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

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 + $\frac{1}{2}$), (BM := BM + 1), (BM := BM − $\frac{1}{2}$), (BM := BM − 1).
vi) Four orders of the form: *IF* BT ≠ 0, BA = N *AND*: (BM := BM + $\frac{1}{2}$), (BM := BM + 1), (BM := BM − $\frac{1}{2}$), (BM := BM − 1).

Note that BA was thought of as the "arithmetic" B-line and BM as the "address-modification" B-line. Adding $\frac{1}{2}$ 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.