkien, which was required reading for students. A funny thing: we asked for a telephone line, and without prompting the number that we were given was 399 1300. I took this as a sign that we doing something right.

Moving the machine to Surbiton in another tail-lift lorry was easy enough, except for one unit that was too wide for the passageway. So we got out a hacksaw and attacked some runner extensions. These runners have to be temporarily attached to the main console, to stop it falling over on its face. This was all done outside, on the pavement, in the main shopping street, at midnight.

The most time-consuming part of re-installation was, of course, reconnecting thousands of wrapped wire links between the bays. We made a jig, and manufactured pre-formed links in three sizes with insulated wire. Instead of wrapping the joints we soldered them. They were colour-coded. We discovered that some corners were best connected by a left-handed person, and others by right-handed persons. Soldering made subsequent moving and reassembly easier for the next owner. There was no risk of making a wrong connection, they were all one to one links; the design was ideal. It was a lovely machine to work on.

Instead of the standard in-line arrangement of printer and console, we angled the printer back about 45 degrees to make it fit in the building. The triangle that this opened up allowed a convenient extension of the console work surface.

The first power-on was not a success: no smoke, but the clock waveform was all over the place. This derived (more or less) from simply amplifying a drum track. We wanted to load some test programs, using an engineer's trick whereby the card punch can be used as a card reader since it incorporates a check-reading station that's normally used to verify the correct punching of holes. The card punch made a horrid noise and jammed, because the motor was revolving in the wrong direction.

All the motors were revolving in the wrong direction, including the drum. The three-phase mains was wrongly connected somewhere, although the colour sequence was right. The fault was probably out in the street, but easily fixed by swapping a couple of phases over. That was better: a decent phase-locked clock signal. Now we had to find where the faulty circuit boards might be, and anything else that wasn't working.

Fault finding was pretty easy, because there were test points everywhere; in fact, it was all test points. The diagrams were faultless and identified every pin and wire. And the test programs were pretty good at identifying the problem area. A measure of ingenuity and imagination was needed to translate the initial symptoms into an identification of the faulty gate or board and, for me, that satisfying process of deduction was what made the project so worthwhile — along with the things we used it for.

With multiple faults masking each other, it did take some months to commission the machine. I remember the mechanical timing adjustments on the printer were particularly tedious. But once it had settled down, a fresh fault occurred only about once a month.

The blue sections on the control panels contained engineer's controls. On the main console, the engineer's section monitored the internal state of sub-systems like the DTU and all the power supplies. From there, the power voltages could be varied, up or down, in an attempt to provoke faulty behaviour before it became a problem in normal use.

Some sections were quite a puzzle, because they didn't go wrong often enough to let us become fully familiar with the circuitry. It could take a day to pin down a subtle fault in the DTU or tape systems, but we would learn a lot by the end of it.

Our test tools were an oscilloscope and a voltmeter. DC and 1MHz were easy to measure. An operator sitting at the console could send specific signals, data values or operations to most parts of the machine, and the effects could be monitored on the gates by a colleague wheeling this enormous oscilloscope from one bay to another. Sitting with the diagrams open at the console, he would call out things like “See if 8G27-5 goes high when I clear CR2”. And then power off, and whip out the faulty card.

Having wire wraps everywhere certainly slowed things down, and it was worth making some additional tests to confirm where the fault was before getting out the special tools. There were rubber strips with a row of holes called “quick wraps” that would hold a card temporarily in place for testing.

This was an ideal engineer-friendly scale of machine. We replaced individual faulty transistors when there were no spare circuit boards of the right type. You couldn’t do that with an integrated circuit or even TTL logic. The mechanical parts were nice to work on too, being well made with heavy castings and oil-filled gearboxes. Lovely stuff.

Having unbuffered peripherals was annoying, but did keep the price down. The main program had to loop and wait for the print barrel to rotate to the next letter that you wanted to print. This could be concealed in modular routines but was hardly efficient.

It was quite a marvel that it worked at all, as evidenced by the measures taken to achieve the clock rate of 1MHz. The many capacitors and coils that populated the circuit cards were there to help these specks of germanium switch at these speeds, and do so reliably when attached to perhaps 20 feet of wiring.

Apart from what came with the machine, spare parts came from dismantling other 1300s wherever we could find them, with our troop of friends and a big lorry. As machines approached the end of life, maintenance must have suffered, for we saw a lot of quick-wraps in place permanently. There was one machine in Bridgewater; a multi-tasking 1302 near Hastings with one inch tapes; another at the office of the Official Receiver on, I think, Millbank, where we were told a bomb on the window ledge did nothing more than dent a panel, and where security insisted on watching while the drum was dismantled and the magnetic oxide washed off, in case it contained sensitive information.

In this way we were able to add a couple of extra drum stores, and bring it up to six half-inch magnetic tape drives, and the full complement of 2000 words of 48-bit magnetic core store. A full set of spares was put by, and we actually swapped the card reader when we were unable to keep the original one running. We added an 8-track paper tape reader and punch, and picked up a variety of punch card preparation machines and verifiers, with real keyboards instead of a finger dance.

Parts had got all over the place. The BBC props department used to have a console, which turned up in *Doctor Who* occasionally.

Modifications were easy to make on the 1301. There were gaps in the instruction codes that could be used. You just had to add some gates to decode the instructions and do something interesting. An approved modification called the "black box" involved installing several racks of boards and let the 1301 drive a 1900 series tape deck, and this was used to help installations that were converting their data. One of these was in Glasgow and had quarter-inch tape which involved a two-stage transfer.

The famous sterling arithmetic instructions were achieved by messing about with the carrying of overflows from one digit to the next, as indeed were the decimal instructions. Since the national currency had been 'decimated' by the time we got the machine, the sterling instructions were redundant, and we made some alterations to the mill to treat the commands as plain binary, which was more useful to us. All changes were documented, and we could revert to sterling by operating a switch in one of the bays.

One practical change was to add some buttons on the console to clear an entire register at one go, without having to dial up a lot of zeroes first. This involved attaching a lot of diodes to the back of the dials to form simple pull-up gates.

We added an instruction to index or modify stored instructions. It added the bottom six digits of the IAS location to CR2 before executing the modified instruction.

Some changes were made to the original fixed program Initial Orders. This let the operator set some manual indicators to do things like unload all the tapes, and a very big help was the invention of the G card for unattended continuous running. When a program was being loaded from punched cards, normally the last card read was an E or entry card, whereupon the processor stopped and displayed all the ones in CR1 waiting for the operator to set the program running. If you overpunched the card's E with a G, it was read as a G, and the modified Initial Orders didn't stop but ran the program at once.

This could save a lot of time. It worked because G has a numeric of 7 which masked the 5 of the E in binary. My colleague Andrew Keene can take the credit for most of this trickery. He also taught Initial Orders to sumcheck program images in IAS/drum memory, and reload it from a library tape only when necessary.

We added an electric typewriter and a keyboard of sorts. This was used, with Initial Orders, to do simple things like type the label details of a magnetic tape. The interface used TTL logic which has a positive ground, so it had to be built "upside down"and floating to attach it to the 1301.

The processor stopped whenever it needed to tell you something, and we weren't going to stand over it all the time. So we ran a line up to the house, isolated with a relay, to flash lights when it was lonely and in need of attention.

I wanted to get it to control the model train that ran around our flat. It got quite hot sometimes, and I quite wanted to add an instruction that could be triggered by temperature to release the door panels and flap them about a bit. As it was, we fitted a chimney or two in the central units and a big extractor fan in the roof space of the pre-fab building that kept it under control.

The commercial three-phase electricity tariff gave us cheap power overnight. A nocturnal lifestyle suited us students, but confused our clients. We had occasional visits from the Amateur Computer Club and a few schools, but most activity was with customers who appreciated our low overheads.

We wrote a program to look after club and society membership lists. It was called Clubsoc, and was used by some 35 organisations eventually. Nowadays they'd do it all themselves with PCs, but in the mid seventies a cheap bureau service was ideal.

It started with London Village, and went on to include the UK administration sections of Amnesty International and of Friends of the Earth; the Town and Country Planning Association; the National Council for Civil Liberties; the Legalise Cannabis Campaign; and Vole Magazine, to name just a few. The data was punched onto cards by a separate bureau, and we would produce lists and mail-out labels.

We also produced printed biorhythm charts for a client, and later on processed the complex administration for the Southern Counties Cat

Fanciers shows.

An attempt was made to generate school timetables by machine, but the complexities of split sites and individual pupils' requirements meant that this project was not a real success.

The programs were written in machine code and assembler. The relativised loader is an excellent way to combine separate modules, and it was good training in how language compilers ultimately work. The assembler was extended by us to allow symbolic names and comments.

We did a six month period of continuous contract operating in two 12-hour shifts with the 1301 machines at the Liverpool Victoria Friendly Society. Their machines were Samantha and Arthur and survived a Y2K-type crisis when interest rates went over 10 per cent and needed two digits. We used the G card, and told them that Initial Orders had always had this capability.

We did enough like this to keep the business running, and kept on buying scrap machines, progressing through the 1900 series, which contained useful amounts of reclaimable gold. We moved on to Cobol and George 3 and to bigger premises.

The 1301 was of a size where it felt just possible that one person could get to know everything about it, hardware and software, if they had to. Somehow that's important. It feels uncomfortable using modern PCs and stuff, where one only has the roughest idea of what it's up to. It doesn't matter; it just feels uncomfortable. Working around Flossie was one of the most satisfying things I've done, and much credit goes to the designers.

Editor's note: this article is based on a talk by Stuart Fyfe to the Computer Conservation Society on Thursday 20th April 2000.

Punched Cards Are Not Dead Yet

Hamish Carmichael

It was an odd enquiry, but intriguing: did the Computer Conservation Society know of any place where punched cards could be reproduced? There were only two possible answers. The first, truthful but boring, was “No”. The other was a rapid fire of return questions: Who wants to know? How many cards? 80-column or 40-column? How urgent? A one-off requirement or a regular job? Is there any money in it? Readers can easily invent lots of others.

The “who?” was easy — he was an academic in Edinburgh. Which made one other answer equally obvious — No, he didn’t have any money. Why was he asking? This is where it gets interesting.

Twenty years ago a chap called Rex Sawyer wrote a book called “Pollen Identification for Bee Keepers”, which was edited and published by a Dr RS Pickard of University College, Cardiff. It was based on the argument that you can identify grains of pollen, and hence deduce what the bees have been feeding on, if you observe such characteristics of a pollen grain as Size, Shape, Colour, Aperture numbers, Aperture types, Surface, Exine (whatever that is), and Other features. The book was very well illustrated, with photos and tables, and told you how to judge actual pollen examples against the tables. Each identified pollen was given a unique number.

Then, to make the process almost infallible, the book was accompanied by a pack of 50 punched cards. Each card was printed with 20 columns of numbers, from 1 to 200, allowing the recording of the details of up to 200 types of pollen (only 150 are needed to identify all the pollens found in British honey). Each card represented one value of one of the identifying characteristics.

For example, there was one card for “Size — Very Small”, which was punched with holes alongside all the numbers of the pollens which share that characteristic. Thus if you want to identify a pollen which is Very Small, Yellow, and Triangular, you pick up the relevant Size, Colour and Shape cards, hold them up to the light, and you can see that it’s either pollen number 73 or pollen number 94. Usually three cards are sufficient. Occasionally, as in this example, a fourth characteristic is used to find a unique identification.

The Moir Library in Edinburgh is believed to have the most comprehensive collection of books about bee-keeping anywhere in the world. It therefore had four copies of the Sawyer/Pickard book, but all the accompanying punched card packs had gone missing. The enquirer had one master pack, which he was not prepared to entrust to the post, but he did send me xerox copies of each card against a black background.

He wanted four replacement packs, plus two spares. At 50 cards per pack, that meant 300 cards in total. It was not worth refurbishing a reproducer for that lot, even if one could be found. So out with the old hand punch, and grudgingly extract some of the blank cards from one's lifetime supply of bookmarks, lecture notes, and shopping lists.

The enquirer thought he would be able to get the pollen numbers printed onto the cards by some friend in Edinburgh, so the punching was done onto 'blank' cards with the printed side down.

Just to make it more fun, the punching only occurred in every third column, starting at column 9 and continuing to column 48. And what we know as positions 0 to 9 were printed 1 to 10 on the master cards. And there could be multiple punches — up to 10, in fact — in the same column, so hold one key down until you've punched the next, which was not what we were taught in UAKP training in 1958.

Verifying was easy: lay the newly punched card on top of the xerox master. If all the holes filled with black, all was well, because clearly the new holes all corresponded to holes in the master card. Any discrepancy — ditch the card and punch another.

I won't tell you how many days it took, or how many cards were mispunched in the process. But I do hope he doesn't extend the system to the pollens found in foreign honeys.

CCS Web site information

The Society has its own World Wide Web (WWW) site: it is located at **http://www.cs.man.ac.uk/CCS/**. This is in addition to the FTP site at **ftp.cs.man.ac.uk/pub/CCS-Archive** (please note that these URLs are case-sensitive). Our Web site includes information about the SSEM project as well as selected papers from *Resurrection.* Readers can download files, including issues of *Resurrection* and simulators for historic machines.

Zuse Z3 replica is operational

Martin Campbell-Kelly

The project to rebuild Konrad Zuse's Z3 relay computer is now complete, and the machine was formally inaugurated at a symposium entitled *Sixty Years of Computation: Konrad Zuse's Computing Machine Z3: 1941 - 2001* held at the Konrad-Zuse-Zentrum für Informationstechnik, Berlin on 11 May 2001.

The Z3 rebuild team was led by Raul Rojas of Technischen Universität Berlin. Unlike most reconstruction projects the new Z3 uses modern relay technology instead of authentic components. As a result the machine is faster, smaller, and more reliable than the original, while still capturing something of the authentic feel of a relay machine.

The symposium itself was opened with a welcome address from Peter Deuflhard, Director of the Konrad-Zuse-Zentrum. The first paper was read by K Vollmar (Universität-Karlsruh), who recounted the history of Konrad Zuse's commercial ventures and companies. The next speaker, Horst Zuse (TU Berlin), son of the late Konrad Zuse, gave a detailed history of Zuse's machines from the pre-war Z1 up to the commercially manufactured Z11 of 1955. His presentation was complemented by some charming history and photographs of the Zuse family.

Raul Rojas then gave an architectural overview of Zuse's computers from the Z1 to the Z4. His presentation included a demonstration of the reconstructed Z3 in which LEDs, fired by relay operations, enabled the movement of data and the contents of the storage unit to be visualized.

The symposium concluded with an international perspective from two English-speaking computer historians. Michael Williams (University of Calgary) described the evolution of calculating instruments, providing a context for German developments, while Martin Campbell-Kelly (University of Warwick) spoke on Charles Babbage's Analytical Engine. The symposium was followed by an alfresco lunchtime aperitif in perfect spring weather. During the afternoon CCS Vice-Chairman Tony Sale gave a well attended talk on the reconstruction of the Colossus.

Editorial contact details

Readers wishing to contact the Editor may do so by fax to 020 8715 0484 or by e-mail to <NEnticknap@compuserve.com>.

In Memoriam ICL

Hamish Carmichael

When ICL was formed in 1968, one of the earliest interpretations offered for the new acronym was 'It Can't Last'. (Stand up the boy who said that that originated in IBM!) Now, 33 years later, the name is indeed coming to an end. But that in itself is no great pity. Many of us who served in ICL in its heyday have felt that the spirit of the company itself has been dying over the past few years, and that the organisation which once represented the sum total of the native British computer industry has been allowed to dwindle into a pathetic shell.

I find a common feeling among many former colleagues who have retired or been redunded or have just got fed up and left of their own accord. Better to be away from it, with our proud memories intact, than to stay propping up its remains.

What a heritage it was — Hollerith, Powers, Ferranti, Elliott, Leo, English Electric, EMI and all the others. We all brought into ICL our own pride and our own loyalties, yet these melded into a common pride that was greater than them all, and we all became glad to be known as ICL people. Nevertheless after years of joint service you could still often tell from which tradition someone derived. The last embers of the rivalry between Hollerith and Powers may now at last have cooled, but it was only a little earlier this year, after a committee meeting (nothing at all to do with ICL) that a friend and I agreed "that was rather an English Electric occasion", and both knew at once what we meant.

Think of the techniques we pioneered — multiprogramming, store protection, paging and virtual storage to name but a few — typically years ahead of the Americans. And technical innovation was an abiding strength: think of DAP, the first practical parallel processor, and Cafs, for so long my speciality (and hobby horse).

It's still probably too soon to write the story of the 2900 and its successors, and VME, the operating system that started off pretty disastrously but grew into the best the world has seen. Nowadays, when 'operating system' seems to mean a pile of incompetent junk scrambled together by nincompoops, an inviting playground for the world's hackers, with 'security' about as resilient as cold rice pudding, we know, indeed we know, that we knew better. And behind the world's curtains VME still does a fair share of the world's work.

Having fought shy of management responsibility all my life, I'm not the one to judge or criticise the managers who have presided over the company's decline, but there's one general point I do feel qualified to make. The company's strong periods have been when among those at the helm there has been a strong understanding of the industry, and an engineer's awareness of the challenges and opportunities. You can't run a computer company with a combination of marketeers and planners and accountants.

So farewell ICL. It Couldn't Last — but it was good while it lasted.

CCS Collection Policy

The Committee of the Society has formulated a policy statement concerning procedures for dealing with computers of historical interest that come to the Society's attention. This is published in full below.

1. The Society has no Collection of its own, and no premises in which to house one. There is no intention to change this.
2. When the Society hears of historic equipment which is becoming available for conservation, it will attempt to find a suitable home for it in one of the following major collections:
    - The Bletchley Park Museum Trust
    - The Science Museum, South Kensington
    - The Museum of Science and Industry, Manchester
3. The Society will also alert other collections to the availability of surplus equipment, where the major collections are unable to offer to house it, if it fits the appropriate area of interest. Members who know of such collections are asked to ensure that the Secretary is aware of their location and subject matter.

Society Activity

Small-Scale Experimental Machine Project

Chris Burton

The machine continues to be shown off on Tuesdays by a loyal band of demonstrators. It continues to be fairly reliable, though now, three years after installation, it could do with a maintenance blitz. Work is proceeding on design and build of a Cathode Ray Tube test rig to sort out which CRTs are likely to be most reliable as storage tubes. We now have about 50 CRTs in our spares holding. The bulk of the spares were purchased by courtesy of ICL.

Bombe Rebuild Project

John Harper

At last, we have visible progress. Towards the end of May 2001, we were able to fit a DC motor, clutch and gearbox into the bottom of the frame at Bletchley Park. Using our DC power supply we were able drive the gearbox and activate and deactivate the clutch. The new gears ran very smoothly and soon bedded in. To the casual observer this milestone may not seem very inspiring. However, when one realises that with the exception of ball bearing races everything was made from raw material, one will have a better understanding of the effort involved. For example, castings started as billets of cast iron, gears and clutch items were cut from basic material and coils were wound from scratch. Twenty members of our Rebuild team were involved one way or another in achieving this milestone.

Our immediate objective is to have all mechanical parts turning under DC power in the near future. As I write at the end of October 2001, 99% of all the mechanical parts required are manufactured and delivered. The remainder is all in hand and is expected to be with us within the next few weeks.

Mechanical assembly is well under way but before we start mounting too many parts on the full frame, we are assembling a smaller test rig consisting of four Letchworth Enigmas in a single bank. The purpose of this is to carry out assembly and fitting exercises at locations more convenient than Bletchley Park. As I write, this rig is about 75% assembled. Valuable lessons are being learnt about the order of assembly. These are expected to save a great deal of time when we assemble 36 Letchworth Enigmas on the full frame. Current estimates are that the test rig will be completed before the end of this calendar year, and the full set assembled at Bletchley Park early in 2002.

Looking forward to the 'electrical' phase, long lead items such as the cableforms are making good progress. It is estimated that the last one will be laid up and laced early in 2002. No terminations have been fitted yet. This we expect to take place during and before the end of 2002. New coil winding is progressing slowly but surely, as are many other electrically related items.

Our general plan now is to demonstrate in the new year, to all those who are interested, the mechanical parts more or less complete. This we feel is the appropriate time to carry out a major fund raising activity. We do not have sufficient funds to complete the Rebuild using full commercial manufacturing facilities. The home workshop facilities that we have used so far are not appropriate for much of what still has to be done on the electrical side. We either need to find companies who will make large numbers of parts on their CNC or automatic machinery or alternatively raise sufficient funds to have these operations carried out at full commercial rates.

However, we do not have to wait until we have demonstrated the mechanical parts working before we start fund raising or the application of CNC machinery. If anybody can help now we have the drawings and automation files ready to send to those who might be prepared to assist.

As before, any readers who feel they would like to help or find out more about our Project can find our details inside the back cover, or via our Web site at: <www.jharper.demon.co.uk/bombe1.htm>.

Elliott 401 Working Party

Chris Burton

No significant work has ben done recently due to the pressure of other activities on members of the Working Party. We have room for two or three more engineers who might like to work on this historic valve computer. Requirements are: availability to work on an occasional week day; ability to get to Blythe House in West Kensington; a confident understanding of electronics and computer logic (familiarity with valve electronics would be a bonus); keen sensitivity to the need for curatorial care of the object;

an email address. If this sounds like you, introduce yourself to me at <cpb@envex.demon.co.uk>.

Mil-DAP Working Party

Brian Russell

We have finished the 'dirty work' of physically cleaning up the machine. After one small hiccup, all the equipment has now passed the tests for electrical safety.

With the help of the readers of *Resurrection*, we have now tracked down three K110 Array Boards. All have been used for display purposes and are therefore likely to be non-functional, but may be repairable or usable with restrictions. One of the boards is now in our possession. After cleaning and fitting with an airseal and metalwork, it will be plugged in.

I would like to thank all who responded to our request for 8-inch floppy discs. The discs are not as rare as we at first thought. We now have over a hundred! This should be sufficient for all our needs; and we know whom to contact if we do need any more. It is most gratifying to get support from so many CCS members.

The machine is still at West Gorton. It has been moved out of the 'dirty' workshop into the laboratory. This would not have been possible only 12 months ago, as the laboratory was then very much in use. With the company shifting its emphasis away from hardware, the lab is often now to be found totally deserted. We have cleared a bench for ourselves and set up the Mil-DAP and our two Perqs ready to begin the software resurrection stage.

We got ourselves a copy of POS (the Perq operating system), but have since discovered that what we need is 'PNX for Perq-2' (the Perq version of Unix, for Type 2 workstations). We are still missing the HCU-CP, the MCU-CP and the DAP Device Driver.

There are still only two of us working on Mil-DAP, Bob Whittaker and myself. Over the winter months, we hope to make a start on running some of the Perq software. Any thought of trying to communicate with the DAP itself is still some time off.

Letters to the Editor

Dear Editor,

I wonder if any *Resurrection* readers were at the historic occasion when the Prudential officially commissioned its first computer, an IBM machine, in I think late 1961. I call it historic because of the Chairman's remark that he could not possibly see any future for such equipment. When the cost of installing a false floor and air conditioning in the huge High Holborn building was taken into account, the economics were then, perhaps, dubious. I would be interested if anyone could tell me the number of the IBM machine. Previously, I had been on the IBM team maintaining the punches and verifiers used to convert the hand written information onto cards.

Best regards,

Eric Russell

by email

14 May 2001

Dear Editor,

I write in reply to Michael Bramsom's query about a 10x10 simultaneous equation solver in the last issue of *Resurrection.*

One possibility is that the machine described was the Mallock Machine, although it would have been very dated by the late 1940s. The Mallock Machine was designed by RRM Mallock, a demonstrator in the Cambridge University Engineering Department. It was actually built by the Cambridge Scientific Instrument Company in 1933. After the war it was housed in the Cambridge Mathematical Laboratory until computer developments made it obsolete.

The machine used a series of interconnected transformers to represent the unknowns in a system of linear simultaneous equations. Coils on the transformers represented the coefficients in the equations: the number of turns on the coils representing specific coefficients. Coils on separate transformers but relating to the same equations were interconnected to form a closed circuit. If the problem had six unknowns then there were six closed circuits. Alternating current was applied to the transformers and when the voltage levels reached an equilibrium the solutions to the equations were read from the machine.

As far as I know the machine was not very successful — although it was mentioned in several contemporary articles. The machine could not adequately deal with ill conditioned equations, letting out a very sharp whistle when equilibrium could not be reached.

Mallock's original article describing the machine is given in *Proceedings of the Royal Society A, Vol 140, 1933, pp457-483.*

If it were not this machine it could have been a Network Analyser of the kind used by electricity companies to simulate power networks, which could also be used to solve simultaneous equations.

If you find out that it was another machine I would be very interested to hear about it.

Yours sincerely,

Mary Croarken
Sackler Fellow
Centre for Maritime Research
National Maritime Museum
Greenwich, London
by email
16 May 2001

Dear Editor,

Hugh McGregor Ross's article on the Ferranti London Computer Centre contains some errors of fact relating to AV Roe, the aircraft company. This company was founded by Alliot Verdon Roe (not Arthur Vernon Roe) and his brother Humphrey Verdon Roe, and was registered in 1910. Alliot Verdon Roe sold his interest in the company in 1928, and went on to found Saunders Roe in the Isle of Wight.

AV Roe & Co later became part of the Hawker Siddeley Group. If BV Bowden did speak to anyone at Avro about the advantages of installing a computer, it was most likely Sir Roy Dobson, who was head of Avro at that time.

David S Wilde
by email
18 May 2001

Forthcoming Events

Every Tuesday at 1200 and 1400 Demonstrations of the replica Small-Scale Experimental Machine at Manchester Museum of Science and Industry

Every weekend Guided tours and exhibition at Bletchley Park, price £3.00, or £2.00 for concessions

Exhibition of wartime code- breaking equipment and procedures, including the replica Colossus, plus 90 minute tours of the wartime buildings

5-6 November 2001 Leo 50th Anniversary Conference

to be held in the Guildhall, City of London, price £75.00: see News Round-Up

27 November 2001 NWG meeting titled "Ferranti Orion 1: An Introductory Overview"

Speaker Chris Burton

22 January 2002 NWG meeting titled "Altair and After: The Original Personal Computer Revolution"

Speaker Robin Shirley

12 February 2002 NWG meeting titled "The RM machine"

Speaker John Leighfield

19 March 2002 NWG meeting titled "The Turing Bombe Rebuild Project"

Speaker John Harper

North West Group meetings take place in the Conference room at the Manchester Museum of Science and Industry, Liverpool Road, Manchester, starting at 1730; tea is served from 1700.

Queries about London meetings should be addressed to George Davis on 020 8681 7784, and about Manchester meetings to William Gunn on 01663 764997 or at <bengunn@compuserve.com>.

Committee of the Society (members)

Point of Contact

Readers who have general queries to put to the Society should address them to the Secretary: contact details are given on the page opposite.

Members who move house should notify Hamish Carmichael of their new address to ensure that they continue to receive copies of *Resurrection.* Those who are also members of the BCS should note that the CCS membership is different from the BCS list and so needs to be maintained separately.

Resurrection is the bulletin of the Computer Conservation Society. Copies of the current issue are available from the Secretary for £5.00 each.

Editor – Nicholas Enticknap
Typesetting – Nicholas Enticknap
Typesetting design – Adrian Johnstone
Cover design – Tony Sale
Printed by the British Computer Society

©Computer Conservation Society

Committee of the Society (Officers)