Atlas

BUSINESS HISTORY, VOL. 26, 1984

PROLONGED NEGOTIATIONS: THE BRITISH FAST COMPUTER PROJECT AND THE EARLY HISTORY OF THE BRITISH COMPUTER INDUSTRY

By JOHN HENDRY

This paper stems from a merging of two historical interests, one in the strategies adopted for the exploitation of technological invention and innovation, and the other in the development and handling of government–industry relationships. The development of the British computer industry in the post-war years, in common with that of other 'high-technology' industries, was both sponsored and in part directed by government agencies. Again in common with other high technology industries, it was not exactly noted for its success. The question is, in crude terms, what went wrong – and could it have gone right?

This question has of course been answered many times and by many people. But for the most part the answers given have been based on statistical generalisations or rather vague theoretical ideas. They have not been supported by analysis at the micro-level of individual attitudes and decisions, nor have they been sensitive to the very real problems faced by individual decision-makers. To a large extent the analysis presented here suffers from that same fault, for much of it is based on readily available material rather than on a full programme of detailed historical research. Nevertheless, the analysis does penetrate, however imperfectly, to the level of individual men, the problems they faced, the arguments they put forward, and the decisions they made, and it does give an indication of what may perhaps be achieved by more detailed study in this vein.

That an analysis of this kind might be possible for the British scientific computer industry (as opposed to the business computer industry) was made apparent to me by a reading of P. Drath's Manchester University Ph.D. thesis, 'The Relationship between Science and Technology: University Research and the Computer Industry, 1945–1962' (1973), on which I have drawn heavily here, and by conversations with a number of people who were concerned with computers in that period. The present paper owes more to the recollections of these pioneers than to the records of the Atomic Energy Authority establishments, in which they worked. However, as these establishments were the principal customers for the industry their archives contain valuable documentation, and I am very grateful to the Authority for permission to draw on some of this material. Needless to say, the Authority, and members of its staff, are not responsible for the use made of the material or the views expressed by the author.

The Early Years of the British Computer Industry

By 1949 the basic concept of the modern electronic stored-program digital computer was well established and, following publication of much of the thinking behind the American Moore School EDVAC project, well disseminated. There were, moreover, a number of computers in operation, with Britain and the USA running roughly abreast in terms of technological development, and with information passing relatively freely between the pioneer projects. Three of the pioneer machines, the Eckert–Mauchly BINAC in America and the Manchester University MADM and Cambridge University EDSAC in Britain, ran their first programs within a few months of each other in the spring of 1949, and each has some claim to being the first stored-program electronic computer. Other projects bore fruit soon after, including the APEXC at Birkbeck College London and the Pilot ACE at the National Physical Laboratory in Teddington.

Apart from having its fair share of prototype machines, Britain was also the first country to produce a commercial computer, with development again proceeding roughly abreast of the USA. In 1949 the electronics manufacturer Ferranti, faced with a post-war depression in the market for radar and electronic instrumentation, looked around for a new field of activity and set up a computing department to manufacture a development of the Manchester University computer. F.C. Williams, who had led the Manchester project, had not in fact welcomed the Ferranti overtures. But after the intervention of Patrick Blackett, then Professor of Physics at Manchester, Ben Lockspeiser of the Ministry of Supply had lent his weight to the collaboration and arranged for financial support of a joint Ferranti–Manchester University venture in the form of a Ministry of Supply contract. Williams was reluctantly won over and the first Ferranti Mark 1 computer was commercially installed in February 1951, just four months ahead of the first American UNIVAC 1.

A second commercial development of 1949 was a decision by Elliott Bros. to enter the computer field, sponsored by the Admiralty, but developing their own relatively low-power computers from scratch. In 1953 Elliott Bros. became the second British firm to install a commercial computer, although the scope of their project was limited and a few years later they closed down their own computer design section, going into league with the American giant NCR. Meanwhile a number of other companies also entered the field. British Tabulating Machines developed the Birkbeck College APEXC and installed their first BTM 1200 in 1955. The catering firm J. Lyons developed the Cambridge EDSAC, completing a prototype LEO I in 1953 and installing their first commercial LEO II in 1958. English Electric worked with the National Physical Laboratory and installed their first DEUCE, an updated and engineered version of the Pilot ACE, in 1955.

Clearly the commercial development of computers in Britain was not neglected. But apart from the Ferranti and Elliott projects it was not

pursued very rapidly, and neither of these firms was in a position to compete with the large American corporations who entered the field at about the same time as they did. This situation arose, moreover, in spite of very early and considerable attempts by government agencies to sponsor the establishment of the industry.

Again the crucial developments took place in 1949. The first was the setting up following the 1948 Development of Inventions Act of the National Research and Development Corporation (NRDC) with Lord Halsbury as managing director and with the functions:

a) of securing, where the public interest so requires, the development or exploitation of inventions resulting from public research, and of any other invention as to which it appears to the Corporation that it is not being developed or exploited or sufficiently developed or exploited.
b) of acquiring, holding, disposing of and granting rights (whether gratuitously or for a consideration) in connection with inventions resulting from public research and, where the public interest so requires, in connection with inventions resulting from other sources.

The NRDC was set up to meet precisely the situation that had arisen with computers. The work at Elliott Bros. was being sponsored by the Admiralty and that at Ferranti by the Ministry of Supply. The Pilot ACE project was being conducted within the National Physical Laboratory, and the Ministry of Supply was also sponsoring research at the Telecommunications Research Establishment (TRE) at Malvern. A large slice of computer development research in Britain was thus being financed by the Government, who controlled the majority of the relevant patents, and this made computer development a natural choice of activity for the new corporation.

The second development was the setting up in October 1949 of the Brunt Committee to oversee computer developments. This resulted from an initiative due largely to Lockspeiser, who had moved in 1949 to the Department of Scientific and Industrial Research (DSIR), and was a joint committee of the DSIR and the Advisory Council on Scientific Policy.

Of the two new bodies the NRDC in particular promised great things. Even before they entered the picture the degree of government sponsorship of computer development was considerable, but it was limited fairly strictly to the satisfaction of defence needs, and hampered by the familiar accoutrements of Civil Service bureaucracy. The NRDC was also restricted, in particular by the requirement that it be self-financing with only a £5 million loan, over a mere five years, for capital. But it was at least commercially orientated and free from the massive hierarchies of the large ministries. From the very beginning, however, the story of its involvement with the computer industry was one of dissatisfaction on both sides.

At first the NRDC seems to have tried simply to persuade the main punched-card machine firms (BTM and Powers–Samas) and electronics firms (AEI, Elliott Bros., English Electric, EMI, Ferranti, GEC and Plessey) to enter the industry, without offering any financial assistance. But apart from Elliott Bros., who were interested only in low-power computers and not in the high-power ones NRDC were seeking to have developed, and Ferranti, who had their hands full with the Manchester project, none of the firms seem to have been convinced at this stage that the rapid development of computers was a commercial proposition. Although this seems short-sighted in retrospect, it is not perhaps surprising. D. Hartree, one of the leading authorities on computational techniques, had recently forecast that a single Pilot ACE would be enough to handle the entire computational needs of Britain. William Penney, who was as director of the British atomic weapons programme the man with the greatest computational needs in the country, agreed with this assessment, and enquiries made of potential computer users through the Federation of British Industries led to similar conclusions. Even in America, where the potential market was substantially greater, and where a number of firms had been engaged in research projects of their own on computers, commercial interest was still effectively limited to the company set up by Eckert and Mauchly and taken over in 1950 by Remington Rand. Since that takeover had been necessitated by the original company going bankrupt as development and production costs mounted to several times those allowed for, this particular activity was scarcely encouraging. IBM only decided to enter the field at the end of 1950, and then for reasons that had more to do with the Korean War than the long-term commercial viability of computers.

The attitude of British firms in 1949–50 was thus quite understandable, and it was rather the enthusiasm of the NRDC that was remarkable. But by the end of 1950 there were clear signs of interest picking up in the USA. With most potential British manufacturers still uninterested, NRDC decided in January 1951 to sponsor further development and production at Ferranti and Elliott Bros. In particular they proposed to advance a proportion of the capital required for the production of ten machines by each firm in return for a share of the profits made on the sale of such machines.

The idea of receiving financial support from the NRDC was naturally welcomed by both firms but, as Drath has recorded for the Ferranti case, the achievement of arrangements satisfactory to both parties proved difficult. Ferranti, writing to the NRDC in February 1951, sought a small but straightforward contribution of about £50,000 to their research and development costs. The NRDC on the other hand, bound by the requirement that they be self-financing, insisted on recovering their investment from future computer sales. It eventually took them nine months to formulate their offer, and it was then unacceptable to Ferranti: they could borrow the money from the bank more easily.

After this first failure, NRDC did agree to Ferranti's next suggestion,

that they act as guarantors for the Mark 1 computer by purchasing four of these on a 'cost plus' basis for resale through Ferranti as agents. Ferranti were able to offload much of their risk, while NRDC stood, if the project were successful, to make a healthy profit. This was the formula on which the concept of NRDC was based and it worked well for the Mark 1 computer, both Ferranti and NRDC showing a profit. However, when the agreement was extended to cover the updated Mark 1* computer in 1953, the NRDC took a sufficient loss to cancel out their former gain. And on the next Ferranti computer, the PEGASUS, both sides suffered heavily. The PEGASUS was a small package computer designed by Strachey for NRDC on the basis of the Elliott 401, which had also been financed by NRDC but the development of which had been wound up following the disintegration of the Elliott team. In 1954 NRDC placed a contract for nine PEGASUS machines, but although the computer was a reasonable success (over 30 were sold) the costing was based on insufficient information and they ended up losing well over £200,000. In this case Ferranti reimbursed NRDC for their loss, but the episode led to increased caution on the part of NRDC, and to a mistrust of Strachey on the Ferranti side.

The next Ferranti project, the MERCURY, was again based on a Manchester design, this time the Mark 2, and was for a relatively sophisticated and potentially reliable computer that appealed considerably to Lord Halsbury. Ferranti had run into trouble with the interpretation of the DSIR (formerly Ministry of Supply) contract for work at Manchester, which had been terminated on the advice of the Brunt Committee on the grounds that the work being done was not directed to specific DSIR needs. They were therefore keen to have NRDC support and in August 1954 an agreement in principle was reached for NRDC to finance the development, estimated to cost £450,000, of MERCURY. But NRDC insisted on recovering their proposed investment in the manner they had suggested before, by a levy on all future Ferranti computer sales. In the course of the subsequent negotiations EMI, whose chairman Joseph Lockwood was also on the NRDC board, did in fact sign a contract with NRDC based upon similar conditions. However, Ferranti could see no sense in the scheme and in September 1955 they decided to go it alone. This time, however, arguments developed between the Manchester designers and the Ferranti engineers, and the effect of the collaboration was once again to store up problems for the future.

Meanwhile, as a result of the 1954 Development of Inventions Act, the NRDC terms of reference had been loosened so as to allow it to play a more active role than hitherto. Rather than increasing its willingness to take large financial risks this seems if anything to have had the opposite effect. The new terms of reference allowed NRDC to take a direct part in the projects with which it was concerned, and this was seen as a way of controlling the projects and so securing the investment. The manufacturers did not want this sort of interference, however, and beyond the PEGASUS fiasco NRDC's only further role in computers before 1957

was the small contract with EMI. It was during this very period, however, that the vulnerability of the British computer industry to American competition became apparent.

Although the American challenge operated across the board of computer types, concern focused on computers for scientific use such as those made by Ferranti, the production of which required a high capital and high risk investment. The most important British user of such computers during this period was the Atomic Weapons Research Establishment (AWRE) of the Atomic Energy Authority at Aldermaston. The AWRE computer experts had, together with their colleagues from the Atomic Energy Research Establishment at Harwell, kept a close watch on the world-wide development of computers, and their experience was instructive. One of the AWRE experts, Glennie, had been closely involved with the development and programming of both the Cambridge and the Manchester computers, and on the basis of his experience the AWRE had turned down a Manchester offer of the first Mark 1 on the grounds that it was too unreliable. They had preferred instead to wait for the Mark 1*, and had taken delivery of one of these in early 1955. As anticipated this had proved very reliable indeed. However, they quickly found out that their earlier estimates of computational needs had been grossly understated, and that the Mark 1* was totally inadequate, even though supplemented by time on the National Physical Laboratory Pilot ACE and by a large punched-card installation. As an immediate measure they ordered one of the new English Electric DEUCE computers and hired time on another, but they were still an order of magnitude short of their requirements. Although by this time the MERCURY project was under way, even this computer looked unlikely to come anywhere near meeting their needs. In America, on the other hand, IBM were developing their 704, and Univac their 1103A.

By 1956 there were 40 computers installed in Britain, a large proportion being the small Elliott machines. In the USA the large and expensive UNIVAC 1 alone was produced in greater numbers. IBM had installed their first 701 in December 1952, and other manufacturers had soon entered the field. With a 32,000-word ferrite core memory (compared with the MERCURY's 1024) and eight magnetic tape drives the IBM 704 promised to be an order of magnitude more powerful than the MERCURY, and seemed to fill the AWRE requirements precisely. Moreover, despite the discrepancy in specification, it was scheduled for production before the MERCURY. It was of course much more expensive, and its development had not been trouble-free. But having kept themselves closely informed of its progress and of the performance of its predecessors, the AWRE staff had no doubts that when produced it would be both reliable and efficient. They duly ordered one. It was delivered ahead of time in February 1957 and worked perfectly thereafter. Harwell, on the other hand, unable to draw on urgent defence requirements to justify dollar expenditure, went for a MERCURY.

Ordered in 1955 it was delivered well behind schedule and was not operating satisfactorily until early 1958.

The contrasting experiences of Harwell and the AWRE were no reflection upon the Ferranti company, for although there had been tensions between the firm and the university on this project the true problems lay elsewhere. Given their massive resources and encouraged by large and firm government contracts, IBM could design and manufacture a prototype, sort out any problems, and begin a production run even of the expensive 704 all within a couple of years. Ferranti, with less men, less money, and less concrete encouragement, had to struggle even to develop the prototype of MERCURY. So far as teething problems were concerned their production models were little better than prototypes themselves.

The implications for the British scientific computer industry were severe for, quite apart from the discrepancy between the existing IBM and Ferranti models (a discrepancy that was matched by American superiority in the field of business computers), there was the fact that whereas there were no British computers on the drawing board to succeed MERCURY, IBM and other American firms were already planning sequences of new models. Beyond the IBM 704 there were the transistorised 7040, the still larger 709 and 7090, and above all the 7030 or STRETCH.

Control Data Corporation (CDC), formed by a group of engineers from Remington Rand and rapidly becoming technological leaders in the big computer field through the design work of the brilliant Seymour Cray, were developing their transistorised 1604 and 3000 and 6000 series. Univac were working towards their LARC. STRETCH, LARC and the CDC 6600 were all advanced computers of what then seemed massive power. Preliminary work on all three was already under way, and both the STRETCH and LARC projects were being heavily funded by the American government through the Los Alamos and Livermore atomic energy laboratories respectively. How was Britain, with her more limited resources and a relatively underdeveloped computer industry, to respond?

The Conception of the British Fast Computer

At the end of 1956 the large American projects were still in their early stages, but a preliminary specification of the IBM STRETCH had been published at the Joint Eastern Computer Conference and it was already clear to many of the computer experts in Britain how the situation was developing. The AWRE, whose IBM 704 was to be installed in a few months' time, had been kept closely in touch with developments at IBM. In the summer of 1956 a team of experts from elsewhere – A. S. Douglas of Cambridge, D. W. Davies and J. H. Wilkinson from the National Physical Laboratory, and Jack Howlett from Harwell – had visited American computer companies and installations. There was a growing

awareness on all sides that Britain was on the point of falling well behind in, if not altogether out of, the computer race, and suggestions as to what might be done about this came from two quarters. In January 1957 the Computer Sub-Committee of NRDC took note of the report of the previous summer's American visit, and requested Lord Halsbury to review the field and to consider the possibility of a British research and development programme to match that in the USA. One suggestion put forward was for 'some form of national computer establishment which would be primarily concerned with new computer developments', an idea that was to recur frequently during the years to come. Meanwhile, in December 1956, Howlett at Harwell and John Corner at the AWRE had put their heads together and come up with a proposal for a British rival to STRETCH. Their target was a computer that could be used to handle three-variable equations, which meant something as powerful as or more powerful than STRETCH, and their tentative suggestion was that a design competition be held between British firms to select a manufacturer. The development costs, which they put optimistically at £½ to £¾ million, could be borne by the Atomic Energy Authority.

Howlett and Corner submitted their proposals to their respective establishment directors, John Cockcroft and William Penney. Cockcroft seems to have been enthusiastic, and sent a copy of the proposals to Halsbury, who informed him of similar thoughts current at NRDC. Penney, however, appearing to have been more cautious, was prepared to see a case made out for the project but doubtful whether Britain could afford to undertake it given her limited scientific resources and the limited demand for such a large machine. Halsbury was also conscious of the problems to be faced if any advance were to be made upon STRETCH, especially in terms of operating speed. New components would be needed, and to limit machine delays the size of the computer itself would have to be drastically reduced. Discussing the project with his Computer Sub-Committee, he concluded that if it were to be successful it would probably have to be restricted to a computer slightly less powerful than STRETCH; and even then some sort of national collaboration and co-ordination would be needed.

Halsbury's feeling that no one British laboratory or manufacturer could take on a project such as that envisaged was generally shared. Even in America the STRETCH project was dependent on strong government backing, and by May 1957 IBM were reported to be spending $28 million a year on their overall research and development and to be employing 300 graduates on the STRETCH project alone. This corresponded to the entire resources of the British computer industry multiplied several times over, and while a project of such immense size was not considered necessary in Britain even one on the scale proposed by Howlett and Corner or the NRDC looked likely to require some kind of centralised funding and control. The NRDC Computer Sub-Committee had recognised this and had proposed in February that activities might be co-ordinated by NRDC itself, a course of action made possible by their

widened terms of reference. Meanwhile at Harwell discussions between Brian Flowers and E. H. Cooke-Yarborough led to the idea of a national institute for research in electronics, a concept clearly related to the new National Institute for Research in Nuclear Science (NIRNS), in the advocacy of which Flowers had also played a leading part. The suggestion was for a nationalised British equivalent of the American Bell Laboratories, run by the Atomic Energy Authority and charged among other things with the development of new components and circuitry for a generation of computers beyond STRETCH. To try and compete with STRETCH itself in the time available was not thought by Flowers and Cooke-Yarborough to be feasible; but to maintain an advanced British computer industry in the long term they held to be essential.

By April 1957 there were thus already several different concepts of the fast computer project: the original Howlett–Corner concept of a computer comparable with STRETCH, the less ambitious concept of NRDC, and the more ambitious but longer-term concept of Flowers and Cooke-Yarborough. No-one as yet envisaged more than a single project, however, and the issues of what this should entail and how it should be organised were thoroughly confused. In respect of the organisation, Halsbury found his board at NRDC very sympathetic to putting up £1 million for the project, and at the beginning of April he put forward a proposed form of organisation consistent with this. The NRDC would appoint an advisory panel composed provisionally of Williams, Wilkes, Uttley from the National Physical Laboratory and Cooke-Yarborough, and would strengthen their own design team under Strachey. They themselves would design the computer and place development contracts with universities and government laboratories, leaving open for the time being the decision as to whether the machine should be manufactured by an established firm or by a specially assembled team. The aim would be to produce a computer roughly comparable with STRETCH in 1961, one year after STRETCH itself. The initial specification would be to Atomic Energy Authority requirements and that Authority would be the first customer.

Halsbury's proposal was clearly dependent on Atomic Energy Authority backing, and relied in particular on there being no rival Authority project. But although Cooke-Yarborough, for example, was unhappy with the form of organisation proposed, Cockcroft was more encouraging and Halsbury, taking this encouragement as positive support (a mistake many people made with Cockcroft) proceeded to try and realise his plans.

Meanwhile, Harwell proceeded with their own investigation of the technical side of the problem, and at the end of April a meeting was held at Harwell to define the technical requirements of a fast computer. For this meeting the Harwell experts were joined by Uttley, Wilkes, Strachey, Kilburn from Manchester, and others, and the conclusion was reached that there might be a demand for six very fast computers in the Atomic Energy Authority and the nuclear power industry alone. This conclusion

was interpreted by the NRDC Computer Sub-Committee as indicating further support for Halsbury's proposals, but the meeting also led to a specification of the technical and organisational requirements of a project that differed considerably from Halsbury's.

These requirements were set out in a paper on the 'Super Computer' circulated by Cooke-Yarborough, Flowers and Howlett. In this paper the authors assumed that several STRETCH type machines would be needed in Britain and that it would be worth developing them even if they were not ready until a little later than STRETCH itself. In this respect they were in line with NRDC proposals, but whereas Halsbury had intended to base the computer development upon improved logic, for which he relied on Strachey, the Harwell authors felt that this approach was short-sighted since limited, and that progress should rather be sought through improved components. This was the basis of the STRETCH project; it provided a cheaper final product, was necessary anyway for further developments in the future, and should also lead to applications beyond the field of computers. The authors also criticised the proposed NRDC organisation, or rather the lack of it, and the general NRDC philosophy of recovering their investment largely from the sale of the first machine, which would of course be to the Atomic Energy Authority. If the Atomic Energy Authority were to end up paying for the project anyway, it was argued, they might as well sponsor it directly, especially as it was they who had the general need for advanced components.

The Harwell proposals had much to commend them, but they had no official backing. To many within the Atomic Energy Authority they typified a Harwell tendency to get carried away with speculative projects and in so doing to get drawn away from their proper concern with atomic energy. There was also a feeling that if there were to be a British fast computer project it would have to be large, powerful and determined, and that given what were considered to be the country's scarce scientific resources it would be unwise to invest heavily in such a project, particularly as the economic outcome was uncertain.

Before these doubts could reach him, Halsbury, emboldened by the general enthusiasm shown at the Harwell meeting, tried to float the NRDC programme officially. He approached the Board of Trade, whose approval for the project seemed to be required, citing an immediate Atomic Energy Authority requirement for four of the proposed computers, a figure apparently derived from the conclusions of the Harwell meeting.

Clearly, everything now depended upon the official view of the Atomic Energy Authority, and given the understandable reaction within Authority circles against the ambitious and unauthorised Harwell proposals the prospects were not good. Now realising this, Halsbury pressed his case hard with a barrage of considerations ranging from the fate of Britain as a high technology exporting nation and her claims as the inventor of computer technology (*sic*) to the potential economic gains of the computer industry. Above all he stressed that, given the history of

computers so far, suggestions that only a very few meeting the new specification would be needed were likely to prove unduly pessimistic. His overtures, however, seem to have made little impression. The price of STRETCH had recently been forecast as £1¼ million, and this did not imply sales of the proposed computer sufficient to justify the project. Moreover, the Atomic Energy Authority would hardly want to commit themselves to a completely unknown computer type when the main part of their requirement was for a military programme requiring tight deadlines and complete reliability, and when their non-military computing requirements remained unknown.

Given these circumstances it is most unlikely that the Board of Trade would have given their approval. It soon turned out, however, that Halsbury did not in fact need this approval for the project as a whole, only for some items within it. The discrepancy between Halsbury's original estimate of the Atomic Energy Authority requirements and the estimates prevailing within the Authority itself therefore ceased to be formally critical, and towards the end of June 1957 the board of NRDC, acting on the recommendation of their Computer Sub-Committee, approved in principle the allocation of £1 million, thus giving Halsbury authority to proceed. Meanwhile, Cockcroft, pursuing the possibilities raised at Harwell, discussed with Wansbrough-Jones of the Ministry of Supply the possibility that TRE Malvern might be able to test the envisaged new components, and the further possibility that they might take a leading part in the design of the new computer, as envisaged at Harwell. Wansbrough-Jones, who had been kept informed of earlier NRDC developments, was apparently sympathetic to both suggestions, but he also seems to have been concerned at the lack of co-ordination between the Harwell and NRDC plans, and at the beginning of August Cockcroft and Halsbury met at his invitation at the Ministry of Supply to explain their respective proposals.

Since Halsbury had both the permission and the finance to go ahead the ball was very much in his court, and the meeting naturally focused upon his proposals. These were now quite simply for a board of directors appointed by NRDC at a technical level who would place development contracts with industrial and government concerns. Against this Cockcroft argued for at the very least a co-ordinating design committee, and suggested that Harwell might have nothing to do with the project if such a committee were not appointed. Halsbury was reluctant to give in on this point and Cockcroft left the meeting, having secured nothing but an agreement to a further technical meeting at Harwell a fortnight later. At this second meeting, which was attended by Flowers and Cooke-Yarborough from Harwell and Crawley and Hennessey from NRDC, Crawley put forward the NRDC case in more detail. Given their experience of the British computer industry to date they did not consider that a single firm could manage the project, or that a consortium would be effective. In line with their normal practice they would wish to recover their investment with a profit, and the best way of doing this seemed to be

to set up a new body, as far as possible autonomous, but subject to general directives from NRDC. This body, which was now envisaged as a subsidiary company of NRDC, would implement the design of the computer and place contracts for its development and manufacture. If necessary it would have its own development laboratory and it might even carry out the manufacturing as well, if this seemed best.

The NRDC proposals were intended to secure NRDC control of the project and thus of the fate of their investment, but they seemed to make little sense otherwise. The Harwell team objected quite reasonably that it seemed undesirable to set up a new organisation that would inevitably duplicate existing facilities, and that the early stages of the project should be undertaken in a large and experienced laboratory, such as Harwell or TRE Malvern, where the resources and range of expertise needed to meet the inevitable problems would be available. They felt strongly, moreover, that the ultimate manufacturer, who should again be a well-established company, should play a major part in the design work. They therefore proposed a two-stage project, the first stage being carried out at an established laboratory with participation from a manufacturing firm, and the second at the manufacturer, with the continued participation of the laboratory.

The idea of basing the development stage at Harwell was scarcely in line with official Atomic Energy Authority policy, in which there was no place for non-atomic research. That apart, the objectives outlined by Flowers and Cooke-Yarborough seem to have been shared by Cockcroft, and there seems to have been a complete deadlock between Harwell and the NRDC. Further discussions involving Joseph Lockwood of EMI failed to resolve the situation. The NRDC Computer Sub-Committee decided to push ahead regardless, calling a meeting of all interested parties for 22 October and pre-circulating their ideas in the form of firm proposals. However, the meeting was never held. Williams, who was envisaged by NRDC as a key figure in their proposed subsidiary company, was ill, and when he did meet the Computer Sub-Committee, with Wilkes and Brunt in early November, he expressed a very strong opposition to their proposals and suggested that it would be much better to invest the £1 million in an existing company and let them get on with the job unhindered.

In a final attempt to secure a consensus, the Computer Sub-Committee asked Williams, Wilkes, Brunt, and Tizard of the National Physical Laboratory whether they would be prepared to act jointly to advise NRDC and, if so, to suggest a form of constitution suitable for the continuation of their advisory function. Their response was that they would in fact be prepared to form the nucleus of the board of an NRDC subsidiary, but only if they were allowed to work with a minimum of financial interference from NRDC itself, a condition that was found wholly unacceptable. At the beginning of the new year, 1958, the British fast computer project was back to square one.

The Birth of Atlas

When it became clear that their proposals stood no chance of gaining approval NRDC threw in their hand as organisers and attention shifted back to the Harwell concept of a project based at Malvern. However, NRDC did still have £1 million to spend, and so in January 1958 Patrick Blackett proposed to their board (of which he was then a member) that TRE Malvern should be appointed by NRDC as agents for their project. Blackett's idea, which arose from discussions with Cockcroft, was that Malvern should take on the design of the computer, if possible recruiting Tizard from the National Physical Laboratory as project leader, and place development contracts with private industry. Harwell would give some research assistance but would not play a major role in the project. The new proposal does not seem to have appealed to Wansbrough-Jones, but he agreed subject to higher approval and on 21 January he set the conditions of this approval before the NRDC Computer Sub-Committee. Although these implied a project very different from that that NRDC had formerly envisaged he was duly requested by their main board on the following day to act as their agent as proposed. The conditions were that there should be a clear objective to produce a machine far in advance of anything then available, and a clear prospect of an outstanding success. TRE Malvern should be entirely responsible for the work, all contracts being handled by the Ministry of Supply in the usual way. A single well-established manufacturing firm should be introduced as prime contractor at an early stage of the project. Continued and active co-operation should be supplied by a technical panel outside the TRE, and this panel would have to agree on a full design specification of the computer before the Ministry of Supply allowed work to begin.

In order to meet these conditions it was agreed that yet another meeting of technical experts should be held and this was arranged for the end of February at Harwell. This time clear terms of reference were set for the meeting: to determine the objectives and requirements of the new computer; to set out the best technical approach towards meeting these requirements; to agree on a format for continued technical co-ordination; and to agree on an assurance that the computer proposed would constitute a major step forward such as could not have been achieved by NRDC spending their money elsewhere, a form of guarantee against blame should the project fail. The new meeting was attended by almost all the recognised technical experts. The result was, remarkably, an agreed proposal, but for two projects rather than one, neither of them corresponding quite to what they were supposed to be approving.

The first project proposed was a short-term one aimed at producing a computer based on existing components technology within five years. This concept seems to have had its roots in Manchester, and it was suggested that the current Manchester computer project, MUSE, might form the basis of the new machine. The initial expenditure for this project was estimated at £½ million, and the outcome was reckoned to be a

computer two or three times slower than STRETCH but considerably cheaper.

The second project was a long-term one aimed at developing and using new components and techniques and at going well beyond STRETCH. This was essentially the old Harwell proposal and it was recommended that it should be conducted at Malvern, under the guidance of a small advisory committee.

Despite the dual recommendations and their familiar ring the Harwell meeting was a great step forward. The wide representation of expertise ensured that the recommendations carried authority. For the first time the two very different concepts that made up people's ideas of a new fast computer, which had previously been repeatedly confused, were clearly and unambiguously separated. Finally, the assessment of what could in practice be done was a reasonably realistic one.

Given this realism, the immediate focus of attention was upon the short-term project, for a computer not unlike that originally proposed by Halsbury. The question that arose here, and that was not answered by the Harwell meeting, was whether the Manchester MUSE would in fact satisfy the criteria set. Its designers claimed that it would, but the meeting thought it desirable to set up a working party, composed of Kilburn, Strachey, Wheeler, Howlett and Cooke-Yarborough. Their report, when it came, did not agree with this assessment. The Manchester project was aimed at the production of a new fast computer using existing components in about three to four years, but the specification of this computer seemed to fall far short of that envisaged at the Harwell meeting, especially in terms of storage capacity and peripherals. Neither the design of the computer nor the effort being put into it seemed to be sufficient for a national project and the working party could not recommend them as the basis for such. An enlarged project, they reported, might meet the criteria, but Kilburn, whose project it was, disagreed with the rest of the working party as to how such an enlarged project should be organised. Since the existing project was not dependent upon NRDC funding this ensured yet another stalemate.

The report of the working party was not completed until May and was not discussed at the NRDC Computer Sub-Committee until June, by which time new factors had come into play. In the first place English Electric had begun to consider the development of a STRETCH type computer, and in the second place Ferranti, who had manufactured the earlier Manchester designs, had begun to consider the possibility of developing the MUSE. Ferranti had been looking in general terms at the possibility of extending their range to a larger machine ever since the AWRE had chosen the IBM 704 in preference to a MERCURY, and they had recently tried to persuade the Atomic Energy Authority to sponsor the development of a new computer comparable with the 704. Such sponsorship had not been forthcoming, but Cockcroft did give his personal support to Ferranti's idea for a larger computer based on MUSE.

The existence of two rival possibilities ensured that no positive action could be taken by NRDC for the time being, as they felt an obligation to examine both very closely before committing their resources to either. Moreover, the English Electric project was no more than a vague and distant possibility, while Halsbury had doubts about the Ferranti–Manchester proposal. His previous experience and the response of Kilburn to the working party recommendations both suggested that the collaboration was unlikely to be a smooth one. Difficulties had arisen in the past between the Ferranti engineers and the university programmers and designers and he could see no reasonable prospect of the situation improving. Besides this, he was also concerned by Cockcroft's interest and by the thought that the project might be linked more closely to the immediate needs of Harwell than to the more general aims of NRDC. Although he had argued for a conservative project against more ambitious Harwell proposals he tended to view this project as too conservative.

Given these reservations and the negative conclusions of the working party, NRDC could hardly support the Ferranti–Manchester project, but the involvement of Ferranti did imply that the machine would be better equipped than had been supposed by the working party, and it was felt that the opportunity presented should not be wasted. So Wansbrough-Jones, who was himself a member of the main NRDC board, was invited to write to Cockcroft suggesting that the idea of a Ferranti–Manchester collaboration be pursued, but on the basis of contracts from TRE Malvern, who would take over the design and contracting responsibilities for the project as agents for NRDC – effectively a revival of the last NRDC proposal. Cockcroft was also to be invited, as representing the main prospective user of the finished computer, to discuss the situation with Williams.

Williams agreed with the scheme in principle, but he was worried as before that NRDC might interfere with the design of the computer, and he insisted that all responsibility would have to rest with the Ministry of Supply. The proposal therefore went back to NRDC in the form of a project for which TRE Malvern were not only the contractors but also the ultimate design authority, working with the benefit of NRDC's £1 million but without being subject to any technical control from that body. The only influence NRDC would be able to exert would be through the advisory panel of experts, on which they would be represented. Needless to say, this complete delegation of authority did not prove attractive to NRDC, who were still concerned, moreover, as to how they would recover their investment and show a profit. At the meeting of the Computer Sub-Committee to which the proposals were put in mid-July, Halsbury made it clear that in his view they left NRDC with insufficient control, and that they did nothing to resolve what he saw as the basic organisational problem of a Ferranti–Manchester collaboration, that of the differences of opinion between designers and engineers.

Halsbury's recommendation was that action be deferred until the

autumn. Meanwhile, he himself tried a different approach, bypassing the experts altogether and going straight to the Atomic Energy Authority as a possible user and to Ferranti as a possible manufacturer with the simple proposal that NRDC should give Ferranti a contract to build a computer based on MUSE, but designed to an NRDC specification.

The new proposal does not seem to have been entirely without support in the Atomic Energy Authority. Cockcroft was well aware of the need to think of the national interest as well as of immediate Authority needs and apparently saw the proposed Ferranti computer as a feasible project that would tie in relatively well with Atomic Energy Authority's requirements. Penney, however, seems to have remained doubtful, and for very good reasons. The latest information from IBM suggested that STRETCH was costing £5 million to develop and would cost another £5 million for each finished machine, quite possibly making a substantial loss. It was still difficult to see a need for more than a few machines of that size in Britain and the AWRE would be compelled to order on the basis of speed of delivery and expectation of reliability, which would probably mean STRETCH. Furthermore, doubts developed at Harwell. Probing by Cooke-Yarborough led to the realisation that there were still conflicting views of the responsibilities that would be involved in the proposed project. There was also a widespread feeling that it was now too late to challenge the Americans, and that the finances proposed with which to do it were woefully inadequate. If anything were feasible it was likely to be the long-term project based on advanced component development, and aimed at advanced small computers rather than very large ones.

Ferranti's initial response to NRDC was that they would indeed like to be the main contractors for a project, provided only that it would not put them in the bankruptcy courts. Financial negotiations began forthwith. Since, however, it was an unalterable requirement of NRDC that they should expect to recover their investment, and since Ferranti were not prepared to take all the risks themselves without at the same time having complete control over the design, there was little chance of agreement. It was soon recognised that the situation was insoluble, and at a meeting of the NRDC Computer Sub-Committee in mid-September it was agreed that NRDC were unlikely to find any contractor prepared to agree to their terms. Blackett had suggested to Halsbury a few days earlier that he could find out whether the problems were genuine or whether Ferranti were just being difficult by approaching another firm such as English Electric, and it was agreed to do this. But by the next meeting of the committee in mid-October the same response had been received from English Electric as from Ferranti. It was clear that the NRDC project in its present form was doomed.

Ironically, it was the very failure of NRDC proposals in the autumn of 1958 that provided a situation in which a workable project could at last be got under way, and just when the intended principal user, the Atomic Energy Authority, had come out unequivocally against it. There was still interest in Harwell in the long-term project of developing a highly

advanced computer, perhaps ten times more powerful than STRETCH, and at Cockcroft's request this project was discussed by NRDC early in 1959. It was agreed, however, that in this case the involvement of NRDC would not be appropriate until a much later stage, and following discussions with TRE Malvern and the DSIR the responsibility for the first four years of component development and feasibility studies was taken on by Malvern, with assistance from the National Physical Laboratory and funding from the DSIR. In this way the short- and long-term projects were finally separated. Malvern concerned themselves solely with the latter and the Atomic Energy Authority, although maintaining an interest in both, effectively dropped out altogether. The latter move was in some quarters a reluctant one, as Cooke-Yarborough in particular was critical of the DSIR programme and would have liked to have been in a position to alter it. However, in 1959 Cockcroft left Harwell to take up the mastership of Churchill College Cambridge, and the Atomic Energy Authority computer policy was taken over entirely by Penney, who was less sympathetic towards Harwell diversification, and more careful about getting involved in potentially embarrassing commitments. So far as the short-term project was concerned the departure of the Atomic Energy Authority and Malvern from the stage had the principal effect of reducing the number of competing interests, and if NRDC and the manufacturers could not yet reach any financial agreement at least the prospects for a technical consensus were improved.

Following the failure of NRDC to reach an agreement with either Ferranti or English Electric they sent a request to several firms, including these two and EMI, for proposals as to what form of agreement would be acceptable to them. As a result of this both EMI and Ferranti submitted proposals, and when EMI expressed a willingness to share some of the risk involved by offering NRDC repayment of a loan, as in their earlier contract, through a levy on subsequent computer sales, Ferranti at last felt obliged to follow suit. Since both proposals were for lesser sums than the original £1 million the solution of the repayment problem was in fact considerably simplified, and there was at least some prospect of financial agreement. Halsbury seems, moreover, to have overcome his scruples as to the proposed machine constituting a significant advance and, on the verge of retiring from NRDC, to have settled on the priority of handing out some money before the idea was abandoned altogether. Although the Ferranti and EMI proposals amounted to less than the £1 million available between them, however, the Computer Sub-Committee was not prepared to fund both projects, at least to the degrees proposed, and met in March 1959 to consider the alternatives.

The choice presented, in a paper by Halsbury, was between spending £500,000 on the Ferranti–Manchester project, the total development cost of which was estimated at £850,000, or £280,000 out of a total of £374,000 on EMI. EMI's proposal, to convert their 2400 business computer to meet the needs of scientific users, bore little relation to the original NRDC intentions. However, Strachey assessed the two projects as technically of equal merit and, on the understanding that there was only 'the narrowest possible margin for decision', the committee recommended that the EMI proposal be accepted. The main NRDC board at their next meeting drew attention to the narrowness of the decision and, acting on Halsbury's recommendation, agreed to support both projects but with smaller loans than proposed: £240,000 for EMI and £300,000 for Ferranti. The whole proceedings made little sense. Even allowing for the possibility that Lockwood may have dreamt up the EMI proposal specifically to help Halsbury get the terms he wanted out of Ferranti (the EMI project was never carried out), it is difficult to see why it should at first have been thought wrong to finance both projects, since they were not for comparable computers; why that favoured was the one further removed from what NRDC had been seeking for years; and why when both were financed it was not to the full extent proposed. But at least a British 'fairly fast' computer project, in the form of the Ferranti–Manchester ATLAS, the codename for the development of MUSE, was under way.

The Fate of ATLAS, I

The sums finally offered by NRDC were not such as to make much difference to the manufacturers' estimates of their programmes. The EMI project came to nothing anyway, and by the time the offer was announced Ferranti had already decided to continue with their project regardless. They thought of calling their new machine BISON (for 'Built In Spite Of NRDC'). It was actually christened ATLAS and was a relatively sophisticated and technically very impressive computer that represented an outstanding achievement by their very small team. Despite these undoubted merits both the project and the product had what were in retrospect fatal flaws. ATLAS was a true child to its parents, and its fate constitutes a pertinent reflection on its origins.

As a project, ATLAS faced two main problems, both of which had been predicted. First it turned out to be too large a project for the small Manchester University and Ferranti Computer Department teams. The number of men who could be employed on the project was a mere tenth of that on STRETCH, and while their achievement was considerable the amount of research and development required was far greater than either the firm or the university team had realised. The programme overran its target dates and ran into considerable problems as the hurry to complete it left inadequate time for sorting out teething troubles, and as the delays led to it being rapidly overtaken by more generously equipped rival projects. The second and related problem was that of finding customers. At one stage the range of interested potential customers looked very encouraging, including as it did ICI, CERN, the CEGB, and Australian and American organisations as well as the larger British universities. But again the project was sandwiched between converging pressures. The customers would not place orders until the computer had been proved,

by which time its rivals, and in particular the comparable computer developed by Seymour Cray at CDC, had established themselves firmly.

In short, the ATLAS effort was too small and too late, as the Harwell men had predicted. But its fate is still of interest as showing the way in which this situation was handled by the manufacturer, the users and the Government. In the early stages of the machine's development the most obvious markets were seen as the Universities of Manchester (for the prototype) and London and, despite the Atomic Energy Authority's refusal to back the project, Harwell. Informal talks between Ferranti and Harwell took place from the early stages of the project, and although the firm did not neglect the wider commercial appeal of their machine they were conscious both of the expertise at Harwell and of the importance of an Atomic Anergy Authority order, and so tailored the ATLAS to some extent to meet Harwell requirements. In the summer of 1959 a team of Atomic Energy Authority experts from all their establishments visited Ferranti and returned very impressed by the design of the ATLAS. Although less powerful than the STRETCH it was much more sophisticated and advanced and looked likely to be more flexible.

By this time Penney was in charge of the Atomic Energy Authority's computer policy and at the beginning of 1960 he decided to base this policy on the concept of two large computers, one at the AWRE and the other at Risley, headquarters of the industrial groups in Lancashire. For the AWRE, reliability was clearly essential, and so was full compatibility with the American weapons programme. Talks with IBM had been continuing on a regular basis, and in early 1960 Penney requested permission to hire a STRETCH. Since the British Government were financing the ATLAS project, albeit indirectly and in very small measure, approval of such a move was not automatic. The costs of both STRETCH and ATLAS had escalated, and although the purchase price of an ATLAS was now estimated at £2 million to £3 million this amounted to only about two years hire of a STRETCH, making ATLAS seem the better buy. But time and military requirements were paramount and by the summer, approval to hire a STRETCH had been granted. For the second computer, for civil use, the obvious policy was to follow an established precedent and transfer the latest AWRE machine, an IBM 7090, to Risley when the STRETCH was delivered. But ATLAS was now getting very high recommendations from the Atomic Energy Authority computer experts, and now that the project was established Penney and his colleagues were naturally anxious to help it as best they could. Perhaps more significantly, it was recognised that with a British firm making a big machine the AWRE would probably have to move over to it, even for weapons work, at some stage in the future.

Harwell's experience of the Ferranti MERCURY had been of late delivery and, although the machine now worked very well, of a year or more of severe teething troubles. The ATLAS project was already encountering delays, and the widespread confidence in the design of the computer was offset in some quarters by a lack of confidence in the

manufacturer to undertake such an extremely difficult task. To some of the AWRE experts, such as Corner, the evidence argued convincingly against ordering an ATLAS, but Penney may have reasoned that if he was going to have to order one for weapons work he had better get all the problems sorted out as quickly as possible by going for fast delivery of a machine for civil use. He certainly maintained close contact with Ferranti and concentrated on encouraging them to bring their specifications in line with Atomic Energy Authority needs.

Penney's interest in ATLAS was no doubt encouraging to Ferranti, especially as he had been among the project's earlier critics. However, what they needed now was a firm order, preferably in time to prevent a gap in production following completion of the University of Manchester prototype. Throughout the first half of 1960, Sebastian de Ferranti lobbied the Atomic Energy Authority and the Government. By June of that year he had spent £1 million on the project and a firm government order would, to put it mildly, have been welcome. By the time the decision to hire a STRETCH for the AWRE became known in August, however, there was still no sign of such an order, and while Ferranti had several imminent potential orders they badly needed the vote of confidence that a government order would entail. (The fact that the STRETCH was being hired rather than purchased reflected normal IBM policy and was no encouragement). Faced with a desperate situation, de Ferranti stepped up his lobbying, emphasising the encouragement that had been received from the Atomic Energy Authority and offering to accept heavy penalty clauses if these were necessary to secure a contract.

Feelings within the Atomic Energy Authority were mixed, but the general mood seems to have been sympathetic. Penney in particular had encouraged the Ferranti project once it was under way and had indeed been taking steps to secure an order for an ATLAS. Although the policy remained that of two large IBM computers at Risley and the AWRE it was becoming clear that both Harwell and NIRNS, sited outside the security fence alongside the Harwell establishment, would soon need considerably increased facilities. The Harwell computer experts had been pressing for an ATLAS to be shared with NIRNS and the universities, and in late summer Penney took up their idea and proposed that the Atomic Energy Authority should be authorised to order an ATLAS to be sited if possible outside the fence at Harwell, for the general use of universities and government establishments such as Harwell, NIRNS, the DSIR, the Ministry of Aviation and the Meteorological Office. This proposal led to a flurry of activity and discussion and no immediate decision, but by Christmas an inter-departmental working party had recommended acceptance of the proposal, with the slight alteration that that computer should be managed by NIRNS rather than by Harwell. About three months later the Atomic Energy Authority were given authority to begin negotiations. Unfortunately this order came about a year too late to help Ferranti win over their other potential customers. As the prospects of a long ATLAS production run had dwindled, the profit

they sought on the Atomic Energy Authority sale had risen, and the price now stood at about £3½ million. Meanwhile, however, STRETCH had run into difficulties, its performance well below the original specification. The ATLAS had thus become more attractive by comparison and an order was placed later in the year, with delivery set for early 1964.

For the reasons already mentioned the ATLAS sales predicted by Ferranti never materialised. A second production machine was acquired by London University, but other customers either felt unable to justify such a large and expensive machine or went for the rival CDC 6600 instead. Although it had much in common, technically, with the ATLAS, this had the advantages of a developed satellite system and much better peripherals. The absence of a satellite system for ATLAS, and the weakness of its peripherals, probably cost Ferranti contracts in Australia and at ICI.

The Fate of ATLAS, II

By 1964 IBM had announced their versatile new 360 series of computers, sophisticated machines that were to be fully inter-compatible, capable of being linked into systems and covering the full range from medium-sized accounting machines to large scientific computers much more powerful than ATLAS. CDC had cornered the specific market intended for ATLAS, and ATLAS itself was effectively obsolete. After a succession of delays the machine for NIRNS was eventually commissioned towards the end of 1964, but before this had taken place both Harwell and NIRNS were budgeting for their own large IBM computers. In 1965 the Flowers Report on computers for research pointed to the limitations of the ATLAS facilities at the universities and proposed their replacement by large American computers as soon as possible. This was not quite the end of the ATLAS project, however.

The possibility that the AWRE STRETCH would eventually be replaced by an ATLAS had long been envisaged. With the lack of any commercial interest in the ATLAS the possibility receded between 1960 and 1963, but it was revived in a slightly altered form as a result of subsequent developments at Ferranti. The first of these developments was rooted in the early negotiations for the sale of ATLAS, when the proposition had been put to Wilkes in Cambridge that Cambridge University be given ATLAS hardware on special terms in return for work on a modified and cheaper design. This offer was taken up and by 1963 the design of the modified ATLAS 2 was well advanced. The second development concerned Ferranti's whole interest in building computers. At about the time of the eventual NRDC loan offers to Ferranti and EMI these two companies had been talking with the newly formed firm ICT, the result of a merger between the punched-card companies BTM and Powers–Samas, about possible collaboration and even a jointly owned company. These talks had been prompted by the growing conviction that the British companies with interests and experience relevant to computers would have to pool a large part of their resources if they were to compete successfully with foreign competition. These talks came to nothing, but as the ATLAS project progressed Ferranti began to realise that they had taken on more than their limited production and marketing facilities could cope with. For the manufacture of ATLAS to be a commercial success the machine would have to be regularly updated using new components and produced and sold in quantity, and in 1963 Ferranti decided to sell off their computer interests to ICT, who had the facilities to make something of them.

The eventual outcome of this deal was that ICT decided to drop the ATLAS in favour of their own anticipated 1900 series of versatile and inter-compatible computers. However, while the discussions were taking place Ferranti, wishing to get a good price for their operation but faced with the commercial failure of the original ATLAS, naturally sought orders for the ATLAS 2. In the course of discussions with the Atomic Energy Authority in the summer of 1963, they brought the price of the ATLAS 2 down to under half that of the original ATLAS, and did everything they could to increase its acceptability.

Meanwhile at the Atomic Energy Authority the ATLAS 2 had already been seriously considered. At the beginning of March 1963 an alternative possibility had been raised. Like ATLAS, STRETCH had also been a commercial failure, and realising that they could do nothing with the AWRE machine if the Atomic Energy Authority decided to cease hiring it, IBM had indicated that they might be prepared to sell it outright for a relatively low price. Since the AWRE were fully geared to the operation of the STRETCH, and since this operation had proved technically satisfactory, the suggestion was appealing. Even allowing for the decrease in the price of an ATLAS 2, the option of buying one of these did not offer a sufficient financial incentive to compensate for the costs and risks of changing over from one system to another. Although the purchase of the STRETCH was the favoured technical option, however, political considerations also had to be taken into account, and these eventually prevailed. An order for an ATLAS 2 was placed in the autumn.

There were of course still problems in store, for ICT found themselves landed not only with an uneconomically low price for the ATLAS 2, but also with severe penalty clauses accepted by Ferranti. An agreement was finally reached based on a reasonable price and a rising scale of damages for delay, but the re-design of the ATLAS involved more work than had been anticipated and despite considerable assistance from the AWRE and Harwell computing teams, the delays mounted quickly. By the time the computer was accepted, 14 months late, at the end of February 1966, the Atomic Energy Authority had had to buy the STRETCH anyway, to fill the gap.

The trials and tribulations of the scientific computer industry were matched by those in the business computer side. This volume market was the one at which BTM, later ICT, had aimed, and in which they were not altogether unsuccessful. However, they could not compete with the

Americans. By March 1965, ICT were estimated to have sold 439 computers world-wide, other all-British manufacturers 135, and NCR–Elliott 318. By July 1964, in contrast, IBM alone were estimated to have sold 11,025 computers, excluding three models then out of production, while the total sales by American firms were estimated at around 20,000 machines.

Conclusions

This study covers only a small part of the story of the early British computer industry, and as such it poses many more questions than it resolves. In particular, one would like to know much more about the history from the viewpoints of the firms: why did they enter the industry when they did (and not before) and in the ways that they did? how did they go about assessing the market for computers, and with what results? and to what extent did they seek to educate and develop the market in their new products? One would also like to know more about the attitudes of candidate users of the machines. Despite its major limitations, however, the study does already raise a number of interesting points.

One of the most striking features of the story is the lack of agreement between the technical experts, which repeatedly held up the fast computer project. The problem of reaching technical agreement on the development of a new technology is not one that is often discussed, but it seems to be one of some general significance. In America too the Eckert–Mauchly initiative was very nearly shot down by adverse criticism of their UNIVAC proposals from other experts in the field, and very similar problems have also bedevilled the attempts to commercialise nuclear power. For countries who could afford only one main line of development the bitter rivalry between proponents of gas and water moderator technologies (Britain) or between gas and sodium cooling in the fast reactor context (Germany) have severely hampered the progress of development. That scientists and engineers will disagree, often strongly, about appropriate development paths is clearly apparent from these and many other examples. It is a phenomenon, however, that NRDC in the 1950s was clearly not equipped to handle, and one with which neither governments nor, in many cases, firms have yet come fully to grips.

Another very striking observation concerns the apparent total failure of Ferranti to see who their main competitor was, or at least to respond appropriately. The ATLAS was built as a response to the IBM STRETCH, but the STRETCH itself was seen in some quarters as a pre-emptive response to the computers planned by CDC. By the time the ATLAS project got under way both the aims and the abilities of CDC were well known. In the end it was their machines which deprived ATLAS of the market it might have had. To establish the reasons for Ferranti's failure to meet this challenge would clearly require further research, but two suggestions may be made at this stage. First, the ATLAS combined a very sophisticated and well-developed design of the

main computer with a relatively primitive attitude towards peripherals and other customer requirements; it was an engineers' machine rather than a users' machine. It is often said that Britain is good at research and bad at development but this example suggests that the diagnosis should perhaps be refined to distinguish between different aspects of development. The Ferranti team certainly had the ability to develop their machine in a way that would have competed with CDC, but they may well have lacked the inclination. The second suggestion is that Ferranti may have placed too much faith on an AEA contract. They designed the machine to AEA rather than commercial requirements, sought to recover much of their development costs from an AEA contract, and treated such a contract as the lynchpin of their marketing strategy. This approach was well ingrained, for Ferranti depended heavily on defence orders and government-financed military research and development, but in this, as in other cases, it may well have been counter-productive.

Quite apart from the general danger of tailoring a product to a single customer and of the commercial slackness engendered by generous defence contracts, another observation here is that the AEA were not in fact in a position to act as sponsors, and this raises the general question of government sponsorship of the computer industry. If we look at the situation in America, we find that the Eckert–Mauchly venture was sponsored to the tune of $300,000 by the National Bureau of Standards, who also sponsored work at Raytheon. Other pioneer projects were sponsored by the Office for Naval Research. The entry of IBM to the computer field was almost entirely funded by defence contracts, and between 1949 and 1959 that firm alone spent over $250 million of government money on research and development (not all of it, of course, on computers). Between 1951 and 1959 the major US electronics firms spent over $8 billion on research and development, of which more than half came from government contracts.

A straight comparison between Britain and the USA is scarcely feasible. The USA was a much larger and much richer country. It did not face the British problems of post-war reconstruction, and was mobilised at a crucial period for the Korean War. A sympathetic view of the story told here might well see the British effort as a valiant struggle against insuperable odds, and the fact that the British computer industry fared better, in the early period, than any other outside the USA would support such a view. But there is also a sense in which the British industry floundered in the way in which government assistance was given, rather than on the financial extent of that assistance. The corresponding antipathetic view would be one of excessive red tape, and bold initiatives stifled by restricted terms of reference and bureaucratic decision-making.

The firms, and in particular Ferranti, looked for support to the AEA, and as they were the principal prospective users of scientific computers this was quite understandable. The development of the perceived rivals to ATLAS, LARC and STRETCH, was financed entirely by the American atomic energy programme. But there were three significant obstacles

to more active AEA participation: their terms of reference did not allow them to undertake or sponsor research other than that directed towards atomic energy; they could not afford either to rely on untried prototype computers or to insure themselves in this respect by sponsoring two or more rival systems; and for military and political reasons it was important if not essential that their computing facilities be compatible with those of the American nuclear weapons programme. (This is not, however to deny that a conservative attitude on the part of customers was a problem. Lord Halsbury expressed the view that the root problem lay in just this, in the reluctance, specifically, of British computer users to support their manufacturers consistently and enthusiastically as did their American counterparts by placing advance orders for new machines. Although I argue that his main criticism here, of the AEA, is largely unfair, this is too general a phenomenon in British industry to be ignored.) Given their eagerness, nevertheless, to help as best they could it is apparent that the restrictions were not made clear to the firms. Whether the information was not communicated, or could not be communicated, or whether the signals were simply not picked up by the firms, is not clear, but there is a strong suggestion that the messages intended and those received were sometimes at variance. Given the very different languages of the two worlds of private industry and government service this would not be surprising. If it should have been the case, then the phenomenon is one of very wide interest.

The NRDC too were bound by terms of reference, and also by lack of experience, but while their efforts to stir up British industry in the 1940s and early 1950s certainly merit praise, the way they handled things later does seem open to criticism. Although again one would need to know more of the thinking behind their actions, one would not now consider a five-year loan as at all suitable for the financing of a new ventures operation, and it is clear that the requirements this placed on them in respect of a short-term return on their investments was a serious handicap to their operations. Even allowing for this, however, there seems little excuse for their apparent insistence on recovering their investment in each and every case. Such a conservative practice seems totally out of keeping with the running of what was by its very nature a high-risk portfolio. It also seems likely that for all their efforts the NRDC did more to delay than to accelerate the computer industry, partly as a consequence of trying to be scrupulously fair to all the potential manufacturers, and partly because of the technical disagreements already discussed.

If the story of the fast computer project has an overall flavour, it is probably that imparted by the last mentioned point. American success owed a lot to individual entrepreneurship. Besides the scientist-entrepreneurs such as Eckert and Mauchly there were also people in industry (principally T. J. Watson Jr. at IBM) who preferred to act in an entrepreneurial fashion. There are, of course, strong cultural differences between America and Britain in this respect and even had some of the technical experts tried to set up a company to manufacture computers one

can scarcely envisage government agencies providing them with contracts, or banks lending them the large sums of money they would have needed. But it is notable that none of the British scientists and engineers – who in this case could and did rival the Americans in both fundamental research and engineering development – became entrepreneurs. Men such as Cooke-Yarborough could have made a lot of money by applying their inventiveness commercially rather than in government service, but they chose not to do so. Similarly within the firms, only J. Lyons adopted a really pioneering attitude, and they were scarcely well equipped to succeed in this field.

In place of bold entrepreneurship, the situation in Britain was characterised by careful and prolonged analysis and the search for safe options. Lord Halsbury criticised the firms for agonising over alternative specifications instead of getting on with the manufacture of one design or another, but NRDC were perhaps even more at fault in this respect. So too, although with more excuse, were the other government agencies and establishments, who were rather more subject to the cumbersome multi-level decision-making processes that were the norm within the civil service. In all quarters analysis and negotiation were allowed to drag on while market opportunities were missed, and by the time decisions were made it was on the basis of out-of-date information. What British industry needed in this case was help in developing a product fast enough to ensure a viable market share, and in launching that product so as to achieve such a share. Speed was of the essence. What they got, in complete contrast, were prolonged negotiations – and those were almost inevitably fatal.

London Business School

REFERENCES

In accordance with the established practice of official histories, references to official papers that are not yet publicly available have been omitted. In these circumstances it has also been considered more appropriate to provide an annotated bibliography of published sources used than to give detailed references to these sources. A fully documented version of the paper has however been prepared, and this will be made available to scholars when the official papers become publicly available under the 30-year rule. The papers concerned consist mainly of two files on computer development in the archives of the Atomic Energy Research Establishment at Harwell, together with miscellaneous items from these and other Atomic Energy Authority archives.

Apart from official papers there are two main sources for the present work. One is a Manchester Ph.D. thesis (1973) by P. Drath, 'The Relationship between Science and Technology: University Research and the Computer Industry, 1945–1962', from which are derived all references to the activities of the NRDC Computer Sub-Committee, based largely on the papers and minutes of that committee. Much of the information on the activities of Ferranti and their interaction with the NRDC, based on material in the Ferranti archives, is also drawn from this source. The second major source is information gained from interviews with some of those involved with computer development in the AEA. Of particular value in this respect were discussions with Dr Howlett. Also useful were conver-

sations with Mr E. M. Cooke-Yarborough, Lord Flowers, Lord Penney and Dr K. Roberts. It should be stressed, however, that a comprehensive programme of interviewing still remains to be carried out. Interviews have not yet been conducted with many of the most important actors in the story which is, in this respect, as in many others, only a preliminary one.

Of the other published sources used, the most significant is perhaps a paper by Lord Halsbury, 'Ten Years of Computer Development', *The Computer Journal*, Vol. I (1959), pp. 153–9, in which the views attributed to him in the *Conclusion* are stated. Also used were:

Annals of the History of Computing, Vol. 3 No. 2 (1981); Vol. 5 Nos. 1 and 2 (1983). (Special issues covering the early history of computing in America).
T. G. Belden and M. R. Belden, *The Lengthening Shadow. The Life of Thomas J. Watson* (Boston, 1962).
Cmnd 2883 (Flowers Report on Computers for Research).
S. W. Dunwell, 'Design Objectives for the IBM Stretch Computer', *Proceedings of the Eastern Joint Computer Conference, December 1956* (New York, 1957).
S. H. Lavington, *A History of Manchester Computers* (Manchester, 1975).
N. Metropolis, J. Howlett and G. C. Rota (eds.), *A History of Computing in the Twentieth Century* (New York, 1980) (Essay on the history of computing, mainly before 1950).
Nucleonics (May 1957) (Estimates of STRETCH cost).
W. Rodgers, *Think. A Biography of the Watsons and IBM* (New York, 1969).
N. Stearn, *From ENIAC to UNIVAC* (Redford, MT, 1981).
P. Stoneman, *Technological Diffusion and the Computer Revolution: The UK Experience* (Cambridge, 1976) (A useful economic and historical survey).

CHOICES IN OIL REFINING: THE CASE OF BP 1900–60

By MARI E. W. WILLIAMS

1. Introduction and Historical Background

Within the complex procedures for obtaining consumable products from crude oil the process of refining has played a crucial, but varying role during this century. At the turn of the last century most refineries of significant capacity were in the United States or Russia and were based on fairly rudimentary processes. By the mid-twentieth century refining took place in the Middle East, South America, Australia, the Caribbean and, most importantly for this study, Western Europe. In addition the number and sophistication of processes had increased enormously: a world-wide refining industry had come into existence. The purpose of this paper is to trace the changes in refining within the development of one company – British Petroleum – and to use the evidence from that company to illuminate certain factors behind the general development of the refining industry, particularly in the United Kingdom.

Although the essay will focus on the technical changes involved, it would be impossible to speak of developments in refining simply as a product of improved technology. Technical expertise must always have been a limiting factor, but it was not necessarily the determining one. As the oil business grew, refining took its place within the long and involved series of events somewhere between the discovery of potentially viable sources of crude and the sale of oil products. It is at the refining stage that large-scale chemical transformation of crude into substances more suitable for the consumer occurs, but before that can happen the crude must be extracted from the reservoir and transported to the refinery; and once the refining is complete, the products must be packaged, distributed and marketed. The history of refining thus encompasses a wide variety of activities: logistics – where to site a refinery with respect to the source of the crude and the markets; company administration and government planning; financial planning; the development of new products; and finally of course, the huge technical and engineering expertise behind the construction and everyday working of a large refinery.

One of the main issues to be addressed in this essay will be the role of technical change or innovation within the development of BP's refining capabilities, and it is intended to reveal a number of trends. First, a distinction must be drawn between different levels of innovation: the level of the oil industry in general at which changes break new ground for the industry as a whole, and the level of the company at which individual

ONE-LEVEL STORAGE SYSTEM

BY

T. KILBURN, D. B. G. EDWARDS, M. J. LANIGAN AND F. H. SUMNER

Reprinted from IRE TRANSACTIONS
ON *ELECTRONIC COMPUTERS*
Volume EC-11, Number 2, April, 1962

PRINTED IN THE U.S.A.

R. N. Tibbett

One-Level Storage System*

T. KILBURN†, D. B. G. EDWARDS†, M. J. LANIGAN†, AND F. H. SUMNER†

Summary—**After a brief survey of the basic Atlas machine, the paper describes an automatic system which in principle can be applied to any combination of two storage systems so that the combination can be regarded by the machine user as a single level. The actual system described relates to a fast core store-drum combination. The effect of the system on instruction times is illustrated, and the tape transfer system is also introduced since it fits basically in through the same hardware. The scheme incorporates a "learning" program, a technique which can be of greater importance in future computers.**

I. Introduction

IN A UNIVERSAL high-speed digital computer it is necessary to have a large-capacity fast-access main store. While more efficient operation of the computer can be achieved by making this store all of one type, this step is scarcely practical for the storage capacities now being considered. For example, on Atlas it is possible to address 10^6 words in the main store. In practice on the first installation at Manchester University a total of 10^5 words are provided, but though it is just technically feasible to make this in one level it is much more economical to provide a core store (16,000 words) and drum (96,000 words) combination.

Atlas is a machine which operates its peripheral equipment on a time division basis, the equipment "interrupting" the normal main program when it requires attention. Organization of the peripheral equipment is also done by program so that many programs can be contained in the store of the machine at the same time. This technique can also be extended to include several main programs as well as the smaller subroutines used for controlling peripherals. For these reasons as well as the fact that some orders take a variable time depending on the exact numbers involved, it is not really feasible to "optimum" program transfers of information between the two levels of store, *i.e.*, core store and drum, in order to eliminate the long drum access time of 6 msec. Hence a system has been devised to make the

* Received September 11, 1961.
† Department of Computer Engineering, University of Manchester, Manchester, England.

core drum store combination appear to the programmer as a single level of storage, the requisite transfers of information taking place automatically. There are a number of additional benefits derived from the scheme adopted, which include relative addressing so that routines can operate anywhere in the store, and a "lock out" facility to prevent interference between different programs simultaneously held in the store.

II. The Basic Machine

The arrangement of the basic machine is shown in Fig. 1. The available storage space is split into three sections; the private store which is used solely for internal machine organization, the central store which includes both core and drum store, in which all words are addressed and is the store available to the normal user, and finally the tape store, which is the conventional backing-up large capacity store of the machine. Both the private store and the main core store are linked with the main accumulator, the *B*-store, and the *B*-arithmetic unit. However the drum and tape stores only have access to these latter sections of the machine via the main core store.

The machine order code is of the single address type, and a comprehensive range of basic functions are provided by normal engineering methods. Also available to the programmer are a number of extra functions termed "extracodes" which give automatic access to and subsequent return from a large number of built-in subroutines. These routines provide

1) A number of orders which would be expensive to provide in the machine both in terms of equipment and also time because of the extra loading on certain circuits. An example of this is the order:
   Shift accumulator contents $\pm n$ places where n is an integer.
2) The more complex mathematical operations, *e.g.*, sin x, log x, etc.,
3) Control orders for peripheral equipments, card readers, parallel printers, etc.,
4) Input-output conversion routines,
5) Special programs concerned with storage allocation to different programs being run simultaneously, monitoring routines for fault finding and costing purposes, and the detailed organization of drum and tape transfers.

All this information is permanently required and hence is kept in part of the private store termed the "fixed store"[1] which operates on a "read only" basis. This store consists of a woven wire mesh into which a pattern of small "linear" ferrite slugs are inserted to represent digital information. The information content can only be changed manually and will tend to differ only in detail between the different versions of the Atlas computer. In Muse this store is arranged in two units each of 4096 words, a unit consisting of

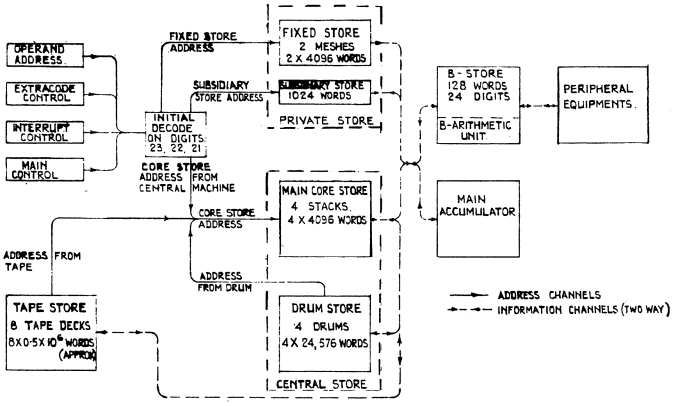


Fig. 1—Layout of basic machine.

16 columns of 256 words, each word being 50 bits. The access time to a word in any one column is about 0.4 μsec. If a change of column address is required, this figure increases by about 1 μsec due to switching transients in the read amplifiers. Subsequent accesses in the new column revert to 0.4 μsec. The store operates in conjunction with a subsidiary core store of 1024 words which provides working space for the fixed store programs, and has a cycle time of about 1.8 μsec. There are certain safeguards against a normal machine user gaining access to addresses in either part of the private store, though in effect he makes use of this store through the extracode facility.

The central store of the machine consists of a drum and core store combination, which has a maximum addressable capacity of about 10^6 words. In Muse the central store capacity is about 96,000 words contained on 4 drums. Any part of this store can be transferred in blocks of 512 words to/from the main core store, which consits of four separate stacks, each stack having a capacity of 4096 words.

The tape system provides a very large capacity backing store for the machine. The user can effect transfers of variable amounts of information between this store and the central store. In actual fact such transfers are organized by a fixed store program which initiates automatic transfers of blocks of 512 words between the tape store and the main core store. The system can handle eight tape decks running simultaneously, each producing or demanding a word on average every 88 μsec.

The main core store address can thus be provided from either the central machine, the drum, or the tape system. Since there is no synchronization between these addresses, there has to be a priority system to allocate addresses to the core store. The drum has top priority since it delivers a word every 4 μsec, the tape next priority since words can arise every 11 μsec from 8 decks and the machine uses the core store for the rest of the available time. A priority system necessarily takes time to establish its priority, and so it has been arranged that it comes into effect only at each drum or tape request. Thus the machine is not slowed down in any way when no drum or tape transfers take place. The effect of drum and tape transfers on machine speed is given in Appendix I.

[1] T. Kilburn and R. L. Grimsdale, "A digital computer store with a very short read time," *Proc. IEE*, vol. 107, pt. B, pp. 567–572; November, 1960.

To simplify the control commands given to the drum, tape, and peripheral equipment in the machine, the orders all take the form $b \rightarrow S$ or $s \rightarrow B$ and the identification of the required command register is provided by the address S. This type of storage is clearly widely scattered in the machine but is termed collectively the V-store.

In the central machine the main accumulator contains a fast adder[2] and has built-in multiplication and division facilities. It can deal with fixed or floating point numbers and its operation is completely independent of the B-store and B-arithmetic unit. The B-store is a fast core store (cycle time 0.7 μsec) of 120 twenty-four bit words operating in a word selected partial flux switching mode.[3] Eight "fast" B lines are also provided in the form of flip-flop registers. Of these, three are used as control lines, termed main, extracode, and interrupt controls respectively. The arrangement has the advantage that the control numbers can be manipulated by the normal B-type orders, and the existence of three controls permits the machine to switch rapidly from one to another without having to transfer control numbers to the core store. Main control is used when the central machine is obeying the current program, while the extracode control is concerned with the fixed store subroutines. The interrupt control provides the means for handling numerous peripheral equipments which "interrupt" the machine when they either require or are providing information. The remaining "fast" B lines are mainly used for organizational procedures, though $B124$ is the floating point accumulator exponent.

The operating speed of the machine is of the order of 0.5×10^6 instructions per second. This is achieved by the use of fast transistor logic circuitry, rapid access to storage locations, and an extensive overlapping technique. The latter procedure is made possible by the provision of a number of intermediate buffer storage registers, separate access mechanisms to the individual units of core store and parallel operation of the main accumulator and B-arithmetic units. The word length throughout the machine is 48 bits which may be considered as two half-words of 24 bits each. All store transfers between the central machine, the drum and tape stores are parity checked, there being a parity digit associated with each half-word. In the case of transfers within the central store (*i.e.*, between main core store and drum) the parity digits associated with a given word are retained throughout the system. Tape transfers are parity checked when information is transferred to and from the main core store, and on the tape itself a check sum technique involving the use of two closely spaced heads is used.

The form of the instruction, which allows for two B-modifications, and the allocation of the address digits is shown in Fig. 2(a). Half of the addressable store locations

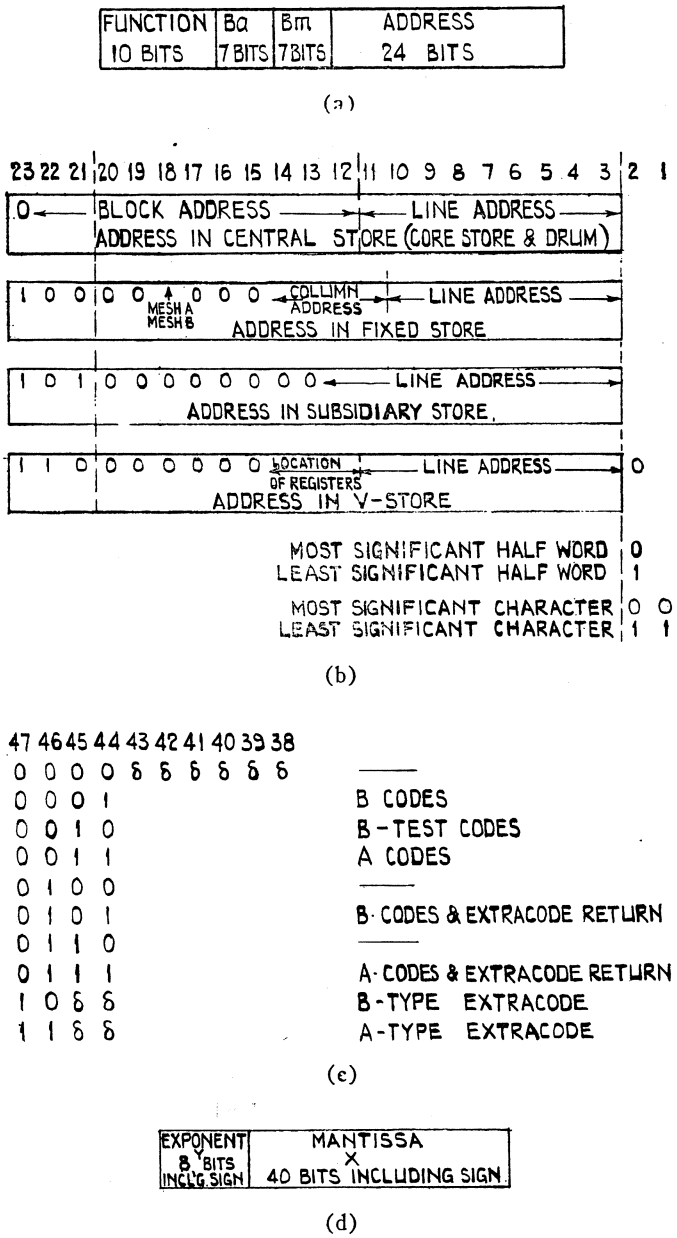


Fig. 2—Interpretation of a word. (a) Form of instruction. (b) Allocation of address digits. (c) Function of decoding. (d) Floating point number $X8^Y$.

are allocated to the central store which is identified by a zero in the most significant digit of the address. [See Fig. 2(b).] This address can be further subdivided into block address, and line address in a block of 512 words. The least significant digits, 0 and 1, make it possible to address 6 bit characters in a half word and digit 2 specifies the half word.

The function number is split into several sections, each section relating to a particular set of operations, and these are listed in Fig. 2(c). The machine orders fall into two broad classes, and these are

1) *B codes:* These involve operations between a B line specified by the B_A digits in the instruction and a core store line whose address can be modified by the contents of a B line determined by the B_m digits. There are a total of 128 B lines, one of which, B_0, al-

[2] T. Kilburn, D. B. H. Edwards and D. Aspinall, "A parallel arithmetic unit using a saturated transistor fast-carry circuit," *Proc. IEE*, vol. 107, pt. B, pp. 573–584; November, 1960.

[3] D. B. G. Edwards, M. J. Lanigan and T. Kilburn, "Ferrite-core memory systems with rapid cycle times," *Proc. IEE*, vol. 107, pt. B, pp. 585–598; November, 1960.

ways contains zero. Of the other lines 90 are available to the machine user, 7 are special registers previously mentioned, and a further 30 are used by extracode orders.

2) *A codes:* These involve operations between the Accumulator and a core store line whose address can now be doubly modified first by contents of B_m and then by the contents of B_A. Both fixed and floating point orders are provided, and in the latter case numbers take the form of $X8^Y$, the digit allocation of X and Y being shown in Fig. 2(d). When fixed point working occurs, use is made only of the X digits.

III. One Level Store Concept

The choice of system for the fast access store in a large scale computer is governed by a number of conflicting factors which include speed and size requirements, economic and technical difficulties. Previously the problem has been resolved in two extreme cases either by the provision of a very large core store, *e.g.*, the 2.5 megabit[4] store at M.I.T., or by the use of a small core store (40,000 bits) expanded to 640,000 bits by a drum store as in the Ferranti Mercury[5] computer. Each of these methods has its disadvantages, in the first case, that of expense, and in the second case, that of inconvenience to the user, who is obliged to program transfers of information between the two types of store and this can be time consuming. In some instances it is possible for an expert machine user to arrange his program so that the amount of time lost by the transfers in the two-level storage arrangement is not significant, but this sort of "optimum" programming is not very desirable. Suitable interpretative coding[6] can permit the two-level system to appear as one level. The effect is, however, accompanied by an effective loss of machine speed which, in some programs and depending on details of machine design can be quite severe, varying typically, for example, between one and three.

The two-level storage scheme has obvious economic advantages, and inconvenience to the machine user can be eliminated by making the transfer arrangements completely automatic. In Atlas a completely automatic system has been provided with techniques for minimizing the transfer times. In this way the core and drum are merged into an apparent single level of storage with good performance and at moderate cost. Some details of this arrangement on the Muse are now provided.

The central store is subdivided into blocks of 512 words as shown by the address arrangements in Fig. 2(b). The main core store is also partitioned into blocks of this size which for identification purposes are called pages. Associated with each of these core store page positions is a "page address register" (P.A.R.) which contains the address of the block of information at present occupying that page position. When access to any word in the central store is required the digits of the demanded block address are compared with the contents of all the page address registers. If an "equivalence" indication is obtained then access to that particular page position is permitted. Since a block can occupy any one of the 32 page positions in the core store it is necessary to modify some digits of the demanded block address to conform with the page positions in which an equivalence was obtained.

These processes are necessarily time consuming but by providing a by-pass of this procedure for instruction accesses (since, in general, instruction loops are all contained in the same block) then most of this time can be overlapped with a useful portion of the machine or core store rhythm. In this way information in the core store is available to the machine at the full speed of the core store and only rarely is the over-all machine speed effected by delays in the equivalence circuitry.

If a "not equivalence" indication is obtained when the demanded block address is compared with the contents of the P.A.R.'s then that address, which may have been B-modified, is first stored in a register which can be accessed as a line of the V-store. This permits the central machine easy access to this address. An "interrupt" also occurs which switches operation of the machine over to the interrupt control, which first determines the cause of the interrupt and then, in this instance, enters a fixed store routine to organize the necessary transfers of information between drum and core store.

A. Drum Transfers

On each drum, one track is used to identify absolute block positions around the drum periphery. The records on these tracks are read into the θ registers which can be accessed as lines of the V-store and this permits the present angular drum position to be determined, though only in units of one block. In this way the time needed to transfer any block while reading from the drums can be assessed. This time varies between 2 and 14 msec since the drum revolution time is 12 msec and the actual transfer time 2 msec.

The time of a writing transfer to the drums has been reduced by writing the block of information to the first available empty block position on any drum. Thus the access time of the drum can be eliminated provided there are a reasonable number of empty blocks on the drum. This means, however, that transfers to/from the drum have to be carried out by reference to a directory and this is stored in the subsidiary store and up-dated whenever a transfer occurs.

When the drum transfer routine is entered the first action is to determine the absolute position on a drum of the required block. The order is then given to carry out the transfer to an empty page position in the core store. The

[4] W. N. Papian, "High-speed computer stores 2.5 megabits," *Electronics*, vol. 30; October, 1957.

[5] K. Lonsdale and E. T. Warburton, "Mercury: a high speed digital computer," *Proc. IEE*, vol. 103, pt. B (suppl. 2), pp. 174–183; 1956.
T. Kilburn, D. B. G. Edwards, and C. E. Thomas, "The Manchester University Mark II Digital Computing Machine," *Proc. IEE*, vol. 103, pt. B (suppl. 2), pp. 247–268; 1956.

[6] R. A. Brooker, "Some techniques for dealing with two-level storage," *The Computer Journal*, vol. 2; 1960.

transfer occurs automatically as soon as the drum reaches the correct angular position. The page address register in the vacant position in the core store is set to a specific block number for drum transfers. This technique simplifies the engineering with regard to the provision of this number from the drum and also provides a safeguard against transferring to the wrong block.

As soon as the order asking for a read transfer from the drum has been given the machine continues with the drum transfer program. It is now concerned with determining a block to be transferred back from the core store to the drum. This is necessary to ensure an empty core store page position when the next read transfer is required. The block in the core store to be transferred has to be carefully chosen to minimize the number of transfers in the program and this optimization process is carried out by a learning program, details of which are given in Section V. The operation of this program is assisted by the provision of the "use" digits which are associated with each page position of the core store.

To interchange information between the core store and drums, two transfers, a read from and a write to the drum are necessary. These have to be done sequentially but could occur in either order. The technique of having a vacant page position in the core store permits a read transfer to occur first and thus allows the time for the learning program to be overlapped either into the waiting period for the read transfer or into the transfer time itself. In the time remaining after completion of the learning program an entry is made into the over-all supervisor program for the machine, and a decision is taken concerning what the machine is to do until the drum transfer is completed. This might involve a change to a different main program.

A program could ask for access to information in a page position while a drum or tape transfer is taking place to that page. This is prevented in Atlas by the use of a "lock out" (L.O.) digit which is provided with each Page Address Register. When a lock out digit is set at 1, access to that page is only permitted when the address has been provided either by the drum system, the tape system, or the interrupt control. The latter case permits all transfers from paper tape, punched card, and other peripheral equipments, to be handled without interference from the main program. When the transfer of a block has been completed the organizing program resets the L.O. digit to zero and access to that page position can then be made from the central machine. It is clear that the L.O. digit can also be used to prevent interference between programs when several different ones are being held in the machine at the same time.

In Section III it was stated that addresses demanding access to the core store could arise from three distinct sources, the central machine, the drum, and the tape. These accesses are complicated because of 1) the equivalence technique, and 2) the lock out digit. The various cases and the action that takes place are summarized in Table I.

TABLE I

COMPARISON OF DEMANDED BLOCK ADDRESS WITH CONTENTS OF THE P.A.R.'S RESULTANT STATE OF EQUIVALENCE AND LOCK OUT CIRCUITS

Source of Address	{Equivalence, Lock out = 0} [E.Q.]	Not Equivalence [N.E.Q.]	{Equivalence, Lock out = 1} [E.Q. & L.O.]
1. Central Machine	Access to required page position	Enter drum transfer routine	Not available to this program
2. Drum System	Access to required page position	Fault condition indicated	Fault condition indicated
3. Tape System	Access to required page position	Fault condition indicated	Fault condition indicated

The provision of the Page Address Registers, the equivalence circuitry, and the learning program have permitted the core store and drum to be regarded by the ordinary machine user as a one level store, and the system has the additional feature of "floating address" operation, *i.e.*, any block of information can be stored in any absolute position in either core or drum store. The minimum access time to information in this store is obviously limited by the core store and its arrangement and this is now discussed.

B. Core Store Arrangement

The core store is split into four stacks, each with individual address decoding and read and write mechanisms. The stacks are then combined in such a way that common channels into the machine for the address, read and write digits are time shared between the various stacks. Sequential address positions occur in two stacks alternately and a page position which contains a block of 512 sequential addresses is thus arranged across two stacks. In this way it is possible to read a pair of instructions from consecutive addresses in parallel by increasing the size of the read channel. This permits two instructions to be completely obeyed in three store "accesses." The choice of this particular storage arrangement is discussed in Appendix II.

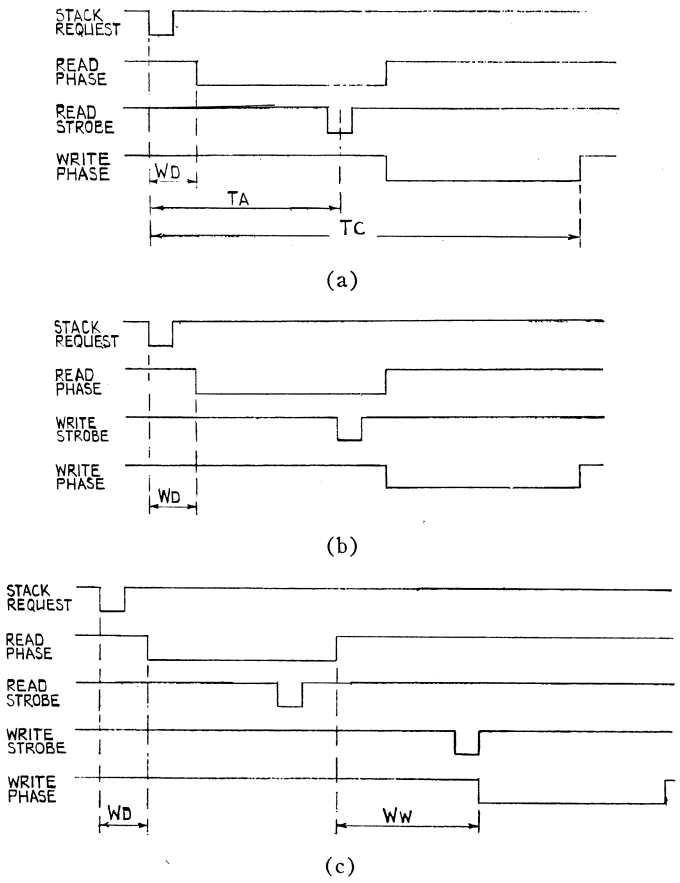
The coordination of these four stacks is done by the "core stack coordinator" and some features of this are now discussed, starting with the operation of a single stack.

C. Operation of a Single Stack of Core Store

The storage system employed is a coincident current M.I.T. system arranged to give parallel read out of 50 digits. The reading operation is destructive and each read phase of the stack cycle is followed by a write phase during

which the information read out may be rewritten. This is achieved by a set of digit staticizors which are loaded during the read phase and are used to control the inhibit current drivers during the write phase. When new information is to be written into the store a similar sequence is followed, except that the digit staticizors are loaded with the new information during the read phase. A diagram indicating the different types of stack cycle is shown in Fig. 3.

There is a small delay W_D ($\simeq$100 mμsec) between the "stack request" signal, SR, and the start of the read phase to allow for setting of the address state and the address decoding. The output information from the store appears in the read strobe period, which is towards the end of the read phase. In general, the write phase starts as soon as the read phase ends. However, the start of the write phase may be held up until the new information is available from the central machine. This delay is shown as W_w in Fig. 3(c). The interval T_A between the stack request and the read strobe is termed the stack access time, and in practice this is approximately one third of the cycle time T_C. Both T_A and T_C are functions of the storage system and assuming that W_w is zero have typical values of 0.7 μsec and 1.9 μsec respectively. A holdup gate in the request channel prevents the next stack request occurring before the end of the preceding write phase.



T_A = Access time, T_C = Cyclic time, W_D = Wait for address decoding and loading of address register, W_W = Wait for release of write hold up.

Fig. 3—Basic types of stack cycle. (a) Read order ($s \rightarrow A$). (b) Write order ($a \rightarrow s$). (c) Read-write order ($b+s \rightarrow S$).

D. Operation of the Main Core Store with the Central Machine

A schematic diagram of the essentials of the main core store control system is shown in Fig. 4. The control signals SA_1 and SA_2 indicate whether the address presented is that of a single word or a pair of sequentially addressed instructions. Assuming that the flip-flop F is in the reset condition, either of these signals results in the loading of the buffer address register (B.A.R.). This loading is done by the signal B.A.B.A. which also indicates that the buffer register in the central machine has become free.

In dealing with the first request the block address digits in the B.A.R. are compared with the contents of all the page address registers. Then one of the indications summarized in Table I and indicated in Fig. 4 is obtained. Assuming access to the required store stack is permitted then a set C.S.F. signal is given which resets the flip-flop F. If this occurs before the next access request arises, then the speed of the system is not store-limited. In most cases SET CSF is generated when the equivalence operation on

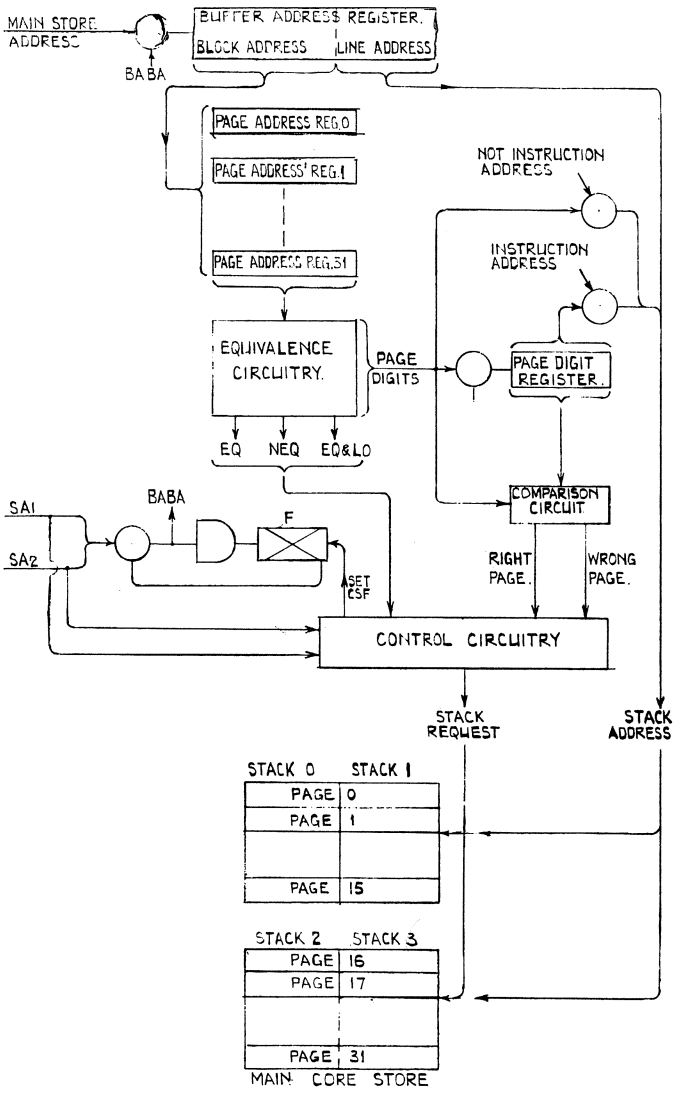


Fig. 4—Main core store control.

the demanded block address is complete, and the read phase of the appropriate stack (or stacks) has started. Until this time the information held in the B.A.R. must not be allowed to change. In Fig. 5 a flow diagram is shown for the various cases which can arise in practice.

When a single address request is accepted it is necessary to obtain an "equivalence" indication and form the page location digits before the stack request can be generated. The SET CSF signal then occurs as soon as the read phase starts. If a "not equivalent" or "equivalent and locked out" indication is obtained a stack request is not generated, and the contents of the B.A.R. are copied in to a line of the *V*-store before SET CSF is generated.

When access to a pair of addresses is requested (*i.e.*, an instruction pair) the stack requests are generated on the assumption that these instructions are located in the same page position as the last pair requested, *i.e.*, the page position digits are taken from the page digit register. (See Fig. 4.) In this way the time required to obtain the equivalent indication and form the page location digits is not included in the over-all access time of the system. The assumption will normally be true, except when crossing block boundaries. The latter cases are detected and corrected by comparing the true position page digits obtained as a result of the equivalence operation with the contents of the page digit register and a "right page" or "wrong page" indication is obtained. (See Fig. 4.) If a wrong page is accessed this is indicated to the central machine and the read out is inhibited. The true page location digits are copied into the page digit register, so that the required instruction pair will be obtained when next requested. The read out to the central machine is also inhibited for "not equivalent" or "equivalent and locked out" indications.

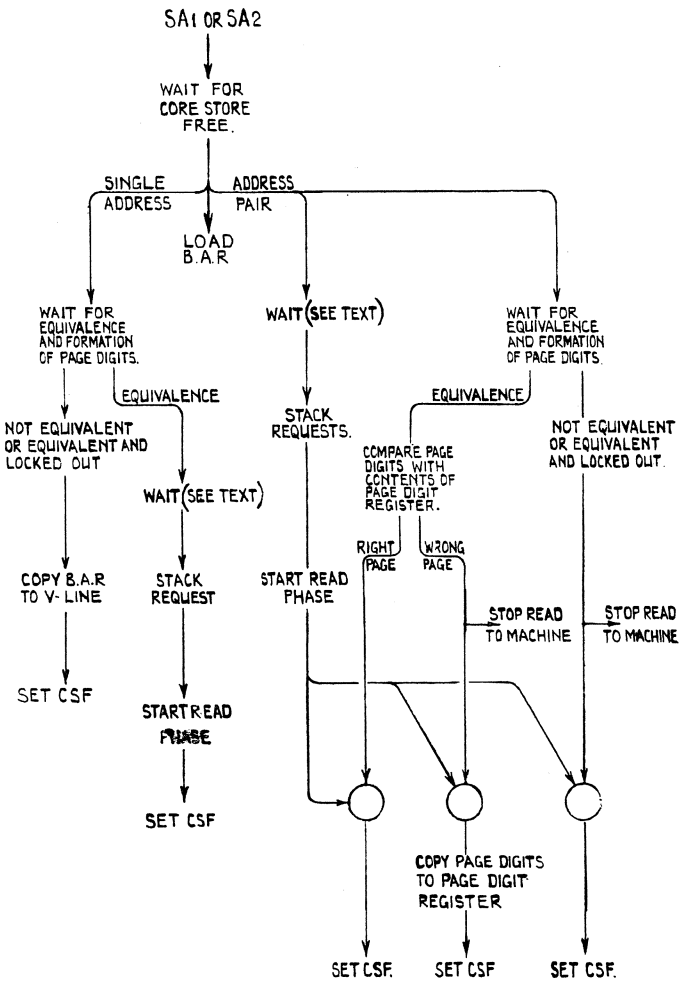


Fig. 5—Flow diagram of main core store control.

In Fig. 5 the waiting time indicated immediately before the stack request is generated can arise for a number of reasons.

1) The preceding write phase of that stack has not yet finished.
2) The central machine is not yet ready either to accept information from the store, or to supply information to it.
3) It is necessary to ensure a certain minimum time between successive read strobes from the core store stacks to allow satisfactory operation of the parity circuits, which take about 0.4 μsec to check the information. This time could be reduced, but as it is only possible to get such a condition for a small part of the normal instruction timing cycle it was not thought to be an economical proposition.

The basic machine timing is now discussed.

IV. Instruction Times

In high-speed computers, one of the main factors limiting speed of operation is the store cycle time. Here a number of techniques, *e.g.*, splitting the core store into four separate stacks and extracting two instructions in a single cycle, have been adopted despite a fast basic cycle time of 2 μsec in order to alleviate this situation. The time taken to complete an instruction is dependent upon

1) The type of instruction (which is defined by the function digits),
2) The exact location of the instruction and operand in the core or fixed store since this can affect the access time,
3) Whether or not the operand address is to be modified,
4) In the case of floating point accumulator orders, the actual numbers themselves,
5) Whether drum and/or tape transfers are taking place.

The approximate times for various instructions are given in Table II. These figures relate to the times between completing instructions when a long sequence of the same type of instruction is obeyed. While this method is not ideal, it is necessary because in practice obeying one instruction is overlapped in time with some part of three other instructions. This makes the detailed timing complicated, and so the timing sequence is developed slowly by first considering instructions obeyed one after another. It is convenient to make these instructions a sequence of floating point additions with both instruction and operand in the core store and with the operand address single *B*-modified.

TABLE II

APPROXIMATE INSTRUCTION TIMES

Type of Instruction	Number of Modifications of Address	Instruction in Core Store. Operands in Core Store. Time in μsec	Instructions in Fixed Store. Operands in Core Store. Time in μsec	Instructions in Fixed Store. Operands in Fixed Store. Time in μsec
Floating Point Addition	0	1.4	1.65	1.2
	1	1.6	1.65	1.2
	2	2.03	1.9	1.9
Floating Point Multiplication	0, 1 or 2	4.7	4.7	4.7
Floating Point Division	0, 1 or 2	13.6	13.6	13.6
Add Store Line to an Index Register	0	1.53	1.65	1.15
	1	1.85	1.85	1.85
Add Index Register to Store Line and Rewrite to Store Line	0	1.63	1.65	—
	1	1.8	1.7	—

To obey this instruction the central machine makes two requests to the core store, one for the instruction and the second for the operand. After the instruction is received in the machine the function part has to be decoded and the operand address modified by the contents of one of the *B* registers before the operand request can be made. Finally, after the operand has been obtained the actual accumulator addition takes place to complete the instruction. The time from beginning to end of one instruction is 6.05 μsec and an approximate timing schedule is as follows in Table III.

If no other action is permitted in the time required to complete the instruction (steps 1 to 8 in Table III), then the different sections of the machine are being used very inefficiently, *e.g.*, the accumulator adder is only used for less than 1.1 μsec. However, the organization of the computer is such that the different sections such as store stacks, accumulator and *B*-arithmetic unit, can operate at the same time. In this way several instructions can be started before the first has finished, and then the effective instruction time is considerably reduced. There have, of course, to be certain safeguards when for example an instruction is dependent in any way on the completion of a preceding instruction.

In the time sequence previously tabulated, by far the longest time was that between a request in the central machine for the core store and the receipt in the central machine of the information from that store. This effective access time of 1.75 μsec is made up as shown in Table IV. It has been reduced in practice by the provision of two buffer registers, one in the central machine and the other in the core stack coordinator. These allow the equivalence and transfer times to be overlapped with the organization of requests in the central machine.

In this way, provided the machine can arrange to make requests fast enough, then the effective access time is reduced to 0.8 μsec. Further, since three accessses are needed to complete two instructions (one for an instruction pair and one for each of the two operands) the theoretical minimum time of an instruction is 1.2 μsec $3\times0.8/2$ and it then becomes store limited. Reference to Table III shows that the arithmetic operation takes 1.2 μsec to complete so that, on the average, the capabilities of the store and the accumulator are well matched.

Another technique for reducing store access time for instructions has also been adopted. This permits the read cycles of the two stacks to start assuming that the same page will be referred to as in the previous instruction pair.

TABLE III*

TIMING SEQUENCE FOR FLOATING POINT ADDITION
(Instructions and Operands in the Core Store)

Sequence	Time Interval Between Steps μsec	Total Time μsec
1. Add 1 to Main Control (Addition time)	0.3	0
2. Make Instruction Request (Transfer times, equivalence time and stack access time)	1.75	0.3
3. Receive Instruction in Central Machine (Load register and decode)	0.2	2.05
4. Function decoding complete (Single address modification)	0.85	2.25
5. Request Operand (Transfer times, equivalence time and stack access time)	1.75	3.10
6. Receive Operand in Central Machine (Load register)	0.1	4.85
7. Start Addition in Accumulator (Average floating point addition, including shift round and standardise)	1.1	4.95
8. Instruction complete		6.05

* In step 4, time is for single address modification. Times for no modification and two modifications are 0.25 μsec and 1.55 μsec respectively.

TABLE IV

EFFECTIVE STORE ACCESS TIME

Sequence	Total Time μsec
1. Request in Central Machine	0
2. Request in Core Stack Coordinator	0.25
3. Equivalence complete and request made to selected stack	0.95
4. Information in Core Stack Coordinator	1.65
5. Information in Central Machine	1.75

This, of course, will normally be true and there is sufficient time to take corrective procedures should the page have been changed. The limit of 1.2 μsec per instruction is not reduced by this technique, but the possibility of reaching this limit under other conditions is enhanced.

A schematic diagram of the practical timing of a sequence of floating point addition orders is shown in Fig. 6. The overlapping is not perfect and in the time between successive instruction pairs the computer is obeying four instructions for 25 per cent of the time, three for 56 per cent and two for 19 per cent. It is therefore to be expected that the practical time for the complete order is greater than the theoretical minimum time; it is in fact approximately 1.6 μsec.

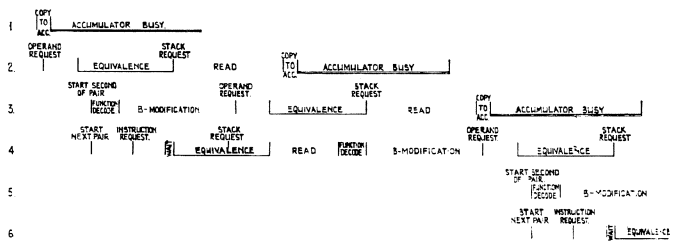


Fig. 6—Timing diagram for a sequence of floating point addition orders. (Single address modification.)

For certain types of functions the reading of the next pair of instructions before completing both instructions of the first pair would be incorrect, *e.g.*, functions causing transfer of control. Such situations are recognized during the function decoding, and the request for the next instruction pair is held up until a suitable time.

In a sequence of floating point addition orders with the operand addresses unmodified the limit is again 1.2 μsec while the time obtained is 1.4 μsec. For accumulator orders in which the actual accumulator operation imposes a limit in excess of 2 μsec then the actual time is equal to this limit.

Perhaps a more realistic way of defining the speed of the computer is to give the time for a typical inner loop of instructions. A frequently occurring operation in matrix work in the formation of the scalar product of two vectors, this requires a loop of five instructions:

1) Element of first vector into accumulator. (Operand *B*-modified.)
2) Multiply accumulator by element of second vector. (Operand *B*-modified.)
3) Add partial product to accumulator.
4) Copy accumulator to store line containing partial product.
5) Alter count to select next elements and repeat.

The time for this loop with instructions and operands on the core store is 12.2 μsec. The value of the overlapping technique is shown by the fact that the time from starting the first instruction to finishing the second is approximately 10 μsec.

When the drum or tape systems are transferring information to or from the core store then the rate of obeying instructions which also use the core store will be affected. The affect is discussed in more detail in Appendix I. The degree of slowing down is dependent upon the time at which a drum or tape request occurs relative to machine requests. It also depends on the stacks used by the drum or tape and those being used by the central machine. The approximate slowing down is by a factor of 25 per cent during a drum transfer and by 2 per cent for each active tape channel. (See Appendix I.)

V. The Drum Transfer Learning Program

The organization of drum transfers has been described in Section IIA. After the transfer of the required block from the drum to the core store has been initiated, the organizing program examines the state of the core store, and if empty pages still exist, no further action is taken. However, if the core store is full it is necessary to arrange for an empty page to be made available for use at the next non-equivalence. The selection of the page to be transferred could be made at random; this could easily result in many additional transfers occurring, as the page selected could be one of those in current use or one required in the near future. The ideal selection, which would minimize the total number of transfers, could only be made by the programmer. To make this ideal selection the programmer would have to know, 1) precisely how his program operated, which is not always the case, and 2) the precise amount of core store available to his program at any instant. This latter information is not generally available as the core store could be shared by other central machine programs, and almost certainly by some fixed store program organizing the input and output of information from slow peripheral equipments. The amount of core store required by this fixed store program is continuously varying.[7] The only way the ideal pattern of transfers can be approached is for the transfer program to monitor the behavior of the main program and in so doing attempt to select the correct pages to be transferred to the drum. The techniques used for monitoring are subject to the condition that they must not slow down the operation of the program to such an extent that they offset any reduction in the number of transfers required. The method described occupies less than 1 per cent of the operating time, and the reduction in the number of transfers is more than sufficient to cover this.

That part of the transfer program which organizes the selection of the page to be transferred has been called the "learning" program. In order for this program to have some data on which to operate, the machine has been designed to supply information about the use made of the different pages of the core store by the program being monitored.

[7] T. Kilburn, D. J. Howarth, R. B. Payne and F. H. Sumner, "The Manchester University Atlas Operating System. Part I: Internal Organisation," *The Computer Journal*, vol. 4; October, 1961.

With each page of the core store there is associated a "use" digit which is set to "1" whenever any line in that page is accessed. The 32 "use" digits exist in two lines of the V-store and can be read by the learning program, the reading automatically resetting them to zero. The frequency with which these digits are read is governed by a clock which measures not real time but the number of instructions obeyed in the operation of the main program. This clock causes the learning program to copy the "use" digits to a list in the subsidiary store every 1024 instructions. The use of an instruction counter rather than a normal clock to measure "time" for the learning program is due to the fact that the operations of the main program may be interrupted at random for random lengths of time by the operation of peripheral equipments. With an instruction counter the temporal pattern of the blocks used will be the same on successive runs through the same part of the program. This is essential if the learning program is to make use of this pattern to minimize the number of transfers.

When a nonequivalence occurs and after the transfer of the required block has been arranged, the learning program again adds the current values of the "use" digits to the list and then uses this list to bring up to date two sets of times also kept in the subsidiary store. These sets consist of 32 values of t and T, one of each for each page of the core store. The value of t is the length of time since the block in that page has been used. The value of T is the length of the last period of inactivity of this block. The accuracy of the values of t and T is governed by the frequency with which the "use" digits are inspected.

The page to be written to the drum is selected by the application in turn of three simple tests to the values of t and T.

1) Any page for which $t>T+1$,
or 2) That page with $t\neq 0$ and $(T-t)$ max,
or 3) That page with $T_{\max}$ (all $t=0$).

The first rule selects any page which has been currently out of use for longer than its last period of inactivity. Such a page has probably ceased to be used by the program and is therefore an ideal one to be transferred to the drum. The second rule ignores all pages with $t=0$ as they are in current use, and then selects the one which, if the pattern of use is maintained, will not be required by the program for the longest time. If the first two rules fail to select a page the third ensures that if the page finally selected is wrong, in that it is immediately required again, then, as in this case, T will become zero and the same mistake will not be repeated.

For all the blocks on the drum a list of values of τ is kept. The values of τ are set when the block is transferred to the drum:

τ = Time of transfer—value of t for transferred page.

When a block is transferred to the core store the value of τ is used to set the value of T.

T = Time of transfer—value of τ for this block
= Length of last period of inactivity.

For the block transferred from the drum t is set to 0.

In order to make its decision the learning program has only to update two short lists and apply at the most three simple rules; this can easily be done during the 2 msec transfer time of the block required as a result of the nonequivalence. As the learning program uses only fixed and subsidiary store addresses it is not slowed down during the period of the drum transfer.

The over-all efficiency of the learning program cannot be known until the complete Atlas system is working. However, the value of the method used has been investigated by simulating the behavior of the one-level store and learning program on the Mercury computer at Manchester University. This has been done for several problems using varying amounts of store in excess of the core store available. One of these was the problem of forming the product A of two 80th order matrices B and C. The three matrices were stored row by row each one extending over 14 blocks, only 14 pages of core store were assumed to be avilable. The method of multiplication was

$b_{11}\times$1st row of C = partial answer to 1st row of A,
$b_{12}\times$2nd row of C + partial answer = second partial answer, etc.,

thus matrix B was scanned once, matrix C 80 times and each row of matrix A 80 times.

Several machine users were asked to spend a short time writing a program to organize the transfers for a general matrix multiplication problem. In no case when the method was applied to the above problem were fewer than 357 transfers required. A program written specifically for this problem which paid great attention to the distribution of the rows of the matrices relative to block divisions required 234 transfers. The learning program required 274 transfers, the gain over the human programmer was chiefly due to the fact that the learning program could take full advantage of the occasions when the rows of A existed entirely within one block.

Many other problems involving cyclic running of single or multiple sets of data were simulated, and in no case did the learning program require more transfers than an experienced human programmer.

A. Prediction of Drum Transfers

Although the learning program tends to reduce the number of transfers required to a minimum, the transfers which do occur still interrupt the operation of the program for from 2 to 14 msec as they are initiated by nonequivalence interrupts. Some or all of this time loss could be avoided by organizing the transfers in advance. A very experienced programmer having sole use of the core store could arrange his own transfers in such a way that no unnecessary ones ever occurred and no time was ever wasted waiting for transfers to be completed. This would require a great deal of effort and would only be worthwhile for a program that was going to occupy the machine for a long time. By using the data accumulated by the learning program it is possible to recognize simple patterns in the use made by a

program of the various blocks of the one level store. In this way a prediction program could forecast the blocks required in the near future and organize the transfers. By recording the success or failure of these forecasts the program could be made self-improving. For the matrix multiplication problem discussed above the pattern of use of the blocks containing matrix C is repeated 80 times, and a considerable degree of success could be obtained with a simple prediction program.

VI. Conclusions

A specific system for making a core-drum store combination appear as a single level store has been described. While this is the actual system being built for the Atlas machine the principles involved are applicable to combinations of other types of store. For example, a tunnel diode-fast core store combination for an even faster machine. An alternative which was considered for Atlas, but which was not as attractive economically, was a fast core-slow core store combination. The system too can be extended to three levels of storage, and indeed if 10^6 words of total storage had to be provided then it would be most economical to provide it on a third level of store such as a file drum.

The automatic system does require additional equipment and introduces some complexity, since it is necessary to overlap the time taken for address comparison into the store and machine operating time if it is not to introduce any extra time delays. Simulated tests have shown that the organization of drum transfers are reasonably efficient and other advantages which accrue, such as efficient allocation of core storage between different programs and store lock out facilities are also invaluable. No matter how intelligent a programmer may be he can never know how many programs or peripheral equipments are in operation when his program is running. The advantage of the automatic system is that it takes into account the state of the machine as it exists at any particular time. Furthermore if as in normal use there is some sort of regular machine rhythm even through several programs, there is the possibility of making some sort of prediction with regard to the transfers necessary. This involves no more hardware and will be done by program. However, this stage will probably be left until results on the actual system are obtained.

It can be seen that the system is both useful and flexible in that it can be modified or extended in the manner previously indicated. Thus despite the increase in equipment, the advantages which are derived completely justify the building of this automatic system.

VII. Appendix I

Organization of the Access Requests to the Core Store

There are three sources of access requests to the core store, namely the central machine, the drum, and the tape systems. In deciding how the sequence of requests from all three sources are to be serialized and placed in some sort of order, a number of facts have to be considered. These are

1) All three sources are asynchronous in nature.
2) The drum and tape systems can make requests at a fairly high rate compared with the store cycle time of approximately 2 μsec. For example, the drum provides a request every 4 μsec and the tape system every 11 μsec when all 8 channels are operative.
3) The drum and tape systems can only be stopped in multiples of a block length, *i.e.*, 512 words. This means that any system devised for accessing the core store must deal with both the average rates of drum drum and tape requests specified in 2). Only the central machine can tolerate requests being stopped at any time and for any length of time. From these facts a request priority can be stated which is
   a) Drum request.
   b) Tape request.
   c) Central machine request.
4) A machine request can be accepted by the core store, but because there is no place available to accept the core store information, its cycle is inhibited and further requests held up. In the case of successive division orders this time can be as long as 20 μsec, in which case 5 drum requests could be made. To avoid having an excessive amount of buffer storage for the drum two techniques are possible:
   a) When drums or tapes are operative do not permit machine requests to be accepted until there is a place available to put the information.
   b) Store the machine request and then permit a drum or tape request.

   The latter scheme has been adopted because it can be accommodated more conveniently and it saves a small amount of time.
5) If the central machine is using the private store then it is desirable for drum and tape transfers to the core store not to interfere with or slow down the central machine in any way.
6) When the central machine, drum and tape are sharing the core store then the loss of central machine speed should be roughly proportional to the activity of the drum or tape systems. This means that drum or tape requests must "break" into the normal machine request channel as and when required.

The system which accommodates all these points is now discussed. Whenever a drum or tape request occurs inhibit signals are applied to request channel into the core stack coordinator and also to the stack request channels from this coordinator. This results in a "freezing" of the state of flip-flop F (Fig. 5) and this state is then inspected (Fig. 7, point X). If the state is "busy" this means that a machine order has been stopped somewhere between the loading of the buffer address register (B.A.R.) and the stack request. Normally this time interval can vary from about 0.5 μsec if there are no stack request holdups, to 20 μsec in the case of certain accumulator holdups. In either case sufficient time is allowed after the inspection to ensure that the equivalence operation has been completed. If an equiva-

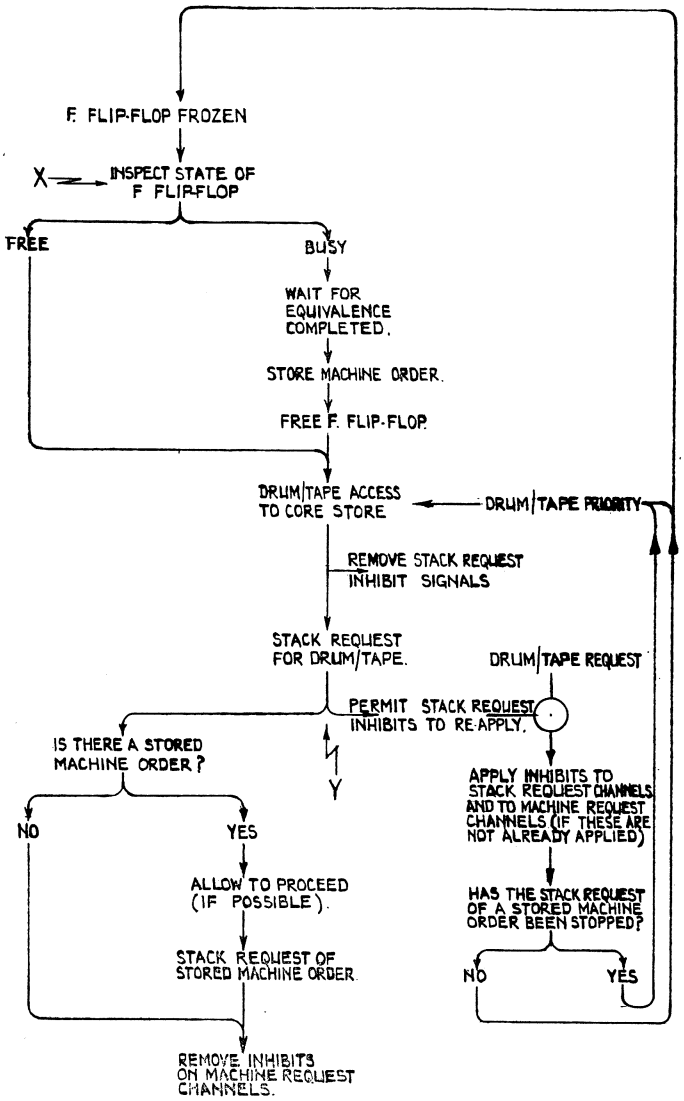


Fig. 7—Drum and tape break in systems.

lence indication is obtained all the information relevant to this machine order (*i.e.*, the line address, page digits, stack(s) required and type of stack order) are stored for future reference. Use is made here of the page digit register provided to allow the by-pass on the equivalence circuitry for instruction accesses. The core store is then made free for access by the drum or the tape. If the core store had been found to be free on inspection, the above procedure is omitted.

A drum or tape access (as decided by the priority circuit) to the core store then occurs, which removes the inhibits on the stack request channels. When the stack request for the drum or tape cycle is initiated these inhibits are allowed to reapply. At this stage (Fig. 7, point *Y*), if there is a stored machine order it is allowed to proceed if possible. The inhibits on the machine request channels are removed when the stack request for the stored machine order occurs. If there is no stored machine order this is done immediately, and the central machine is again allowed access to the core store. However, another drum or tape request can arise before the stack request of the stored machine order occurs, in particular because this latter order may still be held up by the central machine. If this is the case the drum or tape is allowed immediate access and a further attempt is made to complete the stored machine order when this drum or tape stack request occurs.

If the stored machine order was for an operand, the content of the page digit register will correspond to the location of this operand. The next machine request for an instruction pair will then almost certainly result in a "wrong page" indication. This is prevented by arranging that the next instruction pair access does not by-pass the equivalence circuitry.

The effect on the machine speed when the drum or tapes are transferring information to or from the core store is dependent upon two factors. First, upon the proportion of time during which the buffer register in the core coordinator is busy dealing with machine requests, and secondly, upon the particular stacks being used by the central machine and the drum or tape. If the computer is obeying a program with instructions and operands on the fixed or subsidiary store then the rate of obeying instructions is unaffected by drum or tape transfers. A drum or tape interrupt occuring when the B.A.R. is free prevents any machine address being accepted onto this buffer for 1.0 μsec. However, if the B.A.R. is busy then the next machine request to the core store is delayed until 1.8 μsec after the interrupt if different stacks are being used, or until 3.4 μsec after the interrupt if the stacks are the same.

When the machine is obeying a program with instructions and operands on the core store the slowing down during drum transfers can be by a factor of two if instructions, operands, and drum requests use the same stacks. It is also possible for the machine to be unaffected. The effect on a particular sequence of orders can be seen by considering the one discussed in Section IV and illustrated in Fig. 6. In this sequence the instructions are on stacks 0 and 1 while the operands are on stacks 2 and 3. If the drum or tape is transferring alternately to stacks 0 and 1 then the effect of any interrupt within the 3.2 μsec of an instruction pair is to increase this time by between 0.5 and 3.4 μsec depending upon where the interrupt occurred. The average increase is 1.8 μsec and for a tape transfer with interrupts every 88 μsec the computer can obey instructions at 98 per cent of the normal rate. During drum transfers the interrupts occur every 4 μsec which would suggest a slowing down to 60 per cent of normal. However, for any regular sequence of orders the requests to the core store by the machine and by the drum rapidly become synchronized with the result in this particular case that the machine can still operate at 80 per cent of its normal speed.

Appendix II

Methods of Division of the Main Core Store

The maximum frequency with which requests can be dealt with by a single stack core store is governed by the cycle time of the store. If the store is divided into several stacks which can be cycled independently then the limit

imposed on the speed of the machine by the core store is reduced. The degree of division which is chosen is dependent upon the ratio of core store cycle time to other machine operations and also upon the cost of the multiple selection mechanisms required.

Considering a sequence of orders in which both the instruction and operand are in the core store, then for a single stack store the limit imposed on the operating speed by the store is two cycle times per order, *i.e.*, 4 μsec in Atlas. This is significantly larger than the limits imposed by other sections of the computer (Section IV). If the store is divided into two stacks and instructions and operands are separated, then the limit is reduced to 2 μsec which is still rather high. The provision of two stacks permits the addressing of the store to be arranged so that successive addresses are in alternate stacks. It is therefore possible by making requests to both stacks at the same time to read two instructions together, so reducing the number of access times to three per instruction pair. Unfortunately such an arrangement of the store means that operands are always on the same stacks as instruction pairs, and the limit imposed by the cycle time is still 2 μsec per order even if the two operand requests in the instruction pair are to different stacks and occur at the same time.

Division into any number of stacks with the addressing system working through each stack in turn cannot reduce the limit below 2 μsec since successive instructions normally occur in successive addresses and are therefore in the same stack. However, four stacks arranged in two pairs reduces the limit to 1 μsec as the operands can always be arranged to be on different stacks from the instruction pairs. In order to reduce the limit to 0.5 μsec it is necessary to have eight stacks arranged in two sets of four and to read four instructions at once, which would increase the complexity of the central machine.

The limit of 1 μsec is quite sufficient and further division with the stacks arranged in pairs only enables the limit to be more easily obtained by suitable location of the instructions and operands.

The location of instructions and operands within the core store is under the control of the drum transfer program; thus when there are several stacks instructions and operands are separated wherever possible. Under these

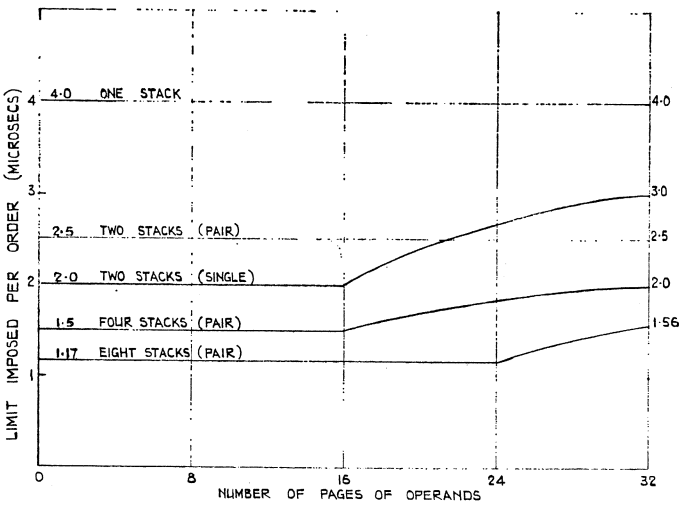


Fig. 8—Limit imposed by cycle time on operating speed for difficult divisions of the core store.

conditions it is possible to calculate the limit imposed on the operating speed by the cycle time for different divisions of the core store. The results are shown in Fig. 8, for stacks arranged in pairs instructions are read in pairs and in all cases both instructions and operands are assumed to be on the core store. Operands are assumed to be selected at random from the operand space, for instance in the case of two stacks arranged as a pair, successive operand requests have equal probability of being to the same stack or to alternate stacks.

The limit imposed by a four stack store is never severe compared with other limitations, for example the sequence of floating point addition orders discussed in Section IV required 1.6 μsec per order with ideal distribution of instructions and operands. Division into eight stacks, although it reduces the limit, will not have an equivalent effect on the over-all operating speed, and such a division was not considered to be justified.

Acknowledgment

The authors gratefully acknowledge the contributions made to this work by all members of the Atlas computer team at both Manchester University and Ferranti Ltd.

THE FACILITIES OF ATLAS BASIC LANGUAGE

ROUTINES:

Rn (1 ⩽ n ⩽ 3999)

PARAMETERS:

Routine: An (1 ⩽ n ⩽ 3999)
Preset: Pn (0 ⩽ n ⩽ 99)
Global: Gn (0 ⩽ n ⩽ 3999)

OPERATORS:

Logical shifts: Da, Ua.
Masks: B (block), W (word).
Binary complement: ' (apostrophe).

SEPARATORS:

+, −, X, Q (divide: quotient is always an integer), & (or M), V ('or'), N (non-equivalence).

EXPRESSIONS: 24 bits as address, bits 14 to 20 as Ba or Bm, bits 15 to 20 as a character
GENERAL FORM = ELEMENTS combined with SEPARATORS (interpreted from left to right: e.g. A1 + P2 x 3 = (A1 + P2) x 3).

ELEMENT defined as:

*, any parameter, a constant (a:b.c, Ka.b, Ja, Ya), any EXPRESSION enclosed in brackets, an ELEMENT followed by an OPERATOR.

DIRECTIVES:

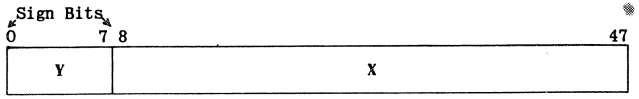
E (enter), ER (read more program), EX (interlude), H (half-word prefix), La (library routine a), Ta (title to output a), F (floating point number), C (store characters), Z (terminate routine), S (6-bit integer prefix), UPa (unset parameter Pa), ½ (output store contents).

FLOATING POINT NUMBERS

Written in Atlas Basic Language [See NOTE 1]	Value Stored in standardized form [See NOTE 2]
±a	±a
±a (⊙ b)	±a x $10^{\pm b}$
±Ka	±Ka
±Ka (:⊙ c)	±Ka x $8^{\pm c}$

NOTES: 1. All signs are obligatory, except when indicated thus ⊙ K precedes octal numbers; others are decimal. 2. To store the same values in substandard form with exponent ±d (or ±Kd) follow the written form with :⊙ d (or :⊙ Kd).

The number z = X x 8^{Y} is stored as follows:



In standardized form the mantissa X of z satisfies either X = 0 (in which case Y = −128 and z = 0), 1/8 ⩽ X < 1, or −1 ⩽ X < −1/8. For any z, standardized or not, Y is an integer in the range −128 ⩽ Y ⩽ 127.

LIBRARY ROUTINES: INPUT AND OUTPUT

Input Routine L100 Entries:-

A1 Read number to Am. QR+.
A2 Read 21-bit integer to B81.
A3 Read character to bits 18 to 23 of B81.
A4 Lose rest of line.
A6 Read text to (b89) onwards.
A7 Read 24-bit integer to B81.
A8 Read 21-bit integer + 3-bit octal fraction to B81.

Output Routine L1 Entries:-

A1 Output *am* (style in B89).
A2 Output b81 (style in B88).
A3 Output character from bits 18 to 23 of B81.
A4 Output NEWLINE.
A5 End record (carriage control character in B87).
A6 Output text from (b89) onwards.

DOCUMENT LAYOUT EXAMPLE

Job/Program Document:-

```
JOB
F163/SURVEY ANALYSIS
INPUT
1 SURVEY RESULTS 1962
OUTPUT
0 ANY 2 BLOCKS
1 LINE PRINTER 3 BLOCKS
COMPUTING 6.5 SECONDS
STORE 25
COMPILER ABL
   |
(Program)
   |
* * * Z
```

Data Document:-

```
DATA
SURVEY RESULTS 1962
   |
 (Data)
   |
* * * Z
```

Alternative Output Peripherals:-
SEVEN-HOLE
FIVE-HOLE
CARDS

NOTATION

Q Accumulator Standardised.
R Accumulator rounded by 'forcing'.
E Exponent overflow may occur.
DO Division overflow occurs if the divisor is zero or unstandardised.
AO Accumulator overflow may occur.
NL L is not cleared initially (as it is in all other basic Accumulator codes).
aq The contents of *Am* following the operation *am'* = *a* QRE.
s: The sum of the contents of store lines *S* and *S*+1.

Magnetic Tape Instructions

1001 Search for section **n** of tape **Ba**.

1002* Read next **K**+1 sections from tape **Ba** to store blocks **P**, **P**+1,..., **P**+**K**.

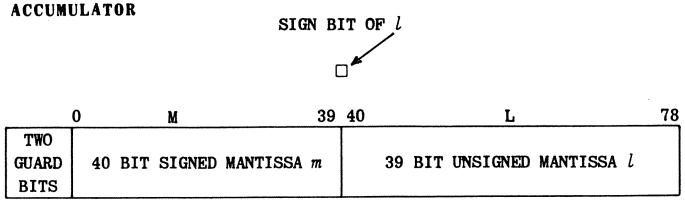
1003* Read previous **K**+1 sections from tape **Ba** to store blocks **P**+**K**,..., **P**+1, **P**.

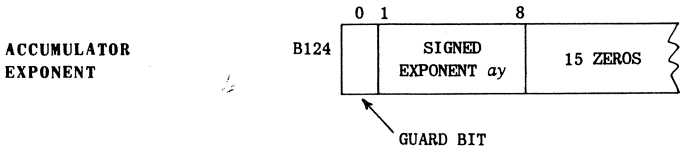
1004* Write store blocks **P**, **P**+1,... **P**+**K** to the next **K**+1 sections of tape **Ba**.

1005* Move tape **Ba** forward **K**+1 sections.
1006* Move tape **Ba** backward **K**+1 sections.

* **n** is written P:0.K

ACCUMULATOR





CONTENTS OF ACCUMULATOR

am = $m \times 8^{ay}$
al = $l \times 8^{ay}$ (includes sign bit of *l*)
a = $ax \times 8^{ay}$

Shifts

Circular

105 $ba' = 8^2ba + s$
125 $ba' = 8^2ba + n$
143 $ba' = \frac{1}{2}ba - s$
163 $ba' = \frac{1}{2}ba - n$
1342 Right *n* binary places
1343 Left *n* binary places

Subroutine Entry

1100 Enter at *s*, $ba' = c + 1$
1101 Enter at *S*, $ba' = c + 1$
1102 Enter at *bm*. $ba' = c + 1$
1362 Enter at n, link in B90

Arithmetic Using 'n'

1520	$am' = am + n$	QRE
1521	$am' = am - n$	QRE
1524	$am' = n$, $l' = 0$	Q
1525	$am' = -n$, $l' = 0$	Q
1534	$am' = n$, $l' = 0$, $ay' = 12$	
1535	$am' = -n$, $l' = 0$, $ay' = 12$	
1562	$am' = am \times n$	QRE
1574	$am' = am/n$	QRE
1575	$am' = aq/n$	QRE

Input and Output

1050 Select input *n*
1051 *ba'* = selected input
1060 Select output *n*
1061 *ba'* = selected output

Basic B-Register Arithmetic

100	$ba' = s - ba$	110	$s' = s - ba$	120	$ba' = n - ba$
101	$ba' = s$	111	$s' = -ba$	121	$ba' = n$
102	$ba' = ba - s$	112	$s' = ba - s$	122	$ba' = ba - n$
103	$ba' = -s$	113	$s' = ba$	123	$ba' = -n$
104	$ba' = ba + s$	114	$s' = ba + s$	124	$ba' = ba + n$

Note: The B-Carry Digit is set by instructions ending 0, 2 or 4.

Logical B-Register Operations

107	$ba' = ba \,\&\, s$	106	$ba' = ba \neq s$
117	$s' = ba \,\&\, s$	116	$s' = ba \neq s$
127	$ba' = ba \,\&\, n$	126	$ba' = ba \neq n$
164*	$ba' = ba + (bm \,\&\, n)$	147	$ba' = ba \vee s$
165*	$ba' = bm \,\&\, n$	167	$ba' = ba \vee n$

* If Bm is BO, $bm \,\&\, n = n$.

Note: 164 sets the B-Carry Digit.

Test Instructions

$ba' = n$ if:-

210	bm is odd (i.e. bit 23 = 1)	224	$bt = 0$
211	bm is even (i.e. bit 23 = 0)	225	$bt \neq 0$
214	$bm = 0$	226	$bt \geqslant 0$
215	$bm \neq 0$	227	$bt < 0$
216	$bm \geqslant 0$	234	$ax = 0$
217	$bm < 0$	235	$ax \neq 0$
		236	$ax \geqslant 0$
		237	$ax < 0$

Test and Count Codes

200	If $bm \neq 0$, $ba' = n$ and $bm' = bm + 0.4$
201	If $bm \neq 0$, $ba' = n$ and $bm' = bm + 1.0$
202	If $bm \neq 0$, $ba' = n$ and $bm' = bm - 0.4$
203	If $bm \neq 0$, $ba' = n$ and $bm' = bm - 1.0$
220	If $bt \neq 0$, $ba' = n$ and $bm' = bm + 0.4$
221	If $bt \neq 0$, $ba' = n$ and $bm' = bm + 1.0$
222	If $bt \neq 0$, $ba' = n$ and $bm' = bm - 0.4$
223	If $bt \neq 0$, $ba' = n$ and $bm' = bm - 1.0$

Accumulator Transfer Instructions

324	$am' = s$	Q
325	$am' = -s$	QE
334	$am' = s$	
335	$am' = -s$	AO
314	$am' = s$	NL
315	$am' = -s$	NL AO
367	$am' = \|s\|$	QE
344	$l' = sx$, $am' = am$	
345	$l' = sx$, $m' =$ sign of s $ay' = ay$	
346	$s' = am$, $a' = 0$	
356	$s' = am$, $a' = a$	
347	$s' = al$, $l' = 0$	
357	$s' = al$, $a' = a$	

Other Accumulator Instructions

364	$ax' = 8ax$, $ay' = ay$	
365	$ax' = ax/8$, $ay' = ay$	
310	$a' = a + s$ if $ay \leqslant sy$ (otherwise $am' = am + s$, l spoiled)	QE NL
311	$a' = a - s$ if $ay \leqslant sy$ (otherwise $am' = am - s$, l spoiled)	QE NL
352	$a' = am \times s$, sign l' = sign m'	E AO
353	$a' = -am \times s$, sign l' = sign m'	E AO
376	$al' = a/\|s\|$ if $a \geqslant 0$ $ay' =$ exponent of quotient, $m' =$ "remainder"	E DO
377	$al' = \|am\|/\|s\|$ $ay' =$ exponent of quotient, $m' =$ "remainder"	E DO
361	$am' = a$	RE NL
354	$am' = a$ rounded by adding	AO NL
355	$a' = al \times 8^{-13}$	Q
366	$am' = \|am\|$	QE

B-Register Extracodes

1300	$ba' =$ int. pt. of s $am' =$ frac. pt. of s	
1301	$ba' =$ int. pt. of am $am' =$ frac. pt. of am	
1302	$ba' = ba \times n$	21-bit integers
1303	$ba' = -ba \times n$	
1304	$ba' = ba/n$ $b97' =$ remainder	
1312	$ba' = ba \times n$	24-bit integers
1313	$ba' = -ba \times n$	
1314	$ba' = ba/n$ $b97' =$ remainder	

Shifts

1340	$ba' = ba \times 2^{-n}$ arithmetic
1341	$ba' = ba \times 2^{n}$ arithmetic
1342	$ba' = ba \times 2^{-n}$ circular
1343	$ba' = ba \times 2^{n}$ circular
1344	$ba' = ba \times 2^{-n}$ logical
1345	$ba' = ba \times 2^{n}$ logical

Trigonometric Functions

1730	$am' = \sin s$	
1731	$am' = \sin aq$	
1732	$am' = \cos s$	
1733	$am' = \cos aq$	
1734	$am' = \tan s$	
1735	$am' = \tan aq$	
1720	$am' =$ arcsin s $(-\pi/2 \leqslant am' \leqslant \pi/2)$	
1721	$am' =$ arcsin aq $(-\pi/2 \leqslant am' \leqslant \pi/2)$	
1722	$am' =$ arccos s $(0 \leqslant am' \leqslant \pi)$	
1723	$am' =$ arccos aq $(0 \leqslant am' \leqslant \pi)$	
1724	$am' =$ arctan s $(-\pi/2 < am' < \pi/2)$	
1725	$am' =$ arctan aq $(-\pi/2 < am' < \pi/2)$	
1726	$am' =$ arctan (aq/s) $(-\pi < am' \leqslant \pi)$	

Arithmetic Extracode Functions

1700	$am' =$ log s	
1701	$am' =$ log aq	
1702	$am' =$ exp s	
1703	$am' =$ exp aq	
1704	$a' =$ integral part of s	
1705	$a' =$ integral part of a	
1706	$a' =$ sign of s	
1707	$a' =$ sign of a	
1710	$am' = \sqrt{s}$	
1711	$am' = \sqrt{aq}$	
1712	$am' = \sqrt{aq^2 + s^2}$	
1760	$am' = am^2$	
1713	$am' = aq^s$ $(aq \geqslant 0)$	
1714	$am' = 1/s$	DO
1715	$am' = 1/am$	DO
1757	$am' = s/am$	DO
1774	$am' = am/s$	QRE
1775	$am' = aq/s$	
1776	$s' =$ quotient $am' =$ remainder if used immediately after 1574, 1575, 1774, 1775.	Q Q
1756	$s' = am$, $am' = s$	
1766	$am' = \|s\|$	AO
1767	$am' = \|am\|$	AO

Double Length Arithmetic

1500	$a' = a + s$:
1501	$a' = a - s$:
1502	$a' = -a + s$:
1504	$a' = s$:
1505	$a' = -s$:
1542	$a' = a \times s$:
1543	$a' = -a \times s$:
1556	s: $' = a$
1565	$a' = -a$
1566	$a' = \|a\|$
1567	$a' = \|s$: $\|$
1576	$a' = a/s$:

Standardised, Rounded Arithmetic

320	$am' = am + s$	QRE
321	$am' = am - s$	QRE
322	$am' = -am + s$	QRE
360	$am' = a$	NL, QRE
362	$am' = am \times s$	QRE
363	$am' = -am \times s$	QRE
374	$am' = am/s$	QRE DO

Standardised, Unrounded Arithmetic

300	$a' = am + s$	QE
301	$a' = am - s$	QE
302	$a' = -am + s$	QE
340	$a' = a$	NL, QE
342	$a' = am \times s$	QE
343	$a' = -am \times s$	QE

Unstandardised Arithmetic

330	$a' = am + s$	AO
331	$a' = am - s$	AO
332	$a' = -am + s$	AO
341	$a' = a$	NL, E
372	$a' = am \times s$	EAO
373	$a' = -am \times s$	EAO
375	$al' = a/\|s\|$ $m' =$ remainder	E

Set B-test

150	$bt' = s - ba$
152	$bt' = ba - s$
170	$bt' = n - ba$
172	$bt' = ba - n$

Note: These 4 instructions set the B-Carry Digit

I·C·T

ATLAS COMPUTER
SUMMARIZED PROGRAMMING INFORMATION

INTERNATIONAL COMPUTERS AND TABULATORS LTD.
I.C.T. HOUSE, PUTNEY, LONDON. S.W.15

Telephone: PUTney 7272

List CS 384

Computer Systems

G. Bell, D. Siewiorek, and S. H. Fuller, Editors

The Manchester Mark I and Atlas: A Historical Perspective

S. H. Lavington
University of Manchester

In 30 years of computer design at Manchester University two systems stand out: the Mark I (developed over the period 1946–49) and the Atlas (1956–62). This paper places each computer in its historical context and then describes the architecture and system software in present-day terminology. Several design concepts such as address-generation and store management have evolved in the progression from Mark I to Atlas. The wider impact of Manchester innovations in these and other areas is discussed, and the contemporary performance of the Mark I and Atlas is evaluated.

Key Words and Phrases: architecture, index registers, paging, virtual storage, extracodes, compilers, operating systems, Ferranti, Manchester Mark I, Atlas, ICL

CR Categories: 1.2, 4.22, 4.32, 6.21, 6.30

1. Introduction and Overview

In the period 1946–76 five computer systems have been designed and implemented at Manchester University. A general account of the prototypes and their industrial derivatives has been given elsewhere [6], along with a comprehensive list of some 60 references to their hardware and software. The main purpose here is to highlight two of the more significant of these five designs. The latest computer in the Manchester series, MU5, is described fully in a companion article [4].

As far as active University research is concerned, Manchester's involvement with digital computers dates from December 1946 when F. C. Williams and Tom Kilburn joined the University from their wartime posts at the Telecommunications Research Establishment. The computer projects, first in the Department of Electrical Engineering and then since 1964 in the Department of Computer Science, have followed the pattern summarized in Table I. This table relates primarily to hardware development; associated system software activity has naturally spanned similar periods, beginning in a small way in 1949 but gathering momentum with the release of the first compiler (1952).

The five prototype computers in the table are the Mark I, the Meg, an experimental transistor computer, the Muse (later Atlas), and the MU5. The names of the five industrially produced derivatives are respectively the Ferranti Mark I, Ferranti Mercury, Metropolitan-Vickers MV950, Ferranti Atlas, and the ICL 2980. The ICL 2980 is not in fact a direct derivative but its architecture owes much to, and has a great deal in common with, MU5. As may be inferred from the table, the cooperation between industry and university has been a fruitful and continuous process since the autumn of 1948. The only one of the five Manchester projects to receive direct government funding was the MU5, which in addition had significant help from ICL in the form of production facilities and engineering support.

The Mark I and Atlas have been chosen for closer study not only because they contain significant innovations, but because they convey an evolutionary progression with respect to the following design themes: i) Instruction format, ii) operand address-generation, iii) store management, and iv) sympathy with high-level language usage. The evolution is continued in MU5 [4]. Whilst all three machines were conceived as general-purpose computers, the internal architecture has tended to favor high-speed scientific applications.

Of the two Manchester computers omitted from detailed analysis in this paper, the Meg (precursor of the Ferranti Mercury) has been passed over because it was essentially an updating of the Mark I concept. By changing the technology and providing parallel access to the main store, the Meg became faster, more compact and easier to maintain. Apart from the incorporation of hardware floating point arithmetic, the instruction format and repertoire were similar to that of the Mark I. The market area of the Ferranti Mercury was much the same as that of the IBM 704, though the 704 was faster and considerably more expensive. The other Manchester computer to be omitted, the experimental point-contact transistor machine, was designed as a small and economic system using a drum as the main store. To help avoid the consequent latency problems a pseudo two-address (or 1 + 1) instruction format was used, in which the address of the next instruction was contained within each instruction. The transistor computer was in this respect untypical of the

Author's Address: Department of Computer Science, University of Manchester, Manchester, M13 9PL, United Kingdom.

Table I. Summary of Manchester University computer projects and their industrially produced derivatives.

University Project	Industrial Derivative
Manchester Mark I hardware development period: 1946–49 prototype operational: June 1948 enhancements: April and Oct. 1949	Ferranti Mark I first installation: Feb. 1951 last one delivered: 1957
Meg hardware development period: 1951–54 prototype operational: May 1954	Ferranti Mercury first installation: Aug. 1957 last one delivered: 1961
Transistor computer hardware development period: 1952–55 prototype operational: Nov. 1953 enhancement: April 1955	Met-Vickers MV 950 first installation 1956 last one delivered (?) 1958
Atlas (formerly Muse) hardware development period: 1956–62 first installation operational Dec. 1962	Ferranti Atlas first installation: Dec. 1962 last one delivered: 1965
MU5 hardware development period: 1966–74 computer operational Oct. 1974	(ICL 2980) 2900 range officially announced: Oct. 1974

other Manchester designs. The use of a drum for primary storage made the transistor computer slower than the Mark I. Perhaps the most important impact of this machine on the Manchester group was the early experience it provided in transistor circuit techniques. The Meg and the transistor computer are described more fully in [6].

In the following account of the Mark I and Atlas each system is presented in three parts. First the objectives of the project are given, during which the motivation and evolutionary starting points are outlined. Secondly the principal features are given, in describing which, some of the original terminology has been replaced by its nearest modern equivalent for the sake of readability. It should be stated, however, that any serious further study of the designs should start with the original papers quoted in [6]; a useful selection of these is [1, 2, 5, 8, 9, 10]. Finally, each computer system is assessed according to its immediate and long-term impact.

2. The Mark I

2.1 Objectives of the Project

The initial aim was to build a realistic test environment for a novel digital store. The store was the electrostatic Williams Tube [9], and the prototype Mark I simply consisted of a 32×32 bit Williams Tube store plus elementary computational facilities. Nevertheless, when it successfully ran a 52-minute factoring program on 21 June 1948 it became the first general-purpose stored-program computer to work. Thereafter the machine underwent intensive engineering development so that by April 1949 a realistic computer had resulted. The objectives by 1949 were to provide sufficient memory and computational facilities to solve the number-theory problems that were provided by early Manchester users [6].

The computer design activity in 1949 was mainly concerned with the engineering aspects of Williams Tubes and drum memories, from which work some elementary "one-level store" ideas began to emerge (see below). The team throughout the Mark I period averaged about four people, working in relative independence from other groups in England and America. Being basically an engineering project, innovation and improvement were more or less continuous processes up to about October 1949.

2.2 Principal Features

a) Technology. The Mark I logic was implemented with EF50 (CV1091) and EF55 pentodes and EA50 vacuum tube diodes—these types being readily available owing to their extensive use in military equipment. The production Mark I comprised 4050 thermionic tubes and consumed about 25KW of power. The digit period was 8.5 microseconds (extended to 10 microseconds in the Ferranti production version). The Williams store was at first based on a standard CV1131 cathode ray tube, but specially-manufactured CRTs were used later.

Williams Tubes were used not only for the main memory but also for the accumulator and other central registers because this was cheaper than providing flip-flop registers. When compared with the mercury delay line which was the other common form of digital store in the late 1940's, the Williams Tube had the following advantages: i) It was random access (not serial access), and ii) it was cheaper to build and required no special temperature control. Williams Tubes did, however, require electrostatic shielding.

The Mark I backing store was a nickel-alloy plated drum, of 30 milliseconds revolution time. The drum was servo-synchronized to the main CPU clock, thus allowing extension to multiple drums without special buffering. Phase modulation recording was used.

b) Architecture. The Manchester Mark I was a serial-ALU, fixed-point, binary computer employing a single-address instruction format. The original word length was 32 bits, but this was increased to 40 bits in 1949 for the sake of greater computational accuracy. A double-length (80-bit) accumulator facility was also provided. Two 20-bit instructions were packed to a word and addressing was to 20-bit boundaries.

The April 1949 version of the Mark I had a repertoire of 26 functions (op codes) in its instruction

set, including hardware multiply. It also had two 20-bit modifier (index) registers called B-lines, 128 words of random access main store and a 1024-word drum backing store. The Ferranti production Mark I was essentially the same architecture but with the following enhancements: i) An instruction set of 50 op codes, ii) eight modifier registers (B-lines), iii) 256 words of main store (Williams Tubes), iv) 4K (extendable to 16k) of drum store, and v) faster multiply time (2.16 milliseconds).

The characteristics of the production Mark I are now described in greater detail, since they form the definitive expression of ideas contained in the series of University prototypes developed during the period 1946–49.

The 20-bit instruction format was as follows:

ADDRESS	B		FUNCTION
10	3	1	6

The three B digits specified one of eight B-lines and the specification of BO was normally used to indicate no modification. There was a separate B-arithmetic unit and associated eight-line B-store for carrying out modifier-register manipulation. Normal operands were 40-bit words, the bits being treated either as a two's complement number or as an unsigned quantity depending on the instruction. The full instruction set is given in Appendix 1, and it may be seen that considerable help was given with multilength arithmetic. There is also a population count or "sideways add" order (denoted in Appendix 1 by the mnemonic SADD), a facility requested by the Manchester mathematicians for their number theory problems. A similar instruction is provided on some modern computers, e.g. the CDC 7600, for nuclear physics applications programming, etc. The Ferranti Mark I also had a hardware random-number generator, available via mnemonic RNDM in Appendix 1. This somewhat unusual facility was included mainly at the request of the mathematician A.M. Turing, who was at Manchester from September 1948 until his death in June 1954.

Transfers to and from the drum and other peripheral equipment were carried out via 20-bit control words. These had two formats, distinguished by one of the mode bits. For *drum* transfers the format was:

DRUM TRACK ADDRESS	MODE		TUBE
11	4	1	4

Three of the mode bits then specified reading/writing, read-checking/write-checking, single page/double page transfers. The main store was arranged as eight 32-word pages on eight Williams Tubes, backed by the drum(s). The track-address was stored along with each page of information on the drum, and when a page became resident in main store an extra 20-bit line was assigned on each Williams Tube to hold the track-address of that page. This page-address line was normally invisible to the programmer, but could be accessed via the special LDAD instruction (Appendix 1). This was the germ of an idea which later led to page-address registers and virtual-to-real address translation on the Atlas computer.

For input/output transfers the control word format was:

	MODE	
10	5	5

For the early Ferranti Mark I's input was via a 250 character/sec 5-track paper tape reader using the 5-bit teleprinter code, and output was to a tape punch and printer. Four mode bits in the control word specified: Output a character, check output buffer, input a character, send a control character (equal to carriage return, linefeed, figure shift, letter shift) to the output device. Two input/output commands were provided (see Appendix 1): one took its control word from a main store address and the other used a 20-digit pattern set on the console switches—useful during bootstrapping.

c) System Software. In 1949 there was no Mark I system software, except for basic utilities such as input routines. Coding was normally carried out using the symbols of the 5-track teleprinter code. Once the Ferranti Mark I had been installed software development increased, with the emphasis being on embryonic high-level languages. After one earlier effort at compiler writing (1952), the Mark I Autocode [1] was available from March 1954 as an easy-to-learn scientific programming language for users having small or medium-sized problems.

An additional Autocode implementation objective was to simulate a one-level store so that the user had no need to organize his own drum transfers. It was possible to simulate this one-level store on the Mark I in a reasonably balanced way because the access time for reading an operand from the drum happened to be about the same time as a floating-point addition via an interpretive library routine. When running Autocode programs 128 tracks on the drum were reserved for instructions and 128 tracks for variables. Individual routines were transferred to the fast CRT store as they were required. To gain access to a variable an interpretive routine in fast store first determined on which track it lay, then transferred that track or "page" to the fast store, and finally selected the particular line within the page. Since successive operands were quite often located on the same page, steps were taken to avoid unnecessary drum transfers.

Arithmetic in the Autocode system was normally performed on floating-point variables $v1$, $v2$, . . . , etc. with provision for integers $n1$, $n2$, . . . , to be used as indices and counters. Simple conventions also existed for control transfers, intrinsic functions, input/output, and simple job control using symbols from the five-bit teleprinter code. An impression of the neatness of the system may be gained from the following Mark

I Autocode sequence which prints the root mean square (rms) of the variables $v1, v2, \ldots v100$. (Note that the symbol * causes printing of a variable to ten decimal places on a new line, and $F1$ signifies the intrinsic function 'square root'):

```
n1 = 1
v101 = 0
2v102 = vn1 × vn1
v101 = v101 + v102
n1 = n1 + 1
j2, 100 ≥ n1
v101 = v101/100.0
*v101 = F1(v101)
```

2.3 Evaluation

The Mark I, in common with most early computers, was first applied to scientific and engineering problems. Measurements performed on a sample of Mark I jobs estimated that 16% of computing time went on drum transfers, 28% on multiplication and 56% on other arithmetic operations. Multiplication took 2.16 milliseconds and other accumulator orders took 1.2 milliseconds.

A contemporary benchmarking exercise rated the Mark I at about the same raw power as the National Physical Laboratories' ACE computer, even though the ACE had a digit period ten times shorter. The favorable performance of the Mark I was attributed to its random access main memory (ACE had a delay line store) and its relatively fast multiplier. Ferranti delivered nine Mark I and Mark I star machines between 1951 and 1957, three of them being exported (to Canada, Holland, Italy).

The long-term significance of the Manchester Mark I project is threefold. Firstly, it proved the viability of a digital storage technique (the Williams Tube), at a time when the successful implementation of the stored-program concept awaited the development of a suitable storage device. Williams Tubes were adopted by several computers in England, Russia, and America—including the IBM 701. Secondly, the project inspired the British government to give financial support to Ferranti Ltd., thus laying one of the cornerstones of the British computer industry. Thirdly, and of perhaps wider significance, the Mark I project was the first to focus attention onto the problems of linking fast random-access main memory to slower sequential-access rotating memory.

It was in the light of these problems that B-lines were first conceived of as relocation registers. It soon became clear that B-lines could also be used for general address-modification purposes, and so with the inclusion of a B-test facility the modern index register was born. The problem of automating backing store transfers ("overlays") still remained a challenge, but two Mark I facilities were later to suggest a solution to the Manchester team. First there was the fact that every page resident in the Mark I fast store carried with it the corresponding drum address in a special "page-address line" (see above). Second, there was the way in which the Autocode system handled the *drum* address of a user's variables. Out of these two facilities grew the concept of allowing the user always to program in a virtual (or "drum") address space and then providing system hardware and software to achieve automatic translation into the real (or "fast") address space, using information held in a set of associatively interrogated page-address registers. Thus the automated "one-level store" was conceived, and the realization of the other programming advantages to be gained from separating virtual and real address spaces followed shortly. These ideas were implemented in the Atlas computer.

3. Atlas

3.1 Objectives of the Project

By 1956 it was clear that Britain was falling behind the United States in the production of high-performance computers. The MUSE ("micro-second") project, started by Kilburn at Manchester in the autumn of 1956, was a conscious effort to remedy the situation. From January 1959 Ferranti Ltd. officially became involved and a joint University/Ferranti team under Kilburn continued the development of the computer, which was now known as Atlas.

Initial discussions with potential users of high-performance machines, both scientific and commercial, had produced a requirement for instruction times approaching one microsecond, the ability to attach a large number of i/o devices of various types and a main store size approaching 100k words. High computing speeds and rapid turnaround of user jobs became the keynotes of the Atlas design. The principal difficulties in achieving these goals arose from the wide differences in operating speeds between the various types of peripheral equipment and the CPU, and between transistor logic circuits and available core stores. Efficient and economic utilization of equipment was also very much a design-objective, since Atlas was intended to be sold on the open market. The somewhat conflicting requirements for high-speed and relative economy led to the incorporation of many techniques which were not extant when the project started in 1956. Amongst these were multiprogramming, job scheduling, spooling, extracodes, interrupts, pipelining, interleaved storage, autonomous transfer units, virtual storage, and paging. Although not all of these ideas originated in Manchester, they combined to make Atlas probably the most powerful machine available in the early 1960's.

3.2 Principal Features

a) Technology. The Atlas logic circuits were based on an OC170 germanium junction transistor used as an inverter, preceded by germanium OA47

diodes for the logic gating. This gave a typical gate-delay of 12 nanoseconds. Care was taken to avoid saturation (low collector-base volts), since the OC170's response became slow in the saturation region. The parallel adder employed a special symmetrical transistor (the SB240) as a switch in the carry-path, which resulted in a basic add-time of 200 nanoseconds for 48 bits—a significant achievement in 1959. There were about 80,000 transistors in the entire computer, mostly mounted on 8-inch by 5-inch printed-circuit boards.

The main store was 2 microsecond cycle-time core four-way interleaved, backed by drums having a 12 millisecond revolution time and capable of transferring one 512-word block every 2 milliseconds. Two other "private" storage units were provided: A high-speed read-only "fixed store" of 0.35 microsecond access time made from small slugs of copper or ferrite inserted in a woven wire mesh; and a system working store of 2 microsecond cycle-time core, which served as working space for the operating system routines (many of which resided in the fixed store). The size of all these stores varied between production versions of the Atlas. The Manchester prototype had the smallest capacity, expressed in 48-bit words as follows:

i) main store: 16k core, backed by four drums each of 24k
ii) fixed store: 8k
iii) system working store: 1k (later increased to 4k)

The largest production Atlas, installed at the Science Research Council's computing laboratory at Chilton (Harwell), had a main core store of 48k.

Bulk storage was provided by eight (expandable to 32) tape decks on eight channels, each having a transfer rate of 90k characters per second. Preaddressing and fixed 512-word blocks were used, thus allowing a tape to be written to nonsequentially when required. A 16-million word file disk was added later.

The Manchester Atlas had 17 conventional i/o devices, two high-speed data links, an on-line x-ray crystallographic diffractometer and an experimental speech input/output unit. The interrupt structure allowed for the connection of up to 512 peripheral units, with hardware assistance for determining the source of an interrupt.

b) Architecture. Atlas was a 48-bit word parallel computer with a one-address instruction format as follows:

FUNCTION	Ba	Bm	ADDRESS
10	7	7	24

The repertoire of functions or op codes, summarized in Appendix 2, was divided into two groups: normal instructions and extracode instructions. Generally speaking an extracode was a commonly used but relatively complicated function which it was not economic to implement directly as hardwired logic. Instead, an extracode consisted of a sequence of normal instructions (a "macro routine") held in the fixed store. Entry to these macro routines was very rapid and involved no preservation of central registers since there was a dedicated extracode program counter (or control register) and reserved B-lines, and any extracodes needing working space used a private area of the system working store. Amongst extracode instructions available to the user were ones for carrying out the common intrinsic functions such as square root, log, cosine, etc.

Of the normal instructions, Appendix 2 shows that they divide into three subgroups: main accumulator (A) orders, index register (B) orders, and test-and-count orders. There were independent A and B arithmetic units. The instruction set and the Atlas pipeline architecture assumed there would normally be no interchange of operands between the A and B ALUs.

This design philosophy, also to be seen to some degree in MU5, works most effectively for computations such as forming the scalar product of two vectors. By careful pipeline design and by using tricks such as assuming that the next instruction usually occurred in the same page as the last instruction, Atlas could overlap the execution of three A-instructions and then any associated B-instructions were normally executed concurrently with minimal additional time penalty.

The Atlas instruction could be double address-modified, according to the specification of the Ba and Bm bits. There were 127 24-bit B-lines (index registers) for this purpose, mostly held in a 0.7 microsecond cycle-time core store. The top three B-lines, B125–B127, were implemented as flip-flop registers and were reserved for use as independent program counters respectively for interrupt, extracode, and main program control. This explains why no explicit jump (branch) instructions appear in Appendix 2.

Of the 24 address bits in an instruction, 20 were used to cover the virtual address space of one million words, three specified a 6-bit character position within a full word, and one bit distinguished between a normal address and a "*V*-store" address. The *V*-store was the collective name given to all central registers and peripheral device registers which needed to be accessible to a (system) program. Into this category came such things as interrupt registers, page-address registers, and the data and status registers of all i/o units. Since normal instructions could, with suitable protection checks, use *V*-store addresses for operands there was no need for explicit op codes for the control of i/o equipment etc. The incorporation of peripheral devices into the total address space has since been used on other computers such as the PDP11.

The Atlas paging system used 512-word fixed-size pages, with a page-address register for every 512-word section of main core store. Each register contained a lock-out digit, so that pages of more than one program could be resident in core concurrently. The address-translation time, i.e. associative interrogation of the

page-address registers, was 0.7 microseconds, which represented about 40 percent of the total operand fetch time (including cable delays, store access time, etc.). Considerable effort was spent in ensuring efficient page-turning, and the replacement algorithm—contained in the fixed store—used a learning program which attempted to identify pages in main store which had fallen out of use [5]. No copy was normally kept on drum, and each drum had a rotational position indicator to speed the transfer of a replaced page to the first available space on drum. (Many modern paging computers, including MU5, now keep copies on drum and arrange not to write back pages which are unaltered.) The Atlas virtual address space was sufficiently large for the compilers to arrange simple segmentation conventions during the compilation process.

c) System Software. The Atlas Supervisor (operating system) fully exploited the following concepts: i) Multiprogramming (of up to 16 jobs concurrently), ii) on-line spooling of input and output, and iii) job scheduling (according to user-indicated job characteristics such as use of magnetic tapes, volume of output, or priority request). The aim was to keep all of the computer equipment busy while minimizing the turnaround of individual jobs. Since in addition the Supervisor produced comprehensive logging statistics for the user, the operator, and the (daily) accounting program, the computer room operators had little to do except feed in work—an exceptional state of affairs in the 1960's. The programmer was provided with straightforward job control conventions which were simple for simple tasks, while allowing scope for more complex requirements such as multiple input or output streams, or the use of special "private" compilers.

The standard Atlas compilers were mostly implemented using the Compiler Compiler [2], a formal language for the transparent description of syntax and semantics. While compilers for all the common high-level languages such as Algol, Cobol, and Fortran were eventually written for Atlas, the Manchester users programmed extensively in Atlas Autocode to begin with. Atlas Autocode was a block-structured language specified at about the same time as Algol 60, but implemented sooner. Except for a more limited concept of compound statements and *for* clauses, Atlas Autocode was very similar to Algol 60. As an example the root mean square calculation given in Section 2.2 might be programmed in Atlas Autocode as follows:

```
RMS = 0
cycle i = 1, 1, 100
RMS = RMS + X(i) * X(i); repeat
print (sqrt (RMS/100), 6, 4)
```

3.3 Evaluation

Some typical Atlas instruction times were as follows:

fixed-point B addition	1.59 microsec.
floating-point add, no modification	1.61 microsec.
floating-point add, double modification	2.61 microsec.
floating-point multiply, double modification	4.97 microsec.
floating-point division	10.66 to 29.80 microsec.

The overall average instruction time when executing Fortran scientific programs was measured to be about 3.35 microseconds per order. For comparison, the corresponding overall average instruction rates for typical present-day computers executing similar programs range from about 110 nanoseconds per order (CDC 7600) to about 6.6 microseconds per order (IBM 370/135).

In terms of contemporary machines Ferranti salesmen equated one Atlas to four IBM 7094s as regards work throughput. It is disappointing that only three full Atlas's and two scaled-down versions ("Atlas II") were sold between 1962–65. To some extent the marketing of Atlas was overtaken by events: Ferranti was currently developing other systems besides Atlas; Ferranti sold its large computer interest to ICT (later ICL) in 1963; ICT afterwards introduced its 1900 range of computers. By the mid-1960s the direct market competitor to Atlas was the faster CDC 6600, whose designers have said they learned some useful lessons from Atlas. In 1967 a benchmarking comparison for 22 compute-bound Fortran scientific jobs was performed on an Atlas with 32k core, a single-processor Univac 1108 with 64k core, and a CDC 6600 with 64k core [3]. The compilers used were respectively the London nonoptimizing Atlas Fortran V, the Univac F4012 Fortran IV, and the CDC Chippewa Run. Under these conditions the average CP computing speeds for Atlas, Univac 1108, and CDC 6600 were measured to be in the ratio 1: 2.1: 5.9 respectively. By the start of the 1970s ICL had introduced the 1906A computer, which has a work throughput of the order of three times that of Atlas, depending upon the configuration.

The long-term significance of the Atlas project lies in the design-concepts which it introduced. The more important of these are in four general areas: Pipeline techniques for high instruction throughput, paging and virtual storage, operating system features, and extracodes. The first area is now well-defined in respect of single instruction streams. The second has had far-reaching consequences. It has been the subject of much subsequent analysis and development (e.g. for MU5), in the light of which it is interesting to observe that the inspired guess of 512-word pages for Atlas proved to be about right. Programs on Atlas generally produced about one page-exception (page-address register nonequivalence) per 5×10^4 accesses. With regard to operating system features, the Atlas Supervisor would seem to the present-day user to have a very limited concept of file manipulation and no time-sharing (interactive) facilities. However, in all other

respects, especially in job throughput and ease of use, the Atlas Supervisor set a high standard which is still relevant today. Atlas was also one of first computers in which specific hardware facilities were provided at the design stage to aid the operating system—e.g. in the area of peripheral control, interrupt handling, and store management. With regard to extracodes, the concept of providing easy access to commonly required software has been taken up by several other designers. On Atlas the existence of a specially constructed fixed store for extracodes and certain Supervisor routines was partly determined by nonavailability of suitable fast core. It was generally observed that over half of all Atlas user-programs spent more than half their run time in executing common software such as the extracodes and i/o routines, so there was ample justification for having speeded up the access to much of this standard software.

As for the influence of Atlas on the design of its successor at Manchester, MU5, an important lesson was learned concerning the B-lines. The Mark I had had eight such lines and on Atlas about 90 of the 127 B-lines were available to the general user. This was indeed a lavish provision for the *assembler* programmer, but it was observed that the compiled code of the Atlas high-level language programmer could not easily make use of more than a few of these. A compiler writer has a potential need for registers such as B-lines for two main object-code purposes:

i) as bases and pointers for address-generation;
ii) as fast storage for frequently used operands when attempting run-time optimization—(this applies not only to integers for loop-control etc., but also to frequently-used floating-point variables).

For the former application the registers should be capable of reflecting the usage of local and nonlocal name spaces in block-structured languages—though there is no such special requirement for languages such as Fortran. For the latter application it would be advantageous if identification of frequently used operands was automatic at run time, thus saving on compiler complexity. The MU5 design attempts to satisfy these high-level language requirements with specific naming registers and associatively accessed buffers, and from Manchester's viewpoint Atlas marked the end of the road for the general-purpose B-line. The vindication of the MU5 approach lies in the more efficient compiled code which it produces [7].

4. Conclusion

This paper and its companion [4] reveal a developing view of computer design over a period of 30 years. At the beginning of this period the task of inventing the basic functional units and then keeping them running for long enough to obtain useful computation, dictated a spartan engineering approach to machine architecture. As technology advanced and successive Manchester computers were implemented and evaluated, so the designers were able to observe and incorporate in hardware more of the requirements of the system software and the users. These requirements themselves evolved over the years. Although the emphasis of the Mark I, Atlas, and MU5 has been on large high-performance systems, it is evident that the designs have not only kept abreast of the requirements of the general user, but in some cases (e.g. paging and virtual storage) the architectural innovations have been in advance of the facilities expected by normal programmers. In time, with the decreasing cost of logic and main storage, many of the Manchester "high-performance" devices have come to be adopted in succeeding middle-range computers.

Appendix 1

The Instruction Set of the Ferranti Mark I

In order to relate to modern terminology the following notation is used when describing the action of each order:

ACC: the contents of the double-length main accumulator (80 bits)
AM: the most-significant 40 bits of ACC
AL: the least-significant 40 bits of ACC
S: the contents of a store line (40 bits), except that B orders use the least significant 20 bits and control-transfer orders the least significant 10 bits.
B: the contents of a B-line (index register)
D: the contents of the multiplicand register (40 bits)
H: the digits set up on 20 console handswitches.

a) Main Arithmetic and Logical Orders

Mnemonic	Description
LDA	load AL (AM cleared)
LDAS	load AL, sign-extend into AM
LDN	load AL negatively
STA	store AL
STM	store AM
STMC	store AM and clear AM
SWAP	interchange AM and AL
STAM	store AL, move AM to AL and clear AM
STAC	store AL and clear ACC
CLR	clear ACC
ADD	ACC := ACC + S (signed S)
ADDU	ACC := ACC + S (unsigned S)
SUB	ACC := ACC − S (signed S)
ADDM	AM := AM + S
LDDU	load D (unsigned multiplicand)
LDDS	load D (signed multiplicand)
MADU	ACC := ACC + D × S (unsigned S)

MADS	ACC := ACC + D × S (signed S)
MSBU	ACC := ACC − D × S (unsigned S)
MSBS	ACC := ACC − D × S (signed S)
AND	ACC := ACC & S (S sign-extended)
ORA	ACC := ACC or S (S sign-extended)
NEQ	ACC := ACC ≠ S (S sign-extended)
SHLS	ACC := 2 × S (arithmetic shift)
ORS	S := AL := AL or S
ORSC	S := AL or S, then clear ACC

b) B-line (Index-Register) Manipulation

Mnemonic	Description
LDB	load a specified B-line
STB	store a specified B-line
SUBB	B := B − S
LDBX	load a B-line (without modification)
STBX	store a B-line (without modification)
SBBX	B := B−S (without modification)

c) Control Transfer Orders

Mnemonic	Description
JMPA	absolute indirect unconditional jump
JMPR	relative indirect unconditional jump
JGEA	if ACC ≥ 0, absolute indirect jump
JGER	if ACC ≥ 0, relative indirect jump
JGBA	if (last-named B-line) ≥ 0, absolute indirect jump
JGBR	if (last-named B-line) ≥ 0, relative indirect jump

d) Peripheral and Miscellaneous Orders

Mnemonic	Description
IOTH	i/o transfer using H as a control word
IOTS	i/o transfer using S as a control word
NORM	add to AM the position of the most significant one in S
SADD	add to AM the number of 1's in S − (population count)
RNDM	load a random number into AL
LDAD	load a page-address word into AL
DST1	debugging stop (1)
DST2	debugging stop (2)
TIME	S := clock
HOOT	pulse the console hooter
STH	S := console handswitches H
NULL	no operation

Appendix 2

Abbreviated Summary of the Atlas Instruction Set

In describing the action of the orders the following notation is used:

AM: the contents of the main 48-bit accumulator. (For floating-point working a 40-bit mantissa, 8-bit exponent and an octal base is used. For fixed-point working only 40 bits are used.)

AL: for double-length working AL forms a 39-bit mantissa extension.

S: the contents of a store line (normally 48 bits)

BA, BM: the contents of 24-bit B-lines (as addressed by the Ba and Bm fields).

BT: the contents of a B-test register.

N: a 24-bit literal ("immediate operand"), specified by taking the value of the instructions' address field as a two's complement number.

a) Main Accumulator Arithmetic Orders

(Note that several arithmetic orders were repeated with minor differences concerning accumulator standardization, rounding, clearing of AL, etc. Such orders are asterisked).

Mnemonic	Description
LDA	load AM (* four versions)
LDN	load AM negatively (* three versions)
LDL	load AL (* two versions)
LDDL	load AM and AL double-length
LDDLN	load double-length negatively
STA	store AM (* two versions)
STL	store AL (* two versions)
STDL	store AM and AL double length
ADD	fixed-point add
ADFL	single-length floating point add (* two versions)
ADFD	double-length floating point add
SUB	fixed-point subtract
SBFL	single-length floating point subtract (* two versions)
SBFD	double-length floating point subtract
RSUB	fixed-point reverse subtract
RSBFL	single-length floating point reverse subtract (* two versions)
RSBFD	double-length floating point reverse subtract
MPY	fixed-point multiply
MPFL	single-length floating point multiply (* two versions)
MPFD	double-length floating point multiply
NMPY	fixed-point multiply and negate
NMPFL	single-length floating point multiply and negate (* two versions)
NMPFD	double-length floating point multiply and negate
DIV	fixed-point divide
DVFL	single-length floating point divide
DVDL	double-length floating point divide

There were an additional 17 orders for performing miscellaneous minor operations on the accumulator such as negating, taking the modulus, etc.

b) B-line ("Index Register") Manipulations

(Note that orders asterisked used the 'read-pause-write' (split cycle) technique.)

Mnemonic	Description
LDB	load a specified BA (BA′ = S)
LDBN	load negatively a specified BA
LND	load literal (BA := N)
LNN	load negatively a literal (BA := − N)
STB	store a specified BA (S := BA)
STBN	store negatively a specified BA
ADB	add (BA := BA + S)
ADN	add literal (BA := BA + N)
SBB	subtract (BA := BA − S)
RSBB	reverse subtract (BA := S − BA)
SADB	add into store (S := S + BA) *
SSBB	subtract from store (S := S − BA) *
RSSBB	reverse subtract from store (S := BA − S) *
SBN	subtract literal (BA := BA − N)
RSBN	reverse subtract literal (BA := N − BA)
AND	BA := BA & S
ANDS	S := BA & S *
ANDN	BA := BA & N
ANMN	BA := BM & N
ADMN	BA := BA + (BM & N)
NEQ	BA := BA ≠ S
NEQS	S := BA ≠ S *
NEQN	BA := BA ≠ N
ORB	BA := BA or S
ORBN	BA := BA or N

There were a further four miscellaneous simple B orders. More complex B operations, including multiplication, were peformed by extracodes—see Section d below.

c) Test and Count Orders.

i) Six orders of the form: BA := N *IF* BM is: odd, even, = 0 ≠ 0 ≥0, <0.
ii) Four orders of the form: BA := N *IF* BT is: = 0, ≠ 0, ≥0, <0.
iii) Four orders of the form: BA := N *IF* (AM, AL) is: = 0, ≠ 0, ≥0, <0.
iv) Four orders which set BT according to the result of: (S − BA), (BA − S), (N − BA), (BA − N).
v) Four orders of the form: *IF* BM ≠ 0, BA := N *AND*: (BM := BM + ½), (BM := BM + 1), (BM := BM − ½), (BM := BM − 1).
vi) Four orders of the form: *IF* BT ≠ 0, BA = N *AND*: (BM := BM + ½), (BM := BM + 1), (BM := BM − ½), (BM := BM − 1).

Note that BA was thought of as the "arithmetic" B-line and BM as the "address-modification" B-line. Adding ½ to BM allowed halfword boundaries to be accessed.

d) Extracodes. These caused automatic entry to and return from fixed-store routines. Extracodes intended for general use divided into the following groups:

i) 14 extracodes for operating on B-lines, providing multiplication, division, shifting, etc.
ii) 46 extracodes giving additional (AL, AM) facilities such as:
arithmetic using literals
evaluation of standard trigonometric functions
evaluation of other standard functions such as log, exp, sqrt, reciprocal, etc.
iii) Three extracodes for subroutine entry (return link stored in BA).
iv) 20 user-orientated extracodes for the control of magnetic tape, i/o stream selection, etc.

The rest of the fixed store was filled with system software such as the drum learning program, i/o device routines, and standard test programs.

Acknowledgments. The author would like to thank Professor Tom Kilburn and many of the staff in the Department of Computer Science, University of Manchester, for the helpful discussions concerning systems described in this paper.

Received March 1977; revised July 1977

References

1. Brooker, R.A. An attempt to simplify coding for the Manchester electronic computer. *Brit. J. Appl. Physics 6* (1955), 307-311.
2. Brooker, R.A., MacCallum, I.R., Morris, D., and Rohl, J.S. The compiler compiler. *Ann. Rev. in Automatic Programming, 3* (1963), 229ff.
3. Hughes, P.H. University computer benchmark report. Atlas Computing Service, U. of London, July 1967.
4. Ibbett, R.N., and Capon, P.C. The development of the MU5 computer system. *Comm. ACM 21*, 1 (Jan. 1978), 14-25.
5. Kilburn, T., Edwards, D.B.G., Lanigan, M.J., and Sumner, F.H. One-level storage system. *IRE Trans. EC-11*, 2 (1962), 223-235 (Reprinted in Bell, C.G., and Newell, A. *Computer Structurers: Readings and Examples*. McGraw-Hill, New York, 1971).
6. Lavington, S.H. *A History of Manchester Computers*. Nat. Comptng. Ctr. Publications, Manchester, England, 1975 (also published in the U.S. by Hayden, Rochelle Pk, N.J.)
7. Lavington, S.H., and Knowles, A.E. Assessing the power of an order code. Proc. IFIP Congress 77, Toronto, Canada, 1977, pp.
8. Morris, D., Sumner, F.H., and Wyld, M.T. An appraisal of the Atlas Supervisor. Proc ACM Nat. Meeting, 1967, pp. 67-75.
9. Williams, F.C., and Kilburn, T. A storage system for use with binary digital computing machines. *Proc. IEE*, Vol. 96, Pt. 2, No. 30, 1949, p. 183ff.
10. Williams, F.C., Kilburn, T., and Tootill, G.C. Universal high-speed digital computers: A small-scale experimental machine. Proc. IEE, Vol. 98, Pt. 2, No. 61, 1951, pp. 13-28.

Computer Systems

G. Bell, D. Siewiorek, and S. H. Fuller, Editors

The Manchester Mark I and Atlas: A Historical Perspective

S. H. Lavington
University of Manchester

In 30 years of computer design at Manchester University two systems stand out: the Mark I (developed over the period 1946–49) and the Atlas (1956–62). This paper places each computer in its historical context and then describes the architecture and system software in present-day terminology. Several design concepts such as address-generation and store management have evolved in the progression from Mark I to Atlas. The wider impact of Manchester innovations in these and other areas is discussed, and the contemporary performance of the Mark I and Atlas is evaluated.

Key Words and Phrases: architecture, index registers, paging, virtual storage, extracodes, compilers, operating systems, Ferranti, Manchester Mark I, Atlas, ICL

CR Categories: 1.2, 4.22, 4.32, 6.21, 6.30

1. Introduction and Overview

In the period 1946–76 five computer systems have been designed and implemented at Manchester University. A general account of the prototypes and their industrial derivatives has been given elsewhere [6], along with a comprehensive list of some 60 references to their hardware and software. The main purpose here is to highlight two of the more significant of these five designs. The latest computer in the Manchester series, MU5, is described fully in a companion article [4].

As far as active University research is concerned, Manchester's involvement with digital computers dates from December 1946 when F. C. Williams and Tom Kilburn joined the University from their wartime posts at the Telecommunications Research Establishment. The computer projects, first in the Department of Electrical Engineering and then since 1964 in the Department of Computer Science, have followed the pattern summarized in Table I. This table relates primarily to hardware development; associated system software activity has naturally spanned similar periods, beginning in a small way in 1949 but gathering momentum with the release of the first compiler (1952).

The five prototype computers in the table are the Mark I, the Meg, an experimental transistor computer, the Muse (later Atlas), and the MU5. The names of the five industrially produced derivatives are respectively the Ferranti Mark I, Ferranti Mercury, Metropolitan-Vickers MV950, Ferranti Atlas, and the ICL 2980. The ICL 2980 is not in fact a direct derivative but its architecture owes much to, and has a great deal in common with, MU5. As may be inferred from the table, the cooperation between industry and university has been a fruitful and continuous process since the autumn of 1948. The only one of the five Manchester projects to receive direct government funding was the MU5, which in addition had significant help from ICL in the form of production facilities and engineering support.

The Mark I and Atlas have been chosen for closer study not only because they contain significant innovations, but because they convey an evolutionary progression with respect to the following design themes: i) Instruction format, ii) operand address-generation, iii) store management, and iv) sympathy with high-level language usage. The evolution is continued in MU5 [4]. Whilst all three machines were conceived as general-purpose computers, the internal architecture has tended to favor high-speed scientific applications.

Of the two Manchester computers omitted from detailed analysis in this paper, the Meg (precursor of the Ferranti Mercury) has been passed over because it was essentially an updating of the Mark I concept. By changing the technology and providing parallel access to the main store, the Meg became faster, more compact and easier to maintain. Apart from the incorporation of hardware floating point arithmetic, the instruction format and repertoire were similar to that of the Mark I. The market area of the Ferranti Mercury was much the same as that of the IBM 704, though the 704 was faster and considerably more expensive. The other Manchester computer to be omitted, the experimental point-contact transistor machine, was designed as a small and economic system using a drum as the main store. To help avoid the consequent latency problems a pseudo two-address (or 1 + 1) instruction format was used, in which the address of the next instruction was contained within each instruction. The transistor computer was in this respect untypical of the

Author's Address: Department of Computer Science, University of Manchester, Manchester, M13 9PL, United Kingdom.

Table I. Summary of Manchester University computer projects and their industrially produced derivatives.

University Project	Industrial Derivative
Manchester Mark I hardware development period: 1946–49 prototype operational: June 1948 enhancements: April and Oct. 1949	Ferranti Mark I first installation: Feb. 1951 last one delivered: 1957
Meg hardware development period: 1951–54 prototype operational: May 1954	Ferranti Mercury first installation: Aug. 1957 last one delivered: 1961
Transistor computer hardware development period: 1952–55 prototype operational: Nov. 1953 enhancement: April 1955	Met-Vickers MV 950 first installation 1956 last one delivered (?) 1958
Atlas (formerly Muse) hardware development period: 1956–62 first installation operational Dec. 1962	Ferranti Atlas first installation: Dec. 1962 last one delivered: 1965
MU5 hardware development period: 1966–74 computer operational Oct. 1974	(ICL 2980) 2900 range officially announced: Oct. 1974

other Manchester designs. The use of a drum for primary storage made the transistor computer slower than the Mark I. Perhaps the most important impact of this machine on the Manchester group was the early experience it provided in transistor circuit techniques. The Meg and the transistor computer are described more fully in [6].

In the following account of the Mark I and Atlas each system is presented in three parts. First the objectives of the project are given, during which the motivation and evolutionary starting points are outlined. Secondly the principal features are given, in describing which, some of the original terminology has been replaced by its nearest modern equivalent for the sake of readability. It should be stated, however, that any serious further study of the designs should start with the original papers quoted in [6]; a useful selection of these is [1, 2, 5, 8, 9, 10]. Finally, each computer system is assessed according to its immediate and long-term impact.

2. The Mark I

2.1 Objectives of the Project

The initial aim was to build a realistic test environment for a novel digital store. The store was the electrostatic Williams Tube [9], and the prototype Mark I simply consisted of a 32×32 bit Williams Tube store plus elementary computational facilities. Nevertheless, when it successfully ran a 52-minute factoring program on 21 June 1948 it became the first general-purpose stored-program computer to work. Thereafter the machine underwent intensive engineering development so that by April 1949 a realistic computer had resulted. The objectives by 1949 were to provide sufficient memory and computational facilities to solve the number-theory problems that were provided by early Manchester users [6].

The computer design activity in 1949 was mainly concerned with the engineering aspects of Williams Tubes and drum memories, from which work some elementary "one-level store" ideas began to emerge (see below). The team throughout the Mark I period averaged about four people, working in relative independence from other groups in England and America. Being basically an engineering project, innovation and improvement were more or less continuous processes up to about October 1949.

2.2 Principal Features

a) Technology. The Mark I logic was implemented with EF50 (CV1091) and EF55 pentodes and EA50 vacuum tube diodes—these types being readily available owing to their extensive use in military equipment. The production Mark I comprised 4050 thermionic tubes and consumed about 25KW of power. The digit period was 8.5 microseconds (extended to 10 microseconds in the Ferranti production version). The Williams store was at first based on a standard CV1131 cathode ray tube, but specially-manufactured CRTs were used later.

Williams Tubes were used not only for the main memory but also for the accumulator and other central registers because this was cheaper than providing flip-flop registers. When compared with the mercury delay line which was the other common form of digital store in the late 1940's, the Williams Tube had the following advantages: i) It was random access (not serial access), and ii) it was cheaper to build and required no special temperature control. Williams Tubes did, however, require electrostatic shielding.

The Mark I backing store was a nickel-alloy plated drum, of 30 milliseconds revolution time. The drum was servo-synchronized to the main CPU clock, thus allowing extension to multiple drums without special buffering. Phase modulation recording was used.

b) Architecture. The Manchester Mark I was a serial-ALU, fixed-point, binary computer employing a single-address instruction format. The original word length was 32 bits, but this was increased to 40 bits in 1949 for the sake of greater computational accuracy. A double-length (80-bit) accumulator facility was also provided. Two 20-bit instructions were packed to a word and addressing was to 20-bit boundaries.

The April 1949 version of the Mark I had a repertoire of 26 functions (op codes) in its instruction

set, including hardware multiply. It also had two 20-bit modifier (index) registers called B-lines, 128 words of random access main store and a 1024-word drum backing store. The Ferranti production Mark I was essentially the same architecture but with the following enhancements: i) An instruction set of 50 op codes, ii) eight modifier registers (B-lines), iii) 256 words of main store (Williams Tubes), iv) 4K (extendable to 16k) of drum store, and v) faster multiply time (2.16 milliseconds).

The characteristics of the production Mark I are now described in greater detail, since they form the definitive expression of ideas contained in the series of University prototypes developed during the period 1946–49.

The 20-bit instruction format was as follows:

ADDRESS	B		FUNCTION
10	3	1	6

The three B digits specified one of eight B-lines and the specification of BO was normally used to indicate no modification. There was a separate B-arithmetic unit and associated eight-line B-store for carrying out modifier-register manipulation. Normal operands were 40-bit words, the bits being treated either as a two's complement number or as an unsigned quantity depending on the instruction. The full instruction set is given in Appendix 1, and it may be seen that considerable help was given with multilength arithmetic. There is also a population count or "sideways add" order (denoted in Appendix 1 by the mnemonic SADD), a facility requested by the Manchester mathematicians for their number theory problems. A similar instruction is provided on some modern computers, e.g. the CDC 7600, for nuclear physics applications programming, etc. The Ferranti Mark I also had a hardware random-number generator, available via mnemonic RNDM in Appendix 1. This somewhat unusual facility was included mainly at the request of the mathematician A.M. Turing, who was at Manchester from September 1948 until his death in June 1954.

Transfers to and from the drum and other peripheral equipment were carried out via 20-bit control words. These had two formats, distinguished by one of the mode bits. For *drum* transfers the format was:

DRUM TRACK ADDRESS	MODE		TUBE
11	4	1	4

Three of the mode bits then specified reading/writing, read-checking/write-checking, single page/double page transfers. The main store was arranged as eight 32-word pages on eight Williams Tubes, backed by the drum(s). The track-address was stored along with each page of information on the drum, and when a page became resident in main store an extra 20-bit line was assigned on each Williams Tube to hold the track-address of that page. This page-address line was normally invisible to the programmer, but could be accessed via the special LDAD instruction (Appendix 1). This was the germ of an idea which later led to page-address registers and virtual-to-real address translation on the Atlas computer.

For input/output transfers the control word format was:

	MODE	
10	5	5

For the early Ferranti Mark I's input was via a 250 character/sec 5-track paper tape reader using the 5-bit teleprinter code, and output was to a tape punch and printer. Four mode bits in the control word specified: Output a character, check output buffer, input a character, send a control character (equal to carriage return, linefeed, figure shift, letter shift) to the output device. Two input/output commands were provided (see Appendix 1): one took its control word from a main store address and the other used a 20-digit pattern set on the console switches—useful during bootstrapping.

c) System Software. In 1949 there was no Mark I system software, except for basic utilities such as input routines. Coding was normally carried out using the symbols of the 5-track teleprinter code. Once the Ferranti Mark I had been installed software development increased, with the emphasis being on embryonic high-level languages. After one earlier effort at compiler writing (1952), the Mark I Autocode [1] was available from March 1954 as an easy-to-learn scientific programming language for users having small or medium-sized problems.

An additional Autocode implementation objective was to simulate a one-level store so that the user had no need to organize his own drum transfers. It was possible to simulate this one-level store on the Mark I in a reasonably balanced way because the access time for reading an operand from the drum happened to be about the same time as a floating-point addition via an interpretive library routine. When running Autocode programs 128 tracks on the drum were reserved for instructions and 128 tracks for variables. Individual routines were transferred to the fast CRT store as they were required. To gain access to a variable an interpretive routine in fast store first determined on which track it lay, then transferred that track or "page" to the fast store, and finally selected the particular line within the page. Since successive operands were quite often located on the same page, steps were taken to avoid unnecessary drum transfers.

Arithmetic in the Autocode system was normally performed on floating-point variables $v1$, $v2$, . . . , etc. with provision for integers $n1$, $n2$, . . . , to be used as indices and counters. Simple conventions also existed for control transfers, intrinsic functions, input/output, and simple job control using symbols from the five-bit teleprinter code. An impression of the neatness of the system may be gained from the following Mark

I Autocode sequence which prints the root mean square (rms) of the variables $v1, v2, \ldots v100$. (Note that the symbol $*$ causes printing of a variable to ten decimal places on a new line, and $F1$ signifies the intrinsic function 'square root'):

$n1 = 1$
$v101 = 0$
$2v102 = vn1 \times vn1$
$v101 = v101 + v102$
$n1 = n1 + 1$
$j2, 100 \geq n1$
$v101 = v101/100.0$
$*v101 = F1(v101)$

2.3 Evaluation

The Mark I, in common with most early computers, was first applied to scientific and engineering problems. Measurements performed on a sample of Mark I jobs estimated that 16% of computing time went on drum transfers, 28% on multiplication and 56% on other arithmetic operations. Multiplication took 2.16 milliseconds and other accumulator orders took 1.2 milliseconds.

A contemporary benchmarking exercise rated the Mark I at about the same raw power as the National Physical Laboratories' ACE computer, even though the ACE had a digit period ten times shorter. The favorable performance of the Mark I was attributed to its random access main memory (ACE had a delay line store) and its relatively fast multiplier. Ferranti delivered nine Mark I and Mark I star machines between 1951 and 1957, three of them being exported (to Canada, Holland, Italy).

The long-term significance of the Manchester Mark I project is threefold. Firstly, it proved the viability of a digital storage technique (the Williams Tube), at a time when the successful implementation of the stored-program concept awaited the development of a suitable storage device. Williams Tubes were adopted by several computers in England, Russia, and America—including the IBM 701. Secondly, the project inspired the British government to give financial support to Ferranti Ltd., thus laying one of the cornerstones of the British computer industry. Thirdly, and of perhaps wider significance, the Mark I project was the first to focus attention onto the problems of linking fast random-access main memory to slower sequential-access rotating memory.

It was in the light of these problems that B-lines were first conceived of as relocation registers. It soon became clear that B-lines could also be used for general address-modification purposes, and so with the inclusion of a B-test facility the modern index register was born. The problem of automating backing store transfers ("overlays") still remained a challenge, but two Mark I facilities were later to suggest a solution to the Manchester team. First there was the fact that every page resident in the Mark I fast store carried with it the corresponding drum address in a special "page-address line" (see above). Second, there was the way in which the Autocode system handled the *drum* address of a user's variables. Out of these two facilities grew the concept of allowing the user always to program in a virtual (or "drum") address space and then providing system hardware and software to achieve automatic translation into the real (or "fast") address space, using information held in a set of associatively interrogated page-address registers. Thus the automated "one-level store" was conceived, and the realization of the other programming advantages to be gained from separating virtual and real address spaces followed shortly. These ideas were implemented in the Atlas computer.

3. Atlas

3.1 Objectives of the Project

By 1956 it was clear that Britain was falling behind the United States in the production of high-performance computers. The MUSE ("micro-second") project, started by Kilburn at Manchester in the autumn of 1956, was a conscious effort to remedy the situation. From January 1959 Ferranti Ltd. officially became involved and a joint University/Ferranti team under Kilburn continued the development of the computer, which was now known as Atlas.

Initial discussions with potential users of high-performance machines, both scientific and commercial, had produced a requirement for instruction times approaching one microsecond, the ability to attach a large number of i/o devices of various types and a main store size approaching 100k words. High computing speeds and rapid turnaround of user jobs became the keynotes of the Atlas design. The principal difficulties in achieving these goals arose from the wide differences in operating speeds between the various types of peripheral equipment and the CPU, and between transistor logic circuits and available core stores. Efficient and economic utilization of equipment was also very much a design-objective, since Atlas was intended to be sold on the open market. The somewhat conflicting requirements for high-speed and relative economy led to the incorporation of many techniques which were not extant when the project started in 1956. Amongst these were multiprogramming, job scheduling, spooling, extracodes, interrupts, pipelining, interleaved storage, autonomous transfer units, virtual storage, and paging. Although not all of these ideas originated in Manchester, they combined to make Atlas probably the most powerful machine available in the early 1960's.

3.2 Principal Features

a) Technology. The Atlas logic circuits were based on an OC170 germanium junction transistor used as an inverter, preceded by germanium OA47

diodes for the logic gating. This gave a typical gate-delay of 12 nanoseconds. Care was taken to avoid saturation (low collector-base volts), since the OC170's response became slow in the saturation region. The parallel adder employed a special symmetrical transistor (the SB240) as a switch in the carry-path, which resulted in a basic add-time of 200 nanoseconds for 48 bits—a significant achievement in 1959. There were about 80,000 transistors in the entire computer, mostly mounted on 8-inch by 5-inch printed-circuit boards.

The main store was 2 microsecond cycle-time core four-way interleaved, backed by drums having a 12 millisecond revolution time and capable of transferring one 512-word block every 2 milliseconds. Two other "private" storage units were provided: A high-speed read-only "fixed store" of 0.35 microsecond access time made from small slugs of copper or ferrite inserted in a woven wire mesh; and a system working store of 2 microsecond cycle-time core, which served as working space for the operating system routines (many of which resided in the fixed store). The size of all these stores varied between production versions of the Atlas. The Manchester prototype had the smallest capacity, expressed in 48-bit words as follows:

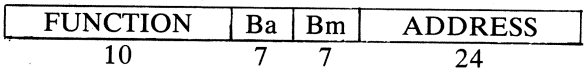
i) main store: 16k core, backed by four drums each of 24k
ii) fixed store: 8k
iii) system working store: 1k (later increased to 4k)

The largest production Atlas, installed at the Science Research Council's computing laboratory at Chilton (Harwell), had a main core store of 48k.

Bulk storage was provided by eight (expandable to 32) tape decks on eight channels, each having a transfer rate of 90k characters per second. Preaddressing and fixed 512-word blocks were used, thus allowing a tape to be written to nonsequentially when required. A 16-million word file disk was added later.

The Manchester Atlas had 17 conventional i/o devices, two high-speed data links, an on-line x-ray crystallographic diffractometer and an experimental speech input/output unit. The interrupt structure allowed for the connection of up to 512 peripheral units, with hardware assistance for determining the source of an interrupt.

b) Architecture. Atlas was a 48-bit word parallel computer with a one-address instruction format as follows:



The repertoire of functions or op codes, summarized in Appendix 2, was divided into two groups: normal instructions and extracode instructions. Generally speaking an extracode was a commonly used but relatively complicated function which it was not economic to implement directly as hardwired logic. Instead, an extracode consisted of a sequence of normal instructions (a "macro routine") held in the fixed store. Entry to these macro routines was very rapid and involved no preservation of central registers since there was a dedicated extracode program counter (or control register) and reserved B-lines, and any extracodes needing working space used a private area of the system working store. Amongst extracode instructions available to the user were ones for carrying out the common intrinsic functions such as square root, log, cosine, etc.

Of the normal instructions, Appendix 2 shows that they divide into three subgroups: main accumulator (A) orders, index register (B) orders, and test-and-count orders. There were independent A and B arithmetic units. The instruction set and the Atlas pipeline architecture assumed there would normally be no interchange of operands between the A and B ALUs.

This design philosophy, also to be seen to some degree in MU5, works most effectively for computations such as forming the scalar product of two vectors. By careful pipeline design and by using tricks such as assuming that the next instruction usually occurred in the same page as the last instruction, Atlas could overlap the execution of three A-instructions and then any associated B-instructions were normally executed concurrently with minimal additional time penalty.

The Atlas instruction could be double address-modified, according to the specification of the Ba and Bm bits. There were 127 24-bit B-lines (index registers) for this purpose, mostly held in a 0.7 microsecond cycle-time core store. The top three B-lines, B125–B127, were implemented as flip-flop registers and were reserved for use as independent program counters respectively for interrupt, extracode, and main program control. This explains why no explicit jump (branch) instructions appear in Appendix 2.

Of the 24 address bits in an instruction, 20 were used to cover the virtual address space of one million words, three specified a 6-bit character position within a full word, and one bit distinguished between a normal address and a "*V*-store" address. The *V*-store was the collective name given to all central registers and peripheral device registers which needed to be accessible to a (system) program. Into this category came such things as interrupt registers, page-address registers, and the data and status registers of all i/o units. Since normal instructions could, with suitable protection checks, use *V*-store addresses for operands there was no need for explicit op codes for the control of i/o equipment etc. The incorporation of peripheral devices into the total address space has since been used on other computers such as the PDP11.

The Atlas paging system used 512-word fixed-size pages, with a page-address register for every 512-word section of main core store. Each register contained a lock-out digit, so that pages of more than one program could be resident in core concurrently. The address-translation time, i.e. associative interrogation of the

page-address registers, was 0.7 microseconds, which represented about 40 percent of the total operand fetch time (including cable delays, store access time, etc.). Considerable effort was spent in ensuring efficient page-turning, and the replacement algorithm—contained in the fixed store—used a learning program which attempted to identify pages in main store which had fallen out of use [5]. No copy was normally kept on drum, and each drum had a rotational position indicator to speed the transfer of a replaced page to the first available space on drum. (Many modern paging computers, including MU5, now keep copies on drum and arrange not to write back pages which are unaltered.) The Atlas virtual address space was sufficiently large for the compilers to arrange simple segmentation conventions during the compilation process.

c) System Software. The Atlas Supervisor (operating system) fully exploited the following concepts: i) Multiprogramming (of up to 16 jobs concurrently), ii) on-line spooling of input and output, and iii) job scheduling (according to user-indicated job characteristics such as use of magnetic tapes, volume of output, or priority request). The aim was to keep all of the computer equipment busy while minimizing the turnaround of individual jobs. Since in addition the Supervisor produced comprehensive logging statistics for the user, the operator, and the (daily) accounting program, the computer room operators had little to do except feed in work—an exceptional state of affairs in the 1960's. The programmer was provided with straightforward job control conventions which were simple for simple tasks, while allowing scope for more complex requirements such as multiple input or output streams, or the use of special "private" compilers.

The standard Atlas compilers were mostly implemented using the Compiler Compiler [2], a formal language for the transparent description of syntax and semantics. While compilers for all the common high-level languages such as Algol, Cobol, and Fortran were eventually written for Atlas, the Manchester users programmed extensively in Atlas Autocode to begin with. Atlas Autocode was a block-structured language specified at about the same time as Algol 60, but implemented sooner. Except for a more limited concept of compound statements and *for* clauses, Atlas Autocode was very similar to Algol 60. As an example the root mean square calculation given in Section 2.2 might be programmed in Atlas Autocode as follows:

```
RMS = 0
cycle i = 1, 1, 100
RMS = RMS + X(i) * X(i); repeat
print (sqrt (RMS/100), 6, 4)
```

3.3 Evaluation

Some typical Atlas instruction times were as follows:

fixed-point B addition	1.59 microsec.
floating-point add, no modification	1.61 microsec.
floating-point add, double modification	2.61 microsec.
floating-point multiply, double modification	4.97 microsec.
floating-point division	10.66 to 29.80 microsec.

The overall average instruction time when executing Fortran scientific programs was measured to be about 3.35 microseconds per order. For comparison, the corresponding overall average instruction rates for typical present-day computers executing similar programs range from about 110 nanoseconds per order (CDC 7600) to about 6.6 microseconds per order (IBM 370/135).

In terms of contemporary machines Ferranti salesmen equated one Atlas to four IBM 7094s as regards work throughput. It is disappointing that only three full Atlas's and two scaled-down versions ("Atlas II") were sold between 1962–65. To some extent the marketing of Atlas was overtaken by events: Ferranti was currently developing other systems besides Atlas; Ferranti sold its large computer interest to ICT (later ICL) in 1963; ICT afterwards introduced its 1900 range of computers. By the mid-1960s the direct market competitor to Atlas was the faster CDC 6600, whose designers have said they learned some useful lessons from Atlas. In 1967 a benchmarking comparison for 22 compute-bound Fortran scientific jobs was performed on an Atlas with 32k core, a single-processor Univac 1108 with 64k core, and a CDC 6600 with 64k core [3]. The compilers used were respectively the London nonoptimizing Atlas Fortran V, the Univac F4012 Fortran IV, and the CDC Chippewa Run. Under these conditions the average CP computing speeds for Atlas, Univac 1108, and CDC 6600 were measured to be in the ratio 1: 2.1: 5.9 respectively. By the start of the 1970s ICL had introduced the 1906A computer, which has a work throughput of the order of three times that of Atlas, depending upon the configuration.

The long-term significance of the Atlas project lies in the design-concepts which it introduced. The more important of these are in four general areas: Pipeline techniques for high instruction throughput, paging and virtual storage, operating system features, and extracodes. The first area is now well-defined in respect of single instruction streams. The second has had far-reaching consequences. It has been the subject of much subsequent analysis and development (e.g. for MU5), in the light of which it is interesting to observe that the inspired guess of 512-word pages for Atlas proved to be about right. Programs on Atlas generally produced about one page-exception (page-address register nonequivalence) per 5×10^4 accesses. With regard to operating system features, the Atlas Supervisor would seem to the present-day user to have a very limited concept of file manipulation and no time-sharing (interactive) facilities. However, in all other

respects, especially in job throughput and ease of use, the Atlas Supervisor set a high standard which is still relevant today. Atlas was also one of first computers in which specific hardware facilities were provided at the design stage to aid the operating system—e.g. in the area of peripheral control, interrupt handling, and store management. With regard to extracodes, the concept of providing easy access to commonly required software has been taken up by several other designers. On Atlas the existence of a specially constructed fixed store for extracodes and certain Supervisor routines was partly determined by nonavailability of suitable fast core. It was generally observed that over half of all Atlas user-programs spent more than half their run time in executing common software such as the extracodes and i/o routines, so there was ample justification for having speeded up the access to much of this standard software.

As for the influence of Atlas on the design of its successor at Manchester, MU5, an important lesson was learned concerning the B-lines. The Mark I had had eight such lines and on Atlas about 90 of the 127 B-lines were available to the general user. This was indeed a lavish provision for the *assembler* programmer, but it was observed that the compiled code of the Atlas high-level language programmer could not easily make use of more than a few of these. A compiler writer has a potential need for registers such as B-lines for two main object-code purposes:

i) as bases and pointers for address-generation;
ii) as fast storage for frequently used operands when attempting run-time optimization—(this applies not only to integers for loop-control etc., but also to frequently-used floating-point variables).

For the former application the registers should be capable of reflecting the usage of local and nonlocal name spaces in block-structured languages—though there is no such special requirement for languages such as Fortran. For the latter application it would be advantageous if identification of frequently used operands was automatic at run time, thus saving on compiler complexity. The MU5 design attempts to satisfy these high-level language requirements with specific naming registers and associatively accessed buffers, and from Manchester's viewpoint Atlas marked the end of the road for the general-purpose B-line. The vindication of the MU5 approach lies in the more efficient compiled code which it produces [7].

4. Conclusion

This paper and its companion [4] reveal a developing view of computer design over a period of 30 years. At the beginning of this period the task of inventing the basic functional units and then keeping them running for long enough to obtain useful computation, dictated a spartan engineering approach to machine architecture. As technology advanced and successive Manchester computers were implemented and evaluated, so the designers were able to observe and incorporate in hardware more of the requirements of the system software and the users. These requirements themselves evolved over the years. Although the emphasis of the Mark I, Atlas, and MU5 has been on large high-performance systems, it is evident that the designs have not only kept abreast of the requirements of the general user, but in some cases (e.g. paging and virtual storage) the architectural innovations have been in advance of the facilities expected by normal programmers. In time, with the decreasing cost of logic and main storage, many of the Manchester "high-performance" devices have come to be adopted in succeeding middle-range computers.

Appendix 1

The Instruction Set of the Ferranti Mark I

In order to relate to modern terminology the following notation is used when describing the action of each order:

ACC: the contents of the double-length main accumulator (80 bits)
AM: the most-significant 40 bits of ACC
AL: the least-significant 40 bits of ACC
S: the contents of a store line (40 bits), except that B orders use the least significant 20 bits and control-transfer orders the least significant 10 bits.
B: the contents of a B-line (index register)
D: the contents of the multiplicand register (40 bits)
H: the digits set up on 20 console handswitches.

a) Main Arithmetic and Logical Orders

Mnemonic	Description
LDA	load AL (AM cleared)
LDAS	load AL, sign-extend into AM
LDN	load AL negatively
STA	store AL
STM	store AM
STMC	store AM and clear AM
SWAP	interchange AM and AL
STAM	store AL, move AM to AL and clear AM
STAC	store AL and clear ACC
CLR	clear ACC
ADD	ACC := ACC + S (signed S)
ADDU	ACC := ACC + S (unsigned S)
SUB	ACC := ACC − S (signed S)
ADDM	AM := AM + S
LDDU	load D (unsigned multiplicand)
LDDS	load D (signed multiplicand)
MADU	ACC := ACC + D × S (unsigned S)

MADS	ACC := ACC + D × S (signed S)
MSBU	ACC := ACC − D × S (unsigned S)
MSBS	ACC := ACC − D × S (signed S)
AND	ACC := ACC & S (S sign-extended)
ORA	ACC := ACC or S (S sign-extended)
NEQ	ACC := ACC ≠ S (S sign-extended)
SHLS	ACC := 2 × S (arithmetic shift)
ORS	S := AL := AL or S
ORSC	S := AL or S, then clear ACC

b) B-line (Index-Register) Manipulation

Mnemonic	Description
LDB	load a specified B-line
STB	store a specified B-line
SUBB	B := B − S
LDBX	load a B-line (without modification)
STBX	store a B-line (without modification)
SBBX	B := B−S (without modification)

c) Control Transfer Orders

Mnemonic	Description
JMPA	absolute indirect unconditional jump
JMPR	relative indirect unconditional jump
JGEA	if ACC ≥ 0, absolute indirect jump
JGER	if ACC ≥ 0, relative indirect jump
JGBA	if (last-named B-line) ≥ 0, absolute indirect jump
JGBR	if (last-named B-line) ≥ 0, relative indirect jump

d) Peripheral and Miscellaneous Orders

Mnemonic	Description
IOTH	i/o transfer using H as a control word
IOTS	i/o transfer using S as a control word
NORM	add to AM the position of the most significant one in S
SADD	add to AM the number of 1's in S − (population count)
RNDM	load a random number into AL
LDAD	load a page-address word into AL
DST1	debugging stop (1)
DST2	debugging stop (2)
TIME	S := clock
HOOT	pulse the console hooter
STH	S := console handswitches H
NULL	no operation

Appendix 2

Abbreviated Summary of the Atlas Instruction Set

In describing the action of the orders the following notation is used:

AM: the contents of the main 48-bit accumulator. (For floating-point working a 40-bit mantissa, 8-bit exponent and an octal base is used. For fixed-point working only 40 bits are used.)

AL: for double-length working AL forms a 39-bit mantissa extension.

S: the contents of a store line (normally 48 bits)

BA, BM: the contents of 24-bit B-lines (as addressed by the Ba and Bm fields).

BT: the contents of a B-test register.

N: a 24-bit literal ("immediate operand"), specified by taking the value of the instructions' address field as a two's complement number.

a) Main Accumulator Arithmetic Orders

(Note that several arithmetic orders were repeated with minor differences concerning accumulator standardization, rounding, clearing of AL, etc. Such orders are asterisked).

Mnemonic	Description
LDA	load AM (* four versions)
LDN	load AM negatively (* three versions)
LDL	load AL (* two versions)
LDDL	load AM and AL double-length
LDDLN	load double-length negatively
STA	store AM (* two versions)
STL	store AL (* two versions)
STDL	store AM and AL double length
ADD	fixed-point add
ADFL	single-length floating point add (* two versions)
ADFD	double-length floating point add
SUB	fixed-point subtract
SBFL	single-length floating point subtract (* two versions)
SBFD	double-length floating point subtract
RSUB	fixed-point reverse subtract
RSBFL	single-length floating point reverse subtract (* two versions)
RSBFD	double-length floating point reverse subtract
MPY	fixed-point multiply
MPFL	single-length floating point multiply (* two versions)
MPFD	double-length floating point multiply
NMPY	fixed-point multiply and negate
NMPFL	single-length floating point multiply and negate (* two versions)
NMPFD	double-length floating point multiply and negate
DIV	fixed-point divide
DVFL	single-length floating point divide
DVDL	double-length floating point divide

There were an additional 17 orders for performing miscellaneous minor operations on the accumulator such as negating, taking the modulus, etc.

b) B-line ("Index Register") Manipulations

(Note that orders asterisked used the 'read-pause-write' (split cycle) technique.)

Mnemonic	Description
LDB	load a specified BA (BA′ = S)
LDBN	load negatively a specified BA
LND	load literal (BA := N)
LNN	load negatively a literal (BA := − N)
STB	store a specified BA (S := BA)
STBN	store negatively a specified BA
ADB	add (BA := BA + S)
ADN	add literal (BA := BA + N)
SBB	subtract (BA := BA − S)
RSBB	reverse subtract (BA := S − BA)
SADB	add into store (S := S + BA) *
SSBB	subtract from store (S := S − BA) *
RSSBB	reverse subtract from store (S := BA − S) *
SBN	subtract literal (BA := BA − N)
RSBN	reverse subtract literal (BA := N − BA)
AND	BA := BA & S
ANDS	S := BA & S *
ANDN	BA := BA & N
ANMN	BA := BM & N
ADMN	BA := BA + (BM & N)
NEQ	BA := BA ≠ S
NEQS	S := BA ≠ S *
NEQN	BA := BA ≠ N
ORB	BA := BA or S
ORBN	BA := BA or N

There were a further four miscellaneous simple B orders. More complex B operations, including multiplication, were peformed by extracodes—see Section d below.

c) Test and Count Orders.

i) Six orders of the form: BA := N *IF* BM is: odd, even, = 0 ≠ 0 ≥0, <0.
ii) Four orders of the form: BA := N *IF* BT is: = 0, ≠ 0, ≥0, <0.
iii) Four orders of the form: BA := N *IF* (AM, AL) is: = 0, ≠ 0, ≥0, <0.
iv) Four orders which set BT according to the result of: (S − BA), (BA − S), (N − BA), (BA − N).
v) Four orders of the form: *IF* BM ≠ 0, BA := N *AND*: (BM := BM + ½), (BM := BM + 1), (BM := BM − ½), (BM := BM − 1).
vi) Four orders of the form: *IF* BT ≠ 0, BA = N *AND*: (BM := BM + ½), (BM := BM + 1), (BM := BM − ½), (BM := BM − 1).

Note that BA was thought of as the "arithmetic" B-line and BM as the "address-modification" B-line. Adding ½ to BM allowed halfword boundaries to be accessed.

d) Extracodes. These caused automatic entry to and return from fixed-store routines. Extracodes intended for general use divided into the following groups:

i) 14 extracodes for operating on B-lines, providing multiplication, division, shifting, etc.
ii) 46 extracodes giving additional (AL, AM) facilities such as:
arithmetic using literals
evaluation of standard trigonometric functions
evaluation of other standard functions such as log, exp, sqrt, reciprocal, etc.
iii) Three extracodes for subroutine entry (return link stored in BA).
iv) 20 user-orientated extracodes for the control of magnetic tape, i/o stream selection, etc.

The rest of the fixed store was filled with system software such as the drum learning program, i/o device routines, and standard test programs.

Acknowledgments. The author would like to thank Professor Tom Kilburn and many of the staff in the Department of Computer Science, University of Manchester, for the helpful discussions concerning systems described in this paper.

Received March 1977; revised July 1977

References

1. Brooker, R.A. An attempt to simplify coding for the Manchester electronic computer. *Brit. J. Appl. Physics 6* (1955), 307-311.
2. Brooker, R.A., MacCallum, I.R., Morris, D., and Rohl, J.S. The compiler compiler. *Ann. Rev. in Automatic Programming, 3* (1963), 229ff.
3. Hughes, P.H. University computer benchmark report. Atlas Computing Service, U. of London, July 1967.
4. Ibbett, R.N., and Capon, P.C. The development of the MU5 computer system. *Comm. ACM 21*, 1 (Jan. 1978), 14-25.
5. Kilburn, T., Edwards, D.B.G., Lanigan, M.J., and Sumner, F.H. One-level storage system. *IRE Trans. EC-11*, 2 (1962), 223-235 (Reprinted in Bell, C.G., and Newell, A. *Computer Structurers: Readings and Examples*. McGraw-Hill, New York, 1971).
6. Lavington, S.H. *A History of Manchester Computers*. Nat. Comptng. Ctr. Publications, Manchester, England, 1975 (also published in the U.S. by Hayden, Rochelle Pk, N.J.)
7. Lavington, S.H., and Knowles, A.E. Assessing the power of an order code. Proc. IFIP Congress 77, Toronto, Canada, 1977, pp.
8. Morris, D., Sumner, F.H., and Wyld, M.T. An appraisal of the Atlas Supervisor. Proc ACM Nat. Meeting, 1967, pp. 67-75.
9. Williams, F.C., and Kilburn, T. A storage system for use with binary digital computing machines. *Proc. IEE*, Vol. 96, Pt. 2, No. 30, 1949, p. 183ff.
10. Williams, F.C., Kilburn, T., and Tootill, G.C. Universal high-speed digital computers: A small-scale experimental machine. Proc. IEE, Vol. 98, Pt. 2, No. 61, 1951, pp. 13-28.

The Outer View
It's always darkest . . .

Nancy Foy, M.A.C.M.

Philip Dorn has unplugged more IBM equipment than most users ever installed. *Datamation's* 'outspoken user' (now a Senior Manager at Equitable Life Assurance in New York) came to London recently to see what was happening in computer systems measurement here. I cornered him after the Infotech session on this subject, and received a two-minute precis: 'Professor Meir Lehman at Imperial College is continuing his enormously impressive work (begun at IBM) on System Growth Dynamics. Ken Kolance is developing a tentative theory of software physics . . . using the concepts of force and power and so on to measure system capability. I was impressed with the work done by the Manchester Group described by Professor Sumner which used ATLAS to ascertain some factual data regarding how users deal with a computer in such areas as instruction mix . . . a most logical and intelligent experiment and one which suggests that the MU-5 could be the next seminal machine as the ATLAS was the last.'

Dorn was surprised at the lack of interest in chargeback systems. 'I heard nothing to deter me from the Equitable's notion that billing should be transaction-oriented rather than resource-oriented,' he commented. 'But none of us know quite how to do transaction billing yet. And nobody seems to understand the distinction between costs and prices. Your Lord Brown has pointed this out in his book *Organisation* and other writings but nobody except John Gosden, a UK-born Equitable V-P, seems to have connected his ideas to computer-related problems.'

Compatible peripherals, food and wine

When he's not claiming to be 'Chief Billing Clerk' for a multi-billion-dollar company, Dorn claims to be a technical man. Certainly he has spent most of his recent years smashing out computer configurations. However, he's slightly ahead of the technical pack, it seems to me. He gets his job satisfactions not from buying more advanced hardware, but from chopping dollars out of the computing budget. That's why he has become one of the world's leading specialists in the plug-to-plug peripheral business. His credentials run to two single-spaced pages (that's things like lumping Joint Computer Conference Speeches '1956 on' or 'Diebold Seminars' into single entries). He's a past president of SHARE, the IBM scientific user group and a former ACM chapter chairman. He has been an ACM national lecturer, a gruelling job, but one of the highest accolades a computer professional can receive in the United States (and one we ought to consider here). His international experience is sufficient to enable him to find good food, fine wine and the brightest professionals for company in any country he visits. As a contributing editor to *Datamation* his was the 'anonymous' voice recently quoted as saying 'If Telex and Memorex (later Datran and MCI) didn't exist, we'd have to invent them to keep IBM (later AT&T) in line'.

There's a quotation from Frank Herbert in the science fiction novel *Dune* that Phil likes even better: 'Progress is the concept that man has developed to shield himself from the horrors of the future.'

He's concerned that the computer manufacturers aren't keeping up with the real needs of users. 'As a large commercial user I'm not interested in engineers sitting around playing with Bessel functions,' he says. 'I want my commercial programmers online, building programs. But the manufacturers don't know or care now about the needs of the user, and they never did.'

'What happens when the computer manufacturer gets back his obsolete hardware?' Dorn asks. 'He eventually puts it to work inside his own company. Therefore, since he's a generation behind his customers anyway, he doesn't understand the interface to the users. He doesn't even understand the interfaces to the machine operators. They go into hysterics when they see some of the new machines. They don't know or care what all those lights mean, but it scares them when the red ones come on.'

1963 Gothic?

Dorn contends that the 360 was designed before 1963, and all the 360 copies and the 370 too are based on the same old architecture. 'Clearly we have better circuits. They're more reliable, and the machines don't collapse quite as often. Our hardware is good. It runs. Unfortunateiy, between the hardware and the user, there is OS and JCL.

'This reminds me of Meir Lehman's work again. His curves, when applied to something like the operating system for a modern computer, show how the system goes critical (the point when correcting one fault induces an additional fault) at an exponential rate. The curves clearly indicate why IBM terminated DOS developmental efforts at Release 26, the "anti-regressive" efforts had become so overwhelming that there were no resources available for "progressive" activities. Professor Lehman has uncovered a strong suggestion that the same curves hold up equally well for decay rates on any large system—such as an urban system.'

Back on the hardware mainstream, he continues to outspeak. 'IBM and other manufacturers talk about fast memory and new circuits. That's hogwash. Pico-seconds just don't interest me, and transfer rate is just something you sell your management on. It's not really important. There are much more vital questions: Can I get the data in? Can I get answers out? Can I use that memory? The concepts of virtual memory, and the multiprogramming multiprocessor are obviously the ways to go, but we don't see many systems like this on the market.

'The processor speed doesn't matter. Good Lord! Nobody uses the speed we've got already. It used to be that an obsolete computer was one to which your vice-president could no longer point with pride. Now the time for buying shiny new computers is past.

'What we really need is reliability and availability. The real question is how often the *system* fails. Whether it's caused by a compiler, the OS, or a $500 typewriter, when a failure occurs the entire system is down. Besides reliable hardware we need reliable software. Programming standards are something we just talk about. Most of us don't even use the ones we already have.'

Philip Dorn insists that one of the most important needs, especially for multinational corporations, is large data files. 'We still have a huge data collection problem. It would be nice to have distributed intelligence, but the minicomputers have only limited storage. Somewhere behind all that distributed intelligence (not necessarily online), I have to have something big. When you begin gathering market and customer information and feeding it into a model, you can't work on small machines. You have to do it on a viable time scale, with a big and accessible machine.'

Data file technology

Hardware for these large data files is just beginning to come along. 'By 1975', says Dorn, 'we will be seeing just the beginning, expensive but real. These developments may come from the laser people (the first of the

The Manchester University Atlas Operating System
Part I: Internal Organization

By T. Kilburn, D. J. Howarth, R. B. Payne and F. H. Sumner

The organization of input and output on the Atlas is described. This is controlled by the supervisor program, and uses varying amounts of core and drum store, supplemented where necessary by magnetic tapes, to obtain, when possible, the maximum overlap between input, computing and output. The system will be used on the Manchester University Atlas, but is suitable for any Ferranti Atlas Installation.

Introduction

Atlas* is the name given to a comprehensive computer system designed by a joint team of Ferranti Ltd. and Manchester University engineers. The computer system comprises the central computer, fixed store, core store, magnetic drum store, magnetic tapes, and a large quantity and variety of peripheral equipments for input and output. The Manchester University Atlas has 32 blocks of core store each of 512 forty-eight bit words. There is also a magnetic drum store, and transfers between core and drum stores are performed automatically, giving an effective one-level store* of over two hundred blocks. The average time for an instruction is between 1 and 2 microseconds. The peripheral equipments available on the Manchester University Atlas include

8 magnetic tape decks	90,000 characters per second
4 paper tape readers	300 characters per second
4 paper tape punches	110 characters per second
1 line printer	600 lines per minute
1 card reader	600 cards per minute
1 card punch	100 cards per minute

The operating system described is applicable to any Ferranti Atlas computer and will be used on the Manchester University Atlas. The system is designed to operate with any configuration of one-level store, magnetic tapes, and peripheral input and output equipments. If no magnetic tapes are available to the system the efficiency may be reduced.

The design of Atlas is such that the transfer of information to and from the core store is controlled by programs contained within the fixed store. In the case of drums and magnetic tapes, information is transferred in "blocks" of 512 words. Word by word transfers proceed automatically; fixed-store programs are only required at the beginning and end of a block. In the case of all other peripheral equipments (e.g. readers, printers, punches) information is copied to and from one-character buffer registers, and is transferred between these registers and the core store by fixed-store programs.

* Papers have been written on the Atlas and on the one-level store, and will be submitted to the Institution of Electrical Engineers and the Institute of Radio Engineers respectively.

These equipments are collectively referred to as "slow peripherals." They may include line printers, paper tape and card punches and readers, special magnetic tapes, graphical output devices, etc.

During the operation of most slow peripherals, the central computer is required for about 1 % of the time. Thus even with many slow peripherals operating, the central computer is available for direct execution of a problem for a large proportion of the time. The design of the Atlas is such that these problems cannot interfere with the operation of the fixed-store programs controlling the slow peripherals, which themselves are prevented from interfering by the *Supervisor Program.* The time taken to switch control between the main program and the fixed-store programs is about 10 μsec, and is included in the above estimate of 1 %. This time-sharing of the central computer is generally referred to as the "overlapping" of input, output, and computing.

If the slow peripheral equipments could always transfer information at the rate required by the central computer for any problem, then the maximum overlap of input, computing and output could be easily achieved. However, the central computer requires and produces information at widely varying rates, with an upper limit in excess of three hundred 512-word blocks per second. On the Manchester University Atlas, with four paper tape readers and one card reader, the maximum input rate with all equipments operating simultaneously is half a block per second. The maximum output rate with four paper tape punches, one card punch, and one line printer is also half a block per second. Magnetic tape input and output can increase these rates to sixteen blocks per second per channel. The use of magnetic tape for input and output requires a large amount of off-line editing equipment, and unless a large number of channels are available, it is still possible for the central computer to be input or output limited for short periods of time.

One method by which the amount of overlap could be increased is to hold several problems within the central computer at any one time, and, by means of the supervisor program, to arrange that control is switched between these problems in such a way that the rates at which information is required or produced by the central computer are reduced to those which can be dealt with

by the available slow peripherals. In order to achieve this, it is necessary that the available problems are such that, whilst some of them are producing results, others are computing and others require the input of further information. This needs either a large number of programs available within the central computer, or off-line organization of the problems presented to the computer. Furthermore, this switching between problems can considerably reduce the efficiency of the computer, as it is necessary to store and reset all the common working registers, requiring up to 1·5 msec on the Manchester University Atlas.

An alternative solution is to designate two areas of the one-level store as input and output wells, and to arrange that the central computer draws information from one well and supplies it to the other. These wells are filled and emptied by the slow peripherals operating in parallel. The use of these wells "smooths out" the variations in the rates at which information is required or produced by the central computer. The larger the wells, the greater the "smoothing out" and the greater the overlap obtainable for any sequence of problems. This method overlaps the computing of one problem with the input and output of others; only one problem need be in the central computer at any one time, and no switching of control between problems is necessary. The division of the one-level store between the central computer and the two wells is arranged by the supervisor program, and is capable of continuous variation. The size of the wells is increased by the use of magnetic tapes, and another magnetic tape is provided to enable the suspension and dumping of problems whose continued operation would disrupt the efficient operation of the system. Suspended problems are re-entered when the supervisor program considers such a re-entry to be worth while.

A block diagram of the system is shown in Fig. 1.

The Input System

In order to accelerate the filling of the input wells by the slow peripherals, all slow peripherals are permitted to operate simultaneously. Furthermore, in order that the input system should be as flexible as possible, it is not necessary for the input of one problem to be on one "tape" (in this context, "tape" implies a paper tape or a stack of cards) or to be through one equipment, or even for the component parts to be supplied in any particular order. All sections of input are headed by a title for the purpose of identification (see the paper by Howarth, Payne and Sumner on p. 226).

A complete problem is referred to as a "job" and one of the component sections contains a job title together with a list of the titles of the other sections. The input information is organized within the system by the supervisor program, which maintains lists of the titles and lists of incomplete and complete jobs. Only jobs whose slow peripheral input is completed are made available to the central computer. The system is best described by considering what happens when "tapes" having the various permissible headings are read into the core and drum store. A tape bearing a "job" title causes this title to be added to the list of incomplete jobs, and a system number is allocated to this job. The component titles of the job description are added to the list of titles. Subsequently, any "tape" bearing one of these titles causes the relevant entry in the title list to be labelled as "present," and the information is stored, together with the title and the job number. The first of these "tapes" to arrive is stored after the job description and, whenever any piece of information overflows from one 512-word block into another, then the last line of the first block contains the number of the next block, and the first line of the second block contains the job number. When all the component titles have been labelled as "present," then the job title is transferred to

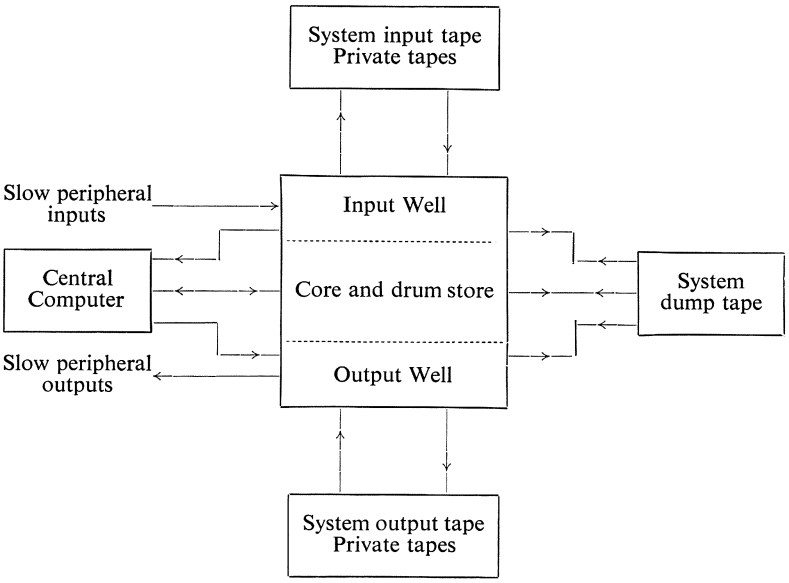


Fig. 1.—Block diagram of system

the list of complete jobs. Whenever the central computer requires a new job it extracts a title from this list, and all the appropriate blocks from the input well, which are then declared to be empty. If a component title is read before its job description, then the title is added to the title list, and the first action on reading a job description is to scan the title list for any component titles which may already be present.

So far no mention has been made of the system input magnetic tape; the use of this tape is not essential to the system provided that the input well in the core and drum store can be made sufficiently large to assemble complete problems read in from slow peripherals, and to retain them until they are required by the central computer for execution. However, without the system input tape, it is possible for the organization to become inefficient under certain circumstances. For instance, if computation on one problem continues for a long time, the amount of input supplied by the slow peripherals will exceed the size of any reasonable well; again, if a problem requires a large amount of store for its execution, then the size of the input well has to be reduced.

When the system input tape is in use, the input well within the core and drum store is divided into two parts. One part is filled by slow peripherals, and complete blocks of information are transferred from it to the system input tape. The other part is filled with complete problems by scanning the system input tape, and these problems are then transferred to the central computer as required. If information is being written to the system tape at point A, then the length of the system tape which may be scanned without effecting the flow of information from the slow peripherals is directly related to the size of the first part of the input well. For example, suppose this part of the input well consisted of five 512-word blocks (out of a total of over two hundred blocks in the core and drum store on the Manchester University Atlas). These can be filled by four paper tape readers and one card reader in 10 sec. The length of the scan is that amount of magnetic tape which can be scanned in 5 sec, that is 75 blocks. Thus the effective size of the input well is 80 blocks, only five of which need be in the core and drum store. These 80 blocks can be filled by the slow peripherals in 2·7 min. If this represented 80 programs in source language, then this well itself is capable of smoothing out an irregularity of 80 : 1 in computing to input time for an isolated problem. If at any time problems not yet dealt with by the central computer extend further along the system input tape than can be efficiently scanned, then either the input well can be increased, which increases the length of the permitted scan, or, if this is not possible, part of the system input tape is transferred to the system dump tape in order to reduce the length to be scanned. Problems so transferred will be treated later at a suitable time chosen by the supervisor program. The size of the second part of the input well will be allowed to change between the widest possible limits, since the rate at which blocks of information can be processed by the central computer will vary extensively.

When the system input tape is being used, all information from slow peripherals is stored on this tape, and a permanent record of all input is preserved. Thus it is possible to correct programs within the computer by reading in a short correction; the amount of off-line editing is therefore correspondingly reduced. The second half of the input well may also be loaded from previous system tapes and from private magnetic tapes, this operation being under the control of the supervisor program.

The Output System

The output well is divided between the slow peripheral output equipments in the approximate ratio of the rates at which they can transmit information. This division can be varied by the supervisor whenever the output for any slow peripheral exceeds its well size, provided such a change does not affect the operation of the other equipments.

The maximum rate at which the output well can be emptied when all the slow peripheral output equipments are operating simultaneously is 1,760 characters per second on the Manchester University Atlas, where the output equipment comprises one line printer, one card punch, and four paper tape punches. For a problem which produces a large amount of output for one or more peripheral equipments, the output well could be filled very quickly. If the wells cannot be extended, and if the system output tape is not being used, then the program producing the results must be suspended and possibly replaced by another.

The organization of the system output tape is similar to that of the system input tape. The output well is divided into two parts, the first part of which is filled by the central computer, and the second of which is filled by the system output tape. The system output tape is partitioned into sections; each section contains one block for each paper tape punch and card punch, and ten blocks for each line printer, i.e. fifteen blocks on the Manchester University Atlas when all slow peripherals are required. A complete section will keep all the slow peripherals operating for approximately 40 sec. The second part of the output well consists of one or more of these sections. If the slow peripherals are to operate at maximum efficiency, it is necessary that only complete sections should exist in the second part of the output well. These complete sections could be assembled by the supervisor in the first part of the output well, but this would require a very large output well if any problem produced several blocks of output for any particular slow peripheral. This could be overcome by permitting a program to produce only one block of output, and then suspending it and obeying other programs until a complete output section is obtained. The organization of such a system would decrease the efficiency of the central computer due to the necessity of program switching. The

solution adopted is to organize the compilation of complete sections along the system output tape. In this way a program can produce several blocks of information, partially filling several sections, before it need be suspended. The supervisor organizes the allocation of output channels in such a way that only complete sections are transferred to the second part of the output well.

Suppose the system output tape is transferring complete sections to the second output well from the point A, and a program has produced so much output for a particular slow peripheral that this output is being written into a section of the system output tape at the point B. Then if the system output tape can be scanned from A to B and back whilst the slow peripherals are dealing with the contents of the second well, the program producing this data need not be suspended. On the Manchester University Atlas, if the second output well is one section of fifteen blocks, this will occupy the slow peripherals for 40 sec. During this time, 630 blocks of magnetic tape may be scanned. Thus the maximum separation of the two points A and B is 315 blocks, or 21 sections. Therefore, in such a system, a program can produce 21 blocks of information, even for the slowest output, before it need be suspended. Ten times this amount of information could be produced for the line printer. This technique of writing to the system output tape is possible on Atlas because pre-addressed, fixed block-length, magnetic tapes are used.

The first part of the output well is not divided into sections, and if necessary the whole of this well can be filled with information for one particular slow peripheral output equipment. The size of the first part of the output well is varied by the supervisor and if at any time more output is produced than can be contained in this well, then this output is written to the dump tape and transferred into the well as soon as possible. Alternatively, the program supplying this large amount of output can be suspended.

The slow peripheral output equipments cannot operate continuously for more than about 10 to 15 min, because they have to be disengaged for loading a new reel of tape or more paper. The reloading operation may take 2 min. It is possible for the system to take account of this situation. At any time the supervisor program knows how many blocks of information have been supplied by the central computer to any slow peripheral, and can know when the tape or paper requires changing. It can then inform the operator of this, and suspend sending information to this output for a reasonable time during which the tape or paper can be renewed. This is easily done by writing incomplete sections on the system output tape. Furthermore, if a section arrives in the second output well and requires a slow peripheral which is disengaged, then the appropriate part of the section is transferred to the first output well, and thence to the system output tape again. If necessary the supply of information for this slow peripheral output from the central computer is suspended. In a similar way, if a slow peripheral output is permanently disengaged and information for it still exists on the system output tape, then this information is recirculated from the second to the first output well and reassigned to one of the slow peripherals which is operating.

The System Dump Tape

The use of this magnetic tape to store excess input and output information has already been described. Also, in the above description, reference has been made to the suspension of programs. When this occurs, the program is transferred to the system dump tape to make room in the core and drum store for a program which succeeds it. Another occasion when a program may be dumped is when it computes for so long that either the input well overflows or the output well is emptied. In either case, it is more efficient to proceed to a different problem in an attempt to maintain continuous overlap of input, computing, and output.

Acknowledgements

This work forms part of the Atlas project. It has benefited from many helpful discussions with the authors' colleagues at Manchester University and Ferranti Ltd., whose permission to publish is acknowledged.

The Manchester University Atlas Operating System
Part II: Users' Description

By D. J. Howarth, R. B. Payne and F. H. Sumner

A system of operating Atlas is described from the point of view of the user. The simple additional statements required from the programmer are supplied to the computer rather than a human operator.

The Manchester University Atlas will be used by various departments of the University and others, and will deal with a large quantity and variety of problems, some of which will complete computing in a few seconds. Consequently it is important that the computer operators be relieved of as much work as possible to ensure a smooth flow of work through the computer. This operating system has been devised by Manchester University and Ferranti Ltd., as a result of discussions between Professor T. Kilburn and the authors.

An important feature of the system is that it does not depend critically on how much peripheral equipment is operating; the system can function, though possibly with reduced efficiency, even if no magnetic tape decks are available to it. Normally three tapes are used to implement the system*:

1. the system input tape
2. the system output tape
3. the system dump tape.

Tape 1 buffers input from slow peripherals and also preserves a record of all input from slow peripherals. Tape 2 is similarly used for output and tape 3 has a variety of uses, including a store for programs temporarily held up and, where necessary, as an overflow of tapes 1 and 2.

The layout of all system tapes is the same, sufficient information being recorded to locate any information given the initial address on tape. Facilities are provided for the programmer to use information from these tapes instead of repeating input on slow peripherals; if a long paper tape is read in, it may be used again by referring to its location on a system tape. The location on the system tapes of all slow peripheral input and output dumps, etc., is printed with the programmer's results.

When large amounts of input or output are involved, the programmer may use private magnetic tapes to record the information. This is done by suitable specification in the title.

Titles and Headings

Jobs are initiated on the computer by input of information on a slow peripheral equipment. A job may consist of several sections of information, each preceded by an identifying title. This title, by which input information is known, consists of one line of printing following a heading such as

COMPILER (x)
DATA
JOB

where x may be

INTERMEDIATE INPUT
MERCURY AUTOCODE
FORTRAN
ATLAS AUTOCODE

If the heading of the input information is

COMPILER MERCURY AUTOCODE
(The title)

then the information is a source program in Mercury Autocode language. More generally, the information obeys the rules of Mercury Autocode, and may therefore include data as well as autocode instructions.

If the heading of the input information is

DATA
(The title)

then the information following is data to be read by a program during execution, and which obeys no rules known to the system.

The heading "compiler" does not itself initiate the appropriate compiling, which is only commenced when a "job" heading is read. If the heading is

JOB
(The title)

then the information following is the job description. In general this information is optional. It is terminated by an end-of-tape marker, in the case of a separate steering tape, or by "compiler" or "data," in which case the title is not repeated. The "job" heading will normally precede the source program tape as follows:

JOB
(The title)
COMPILER MERCURY AUTOCODE

then the source program itself.

Further optional information may be included in the job description such as:

* See the paper by Kilburn, Howarth, Payne and Sumner on p. 3 of this issue.

1. data and program tapes (input)
2. output equipments used
3. magnetic tapes
4. store required, computing time, etc.

These sections of the job description are described below.

Job Description—Input

A program reads in data by means of instructions which are effectively "read next character/string of characters from data tape *n*" where *n* is a decimal integer. (In this context, "data tapes" are intended to include stacks of cards.) The programmer's number of the data tape is specified in the "input" section of the job description. This section begins with the word

INPUT

and is followed by a list of the titles of data tapes used in this job, each preceded by the programmer's number, *n*, e.g.

INPUT
1 (the title of data 1)
2 (the title of data 2)

where there are two data tapes known by the programmer's numbers 1 and 2. These may have been read into the machine on the same input equipment as the "job" tape, either before or after it, or on other input equipments. The programmer's number, $n = 0$, is reserved for the program itself (and may be used in the program to read in data which follows the program as part of the same tape). A separate steering tape might be

JOB
(the title)
INPUT
1 (the title of data 1)
0 (the title of the program tape)

In this case, the name of the compiler to be used is written at the head of the program tape. When the "job" heading is on the beginning of a data tape, the "input" section of the job description must include

SELF = (*n*)

where *n* is the programmer's number by which this data is known within the program.

If the input section of the job description is omitted, it is taken as if

INPUT
SELF = 0

were included, and the program following is compiled and executed.

Since all input is automatically copied to the system input magnetic tape, a programmer may read his tape in again, direct from this input tape (e.g. to make a correction). To do this, in the "input" section of his job description he writes

TAPE (*a*)/(*b*)/(*c*)
(*n*) (the title of his input)

where *a* is the system tape "number,"
b is the number of the 512-word block of tape, and
c is the line within the tape block where his input starts.

His title is, of course, written on the tape at this point, but the title is specified again as a check.

Job Description—Output

A program puts out results by means of instructions which are effectively "print next character or string of characters on output *n*," where *n* is a decimal integer. The output equipments are specified in the "output" section of the job description. This section begins with the word

OUTPUT

and is followed by a list of the output mechanisms used in this job, each preceded by the programmer's number, *n*, e.g.

OUTPUT
1 (type of equipment) (*m*) BLOCKS
2 (type of equipment) (*m*) BLOCKS

The type of equipment may be

LINE PRINTER
TELETYPE
CARDS
FIVE-HOLE TELETYPE
ANY

where "Teletype" means a 7-hole (Teletype) paper tape punch,
"cards" mean a card punch
"any" means output on a line printer, Teletype punch, or cards.

The operators can control which equipments are used most by disengaging the other output equipments. *m* defines the limit of the output, and if the output exceeds *m* blocks of 4,096 characters, the program is stopped. If the number of blocks of output is not specified, it is taken as "1 block." Further, if there is only one output used, the output section may be omitted, and this is taken as if

OUTPUT
0 ANY 1 BLOCK

were included in the job description.

When printed, the output information itself is preceded by

OUTPUT (*n*)
(the title of the job)

and output of system information is always on output 0.

Job Description—Tapes

If a programmer uses magnetic tapes directly in his program (by use of tape instructions as distinct from using tapes in connection with input or output) then he specifies each tape used by two lines in the job description

TAPE
(*n*) (the title which is stored on block 0 of the tape)

where *n* is the programmer's number of the tape. When a new tape is required, the appropriate two lines of the job heading are

TAPE FREE
(*n*) (the title on block 0)

In this case, the title specified is written on Block 0 by the system.

If a file extends over several tapes, this is specified by a modified "tape" heading

TAPE/(*m*)
(*n*) (the title on block 0)

where *m* is the number of the continuation, counting from 1 upwards. The programmer's number *n* is the same for all *m*. The final tape of this file has the heading

TAPE/(*m*) END

If a program involves extensive input, then the job is preceded by copying this input to a magnetic tape. To initiate this copying process the input is headed

COPY TAPE FREE
(the title on block 0)

where the title specified is written on block 0. If a previously used tape is employed, the heading is

COPY TAPE (*b*)
(the title on block 0)

where *b* is the number of the tape block. (The programmer must always begin at the beginning of a tape block.)

Information may be read from this tape subsequently by specification of the tape and title of the information in the "input" section of the job description.

If a program involves extensive output then the output can be written on a private magnetic tape. This is specified in the "output" section of the job description as follows:

OUTPUT
(*n*) TAPE FREE/(type of equipment) (*m*) BLOCKS
(the title on block 0)

where *n*, "the type of equipment" and *m* are as for direct output, and where the title specified is written on block 0. If a previously used tape is employed, the specification is

(*n*) TAPE (*b*)/(type of equipment) (*m*) BLOCKS
(the title on block 0)

where *b* is the number of the tape block.

This private tape is printed by a steering tape consisting of

PRINT TAPE
(the title on block 0)

if the whole tape is to be printed, or

PRINT TAPE (*a*)/(*b*)/(*c*)
(the title of his output)

if one section of tape only is to be printed, from tape *a*, block *b*, word *c*.

Job Description—Miscellaneous

Further information may be given in the job description to indicate

1. the amount of core and drum store used,
2. the time for which the program is expected to compute,
3. the number of drums the program requires for programmed drum transfers.

All three apply to the execution stage of the program, i.e. excluding input from slow peripherals, compiling, and output to slow peripherals. These are specified by

	STORE	*s*	
	COMPUTING	*p.q*	HOURS
or	COMPUTING	*p.q*	MINUTES
or	COMPUTING	*p.q*	SECONDS
	DRUMS	*d*	

where *s* is the maximum number of core and drum 512-word blocks of store in use within the program during the execution stage, *p.q* is a fixed-point decimal number such as

COMPUTING 7·5 SECONDS

where the program is expected to run for not more than $7\frac{1}{2}$ seconds (if the estimate for store used and computing time is exceeded the program is stopped), and where *d* is the number of drums the program requires to reserve for programmed drum transfers.

If the total execution time is significantly different from the actual computing time, because there is considerable tape waiting time, the actual computing time should also be specified, e.g.

EXECUTION 5 MINUTES
COMPUTING 30 SECONDS

If information is not supplied in the job description, then

20 store blocks (10,240 words)
4 seconds computing time

and, of course, 0 drums are reserved. Estimates of the computing and execution times are taken as being equal unless both are specified explicitly.

End of Tape Markers

The end of a section of tape is indicated by

* * * (*x*)

where *x* is *Z*, *A*, *B*, *C* or *T*.

The marker

* * * *Z* indicates the genuine end of the tape/stack of cards

* * * *A* indicates "abandon previous incomplete section, if any" (this may be required by a machine operator)

* * * *B* indicates that a binary tape follows

* * * *C* indicates the end of a section, and that there is another section following on the same tape

* * * *T* indicates a temporary stop within a section.

The number of characters, *n*, on a binary tape may be indicated by

(*n*) * * * *B*

where *n* is a decimal number.

On reading the marker * * * *Z* the peripheral equipment is disengaged by the computer. When the operator next engages this equipment, a new section (with the appropriate heading and title) is read. The marker * * * *C* indicates the end of a section of tape, but the equipment is not disengaged and the next section is automatically read.

On reading the marker * * * *T* for a temporary stop, the equipment is disengaged as for * * * *Z*. However, when the operator next engages this equipment, a continuation of the current section (without a new heading) is read. Finally, on reading the marker * * * *B*, the computer reads the information following, in binary, without testing for further end-of-tape markers.

A better method of specifying the continuation of a section of data, without use of the marker * * * *T*, is by means of a modified "data" heading

DATA/(*n*)

where *n* is the number of the continuation of the section of data; e.g. for a program with data on two distinct paper tapes, the data may be headed

DATA/1
(the title of the data)

and

DATA/2 END
(the same title)

and each tape ends with the marker * * * *Z*. The continuation data tapes may be read into the computer in any order.

Acknowledgements

This work forms part of the Atlas project. It has benefited from many helpful discussions with the authors' colleagues at Manchester University and Ferranti Ltd., whose permission to publish is acknowledged.

An American Federation of Information Processing Societies Publication

THE ATLAS SUPERVISOR

T. Kilburn and R. B. Payne
The University of Manchester, Manchester, England

D. J. Howarth
Ferranti Limited, London, England

Reprinted From — COMPUTERS - KEY TO TOTAL SYSTEMS CONTROL

The Macmillan Company

New York

THE ATLAS SUPERVISOR

T. Kilburn and R. B. Payne
The University of Manchester, Manchester, England

D. J. Howarth
Ferranti Limited, London, England

1. INTRODUCTION

This paper gives a brief description of work originating in the Computer Group at Manchester University. Atlas* is the name given to a large computing system which can include a variety of peripheral equipments, and an extensive store. All the activities of the system are controlled by a program called the supervisor. Several types of store are used, and the addressing system enables a virtually unlimited amount of each to be included. The primary store consists of magnetic cores with a cycle time of under two microseconds, which is effectively reduced by multiple selection mechanisms. The core store is divided into 512 word "pages"; this is also the size of the fixed blocks on drums and magnetic tapes. The core store and drum store are addressed identically, and drum transfers are performed automatically as described in Section 3. There is a fixed store which consists of a wire mesh into which ferrite slugs are inserted; it has a fast read-out time, and is used to hold common routines including routines of the supervisor. A subsidiary core store is used as working space for the supervisor. The V-store is a collective name given to various flip-flops throughout the computer, which can be read, set, and re-set by reading from or writing to particular store addresses.

*A paper has been written on Atlas and it is hoped will be published by the Institute of Electrical Engineers.

The accumulator performs floating point arithmetic on 48-bit numbers, of which 8 bits are the exponent. There are 128 index registers, or B-lines, each 24 bits long; in the instruction code, which is of the one address type, each instruction refers to two B-lines which may modify the address and the average instruction time is between one and two microseconds. There are three control registers, referred to as "main control", "extracode control", and "interrupt control", which are also B-lines 127, 126 and 125. Main control is used by object programs. When main control is active, access to the subsidiary store and V-store is prevented by hardware, and this makes it possible to ensure that object programs cannot interfere with the supervisor. The fixed store contains about 250 subroutines which can be called in from an object program by single instructions called extracodes. When these routines are being obeyed, extracode control is used: extracode control is also used by the supervisor, which requires access to the "private" stores. Interrupt control is used in short routines within the supervisor which deal with peripheral equipment. These routines are entered at times dictated by the peripheral equipments; the program using main or extracode control is interrupted, and continues when the peripheral equipment routine is completed.

The first Atlas installation at Manchester University will include:

16,384 words of core store
8,192 words of fixed store
1,024 words of subsidiary store
98,304 words on drums
8 magnetic tape mechanisms
4 paper tape readers
4 paper tape punches
2 teleprinters
1 line printer
1 card reader
1 card punch

Other Atlas installations will include different amounts of store and peripheral equipments, but the supervisor program herein described is of sufficient generality to handle any configuration through the minor adjustment of parameters.

2. THE CO-ORDINATION OF ROUTINES

The Structure of the Supervisor

The supervisor program controls all those functions of the system that are not obtained merely by allowing the central computer to proceed with obeying an object program, or by allowing peripheral equipments to carry out their built-in operations. The supervisor therefore becomes active on frequent occasions and for a variety of reasons--in fact, whenever any part of the system requires attention from it. It becomes activated in several different ways. Firstly, it can be entered as a direct result of obeying an object program. Thus, a problem being executed calls for the supervisor whenever it requests an action that is subject to control by the supervisor, such as a request for transfer to or from peripheral equipments or the initiation of transfers between core store and magnetic drums; the supervisor is also activated when an object program requires monitoring for any reason such as exponent or division overflow, or exceeding store or time allocation. Secondly, the supervisor may be activated by various items of hardware which have completed their assigned tasks and require further attention. Thus, for example, drums and magnetic tapes call the supervisor into action whenever the transfer of a 512 word block to or from core store is completed; other peripheral equipments require attention whenever the one character or row buffer has been filled or emptied by the equipment. Lastly, certain failures of the central computer store, and peripheral equipments call the supervisor into action.

The central computer thus shares its time between these supervisor activities and the execution of object programs, and the design of Atlas and of the supervisor programs is such that there is mutual protection between object programs and all parts of the supervisor. The supervisor program consists of many branches which are normally dormant but which can be activated whenever required. The sequence in which the branches are activated is essentially random, being dictated by the course of an object program and the functioning of the peripheral equipments.

Interrupt Routines

The most frequent and rapidly activated parts of the supervisor are the interrupt routines. When a peripheral equipment requires attention, for example, an interrupt flip-flop is set which is available to the central computer as a digit in the V-store; a separate interrupt flip-flop is provided for each reason for interruption. If an interrupt flip-flop is set and interruptions are not inhibited, then before the next instruction is started, the address 2048 of the fixed store is written to the interrupt control register, B125, and control is switched to interrupt control. Further interruptions are inhibited until control reverts to main or extracode control. Under interrupt control, the fixed store program which is held at address 2048 onwards detects which interrupt flip-flop has been set and enters an appropriate interrupt routine in the fixed store. If more than one flip-flop is set, that of highest priority is dealt with first, the priority being built-in corresponding to the urgency of action required. By the use of special hardware attached to one of the B register, B123, the source of any interruption may be determined as a result of obeying between two and six instructions.

The interrupt routines so entered deal with the immediate cause of the particular interrupt. For example, when the one-character buffer associated with a paper tape reader has been filled, the appropriate interrupt flip-flop is set and the "Paper tape reader interrupt routine" is entered. This transfers the character to the required location in store after checking parity where appropriate. The paper tape reader meanwhile proceeds to read the next character to the buffer. Separate interrupt routines in the fixed store control

each type of peripheral equipment, magnetic tapes and drums. The interrupt technique is also employed to deal with certain exceptional situations which occur when the central computer cannot itself deal adequately with a problem under execution, for example, when there is an overflow or when a required block is not currently available in the core store. There are therefore interrupt flip-flops and interrupt routines to deal with such cases. Further routines are provided to deal with interruptions due to detected computer faults.

During the course of an interrupt routine further interruptions are inhibited, and the interrupt flip-flops remain set in the V-store. On resumption of main or extracode control, interruptions are again permitted. If one or more interrupt flip-flops have been set in the meantime, the relevant interrupt routines are obeyed in the sequence determined by their relative priority. In order to avoid interference with object programs or supervisory programs, interrupt routines use only restricted parts of the central computer, namely, the interrupt control register, B-lines 123 and 111 to 118 inclusive, private registers in subsidiary store and the V-store and locked out pages in core store (see Section 3). With the exception of the B-lines, no object program is permitted to use these registers. No lock out is imposed on the B-lines, but interrupt routines make no assumptions concerning the original contents of the B-lines and hence, at worst, erroneous use of interrupt B-lines by an object program can only result in erroneous functioning of that particular program. Switching of control to and from an interrupt routine is rapid, since no preservation of resetting of working registers is required.

The interrupt routines are designed to handle calls for action with the minimum delay and in the shortest time; the character-by-character transfers to and from peripheral equipments, for example, occur at high frequency and it is essential that the transfers be carried out with the minimum possible use of the central computer and within the time limit allowed by the peripheral equipment for filling or emptying the buffer. Since several interrupt flip-flops can become set simultaneously, but cannot be acted upon while another interrupt routine is still in progress, it is essential that a short time limit be observed by each interrupt routine. The majority of calls for interrupt routines involve only a few instructions, such as the transfer of a character, stepping of counts, etc., and on conclusion the interrupt routine returns to the former control, either main or extracode. On some occasions, however, longer sequences are required; for example, on completion of the input of a paper tape or deck of cards, routines must be entered to deal with the characters collected in the store, writing them to magnetic tape where appropriate, decoding and listing titles and so on. In such cases, the interrupt routine initiates a routine to be obeyed under extracode control, known as a supervisor extracode routine.

Supervisor Extracode Routines

Supervisor extracode routines (S.E.R.'s) form the principal "branches" of the supervisor program. They are activated either by interrupt routines or by extracode instructions occurring in an object program. They are protected from interference by object programs by using subsidiary store as working space, together with areas of core and drum store which are locked out in the usual way whilst an object program is being executed (see Section 3). They operate under extracode control, the extracode control register of any current object program being preserved and subsequently restored. Like the interrupt routines, they use private B-lines, in this case B-lines 100 to 110 inclusive; if any other working registers are required, the supervisory routines themselves preserve and subsequently restore the contents of such registers. The S.E.R.'s thus apply mutual protection between themselves and an object program.

These branches of the supervisor program may be activated at random intervals. They can moreover be interrupted by interrupt routines, which may in turn initiate other S.E.R.'s. It is thus possible for several S.E.R.'s to be activated at the same time, in the same way as it is possible for several interrupt flip-flops to be set at the same time. Although several S.E.R.'s may be activated, obviously not more than one can be obeyed at any one moment; the rest are either halted or held awaiting execution. This matter is organized by a part of the supervisor called the "co-ordinator routine" which is held in fixed store. Activation of an S.E.R. always occurs via the co-ordinator routine, which arranges

that any S.E.R. in progress is not interrupted by other S.E.R.'s. As these are activated, they are recorded in subsidary store in lists and an entry is extracted from one of these lists whenever an S.E.R. ends or halts itself. Once started, an S.E.R. is always allowed to continue if it can; a high priority S.E.R. does not "interrupt" a low priority S.E.R. but is entered only on conclusion or halting of the current S.E.R. The co-ordinator has the role of the program equivalent of the "inhibit interrupt flip-flop", the lists of activated S.E.R.'s being the equivalent of the setting of several interrupt flip-flops. The two major differences are that no time limit is placed on an S.E.R., and that an S.E.R. may halt itself for various reasons; this is in contrast to interrupt routines, which observe a time limit and are never halted.

In order that the activity of each branch of the computing system be maintained at the highest possible level, the S.E.R.'s awaiting execution are recorded in four distinct lists. Within each list, the routines are obeyed in the order in which they were activated, but the lists are assigned priorities, so that the top priority list is emptied before entries are extracted from the next list. The top priority list holds routines initiated by completion of drum transfers, and also routines entered as a result of computer failures such as core store parity. The second list holds routines arising from magnetic tape interruptions and the third holds routines arising from peripheral interruptions. The lowest priority list contains one entry for each object program currently under execution, and entry to an S.E.R. through an extracode instruction in an object program is recorded in this list. On completion of an S.E.R., the co-ordinator routine selects for execution the first activated S.E.R. in the highest priority list.

The central computer is not necessarily fully occupied during the course of an S.E.R. The routine may, for example, require the transfer of a block of information from the drum to the core store, in which case it is halted until the drum transfer is completed. Furthermore, the queue of requests for drum transfers (see section 3), is maintained in the subsidiary store, may be full, in which case the S.E.R. making the request must be halted. When an S.E.R. is halted for this or similar reasons, it is returned to the relevant list as halted, and the next activated S.E.R. is entered by the co-ordinator routine. Before an S.E.R. is halted, a restart point is specified. A halted routine is made free to proceed when the cause of the halt has been removed—for example, by the S.E.R. which controls drum transfers and the extraction of entries from the drum queue. The S.E.R. lists can therefore hold at any one time routines awaiting execution and halted routines; interrupt routines are written in such a way that the number of such S.E.R.'s activated at any one time is limited to one per object program, and one or two per interrupt flip-flop, depending upon the particular features of each interrupt routine. When an S.E.R. is finally concluded, as distinct from halted, it is removed from the S.E.R. lists and becomes dormant again.

Although S.E.R.'s originate in many cases as routines to control peripheral equipment, magnetic tapes and drums, it should not be supposed that this is the sole function of these routines. Entrances to S.E.R.'s from interrupt routines or from extracode instructions in an object program initiate routines which control the entire operation of the computing system, including the transfer of information between store and peripherals, communication with the operators and engineers, the initiation, termination and, where necessary, monitoring of object programs, the monitoring of central computer and peripheral failures, the execution of test programs and the accumulation of logging information. Each branch of supervisory activity is composed of a series of S.E.R.'s, each one activated by an object program or an interrupt routine and terminated usually by initiating a peripheral or magnetic tape transfer or by changing the status of an S.E.R. list or object program list. The most frequently used routines are held in the fixed store; routines required less frequently are held on the magnetic drum and are transferred to core store when required. Supervisor routines in core and drum store are protected from interference by object programs by use of hardware lock-out and the basic store organization routines in the fixed store.

Object Programs

The function of all supervisor activity is, of course, to organize the progress of problems through the computer with the minimum possible delay. Object programs are initiated by S.E.R.'s, which insert them into the object program list; they are subsequently entered

by the co-ordinator routine effectively as branches of lower priority than any S.E.R. Although object programs are logically subprograms of the supervisor, they may function for long periods using the computer facilities to the full without reference to the supervisor. For this reason, the supervisor program may be regarded as normally dormant, activated and using the central computer for only a small proportion of the available time.

In order to allow object programs to function with the minimum of program supervision, they are not permitted to use extracode control or interrupt control directly, enabling protection of main programs and supervisor programs to be enforced by hardware. Object programs use the main control register, B127, and are therefore forbidden access to the V-store and subsidiary store. Reference to either of these stores causes the setting of an interrupt flip-flop and hence entrance to the supervisor program.

Access to private stores is only obtained indirectly by use of extracode functions, which switch the program to extracode control and enter one of a possible maximum of 512 routines in the fixed store. These extracode routines form simple extensions of the basic order code, and also provide specific entry to supervisor routines to control the transfer of information to and from the core store and to carry out necessary organization. Such specific entrances to the supervisor program maintain complete protection of the object programs. Protection of magnetic tapes and peripheral input and output data is obtained by the use, in extracode functions, of logical tape and data numbers which the supervisor identifies within each program with the titles of the tapes or information. Blocks of core and drum store are protected by hardware and by the supervisor routines in fixed store as described in Section 3.

An object program is halted (by S.E.R.'s) whenever access is required to a block of information not immediately available in the core store. The block may be on the drums, in which case a drum transfer routine is entered, or it may be involved in a magnetic tape transfer. In both cases the program is halted until the block becomes available in core store. In the case of information involved in peripheral transfers, such as input data or output results, the supervisor buffers the information in core and drum store, and "direct" control of a peripheral equipment by an object program is not allowed. In this way, immobilization of large sections of store whilst a program awaits a peripheral transfer can be avoided. A program may however call directly for transfers involving drums or magnetic tapes by use of extracode functions, which cause entrance to the relevant supervisor routines. Queues of instructions are held in subsidiary store by these routines, in order to allow the object program to continue and to achieve the fullest possible overlap between tape and drum transfers and the execution of an object program.

While one program is halted, awaiting completion of a magnetic tape transfer for instance, the co-ordinator routine switches control to the next program in the object program list which is free to proceed. In order to maintain full protection, it is necessary to preserve and recover the contents of working registers common to all programs such as the B-lines, accumulator, and control registers, and to protect blocks in use in core store. The S.E.R. to perform this switching from one object program to another occupies the central computer for around 750 + 12p μsecs, where p is the number of pages, or 512 word blocks in core store. On the Manchester University Atlas, which has 32 pages of core store, the computing time for the round trip to switch from one program to another and to return subsequently is around 2.5 m.secs. This is in contrast to the time of around 60 μsecs. to enter and return from an S.E.R. and even less to switch to and from an interrupt routine. It is therefore obvious that the most efficient method of obtaining the maximum overlap between input and output, magnetic tape transfers, and computing is to reduce to a minimum the number of changes between object programs and to utilize to the full the rapid switching to and from interrupt and supervisor routines. The method of achieving this in practice is described in Section 6.

Compilation of programs is treated by the supervisor as a special case of the execution of an object program, the compiler comprising an object program which treats the source language program as input data. Special facilities are allowed to compilers in order that their allocation of storage space may be increased as need arises, and to allow exit to the supervisor before the execution of

a problem or the recording of a compiled object program.

Error Conditions

In addition to programmed entrances to the supervisor, entrance may also be made in the event of certain detectable errors arising during the course of execution of a problem. A variety of program faults may occur and be detected by hardware, by programmed checks in extracodes, and in the supervisor. Hardware causes entry to the supervisor by the setting of interrupt flip-flops in the event of overflow of the accumulator, use of an unassigned instruction, and reference to the subsidiary store or V-store. Extracode routines detect errors in the range of the argument in square root, logarithm, and arcsin instructions. In the extracodes referring to peripheral equipment or magnetic tapes, a check is included that the logical number of the equipment has been previously defined. In extracodes for data translation, errors in the data may be detected. The supervisor detects errors in connection with the use of the store. All problems must supply information to the supervisor on the amount of store required, the amount of output, and the expected duration of execution. This information is supplied before the program is compiled, or may be deduced after compilation. The supervisor maintains a record of store blocks used, and can prevent the program exceeding the preset limit. In addition, an interrupt flip-flop is set by a clock at intervals of 0.1 secs, and another flip-flop is set whenever 1024 instructions have been obeyed using main or extracode control. These cause entrances to the supervisor which enable a program to be "monitored" to ensure that the preset time limit has not expired, and which are also instrumental in initiating routines to carry out regular timed operations such as logging of computer performance and initiation of routine test programs.

The action taken by the supervisor when a program "error" is detected depends upon the conditions previously set up by the program. Certain errors may be individually trapped, causing return of control to a preset address; a private monitor sequence may be entered if required enabling a program or a compiler to obtain diagnostic printing; failing specification of these actions, some information is printed by the supervisor and the program is suspended, and usually dumped to magnetic tape to allow storage space for another program.

The following sections describe in detail the action of certain supervisor routines, namely, those controlling drums, magnetic tapes, and peripheral equipment and those controlling the flow of information in the computer.

3. STORE ORGANIZATION

Indirect addressing and the One-Level Store

The core store of Atlas is provided with a form of indirect addressing which enables the supervisor to re-allocate areas of store and to alter their physical addresses, and which is also used to implement automatic drum transfers. With each page, or 512 word block, of core store there is associated a "page address register" which contains the most significant address bits of the block of information contained in the page. Every time access is required to a word of information in the core store, the page containing the word is located by hardware. This tests for equivalence between the requested "block address", or most significant address bits, and the contents of each of the page address registers in parallel. Failure to find equivalence results in a "non-equivalence" interruption. The page address registers are themselves addressable in the V-store and can thus be set appropriately by the supervisor whenever information is transferred to or from core store.

One of the most important consequences of this arrangement is that it enables the supervisor to implement automatic drum transfers. The address in an instruction refers to the combined core and drum store of the computer, and the supervisor records in subsidiary store the location of each block of information; only one copy of each block is kept, and the location is either a page of core store or a sector of the drum store. At any moment, only some of the blocks comprising a particular program may be in the core store and if only these blocks are required, the program can run at full speed. When a block is called for which is not in the core store, a non-equivalence interruption occurs, which enters the supervisor to transfer the new block from a sector of the drum to a

page of the core store. During this operation the program that was interrupted is halted by the supervisor.

The block directory in subsidiary store contains one entry for each block in the combined core and drum store. It is divided into areas for each object program which is in the store; a separate program directory defines the area of the block directory occupied by each program. The size of this area, or the number of blocks used by a program, is specified before the program is obeyed in the job description (see Section 6). The entry for block n contains the block number n together with the number of the page or sector occupied by the block, and, if possible, is made in the n^{th} position in the area; otherwise the area is filled working backwards from the end. In this way, blocks used by different object program are always kept distinct, regardless of the addresses that are used in each program. A program addresses the combined "one-level store"* and the supervisor transfers blocks of information between the core and drum store as required; the physical location of each block of information is not specified by the program, but is controlled by the supervisor.

There are occasions when an object program must be prevented from obtaining access to a page of the core store, such as one involved in a drum or tape transfer. To ensure complete protection of such pages, an additional bit, known as a lock out bit, is provided with each page address register. This prevents access to that page by the central computer, except when on interrupt control, and any reference to the page causes a non-equivalence interruption. By setting and resetting the lock out bits, the supervisor has complete control over the use of core store; it can allow independent object programs to share the core store, it can reserve pages for peripheral transfers and can itself use parts of the core store occasionally for routines or working space, without any risk of interference. This is done by arranging that, whenever control is returned to an object program, pages that are not available to it are locked out.

*A paper on the one-level store has been written, and will, it is hoped, be published by the Institute of Radio Engineers.

A block of information forming part of an object program may also be locked out from use by that program because an operation on that information, controlled by the supervisor, is not complete. A drum, magnetic tape, or peripheral equipment transfer involving this block may have been requested. The reason for the lock out of such a block is recorded in the block directory, and if the block is in the core store, the lock out digit is also set. If reference is made to such a block by the object program, a non-equivalence interruption occurs and a supervisor extracode routine halts the program. This S.E.R. is restarted by the co-ordinator routine when the block becomes "unlocked", and the object program is re-entered when the block is available in core store.

The Drum Transfer Routine

The drum transfer routine is a group of S.E.R.s which are concerned with organizing drum transfers, and updating page address registers and the block directory. Once initiated, the transfer of a complete block to or from the drum proceeds under hardware control; the drum transfer routine initiates the transfer and identifies the required drum sector by setting appropriate bits in the V-store. It also identifies the core store page involved by setting a particular "dummy" block address, recognized by the drum control hardware, in the page address register; at the same time, this page is locked out to prevent interference from object programs while the transfer is in progress.

On completion of a transfer, an interruption occurs which enters the drum transfer routine. The routine can also be entered from the non-equivalence interrupt routine, which detects the number of the block requested but not found in the page address registers. Finally, the drum transfer routine can be activated by other parts of the supervisor which require drum transfers, and by extracode instructions which provide a means whereby object programs can if they wish exert some control over the movement of blocks to and from the drum store. A queue of requests for drum transfers, which can hold up to 64 requests, is stored in the subsidiary store; when the drum transfer routine is entered on completion of a transfer, the next transfer in the queue is initiated.

Whenever the supervisor wishes to enter another request for a drum transfer, three possible situations arise. Firstly, the queue is empty and the drum transfer can be started immediately. Secondly, the queue is already partly filled and the request is entered in the next position in the queue. Thirdly, the queue is full. In this case the routine making the request is halted by the co-ordinator routine, and is resumed when the queue can receive another entry. In the first two cases the supervisor routine is concluded when the request reaches the queue.

A non-equivalence interruption, which implies a drum transfer is required, is dealt with as follows. The core store is arranged to always hold an empty page with no useful information in it, and when required, a transfer of a block of information from the drum to this empty page is initiated. While this drum transfer is proceeding, preparation is made to write up the contents of another page of core store to the drum to maintain an empty page. The choice of this page is the task of the "learning program" which keeps details of the use made of blocks of information. This learning program will be described in detail elsewhere; it predicts the page which will not be required for the largest time, and is arranged with a feed-back so that if it writes up a block which is almost immediately required again, it only does this once. The number of the chosen page
the drum queue entry is converted to a request to write this page to the drum. This supervisor routine is now concluded and returns control to the co-ordinator routine.

When the drum transfer is completed, the drum transfer routine is again entered. This updates the block directory and page address register, makes the object program free to proceed and initiates the next drum request, which is to write the chosen page to the drum. This routine is now concluded and the co-ordinator is re-entered. The supervisor is finally entered when the write to drum transfer is complete. The block directory is updated, a note is made of the empty page, and the next drum request is initiated.

The Use of Main Store by the Supervisor

Some routines of the supervisor are obeyed in the main store, and these and others use working space in the main store. Since the supervisor is entered without a complete program change, special care must be taken to keep these blocks of store distinct and protected from interference. The active supervisor blocks of main store are recorded in the area for program 0 in the block directory. There are also some blocks of the supervisor program which are stored permanently on the drum; when one of these permanent blocks is required, it is duplicated to form an active block of the supervisor or, as in the case of a compiler, to become part of an object program.

Of the possible 2048 block numbers, 256 are "reserved" block numbers which are used exclusively by the supervisor and are not available to object program; object program are restricted to using the remaining "non-reserved" block numbers. Blocks with reserved block numbers may be used in the core store at any time by the supervisor, and the co-ordinator routine locks out these pages of core store before returning control to an object program. The supervisor also uses some blocks having non-reserved block numbers to keep a record of sequence of blocks of information such as input and output streams. When a non-reserved supervisor block is called to the core store, the page address register is not set, since there may be a block of an object program which has the same block number already in the core store. Instead, the page address register is set to a fixed reserved block number while it is in use, and is cleared and locked out before control passes to another routine.

Not all the reserved block numbers are available to the supervisor for general use, since certain block numbers are temporarily used when drum, tape, and peripheral transfers are proceeding. These block numbers do not appear in the block directory. For example, when a magnetic tape transfer is taking place, the page of core store is temporarily given a block number which is recognized by the hardware associated with that tape channel. When the transfer is complete, the appropriate block number is restored. During a peripheral transfer, and also on other occasions, it is necessary that a block should be retained in the core store and should not be transferred to the drum. The relevant page of core store is "locked down" by setting a digit in the subsidiary store; the learning program never selects for transfer to the drum a page for which this lock-down digit is set.

4. MAGNETIC TAPE SUPERVISOR ROUTINES

The Magnetic Tape Facilities

The tape mechanism used on Atlas is the Ampex TM2 (improved FR 300) using one inch wide magnetic tape. There are sixteen tracks across the tape - twelve information tracks, two clock tracks, and two tracks used for reference purposes. The tapes are used in a fixed-block, pre-addressed mode. Information is stored on tape in blocks of 512 forty-eight bit words, together with a twenty-four bit checksum with end around carry. Each block is preceded by a block address and block market and terminated by a block marker; the leading block address is sequential along the tape, and what is effectively the trailing block address is always zero. Tapes are tested and pre-addressed by special routines before being put into use, and the fixed position of the addresses permits selective overwriting and simple omission of faulty patches on the tape. Blocks can be read when the tape is moving either in the forward or reverse direction, but writing is only possible when the tape is moving forward. The double read and write head is used to check read when writing on the tape. When not operating the tape stops with the read head midway between blocks.

Atlas may control a maximum of 32 magnetic tape mechanisms. Each mechanism is connected to the central computer via one of eight channels, all of which can operate simultaneously, each controlling one read, write or positioning operation. It is possible for each tape mechanism to be attached to either one of a pair of channels, the switching being under the control of supervisory program through digits in the V-store. Fast wind and rewind operations are autonomous and only need the channel to initiate and, if required, terminate them. Transfer of a 512-word block of information between core store and tape is effected via a one-word buffer, the central computer hesitating for about 1/2 μsec, on average, each time a word is transferred to or from the core store. During a transfer the page of core store is given a particular reserved block number and the contents of the page address register are restored at the end of the transfer.

Supervisory programs are only entered when the block addresses are read before and after each block, and when the tape stops. As each block address is read, it is recorded in the V-store and an interrupt flip-flop is set, causing entrance to the block address interrupt routine.

The Block Address Interrupt Routine

This routine is responsible for initiating and checking the transfer of a single block between tape and core store, and searching along the tape for a specified block address. Digits are available in the V-store to control the speed and direction of motion of the tape and the starting and termination of read or write transfers. The block addresses are checked throughout and, in particular, a write transfer is not started until the leading block address of the tape block involved has been read and checked. Hardware checking is provided on all transfers, and is acted upon by supervisor routines. A 24-bit check sum is formed and checked as each block is transferred to or from a tape, and a digit is set in the V-store if any failure is detected. Similarly a digit is set in the event of failure to transfer a full block of 512 words. These digits are tested by the block address interrupt routine on the conclusion of each transfer. Parity failure either on reading from core store or on formation of the parity during a transfer to core store causes the setting of interrupt flip-flops. If a tape fails to stop, this is detected by the block address interrupt routine as a particular case of block address failure. Failure to enter the block address routine (for example, through failure to read block markers) is detected by the timed interrupt routine at intervals of 100 milliseconds. Finally, failures of the tape mechanism, such as vacuum failure, set a separate interrupt flip-flop. The detection of any of these errors causes entry to tape monitor routines, whose action will be described later.

Organization of Tape Operations

Magnetic tape operations are initiated by entrance to the tape supervisor routines in the fixed store from extracode instructions in an object program or, if the supervisor requires the tape operation for its own purposes, from supervisor extracode routines. From a table in subsidiary store, the logical tape number

used in a program is converted to the actual mechanism number, and the tape "order" is entered to a queue of such orders, in subsidiary store, awaiting execution. A tape order may consist of the transfer of several blocks and any store blocks involved are "locked out" to prevent subsequent use before completion of the transfers; if any block is already involved in a transfer, the program initiating the request is halted. Similarly, the program is halted if the queue of tape instructions is already full. If the channels to which the deck can be connected are already occupied in a transfer or positioning, the tape supervisor returns control to the object program, which is then free to proceed. A program may thus request a number of tape transfers without being halted, allowing virtually the maximum possible overlap between the central computer and the tape mechanisms during execution of a program. Should a channel be available at the time a tape order is entered to the queue, the order is initiated at once by writing appropriate digits to the V-store, and by writing reserved tape transfer block numbers to the appropriate page address registers if the order involves a read or write transfer. The tape supervisor then returns control to the object program or supervisor routine.

One composite queue of tape orders is used for orders relating to all tape mechanisms and orders are extracted from the queue by S.E.R.'s entered from the block address interrupt routine. On reading the penultimate block address involved in an operation (for example, the last leading block address in a forward transfer) the next operation for the channel is located, and if it involves the same mechanism as the current order, and tape motion in the same direction, the operation is "prepared" by calling any store block involved to core store. On reading the final block address and successfully concluding checks, the block address interrupt routine initiates the next operation immediately if one has been prepared, thus avoiding stopping the tape if possible. If no operation has been prepared, the interrupt routine stops the tape by setting a digit in the V-store, and a further "block address interruption" occurs when the tape is stopped and the channel can accept further orders. This interruption enters an S.E.R. which extracts the next order for the channel from the tape queue, and the cycle of events is repeated until no further order for this channel remains. As each transfer is concluded, any object program halted through reference to the store block is made free to proceed.

An exception to the above process is when a long movement (over 200 blocks) or a rewind is required. In this case, the movement is carried out at fast speed, with block address interruptions inhibited, and the channel may meanwhile be used to control another tape mechanism. The long movement is terminated by checking the elapsed time and at the appropriate moment, entering the tape supervisor from the timed interrupt routine. The mechanism is then brought back "on channel" and the speed is returned to normal. When reading of block addresses is correctly resumed, the search is continued in the normal manner.

The Title Block

The first block on each magnetic tape is reserved for use by the supervisor, and access to information in this block by an object program is through special instructions only. This block contains the title of the tape, or an indication that the tape is free. When magnetic tapes are required by the supervisor or by an object program, the supervisor prints instructions to the operator to load the named tape and to engage the mechanism on which it is loaded. The engage button of each mechanism (see section 5) is attached to a digit in the V-store, and these digits are scanned by the supervisor every one second. When a change to "engaged" status has been detected, the tape supervisor is entered to read the first block from the tape. The title is then checked against the expected title. In this way, the presence of the correct tape is verified, and furthermore the tape bearing the title becomes associated with a particular mechanism. Since the programmer assigns a logical tape number to the tape bearing a given title, this logical tape number used in extracode instructions can be converted by the supervisor to the actual mechanism number. Other supervisory information is included in the first block on each tape, including a system tape number and the number of blocks on the tape. Special supervisory routines allow Atlas to read tapes produced on the Ferranti Orion computer, which used the same tape mechanisms but can write blocks of varying lengths on the tape. These tapes are distinguished on Atlas by a marker written in the title block.

Magnetic Tape Failures

All failures detected by the interrupt routines cause the block address interrupt routine to stop the tape at the end of the current block when possible, and then to enter tape monitor supervisory routines; if the tape cannot be stopped, it is disengaged and the tape monitor routines entered. These routines are S.E.R.'s designed to minimize the immediate effect on the central computer of isolated errors in the tape system, to inform maintenance engineers of any faults, and to diagnose as far as possible the source of a failure. As an example of the actions taken by monitor routines, suppose a check sum failure has been detected while reading a block from tape to core store. The tape monitor routines make up to two further attempts to read the block; if either succeeds, the normal tape supervisor is re-entered after informing the engineers. Repeated failure may be caused by the tape or the tape mechanism; to distinguish these, the tape is rewound and an attempt is made to read the first block. If this is successful, a tape error is indicated, and an attempt is made to read the suspect block with reduced bias level. Failure causes the mechanism to be disengaged and the program using the tape to be suspended. If the "recover read" is successful, the tape is copied to a free tape and the operators instructed to re-address the faulty tape, omitting the particular block which failed. If on rewinding the tape, the first block cannot be read successfully, failure in the tape mechanism is suspected and the operator is instructed to remount the tape on another mechanism. Other faults are monitored in a similar manner, and throughout, the operator and engineers are informed of any detected faults. Provision is made for the program using the tape to "trap" persistent tape errors and thereby to take action suitable to the particular problem, which may be more straight-forward and efficient than the standard supervisory action.

Addressing of new tapes and re-addressing of faulty tapes are carried out on the computer by supervisory routines called in by the operator. A tape mechanism is switched to "addressing mode", which prohibits transfers to and from the core store, permits writing from the computer to the reference tracks and to the block addresses on tape, and activates a timing mechanism to space the block addresses. When a new tape is addressed, addresses are written sequentially along the tape and the area between leading and trailing block addresses is checked by writing ones to all digit positions and detecting failures on reading back. Any block causing failure is erased and the tape spaced suitably. On completion, a special block address is written to indicate "end of tape" and the entire tape is then checked by reading backwards. Any failures cause entry to the re-addressing routine. Finally, the tape mechanism is returned to "normal" mode, a little block is written containing the number of blocks on tape, a tape number, and the title "Free", and the tape is made available for use. A tape containing faulty blocks is re-addressed, omitting such blocks, by entry to the re-addressing routine with a list of faulty blocks; the faulty blocks are erased and the remaining blocks are re-labelled sequentially, the tape being checked as when addressing a new tape.

5. PERIPHERAL EQUIPMENT

Peripheral Interruptions

As mentioned in the Introduction, a large number and variety of peripheral equipments may be attached to Atlas. However, the amount of electronics associated with each equipment is kept to a minimum, and use is made of the high computing speed and interruption facilities of Atlas to provide control of these equipments and large scale buffering.

Thus the paper tape readers, which operate at 300 characters per second, set an interrupt flip-flop whenever a new character appears. (Characters may be either 5 or 7 bits depending on which of two alternative widths of tape is being read.) Similarly the paper tape punches, and the teleprinters which print information for the computer operators, cause an interruption whenever they are ready to receive a new character; these equipments operate at 110 and 10 characters per second respectively.

The card readers read 600 cards per minute, column by column, and interrupt the computer for every column. The card punches at 100 cards per minute, punch by rows and interrupt for each row.

The printers, which have 120 print wheels bearing 50 different characters, cause an

interruption as each character approaches the printing position so that the computer may prime the hammers for those wheels where this character is to be printed. There are therefore 50 interruptions per revolution, or one every 1-1/2 millisecs.

All the information received from, or sent to, these peripheral equipments does so via particular digit positions in the V-store. For example, there are 7 such bits for each tape reader, and 120 for each printer, together with a few more bits for control signals.

The majority of interruptions can be dealt with simply by the interrupt routine for the particular type of equipment. Thus the paper tape reader interrupt routine normally has merely to refer to a table of characters to apply code conversion and parity checks, and to detect terminating characters and if all is well to store the new character in the next position in the store.

The interrupt routines for the printers and card punches are not expected to do the conversion from character coding to row binary; this is done by an S.E.R. before the card or line of print is commenced. The card routines are however complicated by the check reading stations; punching is checked one card cycle afterwards, and reading is checked 3 columns later. The interrupt routines apply these checks, and in the event of failure a monitor S.E.R. is entered.

The printer interrupt routine counts its interruptions to identify the character currently being printed; a check is provided once each revolution when a datum mark on the printer shaft causes a signal bit to appear in the V-store. This counting is maintained during the paper feed between lines, which occupies several milliseconds.

Attention by Operators

Whenever equipment needs attention it is "disengaged" from the computer. In this state, which is indicated by a light on the equipment and a corresponding bit in the V-store, it automatically stops and cannot be started by the computer.

The operator may engage or disengage an equipment by means of two buttons so labelled. The equipment may also be disengaged by the computer by writing to the appropriate V-store bit, but the computer cannot engage it.

The "engage" and "disengage" buttons do not themselves cause interruptions of the central computer. Instead, the "engaged" bits in the V-store are examined every second (this routine is activated by the clock interruption) and any change activates the appropriate S.E.R. Disengaging a device does not inhibit its interruptions, so that if the operator disengages a card machine in mid-cycle to replenish the magazine or to empty the stacker, the cycle is completed correctly.

There are also other special controls for particular equipments, e.g., a run-out key on card machines, and a 5/7-hole tape width selector switch on punched tape readers.

Most devices have detectors that indicate when cards or paper are exhausted or running low. These correspond to bits in the V-store that are read by the appropriate S.E.R. The paper tape readers however have no such detector, and the unlikely event of a punched tape passing completely through a reader (due to the absence of terminating characters) appears to the computer merely as a failure to encounter a further character within the normal time interval. This condition is detected by the one-second routine.

Store Organization of Input and Output Information

In general, input information is converted to a standard 6-bit internal character code by the interrupt routine concerned, and placed in the store 8 characters to a word. (An exception to this occurs in the case of card readers when they are reading cards not punched in a standard code, in which case the 12 bits from one column are simply copied into the store and occupy two character positions. A similar case is 7-hole punched tape, when this is used to convey 7 information bits without a parity check. Such information is distinguished by warning characters, both on the input medium and in the store.)

A certain amount of supervisor working space in the core store is set aside to receive this information from the interrupt routines, and is subdivided between the various input peripherals. The amount of this space depends on the number and type of peripherals attached; the first two Atlases will normally use one block (512 words). This block will be locked down in a page of the core store whenever any input peripheral is operating (i.e., most of the time).

As each input equipment fills its share of this block, the information is copied by an S.E.R. into another block devoted exclusively to that equipment. These copying operations are sufficiently rare that the latter block need not remain in core store in the meantime; in fact it is subject to the same treatment as object programs by the drum transfer routine, and may well be put onto a drum and brought back again for the next copying operation. Thus only one page of core store is used full time during input operations, but nevertheless each input stream finds its way into a separate set of blocks in the store.

The page that is shared between input peripherals is subdivided in such a way as to minimize the number of occasions on which information must be copied to other blocks; it turns out that the space for each equipment needs to be roughly proportional to the square root of its information rate.

Similarly, information intended for output is placed in a common output page, subdivided for the various output devices, and is taken from there by the interrupt routines as required. The interrupt routines for teleprinters and tape punches do the necessary conversion from the internal character code used by the device. As soon as the information for a particular device is exhausted, an S.E.R. is activated to copy fresh information into the common output page. Again, the page is subdivided roughly in proportion to the square roots of the information rates.

For card punches and printers, whose interrupt routines require their information arranged in rows of bits, a further stage of translation is necessary. In these cases, on completing a card or line, internal 6-bit characters are converted by an S.E.R. into a card or line image also in the output block. In fact, the desirable amount of working space for output buffering somewhat exceeds one block, and the spare capacity of the subsidiary store will be utilized to augment it.

6. THE OPERATING SYSTEM

The following is a synopsis of work explained in detail in a paper the Computer Journal.

Input

The fast computing speed of Atlas and the use of multiple input and output peripheral equipments enable the computer to handle a large quantity and variety of problems. These will range from small jobs for which there is no data outside the program itself, to large jobs requiring several batches of data, possibly arriving on different media. Other input items may consist of amendments to programs, or requests to execute programs already supplied. Several such items may be submitted together on one deck of cards or length of punched tape. All must be properly identified for the computer.

To systematize this identification task, the concept of a "document" has been introduced. A document is a self-contained section of input information, presented to the computer consecutively through one input channel. Each document carries suitable identifying information (see below) and the supervisor keeps in the main store a list of the documents as they are accepted into the store by the input routines, and a list of jobs for which further documents are awaited.

A job may require several documents, and only when all these have been supplied can execution begin. The supervisor therefore checks the appearance of documents for each job; when they are complete the job scheduling routine is notified (see below)

Normally, the main core and drum store of the computer is unlikely to suffice to hold all the documents that are waiting to be used. The blocks of input information are therefore copied, as they are received, onto a magnetic tape belonging to the supervisor, called the "system input tape." Hence, if it becomes necessary for the supervisor to erase them from the main store, they can be recovered from the system input tape when the job is ready for execution.

The system input tape thus acts as a large scale buffer, and indeed it plays a similar part to that of the system input tape in more conventional systems. The differences here are that the tape is prepared by the computer itself instead of by off-line equipment, and that there is no tape-handling of manual supervision required after the input of the original documents - an important point in a system designed to handle many miscellaneous jobs.

This complete bufferage system for input documents is called the "input well." Documents awaiting further documents before they can be used are said to be in "input well A"; complete sets of documents for jobs from "input well B." Usually documents being

accepted into input well B must be read from the system input tape back into the main store so that they are ready for execution; often however they will already be in input well A in the main store, so that only an adjustment of the block directory is required.

One result of this arrangement is that the same tape is being used both to write input blocks, in a consecutive sequence, and to read back previously written blocks to recover particular documents as they are required. The tape will therefore make frequent scans over a few feet of tape, although it will gradually progress forwards. The lengths of these scans are related to the main store space occupied by input well A. For example, so long as the scans do not exceed about 80 feet (130 blocks) the waiting time for writing fresh blocks will remain less than the time for input of three blocks from a card reader, so that comparatively little main store space need be occupied by input well A. To ensure that scans are kept down to a reasonable limit, any documents left on the system input tape for so long that they are approaching the limit of the scannable area are copied to the system dump tape (see below). If the number of these becomes large, the computer operators are warned to reduce the supply of documents through the input peripherals.

Output

The central computer can produce output at a much greater rate than the peripheral equipments can receive it, and an "output well" is used in a manner analogous to the input well. This well uses a "system output tape" to provide bulk buffering.

Output for all output peripherals is put onto the same tape, arranged in sections that are subdivided so that the contents of a section will occupy all currently operating peripherals for the same length of time. Thus if, for example, a burst of output is generated for a particular peripheral, it is spaced out on the system output tape, leaving spare blocks to be filled in later with output for other peripherals (this is possible because Atlas uses pre-addressed tape). In this way, the recovery of information from the tape into "output well B" as required by the various peripherals merely involves reading complete sections from the tape.

Again, there is a limit to the amount of information that can usefully be buffered on the output tape, due to the time required to scan back and forth between writing and reading regions, and this limit depends on the space available in the main store for output well B. An S.E.R. keeps a check on the amount of information remaining in output well B for each equipment, and relates this to the present scan distance to decide when to start to move the tape back for the next reading operation. If the amount of output being generated by object programs becomes too great some of it is put instead on the dump tape (see blow) or a program is suspended.

The System Dump Tape

The system input and output tapes operate essentially as extensions of the main store of the computer. Broadly speaking, documents are fed into the computer, programs are executed, and output is produced. The fact that the input and output usually spends some time on magnetic tape is, in a sense, incidental. This input and output buffering is, however, a continuous and specialized requirement, so that a particular way of using these tapes has been developed and special S.E.R.'s have been written to control them.

When demands on storage exceed the capacity of the main store and input and output tapes, a separate magnetic tape, the system dump tape, is used to hold information not required immediately. This tape may be called into use for a variety of reasons. Execution of a problem may be suspended and the problem recorded temporarily on the dump tape if other problems are required to fill the output well, or alternatively if its own output cannot be accommodated in the output well. Also, as already described, the input and output wells can "overflow" to the system dump tape. This tape is not used in a systematic manner, but is used to deal with emergencies. However, the system is such that, if necessary, the system input and output tapes can be dispensed with, thereby reducing the input and output wells and increasing the load on the system dump tape. In an extreme case, the system dump tape itself can be dispensed with, implying a further reduction in the efficiency of the system.

Headings and Titles

Every input document is preceded by its identifying information, mentioned above.

This consists of two lines of printing, forming the heading and the title respectively.

The heading indicates which type of document follows. The most common headings are

COMPILER followed by the name of a program language, which means that the document is a program in the stated language;

DATA which means that the document is data required by an object program; and

JOB which means that the document is a request for the computer to execute a job, and gives some relevant facts about it.

The last type of document is called a "job description." It gives, for example, a list of all other documents required for the job, a list of output streams produced, any magnetic tapes required and upper limits to the storage space and computing time required. Many of these details are optional; for example if storage space and computing time are not quoted a standard allowance will be made.

For example, if a program operates on two data documents which it refers to as data 1 and data 2, the job description would contain:

```
INPUT

    1 followed by the title of data 1
    2 followed by the title of data 2
```

The program would appear in this list as data 0. Alternatively, a job description may be combined with a program, forming one composite document, and this will usually happen with small jobs.

Each output stream may be assigned to a particular peripheral or type of peripheral, or may be allowed to appear on any output equipment. The amount of output in each stream may also be specified. For example, a job description may include:

```
OUTPUT

    1 LINE PRINTER 20 BLOCKS
    2 CARDS
    3 ANY
```

Each magnetic tape used by a program is identified by a number within the program, and the job description contains a list of these numbers with the title that appears in block 0 of each tape to identify it; for example:

```
TAPE

    1 POTENTIAL FIELD CYLIND/204-
      TPU5.
```

If a new tape is required, a free tape must be loaded, which the program may then adopt and give a new title. This is indicated thus:

```
TAPE FREE

    2 MONTE CARLO RESULTS K49-
      REAC-OR4.
```

The loading of tapes by operators is requested by the supervisor acting on the information in job descriptions.

Finally, the end of a document is indicated by

```
* * *
```

and if this is also the end of the punched tape or deck of cards it is followed by the letter Z. On reading this the computer disengages the equipment.

Logging and Charging for Machine Time

As problems are completed, various items of information on the performance of the computing system are accumulated by the supervisor. Items such as the number of program changes and the number of drum transfers are accumulated and also, for each job, the number of instructions obeyed, the time spent on input and output, and the use made of magnetic tapes. These items are printed in batches to provide the operators with a record of computer performance, and they are also needed for assessing machine charges.

The method of calculating charges may well vary between different installations, but one desirable feature of any method is that the charge for running a program should not vary significantly from one run to another. One difficulty is that the number of drum transfers required in a program may vary considerably with the amount of core store which is being used at the same time for magnetic tape and peripheral transfers. One method of calculating the charge so as not to reflect this variation is to make no charge for drum transfers, but to base the charge for computing time on the number of instructions obeyed in a program. This, however, gives no incentive to a programmer to arrange a program so as to reduce its drum transfers, and more elaborate schemes may eventually be devised. The charge for using

peripherals for input and output can be calculated from the amount of input and output. For magnetic tapes, the charge can be based on the length of time for which the tape mechanism is engaged, allowance being made for the time when the program is free to proceed but is held up by a program of higher priority. All this information is made available to the S.E.R. responsible for the costing of jobs.

Methods of Using the Operating System

The normal method of operating the computer is for documents to be loaded on any peripheral equipment in any order, although usually related documents will be loaded around the same time. The titles and job descriptions enable the supervisor program to assemble and execute complete programs, and the output is distributed on all the available peripherals. Usually programs are compiled and executed in the same order as the input is completed, but the supervisor may vary this depending on the load on different parts of the system. For example, a problem requiring magnetic tape mechanisms which are already in use may be by-passed in favour of a problem using an idle output peripheral; a problem which computes for a long time may be temporarily suspended in order to increase the load on the output peripherals. By these and similar methods, the S.E.R. responsible for scheduling attempts to maintain the fullest possible activity of the output peripherals, the magnetic tape mechanisms and the central computer.

Documents may also be supplied to the computer from magnetic tapes; these tapes may be either previous system input tapes or library tapes or tapes on which "standard", frequently used, programs are stored. Such documents are regarded as forming part of Input Well B and are read into main store when required. An alternative method of operating may be to use the computer to copy documents to a "private" magnetic tape, rather than to use the system input tape, and at a later time to supply the computer with a succession of jobs from this tape. Similarly, output may be accumulated on a private magnetic tape and later passed through the computer to one or more peripheral equipments. Routines forming part of the supervisor are available to carry out such standard "copying operations.

Provision is also made for the chief operator to modify the system in various ways; for example, priority may be given to a particular job, or a peripheral equipment may be removed from general use and allocated a particular task. An "isolated" operating station may, for example, be established by reserving a particular output equipment for use by problems loaded on a particular input equipment.

7. Conclusion

The Atlas supervisor program is perhaps the most advanced example so far encountered of a program involving many parallel activities, all closely interconnected. Although a great deal of it has already been coded at the time of writing, there are still a few details to be thrashed out, and no doubt many changes will have to be made to suit conditions existing at various installations. The overall structure of the program is therefore of prime importance: only if this structure is adequate, sound and systematic will it be possible to complete the coding satisfactorily and to make the necessary changes as they arise.

The structure described in this paper, involving interrupt routines and "supervisory extracode routines" controlled by a coordinating routine, has proved eminently satisfactory as a basis for every supervisory task that has so far been envisaged, and it is expected that all variations that might be called for will fit into this structure.

There is no doubt that its success as a supervisory scheme for a very fast computer is due largely to certain features of the Atlas hardware, in particular the provisions for protecting programs from interference, and the page address registers. The way in which the latter permit information to be moved around at run time without the need to recompile programs gives the supervisor an entirely new degree of freedom. The possible uses of this facility have turned out to be so extensive that the task of utilizing such a large computer effectively without it seems by comparison to be almost impossible.

Acknowledgments

The work described in this paper forms part of the Atlas project by a joint team of Manchester University and Ferranti Ltd., whose permission to publish is acknowledged. The authors wish to express their appreciation of the assistance given by the other members of the Atlas team.

Ferranti Ltd

COMPUTER DEPARTMENT

Enquiries to

London Computer Centre, 68 NEWMAN STREET, LONDON, W.1

Telephone MUSeum 5040

and

21 PORTLAND PLACE, LONDON, W.1

Telephone LANgham 9211

Office, Works and Research Laboratories

WEST GORTON, MANCHESTER, 12. Telephone EASt 1301

Research Laboratories

WESTERN ROAD, BRACKNELL, BERKS

February 1962 Printed in England Reprint R 58

AN APPRAISAL OF THE ATLAS SUPERVISOR

D. MORRIS

F.H. SUMNER

MANCHESTER UNIVERSITY

M.T. WYLD

INTERNATIONAL COMPUTERS
AND TABULATORS LIMITED

This paper has been submitted to the 1967 National A.C.M. Conference and is for restricted circulation only.

ABSTRACT

The Atlas computer system at Manchester University has now been in full 3 shift operation for over three years.

This paper is a description of the operating system now in use with an indication of the differences from the system which was proposed in 1963. The automatic production of operating statistics is described and a detailed analysis of these is given in order to show the behaviour of the operating system under differing work loads. The various factors affecting the efficiency of the system are discussed and methods of improvement are suggested.

Department of Computer Science

MANCHESTER UNIVERSITY

OXFORD ROAD

MANCHESTER 13.

An Appraisal of The Atlas Supervisor

Introduction

This report presents the performance of the Supervisor System for the Atlas Computer at Manchester University, and describes some of the changes made as a result of our experience with the system. The machine is used jointly by I.C.T. Computing Service Divison and the University Computing Service (U.C.S.). The figures presented are derived from the U.C.S. use of the machine.

A retrospective appraisal of the system is appropriate at this time because the ideas of multiprogramming and multiple on-line peripherals pioneered by Atlas now feature on most current and projected machines.

Before discussing its performance and throughput we give an outline of the system and describe in some detail its main sections. Although the ideas have been previously presented (1,2,3,4) the system that has evolved from them has deviated in some cases.

Logically the system is made up of several distinct parts which communicate through small well defined interfaces. Figure 1 is a schematic representation of the system, however, the actual implementation is more complicated than this superficial description. It can be seen from Fig. 1 that the path of a normal job is from the input peripherals via the input supervisor into the input well which is at present magnetic tape. When the complete job has been input the input supervisor makes an entry in the job list. This job entry contains sufficient information to allow the scheduler to decide, by means of fairly simple algorithms, when to start the job. The scheduler then passes the job entry onto the job assembler which requests the loading of any magnetic tapes required by the job, and transfers the relevant input files and compiler from the input well and the magnetic tape holding the compilers.

On this compiler tape there are 25 'compilers' which occupy about 250K words of program (no distinction is made between compilers, assemblers and loaders.) The job is now ready for execution and a modified job entry is passed on to the central executive. This job entry defines where the relevant compiler and input are positioned in store and on which decks the required magnetic tapes are mounted. The central executive organizes fully the execution of the job. The output from the job passes through the output assembler which buffers it into the output well, again this is stored on magnetic tape. On completion of a job, an entry is made for each of its output files on the output list. The actual output will then be organised by the output supervisor as soon as the required peripheral is available.

Thus, although the main I/O peripherals are on-line to the machine they are not on-line to the jobs, and the I/O is buffered as in batch processing. This does not mean that the jobs are executed in the order in which they are presented. Also special peripherals can be on-line to the jobs in the same way that magnetic tape decks are on-line. The Manchester Installation has an x-ray diffractometer and an audio I/O device in this category.

An Appraisal of The Atlas Supervisor

The Input Supervisor

This part of the supervisor controls the input peripherals and the passage of jobs through them into the input well. Input is delivered character by character by the input peripherals. It is parity checked, converted to a common internal code and packed four 6 bit characters per 24 bit word (i.e. half word). Carriage control characters (e.g. newline, carriage return, line feed, end of card) are regarded as end of record characters and are treated specially. The characters up to and including the carriage control character comprise a record. They are packed as described above in consecutive half words with one additional half word preceding them giving the number of characters in the record. If the last half word is not full it is filled out with zero characters and the next record starts in the next half word. A block of 1024 half words comprise a block of input and as each block is filled it is transferred to the input well. The first sixteen half words of each block are used for control purposes and the first of these is a link which gives the address of the previous block of the file in the input well (the link in the 1st block is zero). The address of the last block of a file is remembered as the file address.

A job may involve up to sixteen input files and their titles must uniquely identify them. The input section of the job description (1,2) lists the titles of all files associated with the job and assigns a stream number to each so that the program may specify dynamically from which file the input is to be taken. Only when all the files specified in the job description have been input is an entry made for the job in the job list. A job entry contains a summary of the character of the job (e.g. the number of magnetic tapes required, the compiler required, the principle output device etc.) which is used for scheduling purposes, and the address in the input well of the processed job description.

This processed job description contains the addresses of the associated files as well as the encoded form of the job description itself.

An Appraisal of The Atlas Supervisor

Faults which occur at the input stage can be particularly frustrating to the user. Since there is no formal output he relys entirely on information relayed back to him via the operators. Incorrect statements in job descriptions and parity faults on the tape cause the input supervisor to reject the job and print a message to the operator. Discrepencies between the titles of files and the input section of the job description are more elusive since the job is never entered on the job list and the operator may not realise this.

Large input files or files which have to be processed many times can be recorded on a private magnetic tape and subsequently read from this. A special heading, specifying a tape title and starting block, causes the input supervisor to route the following input file onto the private tape. The format of input on a private tape is the same as that in the input well but the blocks are linked forwards. On completion of the file the operator is informed where the input terminates and the file could be extended by using this information as the starting block in a further similar operation.

There is no question of the input supervisor making an entry on the job list at this stage of the process. Input which has been recorded on magnetic tape in this manner can be read by jobs which incorporate the appropriate statement in the input section of their job description.

The Scheduler

In the original design of the system the aim of the scheduler was to have at least three jobs in their execute phase at the same time. These three jobs would be a magnetic tape job, an output limited job and a computer limited job. The objective was to maintain a high level of activity on the tape system, and force short development jobs through the computer limited ones in order to give them a shorter turn-around time.

Also the computer limited job was treated as a base load to absorb central processor (CPU) time available whilst the tape and peripheral jobs were waiting for magnetic transfers.

In practice it was found that users store requests, especially during compiling, were often too large to allow this. Even when this did not apply a further serious disadvantage was that the system only executed the computer-limited job when the magnetic tape job was halted for transfers or searches, and the peripheral limited job could not use the CPU. As a result of this the actual running time of computer limited jobs became extended by a factor of ten or more. Because the users do not accept the job time as a measure of priority, this created obvious management difficulties in guaranteeing reasonable turn-around times for the computer limited jobs. It is also an embarrassment to the system to artificially prolong the run time of the longest jobs by a factor of ten. For those reasons the distinction between computer and output limited jobs has been removed. Now the scheduler aims to maintain in the store one job using magnetic tape, one other job (normally one not using magnetic tape) and a successor in the assembly phase. If this can be achieved, the magnetic tape job (which is given higher priority) uses the CPU whenever it is able and the other job uses the CPU when the tape job is waiting for a tape search or block transfer to be completed. Thus the CPU is theoretically kept busy all the time, the tape system nearly so, and there is no gap between jobs. However, in practice, if the size of jobs exceed one half the size of the store, then either the tape system or the CPU is idle at any instant. Also the machine may idle between jobs because there is not enough space to have the next job assembled and waiting.

Magnetic tape jobs are selected for assembly in the order in which they are presented to the machine, but if the magnetic tapes for a job are not mounted upon request, then assembly of the next tape job may begin. If the tapes for the second job are mounted first then the first job will be by-passed.

This overcomes most delays due to programmers asking for non-existant tapes or operational delays in finding tapes. Non-magnetic tape jobs are streamed according to the dominant output request in the job description. Jobs for the slowest peripherals are given priority until a reasonable amount of output for the device is waiting in the output buffer, then the scheduler moves on to the next slowest peripheral. It has been found, by sensibly defining this reasonable amount of output, that each peripheral tends to be kept in maximum use. Obviously the useage of a peripheral is dependant entirely on the number of jobs in the system requiring it.

The Central Executive

One obvious task of this part of the supervisor is to start, end and monitor the running of jobs. This includes trapping jobs which violate the limits specified in the job description (computing time, store requirement and amount of output), or commit errors such as division by zero, reference to sacred store etc. The job is terminated for these violations unless the facility for setting a private trap has been invoked (most compilers in fact do this). In this case control will return to a preset address in the program, and if a limit has been exceeded then a small additional allowance is made to allow a fault postmortem. For other errors the program can be designed to rescue itself and continue. This part of the supervisor is also responsible for the coordination of extracode functions which interact with the system, such as input, output and magnetic tape operations.

Storage organisation is a second major task of the central executive. In Atlas the instruction format permits the programmer to address directly 2048 blocks each of 512 words. Of these 2048 block addresses the most significant 256 are reserved for the use of the supervisor and are sacred, i.e. any attempt by a user program to access these reserved addresses is prevented.

An Appraisal of The Atlas Supervisor

The maximum amount of physical store available to a program on the Manchester University Atlas is 165 blocks. The addresses by which those blocks are referred to by the programmer may, however, be anywhere within the permitted range of 1792 blocks. Each program defines within its job description the amount of store that it will require for both the compile phase (compile store) and the execute phase (execute store) and the supervisor allocates to the program a block directory of the appropriate length. Within this directory each entry is in two parts, the first being the block address in the program address space and the second is the physical position of the block within the store. It would be possible for all store accesses to be translated into physical addresses by reference to this block directory. Such a technique would take an excessive amount of time and it is therefore not used.

On the Manchester University Atlas only 32 of the blocks are located in directly addressable core store, the remainder are on magnetic drums. In order that program addresses can be translated into physical addresses within the store without significant delay an indirect addressing scheme has been provided by hardware. Associated with each block, (or page), of core store is a 12 bit register, generally referred to as a page address register (PAR), into which the supervisor can place the address, in the program address space, of the block actually in that page of core store. Whenever a store request is made, the block part of the address is compared with all 32 PAR's in parallel and the physical position of the block in the core store is determined. The cost of this indirect addressing is an increase in the store access time of approximately 30 percent but since this access time is in general overlapped with other operations of the CPU there is no significant loss.

If the requested address is not in a PAR then an interrupt occurs and the central executive is entered. The block directory is then scanned to find the position of the required block on the drum store and the block is transferred to a vacant page of the core store. The associated PAR is set to the correct block number and the main program is restarted at the instruction which caused the interrupt. Whilst the block required by the main program is being transferred back from the drum, the central executive selects another page to be transferred back to the drum, thus maintaining an empty page in the core store.

This transfer is initiated before re-entering the program and is overlapped with normal program operation. For each page of the core store there is a use digit which is set when the page is accessed. All use digits are scanned and set at regular intervals by the central executive and a pattern of use is established. The selection of the page to be transferred to the drum is made with respect to this pattern of use. The algorithm used is referred to as the one level store learning program (5). The efficiency of this one level store is given later but on average 99.99 percent of store requests are for blocks currently in the core store. When this is not the case the average CPU time lost per one level store transfer is 20 m sec, 6 m sec being in the central executive and 14 m sec waiting for the transfer.

The most significant digit in a PAR is not part of the address. When it is set the page is locked out, and cannot be accessed by user programs. The central executive organises program protection by locking out all pages which are not part of the current program (i.e. the one occupying the central registers). Pages of the current program involved in peripheral or magnetic transfers are also locked out. Protection between program areas on the drum is by software.

The final major task of the central executive is timesharing the CPU between supervisor and user programs. The Supervisor takes priority over user programs but further priority rules apply to each. There is also an interrupt mode of operation which takes priority over both and is a hardware feature (6).

The Supervisor is initiated by entry from any interrupt routine to a central coordinating routine and one of three priority streams is specified. The priorities are used as follows:-

1) Drum and one level store activities use top priority

2) Magnetic tape operations take middle priority

3) Slow peripheral routines have bottom priority

An Appraisal of The Atlas Supervisor

If the supervisor is not active when a request is made, entry is immediate and takes only 0.08 msec, because it uses only a reserved set of B registers and has its own working space. If it is active then the request is added to the appropriate queue. Also the current supervisor activity may add other entries to these queues. All exits from the supervisor are to another address in the coordinating routine which scans these queues in order of priority and initiates the first waiting entry. Supervisor activities are never stopped to allow higher priority ones to proceed, but will be halted if they are waiting for completion of some action on drum, tape or slow peripheral. In this case a dormant entry is made at the bottom of the appropriate queue, containing a preset re-entry address. This entry will move to the active part of the queue when freed by the completion of the action it awaits. Switching the CPU between supervisor activities takes approximately 0.2 ms.

User program priorities are established by the scheduler when it enters them on the execute list. On this list, entries are linked in descending order of priority, which is :-

1) Jobs assigned top priority by operators

2) Magnetic tape jobs

3) Peripheral limited jobs

4) Computer limited jobs (not now distinguished from (3))

5) Jobs assigned bottom priority by operators

Jobs on the execute list may be in one of two states, free to go or halted awaiting completion of a magnetic or peripheral activity. When all supervisor activity is completed, the highest free program on the list is started. If it is not the current program then a program change is performed which takes 2ms. An exception to this rule, which has been implemented only on the Manchester Installation, is that the program change has been inhibited when the current program is of higher priority than the first free program and is halted for a one level store transfer. The extent by which this has reduced the total number of one level store transfers has resulted in a net gain in efficiency. On the Atlas Installations with larger core stores, program changing on one level store transfers is effective.

Magnetic Tape Supervisor.

Atlas magnetic tape is pre-addressed tape with fixed length blocks of 512 words each of 48 bits. The blocks are numbered sequentially from 0 to n, where $n \leq 5000$ depending on the size required by the user. Block 0 is not accessable to the user, and is used to record the tape title and various pieces of information connected with the tape (length of tape, date when addressed etc).

The main extracode instructions available to the user for manipulating magnetic tapes perform the following operations:-

1. Search for any specified block on tape - this positions the magnetic tape with the read/write head immediately before the specified block.
2. Read forwards n blocks ($n \leq 8$). This extracode reads the next n blocks from the magnetic tape to the specified store blocks.
3. Read backwards n blocks ($n \leq 8$). This reads the previous n blocks from the tape to the specified store blocks. The reading is performed whilst the tape is moving in reverse, but the order of the words in each block is the same as for forward read.
4. Write forward n blocks ($n \leq 8$). This extracode writes n blocks from the specified address in store to the next n blocks on tape.

All tape extracodes cause the magnetic tape supervisor to be activated. This first locks out the blocks of store which are involved in the transfer, and thus prevents them being accessed before the transfer has been completed. It then records the order in a queue for the appropriate deck. When this order reaches the top of the queue its execution is initiated by the magnetic tape supervisor. In the mean time control is returned to the next instruction of the user program. With a fairly small amount of attention from the supervisor the order will be executed at the transport speed of the tape autonomously with computation in the CPU. At the end of the operation the lock out on the store block is removed. Providing the job does not refer to a locked out block its computation will be overlapped with the tape operations. If it does refer to a locked out block it will be halted until the lock-out is removed and a program change will be performed if another program is free to go. The total time during which a tape job is halted for this reason (magnetic tape halt time) is recorded for costing purposes. It is always advantageous to minimise halt time particularly in jobs using a lot of store. Typical magnetic tape speeds are 16 blocks/sec for transfers and short searches (<100 blocks), searches greater than 100 blocks are slightly faster.

An Appraisal of The Atlas Supervisor

Output Assembly.

Output normally comes from a user program character by character, this is packed into blocks and transferred to the output well. The format of the blocks in the output well is the same as those in the input well. As each block of output is filled it is linked into the appropriate file (a program may be generating output for up to 16 different files). When the user program breaks an output file (this can be done by the user during execution but is also done automatically at end of job), the output assembler makes an entry for the file in the output list, and then activates the output scheduler. Also at this stage, the records showing amount of output waiting for each device are updated. These are required by the (job) scheduler in deciding which output peripherals need more output.

It is also possible for the user, instead of allowing his output to go to the output well to route it to a private magnetic tape. It can be printed later in a simulated off-line manner, by using a simple steering tape, or it can be used as input for another job.

The output scheduler takes the leading entry from the output list for each output device and initiates the transmission of the output file through the peripheral. If necessary it prints a request for the operator to engage the peripheral. If the device is in use then a flag is set to show more output is waiting for this device.

An Appraisal of The Atlas Supervisor

SYSTEM PERFORMANCE

In evaluating the performance of a computing system one is tempted to use the terms throughput and turnaround time, but both these have nebulous connotations. The most general interpretation of throughput would be a function of :-

1) The power of the programming languages
2) Compiler efficiencies
3) The power of debugging aids and the extent of file storage and editing facilities
4) The turnaround time of jobs submitted to the system
5) System efficiency
6) Store size and configuration
7) The speed and structure of the order code

These features are not mutually independent, there are for example, important interactions between (1) and (2) and between (3) (5) and (7). However we shall only discuss here system efficiency, turnaround time and the structure of the job mix being processed. We define system efficiency as the percentage time the CPU is active on user jobs. This is the time the CPU is occupied executing instructions in compilers (compile time) and compiled user programs (execute time). The sum of these two we refer to as compute time. It excludes all waits for magnetic and peripheral transfers which are not eliminated by multiprogramming (idle time), and time spent inside the supervisor (supervisor time). The mechanism by which these quantities are measured is of interest.

An Appraisal of The Atlas Supervisor

The Atlas computer has a counter which counts all instructions obeyed on main or extracode control. Although it was provided mainly to implement the one level store learning program, the supervisor also makes use of it to determine how many instructions are obeyed in compiling, execution, supervisor and idling. The conversion of these instructions counts to real time is not straightforward as Atlas is an asynchronous machine. Instruction times depend upon type, operands involved and context. However, variations are not large except for multiplication and division. A modification to the counter to count two and four respectively for these instructions has resulted in an average of three microseconds per instruction count. This average has been checked for each of the four activities defined above and has been found satisfactory.

The instruction counter does not operate for interrupt control and an estimate of the time spent in this mode is arrived at indirectly. The principal routines obeyed on interrupt control are those which control the input/output peripherals, magnetic tape and drums. The number of times that these routines are entered is known from the amount of input/output and the total number of transfers, and from these the estimate is made. It has also been checked by attaching a 1 microsec clock to the machine whose pulses were gated with the flip flop which indicates interrupt mode. We have found that interrupt time averages less than 5 percent and have ignored it in all efficiency measurements.

The supervisor produces an automatic log on paper tape which consists of a record for each job giving the following statistics.

Time started

Job identification

Amount and type of input

Amount and type of output

Instructions obeyed during compiling

Instructions obeyed during execution

Amount of store used during compiling

Amount of store used during execution

Number of one level store transfers

Number of tape transfers

Amount of idling since end of previous job

Instructions obeyed in supervisor since end of previous job

Time completed

These figures provide an accurate picture of the behaviour of the system, and are also used for accounting.

Typical average performance figures for the Manchester Atlas during the time it is available to the University Computing Service are:

56% execute time

5% compile time

26% idle time

13% supervisor time

or 61% system efficiency

they ignore an approximate 5% interrupt time.

These figures compare favourably with the figures given by Dr. Amdahl for the IBM 7090 (7). A substantial part of the 'lost time' is due to the one level store namely 11% idle time 5% supervisor time. This is of course less on the other Atlas Installations which have larger core stores. We shall discuss later how variations in the job mix affect the system efficiency.

An Appraisal of The Atlas Supervisor

Turnaround Time

Job turnaround time is a function of the system speed, the number of users, their proximity to the machine and management policy. The weekly turnaround of the Manchester University Computing Service (UCS) is currently 2000 to 2500 jobs. Obviously the onsite user with access to the operator request to give his job top priority can achieve very good turnaround times. The turnaround time for the remote non-privileged user is at least the transport time to the installation plus N/2000 weeks, where N is the number of jobs before his in the queue. In order to keep N within reasonable bounds, the management policy of the computing service is to give each group of users (usually a department of the university) a daily allocation, thus N/2000 cannot exceed one day. The actual turnaround a user can achieve is also affected by the access of the UCS to the computer and the interface between the user and the UCS. The UCS has the use of the computer for alternate 2 or 3 hour periods throughout the 24 hours of a day.* The interface is a set of wire baskets into which the user places his job and to which the UCS return his results. The wire basket buffer is only processed between 8.0 am and 5.0 pm, due partly to staffing difficulties and partly to a belief that users are not very interested in either presenting jobs or receiving results outside these hours. The turnaround achieved by a user is thus at least once per 24 hours. If his allocation permits and if he can match his pattern of work to that of the service, he can achieve 2 turnarounds per day. If a better turnaround than this were to be provided it would require a priority system for bypassing the wire basket buffer. Obviously this kind of system defeats itself unless its use is seriously restricted. One possible use of the disc store which is being attached to Atlas would be to dispense with the wire basket buffer and store all waiting jobs on the disc. Figure 2 shows the possible system modification under consideration. Here jobs would go through the input supervisor to the job buffer and entries would be made on the job buffer directory streamed into user groups.

---

* There is some loss in efficiency due to the enforced discontinuation which occur as a result of this.

Whenever the number of jobs in the input well was small, new jobs would be selected from the job buffer and transferred to the input well and job list. A simple algorithm for choice would be to select jobs from the user group with the smallest time used/time allocated ratio. Jobs with a special operators code would be allowed to bypass the job buffer and go straight into the input well. A service job would be provided to list all jobs of a given code in the job buffer together with their expected completion time and could possibly reposition selected jobs to achieve a quicker turnaround.

The Job Mix

The job mix being processed by the UCS is by no means constant. However, there is some pattern in the variation. One type of variation is seasonal. During term time, in particular the Michaelmas Term, there are a lot of short development jobs from undergraduates and beginners. During the vacations, in particular the long summer vacation, the work includes more production runs from the experienced users. This results in longer average compute times. There is also a marked difference between compute times on weekdays and weekends. It is the users habit to give priority to the shorter development jobs during the week and to hold back the longer jobs for weekend running. The difference between compute time during day and night sessions is small on the Manchester Installation and has been ignored.

In order to demonstrate the above variations we have investigated in some detail the four periods

A) four weeks during the summer vacation of 1966

B) four weeks during the Michaelmas Term of 1966

C) the weekdays only of B

D) the weekends only of A

Table 1 shows the variations of the average job profile during these four periods.

It can be seen that in all four cases the major input media is paper tape. Output is mainly on the lineprinters although some is taken on paper tape for subsequent off-line transmission to remote users. Magnetic tape usage is shown separately in Table 11. The blocks transferred and the halt time are averaged over only those jobs which use magnetic tape. Most magnetic tape users use it for storing private compilers and dumping compiled programs or restart data, hence the average number of blocks transferred is small. The large halt time is the result of users sharing tapes and having to wind past the parts used by others.

The significance of the variations in the average job profiles can be seen from table 111. This gives the percentage of jobs falling into each of five ranges of compute time and the percentage CPU time required by each category is given in table 1V. It is interesting to note that for the mean mid-term case (period B) :-

59.4 percent of the jobs require only 3.1 percent of the CPU time whilst 8.1 percent (i.e. the longer jobs) require 71 percent of the CPU time.

As one would expect there is some correlation between the compute time of jobs and their other demands on the system. Table V shows the variation of the main features of the job profile against compute time. The high proportion of one level store time for short jobs is due to the high rate of 'page turning' during the initiation of the compile phase. It is obvious from the mean compute times and the input/output transmission times given in table V that a simple system with no buffering would suffice for the longer jobs, and it is the shorter ones which justify the complexity of the Atlas buffering system. Even so the system is unable to keep pace with the shortest jobs, and idling occurs whilst they are being assembled. The job assembly time, ignoring delays in accessing the processed job description and the input files, is the time taken to transfer the required compiler into store. The shortest time for this is five seconds because most compilers are approximately 40 blocks long, and repeated use of the same compiler involves a 40 block rewind followed by reading forwards 40 blocks.

In practice there is a delay in accessing the input well, which is dependent on the number of jobs it contains. Also the compiler assembly times are longer than the above because several different compilers are used. For these reasons the system efficiency (table VI) falls as the percentage of short jobs increases (table III). Clearly the best system efficiency that can be achieved for the shortest 23.5 percent of jobs in period B (with a mean compute time of 0.5 secs) is 10 percent.

It is interesting to note that the percentage of magnetic tape jobs (table II) increases with the system efficiency, which indicates that the magnetic tape halt time is largely absorbed by multiprogramming.

An approximate breakdown of the mean idle time and supervisor time per job is given in table VII. The first item is the idle time occuring due to one level store holdups, and is computed from the mean number of drum transfers per job. The other items are obtained from an analysis of the automatic log. This enables us to locate the idle periods due to no jobs being in the execute phase. Whenever these interjob gaps are larger than 30 secs we assume that the input well is empty or contains only magnetic jobs and the requested tapes have not been mounted.* This idle time is called enforced idling. We shall not discuss how it arises since it is external to the system and concerned with Computing Service management and site conditions. Although the system fails to completely eliminate the remaining interjob gaps, there are small compared to the job assembly times and very small compared to the input/output times of table V. Replacement of magnetic tape by disc file for the input/output wells will reduce these gaps significantly. A further part of the total idle time is accounted for by system restarts between sessions and after machine faults. In the latter case the system restarts automatically but some time is lost on fault printouts and repositioning tapes. The remaining idle time is difficult to account for but some is undoubtedly due to the system not fully utilizing the CPU during the magnetic tape idle time (see table 1).

---

* This happens infrequently since the requests for tapes are printed several jobs ahead, and usually means that tapes are not directly available.

An Appraisal of The Atlas Supervisor

Table VII also gives an approximate breakdown of the mean supervisor time per job. The first item of this section is an estimate of the time spent organising one level store transfers within user programs. The system housekeeping also causes some one level store transfers, the time taken organizing these is listed seperately. Both these items include the time used in the decision algorithms. The time spent organising the input/output buffering is also given. A detailed breakdown of the remaining supervisor time is not attempted. It is used mainly for magnetic tape control and general housekeeping associated with starting and ending jobs, storage allocation, program changing and organisational extracode functions.

CONCLUSIONS

It is obvious from the figures presented that a major factor to be considered in the design of an operating system is the expected job mix. We believe our job mix to be fairly typical for those installations which serve large groups of users with varied interests, having access to no other machine. However, it is artificially constrained in some ways by the UCS policy, for example magnetic tape usage is restricted. Apart from their greater magnetic usage the job mix of the ICT Computing Service on the Manchester Atlas is not significantly different.

It is interesting to speculate on the probable job mix and hence the efficiency of a system of ten times the 'power' of Atlas. The problems associated with a high proportion of short jobs will be accentuated since current hardware developments are increasing CPU speeds at a greater rate than the data movement through storage hierarchies. We see no reason for this proportion to fall, indeed the provision of file storage, editing facilities and on-line access will probably increase it. Some reduction in the amount of data movement will result from the use of larger direct access stores and the use of pure procedures. However, users appear to have an insatiable appetite for store and we consider large scale one level store techniques to be vital.

A final lesson learnt from the Atlas system is that users should not be expected to supply accurate information on which sheduling and storage allocation are based. See for example, figure 3 which gives the percentage accuracy of users time and storage requirements. The decision mentioned in the section on the Scheduler, to abandon the distinction between long and short jobs, was influenced by this. It could be profitably reversed if that 59.4 of jobs in period B which require only 3.1 of the CPU time could be located from the users estimates.

An Appraisal of The Atlas Supervisor

Acknowledgements

The Atlas Computer and its Supervisor are the result of a joint project between Professor Kilburn's group at Manchester University and Ferranti Ltd., (more recently International Computers and Tabulators Ltd.). The development of the supervisor has involved a large number of people but we particularly wish to acknowledge the major contribution by Dr. D.J. Howarth of I.C.T. throughout the whole project.

We also acknowledge the assistance of our colleagues in collecting and analysing the data, in particular Dr. J.S. Rohl of Manchester University.

REFERENCES

1. 'The Manchester University Atlas Operating System' - Part 1,
Kilburn, T; Howarth, D.J; Payne, R.B; Sumner, F.H.
'The Computer Journal' October 1961.

2. 'The Manchester University Atlas Operating System' - Part 2,
Howarth, D.J; Payne, R.B; Sumner, F.H.

3. 'The Atlas Supervisor'
Kilburn, T; Payne, R.B; and Howarth, D.J.
'Computers - Key to Total Systems Control' AFIPS 1961.

4. 'The Atlas Schoduling System'
Howarth, D.J; Jones, P.D; and Wyld, M.
'The Computer Journal' October 1962.

5. 'One-Level Storage System'
Kilburn, T; Edwards, D.B.G; Lanigan, M.J; Sumner, F.H.
'IRE Transactions on Electronic Computers', vol. 11, No. 2,
April 1962.

6. 'The Central Control Unit of the Atlas Computer'
Sumner, F.H; Chen, E.C.Y; Haley,G.
'Proc. IFIP Congress' 1962.

7. 'Architecture of the IBM System/360'
Amdahl, G.M; Blaauw, G.A; Brook, F.P.
'I.B.M. Journal of Research and Development' vol.8, No.2,
April 1964.

An Appraisal of The Atlas Supervisor

APPENDIX OF TABLES AND DIAGRAMS

TABLE 1

		TIME PERIOD			
	UNITS	A	B	C	D
compute time	SECS	63.6	41.7	33.4	273.9
compile time	SECS	5.2	3.9	3.7	6.8
idle time	SECS	25.9	19.1	18.4	40.1
supervisor time	SECS	13.0	9.7	9.3	20.6
compile store requested	BLOCKS	71	67	66	71
compile store used	BLOCKS	50	47	47	49
execute store requested	BLOCKS	42	38	38	48
execute store used	BLOCKS	28	26	26	34
paper tape input	BLOCKS	4.2	3.8	3.6	5.7
card input	CARDS	1	3	2	6
lineprinter output	LINES	373	286	276	527
paper tape output	BLOCKS	0.3	0.5	0.5	0.3
card output	CARDS	2	1	1	2

AVERAGE JOB PROFILES

TABLE 11

		TIME PERIOD			
	UNITS	A	B	C	D
Jobs using magnetic tapes	%	17.4	15.4	14.9	27.1
Number of blocks transferred	BLOCKS	146	113	115	107
Magnetic tape halt time	SECS	35.8	26.0	24.7	46.7

MAGNETIC TAPE USAGE

TABLE 111

	TIME PERIOD			
computo time	A	B	C	D
< 1 soc	10.6	23.5	24.5	4.7
1 - 8 secs	37.3	35.9	37.8	15.6
8 - 120 socs	39.4	32.5	30.9	43.1
120 - 960 socs	11.1	7.6	6.8	23.4
> 960 secs	1.6	0.5	0	13.2

PERCENTAGE DISTRIBUTION OF JOBS AGAINST computo time

TABLE IV

	TIME PERIOD			
computo time	A	B	C	D
< 1 soc	0.1	0.3	0.4	0.006
1 - 8 socs	1.9	2.8	3.8	0.2
8 - 120 socs	19.5	25.3	30.7	5.7
120 - 960 socs	48.5	59.0	65.1	38.0
> 960 socs	30.0	12.5	0	56.1

PERCENTAGE DISTRUBTION OF C.P.U. TIME AGAINST computo time

TABLE V

compute time range	mean	one level store time	paper tape input	lineprinter output	transmission time I/O
<1sec	0.5sec	1.5 sec	1.4 blocks	45 lines	21.8 secs
1-8sec	3.3sec	3.7 sec	3.5 blocks	160 lines	53.3 secs
8-120sec	32 secs	13.5 secs	6 blocks	500 lines	118.0 secs
120-960sec	310 secs	50 secs	7 blocks	720 lines	148.6 secs
>960sec	1200 secs	40 secs	4.5 blocks	700 lines	114.5 secs

VARIATION OF MAIN ELEMENTS OF JOB PROFILE WITH compute time

TABLE VI

	TIME PERIOD			
	A	B	C	D
compute time	62.0	59.2	54.7	81.8
idle time	25.3	27.1	30.1	12.0
supervisor time	12.7	13.7	15.2	6.2
SYSTEM EFFICIENCY	62.0	59.2	54.7	81.8

SYSTEM PERFORMANCE (PERCENT)

TABLE VII

	TIME PERIOD			
	A	B	C	D
IDLING DUE TO DRUM TRANSFER	10.9	7.5	7.1	20.6
IDLING DUE TO RESTARTS	5.7	3.4	3.3	5.9
IDLING DUE TO INTERJOB GAPS	2.8	4.2	4.3	2.8
ENFORCED IDLING	5.6	3.1	3.0	9.6
OTHER IDLING TIME	0.9	0.9	0.7	1.2
TOTAL IDLING TIME	25.9	19.1	18.4	40.1
SUPERVISOR TIME DUE TO USER DRUM TRANSFERS	4.7	3.2	3.0	8.8
SUPERVISOR TIME DUE TO INPUT	1.3	1.2	1.1	1.8
SUPERVISOR TIME DUE TO OUTPUT	2.1	1.7	1.7	2.9
SUPERVISOR TIME DUE TO SYSTEM DRUM TRANSFERS	1.2	1.3	1.3	2.6
OTHER SUPERVISOR TIME	3.7	2.3	2.2	4.5
TOTAL SUPERVISOR TIME	13.0	9.7	9.3	20.6

BREAKDOWN OF IDLE AND SUPERVISOR TIME (SECS)

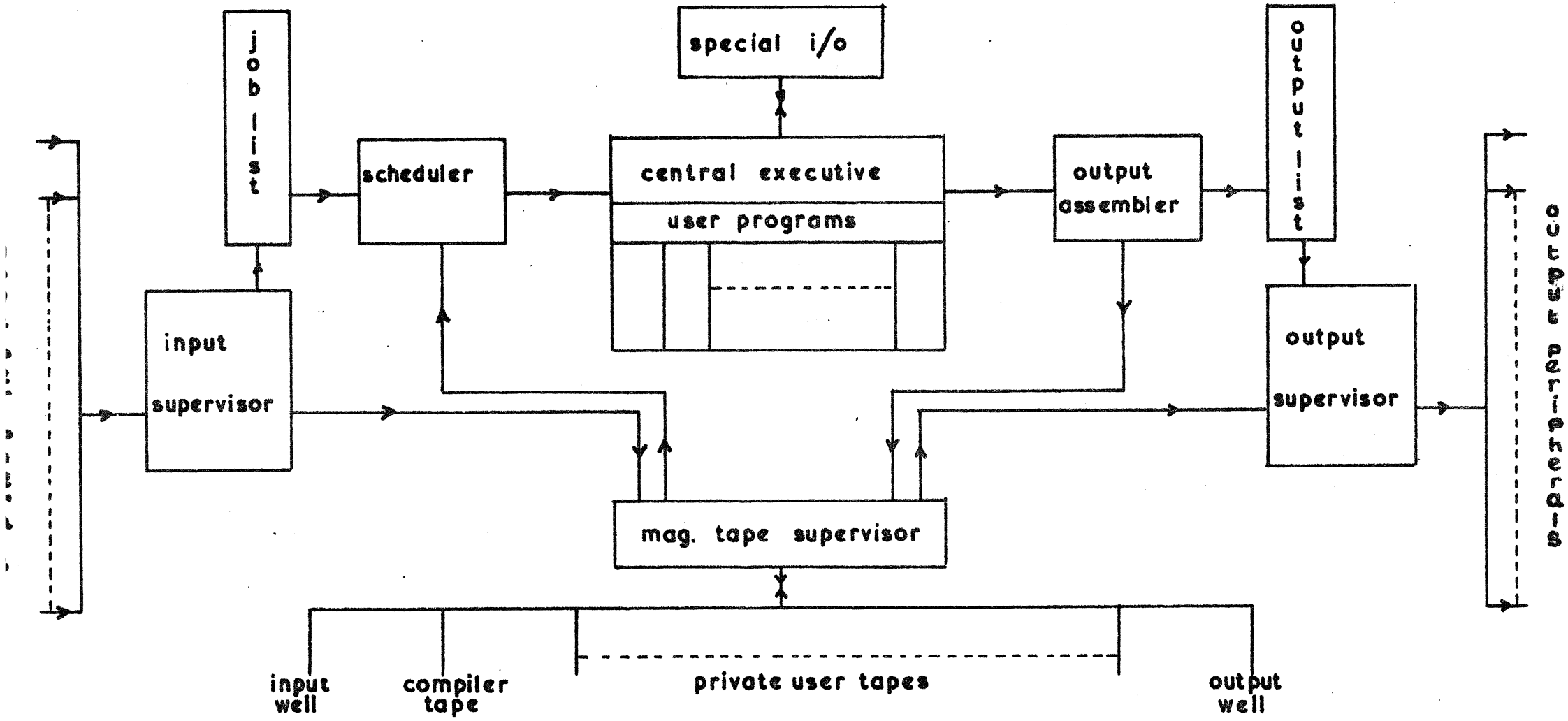

job list
special i/o
output list
scheduler
central executive
user programs
output assembler
input supervisor
output supervisor
mag. tape supervisor
input well
compiler tape
private user tapes
output well

FIG 1

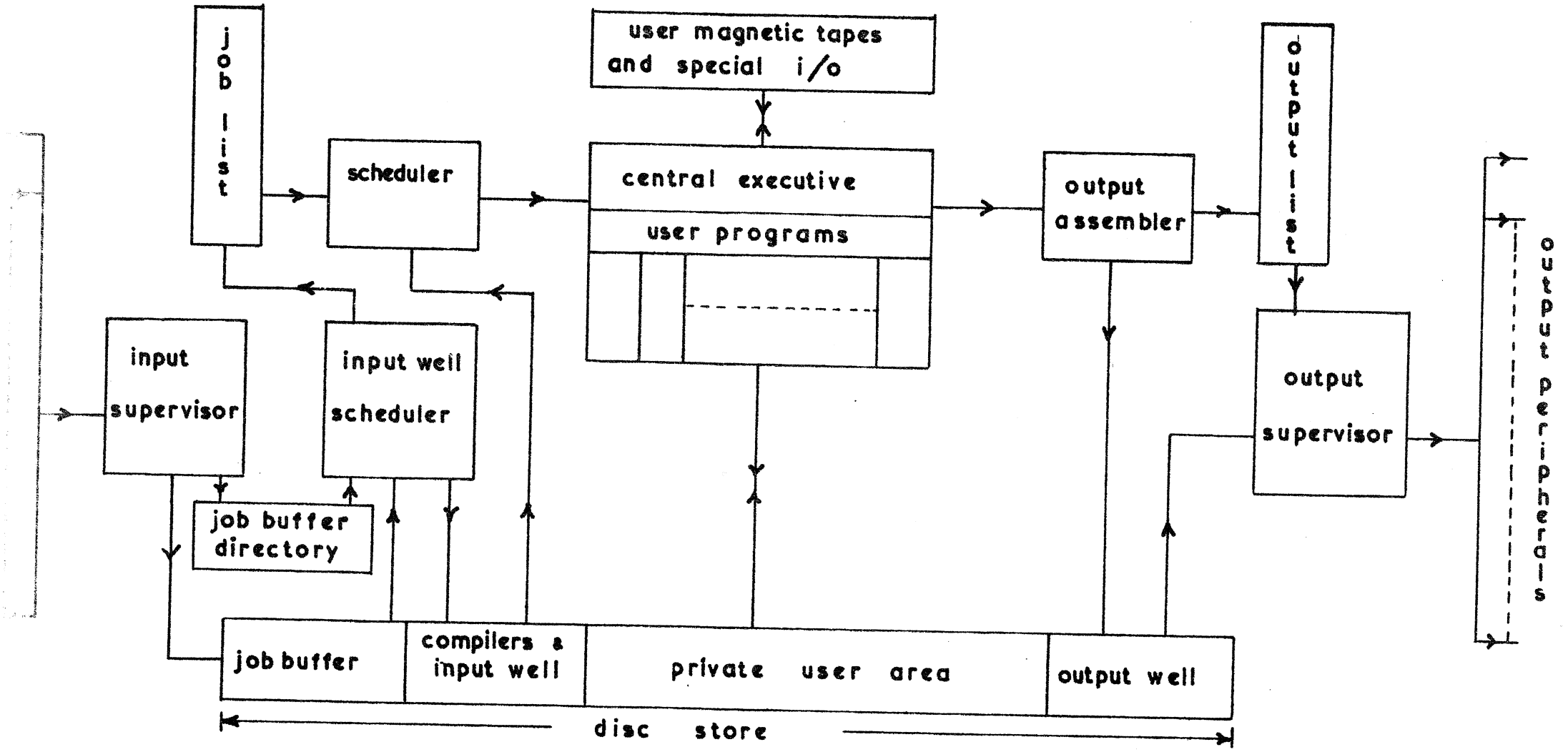

job list
scheduler
user magnetic tapes and special i/o
central executive
user programs
output assembler
output list
output peripherals
input supervisor
input well scheduler
output supervisor
job buffer directory
job buffer
compilers & input well
private user area
output well
disc store

FIG2

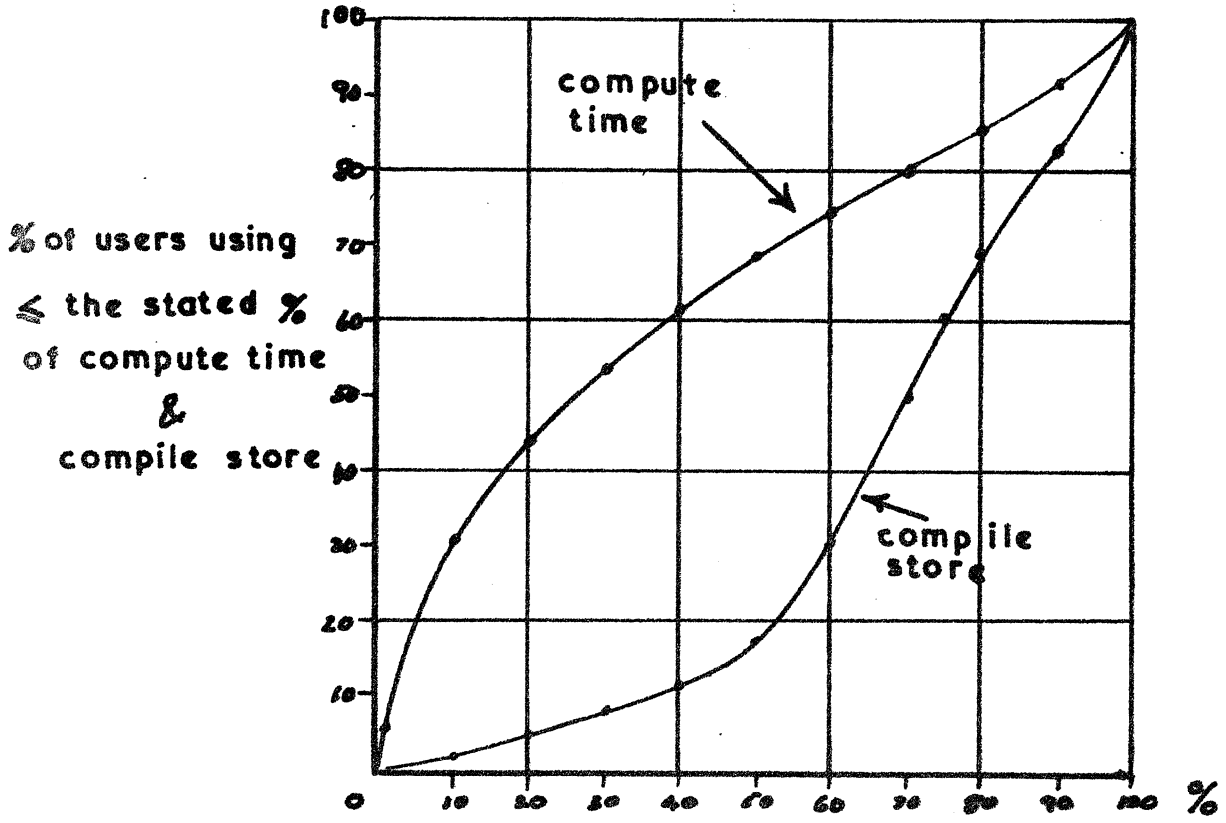

% of users using
≤ the stated %
of compute time
&
compile store
compute time
compile store
100
90
80
70
60
50
40
30
20
10
0
10
20
30
40
50
60
70
80
90
100
%

FIG3

Basic B-Register Arithmetic

100	$ba' = s - ba$	110	$s' = s - ba$	120	$ba' = n - ba$
101	$ba' = s$	111	$s' = -ba$	121	$ba' = n$
102	$ba' = ba - s$	112	$s' = ba - s$	122	$ba' = ba - n$
103	$ba' = -s$	113	$s' = ba$	123	$ba' = -n$
104	$ba' = ba + s$	114	$s' = ba + s$	124	$ba' = ba + n$

Note: The B-Carry Digit is set by instructions ending 0, 2 or 4.

Logical B-Register Operations

107	$ba' = ba \,\&\, s$	106	$ba' = ba \neq s$
117	$s' = ba \,\&\, s$	116	$s' = ba \neq s$
127	$ba' = ba \,\&\, n$	126	$ba' = ba \neq n$
164*	$ba' = ba + (bm \,\&\, n)$	147	$ba' = ba \vee s$
165*	$ba' = bm \,\&\, n$	167	$ba' = ba \vee n$

* If Bm is BO, $bm \,\&\, n = n$.

Note: 164 sets the B-Carry Digit.

Test Instructions

$ba' = n$ if:-

210	bm is odd (i.e. bit 23 = 1)	224	$bt = 0$
		225	$bt \neq 0$
211	bm is even (i.e. bit 23 = 0)	226	$bt \geqslant 0$
		227	$bt < 0$
214	$bm = 0$	234	$ax = 0$
215	$bm \neq 0$	235	$ax \neq 0$
216	$bm \geqslant 0$	236	$ax \geqslant 0$
217	$bm < 0$	237	$ax < 0$

Test and Count Codes

200	If $bm \neq 0$, $ba' = n$ and $bm' = bm + 0.4$
201	If $bm \neq 0$, $ba' = n$ and $bm' = bm + 1.0$
202	If $bm \neq 0$, $ba' = n$ and $bm' = bm - 0.4$
203	If $bm \neq 0$, $ba' = n$ and $bm' = bm - 1.0$
220	If $bt \neq 0$, $ba' = n$ and $bm' = bm + 0.4$
221	If $bt \neq 0$, $ba' = n$ and $bm' = bm + 1.0$
222	If $bt \neq 0$, $ba' = n$ and $bm' = bm - 0.4$
223	If $bt \neq 0$, $ba' = n$ and $bm' = bm - 1.0$

Accumulator Transfer Instructions

324	$am' = s$	Q
325	$am' = -s$	QE
334	$am' = s$	
335	$am' = -s$	AO
314	$am' = s$	NL
315	$am' = -s$	NL AO
367	$am' = \lvert s \rvert$	QE
344	$l' = sx$, $am' = am$	
345	$l' = sx$, $m' =$ sign of s; $ay' = ay$	
346	$s' = am$, $a' = 0$	
356	$s' = am$, $a' = a$	
347	$s' = al$, $l' = 0$	
357	$s' = al$, $a' = a$	

Other Accumulator Instructions

364	$ax' = 8ax$, $ay' = ay$	
365	$ax' = ax/8$, $ay' = ay$	
310	$a' = a + s$ if $ay \leqslant sy$ (otherwise $am' = am + s$, l spoiled)	QE NL
311	$a' = a - s$ if $ay \leqslant sy$ (otherwise $am' = am - s$, l spoiled)	QE NL
352	$a' = am \times s$, sign l' = sign m'	E AO
353	$a' = -am \times s$, sign l' = sign m'	E AO
376	$al' = a/\lvert s \rvert$ if $a \geqslant 0$; ay' = exponent of quotient, m' = "remainder"	E DO
377	$al' = \lvert am \rvert / \lvert s \rvert$; ay' = exponent of quotient, m' = "remainder"	E DO
361	$am' = a$	RE NL
354	$am' = a$ rounded by adding	AO NL
355	$a' = al \times 8^{-13}$	Q
366	$am' = \lvert am \rvert$	QE

B-Register Extracodes

1300	$ba' =$ int. pt. of s; $am' =$ frac. pt. of s	
1301	$ba' =$ int. pt. of am; $am' =$ frac. pt. of am	
1302	$ba' = ba \times n$	21-bit integers
1303	$ba' = -ba \times n$	21-bit integers
1304	$ba' = ba/n$; $b97' =$ remainder	21-bit integers
1312	$ba' = ba \times n$	24-bit integers
1313	$ba' = -ba \times n$	24-bit integers
1314	$ba' = ba/n$; $b97' =$ remainder	24-bit integers

Shifts

1340	$ba' = ba \times 2^{-n}$ arithmetic
1341	$ba' = ba \times 2^{n}$ arithmetic
1342	$ba' = ba \times 2^{-n}$ circular
1343	$ba' = ba \times 2^{n}$ circular
1344	$ba' = ba \times 2^{-n}$ logical
1345	$ba' = ba \times 2^{n}$ logical

Trigonometric Functions

1730	$am' = \sin s$	
1731	$am' = \sin aq$	
1732	$am' = \cos s$	
1733	$am' = \cos aq$	
1734	$am' = \tan s$	
1735	$am' = \tan aq$	
1720	$am' = \arcsin s$ $(-\pi/2 \leqslant am' \leqslant \pi/2)$	
1721	$am' = \arcsin aq$ $(-\pi/2 \leqslant am' \leqslant \pi/2)$	
1722	$am' = \arccos s$ $(0 \leqslant am' \leqslant \pi)$	
1723	$am' = \arccos aq$ $(0 \leqslant am' \leqslant \pi)$	
1724	$am' = \arctan s$ $(-\pi/2 < am' < \pi/2)$	
1725	$am' = \arctan aq$ $(-\pi/2 < am' < \pi/2)$	
1726	$am' = \arctan (aq/s)$ $(-\pi < am' \leqslant \pi)$	

Arithmetic Extracode Functions

1700	$am' = \log s$	
1701	$am' = \log aq$	
1702	$am' = \exp s$	
1703	$am' = \exp aq$	
1704	$a' =$ integral part of s	
1705	$a' =$ integral part of a	
1706	$a' =$ sign of s	
1707	$a' =$ sign of a	
1710	$am' = \sqrt{s}$	
1711	$am' = \sqrt{aq}$	
1712	$am' = \sqrt{aq^2 + s^2}$	
1760	$am' = am^2$	
1713	$am' = aq^s$ $(aq \geqslant 0)$	
1714	$am' = 1/s$	DO
1715	$am' = 1/am$	DO
1757	$am' = s/am$	DO
1774	$am' = am/s$	QRE
1775	$am' = aq/s$	
1776	$s' =$ quotient; $am' =$ remainder; if used immediately after 1574, 1575, 1774, 1775.	Q Q
1756	$s' = am$, $am' = s$	
1766	$am' = \lvert s \rvert$	AO
1767	$am' = \lvert am \rvert$	AO

Double Length Arithmetic

1500	$a' = a + s$:	
1501	$a' = a - s$:	
1502	$a' = -a + s$:	
1504	$a' = s$:	
1505	$a' = -s$:	
1542	$a' = a \times s$:	
1543	$a' = -a \times s$:	
1556	$s:' = a$	
1565	$a' = -a$	
1566	$a' = \lvert a \rvert$	
1567	$a' = \lvert s: \rvert$	
1576	$a' = a/s$:	

Standardised, Rounded Arithmetic

320	$am' = am + s$	QRE
321	$am' = am - s$	QRE
322	$am' = -am + s$	QRE
360	$am' = a$	NL, QRE
362	$am' = am \times s$	QRE
363	$am' = -am \times s$	QRE
374	$am' = am/s$	QRE DO

Standardised, Unrounded Arithmetic

300	$a' = am + s$	QE
301	$a' = am - s$	QE
302	$a' = -am + s$	QE
340	$a' = a$	NL, QE
342	$a' = am \times s$	QE
343	$a' = -am \times s$	QE

Unstandardised Arithmetic

330	$a' = am + s$	AO
331	$a' = am - s$	AO
332	$a' = -am + s$	AO
341	$a' = a$	NL, E
372	$a' = am \times s$	EAO
373	$a' = -am \times s$	EAO
375	$al' = a/\lvert s \rvert$; $m' =$ remainder	E

Set B-test

150	$bt' = s - ba$
152	$bt' = ba - s$
170	$bt' = n - ba$
172	$bt' = ba - n$

Note: These 4 instructions set the B-Carry Digit

I·C·T

ATLAS COMPUTER
SUMMARIZED PROGRAMMING INFORMATION

INTERNATIONAL COMPUTERS AND TABULATORS LTD
I.C.T. HOUSE, PUTNEY, LONDON. S.W.15

Telephone: PUTney 7272

List CS 384

THE FACILITIES OF ATLAS BASIC LANGUAGE

ROUTINES:

Rn (1 ⩽ n ⩽ 3999)

PARAMETERS:

Routine: An (1 ⩽ n ⩽ 3999)
Preset: Pn (0 ⩽ n ⩽ 99)
Global: Gn (0 ⩽ n ⩽ 3999)

OPERATORS:

Logical shifts: Da, Ua.
Masks: B (block), W (word).
Binary complement: ' (apostrophe).

SEPARATORS:

+, −, X, Q (divide: quotient is always an integer), & (or M), V ('or'), N (non-equivalence).

EXPRESSIONS: 24 bits as address, bits 14 to 20 as Ba or Bm, bits 15 to 20 as a character
GENERAL FORM = ELEMENTS combined with SEPARATORS (interpreted from left to right: e.g. A1 + P2 x 3 = (A1 + P2) x 3).

ELEMENT defined as:

*, any parameter, a constant (a:b.c, Ka.b, Ja, Ya), any EXPRESSION enclosed in brackets, an ELEMENT followed by an OPERATOR.

DIRECTIVES:

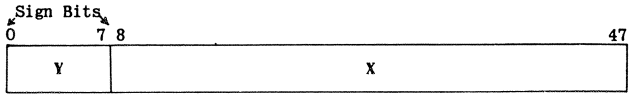
E (enter), ER (read more program), EX (interlude), H (half-word prefix), La (library routine a), Ta (title to output a), F (floating point number), C (store characters), Z (terminate routine), S (6-bit integer prefix), UPa (unset parameter Pa), ½ (output store contents).

FLOATING POINT NUMBERS

Written in Atlas Basic Language [See NOTE 1]	Value Stored in standardized form [See NOTE 2]
±a	±a
±a (⊙ b)	±a x $10^{\pm b}$
±Ka	±Ka
±Ka (:⊙ c)	±Ka x $8^{\pm c}$

NOTES: 1. All signs are obligatory, except when indicated thus ⊙ K precedes octal numbers; others are decimal. 2. To store the same values in substandard form with exponent ±d (or ±Kd) follow the written form with :⊙ d (or :⊙ Kd).

The number z = X x 8^{Y} is stored as follows:



In standardized form the mantissa X of z satisfies either X = 0 (in which case Y = −128 and z = 0), 1/8 ⩽ X < 1, or −1 ⩽ X < −1/8. For any z, standardized or not, Y is an integer in the range −128 ⩽ Y ⩽ 127.

LIBRARY ROUTINES: INPUT AND OUTPUT

Input Routine L100 Entries:-

A1 Read number to Am. QR+.
A2 Read 21-bit integer to B81.
A3 Read character to bits 18 to 23 of B81.
A4 Lose rest of line.
A6 Read text to (b89) onwards.
A7 Read 24-bit integer to B81.
A8 Read 21-bit integer + 3-bit octal fraction to B81.

Output Routine L1 Entries:-

A1 Output *am* (style in B89).
A2 Output b81 (style in B88).
A3 Output character from bits 18 to 23 of B81.
A4 Output NEWLINE.
A5 End record (carriage control character in B87).
A6 Output text from (b89) onwards.

DOCUMENT LAYOUT EXAMPLE

Job/Program Document:-

```
JOB
F163/SURVEY ANALYSIS
INPUT
1 SURVEY RESULTS 1962
OUTPUT
0 ANY 2 BLOCKS
1 LINE PRINTER 3 BLOCKS
COMPUTING 6.5 SECONDS
STORE 25
COMPILER ABL
   |
(Program)
   |
* * * Z
```

Data Document:-

```
DATA
SURVEY RESULTS 1962
   |
 (Data)
   |
* * * Z
```

Alternative Output Peripherals:-
SEVEN-HOLE
FIVE-HOLE
CARDS

NOTATION

Q Accumulator Standardised.
R Accumulator rounded by 'forcing'.
E Exponent overflow may occur.
DO Division overflow occurs if the divisor is zero or unstandardised.
AO Accumulator overflow may occur.
NL L is not cleared initially (as it is in all other basic Accumulator codes).
aq The contents of *Am* following the operation *am'* = *a* QRE.
s: The sum of the contents of store lines *S* and *S*+1.

Magnetic Tape Instructions

1001 Search for section **n** of tape **Ba**.

1002* Read next **K**+1 sections from tape **Ba** to store blocks **P**, **P**+1,..., **P**+**K**.

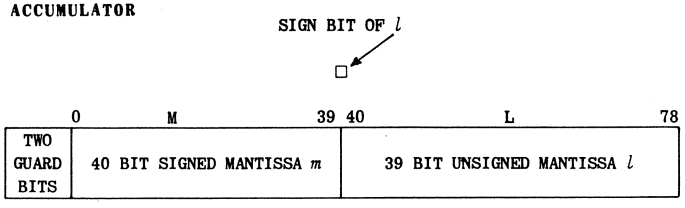
1003* Read previous **K**+1 sections from tape **Ba** to store blocks **P**+**K**,..., **P**+1, **P**.

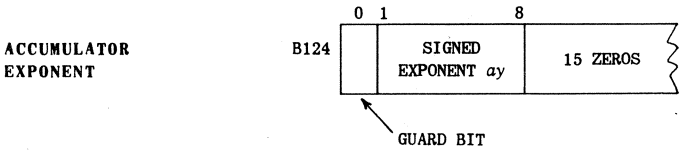
1004* Write store blocks **P**, **P**+1,... **P**+**K** to the next **K**+1 sections of tape **Ba**.

1005* Move tape **Ba** forward **K**+1 sections.
1006* Move tape **Ba** backward **K**+1 sections.

* **n** is written P:0.K

ACCUMULATOR





CONTENTS OF ACCUMULATOR

am = *m* x 8^{ay}
al = *l* x 8^{ay} (includes sign bit of *l*)
a = *ax* x 8^{ay}

Shifts

Circular
105 *ba'* = 8^2*ba* + *s*
125 *ba'* = 8^2*ba* + *n*
143 *ba'* = ½*ba* − *s*
163 *ba'* = ½*ba* − *n*
1342 Right *n* binary places
1343 Left *n* binary places

Subroutine Entry
1100 Enter at *s*, *ba'* = *c* + 1
1101 Enter at *S*, *ba'* = *c* + 1
1102 Enter at *bm*. *ba'* = *c* + 1
1362 Enter at n, link in B90

Arithmetic Using 'n'

1520	*am'* = *am* + *n*	QRE
1521	*am'* = *am* − *n*	QRE
1524	*am'* = *n*, *l'* = 0	Q
1525	*am'* = −*n*, *l'* = 0	Q
1534	*am'* = *n*, *l'* = 0, *ay'* = 12	
1535	*am'* = −*n*, *l'* = 0, *ay'* = 12	
1562	*am'* = *am* x *n*	QRE
1574	*am'* = *am*/*n*	QRE
1575	*am'* = *aq*/*n*	QRE

Input and Output

1050 Select input *n*
1051 *ba'* = selected input
1060 Select output *n*
1061 *ba'* = selected output