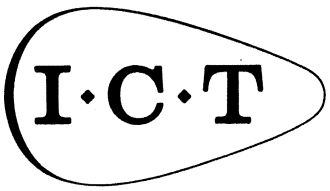
R N Tibbett

INTERNATIONAL COMPUTERS AND TABULATORS LIMITED

THE I.C.T. ATLAS 1 COMPUTER PROGRAMMING MANUAL FOR ATLAS BASIC LANGUAGE (ABL)



68 NEWMAN STREET,
LONDON, W.1.

LIST CS 348A
JANUARY 1965

INTERNATIONAL COMPUTERS AND TABULATORS LIMITED

THE I.C.T. ATLAS 1 COMPUTER

PROGRAMMING MANUAL

FOR

ATLAS BASIC LANGUAGE

(ABL)

68 NEWMAN STREET,
LONDON, W.1.

LIST CS 348A
JANUARY 1965

P R E F A C E

This manual supersedes the manual CS 348, "The Atlas Provisional Programming Manual", January 1963. It provides information for the programming of the Atlas 1 computer in the language known as Atlas Basic Language (ABL). It is a self-contained document, providing sufficient information about the Atlas 1 computer to enable programmers to write and develop programs in ABL without recourse to any other documents about Atlas 1.

The Atlas 1 Computer is the latest result of a long-standing collaboration between Manchester University and Ferranti Ltd. A later version of the computer, known as Atlas 2, has been developed jointly by Cambridge University and Ferranti Ltd. In September 1963 the Computer Department of Ferranti Ltd., was acquired by I.C.T. Ltd., who now manufacture and market the Atlas computers.

Atlas Basic Language (ABL) is a symbolic input language close to "machine language". Each ABL instruction corresponds to one machine instruction, and each part of an ABL instruction to each part of a machine instruction. In its simplest form an ABL instruction consists of four numbers corresponding exactly to the internal machine representation, but extensive facilities are also provided in ABL for the use of a variety of parameters and symbolic expressions which are evaluated by the ABL compiler. ABL also provides a comprehensive system of directives to control the assembly of a complete program. Finally ABL provides facilities for the input, conversion and storage of fixed-point numbers, floating-point numbers and character strings for use by the program.

In this manual no attempt is normally made to differentiate between those facilities which are a basic part of the machine (e.g. the instruction repertoire) and those which are a part of the particular language ABL (e.g. the formats for writing instructions). This is partly because it is impossible to separate them completely - any feature of the machine itself needs a language in which to describe it, and in this case that language is ABL - and partly because it is not normally necessary or helpful for a programmer to be conscious of the distinction. However, inasmuch as certain facilities of the machine itself are described here, parts of this manual are relevant and interesting to users of other Atlas programming languages. In particular, Chapter 10 "Preparing a Complete Program" applies to all Atlas languages.

A word must be said about the enumeration of binary digits: throughout this manual the convention adopted is to number bits as 0, 1, 2 ...., starting always with bit number 0 at the more significant end. This convention differs from that used in documents on the Supervisor and in engineering documents, in which only the accumulator is numbered as here, and in all other cases bit 0 is the least-significant bit.

C O N T E N T S

Chapter 1

AUTOMATIC DIGITAL COMPUTERS

1.1 Introduction

Atlas 1 is a very fast, automatic digital computer with built-in time-sharing facilities enabling a considerable number of problems to be processed simultaneously. This manual is intended for those who will prepare programs giving the machine detailed instructions for the step-by-step solution of individual problems. It is likely that they will have some previous knowledge of computers: should the ideas outlined in this chapter be unfamiliar, the reader is advised to consult an introductory text for further clarification.

1.2 Electronic Computers

The application of electronics has led to the development of the modern high-speed computer; we must distinguish two types of electronic computers:-

Analogue computers represent quantities by some analogous physical quantity and solve problems by working with an actual physical model obeying the desired theoretical equations. Since the quantities involved can be evaluated only by measurement, the attainable precision is necessarily limited. The slide-rule is a familiar example of an analogue computer, using lengths to represent the logarithms of numbers.

Digital computers operate upon numbers in some coded-digit form and make use of standard computational techniques to obtain direct numerical solutions to problems. By increasing the number of digits with which numbers are represented in the machine, the precision may be extended without limit. A desk calculator is a simple form of digital computer.

1.2.1 Digital Computers.

As a more complete form of digital computer, we may consider the combination of a desk calculator and its operator as a single computing unit; this will enable us to introduce the essential features of a typical digital computer:-

(a) *Input and Output.* This is one of the roles performed by the operator of a desk calculator, who must set up numbers in the machine before they can be operated on and also read results from the machine, recording them elsewhere. For an electronic computer, information is transferred to and from the machine by automatic equipment.

(b) *Control.* Here again, it is the operator who must control the sequence of operations on a desk calculator. An automatic computer is controlled by a program consisting of a sequence of detailed instructions in coded form. In the important case of a stored-program computer, the whole

program is stored within the machine before any of it is obeyed; the speed of the computation is then not restricted to that of the input devices.

(c) *Arithmetic Unit.* The mechanism of a desk calculator carries out the individual operations in accordance with the key depressed. Similarly, the arithmetic unit of an electronic computer performs the functions called for by the successive instructions in the program. There will usually be included an accumulator in which the result of each step first appears, similar to the long register on the carriage of a desk machine.

(d) *Storage.* The keyboard and certain other registers of a desk calculator constitute a working store, insofar as the mechanism of the arithmetic unit is able to operate directly upon numbers contained in these registers. An electronic computer commonly has a working store capable of holding several thousand numbers. At any time each of these numbers is immediately available to the arithmetic unit, and so one speaks of a "random access" or "fast" store.

This type of storage is relatively expensive and so if still larger amounts of storage are required this is normally provided by some cheaper form of "backing" store. This will inevitably involve a longer access time, but, when required for computation, data can be transferred in blocks of several hundred numbers at once from the backing store to the working store.

1.3 Addresses

Each of the locations in the backing store and in the computing store is assigned an address. The address is a number, and it is important to distinguish the number which is the address of a location from the number which is contained in that location. As a means of distinction, we shall denote addresses by capital letters and contents by small letters and will assume, for example, that the number s is the content of the location whose address is S. In certain contexts we shall find it necessary to use the notation C(S) as an alternative to s. (Note that C(S + 1) is not the same as s + 1; the notation S^* is sometimes used to denote S + 1, so that $s^* = C(S + 1)$). The contents of S after an operation will be written s′, so that, for example, the equation

$$s' = s + b$$

denotes the operation of adding the number b to the contents of S.

1.4 Instruction Code

We have so far spoken of the contents of store locations as numbers, but they may also be instructions in coded form. Both numbers and coded instructions may be referred to as "words"; it is for the programmer to ensure that no attempt is made to interpret one type of word as the other. An instruction word will usually contain one or more addresses to specify the data to be operated on; there will also be a coded number specifying the operation to be performed. The correspondence between the elementary operations which may be directly carried out in the arithmetic unit and the code numbers which control them constitutes the "order-code" or "instruction-code" of the computer.

1.5 Jump Instructions

Instructions will generally be obeyed in the same sequential order as their addresses occur in the store. However, it is possible to specify by an instruction the address of the next instruction to be obeyed, and hence one may arrange to "jump" out of sequence to an instruction at any desired address. Instructions are provided to make such a jump conditional upon the sign of certain numbers in the machine; these conditional jumps provide the ability to take elementary decisions.

1.5.1 Looping.

By repeatedly jumping back to the same instruction, the computer can be made to obey a "loop" of instructions over and over again; this is a vital feature of high-speed computing, making it possible for a program of reasonable length to control the machine for relatively long periods of time.

1.5.2 Modification.

The utility of computing loops is greatly enhanced by the facility known as "modification", whereby different data is processed in each iteration of the loop. The store address of the number to be operated on is modified before use by the addition of an index stored in one of several special registers. Thus, if the index is increased by unity before each successive iteration, one may operate upon a list of numbers held in the store, and so, for example, form their sun or average.

1.6 Binary Numbers

The storage mechanisms used in electronic digital computers are normally made up of devices having two possible states, just as a switch may be either OFF or ON. If we associate with these two states the symbols 0 and 1 respectively, we are led to adopt the binary number system: the string of 0's and 1's stored on a row of two-state devices is interpreted as a succession of coefficients of powers of two in a polynomial. This is exactly analogous to the conventional decimal notation based on powers of ten. The binary digits 0 and 1 are commonly referred to as "bits".

1.6.1 Negative Numbers.

If a desk calculator is used to subtract some small numbers from zero, the result is characterised by a string of 9's at the more-significant end; the same operation with binary numbers produces a string of 1's. We have here a naturally occurring representation of negative numbers, which is made unambiguous by restricting the range of positive numbers to those having 0 as their most-significant digit. This then becomes a sign digit, and the presence of a 1 in this position will indicate a negative number whose actual value is obtainable by subtracting 2^r, r being the number of bits in the number.

Chapter 2

THE ATLAS 1 COMPUTER

2.1 A General Description of Atlas 1

The main parts of Atlas 1 consist of:-

(a) The control unit
(b) Two arithmetic units
(c) The Supervisor
(d) The storage system
(e) Input and output devices.

2.1.1 The control unit produces in correct sequence the control signals necessary to call for an instruction, to decode it, to modify the address, to obtain the operand from store and to perform the arithmetic operation. The address of the current instruction is held in one of three special index registers, called control registers. Before the current instruction is decoded the contents of the control register in use are increased by one in anticipation of the next instruction.

2.1.2 Arithmetic is mainly done in the accumulator, which is a double-length floating-point register. The accumulator arithmetic unit can obey 49 different instructions, including different types of addition, subtraction, multiplication and division, transfers, tests, shifts etc.

For small integer arithmetic, modification and counting, there are also 128 index registers. These are known as B-registers and have their own arithmetic unit. The B-register arithmetic unit can obey 51 different instructions, including addition, subtraction, logical operations, shifts, tests, counts etc.

2.1.3 Any peripheral transfer on Atlas 1 has only to be initiated, after which it proceeds independently, leaving the central computer free to continue obeying instructions. Suppose there to be only one program in the computer, which might be reading characters from the tape reader and sorting them on magnetic tape, one per word in units of 512 words. The tape reader operates at 300 characters per second and so reads a character once every 3,333 microseconds (μs); a magnetic tape transfer of 512 words takes 46 milliseconds (ms). Between reading characters it would be possible for the central computer to obey about 2,000 instructions, and while executing a magnetic tape transfer, about 50,000 arithmetic instructions could be obeyed. If the computer were to be idle during transfers because the information was wanted immediately (in the next instruction) obviously its utilisation would be very inefficient. Note that if the slow peripheral equipments could always transfer information at the rate required by the central computer for any problem no difficulty would arise. As they cannot, special operating methods have to be used. The method on Atlas 1 is to have a special program called the Supervisor which controls the flow of programs through the computer. The Supervisor is simply a program which attempts to run Atlas in an efficient way, that is, it tries to keep all

the parts of Atlas busy. To achieve this, it shares computing time between programs, and manages all peripheral transfers, including input and output as well as drum and magnetic tape transfers. The Supervisor is described in more detail later; at this stage it must be remarked that although it is not part of the "hardware" in the sense that the core store is, it is the most important single feature of Atlas and quite indispensable.

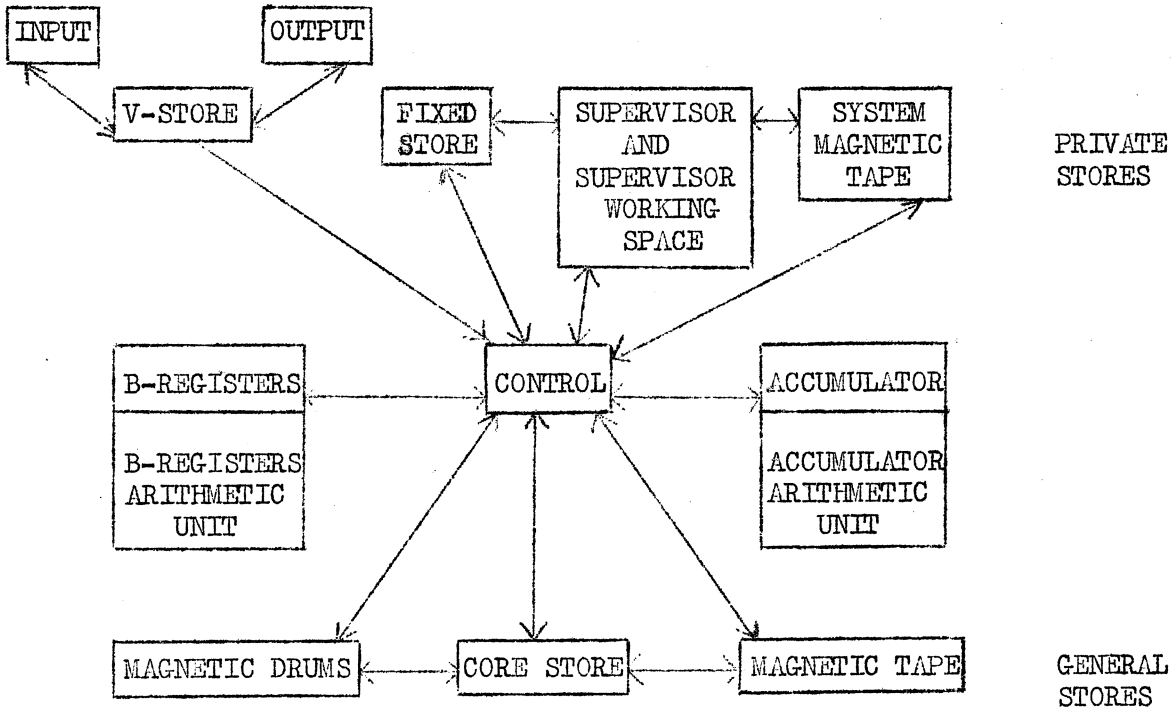
2.1.4 The main store on Atlas 1 consists of core store in units of 8192 words and magnetic drum store in units of 24,576 words. A total of about one million words are directly addressable. Up to 32 magnetic tapes can also be used as a backing store. The main store can consist of core store and drums in any proportion. The programmer treats the store as if it were all core store; he will in fact not know what parts of his program are in the core store at any one time. The Supervisor manages drum transfers behind the scenes as required, and attempts to keep the most used blocks of program always in the core store, by means of a "drum learning" program. This idea of a fast and a slow store appearing as a single fast store is called the "one-level-store" concept.

The Supervisor occupies most of a special store called the fixed store, and some blocks of the main store. The fixed store is a "read only" store in multiples of 4096 words where binary ones and zeros are represented by ferrite and copper slugs in a wire mesh. It is used to represent permanent programs which will not be changed, and besides the Supervisor it holds the "extra-codes". These are extensions to the basic instructions, described later. For working space the Supervisor has a subsidiary core store of 1024 words, in which it keeps parameters associated with programs in the machine, peripherals, etc. The Supervisor also uses three magnetic tape units, called "system tapes" as part of its input/output organisation. All the stores used by the Supervisor are known as private store, and it is not possible for ordinary programs to interfere with them.

2.1.5 A large variety of input and output devices are allowed on Atlas 1. Each type of device is connected to the central computer via the peripheral co-ordinator, which contains buffer registers and information registers concerned with the transfer of data. These registers, which are at different places in the computer, and also some registers connected with the arithmetic units, are collectively referred to as the V-store. They are only accessible to the Supervisor, and form part of the private store. The peripheral co-ordinator allows the following types of input/output equipments to be attached.

ICT TR5 paper tape readers	300 ch/sec
ICT TR7 paper tape readers	1000 ch/sec
Teletype paper tape punches	110 ch/sec
Creed 3000 paper tape punches	300 ch/sec
Creed 75 teleprinters	10 ch/sec
ICT card readers	600 cards/min
ICT 582 card punches	100 cards/min
Anelex line printers	1000 lines/min
Graphical outputs	
Clock	

The following diagram shows the component parts of Atlas 1:-



2.2 The Main Store

Within the programmers store, registers are numbered consecutively from 0 upwards. Registers are arranged in blocks of 512 words called pages, and transfers between the core store and drums or magnetic tape take place in units of 512-word blocks. To increase the computing speed, the core store is divided into stacks, each with its own read/write circuitry. These are known as the even and odd stacks. Each is of 4096 words and they are arranged so that words 0, 2, 4, 6 .... are in the even stack, words 1, 3, 5 .... in the odd. Instructions are always called for by the control unit in pairs consisting of an even and the next odd instruction, although sometimes only one of these instructions may be wanted.

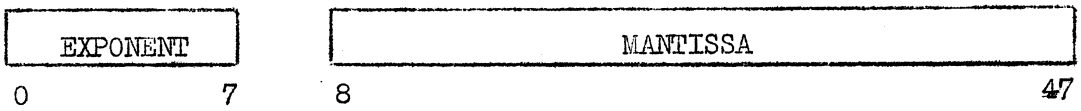
Wherever it is safe to do so, as soon as the control unit has decoded the odd address instruction it sends for the next pair. Thus, there is overlap between the execution of instructions and it is in fact possible for the computer to be in different stages of execution of up to five instructions. As a consequence, the first instruction in a pair must not alter the second, and the second must not alter either of the next pair of instructions. Almost always, if this were done, the unaltered versions would be obeyed, but because of "interrupts" which occur at frequent intervals and which effectively insert instructions between program instructions sometimes the altered versions would be obeyed.

2.3 Storage of information in Atlas 1

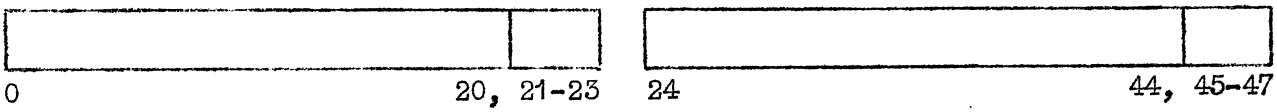
An Atlas 1 register, or word, contains 48 binary digits. These are conventionally numbered from 0 (most-significant) to 47 (least-significant).

A single word can be used to represent any of the following:-

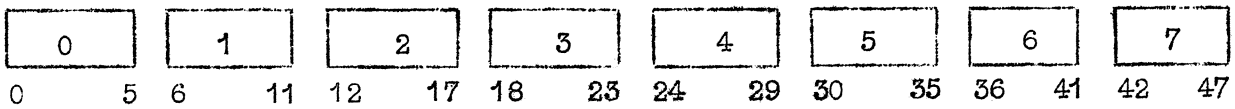
(a) A 48-bit floating or fixed point number, with the 8 most-significant bits representing the exponent and the other 40 the mantissa.



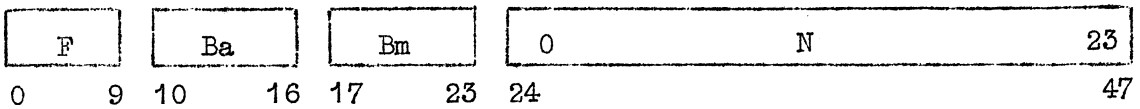
(b) Two 24-bit half-word numbers. These are taken usually as 21-bit signed integers in digits 0-20, with an octal fraction in digits 21-23.



(c) Eight 6-bit characters.



(d) An instruction, specifying a function F (most-significant 10 bits), two index registers Ba and Bm (7 bits each) and an address N (least-significant 24 bits).



Throughout this manual the binary digits of a word are numbered from the most-significant end, starting with bit 0. The engineers' numbering system is the reverse of this, and is used in some documents describing basic programs such as the Supervisor and Engineers' programs: in these documents bit 0 is the least-significant bit.

2.4 Instructions in the machine

N can be taken as an operand directly in some instructions, in which case it is known as n. When N is used as an address, bits 1-20 specify a word in the main store, bit 21 specifies a half-word address and bits 22 and 23 specify a character address within a half-word (for the moment renumbering the bits of N as 0-23). Instructions ignore irrelevant digits in the address. Thus an instruction involving half-words ignores digits 22 and 23 in the address. Digits 12-20 specify the word address within a block, and digits 1-11 specify the block.

Thus a main store address consists of

Block number	Word number in block	Half-word address	Character address
1 11	12 20	21	22 23

For the moment we shall write N as a decimal number and an octal fraction, separated by a point. Then 16.0 is the first half-word in word 16 (i.e. digits 0-23) and 16.4 is the less-significant half-word (digits 24-47), in an instruction which uses a half-word. In instructions for handling characters, the characters in word 16 are 16.0, 16.1, 16.2, ... 16.7; where 16.0 is digits 0-5, 16.1 is 6-11 etc., with 16.7 being 42-47.

The 10-bit function F is written as a single binary digit (f0) followed by three octal digits. For all basic functions f0 is zero, and may be omitted in the written form. The basic functions fall into three categories, depending on the first octal digit (f1 to f3); if this digit is 1, the function is a B-register instruction (B-code), if it is 2, the function is a Test instruction, and if it is 3, the function is an Accumulator instruction (A-code). These instructions are described in detail in later chapters.

The basic order-code is extended by the provision of "extracodes". These are routines, written in basic instructions, which are positioned in the fixed store and which carry out many operations usually managed by a subroutine library. Extracodes are recognised by having $f0 = 1$. When this is encountered, main control is halted and the program continues under a special "extracode" control at an address in the fixed store. The final instruction in this routine (which is recognised by $f1 = f3 = 1$ and obeyed as if $f1 = 0$) has the effect of returning to main control at the program's next instruction. Thus extracodes are subroutines with automatic entry and exit; to the programmer they appear as one instruction.

For example,

Function	113	is a B-code used to store a B-register
	234	is a test-code, transferring n to Ba if the contents of the accumulator are zero
	374	is an A-code, dividing the accumulator by the contents of a store address
	1250	is an extracode, to unpack a 6-bit character from store and place it in Ba.

The B-registers are used in different ways depending on the type of instruction. There are 128 B-registers, B0 to B127, most of them of 24 bits. B120-127 are special purpose B-registers, the rest are general purpose. In A-codes, the contents of Ba and the contents of Bm are added to N to give a modified address. That is, the address S used in the instruction is N+ba+bm. In most B-codes, ba is used as an operand, so only bm is used to modify N, and then S = N+bm. (There are two exceptions, in which bm is also an operand, so no modification takes place.) For most test codes, bm is used as a further operand; where it is not it is used to modify N. In extracode instructions, Ba and Bm are treated in a special way. For the present we shall write Ba and Bm as decimal numbers in the range 0 to 127. B-register arithmetic is described in Chapter 4.

2.5 The written form of Instructions

Instructions are written on one line, with the component parts separated:

	F	Ba	Bm	S	
Thus	113	1	0	16.4	stores b1 in the half word at address 16.4 (B0 is always zero)

If b2 = 1.4 say, then

	113	1	2	16.4	stores b1 at 18.0

If b1 = 3.0 say, then with b2 = 1.4,

	374	1	2	16.4	divides the accumulator by the number at 21.0

Instructions are read into Atlas through the media of punched paper tape, either of 5-track or 7-track width, and 80-column punched cards. 7-track paper tape (the most common input medium) is prepared on a Flexowriter, 5-track paper tape on a Creed teleprinter and cards on a card punch. These three equipments have slightly different sets of characters. In the punched form, the parts of an instruction are punched as written and separated by "multiple spaces" or commas. Multiple space is two or more spaces on the teleprinter, or a tabulate (TAB) on the Flexowriter.

2.6 The Full Range of Atlas 1 Addresses

As explained in 2.4, address bits 1 to 11 represent the number of 512 word main store block to which that address refers, so that an Atlas 1 main store address refers to one of 2048 blocks. In octal (for the J notation see 4.3) these block addresses are J0000, J0001, J0002,...., J3777 and in their decimal representation (see 5.6) 0:, 1:, 2:,..., 2047:. The ABL compiler and the program it is compiling share the same range of block numbers, with ABL occupying blocks in the range J3 to J34. Consequently to avoid over-writing itself ABL will refuse to compile program into any of the 256 blocks 1536: to 1791: Once the program is compiled and ABL has withdrawn from the store these blocks become available again and can be used as working space by the program.

The remainder of the main store block numbers J34 to J4 are illegal. ABL will not store program in block J3 or above and the Supervisor will fault the program if the compiled program attempts to refer to block J34 or above.

In fixed store and private store addresses bit 0 is a 1. There are therefore another 2048 blocks that can be addressed and they have octal addresses J4000 to J7777. The first 16 of these (J4 to J4017) are the block numbers of the fixed store and may be referred to by the programmer if he wishes, although there is generally no reason why he should do so. The block numbers J4020 to J4777 are also quite legal. In effect these block numbers refer to 31 consecutively stored copies of the fixed store. For example either of the instructions

```
101    3    0    J4017777
101    3    0    -1J5
```

would place the same half-word in B3, namely the left half of the last word of the fixed store. In other words addresses in the range J4 to J5 are in effect masked with J40177777 before execution (for a definition of masking, or what is the same thing collating, see 4.3)

The private store block addresses J5 through J7777 are completely forbidden to the ordinary programmer. These addresses can be referred to on extracode control. However it is impossible for the programmer to force an extracode to refer to the private store. He is prevented by faulting an extracode instruction in which the modified address is in the private store but the unmodified address is not. Thus for example a program containing the instruction

```
1730    0    0    J6    am' = sin (J6)
```

would be thrown off because the first instruction of the extracode routine is

```
324    0    119    0    am' = (b119)
```

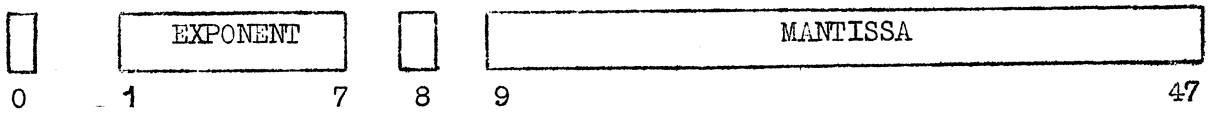
(B119 will contain the address J6 upon entry to the extracode. See 7.2.)

Chapter 3

THE ACCUMULATOR

3.1 Floating-point numbers

When a 48-bit word is used to represent a number it is divided into 40 bits for a signed mantissa x and 8 bits for a signed exponent y.



Digit 0 is the exponent sign, digit 8 the mantissa sign.

The exponent is an octal one, so the number represented has the value

$$x.8^y$$

The exponent is an integer and lies in the range

$$-128 \leq y \leq 127$$

The mantissa is a fraction, with the binary point taken to be after the sign digit, so it lies in the range

$$-1 \leq x \leq 1 - 2^{-39}$$

With this system it is possible to represent positive or negative numbers whose magnitudes are approximately in the range 10^{-116} to 10^{113}.

Example: One way of representing the number 8 is by a mantissa of $+\frac{1}{8}$ and an exponent of +2, i.e. as $\frac{1}{8} \times 8^2$. Thus:-

Exponent	Mantissa
0 0000010	0.001000000..... 000

For clarity, when this is written in binary digits the exponent is spaced out away from the mantissa, the sign digits are slightly separated from the numbers, and the mantissa binary point is shown. This convention will be used in future without more explanation.

3.2 The Accumulator

Floating-point arithmetic is all done in the full accumulator (A) and in order to describe the arithmetic it is first necessary to introduce the accumulator.

The accumulator has an 8-bit signed exponent a_y and a double-length mantissa a_x, of 78-bits and a sign bit. The mantissa A_x is regarded as being divided into M and L, M being the sign digit with the 39 more-significant digits, and L being the 39 less-significant digits. Associated with L is a sign digit called Ls. Ls is situated between M and L, and it is usually irrelevant; that is, arithmetic in the accumulator proceeds as if Ls is not present.

The accumulator is sometimes regarded as holding two single-length floating-point numbers. These are called Am and Al.

Am consists of A_y and M

Al consists of A_y and L with Ls

There are a further two digits to the left of the sign of Am. These are the guard digits and their function is explained later. It is not possible to transfer numbers into or out of the guard digits, and before an accumulator operation they are normally copies of the mantissa sign digit.

The exponent a_y is held in the special B-register B124, which consists only of nine digits at its most-significant end, the other digits 9-23 being zero. The exponent is held in digits 1-8. Digit 0 is used as a guard digit, to detect if the number departs from the range $-128 \leq a_y \leq 127$. Digit 1 is the sign digit and normally digit 0 is the same as digit 1. If the two digits are different, then the exponent has gone outside its permitted range. If $d1 = 1$ and $d0 = 0$, then $a_y > 127$; if $a_y > 127$ at the end of an accumulator operation, then Exponent Overflow is said to have occurred and in some cases the program is monitored (see section 11.1.1). If $d1 = 0$ and $d0 = 1$, then $a_y < -128$; if $a_y < -128$ at the end of an accumulator operation, then Exponent Underflow is said to have occurred and normally the contents of the accumulator are automatically replaced by "standard floating-point zero" (see section 3.3).

Diagram of the whole accumulator

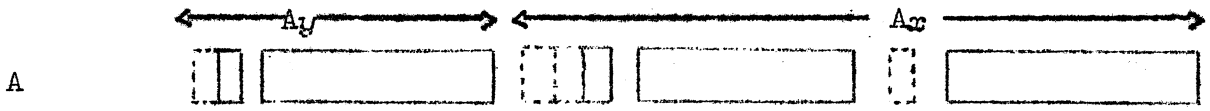
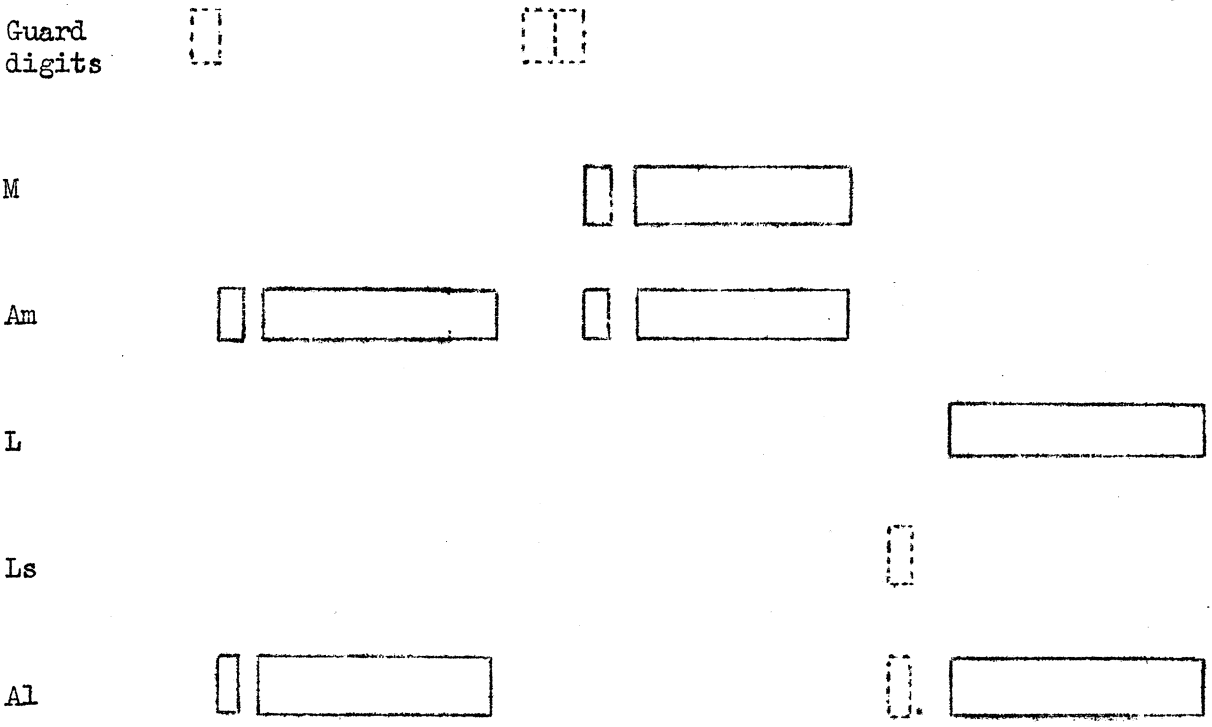


Diagram showing component parts of the accumulator



Note that because of the guard digit, the position of the exponent as held in B124, (digits 1-8) is different from the position of the exponent for a number held in the store, which would be in digits 0-7.

Accumulator instructions usually involve arithmetic on two numbers, one of these being in the accumulator and the other taken from the store. Most accumulator instructions deal with standardised numbers, so these will now be described.

3.3 Standardised numbers

The representation of a floating-point number is not unique. For example $\frac{1}{2} = \frac{1}{2} \times 8^0$ or $\frac{1}{16} \times 8^1$ or $\frac{1}{128} \times 8^2$ etc. so any of the forms below represent $\frac{1}{2}$.

Exponent	Mantissa	Value $= \frac{1}{2}$
0 0000000	000.100000000....	$\frac{1}{2} \times 8^0$
0 0000001	000.000100000....	$\frac{1}{16} \times 8^1$
0 0000010	000.000000100....	$\frac{1}{128} \times 8^2$

(Note that the two guard digits are shown to the left of the mantissa sign digit.

It will be noticed that the value of a number is unchanged if unity is added to or subtracted from the exponent every time the mantissa is shifted 'octally' (by 3 bits) down or up, respectively.

The optimum form of storage of a number in a binary floating-point system is that in which there are the minimum number of 0's (assuming a positive number) between the binary point and the most significant 1 of the fractional part of the number. This enables the maximum number of fractional digits to contribute to the accuracy of the representation.

As Atlas has an octal exponent, shifts of the mantissa may only be by 3 bits at a time, so it is not possible to specify that there should be no 0's immediately after the point. It is possible to specify a maximum of two, and this is known as the standardised condition. A corresponding convention for a minimum number of 1's holds for negative numbers.

An Atlas standardised number is therefore such that the mantissa lies in the range:-

$$\frac{1}{8} \leq x < 1 \text{ or } -1 \leq x < -\frac{1}{8}$$

and it may be necessary to shift the mantissa of the number resulting from an operation up or down, adjusting the exponent accordingly, to achieve this.

If a number is not standard it is either 'substandard' or 'superstandard'. A substandard number is one such that $-\frac{1}{8} \leq x < \frac{1}{8}$. The three most significant digits of ax will be the same as the sign digit (and guard digits), so the number can be standardised by octal shifts up and adjustment of the exponent. A superstandard number is one in which the mantissa has overflowed into the sign and guard digit positions, i.e. it is ≥ 1 or < -1 and it is detected by the guard digits not being the same as the sign digit. To standardise such a number, a single octal shift down is required, with the addition of unity to ay.

Example: the addition of $\frac{5}{8} + \frac{3}{8}$ with a standardising instruction gives the correct result of $+1$ standardised; though immediately after the addition the number is super-standard.

$\frac{5}{8} = \frac{5}{8} \times 8^0$		0 0000000	000.101000..
$+\frac{3}{8} = \frac{3}{8} \times 8^0$		0 0000000	000.011000..
$= 1 \equiv 1 \times 8^0$	(superstandard)	0 0000000	001.000000..
$1 \equiv \frac{1}{8} \times 8^1$	(standardised)	0 0000001	000.001000..

The number zero is a special case. Floating-point zero is represented by a mantissa of 0 and an exponent of -128, and this is regarded as a standardised number. Zero is specially looked for when a number is to be standardised, and, if found, no shifting takes place and the exponent is immediately set to to -128.

3.4 Fixed-point numbers

For some purposes it is inconvenient to deal with floating-point numbers, and accumulator instructions are provided which do not standardise the results of operations. Using these instructions it is possible to regard the binary point as being anywhere it is desired; for example, at the least significant end of M (which means regarding numbers in Am as integers between the range of -2^{39} and $2^{39} - 1$). In this type of arithmetic, the exponents of the numbers must all be the same, and are commonly zero. If they are the same but non-zero, adjustments are required when multiplications and divisions are performed.

Superstandard numbers cannot be automatically corrected in fixed-point working, so if they occur, a special Accumulator Overflow digit (A0) is set, and this digit can be inspected by the program.

For example, if the point is taken at the least significant end of M, then the numbers in the last example, $\frac{5}{8}$ and $\frac{3}{8}$, now have the values $(2^{38} + 2^{36})$ and $(2^{37} + 2^{36})$. Adding them gives a superstandard answer and sets A0.

3.5 Rounding

Most accumulator instructions operate on the two numbers am and s, leaving the answer in A. This answer may be an operand in the next instruction with only am used, the digits in L being cleared before the operation. The process of cutting off these less-significant 39 digits of the answer is called truncation, and this introduces an error each time which could quickly become significant. Rounding is the name given to the process of compensating the answer so as to minimise the effect of truncation.

One method of rounding is to force a 1 (i.e. the logical "OR" operation) in the least significant digit of M if L is non-zero. If L is zero, no forcing takes place. In a sequence of accumulator instructions, the average error introduced by this method is zero, so no bias is introduced.

Further, single-length integer arithmetic in Am can be carried out exactly without any unwanted rounding, as long as numbers never extend into L. The abbreviation R is used in describing some instruction to signify rounding in this way. Notice that L is not changed by the process of rounding.

An instruction is also provided to give rounding by adding a one to the least-significant digit of M if the most-significant digit of L is one. Again, the digits of L are left unchanged. This type of rounding is referred to as R+ Rounding. It is sometimes preferred to the method just described as less accuracy is lost, but is slightly biased in that the rounding is always upwards in the halfway case of L being a binary one followed by a string of zeros.

3.6 Floating-point operations

Before two numbers in floating-point form can be added the numbers must be shifted relative to each other so that digits which have the same significance can be added together, i.e. one number is shifted octally until the two exponents are equal.

In Atlas, the number which has the smaller exponent is the one that is shifted, and it is shifted down into L, irrespective of whether it is the number in A or the number in S. There are four addition instructions, and of these, three clear L before this shifting takes place, so the addition is between am and s.

A_y is then set equal to the larger of the two exponents and the arguments are added together. The addition takes place over the 42 digits of Am with its guard digits. L remains unchanged during this stage.

After the addition, the accumulator may be standardised and rounded, standardised but not rounded, or left unstandardised and unrounded, depending on the instruction. The instructions which standardise check that exponent overflow has not occurred, the instructions which do not standardise look for accumulator overflow.

The addition instruction which does not clear L first is used mainly in double-length arithmetic. To work correctly, the exponent of s must be equal to or greater than a_y, so that the contents of the accumulator are shifted down. If $s_y < a_y$, then s would be shifted down into L. overwriting the original contents of L.

3.7 Standardising and rounding accumulator instructions

We are now in a position to introduce some accumulator instructions.

Function	Description	Notation
320	Clear L; add the contents of S to the contents of Am; standardise the result, round by forcing a 1 into the least-significant bit of M if l is non-zero and check for exponent overflow.	am' = am + s QRE
321	As 320 but subtract s	am' = am - s QRE
322	As 320 but first negate the contents of A	am' = -am + s QRE
324	Transfer the floating point number in S to Am and standardise it	am' = s Q
325	As 324 but transfer negatively and check for exponent overflow	am' = -s QE
356	Store am at S, leaving the contents of A unchanged	s' = am
362	Clear L, multiply am by s, standardise, round and check for exponent overflow	am' = am.s QRE
363	As 362 but multiply negatively	am' = -am.s QRE
374	Divide am by s, leaving the quotient standardised and rounded in Am, with l' = 0. Check for exponent overflow and division overflow. Both am and s must be standardised numbers	am' = am / s l' = 0 QRE DO

The above are the most commonly used accumulator instructions, all but the 356 instruction leaving a standardised rounded number in Am.

Example: given four standardised floating-point numbers a, b, c, d in the first four locations of store, replace a by $(a-b+c)/d^2$

324	0	0	3	put d into Am
362	0	0	3	form d^2
356	0	0	4	store in location 4
324	0	0	0	a into Am
321	0	0	1	subtract b
320	0	0	2	add c
374	0	0	4	divide by d^2
356	0	0	0	store answer in word 0

Note that register 4 is used as working space, and that Ba and Bm are zero in every instruction.

3.8 The timing of instructions

In general, it is not possible to state exactly how much time any instruction will take, because this partly depends on the instructions before and after it. However, in a sequence of additions, subtractions or transfers each unmodified instruction takes about 1.6 μs. If modified by bm, the time is 2.0 μs, and if modified by bm and ba the time is 2.5 μs. The times for multiplication and division are 6.0 μs and about 20 μs respectively. The division time depends on the numbers involved. However, another limitation on the speed is the time needed after reading a number from a stack of core store before another number can be read from it. This is known as the cycle time of the store and is 2 μs. As alternate addresses are in the even and odd stacks, (see section 2.2) if operands are used in sequence then the cycle time is not a limitation. In the example just quoted, the sixth instruction would take 2 μs because the next instruction cannot read its operand until this time is up. In the fifth and sixth instructions the operands are in different stacks so the subtraction would take 1.6 μs. B-register instructions, as they use a different arithmetic unit, can continue while the accumulator is busy. During a division instruction, for example, three or four B-instructions might be completed, effectively taking no time at all.

3.9 Some fixed-point Instructions

Fixed-point working has been introduced in section 3.4

Function	Description	Notation
330	Clear L, add s to am, leaving the result in A. Do not standardise or round but check for accumulator overflow	$a' = am + s$ A0
331	As 330 but subtract s	$a' = am - s$ A0
332	As 330 but negate am before the addition	$a' = -am + s$ A0

In these instructions, if the exponents are the same, no shifting down of either of the numbers into L takes place, so the answer will be in Am.

334	Transfer s into Am without standardising	$am' = s$
335	Transfer the number in S negatively into Am without standardising, and check for accumulator overflow	$am' = -s$ A0

In 335 A0 would be set if s is just a one in the sign position (which we will call -1.0) as negating this sets the sign digit different from the guard digits.

364	Shift the mantissa up one octal place, leaving the exponent unchanged. Accumulator overflow can occur, but no check is made	$ax' = 8ax$ $ay' = ay$
365	Shift ax down one octal place, leaving ay unchanged	$ax' = \frac{1}{8} ax$ $ay' = ay$

The above two instructions are of course also useful in floating-point arithmetic, to multiply or divide by 8. Extracodes are provided to shift any specified number of places up or down.

Chapter 4

THE B-REGISTERS

The index registers, or short accumulators, are known as B-registers on Atlas. There are 128 B-registers. 120 of these are constructed from a very fast core store and are used for general purposes. The remaining 8 are "flip-flop" registers, used for special purposes. The B-registers have addresses from zero to 127, and are referred to by prefixing the address with the letter B or b. Thus B61 is B-register with address 61 and b61 is the contents of this B-register.

4.1 General Purpose B-registers

These are the first 120 registers B0 to B119. Each consists of 24 bits of which the most significant (digit 0) is taken to be the sign digit. For purposes of modification and counting, integers are held one octal place up from the least-significant end of a word, so the binary point is assumed to lie between digits 20 and 21. Thus a B-register can hold a 21-bit signed integer with an octal fraction.

The contents of a B-register are usually written as a signed decimal number and an octal fraction, the two parts separated by a point. Thus 15.3, -2.7, 6.0 etc. When the octal fraction is zero it is usually omitted, the point of course also being omitted. The number in a B-register can take any value in the range -2^{20} to $+2^{20}-0.1$ inclusive. An exception is B0, whose contents are always zero.

Programmers are warned to refer to section 4.10 before using B81-119, whose contents are liable to be overwritten.

The basic instructions which operate on B-registers have already been mentioned. They are known as B-codes and B-Test codes, and will now be described in detail.

4.2 Arithmetic Operations

In the following instructions, arithmetic takes place between a 24-bit number in the store and a number in the Ba B-register. The address is modified by the contents of Bm, the other B-register, to give the half-word store address S of the operand. The contents of this are known as s.

Function	Description	Notation
101	Transfer s to Ba	ba' = s
103	Transfer s negatively into Ba	ba' = -s
104	Add s to the contents of Ba	ba' = ba + s
102	Subtract s from ba	ba' = ba - s
100	Negate ba and add s to it	ba' = -ba + s
111	Store ba negatively at S	s' = -ba
113	Store ba at S	s' = ba
114	Add ba into the contents of S	s' = s + ba
112	Negate the contents of S and add ba	s' = -s + ba
110	Subtract ba from the contents of S	s' = s - ba

Example: If m is held at address 5.4 and n at 6.4, place 3m - 2n at 6, using B1 as working space.

101	1	0	5.4	transfer m to B1
113	1	0	6	store m in half-word 6.0
102	1	0	6.4	m - n in B1
114	1	0	6	2m - n in 6
114	1	0	6	3m - 2n in 6

There are instructions provided which use the address as an operand. That is, N + bm, instead of giving the address of the operand, is used directly as a number n.

Function	Description	Notation
121	Place n in Ba	ba' = n
123	Place a negatively in Ba	ba' = -n
124	Add n to the contents of Ba	ba' = ba + n
122	Subtract n from ba	ba' = ba - n
120	Negate ba and add n	ba' = -ba + n

Examples:

1. Replace the number m in 17.4 by the number 64 - m

121	1	0	64	put 64 into B1
112	1	0	17.4	-m + 64 in 17.4

2. Copy the number in B2 into B3

121	3	2	0	$b3' = 0 + b2$

This has the effect of placing 0, modified by the contents of B2, into B3 i.e. places b2 into B3.

3. Similarly, the number in B4, for example, can be doubled by the instruction

124	4	4	0	$b4' = b4 + b4 + 0$

4.3 Logical Operations

Three types of logical operations can be carried out in B-register arithmetic. These are collating, non-equivalencing and "OR" ing. They operate on pairs of numbers simply as strings of binary digits, and form a third number from the pair.

The collate operation, which is denoted by &, gives a 1 in the result in every position where both numbers have a 1, and 0's elsewhere. For example, the result of collating

```
    00010110
with 01110100
is   00010100
```

The non-equivalence operation, denoted by $\not\equiv$ gives a 1 in the positions in which the corresponding digits of the two numbers differ, and 0's elsewhere.

The result of non-equivalencing

```
    00010110
with 01110100
is   01100010
```

The OR operation, denoted by v, gives 1 in those positions in which either (or both) of the corresponding digits of the two numbers is a 1, and 0's elsewhere.

The result of ORing

```
    00010110
with 01110100
is   01110110
```

Function	Description	Notation
107	Collate the digits of Ba with the digits of s placing the result in Ba	$ba' = ba \,\&\, s$
106	Non-equivalence ba with s, placing the result in Ba	$ba' = ba \not\equiv s$
147	OR ba with s, placing the result in Ba	$ba' = ba \vee s$
117	As 107, but placing the result in S	$s' = s \,\&\, ba$
116	As 106, but placing the result in S	$s' = s \not\equiv ba$
127	Collate ba with n, placing the result in Ba	$ba' = ba \,\&\, n$

Function	Description	Notation
126	Non-equivalence ba with n, placing the result in Ba	$ba' = ba \not\equiv n$
167	OR ba with n, placing the result in Ba	$ba' = ba \vee n$

Examples:

1. Clear the most-significant 17 bits of B99 and leave the other bits unchanged.

127	99	0	15.7	Collate b99 with a number consisting of ones in the 7 required positions.

When n is used in this way it is called a "mask". It is often inconvenient to have to work out masks as decimal numbers with an octal fraction, so other ways of writing the address are allowed.

For example, if it is required to leave the most-significant 7 bits unchanged and to clear the rest of B99, then the mask required consists of ones in the 7 most-significant positions (0 - 6).

The two letters K and J introduce numbers written in octal notation.

K, followed by up to seven octal digits, positions the number from digit 20 upwards. Thus K 3642 places the number 00036420 in the address position. Octal zero's at the most-significant end may be omitted, and the least-significant octal fraction if present has to be separated from the number by a point. e.g. K5252525.2 fills the address digits with alternate ones and zeros.

J followed by up to eight octal digits, has the effect of compiling these digits from the most-significant end. That is, the first octal digit goes into bits 0 - 2 the next to 3 - 5 etc. Less-significant zeros may be omitted. Thus J142 places the number 14200000 in the address digits.

2. To leave the most-significant 7 bits of B99 unchanged and to clear the other digits

127	99	0	J771	collate b99 with a mask consisting of ones in the top 7 positions.

3. Replace the number in B62 with a number such that where there were ones there are now zeros and where there were zeros there are now ones. This is known as the 1's complement

126	62	0	J77777777	non-equivalence with a mask consisting of all ones. The mask could also be written K7777777.7 or - 0.1

Forming the 1's complement of a number is often not so simple as in this example, so the operator '(prime) has been provided. Any number followed by ' is interpreted as the 1's complement of that number. Thus the instruction could have been written

```
126     62     0     0'
```

There are two other logical instructions on Atlas, and these use bm as a further operand.

Function	Description	Notation
165	Collate bm with n and place the result in Ba, leaving bm unchanged	ba' = bm & n
164	Collate bm with n and add the result into Ba, leaving bm unchanged	ba' = ba + (bm & n)

Note: If Bm is B0 in the 164 and 165 instructions, then bm & n gives n rather than 0.

Example: Add the 6-bit character in digits 6 - 11 of B1 into B2.

```
164     2     1     J0077
```

4.4 Test Instructions

The following test instructions test bm, and transfer n into Ba if the test succeeds. n cannot be modified as bm is used. If the test fails, ba is unchanged.

Function	Description	Notation
214	If bm is zero, place n in Ba	If bm = 0, ba′ = n
215	If bm is not zero, place n in Ba	If bm ≠ 0, ba′ = n
216	If bm is greater than or equal to zero, place n in Ba	If bm ≥ 0, ba′ = n
217	If bm is less than zero, place n in Ba	If bm < 0, ba′ = n

These tests can be used with any B-registers but are most often used to cause a change of control if a certain condition is satisfied, so the control registers will now be described.

There are three control registers, only one of which is in operation at any given time. These are called main control, extracode control and interrupt control, and are the three special B-registers B127, B126 and B125 respectively. When a program is being obeyed, the address of the current instruction is held in the relevant control register. The control register is increased by one just before the instruction is obeyed in anticipation of the next instruction. Ordinary programs can use only B127.

Unconditional jumps to some address S are effected by placing this address in the control register

121 127 0 5 causes the following instructions to be taken from location 5 onwards.

Example: Two numbers a and b are in locations 14 and 14.4. A program which requires these numbers is in locations from 100.

If a < 0, b ≥ 0 enter this program at register 100
a < 0, b < 0 " " " " " 101
a ≥ 0, b ≥ 0 " " " " " 102
a ≥ 0, b < 0 " " " " " 103

The program is 7 instructions long; let it occupy the first 7 registers of store.

Register	Contents				
0	101	1	0	14	place a in B1
1	101	2	0	14.4	place b in B2
2	216	127	1	5	if a ≥ 0, jump to register 5
3	217	127	2	101	if b < 0, jump to 101
4	121	127	0	100	if not, a < 0, b ≥ 0 so jump to 100
5	216	127	2	102	if a ≥ 0, b ≥ 0, jump to 102
6	121	127	0	103	if not, a ≥ 0, b < 0, so jump to 103.

When writing a program it is helpful to show the possible routes of jumps with arrows. Unconditional jumps are often underlined, to indicate a definite break in control.

4.5 Special Purpose B-registers B120-B127

Although it is not necessary for the ordinary programmer to know about many of these special-purpose B-registers, details of them are given here for the sake of completeness.

It has been mentioned that there are three control registers, B125, B126 and B127, which are called interrupt control (I), extracode control (E) and main control (M) respectively. Ordinary programs use B127, and are prevented from having access to the subsidiary store and V-store.

Interrupt control is used in short routines within the Supervisor, which mainly deal with peripheral equipments. These routines are entered automatically whenever any peripheral equipment needs attention, e.g. when a tape reader has read a character. Occasionally the Supervisor will need to enter relatively longer routines to deal with the cause of interruption, e.g. on completion of the input of a paper tape. Whilst in interrupt control, further interrupts are not possible, so control is switched to extracode whenever the Supervisor enters a more lengthy routine. Both I and E control allow the Supervisor access to all the machine, but extracode control programs can also be interrupted and restarted in the same way as ordinary programs.

Extracode control is also used when any of the 300 or so subroutines in the fixed store are being obeyed. These subroutines have automatic entry and exit and are known as extracodes. When an extracode instruction is encountered, the relevant subroutine entry is placed in B126 and control switched to E. After the final subroutine instruction control is reswitched to M which holds the address of the next program instruction. (The current control register is always increased by one before the instruction is obeyed).

B124 has been introduced as the accumulator exponent ay. It consists of only the 9 most significant digits (0-8) the remaining 15 being always zero. Exponent arithmetic can be carried out by using B-code instructions. When this is done care must be taken to position exponents correctly in the digit positions 1-8 and to set the guard digit (bit 0) correctly.

Example:

```
121    124    0    J004    sets the exponent to +4
```

B123 is a B-register with the special property that a number read from it, instead of being the number last written to it, is the characteristic of the logarithm to base two of the eight least-significant digits of that number.

	Input to B123								Output from B123					
Digits	0-15	16	17	18	19	20	21	22	23	0-16	17	18	19	20	21-23
	x	0	0	0	0	0	0	0	1	0	0	0	0	0	0
	x	0	0	0	0	0	0	1	x	0	0	0	0	1	0
	x	0	0	0	0	0	1	x	x	0	0	0	1	0	0
	x	0	0	0	0	1	x	x	x	0	0	0	1	1	0
	x	0	0	0	1	x	x	x	x	0	0	1	0	0	0
	x	0	0	1	x	x	x	x	x	0	0	1	0	1	0
	x	0	1	x	x	x	x	x	x	0	0	1	1	0	0
	x	1	x	x	x	x	x	x	x	0	0	1	1	1	0
	x	0	0	0	0	0	0	0	0	0	1	0	0	0	0

Using B123, the Supervisor can identify the exact cause of an interrupt as a result of obeying from two to six instructions.

The programmer cannot use B123 directly because of the danger of an intervening interrupt which would alter the contents before they could be read out. A similar warning applies to B125, B126 and to all the B-registers B100-B118.

B122 and B121 are again B-registers provided with special circuitry. Their function is to allow indirect addressing and modification of the Ba operand in an instruction.

B121 behaves as a normal B-register except that it consists of only seven digits (15-21), the remaining bits being always zero. These seven bits allow B121 to hold any of the numbers 0, 0.4, 1, 1.4, ... up to 63.4. When B121 is used in conjunction with B122 its contents are interpreted as the address of a B-register in the range 0-127. That is, 0.4 ≡ B1, 1 ≡ B2, ... up to 63.4 ≡ B127, the B-register address starting from digit 15.

B122 is called the B-substitution register, which gives an indication of its function. When B122 is encountered as Ba in an instruction:

(a) the contents of B121 are taken as a B-register address, Bx say.

(b) the instruction is then obeyed as if the B-register specified in the Ba position was Bx.

A few examples will make this clearer.

Example 1

121	121	0	8.4	sets b121 ≡ B17 address
121	122	0	1	will place the number 1 in B17

Example 2

It is required to copy the contents of B87 into B80, B76, B72 and so on (every fourth B-register) leaving the other B-registers unchanged.

This could be done by the sequence of instructions

121	80	87	0	copy B87 into B80
121	76	87	0	
121	72	87	0	
121	68	87	0	

etc. ..... for 19 instructions in all but by using B121 and B122 we can write a short loop of instructions.

6	121	121	0	40	set address of B80 in B121
7	121	122	87	0	copy b87; into B80 first time B76 second time etc.
8	122	121	0	2	subtract 4 from the B-address
9	215	127	121	7	if b121 $\neq$ 0, jump to the instruction in location 7

When b121 = 0 the jump does not take place, and the program proceeds to the next instruction.

B121 and B122 play an important part in the extracodes. When an extracode instruction is met, just before control is switched to extracode, the Ba digits in the instruction are copied into B121. This allows the extracode routine to operate on Ba by using B122. B-register 119 is also set up in a special way when an extracode instruction is met, to enable the extracode routine to obtain the store operand involved. This is described later.

In between a program's extracode instructions the programmer is able to use B121, B122 as he likes, but caution must be exercised to avoid inadvertent over-writing of their contents when an extracode instruction is called for.

B122 only operates as the B-substitution register when it is in the Ba digits of an instruction. In the two other circumstances possible, its value is zero. These are:

(a) Bm specified as B122

121	1	122	0	always puts zero in B1

(b) Using B122 as Ba when the contents of B121 are 61, i.e. B121 is pointing at B122

121	121	0	61	set b121 = B122
113	122	0	100	writes the number zero into store location 100

Any number written into B120 is displayed as 24 digits on neon lamps on the engineers console. Thus:-

```
121      120      0        J52525252 displays alternate ones
                                    and zeros
```

The engineers console is not normally available for use by the programmer.

Whenever it is attempted to read from B120, the number read out is always zero.

4.6 Modification/Counting Instructions

The technique of modification has already been introduced.

In Atlas instructions, the contents of any of the B-registers not directly concerned in the operation may be used to modify the address. Thus, the instruction

324 0 3 100

copies the contents of location (100 + b3) into the accumulator (and standardises the result).

Suppose we have 20 unstandardised floating-point numbers stored in locations 100-119, and it is required to standardise these numbers and restore them in the same locations. A program to do this might be as follows:-

10	121	3	0	19	set 19 in B3
11	324	0	3	100	$am' = s$, standardised
12	356	0	3	100	$s' = am$
13	122	3	0	1	subtract 1 from b3
14	216	127	3	11	jump to location 11 if $b3 \geq 0$

B3 is used as the modifier and to ensure that the loop is cycled 20 times. This latter process, counting, is of such frequent occurrence that eight basic counting instructions have been provided.

The most important of these are:-

Function	Description	Notation
200	If the contents of Bm are non-zero, add 0.4 into Bm and place n in Ba. If bm = 0, bm and ba are unchanged.	If $bm \neq 0$, $bm' = bm + 0.4$ and $ba' = n$
201	As 200 but increase bm by 1	If $bm \neq 0$, $bm' = bm + 1$ and $ba' = n$
202	If bm is non-zero, subtract 0.4 from it and place n in Ba	If $bm \neq 0$, $bm' = bm - 0.4$, $ba' = n$
203	As 202 but subtract 1 from bm	If $bm \neq 0$ $bm' = bm - 1.$ $ba' = n$

Note: About instructions 200, 201, 202, 203 : If Ba and Bm are the same B-line and the test succeeds, its final contents are n. If Bm is B127 (and Ba is not), these instructions give an unpredictable result.

The last two instructions in the example above would be replaced by

203	127	3	11	If $b3 \neq 0$, $b3' = b3 - 1$ and $b127' = 11$. i.e. jump back with b3 reduced by one.

Examples:

1. At the addresses 50-99.4 inclusive there are 100 half-words. Find how many of these numbers are zero and leave the answer in B7.

0	121	7	0	0	start count of numbers = 0
1	121	2	0	49.4	set count/modifier in B2
2	101	3	2	50	number to B3
3	215	127	3	5	jump to 5 if $b3 \neq 0$
4	124	7	0	1	if $b3 = 0$, add 1 to b7
5	202	127	2	2	count

2. To clear the B-registers B1 to B100

0	121	121	0	50	set count/modifier in B122
1	121	122	0	0	clear B100 first time, then B99 etc.
2	202	127	121	1	count reducing b121 by 0.4 each time and jump back

4.7 The B-test Register

The B-test register Bt consists of two digits only.

When a number is written to Bt, one of these digits is set to show whether the number is $= 0$ or $\neq 0$, and the other to show whether it is ≥ 0 or < 0.

Instructions are provided to write numbers to Bt, to test the above mentioned conditions, and to count. These are:-

Function	Description	Notation
152	Set the B-test register by writing to it the contents of Ba minus the contents of S. ba and s are unchanged.	$bt' \equiv ba - s$
150	Set Bt by writing s minus ba to it. s and ba are unchanged.	$bt' \equiv s - ba$
172	Set Bt by writing ba minus n to it.	$bt' \equiv ba - n$
170	Set Bt by writing n minus ba to it.	$bt' \equiv n - ba$
224	If Bt is set equal to zero place n in Ba	If $bt = 0$, $ba' = n$
225	If Bt is set not equal to zero place n in Ba	If $bt \neq 0$, $ba' = n$
226	If Bt is set greater than or equal to zero, place n in Ba	If $bt \geq 0$, $ba' = n$
227	If Bt is set less than zero, place n in Ba	If $bt < 0$, $ba' = n$
220	If Bt is set non-zero, place n in Ba and add 0.4 to bm. If Bt is set zero, do nothing	If $bt \neq 0$, $bm' = bm + 0.4$ and $ba' = n$
221	If $bt \neq 0$, place n in Ba and add 1 to bm	If $bt \neq 0$, $bm' = bm + 1$ and $ba' = n$
222	As 220 but subtract 0.4 from bm	If $bt \neq 0$, $bm' = bm - 0.4$ and $ba' = n$
223	As 221 but subtract 1	If $bt \neq 0$, $bm' = bm - 1$ and $ba' = n$

Note: In instructions 220, 221, 222, 223, if Ba and Bm are the same B-line and the test succeeds, its final contents are n. If Bm is B127 (but Ba is not), these instructions give an unpredictable result.

Bt is not directly addressed; Bt instructions are recognised by the function digits. The instructions to set Bt are useful for comparing numbers, as the operands are not altered.

The conditional transfer instructions, 224-227 are used to cause a conditional jump, and as bm does not take part in the instructions it can be used to modify n, giving a modified address for the conditional jump.

Example:

In 100 to 199.4 inclusive there are 200 half-words. Find the lowest address of this range which contains the number -3 and store this in 99.4. If no such number exists, set 99.4 = -0.1

0	121	1	0	-3	set required number in B1
1	121	2	0	100	first address in B2
2	152	1	2	0	$bt' = ba - s$
3	224	127	0	7	jump if bt - 0, i.e. $s = -3$
4	170	2	0	199.4	$bt' = 199.4 - ba$
5	220	127	2	2	if $bt \neq 0$, $b2' = b2 + 0.4$, jump back
6	121	2	0	-0.1	if search fails, set -0.1
7	113	2	0	99.4	store result

4.8 The Shifting Instructions

Four instructions are provided which shift the number in Ba. These shifts are either of six places up or of one place down, and are circular shifts. That is, digits which are shifted out of the register at one end re-appear at the other end.

The main purpose of these instructions is to assist in the manipulation of 6-bit characters and to provide ways of shifting ba any specified number of places.

Function	Description	Notation
105	Shift ba up 6 places, copying the initial 6 most-significant bits into the least-significant 6 bits, then add s into ba	$ba' = 2^6 ba + s$, (circular shift)
125	Shift ba as in 105, then add n	$ba' = 2^6 ba + n$, (circular shift)
143	Shift ba down one place, copying the initial least-significant digit into the new most-significant position, then subtract s	$ba' = 2^{-1} ba - s$, (circular shift)
163	Shift ba as in 143, then subtract n	$ba' = 2^{-1} ba - n$, (circular shift)

These basic instructions are intended to be used by extracodes which provide more useful shift functions.

4.9 The Odd/Even Test Instructions

Two further test instructions can be used to test the least-significant bit of Bm. These instructions can be used, for example, to identify a character address.

Function	Description	Notation
210	If the least-significant bit in Bm is a one, place n in Ba	If $\overline{bm}$ is odd, $ba' = n$
211	If the least-significant bit in Bm is a zero, place n in Ba	If $\overline{bm}$ is even, $ba' = n$

Note particularly that it is the very least-significant bit that is tested, and that if Bm contains an address these instructions do not detect whether the address refers to an even or odd numbered word in the store, but rather whether it refers to an even or odd numbered character within the word.

Example:

B1 contains a character address, A.k say. Place this character in digits 18 to 23 (character position 3) of B2 and clear the rest of B2 (digits 0 to 17)

0	101	2	1	0	Read the half-word into B2
1	210	127	1	3	jump if k is .1 or .3 in the half-word
2	125	2	0	0	shift up and round six places
3	163	1	1	0	shift b1 down and round one place, then subtract the original contents of B1 from this. This makes b1 even, if k is .0 or .3
4	211	127	1	7	jump if k = .0 or .3 in half-word
5	125	2	0	0	shift up 12 bits, circularly. The required character is now in c5 of B2
6	125	2	0	0	
7	127	2	0	7.7	clear the unwanted digits

A similar program to this is obeyed, under extracode control, when the programmer specifies extracode 1250.

1250 Ba Bm S

places character s in Ba, clearing the other digits of Ba. So the example above would be simply achieved by

1250 2 1 0

4.10 Restrictions on the Use of B-registers

Although B81-B119 were included in section 4.1 as general purpose B-registers, they are of limited utility for the ordinary programmer, since they are each used by one or more of the system routines which may assume control during the running of the object program. Before using any of these B-registers, the B-test register, the substitution register, or the B-carry digit, the programmer must check to see that there is no danger of their contents being overwritten before he has finished with them.

The routines which use these B-registers are as follows:-

B81-89	Library routines
B90	Return link from library routines
B91-97	Extracodes
B98-99	The logical Accumulator and some less common extracodes
B100-110	Supervisor
B111-118	Interrupt routines
B119	Extracode operand address
B121, 122	Extracodes, library routines
Bt, Bc	Extracodes, library routines

It should be noted that the library routines may use extracodes. This means that when library programs are in use, no B-line above B80 should be used (except for B90). Provided no reference is made to library routines, B81 - B90 may be freely used. Similarly B81 to B99, B119, B121, B122, Bt and Bc are safe to use when neither extracodes nor library routines are in use. It is never safe for an ordinary program to use B100 - B118, since an interrupt can occur at any time and cause control to be transferred to the Supervisor.

4.11 The B-carry digit

When any one of the four addition codes

104	ba' = ba + s
114	s' = ba + s
124	ba' = ba + n
164	ba' = ba + (bm & n)

is used to add two 24-bit quantities, bit 23 of line 6 of the V-store is set to 1 if there is a 'carry' from the addition.

Thus for example the addition of any two 24-bit numbers whose left-most bit is a 1 sets the 'B-carry digit' to one. If there is no 'carry', the B-carry digit is set to 0. When an ABL program is entered the B-carry digit is clear.

The singly-modifed extracode 1223 loads Ba with n if the B-carry digit is set to 1 and does nothing if it is not set. (The extracode does not affect the state of the B-carry digit.) The following example uses 1223 to add B1 to B2 and then add the 'carry', if any, to the bottom of B3, Thus the contents of B1 and B2 are here regarded as 24-bit positive integers whose double length sum is placed in B3 and B2 with the most significant half in B3.

Example:

124	2	1	0
1223	3	3	Y1

Similarly each of the ten instructions

100	ba' = s - ba	102	ba' = ba - s
110	s' = s - ba	112	s' = ba - s
120	ba' = n - ba	122	ba' = ba - n
150	bt' = s - ba	152	bt' = ba - s
170	bt' = n - ba	172	bt' = ba - n

set the B-carry digit to 1 when, regarded as 24-bit positive integers, a larger number is subtracted from a smaller. Otherwise these codes set B-carry to zero. For example, the B-carry digit is set to 1 by the instruction 172, 0, 0, 1.

Example:

In the previous example b1 was added to b2 and the double length sum held in B3 and B2. The following two instructions would subtract b1 off again from the double length sum.

122	2	1	0
1223	3	3	-Y1

Chapter 5

ROUTINES AND DIRECTIVES

5.1 Routines, Subroutines and Symbolic Addresses

For convenience in writing a large program it is broken down into parts, called routines. Each routine usually performs some particular step in the calculation, and the routines once decided on, may be written in any order and then assembled together to form a program.

Many routines, for example one which finds the cube-root of a number, are useful in assisting other routines. Such routines are called subroutines. The programmer may well write his subroutines before the major routines and have his own system of entry and exit from them so that more than one routine can call on a particular subroutine. Generally useful subroutines which have been written for use in any program form a "Library" of routines.

In general subroutines may be "open" or "closed". An open subroutine is simply a group of instructions which may be inserted anywhere in a program. When they are required to be executed, control passes to the first instruction; after the subroutine the next instruction after the group is obeyed. This has the disadvantage that the group of instructions has to be copied into the program wherever it is to be used.

A closed subroutine is one which is entered by jumping into an entry point, often the first instruction, and which ends by returning control to an address set by the program before entry. This exit address, called the "link", is normally by convention set in B90 for the Library routines; these then end with the instruction

121	127	90	0	Copy the return address set by the program in B90 into control.

Particular examples of closed subroutines are the "extracodes" in Atlas. These are called automatic subroutines as entry to them is automatic on an extracode instruction being met. Exit from them is normally to the next program instruction, with no link needed.

Within any routine there may be many jump instructions. It is inconvenient to have to work out where each routine would be in the store so that the addresses for these instructions can be specified. Also, insertion of an extra instruction into a program written with actual addresses might mean that many addresses had to be altered. Addresses are therefore allowed to be defined by means of parameters. Using these, any address can be referred to in a "floating" form. Each time the program is read into the computer, the input routine assembles the true machine addresses and inserts these in place of the parameters.

Besides the instructions themselves, certain additional information has to be provided with the program. This information is:

(a) Where the program is to be located in the store.

(b) Which library routines are required.

(c) The identification of routines, program titles, etc.

(d) Where control has to pass to in order to start obeying the program.

This information is provided by means of directives. Except for (b) these do not produce any actual program within the computer.

In general, directives are simply identifying letters (followed sometimes by numbers) or equations which define the values of parameters.

Instructions, floating-point numbers, half-word numbers, six-bit characters and directives will be collectively referred to as "items". A complete program can then be regarded as a list of items.

Items are terminated by multiple-space, comma or New line. Depending on the input media, which may be 7-track or 5-track paper tape or punched cards, the programmer will use whichever terminator is most convenient.

For simplicity we shall assume that the input medium is 7-track paper tape punched on a Flexowriter, and then state the alternatives for the other media. Multiple-space is defined as two or more consecutive spaces, which can also be achieved by using the character Tabulate on the Flexowriter.

Routines are introduced by the letter R followed by a routine number in the range 1 to 3999. They are terminated either by the directive Z, or by R followed by a new routine number, or by one of the directives which cause the program to be entered. Any program material not introduced by a routine number is automatically assigned to routine 0.

A complete line of ABL input is read, and an image of the print-out is formed, taking correct account of the characters SPACE, BACKSPACE, and TAB. TAB is interpreted assuming 9 fixed TAB positions, at 8, 16, 24, 32, and then every 16 up to 112, character positions from the left-hand margin; TAB always moves the current 'carriage position' along at least two character positions. A maximum of 128 character positions along the line is allowed for; any characters beyond position 127 are ignored. A backspace beyond the left-hand margin is ignored.

In interpreting a line, ERASE, or a composite character including ERASE, is everywhere ignored (except in circumstances where a direct copy of a string of characters is called for -see section 5.10 - C and CT directives - and section 5.13 - T directives, or with the ZL record in library routines - see section 12.7).

The character small l is an illegal character. Otherwise ABL treats upper and lower case letters as being identical. The letters O and I are treated as alternatives to zero and one.

5.2 Routine Parameters

Within any routine, up to four thousand parameters may be used, numbered from 0 to 3999. Parameters 1-3999 can be set up by directive equations or by labelling items (other than directives).

When a parameter is set by an equation, or referred to generally, it is preceded by the letter A.

A1 = 100.4 sets parameter 1 of the current routine to the value 100.4

When a parameter is set by labelling an item it is written before the item and separated from it by a right-hand bracket.

```
1) 121     2     0     0
```

The parameter then has a value equal to the address at which the item so labelled is finally placed. Hence, other instructions which refer to A1 are not affected by the insertion or removal of instructions in between themselves and the labelled instruction.

We have up to now always written the address part of an instruction as a number, either as a decimal number with an octal fraction or as a string of octal digits. In fact, the writing of an address may be done in a great many ways. In particular, parameters may be used to set up addresses, or parts of addresses. Thus

```
121     127     0     A1
```

causes a jump to the location whose address is defined by A1.

Example:

Routine 1 of a program is to clear store locations 512-1535 to floating-point zero for working space and then exit to some as yet unknown address

```
   R1
   324     0     0     A1      set am' = 0
   121     7     0     1023    set b7 = 1535-512
2) 356     0     7     512     store, modified
   203     127   7     A2      count, jump to A2
   121     127   0     A3      exit to A3, not yet set
1) +0
```

Before the program could be run on the computer, A3 would have to be set.

A0 in each routine cannot be set by the programmer; it is automatically set equal to the address of the first stored item of the routine (usually an instruction). A0 can be abbreviated to A.

To permit cross references between routines, parameters are more generally referred to as Am/n, meaning parameter m of routine n. If the /n is omitted the parameter is taken to belong to the current routine.

Examples:

1.	A3/15 = J77				Sets parameter 3 of routine 15 to J77
2.	R5				
	101	10	0	A1	Extract half-word at A1 of R5
	214	127	10	A2	If b10 = zero, jump to A2 of R5
1/6)	217	127	10	A3	This instruction is labelled A1 of R6, so that R6 may refer to it.

An item may be labelled more than once. Thus

1/6) 2/7) 3)	121	127	1	A4	sets A1 of R6, A2 of R7, and A3 of the current routine to the address of this instruction.

5.3 Preset Parameters

These are identified by the letter P followed by the parameter number. One hundred preset parameters P0 to P99 are available for normal use, although certain parameters with numbers greater than this exist, and have special effects. (see section 12.5)

Unlike routine parameters, preset parameters are not associated with any particular routine, but are meant for use by the program as a whole. They can only be set directly by equation, and not by labelling; they may not be referred to before they have been set. Preset parameters are set immediately they are encountered, and hence everything on the right hand side of the equation must itself already have a value.

Preset parameters may be reset by further equations, and may also be unset, using the symbol U, followed by the parameter number.

Ua will unset Pa
Ua-b will unset Pa to Pb inclusive.

Preset parameters may also be set and unset by program, using some of the special parameters listed in section 12.5.

5.4 Global Parameters

These are identified by the letter G followed by the parameter number (0 to 3999).

Like routine parameters, global parameters may be referred to before they are set and cannot be unset. But, like preset parameters, they must always be set explicitly by means of an equation and not merely by labelling an item.

Global parameters are not associated with any particular routine, and they therefore supplement preset parameters as universal parameters for use by all routines.

5.5 Optional Parameter Setting

This facility can best be described by means of an example:-

The library routine L100 uses a parameter, A24, to specify the maximum number of characters permissible on a line of input, which determines the amount of working space needed to hold one line at a time. The programmer may arrange to set A24/L100 outside this library routine, but if he neglects to do so then L100 will itself set A24 to the value 160.

Such an optional setting is obtained by using the symbol ? before the = sign in a parameter setting directive within the subroutine. This has the following effects:-

(a) For preset parameters. The directive is ignored if the parameter is already set, otherwise it is immediately implemented.

(b) For routine parameters and global parameters. The directive is ignored if the parameter has been set by the time the next enter directive is encountered, otherwise it will be implemented at that time.

The library routine L100 contains the directive

A24? = 160

and if the programmer wishes for a different setting he must set A24/L100 in his program. (see section 5.12.)

5.6 Expressions

It is now necessary to explain the many ways in which addresses can be built up.

The general form of an address is an "expression", and the Ba and Bm parts of an instruction, 6-bit characters and half-word numbers can also be formed from expressions.

Basically an expression consists of a mixture of parameters and constants which are combined together according to some relatively simple rules.

We have written most constants as a decimal number with or without an octal fraction, that is as b or b.c, where b is the decimal number and c is the fraction. b goes into digits 0-20, c into 21-23. More generally, one can write a:b.c where b and c are as before and a is a decimal number which is added into digits 0-11. The main use of a is to set up multiples of 512 in the address digits.

Alternative forms are:-

a:b.c

a:b

a:

b.c

b

Examples:

1:35.6 is, in octal, 00010436

2: is 00020000

The symbol / may be used instead of :, as : does not occur in the 5-track paper tape or card codes.

The letter Y followed by a decimal number has the effect of positioning the number from the least significant end of the register instead of one octal place up. Thus

Y19 is 00000023

as opposed to 19 which is 00000230.

We have also written numbers in octal, preceded by J or K.

J followed by a string of up to eight octal digits assembles these from the most-significant octal position and right-hand zeros may be omitted.

K followed by a string of up to seven octal digits assembles these from the right, starting at bit 20, and more-significant zeros can be omitted. Writing .c after these numbers, where c is again an octal digit, places c in digits 21-23.

Examples:

1. J04103 is 04103000
2. K37 is 00000370
3. K31.5 is 00000315

It is also possible to form a number by writing a constant or parameter which is followed immediately by one or more of five "operators". These operators allow numbers to be shifted up or down logically, allow the extraction of the 'block address' digits or 'address within a block' digits, or form the logical binary complement of a number.

We shall use the term "element" to mean either a constant or a parameter, or one of these followed by one or more operators.

The operators are as follows:-

(a) Dn where n is a decimal integer. This causes the previous element to be shifted down logically by n places, i.e., shifted down without duplication of the sign digit. e.g.

```
121    121    0    100D1
```

is an alternative to writing

```
121    121    0    50
```

and sets b121 pointing at B100, for use in conjunction with b122.

(b) Un causes the previous element to be shifted up logically by n binary places (i.e. multiplied by 2^n). e.g.

```
121    124    0    13U12
```

sets the exponent digits 0-8 as +13. (This is more convenient than having to work out the number in octal in the appropriate place.)

(c) B gives the block address, i.e. bits 0-11, of the previous element, with bits 12-23 made zero.

(d) W gives the address within a block, i.e. bits 12-23, of the previous element, with bits 0-11 made zero.

(e) ' (prime) gives the logical binary complement of the preceding element. e.g.

```
121    2    0    J1'
```

sets b2 = J67777777

The use of ' is not encouraged, because it is a symbol so easily overlooked in a program print-out.

5.7 Separators

Elements can be combined together in many different ways to form a final expression,

(i) Elements may be added or subtracted.

Thus 3 + A1, A1 - 3, A1 + A2 - A3 + 6D1 for example, are all allowed. Where the next element is to be added, the + may be omitted if there is nc possible ambiguity. Thus, equivalent forms of the three examples above are 3A1, -3A1, A1A2 - A3 + 6D1. In the last case, the final + cannot be left out as this would form -A36D1.

Examples:

324	0	0	3A1	Sets am' = contents of the third location after the address given by A1.
121	127	0	-2A16/3	Causes a jump to the instruction two before the address given by A16/3.

(ii) The logical operations AND, non-equivalence, and OR can be performed between two elements. The symbols for these are &, N and V respectively. M is an alternative to &.

Example:

K77.7&A2 Extracts the least-significant 9 bits of A2 and sets the other digits to zero.

(iii) Elements may be multiplied and divided. The symbols used are X and Q.

Examples:

A1 X 30	Multiplies A1 by 30
A1 Q 30	Divides A1 by 30

For these two operations, elements are regarded as 21-bit integers with octal fractions. After multiplication, the answer is made a 21-bit integer with an octal fraction; the result is taken modulo 2^{30} and the octal fraction is rounded away from zero. After division the result is an integer in digits 0-20, rounded towards zero, and is always exact if an exact result should be obtained.

Examples:

2.4 X 2.4 = 6.2	Exact
2.1 X 1.2 = 2.6	Result rounds away from zero
J001 X J001 = 0	$2^{30} = 0$, modulo 2^{30}
Y1 X Y1 = 0.1	Result rounds away from zero

17 Q 3 = 5	Result rounds down
14 Q 3.4 = 4	Exact

The symbols +, -, &, M, N, V, X, Q are termed "separators".

An expression consists of a string of elements and separators, and the elements are evaluated and compounded together from left to right.

Elements and separators are allowed to be enclosed in round brackets, and sets of brackets within brackets are permitted. The contents of brackets, beginning at the deepest level, are evaluated first and replaced by single elements before the general left to right evaluation is carried out.

+ and - signs may also precede any element, including the first of an expression, or follow any separator other than themselves.

5.8 The Special Parameter *

When * is set by an equation such as * = expression, it is interpreted as a directive determining where succeeding items are to be placed in the store.

For example

```
* = 100
121     1     0     6
324     0     1     A2
 .      .     .     .
```

arranges that the first instruction is placed in location 100, the next in 101, and so on until a further setting of * is made. If * is not set at the head of an ABL program, * = 1: is assumed.

The right-hand side of the * directive must not contain any parameters which have not been previously set.

Optional setting of *, that is *? = expression, is not allowed.

When * occurs in an expression on the right-hand side of a directive, then * is equal to the address of the next available character position, except when it is set to the address of the next full-word or half-word by the directives F or H respectively. (See sections 5.10 and 5.11)

When * appears in any item other than a directive, it has the value of the store address of the item.

The instruction `121  127  0  *` is a loop stop as * equals the address where this instruction is held.

`121  127  0  3*` causes a jump round the next 2 instructions.

* may be used in expressions as an ordinary parameter. Insertion of extra instructions into a program is more liable to lead to errors if * has been widely used. In the example above, inserting another instruction to be jumped round would also involve altering the jump instructions to

`121  127  0  4*`

There are no applications in expressions for * that cannot be achieved by the use of ordinary parameters.

5.9 The Ba and Bm Parts of an Instruction

An instruction is regarded as an item, although the four parts of an instruction have to be separated by multiple spaces or commas, and no other items may occur on the same line.

The Ba and Bm parts of an instruction have been written so far as integers in the range 0-127. In fact they can be written as expressions like the address part. Bits 14-20 only of the expression are extracted and placed in the Ba or Bm position.

One use of this is in relativisation of B-register addresses. For example, a routine might require the use of some B-registers without it being known at the time of writing which would be most convenient. By writing the B-addresses relative to a parameter the range can be decided later and the parameter then set.

Example:

```
R93
165     A7      1A7     0.7
165     2A7     1A7     J4
214     127     A7      A1
101     3A7     1A7     0
```

If it is decided that B62 onwards can be used for this routine then the directive A7/93 = 62 will set the B-registers referred to as A7, 1A7, 2A7,..... to the values 62, 63, 64,.....

5.10 Half-Words, Six-Bit Words and Characters

The directive H introduces 24-bit numbers. These numbers are written as expressions in exactly the same way as in the address parts of an instruction.

After H, successive expressions on one line are regarded as 24-bit items and placed in the next available half-words. The letter H, which needs no terminator, need only appear before the first expression on each line.

Examples:

```
H1      2      3      4
HA5/2          A6/2   5.6     A7/2 & K77
```

Any of the items can be labelled in the normal manner. If the item to be labelled is one with the directive H, the label can occur before or after the H.

Examples:

```
3)H 1.4   2.4    4)3.4
H 5)A1
```

Other directives may occur mixed with half-words.

Example:

```
H6      7      A1=*      8      9      A2=57
```

The directive H also increases *, if necessary, to the address of a half-word.

The directive S introduces 6-bit words. Its action is almost identical with that of H, except that only bits 15-20 of expressions are used and these are assembled into successive character positions.

Example:

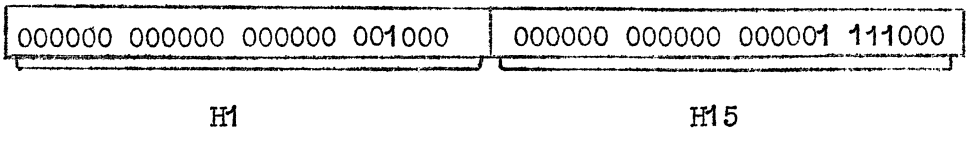
```
S1      22      3      1)4
```

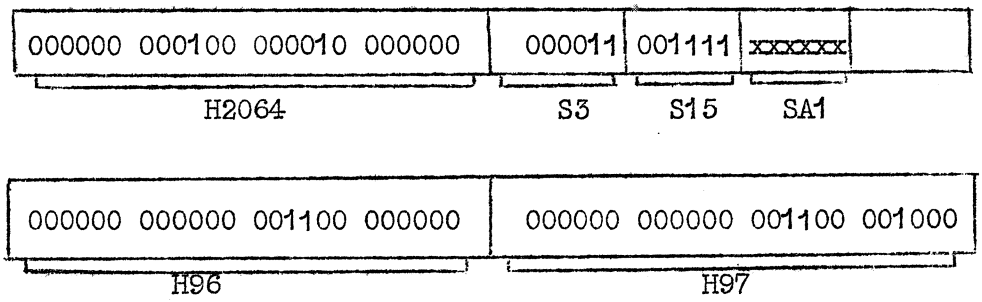
On one line it is possible to write some half-words, and some six-bit words. Thus

```
H1      3)15      2064      S3      15      A1      H96      97
```

is permitted, for example, each directive stating the interpretation to be given to succeeding numbers up to a new directive or new-line.

The numbers would appear in three succeeding words of the store as:-





However, for clarity, the mixing of numbers in this way is not to be encouraged.

Several directives are provided to read in characters from the input media and to store them in internal code ready for output. The string of characters is introduced by one of the following C directives. All subsequent characters after the directive, up to and including newline, are ignored, and all the following characters in the next record except the carriage control character are stored. The line is not reconstructed, characters being stored as they are punched, apart from shift changes. This means that redundant shifts (e.g. run-out on 5 and 7 track tape) are stored.

C The characters are stored in the next available character locations. P 123 gives the number of characters stored plus J4.

Ca As for C, except a is placed in the next character location after the string, to be used as a carriage control character (where a is an octal number less than 77; a point between the digits will be ignored). P 123 gives the number of characters stored (including a).

The first character may be labelled by writing a label before the C or Ca.

CT The next available halfword is set to the number of characters plus J4 and the characters are stored in the following character locations.

CTa As for CT except that a is placed in the next character location after the string, to be used as a carriage control character, and the first half-word is set to the number of characters.

The half-word containing the character count may be labelled by writing a label before the CT or CTa.

An additional C added to the directives C and CT making CC and CCT (orCTC) respectively, has the effect of ensuring that the string of characters ends in inner set, adding a 'shift to inner set' character if necessary. This extra character, if required, is included in the count of characters. An additional C added to the directives Ca and CTa will be ignored. The use of the additional C is intended for continuation of the record with further output which will start in inner set.

In every case, the character count is positioned from the least significant end of the register.

CT is intended for a list of texts any of which can be output by the instructions

```
101     2     1     0
1066    2     1     Y4
```

where B1 contains the address of the label attached to the CT directive. A description of the output procedures is given in Chapter 8.

For example

```
1)C
  No solution
  A2 = -0.1*      A4 = (A1 - A2)U3 + 1
```

assembles the comment 'No Solution' as internal code characters, sets A1 equal to the address of the first character, A2 to that of the last and A4 to the number of characters.

5.11 Floating-Point Numbers

48-bit floating-point numbers may be written in various ways. Each number is assembled into the next full-word location. The ways in which such numbers may be written are listed below.

The following notation is used:-

a is a signed decimal number, which may include a decimal point with any number of digits before or after it.

b, c, d are decimal integers which may be preceded by a sign (optional if +).

In the following cases the exact or nearest possible value is assembled as a standardised floating-point number.

(i) a

Examples:

+1
-16354.77625
+3.14159
-.5
+.1234
-27

(ii) a(b) The value of the number is $a \times 10^b$

Examples:

+1(6)
+3(-2)
-.5(-7)

(iii) a(:c) The value of the number is $a \times 8^c$

Examples:

+1(:3)
-17(:-2)

(iv) a(b:c) The value of the number is $a \times 10^b \times 8^c$

After any of the four ways listed above :d may be written. Then, after the standardised number has been formed it will have its exponent forced equal to d with the mantissa shifted accordingly. Thus a:d, a(b):d, a(:c):d and a(b:c):d are the four ways of writing floating-point numbers with forced exponents.

It is also possible to write any of a, b, c or d as octal numbers.

a can be written as a string to any length of octal digits which may include an octal point, and these must be preceded by + or -, and the letter K.

e.g. +K363.174

-K.265

-K777777

+K0.4

b, c or d can be written as an octal integer preceded by K. The K may be preceded by a sign.

e.g. K14 -K276

The character / can be used as an alternative to :.

Note: If the program contains a floating-point number that is too large to be stored in Atlas standardised form the program will be monitored during compilation and the fault indicated by the monitor printing EXPONENT OVERFLOW. If a floating-point number is too small to be represented in standardised form ABL will store floating-point zero in its place and continue compiling the program.

If the exponent of a floating-point number is forced to a value that is too small to allow the number to be represented in standardised form, the program is monitored and A0 on fixing is printed to identify the fault. For example, +1:0 requests that 1 be stored in floating-point form with exponent zero, which cannot be done (-1:0 would be acceptable).

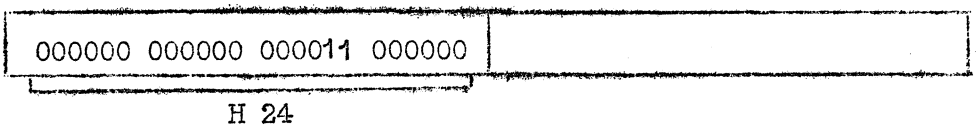
Any floating-point number can be labelled, and more than one may appear on a line. Floating-point numbers can also be mixed with half-words and six-bit words on a line, provided that the first of a group on a line is preceded by the directive F.

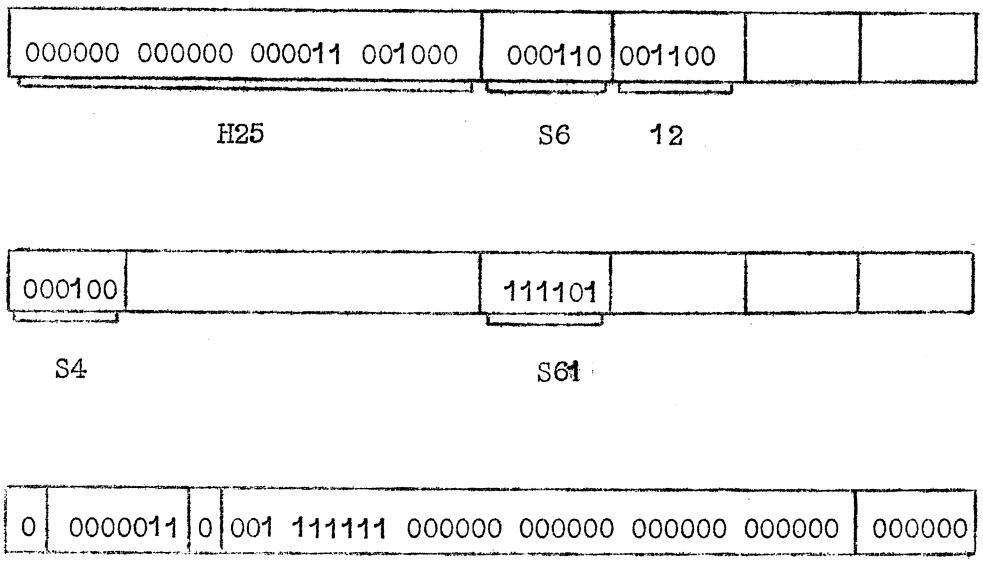
This directive, which does not need a terminator, introduces floating-point numbers. It has also the effect of increasing the value of *, if necessary, to the address of a full-word, and it can be used, for example, immediately before H to ensure that the next half-word is stored as the more-significant half of a full-word.

Example:

H24 F H25 S6 12 F S4 H S61 F+127

would appear in four consecutive words of the store as





F-127

The practice of mixing the different types of number on one line is not encouraged.

5.12 Library Routines

A copy of the standard library routines is held on a system magnetic tape.

To avoid confusion with programmer's routines, library routines are referred to by the letter L followed by a decimal number in the range 1 - 1999.

Parameters in library routines are referred to by elements of the form

Aa/Lc label a of library routine c.

Some library routines may have more than one routine. In such cases, the routine number is written before L.

Aa/bLc label of a routine b of library routine c.

In some programs, it is occasionally convenient to have more than one copy of a particular library routine. To allow references to any particular copy, it is identified as Lc.d, copy d of Lc. Copies 1 - 1999 are permitted.

Aa/bLc.d label a of routine b of library routine c, copy d.

It is possible to refer to library routines in the address parts of instructions or in expressions for half-words etc. without calling for them explicitly by an L directive. If this is done, when the directive E or ER is reached, any such library routines are found and read into the following storage locations.

If it is desired to insert a library routine at a particular address, this may be done by setting the address with a * directive, if necessary, and then writing an L directive, for example:-

```
* = A2   )
        .  } read L10 into addresses from A2 onwards.
L10     )
```

If all library routines mentioned earlier in the program are to be inserted, then no number follows L, for example:-

L read all library routines previously mentioned into the current address onwards.

In this case, the action is just the same as when an E or ER directive is encountered, except that the L directive may be placed anywhere in the program after the library routines have been mentioned.

Private library routines may be incorporated into a program; this is described in Section 12.7.

5.13 Directives

Most of the directives have been introduced in this chapter. This section gives a complete list of the directives and describes those not so far introduced.

Equation directives are used for setting the values of routine parameters (1-3999 per routine), of *, of global parameters (0-3999), and of preset parameters (0-99). They are of the form

Parameter = Expression

Optional parameter setting (except for *) is of the form

Parameter ? = Expression.

Any optional parameter settings to be made for a library routine should occur before the L directive that calls that routine.

'Ignore' directives

(i) vertical line | which does not need a terminator has the effect that all subsequent characters up to the next new-line are ignored. This allows comments and notes to be inserted into a program for the convenience of anyone reading the program print-out. The characters π and £ are alternatives to |.

(ii) Square brackets [ ] which again do not need terminators have the effect that anything contained within them is ignored; the sections to be ignored may stretch over any number of lines and part lines. This facility is intended for lengthy comments or the temporary ignoring of whole sections of program, for example during the development stage of a program.

< and > are alternatives to [ ], but < and > can be used with their meanings of "less than" and "greater than" within square brackets. If < is encountered first then [and] can be freely used within the comments

Both [ ] and < > can be "nested!" When [is encountered what follows is scanned with only [and] being recognised. A count starts at 1 on the first [, is increased by 1 for each further [and reduced by 1 for each]. The comment is considered to have terminated only when this count becomes zero. < and > are treated in exactly the same way.

(iii) Query ? followed by an expression and a terminator, has the effect of ignoring the rest of the line or not, depending on the value of the expression. If this is zero, the remainder of the line is ignored as with |; otherwise, there is no effect. The expression must already have been assigned a value.

Examples:

```
[a > b]

[a >[b - c]>d]

< 1 + a [1 + b [1 + c [.... >
```

```
?P50 [
xxx...
...xx |]
```

This last causes the following lines up to] to be ignored if P50 is non-zero, but to be taken account of if P50 is zero. The | before] ensures that the] is never unmatched.

Note: | π £ [< and ? are directives and not terminators. They must not occur other than after correctly terminated items.

C and CT directives

The directives C and CT on one line introduce 6-bit characters on the following line. (see section 5.10)

E directives

The enter directive, E followed by an expression. The expression gives the address at which the program is to be started when it comes to be executed. The directive has the following effects:-

(a) It terminates the current routine.
(b) All parameters and expressions which have been used are evaluated and inserted into the program.
(c) Library routines are found and inserted at the required places or at the end of the program, depending on whether they were called for by L directives or simply referred to.
Library routines that have been referred to in the program (but not explicitly called by L-directives) will be inserted in store locations immediately after the last item before the E directive. Note that if * directives have been used this is not necessarily the highest address used by the program, and care is required to ensure that library routines do not overwrite any part of the program. The library routines may be stored in, for example, locations A3 + 9 onwards by preceding the E directive by * = A3 + 9.
(d) Fault indications are printed out for parameters which are used but not set, and for any other faults encountered. If there are any faults, the program is normally suspended and not entered. (see section 12.6 for exceptions to this.)
(e) The compiler, which has occupied store locations above $\frac{3}{4} \times 2^{20}$ (that is, J3) is deleted from the store so that storage location numbers up to $\frac{7}{8} \times 2^{20}$ (that is, J34) may now be used. (Note: the Supervisor uses store locations from J34 upwards).

There are also two other enter directives which may be used. These are:-

(a) ER followed by an expression. The effect of ER is the same as E except for part (e). That is, the compiler and parameter lists are retained in the store. The program can then only use storage locations up to J3.
(b) EX followed by an expression. This is the enter interlude directive. All parameters which have been set and all expressions which can be evaluated are inserted into the program. The program is then entered at the specified location irrespective of unset parameters, and without the insertion of

library routines other than those called for by an L directive. The EX directive does not terminate a routine.

B1 to B88 are cleared before a program is entered by E, ER or EX. B89, however, contains the current value of *. After an E directive B90 contains J3; after ER and EX B90 is clear.

Any enter directive may be labelled, and the specified parameter, which is taken to belong to the routine terminated by the enter directive, is set to the current value of *. In the case of E and ER this setting is made after any necessary library routines have been inserted, so the label always refers to the address of the first available character location after the program.

F directive

F introduces a group of floating-point numbers (on the same line) and can also be used to increase the value of *, if necessary, to a full-word address.

H directive

H introduces half-word numbers (interpreted as 21-bit integers plus a single digit octal fraction) and also has the effect of increasing the value of *, where necessary, to a half-word address.

L. Library directives

Lc.d, where c and d are decimal numbers in the range 1-1999, calls for copy d of library routine c to be inserted at the program location indicated.

L, when followed by a terminator but no number, calls for a copy of all library routines mentioned earlier in the program, to be inserted at the program location indicated.

R. Routine directives

Rn, where n is a decimal integer in the range 1-3999, defines the start of a new routine.

S directive

The directive S precedes a group of 6-bit integers which will be stored in successive character positions.

T. Title directive

After reading T followed by new-line, the next line of characters is copied to the program output channel 0. The title directive can also be written as Ta or Ta-b where a and b are decimal integers. In the first case the next line will be copied to the program output channel a, in the second to channels a to b inclusive.

If desired the T, Ta or Ta-b may be terminated by comma or multiple space: the remainder of that line will then be ignored and again the next line will be treated as the title and copied to the output.

As with C directives, there is no line reconstruction of the text.

U. Unset Parameters directive

Un, where n is a decimal integer, causes the preset parameter Pn to be unset. Further, Un-m unsets from Pn to Pm inclusive.

Z. End routine directive

Z indicates the end of a routine. Usually this is not necessary, since a new R directive implies the end of the preceding routine; any program material between Z and the next R will be assigned to routine 0.

Chapter 6

THE REMAINING ACCUMULATOR INSTRUCTIONS

In Chapter 3 we described the accumulator and some of the basic accumulator instructions.

All the accumulator instructions operate on floating-point numbers. They may be divided into groups as follows:-

(a) Standardised rounded operations

(b) Standardised unrounded operations

(c) Unstandardised operations

(d) Test instructions.

The only standardised rounded instruction not so far introduced is

360	Standardise a, round am and check for exponent overflow	$am' = a$ QRE

6.1 Standardised Unrounded Operations

In these instructions, L is cleared before the operation, and after the operation the result is standardised as a double-length number in A. An interrupt occurs if the exponent overflows.

300	Add am and s	$a' = am + s$ QE
301	Subtract s from am	$a' = am - s$ QE
302	Negate am and add s	$a' = -am + s$ QE

The instructions are thus similar to 320-322 except that rounding does not occur.

The following instructions are like 300 and 301 except that L and Ls are not cleared initially.

They provide a limited form of double-length working; limited because the answer is only correct if $a_y \leq s_y$ (i.e. the exponent of s must not be less than the accumulator exponent)

310	Add s to a (pseudo double-length)	$a' = a + s$ QE (if $a_y \leq s_y$)
311	Subtract s from a (pseudo double-length)	$a' = a - s$ QE (if $a_y \leq s_y$)

Before two numbers are added or subtracted in the accumulator, the one with the smaller exponent is shifted down into L and its exponent increased accordingly until the two exponents are the same.

In the 310 and 311 instructions, if $a_y \leq s_y$ then a_x is shifted down correctly. If $s_y < a_y$ then s_x will be shifted down into L, and the original contents of L will be spoiled. In this case the definitions of 310 and 311 will be

310 $am' = am + s$, 1 spoiled. QE

311 $am' = am - s$, 1 spoiled. QE

Extracodes are provided for correct double-length working in all cases and these are described later.

Two store locations are needed to hold a double-length number and it is conventional to store both numbers as standardised numbers with the less-significant half always positive and with an exponent which is at least 13 less than that of the more-significant number.

The contents of the accumulator are

$$am + a1.8^{-13}$$

as the floating point number a1 has an exponent a_y which is 13 more than its true value.

The 355 instruction is provided to position a1 correctly.

355	Copy the special sign bit of L (ls) into all bits of M, then standardise	$a' = a1.8^{-13}$	Q
		$a' = a - am$ when ls = 0	Q

Example:

To store the accumulator contents for double-length working in locations 100 and 101

356	0	0	100	store am
355	0	0	J4	position a1 *
356	0	0	101	store $a1.8^{-13}$ (Q)

* Note: All accumulator instructions make a reference to the store and obtain a store operand, even if the function does not use it. Any store address within the program is of course allowed, but as operands are read from the fixed-store very much faster than from the core-store it is conventional to specify the first address in the fixed-store, J4, in such instructions.

Example:

Locations 100 and 101 contain two numbers to be regarded as a double-length number. Add this number into the accumulator, using locations 98 and 99 as working space.

356	0	0	98	store am
355	0	0	J4	position a1
320	0	0	101	add less-significant halves, QRE, in am.

356	0	0	99	store partial answer
324	0	0	98	replace original am
300	0	0	100	add most-significant halves, result is standardised but not rounded in a
356	0	0	98	store most-significant part of this
355	0	0	J4	position the rest
300	0	0	99	add in less-significant sum
310	0	0	98	final answer in a

(This program is extracode 1500)

The following instructions complete the standardised unrounded operations

340	Standardise a, check for exponent overflow	$a' = a$ QE
342	Multiply am by s, leaving the double-length product standardised in a	$a' = am.s$ QE
343	As 342 but multiply negatively	$a' = -am.s$ QE
366	Clear L, complement am if negative, and standardise	$a' = \|am\|$ QE
367	Clear L, copy the modulus of s to Am, and standardise	$a' = \|s\|$ QE

6.2 Unstandardised Instructions

The unstandardised instructions can be divided into four groups

(a) Those concerned with storing and loading the accumulator.

(b) Multiplications.

(c) Divisions.

(d) Miscellaneous.

6.2.1 The unstandardised instruction which store and load a are described below:-

356	Copy am into S	$s' = am$
357	Copy a1 (that is 1, 1s and ay) into S	$s' = a1$
346	Transfer am to S and clear A (a' = floating-point zero)	$s' = am$, $a' = 0$
347	Transfer a1 to S and clear L and Ls	$s' = a1$, $1' = 0$
344	Copy the argument and sign from S into L and Ls, leaving Am, including the exponent, unchanged	$1' = sx$
345	Copy the argument from S into L and the sign bit from S into Ls and all bits of M. Leave the exponents unchanged.	$1' = sx$, m' = sign of s, $ay' = ay$
314	Copy s into Am leaving L and Ls unchanged	$am' = s$, $1' = 1$
315	Copy s negatively into Am leaving L and Ls unchanged (AO will be set for $s = -1.0$)	$am' = -s$, $1' = 1$ AO

6.2.2 Unstandardised Multiplication Instructions.

372	Multiply am by s and leave the double-length product unstandardised in A. Clear the sign bit of L. Check for exponent overflow and accumulator overflow	$a' = am.s$ $1s' = 0$ EAO
373	As 372 but multiply negatively	$a' = -am.s$ $1s' = 0$ EAO
352	Multiply am by s, leaving the double-length product unstandardised in A, and set the sign bit of L equal to the sign of the product. Check for E and AO	$a' = am.s$ $1s'$ = sign of m EAO
353	As 352 but multiply negatively	$a' = -am.s$ $1s'$ = sign of m EAO

352 and 353 are identical to 372 and 373 except that they set Ls instead of clearing it. These instructions (352 and 353) are intended to form the single-length product of two unstandardised integers and leave the mantissa in L with the correct sign in Ls; they can therefore be redefined as

352 $l' = m.sx$ E

353 $l' = -m.sx$ E

Note, however, that the exponent ay' will be applicable to the double-length product in A, and that the accumulator overflow will not be set when the product overflows into M, but only when the double-length product overflows.

6.2.3 The Division with Remainder Instructions.

There are three division instructions which give the quotient in L and the remainder in M. However, these instructions only operate correctly for numbers which obey certain conditions.

There is a large range of division and remainder extracodes provided, which use these instructions and ensure the required conditions are fulfilled. For most purposes, it is easier to use these extracodes rather than the basic instructions. The only exception to this rule is the use of the 375 instruction for division of positive fixed-point integers, and this special case will therefore be described first:-

A fixed-point integer c can be represented in a 48-bit word by a fractional mantissa $sx = c \times 8^{-n}$, where n is normally 12 or 13, and an exponent sy, usually + 0 or + n. Provided that they are positive and that the divisor has $sx < \frac{1}{2}$, which even with $n = 12$ allows integers up to 30,000 million, the 375 instruction can be used to divide one such number into another. The dividend should first be placed in L, with M clear: this places the dividend in Ax with an additional scale factor of 8^{-13}, and ensures $ax < |sx|$ provided that $sx \neq 0$. The simplest case of the 375 instruction may now be defined as follows:-

375 Fixed-Point Integer Division

Divide a by the modulus of s, placing the quotient in L and the remainder in M. a and s must satisfy $0 \leq ax < \|sx\| < \frac{1}{2}$; the remainder will then lie in the range $0 \leq m' < \|sx\|$.	$l' = a/\|s\|$ E $m' =$ remainder $(0 \leq ax < \|sx\| < \frac{1}{2})$ $0 \leq m' < \|sx\|$ $ay' = ay - sy$

After obeying the 375 instruction the remainder m' will be scaled by the same factor as the dividend, and it should therefore be assigned the same exponent. The quotient l' will be an integer scaled down by 8^{-13} and it must be shifted up one octal place if it is required to store it with a scale factor of 8^{-12}.

Example:

Given two fixed-point integers c and d in A5 and 1A5, each stored with one octal digit after the point; e.g. c is stored with mantissa 8^{-12} c in A 5. Form the quotient and remainder of c/d, in the same form and with exponent 12, and store them in locations 7 and 8.

345	0	0	A5	$ax' = 8^{-48}\ (8^{-12}\ c)$ (i.e. $m' = 0$ and $1' = 8^{-12}\ c$)
375	0	0	1A5	$1' = 8^{-13}$ c/d $m' = 8^{-12}$ x remainder
121	124	0	12U12	$ay' = 12$
356	0	0	8	Remainder am to location 8
364	0	0	J4	$1' = 8^{-12}$ c/d
357	0	0	7	Quotient a1 to location 7

The full definition of the 375 instruction is as follows:

375 Pseudo Fixed-Point Division

Divide a by the modulus of s, placing the quotient in A1 and a form of "remainder", m', in M. If $m' \geq 0$ the true remainder $m'' = m'$, but if $m' < 0$, which will only occur when $m'' \geq \frac{1}{2}$, then $m'' = m' + 1.0$	$a1' = a/\lvert s\rvert$ E $m' =$ "remainder" $(0 \leq ax < \lvert sx\rvert < 1)$ True remainder $m'' = m'$ if $m' \geq 0$ $= m' + 1.0$ if $m' < 0$ $0 \leq m'' < \lvert sx\rvert$

a and s need not be standardised but they must satisfy the restriction $0 \leq ax < |sx| < 1.0$. The true remainder m" will then satisfy the constraint $0 \leq m'' < |sx|$. ay' is the exponent of the quotient. The exponent of the remainder is $ay - 13$, i.e. 13 less than that of the double-length dividend a before the operation.

If $sx \leq ax < 8\ |sx|$ or if $sx = -1.0$ the 375 instruction provides a quotient and remainder m" which are correct if regarded as floating-point numbers but which break the rules of fixed-point division. The remainder may be larger or smaller than with true fixed-point division, its exponent being as follows:

Condition	Exponent of Remainder
$\lvert sx\rvert \leq ax < 8\ \lvert sx\rvert$	$ay - 12$
$0 \leq ax < \lvert sx\rvert < 1.0$	$ay - 13$
$sx = -1.0$ and $ax \geq \frac{1}{8}$	$ay - 13$
$sx = -1.0$ and $ax < \frac{1}{8}$	$ay - 14$

When $|sx| \leq ax$ the adjusted remainder m" may not be exact, because the last octal digit of the correct remainder will have been lost.

If $ax < 0$ then am will be negated before the division takes place but l will not be adjusted.

If $a \geq 1.0$ then ax will be shifted down and its exponent increased by one before the operation.

If $sx = 0$ or $ax \geq 8\ |sx|$, the 375 instruction will not give a correct quotient or remainder.

376 Divide a by the modulus of s, placing the quotient in Al and the "remainder" in M. The dividend a must not be negative and the divisor s must be standardised before the 376 instructed is obeyed. The "remainder" is such that:

$a1' = a/|s|$ $(a \geq 0)$. EDO

Remainder

$m'' = m'$ if $m' \geq 0$

$= m' + 1.0$ if $m' < 0$

mantissa of true remainder $= m$ if $m \geq 0$

$= m + 1.0$ if $m < 0$

exponent of true remainder $= ay - 13$ if $|ax| < |sx|$

$= ay - 12$ if $|ax| \geq |sx|$

The quotient al' is not normally an integer; it is merely the unrounded representation of a/s to such accuracy as is possible in the 39 binary digits of L. The true remainder has no special significance other than that it represents a - s.al' and is always positive or zero. When $|ax| \geq |sx|$ the true remainder m" may not be exact, because the last octal digit of a - s.al' will have been lost.

Exponent overflow is checked for, and division overflow occurs if s is unstandardised or zero. If a is in standardised form before the division, al' will be a standardised quotient, but m' and m" may not be standardised.

377 Divide the modulus of am by the modulus of s, placing the quotient in Al and the "remainder", as defined for 376, in M. Check for E and DO. The divisor s must be standardised. If am is in standardised form before the division, al' will be a standardised quotient, but m' and m" may not be standardised.

$a1' = |am| / |s|$ EDO

Remainder

$m'' = m'$ if $m' \geq 0$

$= m' + 1.0$ if $m' < 0$

6.2.4 Miscellaneous Unstandardised Instructions

354 Round by adding. Add one to the least-significant digit of m if the most-significant digit of l is a one. Accumulator overflow can occur. The contents of L are unchanged.

$am' = a.R+$ AO

341 Check for exponent overflow. a is unchanged.

$a' = a$ E

361 Round am and check for exponent overflow.

$am' = a$ RE

6.3 Test Instructions

The following four instructions are tests on the accumulator mantissa, and comparable to the tests on bt or bm. Note that bm can be used to modify the address.

234	Place n in Ba if the accumulator contains zero.	$ba' = n$ if $ax = 0$
235	Place n in Ba if the accumulator does not contain zero.	$ba' = n$ if $ax \neq 0$
236	Place n in Ba if the accumulator contents are greater than or equal to zero.	$ba' = n$ if $ax \geq 0$
237	Place n in Ba if the accumulator contents are less than zero.	$ba' = n$ if $ax < 0$

All these test ignore the sign bit of L.

For the accumulator to contain zero, both guard bits must be zero; the most-significant guard bit, rather than the sign bit, determines whether the accumulator is greater or less zero. With standardised numbers this is immaterial, as the guard digits will be copies if the sign bit, and with fixed point working the correct result might still be obtained even if accumulator overflow had occured.

Examples:

1. Increase b3 by 0, 1 or 2 depending on whether am is >, = or < the contents of store location 16. Let A10 be the address of a register available for working space.

356	0	0	A10	store am
321	0	0	16	am - s
234	3	3	1	$b3' = b3 + 1$ if am = s
237	3	3	2	$b3' = b3 + 2$ if am < s
334	0	0	A10	restore am

2. B1 and B2 contain positive integers n1 and n2. Form n1 × n2 in store location 5 as a fixed-point integer, represented by mantissa $n1 \times n2 \times 8^{-12}$ and zero exponent. Replace b1 by the integer quotient n1/n2, and place the remainder from this division in B2. Let locations 6 and 7 be available for working space.

113	1	0	6.4	set b1, b2 in the store
113	2	0	7.4	as floating-point numbers
113	0	0	6	with zero exponents
113	0	0	7	
334	0	0	6	$am' = n1 \times 8^{-12}$
352	0	0	7	$a' = n1 \times n2 \times 8^{-24}$
365	0	0	J4	$a' = n1 \times n2 \times 8^{-25}$ i.e. $l' = n1 \times n2 \times 8^{-12}$
357	0	0	5	store $l = n1 \times n2 \times 8^{-12}$

345	0	0	6	set n1 in L with $m' =$ sign of n1 = 0. $ax' =$ n1 $\times 8^{-25}$
375	0	0	7	$1' = (n1/n2) \times 8^{-13}$, $m' =$ remainder $\times 8^{-13}$
356	0	0	6	store remainder am
364	0	0	J4	shift up quotient to integer position. $1' = (n1/n2) \times 8^{-13}$
357	0	0	7	store quotient $1 = (n1/n2) \times 8^{-13}$
101	2	0	6.4	set $b2' =$ remainder
101	1	0	7.4	set $b1' =$ quotient

Note that in this example it is not necessary to set the exponent zero after division because ay is made zero during the multiplication and both division operands have zero exponents.

Chapter 7

EXTRACODE INSTRUCTIONS

7.1 Introduction

The basic instructions consist in just those simple operations which the computer has been designed to execute directly. In the Atlas order-code, however, there are many complicated operations which the computer deals with in a special way; these are known as extracodes and are distinguished from the basic instructions by having a 1 in f_0, the most-significant bit of the 10-bit function number. Upon encountering an instruction with $f_0 = 1$, there occurs an automatic entry to one of many built-in subroutines, the choice being determined by the remaining three octal digits of the function number. The exit from the subroutine is again automatic, and the program proceeds in the usual way with the instruction next after the extracode, unless the extracode subroutine has initiated a jump.

7.1.1 Uses of the Extracode Instructions.

As their name implies, the extracodes provide an extension of the basic order-code, including both those complicated operations which are excluded from the basic instructions, and many of the facilities which on previous machines have been obtained by the use of library subroutines.

Amongst the arithmetic instructions provided by extracodes we may instance those in which the address, interpreted as a floating-point number, is used as an operand; double-length operations; and a full range of elementary functions such as logarithm, square-root, sine etc.

An important group of extracodes deals with the special requirements of input and output and also of magnetic tape transfers; the uses of these will be discussed at some length in Chapters 8 and 9.

The organisational extracodes comprise extensive facilities designed to assist the programmer in making efficient use of the operating system of Atlas. The various aspects of this are described in later Chapters (particularly Chapters 11 and 12).

7.1.2 To the programmer, extracode instructions appear as basic instructions. The two types of instruction can be freely intermixed, and after each instruction control passes sequentially to the next (except for jump instructions). It is therefore not strictly necessary to know how the computer deals with extracode instructions, although this is given for completeness in the next section.

There are 512 function numbers available for extracodes, 1000-1777. Of these, 1000-1477 are singly-modified instructions (B-type) and 1500-1777 are doubly-modified instructions (A-type). In some of the B-type instructions, bm is used as an operand so no modification takes place.

7.2 The Logical Interpretation of Extracode Instructions

When an extracode instruction is encountered the following action takes place:-

(a) The content of Main control, b127, is increased by one to the address of the next program instruction.

(b) The address is modified according to the type (i.e. N + bm for B-type, N + ba + bm for A-type) and the result stored in B119.

(c) The seven Ba digits are placed in bits 15-21 of B121, unless Ba is B122 in which case B121 is left unchanged; this enables B122 to be used to specify a B-register in extracode functions exactly as in basic functions.

(d) The function digits f1 - f9 are placed in extracode control, B126, as shown below.

Bit	0	1-9	10	11	12	13	14	15	16	17	18	19	20	21-23
Value	1	0 0 0 0 0 0 0 0 0	f1	f2	f3	0	0	f4	f5	f6	f7	f8	f9	000

(e) Control is switched from Main (B127) to extracode (B126).

The next instruction to be obeyed is now in the fixed store, under extracode control, at a location determined by the function digits. It is in one of 64 registers (given by f4-f9) in one of 8 tables at intervals of 256 words (given by f1-f3). The tables of 64 registers are called "jump tables". In general this instruction will be an unconditional jump into a routine which performs the required function. These routines are permanently stored in the fixed-store and written in normal basic instructions. Each routine terminates with an instruction in which f1 = f3 = 1 in the function number. This is obeyed as if f1 = 0 and then control is switched back to main control (e.g. 521 is equivalent to 121 followed by "extracode exit"). The next instruction to be obeyed is then the one whose address is in B127; if no jump has been initiated by the extracode this instruction will be the one immediately following the extracode instruction.

The routines that perform extracodes can use B-registers 91 to 99 inclusive and always use B119, B126, and B121 (unless Ba = 122).

Examples:

1. Extracode 1714 is defined as am' = 1/s
Replace the numbers in locations 100 to 105 by their reciprocals.

```
   121      1      0     5      set modifier/count
2)1714      0      1     100    am' = 1/s
   356      0      1     100    store
   203      127    1     A2     count
```

Each time the extracode instruction is encountered b127' = b127 + 1, b121' = 0, b119' = 100 + b1 + b0, b126' = J40034140 = +1804J4 and control is switched to B126. The instruction in the jump table is

```
121    126    0    A14
```

The instructions at A14 are

14)334	0	0	A96	set am = +1
774	0	119	0	374 division 1/s, then reswitch to main control.
96)+1				

2. Extracode 1341 is defined as $ba' = ba.2^n$ (arithmetic shift up) Shift b16 up by 2 more than the integer in B17

1341	16	17	2

This instruction sets b121' = 16D1, b119' = 2 + b17, etc. (Note that b16 is not added to b119 because 1341 is a singly-modified (B-type) extracode).

3. Shift the contents of B20 to B47 inclusive up by 5 places.

121	121	0	20D1	set B121 pointing at B20.
1)1341	122	0	5	shift. As Ba = B122, b121 is left unchanged when the extracode is entered.
172	121	0	47D1	bt' = b121 - 47D1
220	127	121	A1	If bt ≠ 0, b121' = b121 + 0.4 and b127' = A1

Example 3 illustrates the use that can be made of B121 and B122 in extracodes; this is the same as their use in basic instructions except that extracodes with Ba ≠ 122 will overwrite B121.

7.3 Allocation of Functions

The extracodes are divided into sections as shown below, though there are a few functions which do not fit into this pattern. References are given for those subjects described in this chapter.

Functions	Subjects	Reference
1000-1077	Magnetic tape routines, and Input and Output routines.	--
1100-1177	Organisational routines.	--
1200-1277	Test instructions and 6-bit character operations	7.6 & 7.5.2
1300-1377	B-register operations.	7.5.1
1400-1477	Complex arithmetic, vector arithmetic and miscellaneous B-type accumulator routines.	7.4.6 & 7.4.7
1500-1577	Double-length arithmetic and accumulator operations using the address as an operand.	7.4.4 & 7.4.5
1600-1677	Logical accumulator operations and half-word packing.	7.5.3 & 7.4.8
1700-1777	Arithmetic functions (log, exp, sq.rt., sin, cos, tan, etc.) and miscellaneous A-type accumulator operations.	7.4.1, 7.4.2 & 7.4.3

Not all of the 512 extracode functions have been allocated and, where convenient, constants and extracode programs have been packed into the vacant jump-table locations.

This means that the use of an unallocated extracode function may result in an 'unassigned function' interrupt or may cause some extracode to be entered incorrectly. The latter case would give the programmer wrong results.

In particular, the first location in the fixed store, J4, contains the floating-point number $\frac{1}{2}$. This causes an unassigned function interrupt if extracode 1000 is encountered, since J4 is the first register of the first jump-table. Note that floating-point zero is equivalent to the instruction

1000 0 0 0.

There follows a description of many of the extracodes. Where possible, the actual number of basic instructions obeyed in each extracode routine is given in the right hand column.

Appendix E gives an ordered summary of all the extracodes, for easy references.

7.4 The Accumulator Extracodes

7.4.1 The Most Used Arithmetic Functions

The following routines each have two extracode numbers. The first operates on s, which is standardised on entry. The second operates on a, which is standardised, rounded and truncated to a single-length number on entry. For this number we use the notation aq. The results are always standardised rounded numbers in Am.

1700	Place the logarithm to base e of s in Am.	$am' = \log s$	
1701	Place the logarithm to base e of aq in Am.	$am' = \log aq$	
1702	Place the exponential of s in Am.	$am' = \exp s$	43
1703	Place the exponential of aq in Am.	$am' = \exp aq$	42
1710	Place the square root of s in Am.	$am' = +\sqrt{s}$	≤42
1711	Place the square root of aq in Am.	$am' = +\sqrt{aq}$	≤41
1712	Form the square root of $(aq^2 + s^2)$ and place this in Am.	$am' = +\sqrt{aq^2 + s^2}$	≤50

Following the two arc sine extracodes, am′ is in radians, with $-\frac{\pi}{2} \leq am' \leq +\frac{\pi}{2}$

1720	Place the arc sine of s in Am.	$am' = \text{arc sin } s$
1721	Place the arc sine of aq in Am.	$am' = \text{arc sin } aq$

Following the two arc cosine ex tracodes, am′ is in radians, with $0 \leq am' \leq \pi$

1722	Place the arc cosine of s in Am.	$am' = \text{arc cos } s$
1723	Place the arc cosine of aq in Am.	$am' = \text{arc cos } aq$

Following the two arc tangent extracodes, am′ is in radians, with $-\frac{\pi}{2} < am' < \frac{\pi}{2}$

1724	Place the arc tangent of s in Am	$am' = \text{arc tan } s$
1725	Place the arc tangent of aq in Am.	$am' = \text{arc tan } aq$
1726	Divide aq by s and place the arc tangent of this number in Am. am′ is in radians and such that $-\pi < am' \leq \pi$.	$am' = \text{arc tan } (aq/s)$
1730	*Place the sine of s in Am.	$am' = \sin s$ 41
1731	*Place the sine of aq in Am.	$am' = \sin aq$ 40
1732	*Place the cosine of s in Am.	$am' = \cos s$ 42
1733	*Place the cosine of aq in Am.	$am' = \cos aq$ 41
1734	*Place the tangent of s in Am.	$am' = \tan s$ 34
1735	*Place the tangent of aq in Am.	$am' = \tan aq$ 33

* In 1730 -1735, s and aq must be in radians.

7.4.2 Other Floating-Point Arithmetic Functions

1704	Place the integer part of s in A.	$a' =$ int pt s	QE	5
1705	Place the integer part of a in A.	$a' =$ int pt a	QE	4
	See also 1300 and 1301.			
1706	Set $a' = +1$, 0 or -1 as s >, =, or < zero.	$a' =$ sign s	Q	5-6
1707	Set $a' = +1$, 0 or -1 as a >, =, or < zero.	$a' =$ sign a	Q	4-5
1713	Raise aq to the power s and place the result in am, provided that $aq \geq 0$, Fault if $aq < 0$.	$am' = aq^s$ $aq \geq 0$	QRE	
1714	Place the reciprocal of s in Am.	$am' = 1/s$	QREDO	4
1715	Place the reciprocal of am in Am.	$am' = 1/am$	QREDO	4
1754	Round am by R+, clear L and standardise.	$am' = a$, $l' = 0$	QR+	6
1756	Interchange the contents of S and Am (with no standardising)	$am' = s$, $s' = am$		8
1757	Place the result of dividing s by am in Am.	$am' = s/am$	QREDO	4
1760	Square the contents of Am	$am' = am^2$	QRE	3
1774	Divide am by s and place the result in Am. The original numbers need not be standardised.	$am' = am/s$	QREDO	10
1775	Divide aq by s and place the result in Am. The original numbers need not be standardised.	$am' = aq/s$	QREDO	9

1774 and 1775, besides providing a division instruction which operates on unstandardised numbers, store information which enables extracodes 1776 and 1407 to calculate a quotient and remainder.

1776	When used after division extracodes 1774, 1775, 1574 or 1575, with no other extracodes in between and am unaltered, the definition of 1776 is as follows:	$s' =$ quotient $am' =$ remainder	QREDO	13
	Place the quotient of the previous division in s and the remainder in Am, where the remainder has the sign of the divisor.			
1407	As 1776 except that the quotient is integral and is adjusted according to the sign of the remainder, which is specified by Ba as follows:	$s' =$ adjusted integral quotient $am' =$ remainder	QEDO	

Ba	Sign of remainder
0	Same as the denominator

1 Opposite to the denominator

2 Same as the numerator

3 Opposite to the numerator

4 Same as the quotient

5 Opposite to the quotient

6 Positive

7 Negative

1467 Evaluate the polynomial $s_0 + s_1 \cdot am + s_2 \cdot am^2 + \ldots s_{ba} \cdot am^{ba}$ where s_0 is the number at S, s_1 at S + 1, etc. and the order of the polynomial is given as an integer in Ba.

$$am' = \sum_{r=0}^{ba} s_r \cdot am^r$$

where $S_r = S + r$ QRE 6+3ba

1466 Multiply the two numbers at addresses (N + ba + bm) and (N + bm) and add the double-length result into the full accumulator.
Rounding takes place near the least-significant end of L.
(In detail, when the double-length product has been formed, its least-significant half is first added in M to the least-significant half of the original contents of A. This addition is rounded. The rest of the product and the original contents of M are then added into A without rounding).

$$a' = a + C(N+bm) \times C(N+ba+bm)$$

QE 18

1415 Generate pseudo-random numbers (PRN's) in A and S (or S*) from numbers in S and S*. This extracode may be used in several ways.

1. With digit 21 of S equal to 0, the PRN is placed in S and A.

   (a) If $s^*y = 0$, $sx > 0$ and $s^*x > 0$, then s' will be a PRN in the range 0 to 8^{sy}, rectangularly distributed and fixed-point (i.e. sx' is a fixed-point PRN and $sy' = sy$). a' will be a PRN in the range 0 to $s^*x \cdot 8^{sy}$ (with $al' = s'$).

   (b) If $s^*y = 0$, $sx < 0$ and $s^*x > 0$, then as (a) except that ranges become -8^{sy} to 0 and $-s^*x \cdot 8^{sy-1}$ to 0 respectively.

   (c) If $s^*y = 0$ and $s^*x < 0$, then as (a) except that the PRN's alternate in sign.

2. With digit 21 of S = 1, the PRN's are generated in S* and A instead of S and A. The cases are as for 1, interchanging S and S* throughout.

3. Two successive uses of the extracode, with digit 21 of S first = 0 and then = 1, and with $sy = s^*y = 0$, will set PRN's in S and S*, both rectangularly distributed in the range 0 to 1. A will contain the product of two PRN's and so will be distributed in the range 0 to 1 with the probability $-\log x.dx$ of being in the neighbourhood dx of x.

In all cases the generation process must be started with Sx and S^*x containing numbers with a random mixture of binary digits, but with their least-significant bits set to 1.

7.4.3 Accumulator functions suitable for Fixed-Point Working

1752	Shift ax up 12 octal places and subtract 12 from ay.	$m' = ax.8^{12}$ $ay' = ay - 12$	A0	10
1753	Shift m down 12 octal places in ax and increase ay by 12.	$ax' = m.8^{-12}$ $ay' = ay + 12$	A0	6
1755	Force ay to the number ny given in bits 0-8 of n, shifting ax up or down accordingly.	$ax' = ax.8^{ay-ny}$ $ay' = ny$	A0	17
1762	Shift ax up 12 octal places leaving ay unchanged.	$m' = ax.8^{12}$ $ay' = ay$	A0	9
1763	Shift m down 12 places in ax, leaving ay unchanged.	$ax' = m.8^{-12}$ $ay' = ay$	A0	5
1764	Shift ax up n octal places, leaving ay unchanged. If n is negative, shift ax in the opposite direction.	$ax' = ax.8^{n}$ $ay' = ay$	A0	17
1765	Shift ax down n octal places, leaving ay unchanged. If n is negative shift ax in the opposite direction.	$ax' = ax.8^{-n}$ $ay' = ay$	A0	12
1766	Place the modulus of s in Am, without standardising. Accumulator overflow will occur if s is -1.0.	$am' = \|s\|$	A0	4
1767	Place the modulus of am in Am without standardising. A0 will occur if am is -1.0.	$am' = \|am\|$	A0	3
1772	Multiply m by sx, shifting the result up by 12 octal places to be in M, and subtracting 12 from ay.	$m' = (m.sx)8^{12}$ $ay' = ay + sy - 12$	A0	11
1773	Divide a by s, and force ay equal to 12, shifting the result, which is in M, if necessary.	$m' = (ax/sx).8^{ay-sy-12}$ $ay' = 12$	A0	27
1452	Multiply am by s, forming the answer in Ax. Force ay to the number given in digits 0-8 of ba, and shift ax accordingly.	$ax' = m.sx.8^{ay+sy-bay}$ $ay' = bay$	A0	19-23
1473	Divide ax by sx, forming the answer in Ax. Force ay to the number given in digits 0-8 of ba, and shift ax accordingly.	$ax' = (ax/sx).8^{ay-sy-bay}$ $ay' = bay$	A0	24-28

Fixed-Point Divisions with Remainder

The three extracodes 1474, 1475 and 1476 each divide some part of the accumulator by the contents of store location S, placing an unstandardised quotient q in the location whose address is ba and leaving an unstandardised remainder r in Am. In all cases, r retains the original sign of am and has a mantissa in the range $0 \leq |rx| < |sx|$. The quotient is rounded towards zero. Division overflow is set if $sx = 0$ or -1.0 or if $|sx| \leq$ | mantissa of dividend |. Both DO and AO are set when the mantissa of the dividend is equal to -1.0.

If only the remainder is required, one can avoid the need to set ba by putting Ba = B126 in the extracode instruction.

1474	Divide am by s. The exponents of q and r are given by $qy = ay - sy$ and $ry = ay - 13$.	$C(ba)' =$ quotient (am/s) $am' =$ remainder (am/s)	DO AO E	20-29
1475	Divide a by s. The exponents of q and r are given by $qy = ay - sy$ and $ry = ay - 13$	$C(ba)' =$ quotient (a/s) $am' =$ remainder (a/s)	DO AO E	19-28
1476	Divide the integral part of am by s. The exponents of q and r are forced to $qy = 24 - sy$ and $ry = 12$. The condition $\lvert am \rvert < 8^{24} \lvert sx \rvert$ must be observed, otherwise division overflow will occur and the results will be meaningless. The least-significant octal digit of q is always zero, and it is intended that usually $sy = 12$ so that $qy = 12$ also and one is working with integers. (In the case $ay \leq -6$ and $am < 0$, this extracode must be preceded by 217, 124, 124, 0 to ensure the t rue integral part is used).	$C(ba)' = \left(\frac{\text{q int pt am}}{\text{s}}\right)$ $am' = r\left(\frac{\text{int pt am}}{\text{s}}\right)$	AO DO E	28-37

7.4.4 Double-Length Arithmetic

The double-length number s: is stored in two consecutive locations s and s + 1 as two standardised floating-point numbers, where $sy - 13 \geq s^*y$. s^* and a1 are assumed to be always positive. All arithmetic is standardised, rounded and checked for exponent overflow.

1500	Add s: to a	$a' = a + s:$	10
1501	Subtract s: from a	$a' = a - s$	10
1502	Negate a and add s:	$a' = -a + s:$	14
1504	Copy s: into a	$a' = s:$	4
1505	Copy s: negatively into a	$a' = -s:$	3

1542	Multiply a by s:	$a' = a.s:$	15
1543	Multiply a negatively by s:	$a' = -a.s:$	19
1556	Store a at S:	$s:' = a$	5
1565	Negate a	$a' = -a$	5
1566	Form the modulus of a	$a' = \|a\|$	4-6
1567	Copy the modulus of s: into A.	$a' = \|s:\|$	5
1576	Divide a by s:	$a' = a/s:$	19

7.4.5 Arithmetic Using the Address as an Operand

The modified address is taken as a 21-bit integer with an octal fraction. Fixed-point operations imply an exponent of 12.

1441	Store ba in S as a fixed-point number	$sx' = ba$, $sy' = 12$		5
1520	Add n to am	$am' = am + n$	QRE	10
1521	Subtract n from am	$am' = am - n$	QRE	9
1524	Place n into a	$a' = n$	Q	8
1525	Place n negatively into a	$a' = -n$	Q	7
1534	Place n into a, without standardising.	$a' = n$		10
1535	Place n negatively into a, without standardising	$a' = -n$		9
1562	Multiply am by n	$am' = am.n$	QRE	8
1574	Divide am by n	$am' = am/n$	QRE	16
1575	Divide aq by n	$am' = aq/n$	QRE	15

After 1574 and 1575, the extracodes 1776 and 1407 can be used to give a remainder and adjusted integral quotient. See section 7.4.2.

7.4.6 Complex Arithmetic

The "complex accumulator" Ca is taken as a pair of consecutive registers, the address of the first one given by the contents of Ba in the instruction. If Ba is B0, Ca will be locations 0 and 1. As with the double-length arithmetic, s: is a number pair consisting of the two numbers at addresses S and S + 1. For Ca and S:, the real part of the number is in the first location, the imaginary part in the second. Ca may coincide with S: if desired, but the two must not partially overlap, i.e. the difference between ba and S must not equal 1. The accumulator is used for the arithmetic so its original contents on entry are spoiled. All arithmetic is standardised, rounded and checked for exponent overflow.

1400	Place the logarithm of s: in Ca	$ca' = \log s:$	
1402	Place the exponential of s: in Ca	$ca' = \exp s:$	140
1403	Place the conjugate of s: in Ca	$ca' = \text{conj } s:$	5
1410	Place the square root of s: in Ca	$ca: = +\sqrt{s:}$	$\leq$117

1411	Place the argument of s: (radians) in Am.	am' = arg s:	
1412	Place the modulus of s: in Am.	am' = mod s:	≤53
1413	Form the numbers s cos s*, s sin s* and place these in Ca. (s* is in radians).	ca' = s.cos s*, s.sin s*	95
1414	Place the reciprocal of s: in Ca.	ca' = 1/s:	15
1420	Add s: to ca	ca' = ca + s:	8
1421	Subtract s: from ca	ca' = ca - s:	8
1424	Copy s: into Ca	ca' = s:	6
1425	Copy s: negatively into Ca	ca' = -s:	6
1456	Copy ca into S:	s:' = ca	5
1462	Multiply ca by s:	ca' = ca.s:	18

Note: 1400 - the imaginary part of the complex logarithm will lie in the range $-\pi$ (not inclusive) to π (inclusive).
1410 - of the two possible values of the complex square root, the one computed here has a non-negative real part; the remaining ambiguity about the square roots of negative real numbers is removed by computing the one whose imaginary part is positive.

7.4.7 Vector Arithmetic

The following instructions operate on two vectors s_1 and s_2. Both vectors consist of lists of floating-point numbers stored in successive locations. In each instruction the singly-modified address n gives the number of terms in the vectors (i.e. the order) and Ba gives the starting address of s_1. The next B-register after Ba, Ba*, gives the starting address of s_2. Address n must be a positive integer.

Besides their uses in vector and matrix arithmetic, these instructions can be used to manipulate lists of numbers in the store.

The accumulator is used in the arithmetic so its original contents on entry are lost. All operations are standardised rounded and checked for exponent overflow.

1430	Add the vector s_2, which consists of n successive numbers starting at C(ba*) into the vector s_1, which consists of n successive numbers starting at C(ba).	$s_1' = s_1 + s_2$	9 + 4n
1431	Subtract s_2 from s_1.	$s_1' = s_1 - s_2$	9 + 4n
1432	Multiply each term of s_2 by am and store the resultant vector at s_1.	$s_1' = am.s_2$	10 + 4n
1433	Multiply s_2 by am and add this to s_1.	$s_1' = s_1 + am.s_2$	10 + 5n

1434	Copy s_2 to s_1	$s_1' = s_2$	13 + 3n
1436	Form in Am the scalar product: $s_{10} \cdot s_{20} + \cdots + s_{1(n-1)} \cdot s_{2(n-1)}$, where $s_{10}, s_{11}, s_{12}, \cdots$, $s_{1(n-1)}$ are the numbers in S_1, and $s_{20}, s_{21}, \cdots$ are the numbers in s_2.	$am' = \sum_{i=0}^{n-1} s_{1i} \cdot s_{2i}$	
1437	As 1436 but forming the scalar product to double-length accuracy in a.	$a' = \sum_{i=0}^{n-1} s_{1i} \cdot s_{2i}$	10 + 13n

7.4.8 Half-Word Packing

Half-word floating-point numbers consisting of 8-bit exponents and 16-bit mantissae are sometimes useful for low-accuracy calculations where it is necessary to reduce store usage.

1624	Transfer the floating-point number at S into the accumulator, without standardising.	$a' = s$		6
1626	Copy a_y and the 16 most-significant digits of a_x into S after rounding this number in Am by forcing a one in its lowest bit if the rest of a_x is non-zero.	$s' = am$	R	8

7.5 B-Register Arithmetic

7.5.1 General B-Register Operations

1300	Place in Ba the integral part of the floating-point number s. Place the fractional part in Am.	ba' = int pt of s am' = frac pt of s	
1301	Place in Ba the integral part of am. Place the fractional part in Am.	ba' = int pt of am am' = frac pt of am.	

The following six instructions provide integer multiplication and division of ba by n.

For 1302 - 1304, ba and n are interpreted in the normal way as 21-bit integers with a least-significant octal fraction. In the multiplication instructions octal fractions are rounded away from zero, and overflow of the answer is not detected. The accumulator is used in the calculation, but am is preserved.

1302	Multiply ba by n and place the result in Ba.	ba' = ba × n	23-24
1303	Multiply ba negatively by n and place the result in Ba.	ba' = -ba × n	22-23
1304	Divide ba by n. Place the integer quotient in Ba and the remainder, which has the sign of the dividend, in B97.	ba' = int pt (ba/n) b97' = remainder	25-28

For 1312 - 1314, ba and n are interpreted as 24-bit integers, and the result is again a 24-bit integer.

1312	Multiply ba by n and place the result in Ba.	ba' = ba × n	23-24
1313	Multiply ba negatively by n and place the result in Ba.	ba' = -ba × n	22-23
1314	Divide ba by n. Place the integer quotient at the least-significant end of Ba and the remainder, which has the sign of the dividend, as a 24-bit integer in B97.	ba' = int pt (ba/n) b97' = remainder	

The following six instructions provide general n-place shifts of numbers in B-registers.

In arithmetic shifts, the sign digit is propagated at the most-significant end of the register for shifts to the right (i.e. down).

In logical shifts the sign digit is not propagated.

For both arithmetic and logical shifts the result is unrounded on shifts down. In circular shifts, digits shifted off the most-significant end of the register reappear at the least-significant end and vice-versa. n is an integer in bits 0-20 as usual, with no octal fraction. (If n has an octal fraction the answer may be wrong by a shift of one place). In each case, if n is negative a shift of n places in the opposite direction occurs.

1340	Shift ba arithmetically to the right by n places.	$ba' = ba.2^{-n}$	10-22
1341	Shift ba arithmetically to to the left by n places.	$ba' = ba.2^{n}$	9-21
1342	Shift ba circularly to the right by n places.	$ba' = ba.2^{-n}$, circular shift	10-19
1343	Shift ba circularly to the left by n places.	$ba' = ba.2^{n}$, circular shift	9-18
1344	Shift ba logically to the right by n places.		10-21
1345	Shift ba logically to the left by n places.		9-20

The following are miscellaneous arithmetic instructions on half-words and index registers.

1347	Perform the logical "OR" operation on ba and s and place the result in S.	$s' = ba \vee s$	5
1353	Set B123 by writing n to it, and read the result to Ba. This sets ba equal to the position of the most-significant 1 bit in bits 16-23 of n. (B123 is described in Chapter 4.)	$b123' = n$, then $ba' = b123$	
1356	Set the B-test register as the result of non-equivalencing ba and s.	$bt' = ba \not\equiv s$	7
1357	Set Bt as the result of non-equivalencing ba and n.	$bt' = ba \not\equiv n$	5
1376	Set Bt as the result of collating ba and s.	$bt' = ba \,\&\, s$	5
1377	Set Bt as the result of collating ba and n.	$bt' = ba \,\&\, n$	
1364	Preserve the digits of Ba where there are zeros in n and copy digits from Bm into Ba where there are ones in n.	$ba' = (ba \,\&\, \bar{n}) \vee (bn \,\&\, n)$ [also $b119' = (ba \not\equiv bm) \,\&\, n$]	4
1371	Dummy extracode to set up b121 and b119.	$b121' = Ba$, $b119' = N + bm$.	

1771 Dummy extracode to set up b121 and b119. $b121' = Ba$, $b119' = N + ba + bm$.

7.5.2 Character Data Processing

1131 Search for s in table starting at C(ba). If s can be found, ba' will record its address, otherwise the sign bit of ba' will be set to 1. Main control is re-entered at c' = c + 2, and C(c + 1) is used to specify parameters k, l, m as shown below. Up to l + 1 half-words are scanned, starting with C(ba) and continuing at intervals of k half-words, each being masked with m before comparison with s.

bits	0-9	10-20	21-23	0-23
	k	l	spare	m
	interval	count		mask

In the following two instructions S is taken as a character address, the octal fraction giving the address of the 6-bit character within the word.

1250 Place the character s in the least-significant 6 bits of Ba and clear the other digits of Ba. $ba' = \text{char } s$ 7-10

1251 Copy the character from the least-significant 6-bits of Ba into the character position at S, leaving the other characters in the word unaltered. $s' = \text{char ba}$ 11-18

In the following two instructions ba is interpreted as a character address, and the content of the next B-register, ba*, is interpreted as a half-word address. n is used as a count and its octal fraction must be zero.

1252 Unpack n characters. The n characters, packed in successive character positions starting at C(ba), are placed in the least-significant 6-bits of n successive half-words starting at C(ba*). The other digits in each half-word are set to zero. $16 + \text{int pt } (6\frac{3}{4}n)$

1253 Pack n characters. Take the n characters stored in the least-significant 6-bits of n successive half-words starting at C(ba*) and pack these into n successive character positions starting at C(ba). $18 + 5n$

7.5.3 Logical Accumulator Instructions.

B98 and B99 are used in these instructions as a double-length B-register. This is called the logical accumulator and denoted by G.

1204	Starting at the most-significant end, count the number of 6-bit characters which are identical in g and s, continuing only until the first dissimilar characters are found. Place the result in Ba.		10-31
1265	Shift g up by 6 places, writing overspill to Ba, and add n.	ba' = m.s. character of g. $g' = 2^6 g + n$	11
1601	Copy s into G.	$g' = s$	3
1604	Add s into G.	$g' = g + s$	7
1605	Add s into G, adding any overflow carry in again at the least-significant end.	$g' = g + s$ with end-around carry	12
1606	Non-equivalence s with g	$g' = g \not\equiv s$	4
1607	Collate s with g	$g' = g \,\&\, s$	3
1611	Replace g by its logical binary complement.	$g' = \tilde{g}$	3
1613	Copy g into S	$s' = g$	3
1615	Copy g into Am, without standardising.	$am' = g$	4
1630	Form the logical binary complement of s and collate this with g.	$g' = g \,\&\, \tilde{s}$	5
1635	Copy am into G.	$g' = am$	4
1646	"OR" s with g	$g' = g \vee s$	3
1652	Set Bt by the result of subtracting s from g.	$bt' = g - s$	7-9

7.6 Test Instructions

7.6.1 Accumulator Test Instructions

1200	Place n in Ba if the Accumulator overflow (AO) is set. Clear AO.	$ba' = n$ if AO is set.	9
1201	Place n in Ba if AO is not set. Clear AO.	$ba' = n$ if AO is not set.	7
1234	Increase main control by 2 (instead of by 1) if am is approximately equal to s.	$c' = c + 2$ if $am \simeq s$	11
1235	Increase main control by 2 if am is not approximately equal to s.	$c' = c + 2$ if $am \not\simeq s$	11

For 1234 and 1235, approximate equality is defined as

$$\left|\frac{am - s}{am}\right| < C(ba)$$

am must be standardised on entry. By definition, if $am = 0$ then am is not approximately equal to s.

1236	Place n in Ba if am is greater than zero.	$ba' = n$ if $am > 0$	4-6
1237	Place n in Ba if am is less than or equal to zero.	$am \leq 0$	3-5
1255	Place n in Ba if m is neither zero nor all ones.	$ba' = n$ if $m \neq$ all 1's or all 0's	
1727	Depending on whether am is greater than, equal to, or less than s, increase main control by 1, 2 or 3.	$c' = c + 1, 2, 3$ as $am >, =, < s$	7
1736	Increase main control by 2 if the modulus of am is greater than or equal to s.	$c' = c + 2$ if $\lvert am \rvert \geq s$	
1737	Increase main control by 2 if the modulus of am is less than s.	$c' = c + 2$ if $\lvert am \rvert < s$	

In 1234, 1235, 1727, 1736 and 1737 am is preserved but l is not.

7.6.2 B-register Test Instructions

1206	Place n in Ba if the most significant 6-bit character in G is zero.		4

1216	Place n in Ba if bm is greater than zero.	$ba' = n$ if $bm > 0$	
1217	Place n in Ba if bm is less than or equal to zero.	$ba' = n$ if $bm \leq 0$	
1226	Place n in Ba if bt is greater than zero.	$ba' = n$ if $bt > 0$	4-6
1227	Place n in Ba if bt is less than or equal to zero.	$ba' = n$ if $bt \leq 0$	3-5
1223	Place n in Ba if the B-carry digit is set.	$ba' = n$ if $bc = 1$	4

The B-carry digit (Bc) is set to a 0 or a 1 in the following basic instructions.

100 102 104

110 112 114

120 122 124

150 152 164

170 172

Bc records the final carry or borrow generated after the addition or subtraction of the most significant digits of the operands.

When the most-significant digit is taken as a sign bit, which is usually the case, Bc is not a true overflow digit. For example, adding -1 or +1 gives 0 and also sets Bc = 1 as there is a final carry. (See Chapter 4).

7.7 Subroutine Entry

1100	Set link in Ba and enter subroutine at s.	$ba' = c + 1$, $c' = s$	6
1101	Set link in Ba and enter subroutine at n.	$ba' = c + 1$, $c' = n$	5
1102	Set link in Ba and enter subroutine at bm.	$ba' = c + 1$, $c' = bm$	6
1362	Set link in B90 and enter subroutine at n. On completion of this extracode, $b121' = Ba$, so that Ba or ba may be used to carry information into the subroutine.	$b90' = c + 1$ $c' = n$	3

The link set in Ba can be picked up as an exit from the subroutine by the instruction

121 127 Bζ 0

where Bζ is the address of the B-register (Ba) in which the link was set. It is conventional to use B90 for this purpose, and 1362 was provided for that reason.

7.8 Miscellaneous Operations

1117 End program. This extracode is used to end a program unless it is monitored by the Supervisor. Certain information about the program is output, and the program cleared from the computer. (see Chapter 11.)

1120 Record the time in Ba, with hours, minutes and seconds each given by two decimal digits specified in four bits apiece, as follows:- ba' = clock

bits	0-3	4-7	8-11	12-15	16-19	20-23
	tens	units	tens	units	tens	units
	HOURS		MINUTES		SECONDS	

1121 Record the date in Ba, with the day, month and year each given by two decimal digits specified in four bits apiece, as follows:- ba' = date

bits	0-3	4-7	8-11	12-15	16-19	20-23
	tens	units	tens	units	tens	units
	DAY		MONTH		YEAR	

1124 Set the central computer V-store line 6 to n. The least-significant six bits of V-line 6 are used as follows:- v6' = n

bit	Set to 0	Set to 1	
18	13 shift	12 shift	on division; needed to adjust remainder for 376, 377 instructions
19	Qs $\geq$ 0	Qs $<$ 0	Sign of quotient in basic division orders
20	A0 clear	A0 set	
21	bt $\geq$ 0	bt $<$ 0	
22	bt = 0	bt $\neq$ 0	
23	Bo clear	Bo set	

1125 Collate the contents of the central computer V-store line 6 with n, placing the result in Ba. Any digits of v6 may thus be read. ba' = v6 & n

Chapter 8

INPUT AND OUTPUT

This chapter explains the use of library routines L1 and L100 and the input and output extracodes. It also shows the form in which input data must be prepared and explains the internal code used by Atlas as well as the handling of 'pure' binary information.

8.1 Introduction

8.1.1 Peripheral devices.

Atlas can control a large number of input and output mechanisms and these peripheral devices are of various forms. Five- and seven-hole paper tape and punched card readers are used for input, while output may be produced on paper tape, cards or a line printer.

The working rate for such peripherals is of the order of 10^3 characters per second. This is very slow compared with the processing rate of the central computer, which is of the order of 2×10^6 characters per second. Clearly, conventional methods of input and output, in which each character is read from a peripheral as required or written away as soon as it is produced, would lead to considerable inefficiency in Atlas. For this reason, a 'buffering' system is adopted.

8.1.2 The System Input and Output Tapes.

All the information required for a program is fed through the slow peripherals and is automatically stored in a standard form on a magnetic tape. This tape is called the System Input Tape. Work on the program will not begin until all the required information is stored in this way. Similarly the output from a program is stored on magnetic tape (the System Output Tape) and transferred to the peripheral only after the program is completed. In this way the central computer can be occupied with other work while the actual transfer to and from the peripheral takes place.

8.1.3 Internal Code Input and Output.

Normally during the transfer from peripheral to the System Input Tape each character (i.e. a paper tape character or card column) is automatically translated into a six-bit character in Atlas Internal Code. Similarly each character for output is represented in this code on the System Output Tape. The six-bit characters are packed 8 to a word and stored in 'records' (see 8.3 below).

Seven-track input is checked for odd parity and rejected on parity failure. Rejection is prevented by preceding the seven track information by ***P, in which case an even parity character is replaced by the fault character K7.7 (six binary ones) and reading continues. There is no parity check when reading 5-track tape. If the first column of a card is a non-

standard punching the card is stored in binary (see 8.13); if the first column is standard but subsequent columns are non-standard they are represented by the fault character K7.7.

Equivalent characters on punched cards, five- or seven-hole tape and the line printer, are represented by the same character in internal code. In this way one input routine can deal with input from any peripheral and one output routine can build up output for any output mechanism.

Thus, for example, the character 'M', whether it has been read from five- or seven-hole tape or from cards, will be represented on the System Input Tape as

K5.5 or, in binary 101101.

Similarly if 'M' is to be output it will be stored on the System Output Tape in the same way, regardless of the output peripheral for which it is intended. Further details of the internal code are given in 8.2.

Input and output in internal code may be carried out by either the input and output library routines L100 and L1 or by input and output extracodes. Both forms are described in detail below.

8.1.4 Binary Input and Output.

The programmer can, if he wishes cause a direct binary representation of the holes on the paper tape or cards to be stored on the System Input Tape, instead of the internal code representation. This is known as binary input and must be read by extracode and not the library routine L100. Each character so stored will occupy 12 bits and characters will be packed four to a word.

For example, 'M' is represented on seven-hole paper tape by the character

1011.101

where ones represent holes and the full stop represents a sprocket hole. As binary input this would be stored on the System Input Tape as

000 001 011 101

On the other hand the character 'M' is represented on a punched card by holes in the '11' and '4' positions. As binary input this is stored as the 12 bit character

010 000 100 000

(see 8.13 and 8.16)

In a similar fashion, by the use of extracodes, a twelve bit character may be output in binary form. It will be stored on the System Output tape in this form and, when sent to the peripheral, holes will be punched in positions corresponding to the one bits.

Binary input and output are particularly useful when dealing with non-standard paper tape or card codes. More details are given in 8.13 and 8.16.

8.2 The Internal Code.

Each character is represented in internal code by six bits. When a single six-bit character is held in a 24-bit index register it is usually stored in the least significant six bits. It is thus represented by two octal integers separated by a point, and this notation will be used throughout this chapter.

The use of six bits imposes an upper limit of 2^6 or 64 distinct characters. To extend this number, two sets of characters are introduced, the inner set and the outer set. Most of the commonly used characters are contained in the inner set. A character in the outer set will have the same six-bit representation as an inner set character but they will be distinguished in the following way.

Every line, whether input or output, begins in inner set and all characters are taken to be inner set until a 'shift to outer set' character is encountered. All characters will then be interpreted as outer set members until a 'shift to inner set' character or the end of the line is reached.

On output, an internal code character which can not be represented by a character on the required peripheral will be replaced by a full stop on the Anelex printer or the card punch, and by erase on 5 or 7 track tape. Thus an attempt to print ≥ on an Anelex printer would result in a full stop being printed. This also applies to non-printing characters such as back space and tabulate which are only available on seven-hole tape.

8.2.1 Abbreviations.

In the Internal Code Table of 8.2.2 the following abbreviations are used:-

BS = Back Space	SI = Shift to Inner set
ER = ERase	SO = Shift to Outer set
FS = Figure Shift	SP = SPace
FT = Fault	TB = TaBulate
LC = Lower Case	UC = Upper Case
LS = Letter Shift	UL = UnderLine

The availability of characters on the different peripheral devices is indicated as follows:-

u	All peripheral devices
a	Anelex Line Printer
7	Seven-hole paper tape
c	Punched cards
5	Five-hole paper tape

Characters in parentheses are alternatives available on commercial seven-hole or five-hole paper tape codes, as used on Orion and Pegasus.

8.2.2 The Internal Code Table

Internal Code	Character Inner Set		Outer Set		Internal Code	Character Inner Set		Outer Set	
0.0					4.0	'(n)	u		
0.1			SF	u	4.1	A	u	a	7
0.2	TB	7			4.2	B	u	b	7
0.3	BS	7	£	a	4.3	C	u	c	7
0.4			SO		4.4	D	u	d	7
0.5			SI		4.5	E	u	e	7
0.6			LC/LS	7/5	4.6	F	u	f	7
0.7			UC/FS	7/5	4.7	G	u	g	7
1.0	(	u			5.0	H	u	h	7
1.1	)	u			5.1	I	u	i	7
1.2	,	u			5.2	J	u	j	7
1.3	π (£)	u			5.3	K	u	k	7
1.4	?	u	Stop	7	5.4	L	u	l	7
1.5	&	a7c	Punch On	7	5.5	M	u	m	7
1.6	*	u	Punch Off	7	5.6	N	u	n	7
1.7	/	u	:	a7c	5.7	O	u	o	7
2.0	0	u	$\emptyset$(X)	5	6.0	P	u	p	7
2.1	1	u	[	a7c	6.1	Q	u	q	7
2.2	2	u	]	a7c	6.2	R	u	r	7
2.3	3	u	$\rightarrow$	5	6.3	S	u	s	7
2.4	4	u	$\geq$	5	6.4	T	u	t	7
2.5	5	u	$\neq$	5	6.5	U	u	u	7
2.6	6	u	UL	a7c	6.6	V	u	v	7
2.7	7	u	|	a7c	6.7	W	u	w	7
3.0	8	u	$\uparrow$ (%)	a7	7.0	X	u	x	7
3.1	9	u	$\leq$ (v)	5	7.1	Y	u	y	7
3.2	<	a7c	α (10)	a7	7.2	Z	u	z	7
3.3	>	u	β (11)	a7	7.3				
3.4	=	u	$\frac{1}{2}$	a7	7.4				
3.5	+	u	10	a	7.5				
3.6	-	u	11	a	7.6				
3.7	.	u			7.7	FT	7c	ER	75

8.2.3 Shifts and Case Changes

UC, LC, FS, and LS are not stored on input when they are used to change from one case or shift to another. Each of these characters merely alters the meaning of the characters which follow on the paper tape and this alteration will be allowed for when these characters are translated into internal code.

Thus the sequence

Tape (7)

on seven-hole paper tape would be punched as

T LC a p e SP (UC 7 LC)

but it would be stored on the system input tape as

T SC a p e SP SI (7)

Redundant shift characters, such as FS when already in figure shift are stored, however, since they may indicate a fault.

Similarly UC, LC, FS and LS need not be used when preparing output. The internal code representation of the character is specified and shift and case changes will be automatically inserted where necessary.

For example outputting the internal code characters

W2

to five-hole paper tape will cause the characters

LS W FS 2

to be punched.

8.3 Carriage Control Characters and Records

As mentioned in 8.1.3 input is stored on the System Input Tape in records. These records correspond to one line of printing on paper tape or one card if the input is on cards. The last character of a record is called a carriage control character and is not represented in internal code. Carriage control characters have a special code of their own. Input from paper tape or punched cards can give rise to only the following carriage control characters:-

Code	Character
2.1	Newline (7-hole tape) or End-of-card
4.0	Paper Throw (7-hole tape)
2.0	Carriage Return (5-hole tape)
0.1	Line Feed (5-hole tape)

These are abbreviated to NL, PT, CR and LF respectively.

On output, records are built up on the System Output Tape. The last character of each record is interpreted as a carriage control character as follows:-

Codes	Effect
0.0 to 1.7	0 to 15 line feeds without carriage return
2.0 to 3.7	0 to 15 line feeds with carriage return
4.0 to 4.7	Paper throw on channels 0 to 7 without CR
5.0 to 5.7	Paper throw on channels 0 to 7 with CR

Carriage control facilities, and hence the interpretation of carriage control characters, vary from one output device to another.

The number of line feeds is always performed correctly but the following restrictions apply to other facilities.

On the Anelex Line Printer

Line feed and Paper Throw are always accompanied by carriage return.

Channel 0 is the 'top of form' channel. Thus 4.0 and 5.0 mean carriage return to the top of the next form. Channels 1 to 7 provide vertical spacing (always with carriage return) as determined by a loop of paper tape which is fitted to the paper throwing mechanism of the Anelex printer.

The width of a line printed on the Anelex is 120 characters. If more than 120 characters are output to a line a new line is automatically begun and the excess characters are printed on it.

On the Card Punch

Line feed means 'next card'.
Paper throw is replaced by one 'next card' (i.e. 2.1)

Carriage return is ignored.

If more than 80 characters are output to a card a new card is automatically begun and the excess characters punched on it, beginning at the first column.

On Seven-hole Tape

Line feed is always accompanied by carriage return. Carriage return without line feed (i.e. 2,0) is ignored. Paper throw is never accompanied by carriage return. The paper throw character on tape will only take effect if the flexowriter on which it is printed has a paper throw facility.

On Five-hole Tape

Paper throw is replaced by one line feed.

The channel number for paper throw will be taken modulo m, where m is the number of homing channels available on the printer.

The carriage control character 0,0 is ignored by each of the four types of equipment. The character 2,0 (carriage return without line feed) is ignored on 7-hole and card output and correctly done on the line printer and 5-hole tape. Compound characters can therefore be printed on the Anelex and the teleprinter by overprinting.

8.4 Selecting Input and Output

The data required by a program may arise from several different sources and each such batch of data may be prepared as a separate unit called a document. Each document will be on a separate paper tape or deck of cards and each will be allotted an input number as described in Chapter 10. Similarly if several distinct sections of output are to be produced by the program each may be given an output number.

When the program requires input from a given document it first selects the number of that document by extracode. Similarly to send information to a given output stream, the output number must first be selected. The extracodes for these purposes are given below; each one is singly modified.

1050 Select Input n

All succeeding input operations, until the next 1050 instruction, refer to Input n.
If no input document with number n has been defined there will be an exit to the monitor routine.
If input instructions are obeyed without previously selecting an input, Input zero is used (see Chapter 10).

1051 Find Selected Input

ba' = number of currently selected input.
This instruction is particularly useful in subroutines. The current input may be stored at the beginning of the routine by

```
1051    6    0    0    b6' = select input.
```

Another input may then be selected and the original one restored at the end of the subroutine by the instruction

```
1050    0    6    0    Select input b6
```

1060 Select Output n

All succeeding output instructions, until the next 1060 order, refer to output n.
For internal code output n is written without an octal fraction (or with an even octal fraction which will be disregarded).
If binary output is required n should have an odd octal fraction (usually .1). Thus

```
1060    0    0    3.1
```

will select output 3 for binary output.
If output n has not been defined there will be an exit to the monitor routine. If output instructions are obeyed without previously selecting an output, output zero is used (see Chapter 10).

1061 Find Selected Output

ba' = number of currently selected output plus octal fraction as in 1060.
This extracode is used in a similar fashion to 1051.

8.5 Input using L100

Since input and output require fairly complicated programs it is usually convenient to use library subroutines for these purposes. L100 is the Input Library Routine. It will reconstruct a record and present the programmer with a number or a character. It will also read texts.

8.5.1 Line Reconstruction

L100 itself calls in L199 to reconstruct records from the system input tape. The record or 'line' is reconstructed as follows:

a) Each character is stored at the least significant end of a half word.

b) Shifts to inner and outer sets will not be stored. Instead each simple character will be allotted seven bits. The most-significant bit will be zero if the character is a member of the inner set and a one if it belongs to the outer set. Thus 'M' is stored as K5.5 and 'm' as K15.5.

c) The character backspace (BS) will be correctly interpreted but not stored. That is, characters before and after a BS will be combined to form a compound character (see 8.6.3).

d) The character tabulate (TB) is not stored. Instead the correct number of spaces will be inserted (see 8.7.6).

e) The last character of a line will be the carriage control character.

f) The following characters have special representations instead of their normal internal code values:-

Space - SP is stored as 0.0
Erase - ER is stored as J4
Fault - FT is stored as J3
Underline - UL is stored as J2
Figure Shift - FS is stored as J1 (two or more successive figure shifts in the same record are stored as a single J1)

These special representations occur only when reading a reconstructed line; in all other cases, the normal internal code value is read. For example, SP, when read as a single character, or as a terminator to a number, will be represented by 0.1.

Normally only two lines may be reconstructed in this way at one time. Thus only two inputs may be active at the same time (but see 8.7.4)

Texts are read for output using L1; they are not line reconstructed.

8.5.2 Entries to L100

Several entries are provided to L100 to enable the programmer to read

numbers or characters from the reconstructed line. If numbers are read they will be properly translated into forms acceptable by the accumulator or a B-line.

For numerical input the character erase (ER), or any compound character containing ER, will be ignored. For character and text input ER is read.

For all entries to L100, the link must be stored in B90.

A list of entries is given here; further details are to be found in 8.6.

A1/L100	am' = next number QR
A2/L100	b81' = next integer
A3/L100	b81' = next character
A4/L100	Lose the rest of the line
A5/L100	Read text to store line b89 onwards
A6/L100	Read text, following T newline or T/newline to store line b89 onward
A7/L100	b81' = next integer as a 24 bit integer
A8/L100	b81' = next integer plus 3 bit octal fraction
A9/L100	Print reconstructed line

8.5.3 Data Preparation for L100

The following rules must be obeyed when punching data for L100. Throughout this section

a and c are decimal integers
b is a decimal fraction
and k is a one digit octal fraction.

a) A maximum line length of 160 characters plus a carriage control character is allowed (but see 8.7.5).

b) All numbers must be punched in the form:-

< Layout characters><Number><Terminator>

The layout character may consist of any combination of

spaces,
tabs,
new lines,
paper throws,
upper and lower case shifts,
carriage returns,
line feeds,
figure shifts
or back spaces.

No layout character is really necessary.
The terminator may be a

space,
tab,

new line (or end of card)
carriage return followed by line feed,
comma,
paper throw,
or line feed.

c) Numbers to be read by entries to A1/L100 will normally take the form

$$\pm a.b$$

where the '+' may be omitted if desired.
If either a or b is zero it may be omitted; if b is omitted the decimal point is optional.

Example:

10	+7	-4	31.	+167	-8.
.17	+.3	-.761	34.61	+0.357	-26.4

Numbers in floating point form will also be accepted by L100 with this entry. These must be in the form

$$\pm a.b(\pm c)$$

where c is a decimal exponent. a and b are as before; c must be preceded by a '+' or '-'. No spaces may occur between b and the character 'c' or within the brackets.

For example

$$3.141(+7)$$

will be accepted. It has value 3.141×10^7

d) For entry at A2/L100 or A7/L100, numbers must take the form

$$\pm a$$

where the '+' is optional and $a \leq 2^{21} - 1$ for A2/L100 and $a \leq 2^{24} - 1$ for A7/L100

Example:

64	+731	-2

e) For entry at A8/L100, numbers must take the form

$$\pm a.k$$

where '+' is optional and $a \leq 2^{21} - 1$. If either a or k is zero they may be omitted; if k is omitted the octal point is optional.

Examples:

14	+2	-51	12.	+21.	-741.
.7	+.2	-.5	61.1	+7.0	-0.4

f) Texts for input by entry at A5/L100 will normally be punched as one record ending with a carriage control character. If a text takes up more than one record (i.e. more than one line of paper tape or more than one card) the characters ((must be punched at the end of each record except the last.

g) Texts for input by entry at A6/L100 must be preceded by T newline or T/newline (a warning character other than T may be used by the programmer).

h) No texts are line reconstructed.

8.5.4 Punching Errors.

a) If the routine meets an unacceptable character when searching for the beginning of a number, control is switched to A21/L100. For example this would happen on meeting Z in the list of integers

121 316 -7 Z

A21/L100 is normally a fault routine within L100 but can be a private fault routine set by the programmer (see 8.7.2). The disallowed character will be in B82.

b) If the routine meets an unacceptable character in the middle of a number, control is switched to address A22/L100. Again, the programmer may set A22/L100 for private action and the disallowed character is in B82 as before. Such an entry will be caused for example, by attempts to read a number with a fractional part by entry A2/L100. (See 8.7.3)

8.6 The Entries to L100 in Detail

All entries apply to the currently selected input.

8.6.1 A1/L100.

A number is assembled from the line reconstruction and is stored as a floating point number in Am. It is rounded in the normal Atlas fashion and standardised, and L is cleared.

The terminator is given in B82. If the terminator is also a carriage control character the most significant bit of B82 is set as one. Thus, when the last number of a line has been read, B82 will be negative.

For example if the number is terminated by a space, b82 will equal 0.1. If the terminator is new line, however, B82 will contain J4K2.1.

Example:

Read ten numbers from input 2 and store them in floating point form in locations 40 to 49.

121	3	0	-9	b3′ = count
1050	0	0	2	select input 2
5)1101	90	0	A1/L100	Enter L100 to read to Am
356	0	3	49	Store number
201	127	3	A5	Count and return.

8.6.2 A2/L100.

A 21-bit integer is formed from the reconstructed line and stored in bits 0 to 20 of B81; bits 21 to 23 are cleared. The terminator is given in B82 in the same way as with entry A1/L100.

Example:

Read to the end of the line of integers on input 1 and store them in half-words from 12.4 onwards.

121	1	0	0	b1′ = count
1050	0	0	1	Select input 1
121	90	0	3*	Set link
1)124	1	0	0.4	Increase count by one half-word
121	127	0	A2/L100	Read next integer to B81
113	81	1	12.0	Store integer
216	127	82	A1	Test for carriage control character

8.6.3 A3/L100.

A single character or a compound character is taken from the line reconstruction and stored in B81. A single (seven-bit) character goes into bits 17 to 23. The characters ER, FT, UL and FS will have the values given in 8.5.1 (but SP = 0.1).

A compound character arises from seven-hole tape input whenever two or three characters are punched, separated by backspaces. In the case of two characters, during line reconstruction, the numerically smaller character is stored in bits 17 to 23 of the half-word; the other character in bits 10 to 16. In the case of three characters the third will be stored in bits 3 to 9 regardless of numerical value. A punching sequence such as A BS A will be interpreted as A. Such a compound character will be read to B81 by the use of entry A3/L100.

A Compound character may also include the underline character. In this case bit one will be a 1 (i.e. J2 will be added). If a compound character consisting of more than three simple characters, apart from underline or erase, is detected during line reconstruction then a fault is registered and the program suspended. Details are printed on output zero.

For example, if the characters

0 BS / BS UL

are punched on seven-hole tape, the compound character $\not{0}$ will be printed by the flexowriter. This will be reconstructed as follows:-

010	0 000 000	0 010 000	0 001 111
UL	No 3rd character	0 is inner 2.0	/ is inner 1.7

The instruction

1101 90 0 A3/L100

to read this character would produce

B81' = J2K401.7

After the last character of a line has been read, the next entry to A3/L100 causes B81 to be cleared. The carriage control character will be given in B82.

8.6.4 A4/L100

The routine will set up the conditions necessary for line reconstruction on the next entry to L100. In this way the remainder of the currently reconstructed line is lost.

8.6.5 A5/L100

Read the next record(s) from the current input and store it beginning at the half word whose address is given in B89. The number of characters stored will appear in bits 1 to 23 of the half word whose address was specified in B89. The count includes the carriage control character. The final contents of B89 is the address of the half word which immediately follows the stored test. (For texts of several lines see section 8.6.6).

Any amount of carriage control information preceding the text will be treated as part of the text and stored ready for output by A6/L1. In detail, upon entry to A5/L100 all one-character records are read and stored, each preceded by its character count plus J4, until a record containing more than one character is encountered. The latter is stored as described in the preceding paragraph.

8.6.6 A6/L100

This entry is to be made to read a text which appears amidst numerical data at a point not necessarily known in advance to the programmer. The text must be preceded by T Newline or T/Newline according as it is desired later to output the text on a new line or on the current line. (In fact any non-numerical warning character can be used in place of T). The text is then read in by entering A6/L100 via the 'illegal-character-in-place-of-a-number' entry A21/L100 (see 8.7.2). The latter label must therefore be set by the programmer. For example if the following data are being read by A2/L100

```
1, 2, 3, T
TEXT
4, 5, 6
```

then the instructions

```
21/L100) 121   89   0   3:
         1362   0   0   A6/L100
```

will store two records when the T is met. The first stored will consist of a single new line character and the second will be the entire record which follows the T on input. Each record is stored with its character count as described in 8.6.5. The count for the first record has bit 0 equal to 1 as required by the entry A6/L1 to print the text.

Upon exit from A6/L100 B89 is set to the address of the next available half word following the stored text.

On entry to A5/L100 or A6/L100 for a text consisting of one record, the next record from the system input tape will be taken and placed in the store beginning at the address specified in B89. This must be a full word or half word address and the first half word will be used to hold a count of the characters in the record including the carriage control character. The count will be in the character position of the half-word, so for a text of 14 characters the count will be 1.6.

If the continuation mark ((has been used all of the records of the text will be read to store, beginning at location b89. Each record will be preceded by a half word containing a count of the characters and each of these half words except the last will also have bit zero set to one. The text will consist of all characters up to but not including the characters ((, and these characters must be followed immediately by new line.

Example:

Suppose a text on seven-hole tape consisted of the two lines

```
TAPE 12  ((
BLOCK 3
```

The instructions

```
121    89   0   100
1362   0    0   A6/L100
```

would set the contents of half word 100 onwards as follows:-

(100)	= J4K1.1	9 characters including 2 SP and one NL plus a continuation mark
(100.4)	= K6441604.5	TAPE
(101)	= K0121220.1	Sp 12 Sp
(101.4)	= J21	NL
(102)	= K1.0	8 characters including one SP and one NL.
(102.4)	= K4254574.3	BLCC
(103)	= K5301232.1	K sp 3 NL

8.6.7 A7/L100

An integer is formed from the line reconstruction and stored in B81 with its least significant digit as bit 23. The terminator is given in B82.

8.6.8 A8/L100

A 21-bit integer with one octal digit after the point is formed from the line reconstruction and stored in B81. The least significant digit of the integer occupies bit 20 and the octal digit occupies bit 21 - 23.

The terminator is given in B82.

8.6.9 A9/L100

The reconstructed line containing the last information read with L100 is printed on the current output. For this entry, the line is taken to consist of the characters on the printed line, together with the carriage control character following it.

8.7 Optional Parameters of L100

Library routine L100 contains seven parameters which are optionally set. These may be given alternative values by the programmer if he wishes. If L100 is called for explicitly by an L directive these parameters must be set first. This for example allows ABL to leave the right amount of space in the compiled program to contain L100. If L100 is called for implicitly the parameters may be set at any point in the program. In this case L100 is stored somewhere after the compiled program and the exact space it will occupy is irrelevant at the time of the implicit setting.

The parameters are:-

A20/L100 - beginning of line reconstruction storage
A21/L100 - Routine for fault at beginning of number
A22/L100 - Routine for fault in the middle of a number
A23/L100 - Maximum number of active input streams
A24/L100 - Maximum line length in characters
A25/L100 - Tab settings
A26/L100 - Tab routine

These are dealt with individually below.

8.7.1 A20/L100

The programmer may allocate specific storage for the beginning of line reconstruction by setting A20/L100, e.g. A20/L100 = 1000. If A20/L100 is not set by the programmer A20 will follow L199

8.7.2 A21/L100

If during numerical input a spurious character is encountered instead of a number, control is switched to A21/L100 with the character in B82. The programmer can write his own fault routine to deal with such a situation. This is particularly useful in dealing with a number list of unknown length.

Example:
Suppose the list is punched out and the terminator of the last number is followed by the character '*' (internal code 1.6). The following piece of program would read the numbers from input 2 and form their sum in A5. If a spurious character other than * is met (owing to mispunching) control is transferred to store line A6.

```
      1050     0     0     2          Select input 2
       346     0     0     J4         Clear am
    4)1101    90     0     A1/L100    am' = next number
       320     0     0     A5         am' = partial sum
       356     0     0     A5         (A5)' = partial sum
       121   127     0     A4         Go to read next number
    5)+0
21/L100)172    82     0     1.6        bt' = b82 - '*'
       225   127     0     A6         Go to fault routine )
       - - - - Next Instruction        if not *            )
```

If A21/L100 is not set by the programmer a standard fault routine will be entered.

8.7.3 A22/L100.

If during numerical input a spurious character is encountered within a number, control is transferred to A22/L100 with the character in B82. The programmer may write his own fault routine to take care of this situation but he will have a partially assembled number to deal with.

If A22/L100 is not set by the programmer, a standard fault routine will be entered.

8.7.4 A23/L100.

The maximum number of inputs active at one time (i.e. those with a line reconstructed part of which remains to be read) is normally two. The programmer can, if he wishes, alter this by setting A23/L100 to the number he requires. Thus the directive

A23/L100 = 4

would permit four streams to be active at once. This would obviously involve more store being used by the line reconstruction routine to accommodate the reconstructed lines.

8.7.5 A24/L100.

The maximum line length accepted by L100 from any peripheral is 160 characters (excluding carriage control information). If the programmer wishes to use a different line length he must set A24/L100 to the number of characters he requires.

8.7.6 A25/L100.

This parameter is optionally set within the library routine to 15 which gives the standard tab settings of

8, 8, 8, 8, 16, 16, 16, 16 . . . . . .

By setting A25/L100 to -1 or 999 the programmer can arrange for tabs of 16, 16, 16, ... or 8, 8, 8, ... respectively.

8.7.7 A26/L100.

During line reconstruction a standard routine for replacing 'tab' by the correct number of spaces begins at location A26/L100. If the programmer wishes to use a private routine to deal with tab, A26/L100 must be set to the starting address of this routine. Exit from the private routine must be to 1A28/L199.

8.8 Fault Printing by L100

If a fault is encountered while using L100, then control is transferred to a fault routine, unless the programmer has set the appropriate optional parameter. The fault routine will print out an indication of the fault, usually in the form

< explanatory text >

a, b, c

< reconstructed line >

where a is the input stream number

b is the position of the faulty character on the line

and c is the number of characters on the line.

It may not always be possible to completely reconstruct the line.

The routine then ends the run of the program.
The explanatory texts are listed below.

IMP. CH. DURING NUMBER < character >
The character is impermissible within a number.

IMP. CH. BETWEEN NUMBERS < character >
The character is not allowed to separate numbers

INTEGER TOO LARGE
The integer can not be held in the B-line.

IMP. COMPOUND CH.
Impermissible compound character.

IMP. CH. AFTER T
Only /, erase, or newline are allowed after T when using the text input entry A6/100.

UNASSIGNED C. C. CH.
The carriage control character is represented in the computer by a number greater than 5.7.

MAX. LINE LENGTH EXCEEDED
The line length is set by A24/L100

UNASSIGNED CH.
SPARE CH.

These last two texts refer to the characters indicated in Appendix D.

TOO MANY ACTIVE STREAMS
L100 has been asked to deal with more than the number of input streams set by A23/L100. No character position or line reconstruction is printed out.

8.9 Output using L1

L1 is the Output Library Routine and will output a number from the accumulator, an integer from a B-line, a single simple or compound character, or a group of characters forming a text. In actual fact this routine transfers the output information to the System Output Tape whence it is automatically sent to the required peripheral when output is completed. For this reason the single output routine can be used regardless of the output peripheral.

8.9.1 Entry points to L1.

Different entry points are provided for each type of output required and the way in which numbers are to be output is specified by a style number in a B-line before entry to L1.

For all entries the link is stored in B90.

The entry points are as follows:-

A1/L1 Output am in style b89
A2/L1 Output b81 in style b88
A3/L1 Output one character from b81
A4/L1 End line (or card)
A5/L1 End record by carriage control character in B87
A6/L1 Output a text from b89 onwards
A7/L1 Output a text from b89 onwards, with terminating carriage control character from B87

Further details of each entry are given below.

8.10 The Entries to L1 in Detail

All entries apply to the currently selected output and length is limited only by the output peripheral.

8.10.1 A1/L1.

Entry to L1 at this point causes L to be cleared and the contents of the accumulator to be output as a fixed or floating point number. The style of output is determined by the contents of B89, which must be set before entry to L1 by an order of the form

121 89 0 p:q.k

Here p is the number of decimal digits required before the point

$(0 \leq p \leq 127)$

q is the number of decimal digits required after the point

$(0 \leq q \leq 15)$

and k (the octal digit) indicates the form in which the number is to be printed as follows:-

k = 0 Accumulator printed fixed point, signed, on same line
1 Accumulator printed floating point, signed, on same line
2 Accumulator printed fixed point, signed, on new line
3 Accumulator printed floating point, signed, on new line
4 Accumulator printed fixed point, unsigned, on same line
5 Accumulator printed floating point, unsigned on same line
6 Accumulator printed fixed point, unsigned, on new line
7 Accumulator printed floating point, unsigned, on new line

If k is zero, .k may be omitted.

Further details are as follows:-

a) All numbers will be correctly rounded to the last digit printed. The rounding is decimal and of the 'add 5' variety.

b) The contents of the accumulator will be spoiled, but the contents of B80, B88, B89 are preserved. On the other hand, B81 to B86 are destroyed.

c) When k takes the value 2, 3, 6 or 7, the current line is terminated by a single new line character before the output of the number. Otherwise the number follows the last character of the current line.

d) For k = 0 to 3 the number is printed signed, that is preceded by SP for positive numbers or - for negative numbers (but see 8.11.2). With k = 4 to 7 the Sp or - are omitted altogether; for example both +2.5 and -2.5 would be printed 2.5.

e) If k is odd the number is output in floating point decimal form. The mantissa is printed with one non-zero decimal digit before

the point and q digits after (i.e. $1 \leq$ mantissa < 10) and is followed by the exponent. Zero in floating point form will have mantissa of 0. followed by q zeros and an exponent of +0. The number of character spaces to precede the decimal point is p+1 for signed numbers or p for unsigned numbers.

The floating point number appears in the form

mantissa (exponent)

with a two decimal digit exponent preceded by + or - and enclosed in brackets (but see 8.11.4 and 8.11.5.)

A non-significant left hand zero of the exponent will be omitted and a space will be output after the final character (i.e. the ')'). If the exponent is more than two digits it will be printed in full but layout will be spoiled (but see 8.11.4).

f) If $k = 0, 2$ the number is printed in fixed point form with p+1 character positions before the point and q after; if $k = 4, 6$ there are p character positions before the point and q after. Left hand zeros of the integral part are replaced by spaces; right hand zeros of the fractional part are always printed. Positive numbers are printed without a sign.

If the number has more than p places before the decimal point (say P places) it will be printed either as a fixed point number with P places before and q after, or as a floating point number with (p+q) significant figures in the mantissa, whichever form has fewer characters. In either case layout will be spoiled.

g) The input routine L100 uses index registers B88 and B89. Care must therefore be taken not to spoil the style set for an L1 entry by first entering L100.

h) The special cases of $p = 0$ and $q = 0$ are dealt with as follows:-

$\underline{p = 0}$

The integral part of the number in fixed point and of the mantissa in floating point will appear as 0. Thus 0.25 output in style 0:2 would appear as

sp 0.25

while -1.25 output in style 0:3.1 would be printed as

-0.125 (+1) sp

$\underline{q = 0}$

An integer with no decimal point will be printed.

$\underline{p = q = 0}$

The number is printed as sp0 or -0 with a decimal exponent if necessary.

Examples:

1. Print the numbers stored in locations 60 to 79 on output 3.

The numbers should be printed signed in floating point form, each on a new line, with four decimal places after the point.

```
1060     0      0     3        Select output 3
121      1      0     -19      b1' = modifier
121      89     0     1:4.3    Style in B89
1)324    0      1     79       Read next no. to Am
1362     0      0     A1/L1    Print a
201      127    1     A1       Count and return
- - - - Next Instruction
```

2. Read the next number from input 2 and print it on output 1 on the current line. The number should be fixed point, unsigned with four places before and two places after the decimal point.

```
1050     0      0     2        Select input 2
121      90     0     2*       Set link
121      127    0     A1/L100  Read one number
1060     0      0     1        Select output 1
121      90     0     3*       Set link
121      89     0     4:2.4    Set style
121      127    0     A1/L1    Print number
- - - -  Next Instruction
```

8.10.2 A2/L1.

Entry to A2/L1 causes a 21 or 24 bit integer from B81 to be printed. The style of output is determined by the contents of B88 which must be set before entry to L1 by an order of the form

```
121    88    0    p:q.k
```

Signed integers are printed with p+1 character positions before the point, if any (see b) below), and unsigned integers with p character positions before the point, if any. Left hand zeros of the integer are replaced by spaces. Positive signed integers are preceded by a space rather than a plus sign (to print something other than a space, see 8.11.2).

k is interpreted as for accumulator output except that the distinction between fixed and floating point numbers is replaced by that between 21-and 24-bit integers. Thus the interpretation of k is as follows:-

```
k = 0  b81 is printed as a 21-bit integer, signed, on same line
    1  b81 is printed as 24-bit integer, signed, on same line
    2  b81 is printed as a 21-bit integer, signed, on new line
    3  b81 is printed as a 24-bit integer, signed, on new line

k = 4  b81 is printed as a 21-bit integer, unsigned, on same line
    5  b81 is printed as a 24-bit integer, unsigned, on same line
    6  b81 is printed as a 21-bit integer, unsigned, on new line
    7  b81 is printed as a 24-bit integer, unsigned, on new line
```

Further details are:-

a) The contents of the accumulator and of B81 are spoiled, but the contents of B87, B88 and B89 are preserved. B80 is not used but B82-86 are overwritten.

b) If k is even b81 is treated as a 21-bit integer. If $q = 0$, bits 0 to 20 are taken as a pure integer and bits 21 to 23 are ignored. If q takes any non-zero value the integer is followed by a point and one octal digit taken from bits 21 to 23.

c) If k is odd b81 is printed as a 24 bit integer. The value of q is irrelevant.

d) If the integer has more than p digits it will be printed correctly but layout will be spoiled.

Example:

Print the number 97 on a new line on output 1 followed by the contents of store half-word 97 as a 21-bit integer with octal fraction. Allow for six characters before the octal point of the integer.

```
1060    0     0    1        Call
121     88    0    2:0.6    Style in B88
121     81    0    97       b81' = 97
1362    0     0    A2/L1    Print '97'
121     88    0    5:1.0    Style in B88
101     81    0    97       b81' = (97)
1362    0     0    A2/L1    Print (97)
```

8.10.3 A3/L1.

Entry at this point will cause one character, which may be simple or compound, to be printed from B81. Simple characters must be placed in the least significant six bits of B81 with bit 17 indicating whether the character is inner or outer set as 8.6.3.

An alternative way of printing an outer set character is by output of the 'shift to Outer Set' character, K0.4. All succeeding characters are then specified by six bits only and will be printed as outer set characters until the 'shift to Inner Set' character, K0.5 is encountered or the end of the record is reached. Thus to print outer set characters from the next record a further 'shift to Outer Set' character must be output.

Compound characters may also be built up in B81 as described in 8.6.3. Entry at A3/L1 will unpack each character and output them separated by backspaces. (This facility can be employed only if output is to 7-hole paper tape since this is the only output medium with a BS character). Underline characters are allowed in the same way as shown in 8.6.3.

The characters ER, FT, UL and FS may take either the forms adopted by L100 (i.e. J4, J3, J2 and J1) or their internal code numbers (K17.7, K7.7, K12.6 and K0.7). The fault character, which normally arises only from input via L100, is output as erase underlined. The actual printing of the fault character on the different peripherals is as follows (see 8.2).

```
7-track   ER
5-track   ER ER ER
Anelex    .. —
Cards     .. —
```

All compound characters containing J4 (i.e. with bit zero set to one) will be output as erases. If an attempt is made to print an impossible character (e.g. [on 5-hole) a full stop will be printed instead.

Example:
Print, on the current line of output 6, which is a seven hole paper tape punch, the characters

Ab Ø

121	81	0	K4.1	b81' = "A"
1362	0	0	A3/L1	Punch A
121	81	0	K14.2	b81' = 'b'
1362	0	0	A3/L1	Punch b
121	81	0	K0.1	b81' = 'SP'
1362	0	0	A3/L1	Punch SP
121	81	0	J2K401.7	b81' = Ø
				Punch /BS 0 BS UL

While entry A3/1 is useful for the output of compound characters, it is more efficiont to use the extracode 1064 to output single characters (see 8.15).

8.10.4 A4/L1

With this entry, the current line of output is terminated by a single new line (or end-of-card) character. The extracode 1065 0 0 2.1 has exactly the same effect and should normally be used.

8.10.5 A5/L1

If more than one new line character, or some other carriage control output is required, entry may be made to A5/L1. B87 must contain the carriage control information as detailed in 8.3.

Example:
For six new lines the necessary instructions are

121	87	0	K2.6
1362	0	0	A5/L1

The extracode 1065 0 87 0 has the same effect, and should normally be used.

8.10.6 A6/L1

Entry to A6/L1 will output a text from store locations beginning at b89. The text must be stored in the form in which texts are read by L100, (see 8.6.6). That is

a) each record of the text must be preceded by a half word containing a count of the number of characters in the record including the carriage control character.

b) for each record except the last this half word must also have bit zero set to one (i.e. for n characters, the half word must contain YnJ4).

c) characters must be six-bit, internal code, and must be packed eight to the word.

Example:

The text stored by L100 in the example at the end of paragraph 8.6.6 could be printed on output 4 by the instructions:-

```
1060    0    0    4       Select output 4
121     89   0    100     b89' = 100
1362    0    0    A6/L1   Print text.
```

8.10.7 A7/L1

Entry to A7/L1 will cause the output of a text in the same way as entry to A6/L1 except that the carriage control character terminating the (last) record of the text is replaced by a carriage control character specified before entry in B87.

In particular, if further information is to follow on the same line as the last line of the text, entry should be to A7/L1 with B87 zero. The last line of text will not then be terminated.

8.11 Optional Parameters of L1

As with L100, several optionally set parameters are contained within L1 which may be reset by the programmer if he wishes. They are:-

A21/L1 mask for p
A22/L1 mask for q
A25/L1 'sign' for positive number from accumulator
A26/L1 'sign' for positive number from B81
A27/L1 number of characters of floating point exponent
A28/L1) characters before and after floating point
A29/L1) exponent.

Details are given below.

8.11.1 A21/L1, A22/L1

p and q are normally taken modulo 128 and 16 respectively, and this is sufficient for most users. However, it is possible to change these values (which must be powers of 2) by setting suitable masks in A21 and A22. Thus directives

```
A21/L1 = 255:
A22/L1 =  63
```

would cause p to be taken modulo 256, and q modulo 64. The maximum values of p and q are 2048 and 512 respectively.

8.11.2 A25/L1

As stated in 8.10.1 and 8.10.2 positive signed numbers are normally preceded by a space in place of a sign. If it is required to output some other character instead, usually a '+' character, A25/L1 should be set to the internal code value of the character required.

Thus,

```
A25/L1 = K3.5
```

will produce plus signs before positive numbers, including integers.

8.11.3 A26/L1

In a similar fashion, parameters may be set to the internal value of a character to be printed before positive signed numbers from B81. It is optionally set to K0.1 (SP) within the library routine.

8.11.4 A27/L1

The standard form of the exponent of a floating point number, output by entry A1/L1 has five characters consisting of brackets, a sign and two decimal digits. This may be altered by the programmer to the total number of characters required, say

```
A27/L1 = 7
```

If the exponent has less than A27/L1-3 decimal digits, enough spaces will be output after the final bracket to make up the difference. For example

with seven characters, an exponent of 3 would be printed

(+3) SP SP SP.

8.11.5 A28/L1 and A29/L1

These parameters are set to the internal values of the characters before and after the exponent for floating point output. If not set by the programmer they will have the values K1.0 and K1.1 respectively to produce '(' and ')'.

Should the programmer wish for other characters in these positions he may reset A28/L1 and A29/L1.

Thus with

A27/L1 = 5
A28/L1 = K1.2 (comma)
and A29/L1 = K0.1 (space)

The floating point number 3.16×10^{17} will be output as

SP 3.16,+17 SP

instead of the normal

SP 3.16(+17)

If the last character is to be omitted completely, A29/L1 may be set to zero.

8.12 Input and Output by Extracode

Although sufficient for most purposes, L100 and L1 do not cope with the input requirements of all programs. In particular they cannot deal with paper tapes or cards punched in a non-standard code (i.e. binary input or output). To deal with such cases the input and output extracodes are provided.

These extracodes read six-bit characters from the records of the System Input Tape and write characters to the System Output Tape in preparation for punching or printing. Since information on the Input Tape is in records, provision is made for reading either single characters, groups of characters, or complete records. Output may be formed in similar units.

It is important to note that when L100 is used to read input it takes a complete record from the System Input Tape and reconstructs it before presenting the programmer with a character, a number or a text. For this reason the library routine and the input extracodes can not be used to work on the same input record. Output records may, however, be built partly by L1 and partly by extracodes.

The lack of line reconstruction also makes compound characters impossible with extracode input. Thus a seven-hole tape character may be altered or erased by another character, later in the record. For example, suppose a data tape contained the sequence of characters:

b BS ER a

By reading with L100, the first three characters would be combined to form a compound character and, since it contained erase, this character would be ignored. An entry at A3/L100 to 'read next character' would thus produce the character 'a'. Using an extracode to 'read next character' instead would produce 'b'. Such effects as this must be taken into account by the programmer.

8.13 Binary Input and Output

Input and output extracodes may work with either the 12-bit binary characters described in 8.1.4 or 6-bit internal code characters.

On input from punched tape, data documents are read and stored in internal code until one of the markers ***B, ***F or ***E is encountered. These markers are stored. Subsequently a direct binary representation of the input characters is stored as 12-bit characters on the System Input Tape. Any card with a non-standard code punched in its first column will also be stored as binary input. Definitions of the binary markers are given in the chapter on Job Documents.

Seven track input is checked for odd parity when reading binary input and the tape is rejected if an even parity character is encountered. However, if ***P precedes the binary markers ***B, etc., then even parity characters will be accepted and stored as for odd parity characters.

Binary information for five or seven track tape is stored with the three hole side of the punching in the least significant position. Thus the letter M if read in binary from seven track tape would be stored as

000 001 011 . 101

or if read from five track tape, as

000 000 010 . 110

The . indicates the position of the sprocket hole on the tape.

On binary input a punched card is represented by 80 twelve bit characters followed by one six bit zero carriage control character. Each column is stored as a 12-bit binary character with the top bit of the column stored as the most significant bit and the bottom bit as the least significant. Thus one column of a standard ICT punched card is stored as

10 11 0 1 2 3 4 5 6 7 8 9

For output, as described in section 8.4 the least significant bit of the 1060 extracode determines whether output is in internal code or binary.

All binary information, whether input or output is stored as one record with a 0.0 carriage control character. A carriage control character encountered within a section of binary input will be treated as a normal binary character. Thus NL, represented on seven-hole tape by

0000.010

will be stored on the System Input Tape as the 12-bit character

000 000 000 010

Binary input may be used to read tapes or cards punched in non-standard code, since the programmer can provide his own translation routine for the binary representation. Similarly he can cause his results to be punched in a non-standard form by these means.

8.14 The Input Extracodes

The Select Extracodes have been described in section 8.4. The remaining input extracodes are defined below. Each is singly modified and refers to the currently selected input.

1052 Find Input Device Number

ba' = V-store address of the peripheral equipment used for the currently selected input.
If this input originated as output from another program, ba' = 0.
For input from 5-track tape the least significant bit of ba' (i.e. bit 23) is 1; otherwise it is 0. The V-store addresses are described in appendix C.

1053 Test Binary / Internal Code

If the next character to be read from the currently selected input stream is a binary character, ba' = n.
If the next character is in internal code, ba is unaltered.
If there are no characters remaining on the currently selected input stream, an exit is made to the monitor routine.

1054 Read next character to Ba / Jump to n at end of Record

This extracode reads the next 6-bit character from the currently selected input, and places it at the least significant end of ba. With internal code input this will transfer one internal code character. With binary input, where the information is stored in 12-bit characters, the first use of the extracode will read the six most significant bits of the binary character. The next use of the extracode will read the six least significant bits. Normally control will then pass to the next instruction (i.e. b127' = b127+1) but if the last character, apart from the carriage control character, has previously been read, b127' = n and Ba contains the carriage control character in bits 18 to 23.
If all characters of the currently selected input stream have been read, this extracode causes an exit to the monitor routine.

1055 ba' = Number of Blocks Read

This sets in Ba the number of 512 word blocks read from the selected input. In internal code each block holds 4,096 characters, but some of these are carriage control characters and record counts used on the System Input Tape. In binary code, one block holds 2,048 twelve-bit characters.

1056 Read ba characters to S

Before using this extracode the number of 6-bit characters required must be set in the character position of ba. For example, for 18 characters ba must be set to 18D3 or 2.2. The ex-

tracode will then read the next ba characters from the current record of the selected input and place them in Store locations beginning at the half-word address S. Four six-bit characters are packed in each half-word. Bits 22 and 23 of S and bit 0 of ba are ignored.
If the end of the record is not reached, ba is unaltered on exit except for bit 0 which is set equal to one.
If the end of the record is reached no further characters are read and Ba contains the number of characters read in bits 1 to 23. Bit 0 is set to zero. The last character read will be the carriage control character.
If all the characters in the currently selected input stream have already been read, this extracode causes an exit to the monitor routine.

1057 Read next record to S

This extracode reads the next record and places it in the store starting at the half-word address specified in S. Characters will be packed, four six-bit characters to the half-word and bits 22 and 23 of S will be ignored. The last character will be the carriage control character.
On exit Ba contains in bits 1 to 23 the number of 6-bit characters read and bit 0 is zero.
If the record has been partly read, by use of 1054 or 1056, the remaining part of the record is read.
If all records of the currently selected input have been read, this extracode will cause exit to the monitor routine.

Extracodes 1056 and 1057 will run very much faster if no characters have previously been read from the record, or if the number which has been previously read is a multiple of four. Both these extracodes use far fewer instructions per character than does 1054, and are therefore much superior for large amounts of input.

Examples:

1. Input stream 3 consists of one record in internal code followed by a binary marker. Neglect this record and read all the binary information after the marker to store locations 1000.4 onwards, packing the 12-bit characters four to a word. Place the number of 12-bit characters in half-word 1000.

```
1050    0    0    3          Select input 3
1057    1    0    1000.4     Read internal code record
1057    1    0    1000.4     Read binary record
 124    1    1    0          Convert character count
 124    1    1    0          To binary character count
 113    1    0    1000       Store character count
```

2.a) Read the six-bit characters from input stream five and store them at the bottom of separate half-words beginning at location A5 until a binary record is encountered. All carriage control characters are to be ignored and the address of the last stored character is to be left in B1.

b) Store the first 12-bit binary character at the least significant end of the half-word A6 (assume that at least one such 12-bit character exists).

	121	1	0.4	modifier = 0.4
	1050	0	5	Select input 5
1)	1053	127	A2	Go to A2 if next character is binary
	1054	60	-1*	Read next ch. to B60 Check next record if carriage cont. char.
	113	60	1 -0.4A5	Store character
	200	127	1 A1	Go to A1 and add 0.4 to b1
2)	124	1	0 -1A5	b1' = last address
	1054	60	0 0	Read m.s. half of binary character (n is not used)
	125	60	0 0	Shift b60 up 6 binary places
	113	60	0 A6	Store m.s. half
	1054	60	0 0	Read l.s. half
	114	60	0 A6	Store l.s. half
	- - - -	Next Instruction		

3. Read the remainder of the current record of input 2 into locations beginning at 105.4 and place after it, beginning at the next available half word, the same number of characters from the succeeding record. Assume that the succeeding record contains at least as many characters as the current one.

1050	0	0	2
1057	20	0	105.4
1056	20	20	105.7

8.15 The Output Extracodes

The necessary extracodes for selecting output streams have been described in section 8.4. As with input orders each output extracode listed below is singly modified and each refers to the currently selected output.

1062 Find Output Device Type

ba' = V-store address of the peripheral equipment used for the currently selected output.
If this output is to any peripheral (see Job Descriptions, Chapter 10)
then ba' = 0
The V-store addresses are given in appendix C.

1064 Write Character n

This extracode writes the character occupying the six least significant bits of the address to the currently selected output. If the internal code mode has been selected one internal code character will be written. If output is in binary mode, the extracode must be used twice to write the m.s. and l.s. halves of each 12-bit character respectively.

1065 End this Output Record

This writes the carriage control character occupying the six least significant address bits to the currently selected output, and terminates the record. In binary output it is usual to write a zero carriage control character, but in fact the carriage control character is neglected at time of printing or punching and any character would do.

1066 Write ba characters from S

Before entry to this extracode ba must be set as follows:

in bits 1 to 23:-	a character count as with 1056
in bit 0:-	0 if the record is to be ended
	1 if the record is not to be ended

If the record is to be ended the last character is taken as a carriage control character.
The extracode will when write the ba characters beginning at store address S to the currently selected output. The characters must be packed, four six-bit characters to the half-word. The least significant two bits of S are ignored (i.e. S must be a half-word or full-word address)

1067 Write a Record of ba characters from S

The effect of this extracode is exactly the same as using 1066 with bit 0 of ba equal to zero.
Before entry Ba must contain the character count in bits 1 to 23. Bit 0 of ba will be ignored as will bits 22 to 23 of S.

The extracode will write a record of ba 6-bit characters from store locations beginning at S.
The characters must be packed four to the half-word and the last will be taken as a carriage control character.

Extracodes 1066 and 1067 run very much faster if no characters have previously been sent to the record or if the number of characters previously sent is a multiple of four. Both these extracodes use far fewer instructions per character than 1064 and are to be preferred for large amounts of output.

Examples:

1. Read an internal code record from input one and send it to output three.

1050	0	0	1	Select input 1
1060	0	0	3	Select output 3
1057	1	0	100	Read record to locations 100 onward b1' = count of characters
1067	1	0	100	Output the record

2. Write the character stored in B2 to output two and follow it by the six characters in store locations 10.4 to 11.1. End the record with the carriage control character in 11.2.

1060	0	0	2	Select output 2
1064	0	2	0	Output ch. in B2
121	21	0	0.7	Count of characters in B21
1066	21	0	10.4	Output record

3. Output the following items on stream three, which is a seven-hole tape punch.

a) The characters BI.
b) Three new lines.
c) The 39 twelve bit characters which are stored in packed form, from location A12 onwards.
d) The binary character 0011.010

Then end the binary record.

a)	1060	0	0	3	Select output 3 for internal characters
	1064	0	0	4.2	Output B
	1064	0	0	5.1	Output I
b)	1065	0	0	2.3	End record with 3N.L's
c)	1060	0	0	3.1	Select 3 for binary
	121	60	0	39D2	Set count for 78 six-bit characters
	1066	60	0	A12	Output binary ch's
d)	1064	0	0	0	Output m.s. half

1064	0	0	K3.2	Output l.s half
1065	0	0	0	End binary record

The following extracodes do not apply to the currently selected output.

1063 Delete Output n

This deletes any information previously sent to Output n, and prevents it being printed, provided it has not been printed already by use of a 1071 extracode (see below).

1070 Rename Output n as Input Ba

This enables information sent to Output n to be read back by the same program as input. For example

1070 3 0 2

will rename output 2 as input 3.

1071 Break Output n

Normally, as described in chapter 10, all output is stored until the running of the program is completed. Then each output stream is put out separately preceded by the heading

OUTPUT n

and the title of the job.
This extracode indicates that the information so far recorded on Output n may, if convenient to the Supervisor, be treated as separate from any subsequent information sent to that output. The Supervisor will then arrange to send all information before the Break to the peripheral and output it with a heading and job title. Subsequent output will be stored in the usual way and output after the job is complete.

1072 Define Output n

Normally output documents should be defined in the job description in the manner given in Chapter 10. They may alternatively be defined by using this extracode. Before obeying this instruction, ba must be set equal to the maximum number of blocks of 4,096 six-bit characters to be allowed on Output n and ba* must define the output device to be used in the code described in appendix C.

Example:
To define a card punch output with number six to which a maximum of 2 blocks will be sent, the following instructions are required:-

121	25	0	2	Set B25 for 2 blocks
121	26	0	J600422	Set B26 for cards
1072	25	0	6	Define output 6

8.16 Further Information on binary input/output

When using Atlas Internal Code an 80-column punched card is represented by 80 six-bit characters and a next-card carriage control character, i.e. K2.1. Thus, to punch all 80 columns using extracodes 1066 or 1067 it is necessary to specify 81 characters in Ba by an instruction of the type

```
121    Ba    0    81D3
```

Similarly to read a card by extracode 1056, ba should specify at least 81 characters.

In binary a punched card is represented by 80 twelve-bit characters, one for each column, and one six-bit character with the value 0.0.

If more than 80 (internal code or binary) characters are output to a card, the first 80 of them will be punched on one card and the remainder on the next card starting again with column one. A continuous stream of characters output to cards with no carriage control information would accordingly be printed 'punched tape fashion' on successive cards.

The punching *** is not recognized on cards. Instead a card is inserted whose first column is punched 7, 8 and whose last column is punched Z, T, B, etc. The intervening 78 columns can contain anything whatsoever.

When a binary tape (but not a deck of cards) is read to its physical end (B or F) the final half word of the stored input is overwritten with J07070707. This means that the last one or two tape characters are stored as 000111000111. The zero carriage control character is unaffected.

On tape following ***E the warning characters ***C or ***Z are themselves stored in binary. They are not subsequently overwritten and each is followed immediately by the zero carriage control character. On cards following 7,8E the terminating card bearing 7,8C or Z is also stored. On tape and cards the new document which follows a C marker is automatically read in Atlas Internal Code; furthermore, if on seven-track tape, the document will be parity checked, even if ***P headed the previous document.

Chapter 9

MAGNETIC TAPE

9.1 Introduction

Magnetic tape provides an auxiliary store of very large capacity. For many purposes, a magnetic tape can be regarded as a larger but slower form of main store, but it is subject to the restriction that it must be scanned sequentially. It can perhaps best be likened to a notebook whose pages must be turned slowly one at a time: it is possible to ignore a page but it is still necessary to turn it over, and this takes as long as reading it. When using magnetic tape, it is therefore necessary to ensure that the information on the tape is arranged in the order in which it will be required.

Atlas uses two types of magnetic tape, of one and of half inch widths. The system tapes, and most tapes for private use, are one inch wide, pre-addressed tapes, which may be used for fixed or variable length transfers. Reading from the tape is possible in both forward and backward directions. The half inch tape is not preaddressed, and can only be read forwards: transfers are all variable length. One inch tapes prepared on the I.C.T. Orion computer may also be read.

To make efficient use of magnetic tape, it is necessary to overlap magnetic tape transfers and computing as far as possible. This requires care in the timing of transfers and the allocation of storage space when direct transfers to tape are employed. The programmer is, however, relieved of this responsibility when using the extracodes for variable length tape transfers, because these interpose a buffer store between the program and the tape.

Within a program, each magnetic tape is identified by a number. This number, B, is normally written in the Ba digits of an instruction and lies in the range $0 \leq B \leq 99$. The tape number, B, is normally allocated to the appropriate tape by the Job Description, which will be described in Chapter 10.

9.2 Atlas One Inch Tape

Information on each magnetic tape is split up into sections of 512 words. There are 5000 sections on each full-length magnetic tape, and these are numbered from 0 at the beginning of the tape to 4999 at the end. Section 0 is reserved for special purposes, and when a tape is first mounted it is positioned ready to move forwards and use section 1. Normally a program will first use the tape starting at section 1. Later it may require to return to section 1 or go on to some other section, and it must then obey a search instruction. The instruction

```
1001    Ba    0    n
```

will search for the beginning of section n on tape number Ba prior to fixed length transfers. Thus, to search for section 8 on tape 4, we would write

```
1001    4     0    8
```

The 1044 extracode must be used for a search before variable length transfers (see below).

Searching tape is a relatively slow process compared with the computing speed of Atlas, and the time taken is proportional to the number of sections traversed. Therefore the information on tape should normally be stored in consecutive sections starting at section 1, and any search instructions should be given as early as possible in the program.

In this chapter it will sometimes be necessary to refer to "blocks" of store. On Atlas, a block is a unit comprising 512 words of main store; block number P contains the 512 words whose addresses are 512P to 512P + 511. The store structure will be explained in chapter 12 but this simple definition should suffice for the present.

All magnetic tape instructions are singly modified, and throughout this chapter references to the address of an instruction apply to the modified address N + bm. The tape number is normally written in the Ba digits, but if Ba = 122 the tape number is specified by b121.

9.3 Block Transfers on One Inch Tape

Block transfer instructions allow a program to transfer 512-word blocks of information between a magnetic tape and a specified block of store. To obtain maximum efficiency in using magnetic tape, a program should use these block transfer instructions and make its own provision for the overlap of tape transfers and computing.

9.3.1 Block-Transfer Instructions

The section search instruction, 1001, described in section 9.2, may be used to position the tape before block transfer operations. An instruction of the form

```
1002    Ba    0    P:
```

would then read the next 512-word section from tape Ba into block P.

In the block transfer instructions, the octal fraction of the address is used as a parameter K, $0 \leq K \leq 7$, where $K + 1$ specifies the number of blocks involved in the transfer. Thus, to read the next two sections from tape 4 into blocks 5 and 6, we would write

```
1002    4     0    5:0.1
```

The block transfer instructions are as follows:-

1001	Search for the beginning of section n on tape number Ba.
1002	Read the next $K + 1$ sections from tape Ba into store blocks P, $P + 1$, ..., $P + K$.
1003	Read the previous $K + 1$ sections from tape Ba into store blocks $P + K$, ..., $P + 1$, P.
1004	Write store blocks P, $P + 1$, ..., $P + K$ on to the next $K + 1$ sections of tape Ba.
1005	Move tape Ba forwards $K + 1$ sections.
1006	Move tape Ba backwards $K + 1$ sections.

When reading either forward or backward, information will be held in store in the same order as on tape, with the first word in the lowest numbered tape section transferred to the start of the store block with the smallest address. This order of words is also maintained when writing to tape.

Examples:

1. Read section 19 of tape 3 to main store block 6.

```
(a)     1001    3    0    19      Search for section 19
        1002    3    0    6:      Read forward to block 6
(b)     1001    3    0    20      Search for section 20
        1003    3    0    6:      Read backward to block 6
```

The information in block 6 will be the same in either case.

2. Read sections 1, 3, 5, 7 and 9 of tape 66 into main store blocks 20 to 24 inclusive. The previous operation on tape 66 was to write to section 13.

```
        1006    66    0    0.2    Position tape after section 11
        121     1     0    4:     Set block modifier
1)      1006    66    0    0      Move tape back 1 section
        1003    66    1    20:    Read previous section
        123     1     0    1:     Reduce block modifier
        216     127   1    A1     Return if non-negative
```

A considerable saving is obtained in this example by reading backwards. To have searched for section 1 and read forwards would have meant traversing nine sections twice and would have taken almost twice as long.

The instructions have been arranged so that the 1006 instruction comes before the 1003 instruction in the loop. If the 1003 instruction had been put first, the program would have traversed one extra section after the last read instruction; in this particular program the extra section would have been section 0 and the program would have been monitored because the use of section 0 is prohibited.

9.3.2 Use of Block Transfers.

The way in which a program uses magnetic tape will depend very much on the requirements of the process it is performing. Sometimes it is necessary to read a large amount of information, such as a complete matrix, before computing can commence. In this case, shortage of store may prevent the overlap of computing with further tape reading, but at least the next required tape address can be searched for; afterwards it may be possible to overlap the writing of the results to one tape with the reading of the next set of data from another. The technique of branching, to be described in chapter 12, may also help in this situation.

When it is possible to work sequentially through the information on tape, operating on one word or one small group of words at a time, considerable savings can be made by overlapping tape transfers with computing. This is done automatically by the variable length transfers (see below). With block transfers, overlap can be obtained by transferring alternately to two different blocks, computing on one whilst transferring to the other.

The same process can be used when operating on two or more magnetic tapes. When processing longer items, special care is needed if an item overlaps two tape sections.

Example:

To read sections 1 to 2000 of tape 4, presenting each word to a processing routine R3, a control program of the following form would suffice:

	R0	ASSUMED			
	1001	4	0	1	Search for section 1
	1002	4	0	5:0.1	Read to blocks 5 and 6
	121	71	0	511	Set word count
	121	72	0	1999	Set section count
	121	127	0	A4	Jump to label 4
2)	124	2	0	1	Step up address
	203	127	71	A1/3	Count words in block
	121	71	0	511	Reset word count
	203	127	72	*2	Count tape section
	121	127	0	A5	Exit
	1002	4	2	-1:	Read to refill block just emptied
	203	127	73	A1/3	Count blocks
4)	121	73	0	1	Set block count
	121	2	0	5:	Set first block address
	R3				
1)	324	0	2	0	Read word
	:	:	:	:	Process word
	121	127	0	A2/0	Return to R0 for next word

In the preceding example, because it is reading two sections in advance, the program reads one section more than it requires, but does not attempt to process the extra section. This extra read operation could be avoided by an extra test in the example, but it would be unavoidable if the end of the process were detected by the processing routine on receipt of the last word. There is normally no harm in reading extra sections provided that they do not lie outside the range 1 to 4999 inclusive, and provided that these extra sections have been written to since the tape was last addressed. It is therefore advisable to write a few extra sections after the information when a tape is written to. A magnetic tape fault (512-word fault) will show up if an attempt is made to read from a section not previously written to.

9.4 Variable Length Working on One Inch Tape

To simplify the writing of some magnetic tape programs, extracodes are provided which execute the transfer of variable length records between magnetic tape and the main store. Variable length operations must initially be preceded by the variable length word search. The instruction

```
1044    Ba   0    S
```

will search tape number Ba for the section and word contained in the full word with address S. The section number is contained in the more significant, and the word number in the less significant half word, both being held as 21 bit integers. Thus, to search for the eighth word of section 10 on tape 4, we would write

```
1) H10   8
   1044  4    0    A1
```

The 1001 search may not be used with variable length working.

The extracodes for variable length operations require an area of store to be used as a "buffer", to hold information in transit between the tape and the store. This buffer must be set up, and the mode of operation specified, by obeying a "start" extracode for each tape involved in variable length operations. Thereafter, a "transfer" extracode is used to transfer information between the buffer and the program as required. When writing to tape each such transfer forms one "record" on the tape.

Before writing variable length records to magnetic tape, it is necessary to obey a 1032 instruction. This "start writing" instruction normally takes the form:

```
1032    Ba    0    P:0.K
```

This prepares for writing forwards starting at the next word on tape Ba, and selects it for variable length operations. It also sets up a buffer store in blocks P to P + K inclusive; normally K = 1, allowing a two block buffer. Thus, to start writing variable length records to tape 5, using main store blocks 10 and 11 (locations 5120 to 6143) as buffer, we would write

```
1032    5     0    10:0.1
```

Thereafter, information may be transferred to tape 5 by 1040 instructions. Before obeying a 1040 instruction, when writing to tape, the number of words to be transferred and the end-of-record marker must be set in an index register: the number in the integral part and the marker in the octal fraction. This index register must then be specified in the Ba digits of the 1040 instruction. Normally, the end of an ordinary record should have a marker of value 1. Thus, using B6 to specify a transfer of 25 words, we would write

```
121     6     0    25.1
```

To transfer 25 words (as specified in B6) starting at address 2000, we would then write

```
1040    6     0    2000
```

Example:

Given a 30 x 100 matrix stored by rows in location 8000 onwards. Write the 30 rows, of 100 numbers each, as 30 separate records starting at the beginning of section 8 on tape 4.

```
1)H   8       0
      1044    4     0    A1       Search for section 8
      1032    4     0    10:0.1   Start Writing to tape 4
      121     1     0    29       Set row count
      121     2     0    0        Clear modifier
      121     3     0    100.1    Prepare to transfer 100 words
5)    1040    3     2    8000     Transfer
      124     2     0    100      Increase modifier
      203     127   1    A5       Count rows
```

Before reading variable length records from tape, it is necessary to obey a "start reading" instruction. To start reading forwards, a 1030 instruction must be used, and this normally takes the form

```
1030    Ba    0    P:0.K
```

This starts reading forwards from the next word on tape Ba and selects it for succeeding variable length operations. It also sets up a buffer in blocks P to P + K inclusive; normally K = 1, giving a two block buffer. Thus, to start reading variable length records from tape 5, using main store blocks 10 and 11 as buffer, we would write

```
1030    5     0    10:0.1
```

Thereafter, information may be transferred from tape 5, reading forwards, by using 1040 instructions. The Ba digits of the 1040 instruction indicate which index register has been used to specify the amount of information to be transferred. When reading from tape, this index register normally specifies the maximum number of words to be transferred. It may also specify an end-of-record marker whose purpose will be explained later. The number is specified by the integral part, and the marker, if required, by the octal fraction.

To read one record at a time, the maximum length of record should be specified and the marker should be zero (or one). After the transfer, the same index register records in its integral part the number of words actually read, and in its octal fraction the value of the marker at the end of the record. Thus, to read a record of not more than 200 words to location 1500 onwards, we would write

```
121     10    0    200
1040    10    0    1500
```

If the actual record were of 100 words terminated by a marker 1, then B10 would contain 100.1 after the transfer.

Thus, to read to location 200 the first row of the matrix recorded on section 8 in our previous example, we would write:

```
8)H   8       0
      1044    4    0    A8       Search for section 8
      1030    4    0    10:0.1   Start reading from tape 4
      121     3    0    100      Prepare to read up to 100 words
                                  of next record
      1040    3    0    200      Transfer
```

When reading, it is possible to ignore end-of-record markers of less than a given value by specifying that value in the octal fraction of Ba before the transfer. Thus, for example, by setting 300.2 in B3 before the 1040 instruction above, we could read the first 300 elements of the matrix; the marker 1 written at the end of each row is less than the octal fraction 2 set in B3 and would therefore be ignored.

The instructions to start reading or writing assume the tape to be positioned at a marker, either by previous variable length operations or word search.

So far we have only considered working on one magnetic tape at a time, and in this case the start instruction selects that tape for all succeeding tape operations. When working on two or more tapes, it is still necessary to use start instructions to initiate variable length working, but subsequently 1033 instructions must be obeyed to select whichever tape is required. This "select" instruction chooses tape number Ba for succeeding variable length transfers until the next select or start instruction.

Example:

Tape number 2 contains a file of variable-length records starting at section 1. Each record is terminated by a marker 1, except the last record which is terminated by a marker 2. The maximum length of record is 50 words. Copy the file to tape 3, section 1 onwards.

```
2)H   1       0
      1044    2    0    A2       Search for section 1, tape 2
      1044    3    0    A2       Search for section 1, tape 3
      1030    2    0    10:0.1   Start reading, tape 2
      1032    3    0    12:0.1   Start writing, tape 3
1)    1033    2    0    0        Select tape 2
      121     4    0    50  )
                                  Read next record to location
      1040    4    0    A6  )    A6 onwards
      1033    3    0    0        Select tape 3
      1040    4    0    A6       Write record
      210     127  4    A1       Jump if marker odd, End if
                                  marker even.
```

9.4.1 Variable Length Instructions

In the previous section, a selection of the most important variable length magnetic tape instructions have been described and illustrated in order to explain how they are used. In this section, the full range of variable length tape instructions will be defined, and there will be some repetition of information from the previous section.

When using variable length tape transfers, the information is stored on tape in groups of words known as "records", with a 21-bit count and a 3-bit marker on each side to denote the ends of the record. Thus the space on tape occupied by a record is one word more than the number of words of information. Each writing transfer forms one record on tape. A reading transfer may either read a specified number of words or read up to the end of a record; in both cases markers are omitted from the transfer. Instructions to start reading or writing must only be given when the tape is positioned at a marker.

A number of consecutive records often form a larger unit, such as a complete matrix or a complete file, and it is often desirable to mark this in some way. For this reason eight orders of marker, numbered 0 to 7, are provided. Ordinary records may be terminated by a marker of order 1, groups of records by a marker of order 2, and so on up to 7, which normally denotes the beginning or end of a file. Reading transfers may then read up to a marker of a specified order, and the program may test the value of the marker read. Use of the 0 marker is not recommended.

Variable length working must always be initiated by a start instruction. This sets up a buffer store, selects the tape to be operated upon, and specifies the mode of operations, whether write, read forwards or read backwards. A separate start instruction must be given for each tape on which variable length transfers are required, but thereafter a select instruction may be used to choose the tape to be used. The transfer instruction operates on the tape which was last selected by a start or select instruction, and transfers information in the mode selected for that tape. To change the mode it is necessary to obey another start instruction. Unless the tape is already in variable length mode, the instruction to start writing begins by writing a marker of order 7. To start writing without commencing with a 7 marker, the sequence Word Search, Start Reading Forwards, Start Writing, Transfer should be used. Alternatively, the 1042 'mark' instruction may be used after the instruction to start writing.

A start instruction always initiates variable length transfers to or from the next word on tape or the previous word in the case of reading backwards. To begin working at a particular address on tape, the start instruction must be preceded by the word search instruction, 1044; when starting to read variable length records, this starting address must be the address of a marker at one end of a record.

The variable length writing operations do not provide a means of overwriting selected words on a magnetic tape: complete new blocks are formed in the buffer and the previous contents of the tape are not preserved. If it is required to preserve the beginning of the first section, this may be done by preceding the start writing instruction by a start reading forwards instruction with $K = 0$.

It is important to note that, when using a buffer of K+1 blocks, the attempt to read variable length records backwards from any of the first K sections, or forwards from any of the last K sections, will lead to an end-of-tape interrupt. K sections should therefore be left unused at both ends of the tape.

9.4.2 Start and Select Instructions

In the following specifications, P is a block address, and K an octal fraction, as above.

1030 Start Reading Forwards

Start reading forwards from the next word on tape Ba and select it for variable length operations. Set up a buffer in blocks P, P+1,..., P+K.
If this instruction is given when the tape is already in read forwards mode, then the tape will be wrongly positioned.

1031 Start Reading Backwards

Start reading backwards from the previous word on tape Ba and select it for variable length operations. Set up a buffer in blocks, P, P + 1, ..., P + K.

1032 Start Writing Forwards

Prepare to write forwards starting at the next word on tape Ba, and select it for variable length operations. Set up a buffer in blocks P, P + 1, ..., P + K.
Also, write a marker 7 before the first word of information, provided that the given tape is not already in use for variable length working. If the tape is already in use for variable length working, the marker written will be equal to the marker at the end of the previous record.

The three 'start' instructions 1030, 1031 and 1032 must be preceded by a 1044 word search, even when the desired word is the first in the section. To give predictable results, the tape must be positioned at a marker whenever a 'start' instruction is used. The K+1 block buffer must have been allowed for in the job description (see Chapter 10). If one of these blocks is already in use by the program, the information in it will be lost. Strings of information with markers at either end are transferred to and from these blocks by the 1040 instruction. When a buffer block has been filled in the writing mode, it is transferred to tape; when completely read in the reading mode, the next section is read from tape. The buffer blocks are not protected from the program, but should not be referred to directly.

It may be desirable to read variable length records from a tape for a while, and then begin writing to the very next record. 1032 will switch modes in this way; the first record written will begin with the same marker as that terminating the previous record.

1033 Select

Select tape Ba for succeeding variable length operations, in the mode specified by the preceding start instruction for that tape. All succeeding 1040, 1041 and 1042 instructions apply

to tape Ba until another tape is selected. If a write buffer had previously been set up for tape Ba, its contents will be written to tape, and a new buffer set up. Records will be written as if tape Ba were selected throughout.

Extracodes 1030 and 1031 above assume that the tape to be read from has been previously written by variable length operations. If this is not the case, the extracodes 1034 and 1035 should be used instead.

1034 Start Reading Forwards From Fixed Blocks

As 1030, but operating on a tape which has not been written in the form of variable length records.

1035 Start Reading Backwards From Fixed Blocks

As 1031, but operating on a tape which has not been written in the form of variable length records.

9.4.3 Transfer and Organizational Instructions

Notation: b_w = Integral part of ba (bits 1 to 20) $(0 \leq b_w < 2^{19})$

b_k = Octal fraction of ba. $(0 \leq b_k \leq 7)$

1040 Transfer

Transfer up to b_w words between store addresses starting at S and the selected tape, in the mode (reading forwards, reading backwards, or writing) appropriate to that tape. On writing, b_w words from locations S, S + 1, ..., S + b_w - 1 are written to the next b_w locations on the selected tape. A marker b_k is written on tape after them. On Reading, provided that $b_w \neq 0$, the transfer continues until b_w words of information have been read or until a marker $\geq b_k$ is encountered, whichever is the sooner.

b_w' = the number of words of information actually read.

b_k' = 0 if no marker $\geq b_k$ was encountered.

= m if a marker m ($\geq b_k$) terminated the transfer or immediately followed the last word transferred.

When reading forwards, the next b_w' words are read from tape to store locations

S, S + 1, ..., S + b_w' - 1

When reading backwards, the previous b_w' words are read from tape to store locations

S, S - 1, ..., S - b_w' + 1

If $b_w = 0$ when reading forwards, the transfer continues until the first marker $\geq b_k$ is encountered. When reading backwards

with b_k and b_w zero, the transfer continues until the end of the first record, and the b_w' words of the record are read to store locations

$$S,\ S-1,\ \ldots\ldots,\ S-b_w'+1$$

It is not advisable to write with the octal fraction $b_k = 0$, because when the resulting string, which therefore ends with a zero marker, is read back, the octal fraction b_k will be zero regardless of whether reading ended at the zero marker or somewhere short of the marker within the string itself.

1041 Skip

Skip b_w words, terminating on a marker b_k.

Skip operates in the same way as transfer, except that no words are transferred to or from the program store.
When in a writing mode, b_w addresses on tape are skipped and a marker b_k is written after them. Note, however, that the previous contents of these addresses, whether information or marker, are not preserved on tape, except when complete 512-word tape sections are skipped.
When in a reading mode, the skip continues until b_w words of information have been passed or until a marker $\geq b_k$ is encountered, whichever is the sooner.
b_w' = the number of words of information actually skipped.
b_k' = m if a marker m ($\geq b_k$) terminated the transfer or came immediately after word b.

Note that skip is less efficient than search for moving long distances along the tape, and should not be used for skipping more than a few sections.

1042 Mark

Available only when in writing mode.
Writes a marker K ($0 \leq K \leq 7$) after the last word on the selected tape. This marker replaces any marker which was previously on the tape at this point.
After writing a string on tape, it may be discovered that the end of a group has been reached. The mark instruction may then be used to change the marker at the end of the string. It may be used again if it is later found that the end of an even higher order group has been reached.
A mark instruction may also be used immediately after a start-writing instruction, to write the specified marker before the first record to be written.

1043 Stop Variable Length

Stop variable length operations on tape Ba.
After variable length operations for a given tape have been completed, a stop instruction may be given; it will release the buffer blocks associated with those operations. After

writing operations, it will cause the last part-section to be written from the buffer to magnetic tape. The following instructions also have the effect of stopping previous variable length operations:

start, search, unload, release tape, end program.

Whenever variable length writing is terminated on a given tape each buffer block that contains information transferred there by a 1040 instruction is written to tape. A buffer block is not written to tape unless it contains such information. This means that the first few words of the last buffer block written may contain the end of the final record, or even just the final marker, and the rest of the block contain perhaps the remains of some previously transferred records. This fact permits one to overwrite individual records without disturbing the records on either side. This is done by filling the buffer with the record that is to be overwritten along with the records, or part of the records, on either side of it. One aligns on the marker at the beginning of the record and starts reading forward (1030) with a buffer large enough to contain the whole record at one time. One then skips (1041) the record in question, starts reading backward (1031, with the same buffer) and skips back over the record to the beginning marker. One switches to write mode (1032, same buffer again) and writes the new string in its proper place. A 'stop' instruction (1043) will then finish the job by causing the buffer blocks, now containing the new string in place of the old, to be written back to the tape. The two uses of the skip instruction are necessary to make the marker at the beginning of the old string available for the construction of the marker at the beginning of the new string.

1044 Word Search

Search tape Ba for the section and word specified in the full word with address S. Stop variable length operations on tape Ba. The section and word are given as 21 bit integers in the more and less significant halfwords in full word s respectively. The section number must be greater than 0, and less than 5000; the word number is taken modulo 512. This instruction must be used to align the tape on a marker before using a 'start' instruction; after the word search, a 'start' instruction is necessary before further variable length operations on tape Ba.

1036 Set ba' = number of selected tape

Place in bits 10-16 of Ba the program number of the tape currently selected for variable length operation. If there is no tape currently selected, ba' will be negative. Bits 10-16 correspond to the Ba position in the more significant halfword of an instruction. One application is to enable a sub-routine to select a different tape for variable length transfers, and then to re-select the original tape before the main program is re-entered.

1037 Store 'mode of tape Ba' in S

The transfer mode at present selected for tape Ba will be in-

dictated by placing in half-word S the appropriate integer from the following table:-

0 Variable length transfers, reading forwards from variable length records

1 Variable length transfers, reading backwards from variable length records

2 Variable length transfers, writing variable length records.

3 Not currently selected for variable length transfers.

4 Variable length transfers, reading forwards from fixed length blocks.

5 Variable length transfers, reading backwards from fixed length blocks.

Examples:

1. Tape 1 contains a file of variable length records and tape 2 contains amendments to this file. The information on each tape starts at section 1 and the records are not more than 40 words long. Each record is terminated by a marker 1, except the last record which is terminated by a marker 2. Each record is identified by a key in its first half-word, and the records in each file are sorted in ascending order of keys. It is required to form an updated file on tape 3 by inserting the amendment records in place of the corresponding records on the original file.

```
1)H     1       0
        1044    1       0       A1      )
        1030    1       0       20:0.1  )
        1044    2       0       A1      )   Initiate variable length
        1030    2       0       22:0.1  )   operations on tapes,
        1044    3       0       A1      )   1, 2 and 3
        1032    3       0       24:0.1  )

4)      1033    2       0       0       )
        121     12      0       50      )   Read Amendment Record
        1040    12      0       100     )

5)      1033    1       0       0       )
        121     11      0       50      )   Read Main-File Record
        1040    11      0       150     )

        1033    3       0       0           Select Updated File

        101     10      0       150     )   Go to A7 if Amendment Key
        102     10      0       100     )   equals Main-File Key
        214     127     10      A7      )
```

(Program continues on next page)

	1040	11	0	150	Write Main-File Record
	210	127	11	A5	Go to read next record if marker is odd
	1117	0	0	0	End Program
7)	1040	12	0	100	Write Amendment
	210	127	12	A4	Go to read next amendment if amendment marker is odd
	1042	0	0	0.1	Write a marker one
	210	127	11	A5	Go to read next Main-File record if marker is odd
	1042	0	0	0.2	Mark end of updated file
	1117	0	0	0	End program

2. There is a 50 word record in section 10 of tape 33 which is preceded by a '7' marker in word 177. Replace that record with the 50 consecutive words beginning in the store at A3, leaving the '7' marker undisturbed.

3)	H10,177				Section 10, word 177
	1044	33	0	A3	Search for marker
	1030	33	0	1:	Fill the one block buffer
	1041	0	0	0	Skip to the end of next record
	1031	33	0	1:	Read backward
	1041	0	0	0	Skip back to head of record
	1032	33	0	1:	Switch to write mode
	121	75	0	50	
	1040	75	0	A3	Replace record in buffer
	1043	33	0	0	Write block 1: to section 10

Although the length of the string was known in advance and B0 mentioned in both of the skip instructions, one could use one of the skip instructions to pick up the length of the string in a B-line.

If the 1032 write instruction is replaced by a 1042 mark instruction, the program will replace only the mark at the head of the string.

9.4.4 Efficiency of Variable Length Working

The basic magnetic tape operations on Atlas transfer information in blocks of 512 words, but the variable length instructions disguise this fixed block structure. Provided that the length of transfers is small compared with the size of the buffer, these instructions also provide overlap of tape transfers and computing. To achieve these effects, extra words are written on the tape as markers and extra instructions are obeyed to transfer the information between the buffer and the program store. Provided that the variable length records are not too short, the loss of efficiency is not high. With a two-block buffer and records of 50 to 150 words, the use of variable length instructions might be expected to increase the cost of a job by perhaps 1% or 2%. This efficiency will be maintained with longer transfers also, but the automatic overlap of transfers with computing will be lost as the transfer size approaches the buffer size.

9.5 Organizational Instructions for One Inch Tape

A number of organizational instructions are provided to cope with special situations which will arise in some magnetic tape programs. Many of the operations which these instructions perform are usually required near the beginning or end of the program, and they are then best left to the job description or the 1117 (end program) instruction. One exception to this rule is the instruction 1017 (Free Tape), which should be used before the 1117 instruction if the information on any titled tape is not required again.

Programs frequently require to use magnetic tapes as working space, without wanting to keep them after the job has been run. Such magnetic tapes should be requested under a heading TAPE COMMON in the job description.

For a long job, it may be desired to restart the job after a machine failure severe enough for common tapes to be lost; in such a case tapes should be requested under TAPE NEW. In exceptional circumstances a title may be written to a common tape, which is then kept by the user after the job has ended; the operator will be notified, and the Atlas installation may take action to discourage a user from doing this at all often.

The title stored in section 0 of the tape is referred to by extracodes 1014 and 1015 (Write Title & Search for Block 1, and Read Title or Tape Number). These refer by its main store address to a copy of the tape title, stored as one record; the 6-bit characters are packed 8 to a word and the last character is binary zero.

9.5.1 Mount Instructions

If a program requires to use magnetic tape, its job description must indicate the number of magnetic tape mechanisms required. Normally this is done by listing the magnetic tapes which are required to be mounted on these mechanisms initially: the Supervisor will then ensure that these tapes are mounted before the program is entered. The details of how the job description is prepared are given in Chapter 10.

If further magnetic tapes are required after a program has been entered, they should be listed in the job description and may then be called in by obeying "mount" instructions. However, the total number of mechanisms in use at any one time must not exceed the number reserved in the job description. A mount instruction should, if possible, be obeyed at least 2 minutes before the tape to which it refers is required; otherwise the tape may not be ready in time and the program will have to wait. Note that the program will be monitored if it calls for a new tape to be mounted at a time when none of its reserved mechanisms has been made free. If there is a spare tape mechanism, the tape may be mounted on it by the operator before the program calls for it.

The tape reference letter referred to in the mount extracodes is a letter associated with the tape in the job description (see chapter 10).

The mount instructions are as follows:-

1010 Mount

Allocate the number Ba to the tape whose tape reference letter is in the 6 least significant bits of n, in internal code. If this tape is not already available, instruct the operator to mount it on any available tape mechanism. If the tape is in the TAPE category, check its title; if in the TAPE NEW category, write into the section 0 the title given in the job description: if in the TAPE COMMON category, leave the tape untitled.

1011 Mount Free

Allocate the number Ba to a free tape, If no free tape is available, instruct the operator to mount one on any available tape mechanism.

This extracode should only be used in exceptional circumstances; normally all tapes required should be listed in the job description. The operator will be informed whenever this extracode is used, and an installation may take action if it is used overmuch.

1007 Mount Next Reel of File

Allocate the number n to the next reel of file Ba. If this tape is not mounted, instruct the operator to mount it on any available mechanism.

The following mount instructions refer to logical channel number (K = 0 to 3) of a given program. These logical channel numbers do not each refer to a fixed magnetic tape channel: they are merely a device to enable the program to separate on to different channels the magnetic tapes that it requires to operate simultaneously. On an installation with only one tape mechanism on each of the eight tape channels, there is no advantage in specifying channel numbers. On larger installations, the tape mechanisms are grouped together on pairs of channels, and the logical channel numbers then refer to these pairs.

A program's referring to logical channel number K has the following effect. If no channel has been previously designated logical channel K of the program, the new tape is mounted, if possible, on a channel different to any which has been previously designated a logical channel of the program; that channel is then designated logical channel K of the given program. If a channel has been previously designated logical channel K of the program, then the new tape is mounted on the same channel, if possible. A channel may also be designated logical channel K of the program by the Job Description; see chapter 10.

1012 Mount on Channel K

Allocate the number Ba to the tape whose tape reference letter is in the 6 least significant bits of n, in internal code. If this tape is not already available, instruct the operator to mount it if possible, on channel K of this program, where K is the most significant octal digit of n. If the Tape is in the TAPE category, check its title; if in the TAPE NEW category, write into section 0 the title given in the job description; if in the TAPE COMMON category, leave the tape untitled.

1013 Mount Free on Channel K

Allocate the number Ba to a free tape. If no free tape is available, instruct the operator to mount one, if possible, on channel K of this program, where K is the most significant octal digit of n.

Note that this extracode, as 1011, should be used only in very exceptional circumstances; normally all tapes required should be listed in the job description. The operator will be informed whenever this extracode is used and an installation may take action if it is used overmuch.

9.5.2 Other organizational Extracodes

1014 Write Title & Search for Block 1

Write to section 0 of tape Ba the title stored in location S onwards, overwriting any title that may be there. Inform the operator that this has been done.

1015 Read Title or Tape Number

If S is even (least significant bit is 0), then store in locations S onwards the title of tape Ba, i.e. that currently in section 0.

If S is odd (least significant bit is 1), then store in location S the tape serial number of tape Ba.

In 1014 and 1015 S is taken as a half word address. The tape serial number stored by 1015 will be in internal code, left justified, 8 characters in length.

1016 Unload and Store

Rewind tape Ba and disengage the tape mechanism on which it is mounted. Instruct the operator to remove the tape, ensure that the correct title is written on the spool, and store it for later use. If $n \neq 0$, the number of tape mechanisms reserved for the program is reduced by one.

1017 Free Tape

Overwrite the title on section 0 of tape Ba and return the tape to the Supervisor as a Free tape for general use. If $n \neq 0$ the number of tape mechanisms reserved for the program is reduced by one.

1020 Release Tape

Delete tape Ba from the allocation of this program and make it available for another program, which must call for it by its correct title. The tape is not freed and is not normally disengaged. If $n \neq 0$, the number of tape mechanisms reserved for the program is reduced by one.

1021 Free Mechanisms

Reduce by n the number of tape mechanisms reserved for use by the program.

1022 Re-number

Allocate the number n to the tape which was previously referred to in this program as tape number ba.
This enables a tape to be given a different number during a subroutine, and then to have its original number restored at the end of the subroutine. Note that in this instruction the tape number is ba and not Ba.

1023 How Long?

$s' =$ Number of 512-word sections available on tape Ba (excluding section 0). The number of sections available on a standard-length tape is 4999, but this instruction may be of value if shorter tapes are used. The tape will be rewound.

1024 Where am I?

$s' = A$; $s^{*\prime} = W$ $(0 \leq W \leq 511)$.
This instruction places the address of the next section (going forwards) on tape Ba in the first half-word of the specified full-word address. After variable length transfers, the second half-word contains the position in the section of the next word to be used. After block transfers, the second half-word is zero.

9.6 Specification of the Atlas One Inch Tape System

9.6.1 Control

An Atlas installation may have as few as 8 one inch magnetic tape mechanisms, or as many as 32. Each mechanism is connected via one of eight channels which can operate simultaneously, each channel controlling one read, write, or search operation. Wind and rewind operations are autonomous and need the channel only to initiate and, if required, to terminate them.

The layout of the control system depends on the individual installation. When there are only 8 mechanisms, each mechanism has its own control channel. When there are more than 8 mechanisms, the 8 channels are grouped into 4 pairs and some or all of these pairs are adapted to control up to 8 mechanisms; any two of the mechanisms on one such pair of channels can then operate simultaneously at any one time.

9.6.2 The Tape Layout

The tape mechanism is the Ampex TM2, using one inch wide magnetic tape. There are 16 tracks across the tape, used as follows:

12 information tracks

2 clock tracks

1 block-marker track

1 reference-marker track (for Tape Addressing only)

Information is stored on tape in blocks or sections of 512 48-bit words, followed by a 24-bit checksum. Each section is preceded by a leading block marker and a section address, and terminated by a trailing block marker and a zero address. Tapes are tested and pre-addressed by a special run on the machine before they are put into use, and the fixed position of addresses permits selective overwriting of sections. Checksums are of 24 bits with end-around carry; they are used to check the accuracy of all reading and writing operations.

A 48-bit word is represented by four lateral stripes of 12 information bits, and a checksum by two stripes. Each 512-word section of information contains 2050 stripes and has an average length of 5.46 inches, with a gap of 2.3 inches between sections. Tapes are 3600 feet long and hold 5000 sections, or $2\frac{1}{2}$ million 48-bit words.

9.6.3 Performance

The normal tape speed is about 120 inches per second and there are 375 binary digits per inch on each track. This gives an instantaneous transfer rate of 90,000 6-bit characters per second, or one 48-bit Atlas word every 89 microseconds. Allowing for the gaps between sections, the effective transfer rate is about 64,000 characters per second. This is equivalent to one 512-word section every 64 milliseconds, or one word every 125 microseconds, on average. There are also fast wind and rewind operations at about 180 inches per second, and these are used for long searches along the tape.

There are independent write and read heads, separated by a gap of about .39 inches. When not operating, the tape stops with the read head roughly midway between sections, ready to read the next section address. It is possible to read when the tape is moving either in the forward or reverse direction, but writing is only possible when the tape is moving forwards.

9.6.4 Safeguards

A program is held up if it attempts to read from or write to a block of store which is involved in a magnetic tape transfer. The Supervisor may then enter another program until the transfer is completed.

If a magnetic tape block transfer cannot be initiated when it is requested, it is placed in a queue. If the queue is already full the program is held up.

A write-permit ring must be fitted to a reel of tape before that reel can be written on. Tapes containing permanent information will not have such a ring. A write-inhibit switch is also provided on each mechanism, which the operator can use to isolate the tape. It is only possible to write on a tape when the write-permit ring is on and the write current is switched on.

The address of the relevant section on tape is checked before all reading and writing operations, to make sure that the correct section is used. The information in each section is checked by means of a 24-bit checksum at the end of the section: this checksum is used to detect faulty writing or reading, which cause the operation to be repeated under the control of the Supervisor.

When a magnetic tape has useful information on it, a descriptive title of that information is stored in block 0 of the tape. This is in addition to the tape serial number, which is permanently associated with the tape and is unalterable by the user. This tape title can be up to 80 characters long, though the Supervisor prints out only the first 30 characters in operator messages. It is stored on tape in Atlas Internal Code with tab. and multiple spaces (including tab.) replaced by a single space; initial spaces, tabs, full stops, and commas are ignored. Shifts are ignored throughout the title. (see Chapter 10.)

9.7 Orion Tapes

It is possible to read into Atlas tapes prepared on the I.C.T. Orion Computer. This is accomplished by the use of the following two extracodes. P is a block address, and K an octal fraction, of the form P:0.K.

1046 Read Orion tape Ba forwards

A check is first made that tape Ba is in fact an Orion tape (if it is, zero will have been read from the first bit of the first block when the tape was mounted). Then, reading forwards, the next section is read into store blocks P onwards. K + 1 blocks will be reserved for the transfer - if this is not sufficient, the program will be monitored. Up to 4096 words may be transferred, but there is no automatic indication of the number of words actually read.

1048 Read Orion tape Ba backwards

This is very similar to 1046, the difference being that the first word transferred is placed in the last word of block P + K and the tape is read backwards.

Some of the organizational instructions listed in section 9.5 may be used with Orion tapes. These are given below.

1010 Mount

1007 Mount Next Reel of File

1012 Mount on Channel K

1015 Read Title or Tape Number

1016 Unload and Store

1020 Release Tape

1021 Free Mechanisms

1022 Renumber

1023 How Long?

1024 Where am I?

The section number only will be given; the second half word will be zero.

The title read by the 1015 instruction will consist of up to ten words separated by /, the words containing up to eight characters. The characters will be letters, digits or full stop.

In the Job description (chapter 10), Orion tapes must be listed under TAPE headings, and the title must be given in the form described for the 1015 extracode.

Chapter 10

PREPARING A COMPLETE PROGRAM

This chapter explains the way in which a program, and the input information it uses are prepared for running on Atlas.

10.1 Atlas Jobs

Each run of a program on Atlas is known as a job; it may range from a small job, for which there is no data outside the program itself, to a large job requiring several batches of data, possibly arriving on different media, e.g. punched cards and paper tape.

The various parts of a job may be submitted separately to the computer each on one deck of cards or length of paper tape, or two or more parts may be combined on a single deck or tape. In any case each part must be properly identified for the computer and for this purpose the concept of a 'document' has been introduced.

10.2 Documents

A document is a self-contained section of information presented continuously to the computer through one input channel (but see also 10.10.3). Typical examples of a document are a collection of data on a length of paper tape or the program itself.

Each document carries at its head suitable identifying information as detailed in 10.3.1. The end of a document is indicated by an 'end of document marker' which usually consists of the characters ***Z on a new line, or a 7, 8 punching in the first column of a new card which follows the last record of information of the document.

By means of the identifying information the Supervisor prepares a list of documents as they are accepted in the store and it also keeps 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 each document; when all have been entered work on the job may commence. Documents for a job may thus be fed to the computer in any order.

10.3 Document Headings and Titles

Every document is preceded by the identifying information mentioned above. This consists of a heading and the title of the document.

10.3.1 Headings

The heading indicates which type of document follows and must be one of a standard list. The most common types of heading are as follows:-

a) COMPILER (which is followed on the same line or card, after one or more spaces, by the name of a program language). The document following this heading is a program in the stated language. Available languages include Atlas Basic Language (ABL), Extended Mercury Autocode (EMA), ALGOL and HARTRAN (for Fortran).

For an Atlas Basic Language Program the heading will be

COMPILER ABL

b) DATA
The document following consists of data required by a program.

c) JOB
The following document is a request to the computer to execute a job and gives 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 and a list of output streams the program will produce. It is described in detail in sections 10.4, 10.5 and 10.6 below.

10.3.2 Titles

The title of a document consists of one line (or one punched card) immediately following the heading. It may be composed of any combination of characters obeying the rules of section 10.3.3 below. The prime consideration is, however, that it should be unique among all the documents stored in the computer at the same time. This is obviously made essential by the time-sharing facilities of Atlas, to avoid confusion between documents intended for different jobs.

A document will thus usually take one of three forms exemplified by the following, the second line of each document being its title:-

JOB	COMPILER ABL	DATA
F6479,SMITH;I.C.T.	F1, SURVEY PROGRAM	F6479 BEETLE SURVEY
------	------	------
------	------	------
(Job Description)	(Program)	(Data)
------	------	------
------	------	------
* * * Z	* * * Z	* * * Z
Job Description	Program Document	Data Document

10.3.3 Rules for Title Preparation

Besides being unique document titles must obey the following rules:-

a) The title must begin with an identification of the person or organization which originated the document: normally this will take the form of an account number or name. For example, documents prepared for the I.C.T. Atlas Computing Service are identified by a letter F followed by an account number.

b) The number of characters must not exceed 80.

c) 'Backspace' must not be used.

d) A title must not contain three successive asterisks (***).

e) Titles must not begin with the word END or the word TAPE.

Furthermore:

a) The characters of a title are read in and stored in Supervisor records in Atlas internal code in the normal way, but the shift characters, 04, 05, 06, 07 are subsequently removed. This means for example that on the Flexowriter the titles

[Tape]

and

1 TAPE 3

are identical. For the same reason a length of run-out appearing on tape in the midst of a title would not become part of the title.

b) Any number of consecutive spaces and tabs are stored as one space.

c) Erases do not become part of the title.

d) Initial commas, spaces and full stops are ignored.

Documents used in the same job need not have related titles.

The job title itself normally contains the name and abbreviated address for the return of the results, but this is not necessary in the titles of data and program documents.

If necessary the title of a data document or job description (but not a program document) may be on the same line, or card, as the heading, provided sufficient room remains to accommodate all the title on that line or card. In this case the heading and title must be separated by 'comma', 'space' (one or more), or 'tab'.

10.4 The Input and Output Sections of the Job Description

After the heading and title a job description is divided into various sections each one describing a particular aspect of the job, e.g. input documents, store used and so on. These sections may be assembled in any order and are dealt with individually below in sections 10.4.1, 10.4.2, 10.6, 10.7 and 10.8.

10.4.1 The Input Section

This section begins with the word

INPUT

which is followed by the titles of the data documents used in the job, each preceded by the number by which the program refers to them. These numbers must be in the range 0 to 15. The program document itself usually is given number 0, but is in fact always taken to be the lowest numbered data document. Thus, if a program operates on two data documents which it refers to as inputs one and two respectively, the job description would contain

```
INPUT
0 (Title of Program)
1 (Title of Data 1)
2 (Title of Data 2)
```

To take a concrete example:-

```
INPUT
0 F1, SURVEY PROGRAM
1 F6479/2 BEETLE SURVEY DATA/62
2 F6479 BEETLE POPULATION 1961
```

The data document "F6479/2 BEETLE SURVEY DATA/62" could then be selected by the programmer by the instruction

```
1050        0        0        1
```

Data may be placed on the same tape as the program, where it becomes a part of the lowest numbered input stream. With an ABL program such data must come between the enter directive and the end of document marker that terminated the program stream. If the 1050 extracode is not used to select a given input stream, the lowest numbered stream is assumed and one obtains the data which followed the program.

The input section may also contain a reference to a magnetic tape on which an especially large document has previously been stored. This way of handling large amounts of input is explained in section 10.12.1.

10.4.2 The Output Section

This section of the job description specifies the type of peripherals to be used for output. Possible types of equipment are:-

```
LINE PRINTER
SEVEN HOLE PUNCH
CARDS
FIVE HOLE PUNCH
ANY
TAPE
```

Here CARDS means the card punch. Each Atlas installation will specify which types of equipment may be used for output ANY e.g. ANY may produce output on LINE PRINTER or SEVEN HOLE PUNCH. TAPE is used when a private magnetic tape is called for to hold an especially large amount of output (see 10.12.4).

The output section begins with the word

```
OUTPUT
```

which is followed by a list of output mechanisms, each preceded by a programmer's number in the range 0 to 15. For example one might have

```
OUTPUT
2 ANY
1 CARDS
0 LINE PRINTER
```

In this case, in order to send output to the card punch, the programmer would first have to select this form of output by the instruction

```
1060        0        0        1
```

A request will be made to the operator to mount special stationery for a given output stream if an asterisk is placed in front of the word LINE PRINTER. Thus if output stream 3 is to be printed on special stationery, the output section should contain

```
*LINE PRINTER m LINES
```

The type of equipment should normally be followed by a limit on the amount of output, specified as so many lines. One line is the output produced by one use of the 1065 (end current record) or the 1067 (output one record) extracodes. Thus one line means one printed or blank line on 5 or 7 track tape and on the line printer, or one punched card. One writes

```
OUTPUT
0 LINE PRINTER m LINES
```

The maximum amount of output may also be specified as n BLOCKS. A block contains 4096 characters. The number of characters allowed for must in general be larger than the number actually printed or punched. On the average each line output to punched paper tape or line printer requires an additional six characters (the maximum possible is 8) to be allowed for. Exactly 8 additional characters must be allowed for to punch one card (making 88 all told). Furthermore each use of 1065 (end this record) to produce a blank line generates 8 characters to be held in the output well.

If the number of blocks is omitted, one block only is allotted, and if the whole output section is omitted

```
OUTPUT
0 ANY 1 BLOCK
```

is understood.

10.4.3 Output : General Notes

a) Output 0 Output 0 is used by the Supervisor and some compilers, but is still available for normal output from the program. It is on this output that such information as the number of instructions done in compiling and executing the program, the number of store blocks in use when the program ended, the number of blocks accumulated for each 'stream' and other such items are printed. It is also used for fault information if the program goes wrong.

If no output 0 is mentioned in the output section of the job description

0 ANY 1 BLOCK

is assumed.

Similarly if no output stream is selected by the 1060 'select output n' extracode, any subsequent output will go to output 0.

b) Atlas can readily accept two or more streams of output from a program for the same type of equipment, even though only one such equipment may exist. The streams are accumulated independently within the computer and eventually output one after another.

In fact all output is accumulated and none will be printed until all computing has ceased unless the extracode

1071 Break output n

is used. In this case all of the output stream n accumulated so far will be sent to the peripheral.

In either case the output information with programmer's number n will always be preceded by

Supervisory Number / Date. Time
OUTPUT n
(Title of Job)

The last line output gives the number of blocks sent to that output.

10.5 A Complete Job Description

We are now in a position to give an example of a complete job description and for the sake of illustration we include the documents of the job.

```
JOB
F64,  J. Smith, I.C.T. London, METALS

INPUT
0 F64,    ANALYSIS PROGRAM
1 F64/A,  IRON CONTENT
2 F64/B,  COPPER CONTENT
OUTPUT
0 LINE PRINTER
1 SEVEN HOLE PUNCH 3 BLOCKS
* * * Z
```

Job Title

Program Title
Data Tapes

output
streams
End-of-tape marker

Job Description

```
COMPILER ABL
F64, ANALYSIS PROGRAM
------
(Program)
------
------
* * * Z
```

Program Document

```
DATA
F64/A, IRON CONTENT
------
(Data)
------
------
* * * Z
```

Data 1

```
DATA
F64/B, COPPER CONTENT
------
(Data)
------
------
* * * Z
```

Data 2

10.6 The Magnetic Tape Section of the Job Description

Magnetic tapes are used with Atlas in two ways. Firstly, they are used by the Supervisor for such purposes as storing input and output. These are called System Tapes, and under normal circumstances need not concern the programmer at all. Their operation is quite automatic (see also section 10.12.3.)

Secondly, the programmer may use magnetic tapes in his program either:

a) For private input and output purposes

or

b) by magnetic tape extracodes.

The use of magnetic tapes for private input and output purposes will normally only be necessary if there is a very large amount of input or output. Full details of the way in which such tapes are employed are given in section 10.12.

The most common use of magnetic tapes is by extracodes within the program. The tapes required may be mounted on a tape mechanism (a 'deck') before running the program or during the actual execution.

The tapes mounted before the job begins must be listed in the job description. Normally the tapes that are to be mounted while the program is running are also listed in the job description. However, in programs which require a not-easily-predicted number of tapes it is possible to get tapes mounted which are not listed in the job description. (Sections 10.6.1. and 10.6.2. show how to list tapes in the job description. The extracodes for mounting tapes are described in Chapter 9.)

Full information on magnetic tapes can be found in chapter 9. However, for the sake of completeness some of the relevant facts are repeated here.

a) Information is stored on magnetic tapes in blocks of 512 48-bit words.

b) The first block is known as block 0, and is not available to the programmer. Block 0 contains the serial number of the tape, and the title if the tape has one. The title of the tape must obey the rules of document titles given in 10.3.2 but also, if the tape serial number is not listed in the job description along with the tape title, the first 30 characters of the title must identify the tape uniquely. Only those 30 characters are printed by the Supervisor when calling for the tape, but up to 80 characters are stored and checked.

c) In preparation for use each tape is mounted on a tape deck by an operator who receives instructions for the purpose from the Supervisor. When mounted, each tape is positioned by the Supervisor at the beginning of block 1, the first block available for storing information.

10.6.2 Single Tapes

Each tape required for a job is specified in the job description by 2 lines of printing:- a heading, and a description. The heading is one of three:-

TAPE - a tape belonging to the user and already having a title;

TAPE NEW - An untitled tape, not previously belonging to the user, to be titled and kept when the job is over;

TAPE COMMON - an untitled working tape to be retained by the system when the job is over.

The description consists basically of the programmer's number (in the range 0 to 99), the tape serial number (preceded by *) and the title of the tape:

21 *F3699 F1000, LONDON SALES, 1963

This format applies with headings TAPE and TAPE NEW. That for TAPE COMMON is described later in this section.

The programmer's number, 21 in the above example, is the number by which the tape will be referred to in the program. If the tape is not required to be mounted before the job begins, but rather will be called for in the course of execution by a "mount" instruction (see Chapter 9), the programmer's number must be replaced by a "tape reference letter" consisting of a single letter:

D *F3700 F1000, LIVERPOOL SALES, 1962

The mount instruction will then refer to the tape by its tape reference letter and assign to it a programmer's number. In the example just given the tape title will consist of the 28 characters starting with F1000 and ending with 1962 (i.e., the two double spaces will have been replaced by single space).

The tape serial number, F3699 in the first example, is permanently associated with the tape. If possible, the Supervisor will call for the tape by its serial number, under which the tapes are filed; the tape number should therefore be included when possible even when its inclusion is not made obligatory by a particular Atlas installation. If it is omitted, the * is omitted also.

Thus in the example

19 T9824, WOLVERHAMPTON SALES; 1962

the title is the sequence of 31 characters starting with T and ending with 2.

The title in the description is the complete title stored in block 0 of the tape. Under a TAPE heading, this will be used to check that the correct tape has been mounted; under TAPE NEW, the title will be written to block

0 when the tape is mounted. In either case if the tape serial number is absent the Supervisor refers to the tape by the first 30 characters of its title.

The description for TAPE COMMON consists of a programmer's number or tape reference letter only.

```
TAPE COMMON
36
```

All the programmer's numbers and tape references in the job description must, of course, be different. A programmer's number must lie in the range 0 to 99 inclusive; a tape reference can be any letter.

The description may refer to logical channel K of the program; the effect is the same as in extracodes 1012 and 1013. This is done by adding .K to the programmer's number.

Thus:

```
21.3    *F3699    F1000    LONDON SALES
```

would request that the tape be mounted on a 'channel' which can then be referred to in the program as channel 3. Tapes on different channels can be written to or read from simultaneously. The extracodes 1016 and 1012/1013 allow further tapes to be mounted on the same channel. (see Chapter 9.)

If a tape in the TAPE category requires file protection, i.e. no write permit ring and/or no write current, an asterisk should be written immediately before the description:

```
TAPE
*1    *F3002    F1002    PARIS SALES
```

With file protection block 0 cannot be written to, so it will not be possible to change the title of such a tape. The program will be monitored if the 1014 (change title) extracode is used on a tape with file protection.

10.6.2 Files

A collection of information may extend over several tapes although the programmer may wish to treat it as a single unit. Such a collection of tapes is called a file and only one of the tapes needs to be mounted at any one time. Each tape of a file is specified in the Job Description by a modified tape heading as follows:

```
TAPE/m
n (tape serial number) (Title of tape)
```

where m is the number of the tape within the file, counting from 1 upwards. The programmer's number n will be the same for all m of this file. The final tape of the file is entered as

```
TAPE/m  END
n (tape serial number) (Title of tape)
```

For example, suppose a file of information extends over three magnetic tapes which have respectively as serial numbers and titles in block 0

Q1432 F1012 BIRMINGHAM SALES

Q1003 F1012 LONDON SALES

Q1100 F1012 MANCHESTER SALES

These would be referred to in the Job Description as

```
TAPE/1
3 *Q1432 F1012   BIRMINGHAM SALES
TAPE/2
3 *Q1003 F1012   LONDON SALES
TAPE/3           END
3 *Q1100 F1012   MANCHESTER SALES
```

These tapes would then form file 3. The Birmingham tape alone would be mounted and allotted programmer's number 3, i.e. the same as the file number, before the job begins.

The remaining tapes are mounted as required by use of the extracode 1007 (mount next reel of file Ba and allocate number n to it). The tape sections on the second and subsequent reels are not numbered consecutively from the preceding reel, but start again at section 1.

10.6.3 Deck Allotment

As stated above the first tape of every file and each single tape that is given a programmer's number in the job description, will be mounted on separate decks before the start of the job. These decks will then be available to the programmer throughout the course of the job. He can by extracodes cause any of the original tapes to be unloaded and new ones, given a tape reference in the job description, to be mounted in their place.

If however, at some stage in the program he requires more decks than will be allocated to him in this fashion, he must mention the number he will need in the job description. This is done by writing:-

DECKS d

where d is the maximum number of mechanisms that will be in use at any one time. Thus a programmer requiring 3 tapes to be mounted at some stage in the program but only 1 at the beginning, would give the single tape a programmer's number, give the other 2 tapes a tape reference and put in his Job Description:-

DECKS 3

10.7 Time Estimates for a Job

10.7.1 Computing Time

Since each program on Atlas will normally be time sharing with others there can be no direct control of an individual program by an operator. Thus, there is no means by which an operator may tell if a faulty program has entered an infinite loop and is thereby wasting machine time.

To overcome this problem a time limit for a program is placed in the job description and if the program exceeds this limit it is stopped by the computer.

The limit appears in the job description as:-

COMPUTING p.q SECONDS (or MINUTES, or HOURS)

where p.q is a decimal number. This time is found by allowing two micro-seconds to each instruction obeyed and adding to this the expected compiling time. The ABL compiler obeys 1500 to 1700 instructions per instruction compiled.

Alternatively the limit on computing time may be specified as

COMPUTING m INSTRUCTIONS

but one 'instruction' in this context means one instruction interrupt, equal to 2048 basic instructions obeyed. In fact the Supervisor actually times the program in terms of these units of 2048 instructions. Conversion from estimates given in terms of seconds, minutes or hours is made on the basis of 256 'interrupts' per second. Furthermore each multiplication instruction is counted as 2 instructions and each division instruction as 4.

For example a program requiring at most three million instructions, and having a compiling time of one and a half seconds would have the entry

COMPUTING 7.5 SECONDS

or

COMPUTING 1832 INSTRUCTIONS

If the computing time is less than 20 seconds this entry may be omitted completely from the job description. In this case a standard allowance of 20 seconds is made (5120 instruction interrupts).

10.7.2 Execution Time

If a program uses magnetic tapes it may be held up at some stage while a block of information is brought from tape to store, a time of 64 milliseconds. This wait can be eliminated in many cases by calling for a block 64 m.s. before it is needed, or by using a sufficiently large variable length transfer buffer or by resorting to branching. Some tape waiting time, however, may be inevitable and if it is likely to occur it must be shown in the job description.

This is done by the heading:-

EXECUTION p.q SECONDS (or MINUTES or HOURS)

where the time estimate is this case is found by adding an upper limit for the tape waiting time to the COMPUTING time.

For example, if the program quoted in 10.7.1 above was expected to be held up at most 200 times the job description would include:-

COMPUTING 7.5 SECONDS

EXECUTION 20.3 SECONDS

If the EXECUTION section is omitted for the job description the execution time is taken to be the same as the computing time.

10.8 Store Allocation

An estimate of the amount of store needed by the program is also required by Atlas to prevent a faulty program from monopolising the store by producing a large amount of useless information. This is done by the job description entry

STORE s BLOCKS

where s is the maximum number of 512 word blocks used by the program at any one time. No distinction is made between core and drum store. The word "BLOCKS" may be omitted if desired.

This section may be omitted, if the store requirement is less than 32 blocks, in which case 32 blocks will be allocated and charged for. If the estimate for store is exceeded at any time the program is stopped by the computer. One extra block should be allowed for each input and output stream. The blocks used by the compiler are not counted unless the computer is retained in the main store after the program is entered.

10.9 Job Description Format

10.9.1 Order of Sections

Separate sections of the job description may be listed in any order. For instance the OUTPUT section could precede the INPUT section, STORE could be followed by COMPUTING. However, to make it easier to read it is perhaps advisable to keep to a fixed order, for example the order in which the sections have been introduced. That is:-

INPUT
OUTPUT
MAGNETIC TAPES (this is not an actual section heading)
DECKS
COMPUTING
EXECUTION
STORE

10.9.2 Case Changes

Throughout this chapter the sections of the job description have been written in capital or upper case letters since these are common to all forms of input. Changes of case are ignored however, and if the job description is on seven-hole tape, lower case letters could be used well.

10.9.3 Backspace

Throughout the job description 'backspace' is an illegal character.

10.10 Composite Documents

10.10.1 Job Description Combined with Program

A job description may be combined with a program to form one composite document. In this case the last item of the job description will be followed, not by an end of tape marker (***Z) but by the program heading:-

COMPILER (Program Language)

and then by the program itself. No further title will be necessary since the composite document takes the title of the job description. This will be the usual form taken be small programs which are only run once.

If this procedure is adopted, no input zero is mentioned in the INPUT section of the job description and the computer will compile and execute the program immediately following the job description. If there are no separate data documents the INPUT section may be omitted completely.

The examples given below illustrate these facilities.

Example 1

```
JOB
F196, J. BLOGGS:  NUMBER FREQUENCY
INPUT
1 F196-TABULATED DATA
OUTPUT
0 LINE PRINTER
COMPILER ABL
------
------
------
------
***Z
```

Job Description/Program

```
DATA
F196-TABULATED DATA
------
------
------
------
------
------
------
------
***Z
```

Data Document

In this example the job required one other data document so the INPUT section must be included.

Example 2

```
JOB
F324/1, W. BROWN, FERRANTI:  PRIME NUMBERS
OUTPUT
0 LINE PRINTER
1 CARDS 2 BLOCKS
COMPILER ABL
----------
----------
----------
----------
***Z
```

Here no further information is required and the INPUT section has been omitted.

10.10.2 Job Description Combined with Data Document

It is also possible to combine the job description with a data document. This is particularly useful when the same program is to be run more than once, using different data each time. In this case the INPUT section of the job description must include:-

SELF = n

where n is the programmer's number for the data which follows on the same document. The program itself is specified as the lowest numbered input stream, in the same way as when the job description is a separate document, and the last item of the job description is followed by the heading

DATA

and then the data itself. No title is needed to identify the data.

For example, consider a wage calculation carried out each month using one fixed set of data, say a list of P.A.Y.E. codes and a second set of data consisting of a list of hours worked by each member of the staff. The second set of data would, of course, vary from month to month and could be combined with the job description while using the same program and P.A.Y.E. code tapes.

The program and data documents would be:-

```
COMPILER ABL
F900, WAGE CALCULATION

------
------
(Program)
------
***Z
```

```
DATA
F900, P.A.Y.E. CODES

------
------
(Data)
------
***Z
```

The job description could then be combined with the second data document thus:-

```
JOB
F900, J. SMITH LTD:  WAGES OCTOBER 1964
INPUT
0 F900, WAGE CALCULATION
1 F900, P.A.Y.E. CODES
SELF = 2
OUTPUT
0 LINE PRINTER 50000  LINES
1 CARDS 2000  LINES
DATA
------
------
(Data)
------
***Z
```

10.10.3 Data Files

It may be easier, for the purposes of some programs, to treat several distinct paper tapes or stacks of cards as a single data 'document'. Such a combination is called a data file. Each separate document is given a modified data heading of the form

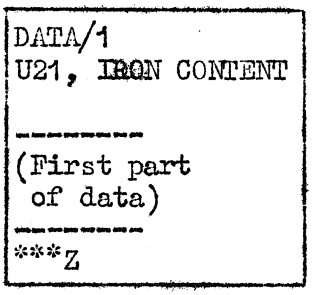
DATA/m

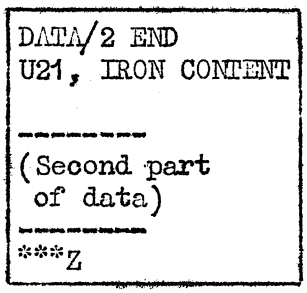
where m is the number of the document within the file. All documents composing the file have the same title and each is ended by a ***Z end-of-tape marker, or a 7, 8 card punching. The last member of such a series must have the heading:-

DATA/m END

These documents may then be fed to the computer in any order and on any peripherals and the computer will combine them as required. In the INPUT section of the job description they will be referred to as one document with the title which each of them bears.

For example, if the data called U21, IRON CONTENT is on two distinct paper tapes these may be headed as follows:-





If the programmer wishes to refer to the file as input two the INPUT section of the job description will contain:-

2 U21, IRON CONTENT

10.11 Tape and Card Markers

So far in this chapter only the marker ***Z and the 7, 8 punching have been considered as an end to a document. These are in fact the most common markers but there are others which are dealt with below.

On punched tape all the markers consist of *** followed by a single letter. On cards *** is not acceptable, and is replaced by punching the 7 and 8 positions in the first column of the card and the letter in the last column. The other 78 columns can contain anything at all. If the last column is blank, 7, 8C is assumed.

10.11.1 The Tape Markers ***Z, C, T and A.

a) ***Z indicates not only the end of a document but effectively also the physical end of the tape. The peripheral equipment concerned is disengaged by the computer and when re-engaged by the operator a new document will be read.

b) ***C indicates that the end of the current document has been reached but that another follows on the same paper tape. The end of the document is noted by the computer and reading is continued for the next document without interruption.

c) ***T indicates a temporary stop. When this marker is encountered the peripheral equipment is disengaged by the computer and when next engaged by the operator a continuation of the same document is read. Thus, if a document consists of two tapes the first part can be ended with a ***T. When this has been fed to the computer the second part is read by the same peripheral with no document heading and the computer will treat the two parts as one document.

If the document is data it is better to use the data file system given in 10.10.3 since the parts may then be fed to the computer in any order, on any peripheral.

d) ***A is used only by a machine operator and is an instruction to the computer to abandon the previous incomplete document and disengage the equipment. It is required if part of a document is damaged before input is complete and the operator requires the computer to disregard the information it has already received.

10.11.2 The Binary Tape Markers ***B, E and F

Each of these markers indicates that a binary tape follows.

a) ***B. When this marker is encountered the computer reads the information following on the same document in binary, instead of internal code, to the physical end of the tape. There is no test for end of tape markers. The last 2 or 4 characters from a paper tape read in this fashion will be overwritten in the store by the 12 bit character 0707 (octal), which replaces any spurious characters generated as the end of the tape passes through the reader (this does not apply to punched cards).

b) ***F causes the mechanism to be disengaged. When it is next engaged by the operator the new paper tape is read to its physical end in binary. Thus ***F combines the effects of ***T and ***B.

c) ***E causes the input following on the same tape to be read in binary but a check is made for ***A, Z or C. When one of these is encountered, binary reading ceases and the appropriate action is taken. If the end of a reel of tape is encountered after ***E and before any of these *** sequences, the last 2 or 4 characters on the tape will be denoted by the 12 bit character 0707 (octal); the next tape will then be read as a continuation of the same binary document. When ***A, Z or C is encountered, it is itself stored in binary. On cards following 7,8E a card bearing 7,8A, Z or C is also stored in binary.

Note that if it is required to read to the physical end of a 7-track tape it is necessary to precede the ***B, E or F by ***P, as described below. If this is not done the tape may be rejected because of a spurious tape parity fault when the end of tape passes through the tape reader.

The marker ***B can also be used to read a fixed number of characters in binary. This is done by prefixing the marker by *n thus:-

*n***B

where n is the number of characters to be read. When the n tape or card characters have been read and stored, reading continues in internal code.

When reading punched cards an alternative way of causing the card to be read in binary is available. If the first column of a card is a standard code the contents of the card are converted to internal code. If the first column is not a standard code the card is assumed to be in binary and is stored as such. Naturally, after a 7,8 B, E or F has been read all cards are taken as binary regardless of their first column.

10.11.3 The Tape Marker ***P

When reading 7-track paper tape in internal code or in binary the Supervisor normally checks the parity of each character (an odd number of holes for correct parity) and rejects the document if a character with wrong parity (blank tape for example) is encountered. This parity checking may be suppressed by punching ***P, but it will be restored again at the end of the given document. If the input is binary and ***P has been punched, wrong parity characters will be recorded as punched, but if the input is internal code they will be replaced by the fault character 7.7 (inner set).

Note that to suppress parity checking on binary input it is necessary for ***P to appear before ***B, E or F; ***P after ***B, E or F will not be recognised.

10.11.4 Card Markers

The same markers are available for use with cards, but with *** replaced by a 7,8 punching in the first column of the card and with the letter in the last column.

10.12 Input and Output using Private Magnetic Tapes

All the documents fed to peripherals for input are stored by the computer on a magnetic tape known as the system input tape. They are then brought into the core store as required. If a program has a large amount of input it may be in the programmer's interest, and in the interest of the efficiency of the computer, to transfer this input to a private tape before starting his program, instead of using the system input tape.

Output is normally accumulated on the system output tape before being sent to the required peripherals. If a program involves extensive output this may be written to a private magnetic tape instead, and later output upon requesting the Supervisor to do so by means of a special document fed in through one of the peripherals after the job is complete.

Both these facilities are dealt with in this section.

10.12.1 Extensive Input

A document can be copied to a NEW magnetic tape by putting above the document heading the directive:-

```
COPY TAPE NEW
*tape number  tape title
```

The title specified will be written in section 0 of the new tape and the document following will be copied into section 1 onwards.

If it is required to store the document on a previously used tape (i.e. one which already has a title) the necessary heading is:-

```
COPY TAPE b
*tape number  tape title
```

Here b is the number of the tape block at which it is intended to begin copying the input.

When the copying process is complete the following information will be printed by the computer:-

```
*tape serial number/section/word
 title document.
```

The section and word indicate where the document is stored.

Example:
To copy the first data document of the job given in 10.5 to the NEW tape F1111 it would be fed to the peripheral in the form:-

```
COPY TAPE NEW
*F1111 DATA FOR F64, METALS
DATA
F64/A, IRON CONTENT
- - - - - - - - - - -
(data)
- - - - - - - - - - -
- - - - - - - - - - -
***Z
```

The information output by the computer would be:-

```
*F1111/1/0
F64/A,  IRON CONTENT
```

The second data document could be sent to section 7 onward of the same tape thus:-

```
COPY TAPE  7
*F1111 DATA FOR F64, METALS
DATA
F64/B, COPPER CONTENT
- - - - - - - - - - -
(data)
- - - - - - - - - - -
- - - - - - - - - - -
***Z
```

This could produce the output:-

```
*F1111/7/0
F64/B, COPPER CONTENT
```

10.12.2 Job Description References

In order to use documents copied to private tape, the tape must appear in the tape section of the job description in the usual way. In the INPUT section of the job description the sub-heading:-

```
i TAPE a/b/c
```

must appear before the title of the document. Here i is the input stream number, a is the programmer's number for the tape, b the block number, and c the word number within block b at which the document starts. Thus the job description to run the program of 10.5 could begin:

```
JOB
F64, STATISTICAL ANALYSIS OF METALS
INPUT
0 F64, ANALYSIS PROGRAM
1 TAPE 27/1/0
F64/A, IRON CONTENT
2 TAPE 27/7/0
F64/B, COPPER CONTENT
OUTPUT
0 LINE PRINTER
1 FIVE HOLE PUNCH 3 BLOCKS
TAPE
27 *F1111 DATA FOR F64, METALS
```

10.12.3 Re-use of Documents on System Tapes

As already explained, under normal circumstances documents are stored on system tapes before use. Among the information on output 0 will be the location of each document on tape. This will take the form:-

```
Sa/b/c
```

where Sa is the system tape number, and b,c are the block and word numbers of the first word of the document. Such tapes will be stored for a fixed period of time and if a document is required again within this time it may be called for direct from system tape, avoiding the use of a slow peripheral. This is done by including in the INPUT section

```
i  TAPE  Sa/b/c
Title of document
```

where i is the programmer's number for the document.

For example, if a program called 'F74 FACTORISATION' has been run and is stored at S7/2/411 it can be re-run with a data document called 'F74 LIST 3A' by a job description as follows:-

```
JOB
F74 FACTORISATION RUN 2
INPUT
0 TAPE S7/2/411
F74 FACTORISATION
1 F74 LIST 3A
OUTPUT
1 LINE PRINTER
***Z
```

Only the job description and the data document would then need to be fed to peripherals. Note that no further reference to the system tape is required.

10.12.4. Extensive Output

Large quantities of output may be written to a NEW magnetic tape by specifying in the OUTPUT section:-

```
OUTPUT
i TAPE a/b/c
type of equipment m BLOCKS
```

Here, i is the output stream number, a the programmer's number for the tape, b the block number, and c the word number where the copying is to start. The last line is as in section 10.4.2.

Thus if output 1 of the job in section 10.5 were 300 blocks instead of 3 the OUTPUT section could be written:-

```
OUTPUT
0 LINE PRINTER
1 TAPE 27/1/0
SEVEN HOLE PUNCH 300 BLOCKS
```

The tape must be specified in the usual way in the tape section of the job description:-

```
TAPE NEW
27 *F1111 F64, OUTPUT METALS
```

This will cause the NEW tape F1111 to be given the title 'F64, OUTPUT METALS' and the output to be sent to this tape beginning at block 1.

If it is required to store the output on a previously used tape, the necessary entry in the OUTPUT section is the same, but the tape must be specified under the heading TAPE in the tape section.

The private tape can be printed by a steering tape consisting of:-

```
PRINT TAPE
*tape serial number/tape title
```

if the whole tape is to be printed

or

```
PRINT TAPE b/c
*tape serial number/tape title
```

if one document is to be printed, from block b word c onwards.

10.13 Job Description Parameter

In a job where different parts of a standard program are required to process different types of data, it may be more convenient if the data type is indicated in the job description rather than in the data or program. The parameter section in the job description provides for this, and is written in the form

PARAMETER

*<number>

The number consists of up to eight octal digits, and is left justified. PARAMETER and the number may be on the same line, separated by one or more spaces.

Examples:

PARAMETER *00061247

PARAMETER
*1060

PARAMETER *1060000

(PARAMETER / *1060 / PARAMETER *1060000 — alternative forms)

When the parameter section appears with other sections, the order of sections is immaterial; if omitted, the parameter is taken to be zero.

The value of the parameter may be read by program using the instruction

```
1140   4   0   S
```

which will set the half-word s' = parameter number (see section 12.9).

Chapter 11

MONITORING AND FAULT DETECTION

11.1 Supervisor Monitoring

With the aid of special hardware, the Supervisor keeps a record of the progress of a program during its run. This supervisor monitoring notes amongst other things the store in use, the computing time taken, and the occurrence of errors which may prevent the successful execution of the program.

In Atlas, provision is made for the automatic detection of a variety of clearly defined fault conditions, which may be due either to mistakes in the program itself, or to errors occurring in the computer or the peripheral equipment. The faults result in entry to a part of the Supervisor known as the 'monitor' program, with the particular fault responsible being distinguished by a mark or count set in B91.

The monitor program consists of a set of routines, some in the fixed store and some in the main store, whose primary purpose is to print out such information as will enable the cause of the fault to be diagnosed. In the case of certain types of program fault, it may be possible for the programmer to decide beforehand what action should be taken to enable the program to be resumed; he will then provide a number of fault routines and list them in the trapping vector. The monitor program will enter the trap indicated in B91, if such indication does in fact correspond with an entry in the trapping vector; otherwise it will proceed to diagnostic printing, or will transfer control to a private monitor routine if so requested. This private monitor program may provide a special form of diagnostic printing - either instead of or in addition to the standard - and may, once and once only, cause the program to be resumed. Any subsequent monitoring will always be followed by the End Program sequence, except when the fault is trapped.

When entry to the private monitor follows expiry of the total time allotted to the program, a further 4 seconds of computing time is allowed for the completion of the private monitor routine. Similarly, if execution time is exceeded, a further minute of execution time is allowed, and if Output is exceeded, a further block is allowed.

11.1.1 Types of Program Faults

We shall here concern outselves solely with program faults: there is little that the ordinary programmer can do about machine faults, other than to minimize their effect by providing adequate re-entry points and an informative private monitor routine.

There are three distinguishable categories of fault detection causing entry to the monitor program:-

(a) *Interrupt Faults:* Some faults are detected by special equipment provided for the purpose: these include exponent overflow, division overflow,

use of an unassigned function code, and 'sacred violation' (i.e. reference to a part of the store reserved by the Supervisor). An interrupt occurs and will set in B91 a digit corresponding to each such fault. Further analysis of the fault is provided by a supervisor extracode routine (S.E.R.), the same routine being used for all faults.

(b) '*Supervisor Faults*' A further class of faults are detected by S.E.R.'s, being especially concerned with faulty use of the store and of peripheral equipments, and with the over-running of time allowances. These faults lead to the setting of an appropriate digit of B91, just as if they had been detected by hardware, as in (a) above; they include over-running of computing time or of tape waiting time, and attempts to exceed the requested store allocation. In the object program, extracode instructions dealing with the store and peripherals will enter a S.E.R. to detect faulty usage. Only one such fault can be detected at once, but it will be recorded as a count in B91 without interfering with any existing record of faults of the interrupt type. The same S.E.R. monitor sequence is entered as in (a) above.

(c) '*Extracode Faults*' Many faults are detected by the ordinary extracode routines themselves; typical of these are errors in the arguments of functions. Only one such fault can be detected at once; the extracode will set a counter in B91 and then jump directly to the common S.E.R. monitor sequence mentioned above.

The various faults which result in entry to the monitor program are listed in the table below:-

FAULT TABLE

FAULT	MONITOR PRINTING	DETECTED BY	MARK OR COUNT IN B91	TRAP NUMBER (IF ANY)
Local time expired	L TIME EXCEEDED	S	Bit 5	0
Division Overflow	DIV OVERFLOW	I	Bit 6	1
Exponent Overflow	EXP OVERFLOW	I	Bit 14	2
Page locked down	PAGE LOCKED DOWN	S	Bit 1	3
Number of blocks exceeded	EXCESS BLOCKS	S	2.0	4
Square root argument < 0	SQRT -VE ARG	E	2.4	5
Logarithm argument ≤ 0	LOG -VE ARG	E	3.0	6
SPARE				7
Inverse trig. function	INVERSE TRIG OUT OF RANGE	E	4.0	8
Reading after Input Ended	INPUT ENDED	S	4.4	9
End of Magnetic tape	END TAPE	S	5.0	10

FAULT	MONITOR PRINTING	DETECTED BY	MARK OR COUNT IN B91	TRAP NUMBER (IF ANY)
Variable length record error	V TAPE ERROR	E	5.4	11
Magnetic Tape failures	TAPE FAIL	S	6.0	12
Computer failures	COMPUTER FAIL	S	6.4	13
Unassigned Function	ILLEGAL FUNCTION	I	Bit 4	
Sacred Violation Instruction	SV INSTRUCTION	I	Bit 8	
Sacred Violation Operand	SV OPERAND	I	Bit 10	
Illegal block number	ILLEGAL BLOCK	S	9.6	
Band not reserved	BAND NOT RESERVED	S	10.2	
Computing time expired	C TIME EXCEEDED	S	Bit 2	
Execution time expired	E TIME EXCEEDED	S	Bit 3	
Input not defined	INPUT NOT DEFINED	S	11.6	
Output not defined	OUTPUT NOT DEFINED	S	12.2	
Output exceeded	OUTPUT EXCEEDED	S	12.6	
Tape not defined	TAPE NOT DEFINED	S	13.2	
Illegal search	ILLEGAL SEARCH	S	13.6	
No selected tape	NO TAPE SELECTED	S	14.2	
No mode defined or attempt to write when not permitted	WRONG TAPE MODE	S	14.6	
Number of decks exceeded	EXCESS DECKS	S	15.2	
No trap set	TRAP UNSET	S	15.6	
Number of branches exceeded	EXCESS BRANCHES	S	16.2	

(I = Interrupt; E = Extracode; S = S.E.R.)

Some of these faults are associated with information in the job description, and are listed below.

Excess Blocks

Computing Time exceeded

Execution Time exceeded

Input not defined

Output not defined

Output exceeded

Tape not defined

Excess Decks

Division overflow occurs with certain division instructions, marked DO, when the divisor is zero or substandard.

Exponent overflow occurs with certain instructions, marked E, when, after completing all functions of the instruction, including rounding and standardisation, the accumulator exponent is greater than +127. The guard and sign bits of b124 will be 0 and 1 respectively. If exponent overflow is trapped, the fault indication will not be reset and may cause further monitoring with a subsequent accumulator operation. This may be avoided by first clearing B124, setting guard and sign bits to zero. If the accumulator exponent becomes less than -128 during multiplication, division or standardisation, then the guard and sign bits of b124 are 1 and 0 respectively. This is exponent underflow. The accumulator is set to floating point zero, but no fault is indicated.

The arguments for square root, logarithmic and inverse trigonometric functions are all tested to determine whether they are within range. They may cause exponent overflow if very large arguments are used.

Programs that employ line reconstruction, such as the ABL compiler and L100, will attempt to read after the input stream is finished if the end of document marker ***Z etc. occurs on the same line as the last information for the program.

Sacred Violation Instruction refers to a transfer of control to address J5 or above. If control is transferred to the fixed store, between J4 and J5, there is no immediate interrupt, but the result is unpredictable.

Sacred Violation Operand always means an attempt to read or write to the private store (i.e. to J5 or above).

Illegal block number indicates a reference to the Supervisor working store, between J34 and J4.

Certain functions are detected as being illegal. The instruction

1000 0 0 0

which is the same as floating point zero, causes entry to extracode control to attempt to obey the instruction

001 0 0 0

or the floating point number $\frac{1}{2}$. This function is recognised as being unassigned.

End of magnetic tape refers to an attempt to obey a transfer instruction involving section 0, or to use tape beyond the last addressed section (section 4999 on a fully addressed 3600 ft. tape).

Illegal search refers to an attempt to search a magnetic tape for section 0 or beyond the last addressed section.

No magnetic tape selected indicates that none of the 'select tape' extracodes has been obeyed.

Variable length record error implies an attempt to transfer when not aligned on a tape marker, or an encounter with an incorrectly made marker.

Should a magnetic tape failure occur, the program may not be monitored immediately, and in this case an attempt will be made, by the Supervisor, to produce a correct transfer. After a set number of failures, usually seven, full monitoring will then occur. An attempt to read from a section that has not previously been written to will be recognised as a tape failure, and hence it is advisable to write to all sections required when a new tape is used. A decimal number may follow the TAPE FAIL text, and this is interpreted as

$$8192A + 512B + C$$

where C is the tape number on which the failure occurred

B is the fault number

and A is the fault type, 0 if E-type, 1 if F-type.

These fault types are described in the Atlas 1 Computer System Operators' Manual.

A monitor for 'Trap Unset' indicates an attempt to enter a trap using extracode 1134 when no tape has been set by extracode 1132 (see section 11.2).

An attempt to exceed the number of active branches specified by extracode 1103 will be monitored (see Chapter 12).

'Page Locked Down' and 'Band Not Defined' monitoring may occur when a program is controlling transfers between the drum and core store (see Chapter 12).

11.1.2 Standard Post Mortem

Following the detection of one or more of the faults listed above, the appropriate text will normally be printed on Output 0, followed by a standard form of post mortem printing, similar to that shown below.

```
BAND NOT RESERVED
INSTR  73        121,127,0   , 549      B127 =  0
INSTR  74       1177,0  ,12  , 0        B12  =  4.6
INSTR  75        101,13 ,12  , 108.4    B13  = -1048574.6 B12 = 4.6
B1  =-699050.6   B2  =-1031932   B3  = 42798.7    B4  =-42799
B5  =-0.1        B6  =-258553    B10 = 18         B12 = 4.6
B13 =-1048574.6  B70 = 0.6       B80 = 0.1        B90 = 36
B91 = 10.3       B92 = 0.1       B93 = 0.1        B94 = 7.1
B95 = 263.1      B96 =-0.3       B97 = 917504.4

    TAPE 1       AT   64
    TAPE 2       AT   72/306
    TAPE 3       AT   1
```

Normally the current address in main control, B127, is printed, followed by its full word contents in instruction form, and the contents of any

B-lines in the range B1 to B99 referred to, unless zero. Irrespective of their true value, the contents of any B-lines in the range B100-B127 referred to will be printed as zero. The two previous words are similarly printed. If one of the locations is in the private store or is undefined, then

INSTR UNALLOCATED

will be printed instead of the normal form. Following an EXCESS BLOCKS fault, b127 will be reduced by 1 before printing instructions. The instructions printed may not be the last three obeyed, as a jump may have occurred to either the second or third instruction printed, as must have happened in the example. Also, because of the overlap in executing instructions with each other, and with other operations such as tape transfers, the printing may occasionally bear little relation to the instruction originally causing the fault, although many extracodes causing faults will appear as the second instruction; basic functions causing faults may appear as any one of the three instructions, or not at all.

After the three instructions, the contents of all B-lines B1-B99 are printed, unless zero.

If one inch magnetic tapes are being used, their current positions are then printed out, indicating the next section number, and, for variable length working, the word number also.

Except for the function codes, which have their usual form, and octal fractions in addresses and B-lines, all numbers are decimal.

11.1.3 Ending a Program

If no fault is found during the execution of a program, then the run will be finished by obeying the 1117 extracode. If a fault is found, then, after post mortem printing, the Supervisor will normally end the run as if the 1117 instruction had been obeyed.

1117 Print monitoring information on output 0. End all program output streams, indicating the amount of output in each stream. Instruct the operators to disengage and rewind all magnetic tapes used. Remove the job from the store; clear Supervisor directories relating to this job.

The form of the monitoring information on output 0 is given below.

```
<Job Title>
INSTRUCTION 8      7
STORE 32       / 4
 2 DECKS 6      TAPE BLOCKS 1      HALT TIME
INPUT 0         1         BLOCKS
OUTPUT 0    (3) ANY 40       RECORDS
```

The meaning of this information is explained below.

INSTRUCTION 8 7

8 instruction interrupts were obeyed in all, of which 7 were used in compiling. An instruction interrupt occurs every 2048 instructions, basic multiplication and division orders being counted as 2 and 4 instructions res-

pectively. At some installations a third number on this line gives the number of program blocks transferred between drum and core store.

```
STORE 32        / 4
```

The job description requested 32 blocks of store, of which 4 blocks were in use when the program was terminated, including one block for each input or output stream.

```
2 DECKS 6       TAPE BLOCKS 1       HALT TIME
```

Two magnetic tape decks were reserved, and six blocks were transferred between tape and main store. The program was suspended by the Supervisor for one second awaiting the completion of tape transfers.

```
INPUT 0          1       BLOCKS
```

One block was needed in the input well to hold the information on input stream 0. There will be similar printing for all other input streams defined.

```
OUTPUT 0      (3) ANY 40      RECORDS
```

The job description requested that output stream 0 should be to ANY type of peripheral. Forty records were output before the program was terminated. If the actual peripheral is a paper tape punch, the output will be measured in blocks; otherwise, in records. The output stream was broken into three parts; this printing is suppressed if the stream was not broken. There will be similar printing for all other output streams defined.

The amount of output, indicated at the end of each stream, is always measured in blocks, in the form

```
END OUTPUT       1 BLOCKS
```

11.2 The Trapping Vector

Upon entry following a fault, the first action taken by the monitor program is a check to see if the fault has been trapped by the programmer; if so, the monitor sequence will at once exit to the trap set.

The trapping vector occupies several successive words of the store, and the address of the first word must be specified as S in the extracode 1132. Each word holds the trapping information for a particular fault, word n being associated with fault type n; the more-significant half-word contains the address of the fault routine to which the trap will transfer control, and the less-significant half-word contains, in bits 15-21, the address of a B register which is used to hold the value of main control when the fault was detected. This may be but is not necessarily the point in the main program at which the fault occurred.

Only some of the faults listed in the table in section 11.1.1 have trap numbers: these are the faults which the programmer might reasonably be expected to deal with before resuming the program; certain traps may be useful as a means of avoiding extra testing in the program. There are faults which are not trappable; these include such faults as sacred violation, which the programmer can be expected to avoid, and deviations from the job description, in exceeding the specified time allowance, for instance.

No trapping will occur unless the program first obeys an 1132 instruction, specifying the address of a trap vector. Subsequently, trapping can be suspended by specifying a negative address in extracode 1132. In order to trap some faults but not others, the programmer should specify a negative jump address in each unwanted trap. Normally the trap vector will be punched as part of the program, and the unwanted entries may be punched as +0 or merely omitted, since floating-point zero produces a negative first half-word, because its exponent is -128.

Two other extracodes associated with the trapping vector are given below.

1133 Place the first address of the trapping vector, if any, in Ba; if no trap has been set, make ba negative. This enables a subroutine to preserve the current trap when a private trapping vector is required.

1134 Obey the entry number Ba in the trapping vector, as though a fault of type Ba had occurred. Ba may range from 0 to 63 inclusive, and may be used to enter standard traps, or as a subroutine entry at addresses listed in the trapping vector for 'fault' numbers 14 onwards.

11.3 Private Monitoring

When the monitor program encounters a fault which is not trapped, it prepares to terminate the program and proceed to diagnostic printing, as described earlier. If he so desires, the programmer may supply a private monitor sequence, whose starting address must be specified by using extracode 1112. The last octal digit of the starting address determines the time of entry to the private monitor as follows:-

Octal Fraction	Entry
1	Before printing the one line explanatory text
0	After printing the text
2	After the standard post mortem printing

When the entry is before any printing, B91 contains the record of faults, and B92 the value of main control when the fault was detected, with the contents of B93 and B121 altered. Otherwise B96 and B97 will also be altered.

In the event of faults in the private monitor sequence itself, it is necessary to avoid the possibility of endless loops of errors; this is accomplished by forbidding a second entry to the private monitor. Any subsequent faults may be trapped, but if they are not trapped the standard monitor will end the program.

Specifying a negative address with extracode 1112 cancels any private monitor routine.

11.4 Restarting and Re-entering a program

Following a failure in the computer or an on-line peripheral, jobs completely in the machine will not be lost, although documents of incomplete jobs, and documents only partially read must be input again. Programs partially executed will normally be restarted from the beginning; there are no facilities at present by which the Supervisor will dump program information to allow re-entry to a point other than the start of a program.

11.4.1 Preventing a Restart

It may be useful to prevent a job using the 'break output' facility from being restarted once a point is reached where the job is substantially complete; alternatively a restart may be impracticable for a job using magnetic tape. To this end, the instruction

```
1113    0    0    -1
```

will prevent the job being restarted if a breakdown occurs following the use of the extracode, but before completion of execution.

11.4.2 Re-entering a Program

After a breakdown, for a program to be re-entered at some point other than its start, it is normally necessary to dump information as the job proceeds, specifying a re-entry point with each dump. Although the Supervisor does not yet provide such facilities, the programmer may provide his own dump routine.

Such a routine, called Dumpling, is described in the I.C.T. Atlas Computing Service Bulletin No. 7. Dumpling occupies one main store block, dumping, on to magnetic tape, the information listed below:-

All defined store blocks, including Dumpling.

The tape numbers and positions of all one inch magnetic tapes working in fixed length mode.

The accumulator (double length)

The logical accumulator

B-lines 1 to 90, and B121

The number of the currently selected input stream.

The number of the currently selected output stream.

V-store line 6 containing A0, Bt, Bc, etc.

The address of the trapping vector, if any.

Details of the dump region.

The dump number.

The re-entry point to the program in the event of a breakdown.

As each dump is made, its location is printed on output stream 0.

After a breakdown, a very short steering program allows the information stored at the last or the penultimate dump to be recovered, and the program continues from the corresponding re-entry point.

11.5 Monitor Extracodes

The monitor extracodes introduced earlier are listed again here.

1112 Set the address of the private monitor routine to S. If the standard monitor program is subsequently entered, following a fault which is not trapped, control will be transferred to the private monitor, possibly after some diagnostic printing. The amount of diagnostic printing is determined by the octal fraction of S, as follows:

Octal fraction	Print-out
0	One line describing the fault.
1	No standard printing.
2	One line describing the fault, followed by standard post mortem print-out.

To subsequently suspend private monitoring, a negative address, S, must be used with 1112.

1113 0 0 -1 Do not restart.
If a breakdown occurs after obeying this instruction but before the job is completed, it will not be restarted.

1117 End program
Print monitoring information on output 0; end all output streams. Instruct the operators to disengage and rewind any magnetic tapes used. Clear all Supervisor references to this job.

1132 Set the address of the trapping vector to S.
This extracode must be obeyed before any trapping can take place; to subsequently suspend trapping, 1132 must be used again, but this time with S negative.

1133 Find address of trapping vector.
Set ba′ to the first address of the trapping vector if one is defined, otherwise set ba′ negative.

1134 Enter trap Ba. ($0 \leq Ba \leq 63$)
Obey entry number Ba in the trapping vector, as though a fault of type Ba had occurred.

11.6 Faults Detected by the Compiler

Apart from faults detected during the execution of the program, many types of error may be found earlier by the ABL compiler. An indication of the type and location of each fault is printed out on the current output stream, usually output 0, and, if necessary, arbitrary values are assigned to expressions to allow compiling to continue.

There are some special preset parameters which determine the exact action taken by the compiler after a fault has been found. These parameters are described in Chapter 12; the compiler action described in this section assumes no program setting of these parameters. In particular, P110 will normally be zero.

11.6.1 B-lines in ABL

It may be useful to know whether a run has ended during compiling or execution. When ABL is in operation, B3 is in the range -127 to 0 inclusive. The most likely value is -127, unless the run has been terminated by INPUT ENDED.

ABL uses most of the B-lines. It preserves its own B-lines when it meets an enter directive, and then clears B1 - B88 before obeying the directive. B89 will contain the current value of *. After an E-directive, B90 contains J3; after ER or EX, B90 is clear.

11.6.2 Indeterminate Items

When ABL needs to evaluate an expression, and can not do so, the expression is faulted as described below, and an arbitrary value assigned. The value depends on the context.

(i) * = expression. When this is faulted, * is given the value 654321P110 or 654321.1P110. This allows ABL to try to check the rest of the program, although it may not be able to enter it.

(ii) In all other cases the expression is given the value J36. If the expression is in an Enter directive, it will cause a monitor on ILLEGAL BLOCK, since J36 is an address in the Supervisor Working Store. A similar monitor is likely if the expression occurs in an instruction which is subsequently obeyed.

When any other faulty item is found, nothing is compiled. As the store is initially cleared to floating point zero, all or part of this bit pattern will remain in the location reserved for the faulty item.

11.6.3 Diagnostic Printing

The first ABL diagnostic printing for a program is preceded by the line of printing,

ABL MONITORING

Each fault causes printing, on a new line, of the location of the fault within the program, together with an explanatory text. The following line will usually be a reconstruction of the line of program containing the fault.

If the error density is higher than 8 faults in 24 lines of print-out, then

TOO MANY ERRORS

is printed, and the run ended (see also P101). Blank lines, and lines containing only erases, are ignored.

Normally, when a fault is found, compiling continues until an E or ER type of enter directive is encountered, so that any further faults may be detected. When the enter directive is reached after one or more faults,

ERRORS DO NOT ENTER

is printed, and the run ended (see also P102).

11.6.4 Fault Location

Each ABL monitor printing begins with a specification of 'where' in the program the offending item occurs. 'Where' means 'in what line of the printed program and in what part of that line'. ABL refers to lines of print in exactly the same way as a programmer does by counting from the last label set, but with the exception that A0 is not used to mean the line on which the first instruction or item of a routine appears but the line in which the R itself appears. Thus the very first line of print in a program is 1 A0/0. Blank lines, and lines containing only erases, are ignored.

When ABL prints 3,4 A1/20 it means that it has read 3 terminators in line 4 after A1 of routine 20 when it has found an error. Thus, for example, if an expression is incomplete (e.g., no close bracket after an open bracket), then the next terminator will have been encountered before ABL can realise that there is an error, and the 'position along the line' will be printed accordingly; but, if a bogus character is found in an expression, then this will be recognized before the next terminator is encountered.

11.6.5 Diagnostic Printing Character Set

In all diagnostic printing, ABL uses its own character set. The following is a list of all available external characters (7-track tape, 5-track tape, and cards) and their corresponding ABL characters. All the ABL characters are contained in the 7-track tape and Anelex line-printer character sets, so that ABL diagnostic printing may always be completely printed or punched on these media.

External: 0 1 2 3 4 5 6 7 8 9
ABL : 0 1 2 3 4 5 6 7 8 9

External: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
ABL : A B C D E F G H 1 J K L M N O P Q R S T U V W X Y Z

External: a b c d e f g h i j k l m n o p q r s t u v w x y z
ABL : A B C D E F G H 1 J K ∝ M N O P Q R S T U V W X Y Z

External: ½ / ∝ β . + - : ' [ ] < > = _ ; ( ) ? ª π &
ABL : β / ∝ β . + - : ' [ ] < > = ∝ ; ( ) ? ∝ | M |

External: * 10 11 % £ ≠ ∨ ≃ ≥ → x ∅ n
ABL : * ∝ ∝ ∝ | ∝ ∝ ∝ ∝ ∝ ∝ ∝ ∝ '

All other single characters are replaced by ∝. All composite characters not including Erase are replaced by β. All composite characters including Erase are ignored. All single spaces are ignored. Two consecutive spaces are replaced by Comma. After a Terminator all spaces are ignored, but not commas (see fault below).

l (lower case L) is an illegal character, unlike the rest of the lower case alphabet; i and o are treated as one and zero respectively, as are capital I and O.

11.6.6 Explanatory Texts

The ABL texts are listed below, with further explanation where necessary.

INSTRUCTION?
ABL thinks that a line contains an instruction, but something is incorrect with the format, i.e. a line begins with a number (unsigned), and does not have four parts.

OCTAL NO. CONTAINS 8/9

IRREGULAR FUNCTION
ABL thinks that a line contains an instruction, and its first part has not been faulted by either of the two preceding monitors, but contains either more than 4 digits, or four with the first equal to 2 or more.

WRONG FORMAT
A line contains items which cannot be identified; whatever kind of items they are meant to be, something is wrong.

NOT TERMINATED

A character which has no meaning in the current context is encountered within an item. When this occurs, ABL skips to the next terminator.

ACCUMULATOR OVERFLOW

Fixed-Point Overflow is caused in the Accumulator as a result of any arithmetic process required because of the form of any item. In practice, this can only occur during the 'de-standardising' process required by 'd' in the Floating-Point Number forms a:d, a(b):d, a(b:c):d, and a(:c):d.

SHIFT >23 PLACES Requested by a U or a D operator.

IMPERMISSIBLE +

Z in R0

* OUT OF RANGE

An attempt is made to compile an item into store with the transfer address (*-P110) greater than or equal to J3, or less than 0. Compiling will continue from *=654321.1 + P110 in an attempt to detect any other faults.

PARAMETER OR ROUTINE NO. TOO BIG

A parameter, routine, or copy number is greater than the permitted highest value, that is

(i) if a reference is made to a routine number greater than 3999

(ii) if a routine parameter Aa is encountered with a >3999

(iii) if a global parameter Ga is encountered with a >3999

(iv) if an attempt is made to set a preset parameter in the range P110 - P119, unless it is one of the Special Preset Parameters, to set a preset parameter Pa with a >119, or to use a preset parameter Pa, with a >129

(v) if a reference is made to a library routine number greater than 1999

(vi) if a reference is made to a copy number of a library routine greater than 1999

NOT IN LIBRARY

One of the forms L, La, La.b, E, ER, RLc appears within a library routine. The monitoring occurs as the library routine is compiled, but this monitor should only affect library routine writers or private library routine users.

EXCESS COMMA

Two or more commas or a comma after Multiple Space appear between items or parts of an instruction on a line. A comma at the beginning of a line will give rise to a WRONG FORMAT monitor. For the purpose of this fault, comma is not the same as multiple space; thus, for example, several TABS are perfectly permissible between items.

LABEL NOT ALLOWED
For example, before an R or a T directive.

? This occurs whenever the Supervisor, as opposed to the ABL compiler, monitors any aspect of the program. The most common cases are:

Exponent Overflow - (in any arithmetic arising from the form of any items)

Input Not Defined - (after a use of the 'P115=expression' directive)

Output Not Defined - (after the use of the 'Ta' or 'Ta-b' directive)

Input Ended.

The normal Supervisor fault printing comes first followed by an ABL '?' monitor giving 'where' and the reconstructed line in the usual form.

This kind of monitoring can only occur once, because of the Supervisor's system of private monitoring. If a second error is caught by the Supervisor, only the Supervisor printing will appear and the job will be terminated.

If this is the first fault in the program, then the line 'ABL MONITORING' will appear after the Supervisor monitor printing but before the '?' monitor printing.

The monitor Input Ended may be due to a variety of causes. The most common of these are:

(i) Unmatched [. In this case the compiler will scan through the whole of the program looking for a matching] until it hits Input Ended. This circumstance can be most easily identified by the Line Count part of 'where' in the accompanying ABL monitoring. The compiler does not recognize labels whilst within a [] sequence, but does count lines in the normal manner, so the line count will be from the last label before the [.

(ii) Un-terminated Enter Directive. This is tricky to spot because the reconstructed line looks normal. In order to avoid this monitor the ***Z or other document terminator must not be on the same line as the final Enter directive, since ABL inputs and reconstructs the whole record before attempting to identify items.

(iii) Enter Directive omitted completely.

(iv) Un-terminated private Library Routine (i.e. the 'ZL' record omitted or punched incorrectly). In this case the whole of the remainder of the program is thought by ABL to be part of the Library Routine.

<Parameter > ALREADY SET AT <where>

An attempt is made to set a parameter (Global or Routine) which has already been set. The monitor here consists of one line only; no 'reconstructed line' is printed. An 'RO' directive will also cause this fault. The parameter will retain its original value.

EXPRESSION INDETERMINATE

An expression, which has not been rejected for any of the above reasons, cannot be fully evaluated when it should be. For example, if it is the right-hand side of a 'P=expression' or '*=expression' directive or an Enter directive, then it needs to be evaluated immediately, and if it cannot be (for example, because of unset parameters), then this monitoring will occur when the item is encountered - in the case of E or ER directives after any required Library Routines have been compiled; or, if it is the address part of an instruction, or a Half-word or Six-bit word, then if it is still not determinate when the next E or ER directive is encountered - after any required library routines have been compiled - then this monitoring will occur when the Enter Directive is encountered, after any monitoring arising from any library routines. For the purpose of counting errors, all parameters unset at the Enter directive are treated as one fault.

The second line of this monitor printing consists not of the complete reconstructed line in which the offending expression occurred, but only of the expression itself, and this is not normally in its original form but has been partly evaluated, including the replacement of all set parameters by their values.

Because of this system of printing the expression, with known parameter values fully substituted, the fact that, for example, the parameter A4/2 appears in the monitor printing indicates that it is that parameter which is unset and is causing the expression to be indeterminate.

A reference in an expression to a parameter of a non-existent library routine will give rise to EXPRESSION INDETERMINATE monitor printing when the next E or ER directive is encountered as well as a monitor printing for 'Library Routine NONEXISTENT' (see Chapter 12).

An attempt to divide by zero in an expression gives rise to EXPRESSION INDETERMINATE monitor printing rather than DIVISION OVERFLOW Supervisor monitor printing, for example, if expressions such as A5Q0 or A5QA2 where A2=0 are encountered.

Example:

A program being compiled attempts to print a title on an undefined output stream, causing fault printing by both Supervisor and compiler. Compiling continues, and further faults are found until the error density is too great.

```
OUTPUT NOT DEFINED
INSTR  787975   122,63 ,24 ,-1048576  B63 =-1048575  B24 = 791293
INSTR  787976  1060,0  ,61 , 16       B61 =-6
INSTR  787977   300,0  ,0  , 788005
B1  = 512        B3  =-127       B4  = 0.1        B7  = 787202
B8  = 787372.4   B9  = 257.2     B10 = 2.4        B12 =-126
B14 =-0.1        B15 = 786432    B16 = 917373.4   B17 = 917373.4
B18 = 917370.4   B20 = 911.4     B24 = 791293     B28 =-1048576
B45 = 2.1        B61 =-6         B63 =-1048575    B70 = 787583
B71 = 917371     B72 =-1         B73 = 132.4      B78 = 128.1
B79 = 917476.4   B81 = 917372.4  B88 = 917373.4   B90 = 787986
B91 = 12.2       B92 = 786740.2  B93 = 786740.2   B94 = 10
B96 =-1044754    B97 =-130560.1  B98 = 262656.5
```

ABL MONITORING

```
1,2 A0/0 ?
T10

1,9 A0/0 SHIFT >23 PLACES
H1U24

0,10 A0/0 * OUT OF RANGE
121,2,0,A5680

4,10 A0/0 PARAMETER OR ROUTINE NO. TOO BIG
121,2,0,A5680

1,11 A0/0 EXPRESSION INDETERMINATE
P1OM7

4,13 A0/0 INSTRUCTION?
121,0,0,0,0

0,14 A0/0 IRREGULAR FUNCTION
123456βP01UY

1,15 A0/0 WRONG FORMAT
F,K1,+,+4(-1,-1β,+3:0

3,15 A0/0 WRONG FORMAT
F,K1,+,+4(-1,-1β,+3:0

TOO MANY ERRORS
```

Chapter 12

FURTHER FACILITIES AND TECHNIQUES

For most purposes, the information given in the earlier chapters is sufficient to allow adequate and efficient programs to be written. Occasionally, however, it may be possible to increase the efficiency of program writing or execution; the following sections describe how this may be effected.

12.1 Programmed Drum Transfers

In the great majority of programs, the user will wish to take advantage of the one level store concept and will regard the core store and drums as a single, large main store. Programs are written as if the entire store were core store, and the Supervisor will automatically control the transfer of 512-word blocks between the drums and the core store as needed.

However, circumstances can arise in which it is useful to exercise some degree of control over these block transfers, both to ensure that blocks of information are already available in the core store when required, and to clear space in the core store by releasing blocks to the drums as soon as they are no longer needed; the extracodes provided for these purposes are all designed to assist towards greater economy of time by the avoidance of unnecessary Supervisor drum transfers. To understand just how the programmer may assist the Supervisor, it is necessary to consider the means provided for the regulation of automatic drum transfers.

In addition to any store location explicit in each instruction, there is implicit a store reference to the location containing the instruction word itself; in either case, the address is taken to specify both a block and a word within that block. The block address is invariably interpreted as a store request, and the Supervisor will initiate a drum transfer if the block is not already in the core store. Normally there will be only one copy of a particular block, occupying either a page in the core store or a sector on one of the drums. With each 512-word page of the core store there is associated a Page Address Register (P.A.R.) containing the number of the block occupying the page at any particular time; there is also a lock-out digit which is set whenever the page is involved in a drum or peripheral transfer and so is not available to the main program. At every store request, the block address is automatically compared with the content of each P.A.R.; if a coincidence is found, the store reference is completed by the extraction of the required word from the appropriate page. Otherwise a non-equivalence interrupt occurs and the Supervisor drum transfer program is entered; this is in two parts, one to carry out the actual block transfers and the other to decide which page of information should next be transferred to a drum to make space available in the core store.

All requests for information transfers between the core and the drum stores, whether originated by the Supervisor or called for directly by an object program, are placed in a drum queue holding up to 64 entries which are dealt with in the order of their occurrence. The drum transfer routine is re-entered repeatedly until the queue is cleared.

It is arranged that there shall always be at least one free page in the core store, so that, whenever the drum transfer routine is entered, the first 'read' request in the drum queue can be implemented. Then, whilst this transfer is taking place, the drum transfer learning program decides which page may next be freed by writing its contents away to a drum. This decision is made on the basis of the frequency of past references to each block of information, and with the intention of choosing the core store page least likely to be referred to.

The drum learning program only attempts to predict future store needs in the light of past requests; it anticipates neither the termination of references to a particular block of information nor the imminent requirement for a new block. Hence, there arise in the main two ways of assisting the Supervisor by means of programmed drum transfers; firstly by releasing core store pages no longer required, and secondly by initiating the reading of a new block of information from a drum to the core store before it is actually referred to. These are both of marked advantage to the system as a whole: the first plainly helps towards efficient utilisation of the available facilities, and the second can often prove of even greater benefit and economy by reducing the time spent in waiting for drum transfers to be completed - this is especially significant in the inner loops of a program. It will be appreciated that the time during which a program is held up waiting for a drum transfer is still wasteful, notwithstanding time-sharing, since it may take from 1.3 to 2.7 milliseconds, depending on the core store size, to switch to another program and a similar length of time to switch back later.

Ideally, a 'read' transfer should be timed to reach completion only just before the first reference is made to the block; otherwise the Supervisor may choose to write the block away to the drum again before the program comes to use it. The actual transfer of a block of 512 words takes 2 milliseconds, but there is an initial delay of up to 12 milliseconds, the revolution-time of the drum.

The core store of Atlas is arranged in 4096 word stacks, with 16 pages of information sharing each pair of stacks, and each stack having its own access equipment; to take advantage of this, and so to attain maximum speed, operands and instructions should be arranged, as far as is possible, in different pairs of stacks. At Manchester, the Supervisor endeavours to read down instructions to pages 0 to 15 and operands to the remaining pages of the machine; in the event of a non-equivalence interrupt the preference is automatic, being determined by the non-availability in the core store of an operand or an instruction as the case may be; in the case of a programmed drum transfer, the preference must be indicated by affixing a bit 1 before the address of a block of instructions, and 0 before the address of a block of operands. This preference bit will be the most significant bit of n (singly modified) or of ba where appropriate. At the other installations, where there is more store, instructions and operands are placed in different pairs of stacks whenever possible. Pages 0-15 form one stack pair, as do pages 15-31, 32-47, etc.

Extracodes are available for the purposes we have discussed and these will now be described with the help of the following notation:

Block address,	$P = P_1$ = bits 1 to 11 of n
Block address,	P_2 = bits 1 to 11 of ba
Number of blocks,	K = bits 21 to 23 of n
Logical band number,	D = bits 13 to 20 of n
Band or page number,	d = bits 13 to 20 of ba
Section number,	k = bits 21 to 23 of ba

In all but two of the extracodes which follow, whenever information is transferred to a new block, the old block is made free. The exceptions are 1162 and 1163, where a block is to be duplicated leaving the original copy intact.

Further, when a block is quoted as the destination of an information transfer, either directly or as the result of renaming, any existing block of the same name is lost. This will apply even if the name quoted as that of the source is unallocated, and will in fact be the only action taken in such a case. Also, in those instructions referring to two block addresses, these addresses should not be the same.

1135 b91′ = c and c′ = n if block number ≥ ba newly defined.
Henceforth, each time a block with a number ≥ ba is newly defined by a non-equivalence, store current control in B91 and jump to n. The block number in ba occupies bits 1 to 11 and the remaining bits of ba are ignored. The contents of Ba are undisturbed. The instruction causing the non-equivalence is not executed. n is singly modified.

1155 ba′ = smallest block label ≥ n defined.
Place in bits 1 to 11 of ba the smallest block number ≥ n which is defined for this progr m. The remaining bits of ba are left cleared. Only bits 1 to 11 of n are used. n is singly modified. If all the program's blocks are < n, then bit 0 of ba′ is made 1 and the remaining bits are cleared.

1160 Read block P.
If P is not already in the core store, the transfer request is inserted in the drum queue exactly as if a non-equivalence interrupt had occurred, but control is restored to the object program immediately the drum queue entry has been made. Should the queue be already full, the object program will be halted until the entry can be inserted.

1161 Release block P from the core store.
This extracode adjusts the parameters used by the drum learning program so as to cause it to choose block P next for writing away to the drum store, if this has not already occurred. No entry is made in the drum queue and the transfer will in general take place earlier than if extracode 1165 (below) had been used.

1162 Duplicate block P_1 as P_2 in the core store.
Any existing block P_2 is always lost, and, if P_1 is allocated,

a copy of it will be formed as P_2 in the core store. Unless the drum store is full, block P_1 will finally be located there; otherwise P_1 will be left in the core store. $P_1 \neq P_2$.

1163 Duplicate block P_1 as P_2 in the drum store.
Provided P_1 is allocated, the effect of this extracode is to form a duplicate copy of it. It will be arranged that one copy shall always be left in the core store and named P_1; the second copy, named P_2, will be put in the drum store unless this is full, in which case it will be left in the core store. Any previously existing block P_2 will be lost in all cases. $P_1 \neq P_2$.

1164 Rename block P_1 as P_2.
If P_1 is allocated, the appropriate entry in the drum directory or the core store P.A.R. is altered to P_2. Any P_2 previously existing will be lost. There must be at least one more block allowed for in the job description than is defined at that moment. $P_1 \neq P_2$.

1165 Write block P.
Provided P is allocated and is not already on a drum, it is transferred to the next empty sector. Should the drum store be full, block P is released, precisely as in 1161.

1166 Read block P to absolute page d.
This extracode makes possible full control of the store by those exceptional programs for which this may be worthwhile. Before using 1166, the program must set a trap in case page d is locked down and reserved by the Supervisor.
d is in the integer position of ba and defines the absolute number of a page in the core store to which block P is to be transferred. Before this transfer takes place, any existing contents of d are copied to a free page.

1167 Lose block P.
If P is allocated, the page or sector occupied by it is made free.

1170 Clear new blocks/Do not clear new blocks.
When a program refers to a main store block for the first time, the Supervisor allocates a free page of the core store; floating-point zero will be written in all 512 words if the clear blocks switch is set. Initially, this switch is set to clear all new blocks, but it may subsequently be set or reset by means of extracode 1170 according to the sign bit of n:-

$n \geq 0$ Clear new blocks.

$n < 0$ Do not clear new blocks.

1171 Change store allocation to n blocks.
Each program has some number of main store blocks assigned to it. This number may be altered during the execution of the program by the use of extracode 1171. If there are less

than n blocks available in the store, then the program will be faulted for ILLEGAL FUNCTION and EXCESS BLOCKS.

1172 Set ba' = number of pages available.
This extracode provides an estimate of the number of core store pages available to the program at a particular moment. It cannot be assumed that this number of pages will continue to be available, since the core store allocations are always fluctuating.

1173 Set ba' = number of blocks available.
At a particular moment, this extracode records the maximum number of main store blocks available, consisting of all unallocated blocks together with those already allocated to the program itself.

1174 Reserve band D.
A complete band of the drum store is reserved for the program and may subsequently be referred to as band D.

1175 Read K + 1 blocks from band d, starting at sector k.
d must already be defined by extracode 1174. The K + 1 successive sectors k, k + 1,..., k + K are read to store blocks P, P + 1,..., P + K. If K is 6 (or 7), sectors k (or k and k + 1) will be read twice. Thus if K = 6, blocks P and P + 6 will both contain sector k. If k is 6 or 7, it is taken as 0 or 1 respectively. All blocks involved are locked down until the entire transfer is complete.

1176 Write K + 1 blocks to drum band d starting at sector k.
d must already be defined by extracode 1174. This extracode writes store blocks P, P + 1..., P + K to drum sectors k, k + 1,..., k + K. Sectors 6 and 7 are the same as sectors 0 and 1. If K exceeds 5 some of the earlier blocks are overwritten. Thus if K = 6, sector k will finally contain block P + K rather than block P.

1177 Lose band D.
The band of the drum store previously reserved as logical band D is freed and made available for general use.

12.2 Optimization of Program Loops

The following table gives the approximate times in microseconds achieved on Atlas for various instructions. The figures are averages for obeying long sequences of each instruction, with the instructions and operands in different stacks of the core store.

Type of Instruction	Number of Address Modifications	Time
$am' = am + s$	0	1.6
	1	1.9
	2	2.4
$am' = am \times s$	0, 1 or 2	5.9
$am' = am/s$	0, 1 or 2	22.5
$ba' = ba + s$	0	1.7
	1	2.0

It is not possible to time a single instruction because, in general, this is dependent on

(a) the exact location of the instruction and operand in the store;
(b) the instructions preceding and following; for whenever possible one instruction is overlapped in time with some part of three other instructions,
(c) whether the operand address has to be modified,
(d) for floating-point instructions, the numbers themselves.

This is illustrated when evaluating a polynomial, using a central loop involving a singly modified accumulator addition, an accumulator multiplication, and a test, count and jump instruction. The average time for this loop is 8.3 μsec, of which the accumulator operations would take 7.8 μsec if their individual times are simply added. This leaves only 0.5 μsec for the test, count and jump, although the average figure for a series of jump instructions on their own might be ten times as large.

We shall consider those factors which control the time taken to obey instructions, to show what advantage can be taken of them in optimizing a program loop which has to be executed many times.

12.2.1 Store Access

The main core store consists of pairs of 4096-word stacks. Each stack can be regarded as a physically independent store, and sequential address positions occur in the two stacks of a pair alternately, the even addresses in one stack, the odd in the other. The cycle time of the core store is 2 μs., that is, the time after reading or writing a number before another number can be read from or written to the same stack is 2 μs.

To reduce the effective access time, instructions are always read in pairs and held in two buffer registers called Present Instruction Even (PIE) and Present Instruction Odd (PIO) whilst waiting to be obeyed. Instructions are executed from PIE, the odd instruction being copied from

PIO into PIE as soon as the even instruction has been initiated.

Because of the 2μs cycle time for each stack the programmer should separate instructions which refer to operands in the same stack.

For example, the instructions

121	1	0	0
324	0	0	A4
362	0	0	A4

would be executed more quickly if written as

324	0	0	A4
121	1	0	0
362	0	0	A4

The maximum overlap is obtained when alternate operands come from alternate stacks.

The Supervisor attempts to organise the store so that instructions and operands are placed in different pairs of stacks. On the Manchester University Atlas, wherever possible, instructions are kept in pages 0-15 and operands in pages 16-31. The programmer can assist the Supervisor to do this by using the extracodes described in 12.1. These are of most use for jobs with a large amount of data. It is then useful to request drum transfers in anticipation, and to release from the core store blocks which will not be wanted again for some time.

12.2.2 The overlapping of Instructions

Instructions on Atlas are overlapped as far as possible. For example, in a sequence of singly-modified accumulator instructions, the computer is obeying four instructions for one quarter of the time, two instructions for one fifth of the time, and three instructions for the remainder of the time.

This overlapping is possible because the accumulator arithmetic, the B-register arithmetic, the function decoding, the B-store, and the main core store are independent of each other to a large extent. A number of rules which enable the programmer to gain as much advantage as possible from the overlapping are given below. It should be noted that these rules cannot always be guaranteed to establish the best way of arranging any particular loop, as in some cases this can only be done by actually running the program; nevertheless the application of these rules, as far as possible, will normally lead to a time reasonably close to the optimum being obtained.

(a) Instructions writing to the main store (usually referred to as store-write instructions) should normally be in odd-numbered locations.

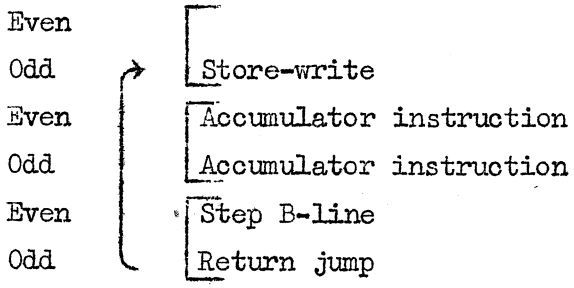
(b) In general, B-type instructions can be obeyed whilst accumulator operations (other than store-write instructions) are going on. Only one accumulator operation can be queued up whilst a previous one (e.g. a division) is proceeding. If a third accumu-

lator operation is encountered, nothing further can be done until the first one is finished. This third accumulator operation should therefore be delayed until all B-instructions and B-tests which can be obeyed before the first accumulator instruction is completed, have been initiated.

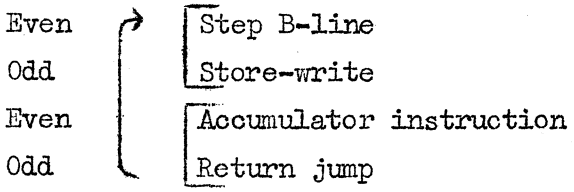
(c) Following a store-write instruction, no further instructions or operands can be extracted until the writing operation is completed. Many typical program loops, however, include such an instruction. It is usually possible to have this instruction at the beginning of the loop, and this enables the B-type instruction and return jump to be obeyed and overlap any accumulator arithmetic still going on. As mentioned in (a) above, the store-write instruction should preferably be in an odd-numbered address. From these two rules, two possible ways of arranging a loop, depending on whether it has an odd or an even number of instructions, emerge.

Examples:

1. Odd number of instructions



2. Even number of instructions



(d) Instructions are extracted from the store in pairs, and, subject to the above rules, a loop with an even number of instructions should begin at an even address, so as to minimise the number of store references.

(e) Test instructions cause more delay when successful than when unsuccessful, and it is usually best to arrange the uncommon case (if it can be determined) to be the one which changes ba.

(f) Jump instructions where the jump will frequently not take place, should preferably be placed in an even-numbered address. Note that this does not apply to return jumps in loops, as these fail to jump only when control leaves the loop.

(g) Singly-modified A-type instructions should always be modified by bm, not ba.

(h) A delay occurs if a B-register is operated on in the Ba position and then used as a modifier in the next instruction. This should therefore be avoided if possible e.g. by inserting some other

instruction in between. Note, however, that

```
124   1   0   1
300   1   2   0
```

is preferable to

```
124   1   0   1
300   2   1   0
```

(i) Given a pair of accumulator instructions, one modified and one not, the unmodified one should occur in the even-address, and the modified one in the odd-address, if possible.

(j) Given an accumulator operation and a B-register operation as an even/odd pair, they should be in this order if possible.

Where the above rules conflict, the order in which they are given should be taken as the order of importance.

12.3 Branching

Branching is a facility which enables different parts of the same program to operate in parallel, using the time-sharing process. Such parallel operation is of value if some parts of the program are liable to be held up waiting for peripheral transfers whilst other parts are still able to proceed. It is important to note that simple operation of peripheral devices in parallel with computing is available without recourse to branching; normally, the program itself is only held up if it attempts to refer to the locations involved in a transfer before the transfer has been completed. Branching is an additional facility which is intended to permit parallel operation of two or more different processes which are liable to be held up by peripheral transfers, where each process involves some computation or organization and does not consist merely of peripheral transfers.

12.3.1 Existing Parallel Operations

When a block transfer to or from a drum or magnetic tape has been initiated, by means of a drum or tape block transfer extracode, the program is allowed to proceed as long as it does not refer to the main store block involved in the transfer. If it does refer to that block, it is held up until the transfer has been completed.

Variable length tape transfers operate by using part of the main store as a buffer. It is usually possible to keep sufficient information in the buffer to permit the actual transfer, between the buffer and the specified store address, to take place as soon as the transfer instruction is encountered. Otherwise the program will be held up until the transfer is complete.

Other peripheral devices, apart from the drums and magnetic tapes, are not normally controlled directly by the program. Instead, the input documents are read and stored on a system magnetic tape before the program is initiated, and output documents are stored on a system tape and printed after the program has been completed.

12.3.2 The Branch Instructions

1103 Permit Ba Branches ($2 \leq Ba \leq 32$)

Before any branching can take place, the program must obey an 1103 instruction, which enables the Supervisor to prepare for branching.

This instruction normally takes the form

1103 Ba 0 pD1

After obeying it, the program is permitted to have up to Ba live branches, including the main program, in progress at any one time; the main program is defined as branch 0. When the Supervisor switches from one branch to another it will preserve certain standard information and also the contents of index registers Bp, B(p + 1), B(p + 2),....., B99. Note that if the N-address is zero all index registers are preserved, and if p = 91 only the extracode index-registers are preserved (these are usually essential).

1104 Start Branch Ba at n ($0 \leq Ba \leq 63$)
The current branch of the program continues at the next instruction, but a new branch, with number and priority Ba, is started at address n. The highest priority is given to the highest-numbered branch: if other branches with the same number Ba have been defined previously, they will take higher priority than the new branch. The main program is initially defined as branch number 0.

1105 Kill Branch Ba or Current Branch
Kill all branches with the number Ba. If Ba = 64, kill the current branch. This prevents any further instructions being obeyed in the specified branches, but peripheral transfers already requested will be completed.

1106 Wait until Branch Ba is Dead.
Halt the current branch of the program if any branch numbered Ba is still live. Proceed to the next instruction when all branches numbered Ba are dead.

1107 Jump if Branch Ba Live.
Transfer control to address n if any branch numbered Ba is still live. Otherwise proceed to the next instruction.

12.3.3 The Use of Branching

A branch is usually started at some point in a program where it is required to carry out two different processes, at least one of which is liable to be held up by peripheral transfers. Usually, the more severely peripheral limited process is put in the new branch, and this is given higher priority. When the program is obeyed, the higher priority branch is allowed to proceed until it is held up waiting for a peripheral transfer; control is then transferred to the other branch, which proceeds either until it is held up, or until the higher priority branch is ready to resume. Similarly, if there are several branches, the Supervisor ensures that control always passes to the highest-priority branch able to proceed. Each time control is switched from one branch to another, the Supervisor stores and restores the contents of the following registers and indicators:

The Accumulator

B119, B121, B124, B126 and B127.

The Index Registers specified in the 1103 instruction.

The Selected Magnetic Tape Number.

B-Test, B-Carry, Accumulator Overflow (V-store Line 6).

Extracode Working-Space.

Thus, each branch can use these registers as though it were one single program uninterrupted by other branches. It is, however, necessary to ensure that two branches which may operate simultaneously do not use the same main store locations, or index registers which are not preserved. It should be noted that the selected Input and Output are not preserved, and therefore input and output can each take place in only one branch at a time.

Once a branch has been started it can be regarded as a 'live' branch, and it remains live, even when it is held up, until its task has been completed. When a branch has completed its task, it must die, and this it does by obeying an 1105 instruction, usually with Ba = 64.

When one branch of a program is ready to make use of the work done by another, it must first ensure that the work has been completed. This may be done by obeying an 1106 instruction, which causes the current branch of the program to be held up until the specified branch is dead, having completed its task.

A simple example of the need for branching arises when it is required to scan a magnetic tape in order to process a selected sample of the information on it. The processing routine and the tape-scanning routine can then be written as two separate branches, with the tape-scanning routine as the higher-priority branch.

Example:

It is required to scan sections 1 to 3000 of tape 4 and to apply a lengthy processing routine R3 to the information in about 25% of these sections. The sections to be processed are to be identified by having a number greater than 0.32 in the first word of the section. The program to do this could be written as follows:

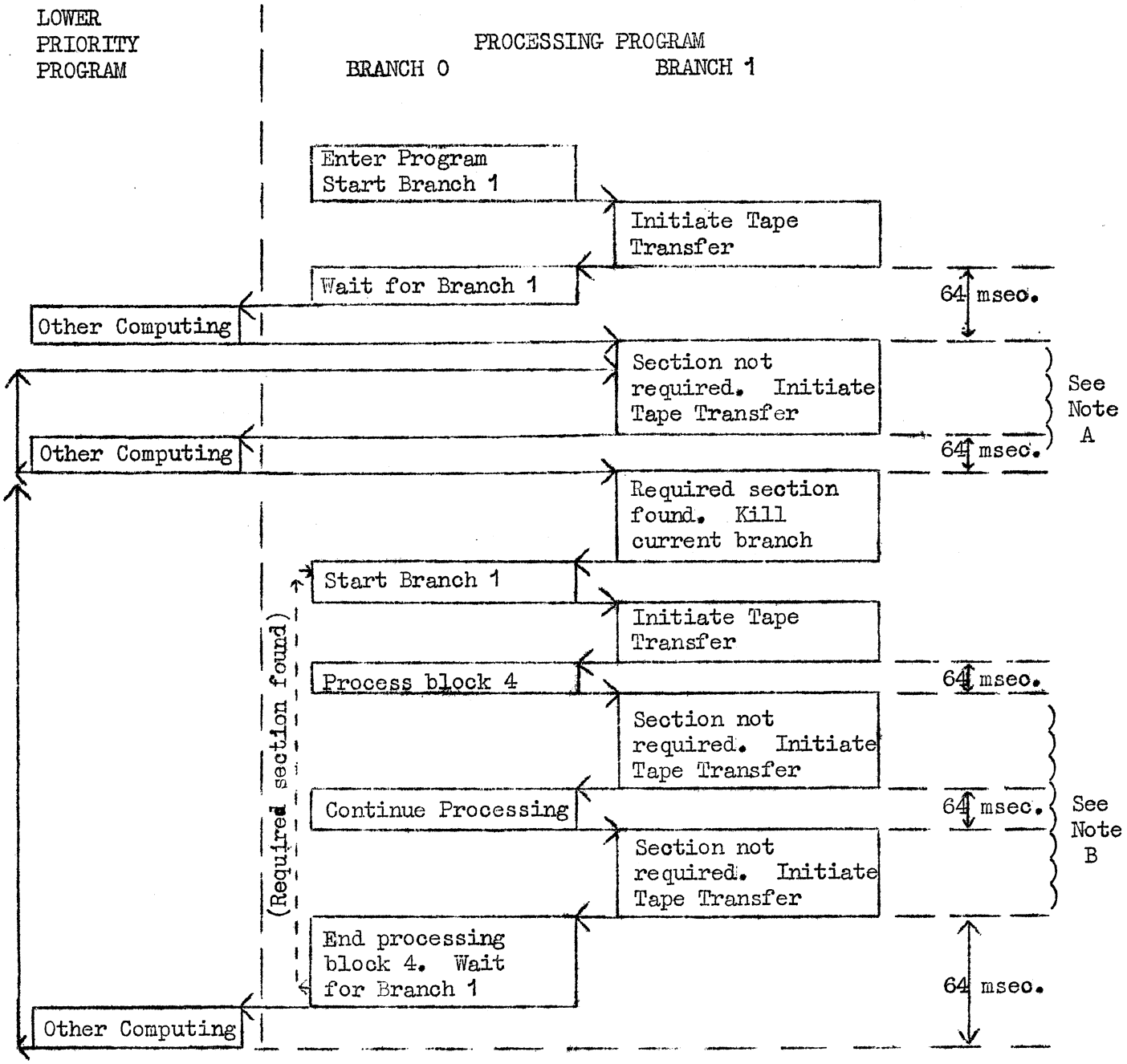
					Branch 0
	1103	2	0	89D1	Prepare to use 2 branches, preserving b89 - b99.
	1001	4	0	1	Search for section 1, tape 4
	121	89	0	2999	Set count for 3000 sections
	121	16	0	0	Clear marker in B16
	1104	1	0	A6	Start branch 1 at A6
5)	1106	1	0	0	Wait until branch 1 dead
	215	127	16	A12	Exit if last section processed
	121	10	0	4:	Rename block 3 as block 4
	1164	10	0	3:	
	1104	1	0	A7	Start branch 1 at A7
	121	90	0	A5	Set Link for return
	121	127	0	A1/3	Enter R3 to process block 4
					Branch 1
6)	1002	4	0	3:	Next section to block 3
	324	0	0	3:	First number in section
	321	0	0	1A8	Subtract 0.32
	236	127	0	A8	Exit if number $\geq$ 0.32
7)	203	127	89	A6	Count tape sections
	121	16	0	7	Mark b16 non-zero

(Program continues on following page)

8)	1105	64	0	0	Kill current branch
	+0.32				
	R3				R3 (Part of Branch 0)
1)	.	.	.	.	Routine to process the information in block 4
	.	.	.	.	
	.	.	.	.	

The chart below shows how control would pass from one branch to the other in a typical sequence of operations when the program is obeyed. Interruptions from the Supervisor and higher-priority programs have been excluded because they would complicate the chart without altering the sequence significantly. The sequence of operations starts at the top with the beginning of the program, runs through the first entry to branch 1, and then cycles round a loop in which branches 0 and 1 operate in parallel. The chart shows branch 0 completing its work before branch 1 has found the next section to be processed, but branch 1 might equally well be finished first, that is, a further required section may be found during processing. It should be remembered that each entry to branch 1 takes only a few microseconds, whereas 64 milliseconds must elapse between successive entries to branch 1 to read one more tape section.

See Chart on following page.



Notes:

A. This loop will be repeated until a required section is found.

B. If a required section is found, then Branch 1 will be killed. When the current block has been processed, Branch 0 will start Branch 1 again, and then process the required section.

12.3.4 Store Requirements

When an 1103 instruction is obeyed, the Supervisor assigns sufficient storage space for the specified number of branches. This storage space is taken out of the main store allocated to the program, either by the job description or by a subsequent use of the extracode 1171, and will be counted in the estimates made by extracodes 1172 and 1173; it is therefore necessary for the programmer to know how much store is required by the Supervisor for branching purposes. Many cases should be covered by the following table, showing the maximum number of branches that can be accommodated in 1, 2 or 3 blocks, depending on the number of index registers preserved.

Index Registers Preserved	Storage Space Allocated		
	1 Block	2 Blocks	3 Blocks
	Maximum Number of Branches Permitted		
B0 to 99	3	10	18
B30 to 99	4	14	24
B50 to 99	5	17	29
B70 to 99	6	22	32
B80 to 99	8	27	32
B90 to 99	10	32	-

If it is necessary to estimate the store required in some case not covered by the above table, it is probably easiest to do this by considering the way in which the Supervisor allocates this store. It takes 300 words at the beginning of the first block to store branching routines, and follows these by 5 words for each branch requested in the 1103 instruction. Each branch is then allocated a further $(11 + \frac{1}{2}m)$ words, where m is the number of index registers in the range 0 to 99 that are to be preserved. The $(11 + \frac{1}{2}m)$ words for one branch must all be in the same block, and if less than $(11 + \frac{1}{2}m)$ words are left at the end of a block the $(11 + \frac{1}{2}m)$ words for the next branch will start at the beginning of a new block.

12.4 Instruction Counters

As each basic instruction is obeyed, an instruction counter is stepped on, normally by one, but by two for multiplication orders, and by four for division. Each time the counter reaches 2048, an interrupt occurs, and an instruction interrupt counter is stepped on by one. This latter counter is used by the Supervisor in monitoring the program, but may also be read by the program using extracode 1136.

1136 Read instruction count.
Set am′ to the number of instructions obeyed from the start of the program; this will be a fixed-point integer with exponent 16, and will be a multiple of 2048.

Besides this count, the program may also use a local instruction counter. A trappable fault will be recognized when this count expires, which may provide a convenient way to end an iterative loop, since the counter may be set as well as read by program.

1123 Set local timer.
Set the local instruction counter to n x 2048 instructions. The Supervisor will override any attempt to set the counter to a figure in excess of the amount of allotted time remaining.
local timer′ = 2048n

1122 Read local timer.
Read local instruction counter into Ba in units of 2048 instructions.
ba′ = local timer

12.5 Re-entering the Compiler

Most programs are compiled completely before they are entered, and therefore it is not normally necessary to retain the compiler in store during the program's execution. The E-type of directive is the only enter directive which deletes the compiler from the store before transferring control to the object program, and so is the most commonly used.

In some circumstances, however, it is necessary to enter the program, and then compile more program later. The first entry may be to actually execute part of the object program, or it may be only to set certain parameters. The compiler must be retained in store for these purposes, and so either an ER or EX type of enter directive must be used. The compiler uses store locations J3 ($3/4 \times 2^{20}$) and above which should not normally be altered by the program, although no check is made except when actually compiling.

The EX-directive is intended for obeying 'interludes' during compiling; an interlude would normally consist of a few instructions only, or of none at all. For example, if it were required to have any ABL fault printing on some output stream other than Output 0, then a one instruction interlude to 'select output' would suffice. If it is only required to set parameters, then the address specified in the EX-directive should cause immediate re-entry to the compiler; such a directive in fact occurs near the beginning of L100, the general input routine, to determine the various optional parameter settings. The EX-directive does not call down any library routines; if these are required in the interlude, they must be called by one of the L-directives before obeying the EX entry. No distinction is made by the Supervisor between compiling proper and obeying an interlude, i.e. the 'Compile/Execute' switch is not changed.

The ER-directive is designed to allow part of a program to be compiled and executed before reading more program, and provides most of the facilities of an E-directive, including the compilation of any library routines mentioned but not called earlier in the program. The routine current when the ER-directive is obeyed will be terminated before more program is read. (The EX-directive does not do this.) The Supervisor recognises that an object program is being executed, and as with the E-type of directive, the 'Compile/Execute' switch is set to 'Execute'.

Two types of list within the compiler are used in connection with parameters in a program. The parameter lists contain all those parameters which are determinate, and if the program refers to a set parameter, these lists are used to replace the parameter by its value. If a program refers to a routine or global parameter before it has been set, then this is noted on a forward reference list, from which it is deleted when the parameter is determinate, and hence, so long as this list is not empty, there are some parameters still to be set. When the compiler is retained in store, these lists also remain, in the same state as when the enter directive was obeyed. If more program is to be read which uses parameters to refer back to the program compiled previously, then it is essential that the lists remain unaltered. If, however, the subsequent sections of program are to be compiled independently of the earlier part, or if the same parameters are to be used again with different values, then the lists must be cleared on re-entry to the compiler. Different re-entry points provide for both requirements, and are listed below. In every case, re-entry to the compiler does

not alter the 'Compile/Execute' switch. After compiling program, B1-B88 will be cleared, and B89 will contain the final transfer address. The other B-lines may be destroyed.

P120 When the compiler is re-entered at address P120, all parameters, forward references after EX, and * (the transfer address) are left unchanged, and more program is read from the current input stream. If there is no *-directive before the first items are read, these will be placed in store sequentially in the usual way, after the last item of program before the previous enter directive. After an ER-directive, library routines may have been compiled into locations beyond the end of the written program.

P120B Re-entry at this address causes the compiler to behave as it does when first called by the Supervisor, but the contents of the store below J3 are left undisturbed. Hence * = 1:0, and the forward reference list and parameter lists are cleared.

The compiler may also be used as a subroutine by a program, control returning to the main program when no more items are to be read. Again there are two modes of entry, depending on what is required of the compiler lists.

If the transfer address is written to location Y4P121, and the link to 129P121, then re-entry to the compiler at P120 will read more program, retaining the compiler lists. Return to the link address in the main program is effected by

EP129
ERP129
or EXP129

If the transfer address is written to B89, and the link to B90, when the compiler is re-entered at P120BY6, more program will be read as if the re-entry were to P120B, except that an attempt to compile into store at an address less than b89 will be faulted, and that the transfer address will be taken as * = b89 unless b89 = 0, when * = 1:0. Return to the main program is again by EP129 etc.

P120, P121, and P129 are examples of special preset parameters, which are described in the next section.

12.6 Special Preset Parameters

Although for normal purposes only Preset Parameters 0 to 99 may be used, some above 100 do exist and these are used in special ways for special purposes. In some cases use of them causes special action by the compiler; in other cases they are used to convey information between the compiler and the program.

P100 to P109 are in many ways like ordinary Preset Parameters; they can be reset by the programmer and no special action is taken by the compiler on encountering them. However, they are initially set by the compiler at the start of compilation and they are referred to by the compiler during the course of compilation.

P110 to P119, if defined, may be reset by the programmer but will have initial values set for them by the compiler. However, whenever an Equation Directive for resetting them is encountered, special action is required by the compiler. An attempt to set one of these parameters not listed below is faulted.

P120 to P129 are preset by the compiler, but may not be reset by program. An attempt to do so is faulted. They are used to convey information from the compiler to the program.

P100 - Optional Printing

At the start of compiling ABL sets P100 to zero. Non-zero settings of P100 cause ABL to print various kinds of information during compiling.

P100 is treated by ABL as made up of 8 octal digits abcdefgh. Each octal digit controls the printing of one kind of information, as indicated.

If the least significant bit of an octal digit is 1 the information controlled by this digit will be printed - on a new line if the middle bit of the digit is 1 and on the same line if the middle bit is 0. If the least significant bit is 0, then the other two bits are ignored. If the most significant bit of the second digit (b) is 1, then printing on a new line will occur when a library routine is compiled. Otherwise the most significant bits of the digits are ignored.

P100 is preserved and set to zero before compilation of each library routine and restored afterwards, so that there will be no other optional printing, unless the library routine contains a 'P100 =' directive.

The kind of printing controlled by each octal digit is as follows. All printing is preceded by a space, except R. L is printed on a new line.

Octal digit	ABL prints this	when it meets this
a	*= p	* = expression
b	Ra * = q La.b N * = q	Ra Library routine named N compiled
c	Z * = q	Z
d	this digit is unassigned and ignored	
e	p	P 111 = expression (see P111)
f	E p	E expression
g	E R p	ER expression
h	E X p	EX expression

where p is the value of the expression met,

q is the current value of asterisk

a and b are integers.

Some examples of useful settings of P100 are

P100 = -Y1 ABL prints on R, L, *, Z and all types of E, each item on a new line

P100 = J03 ABL prints on R only

P100 = J031 ABL prints on R and Z

P101 - Permitted Number of Errors

At the start of compiling ABL sets P101 to 0.2. This is equivalent to infinity since for each error met ABL reduces P101 by 1, and when it reaches zero stops compiling and ends the run after printing

TOO MANY ERRORS

The program may set P101 = n where n is any expression. Compiling will stop when n + 1 errors have been met. P101 = 0 causes ABL to stop on the first error met, which may be useful for a developed program.

No matter how many labels remain unset when the E directive is met ABL lumps them all together as one error for the purpose of counting errors.

P102 - Entry Despite Faults

At the start of compiling ABL sets P102 = 0.2. If any errors have been found, an EX directive will be obeyed, but an E or ER will not, and ABL will print

ERRORS DO NOT ENTER

and end the run.

P102 = 0 allows all 3 E directives to be obeyed despite errors

P102 = 0.3 forbids all 3 E directives after errors

P102 = 0.1 allows E and ER but forbids EX after errors

P104 - Setting Private Monitor

P104 is the address of a private monitor routine, which is set up each time any Enter Directive is encountered. Thus if any monitors occur after the Enter Directive (including an immediate entry to an address holding an Illegal Function due to a wrong Enter Directive address), these will give rise to an entry to Private Monitor according to the rules of extracode 1112 (see section 11.3). If P104 is negative any current setting is terminated.

P110 - Change of Program Location

At the start of compiling, ABL sets P110 = 0. A non-zero setting of P110 specifies the difference between * as evaluated in expressions (say $*_1$) and * indicating where items are to be stored (say $*_2$)

i.e. $P110 = *_1 - *_2$

Setting P110≠0 permits the compiling of program into one set of store addresses - $*_2$ - for later execution in another set of store addresses - $*_1$ (e.g. after 'Renaming' or after storing on magnetic tape). Thus, for example, a program which is to be executed starting at address J3 but compiled initially into store starting at J1 would have the directives

P110 = J2

* = J3 at its start

P111 - Expression printing

When ABL meets the equation

P111 = expression

it evaluates the expression and sets P111 in the usual way. If the appropriate bit of P100 is set, the value of the expression is immediately printed out.

P112 - Unused

This parameter is used by ABL on Atlas 2. No fault will occur if the program sets P112, the value being assigned in the usual way, but the remainder of the line on which the setting occurs will be ignored. The compiler initially sets P112 to J3. P112 should not normally be used with ABL on Atlas 1.

P115 - Change of Input Stream

The equation P115 = n(n is any expression) causes ABL to start reading program from the programmer's input stream n. The rest of the line on which P115 occurs will be ignored.

P120 - Re-entry to the Compiler

This is described in the previous section.

P121 - Preset Parameter List

P121 is the address of the start of the compiler's Preset Parameter list. Halfword P121+n contains the value of parameter n if it has been set. This list can of course only be referred to by a program entered by an ER or an EX directive. This is a list of alternate halfwords, the other halfwords of which are used for other purposes by the compiler and should not be disturbed.

P122 - State of Preset Parameters

P122 is the address of the start of the compiler's list which indicates whether the Preset Parameters are set or not. Bit 1 of halfword P122+n is a 1 if parameter n is unset, 0 if set. This is a list of alternate halfwords, and neither the other bits of the halfwords, nor the other halfwords should be disturbed if subsequent use of the compiler is intended.

Example: An 'interlude' to set P17 = 0 if P35 = 0 and to leave P17 unaltered otherwise

```
5)  121     1    0   P35
    215   127    1   P120
    113     0    0   P121+17
    121     1    0   J2'
    117     1    0   P122+17
    121   127    0   P120
    EXA5
```

P123 - Characters Count with C directives

P123 indicates the number of characters read by means of the previous C directive, described in section 5.10.

P129 - Return from Compiler

P129 is the return address when the ABL compiler is used as a subroutine (see previous section).

12.7 Private Library Routines

12.7.1 Library Routine Titles

Library routines are given numbers and names; the program refers to them by number, but the name may be useful to indicate their purpose. The standard input and output routines, described in Chapter 8, have been given names as follows:

L1	GENERAL OUTPUT
L100	GENERAL INPUT
L199	LINE RECONSTRUCTION

L199 is used by L100, and hence, if the library routine is being compiled implicitly, (whether by an 'L' directive or by an Enter directive), L199 will automatically be compiled with L100 because the latter refers to it, but, if it is required to compile the 'input library' explicitly into a part of the programmer's store area, then it is necessary to use both of the directives.

L100

L199

12.7.2 Undefined Library Routines

All undefined library routines have the name NONEXISTENT and there is a special device to make the optional printing as described for P100 (section 12.6) compulsory for non-existent routines. For example, output such as

L10 NONEXISTENT * = 2:36.3

will result from either an attempt to call for L10 explicitly or when an attempt is made to compile it by an L or Enter Directive when it has been referred to implicitly. The latter would also result in monitor printing about unset labels.

Any defining of a Private library routine (see below) will cause suspension of the NONEXISTENT monitoring for that routine.

12.7.3 Preparing a Private Library Routine

Private Library Routines may be incorporated in the normal program input stream and may be referred to in the program in the same way as public library routines.

The routine is headed by two lines:

RLc
< Name >

where c is the number assigned to the library routine and < Name > is the name of the routine. If no name is required, this line must be left blank, and then a blank title will appear in any optional printing. The Name must not consist of the two-character record ZL.

The routine is terminated by the two-character record ZL. This is not line-reconstructed and may not contain any spaces, erases, backspaces, tabs etc.

The library routine may consist of a single routine (routine 0) or of one or more routines headed by routine directives and optionally terminated by Z directives in the usual way. It may contain any of the normal ABL forms except the directives

L, La, La.b, ER expression, E expression, RLc

All will be monitored. No T or C directive within the library routine may be followed by the two character record ZL.

When the RL directive is encountered, the library routine following is simply copied character by character into the compiler's store area. The routine is not, at that time, compiled or placed in the programmer's store area. This is achieved in the normal way, by an E, ER or 'L' - type of directive.

If a private library routine is given the same number as a public library routine, it replaces the public one for the remainder of that program. This is convenient for the development of new versions of existing public routines.

Private Library Routines must precede any calls for them in the body of the program. The best place for them is at the beginning of the program stream.

A private library of routines required by people working in some limited field (e.g. properties of steam) may be formed by putting the routines on to a titled paper tape (as pseudo-data) and terminating them by P115=0. The master programmer may then write, for example,

INPUT

15 STEAM LIBRARY

in his Job Description, and

P115 = 15

at the head of his program to incorporate these routines effectively as above.

Each private library routine incorporated in the way described in this section counts as one line from the point of view of line counting for error monitoring of the subsequent program.

The directive

P115 = expression

within a library routine, will not cause monitoring, although it should never normally be needed or used. Its effect is in fact to cause switching of the input stream after completing the compilation of the current library routine or routines at the point where these are compiled (i.e. at an L or Enter directive).

12.7.4 Incorporating a new Library Routine into the Public Library

This is done by means of a special job, using a standard program of the system. Essentially, the routine to be incorporated is put at the head of the program stream in the normal way, as described above, and is followed by the standard program. This program uses no labels as the non-empty parameter list would otherwise become part of the compiler.

If the routine being incorporated is a new version of an already existing routine, then this latter is not destroyed or overwritten on the compiler tape. The reference to it in the compiler's library routine list is simply changed and so it becomes 'dead'. A separate special program (or prelude to the above program) may be used from time to time to clear out all 'dead' library routines, but this clears out all live ones as well, so that after this clearing out operation all public library routines must be re-incorporated.

12.7.5 Conventions

The following conventions are recommended for library routine writers and users:-

(i) Communication of parameters and addresses for use at compile time should be by routine parameters of routine 0 of the library routine, since

(a) using routine, global or preset parameters of the master program could easily lead to clashes with other library routines if allowed, and

(b) the master programmer will not be interested in the breakdown of the library routine into sub-routines.

(ii) If preset parameters are used within the library routine, they should be high numbered ones, say P90-99, and should be unset at the end of the library routine. Their use should be mentioned in the specification for the library routine.

(iii) Any special preset parameters used (except P100 and P123) should be preserved and restored.

12.7.6 Referring to the master program from within a library routine

A routine parameter of the master program can be referred to within a library routine by treating the master program as if it were library routine 0, e.g. A6/3L0 is A6/3 of the master program.

12.8 Correction of Programs, and System Peculiarities

12.8.1 Program Alterations

To correct a small program, it is usually simplest to re-punch the tape or cards, making alterations as necessary. This is impracticable for larger programs, but the facilities of ABL may be used to help make corrections.

Very often it is possible to make corrections by overwriting certain store locations, using a *-directive. The corrections, however, must be compiled after the faulty items, or the faults will overwrite the alterations. Hence, the corrections are normally placed just before the enter directive. An enter directive may cause library routines to be compiled from the current transfer address, and so to prevent the program being overwritten from the faulty item onwards it is necessary to insert the correction in one of the following ways.

a) 1) <Last item of program proper>
```
* = 6A10/4
<Correction>
* = 1A1
EA40
```

b) <Last item of program proper>
```
L
* = 6A10/4
<Correction>
```
1) EA40.

In case b), however, if data is read to A1 onwards, this will also overwrite program when the correction is inserted.

It may be convenient to end a program with
```
R10
```
1) <Last item of program proper>
```
P115 = 15
* = 1A1/10
EA40/3
```
so that corrections, if any, will be read from input stream 15, which ends with P115 = 0.

When routine parameters are mentioned in a correction, without specifying any routine, i.e. /n is omitted, the routine to which they refer will be that current before the correction, rather than that at the location to be corrected.

Difficulties may be encountered when using '*=' to overwrite an item containing forward references. The result can be predicted from the

following notes on ABL's handling of forward references.

Two lists are concerned, the forward reference list in which are partly evaluated expressions containing forward references, and the parameter list in which are all parameters found in the program with their values if set.

(i) The forward reference list is initially empty.

(ii) When an expression is read it is added to the end of the forward reference list, which is then condensed starting from that expression. In the case of indeterminate parameters which need to be evaluated before compiling continues, i.e. after EX, ?, *=, Pa=, and Pa?= when Pa has not been set earlier (a is an integer), then the expression will be faulted as EXPRESSION INDETERMINATE (see section 11.6).

(iii) Whenever a routine or global parameter is set the forward reference list is condensed from the beginning.

(iv) On reading an E or ER directive all necessary library routines are read, any outstanding 'A?=' or 'G?=' directives are implemented in the order in which they occur in the program (this may lead to further settings of parameters and so further condensations of the forward reference list), and if the forward reference list is not empty its contents are output as Indeterminate Expression errors.

(v) On reading EX, A?= or G?= are implemented as in (iv).

Condensation of the forward reference list.

1. For each expression in turn, each set routine or global parameter in it is replaced by its value and the expression is partly evaluated. If no parameters remain unset the expression is completely evaluated; in that case, if the expression is not the right-hand side of an 'A?=' or 'G?=' directive, it is planted in the program area or the parameter list and deleted from the forward reference list.

2. If, on reaching the end of the forward reference list, any parameters have been set during 1, the process is repeated from the beginning.

For example, suppose the program begins

```
*=0, HA5
A3=A4-1
*=0, HA3-1
A4=4
```

On reaching the 4th line the forward reference list will contain the expressions A3, A4-1, A3-1. When the 4th line is read the list becomes

A3, A4-1, A3-1, 4;

the '4' is evaluated immediately and A4 becomes set; the list is then completely re-condensed.

A3 is not yet set and so remains.
A4-1 is evaluated and A3 gets set.

A3-1 is evaluated and planted.

The list now consists of only A3 and since a parameter was set in the last condensation the list is re-condensed.

A3 gets evaluated and planted.

It is seen that in this case the half-word ends with the value of A3, i.e. 3.

If a routine or global parameter is optionally set more than once, but is not set otherwise, then the first optional setting will be implemented. The subsequent optional settings will not be checked for faults.

A different mode of correction uses the library facilities of the ABL compiler. A copy of the compiler would be dumped initially onto a private magnetic tape, and subsequently used to compile routines as a library on to the tape. As these routines are corrected, the new versions are introduced, replacing the faulty library routines as described in section 12.7.4.

12.8.2 Further Peculiarities

Floating-point numbers may be represented in the form a(b:c):d or Ka(b:c):d as described in section 5.11. Although the number may be within the range of the accumulator, compiling it may cause exponent overflow unless the following three limitations are observed.

(i) $|b+c| < 100$

(ii) $|b| < 1000$

(iii) a consists of not more than 20 digits

When a parameter optionally set by a ?= directive is actually set elsewhere, the right hand side of the equation for the optional settings may not be checked.

After the final ABL fault printing, no new line is output.

ABL will read program incorrectly after $8191 = 2^{13} - 1$ printed lines without the implicit setting of a routine parameter by labelling.

In fault pointing, the line count will be taken modulo 2^{12}.

No check is made that function codes exist. One and two digit functions are right justified into the function bits.

When obeying program, the compiled value of * will be different from the current value of control, as b127 is stepped on by 1 before starting to obey an instruction. Thus

```
121    69    127    -*
```

would set b69' = 1. For the same reason

```
121    127   127    -1
```

causes a loop stop.

When more than twelve digits in a number are printed by the general output routine L1, digits after the twelfth may be wrong. L1 also may give exponent overflow attempting to print the following numbers:-

Non-zero Mantissa	Exponent
-1	+127
$\pm$ x($x \neq -1$)	-127
$\pm$ x	-128

12.9 Compiler and Supervisor Extracodes

The extracodes given below are used mainly by system programmers. They complete the list of Atlas 1 extracodes.

1126 $v7' = n$
and hoot if the least significant integer bit of n is 1. (bit 20)

1127 $ba' = v7 \ \& \ n$
Mask the digits of the engineer's handswitches with n, and read them to bits 16-23 of Ba.

Line 7 of the central computer V-store consists of 8-bits. Only bits 16-23 may be read, being set from the engineer's handswitches. Other bits are read as zero. Bit 20 controls the hooter and may be set by program; writing to the other bits is ignored.

1140 Read 'parameter' Ba of program to store starting at location S.

Ba	Parameter
0	Job title (10 words)
1	Computing time estimate, in seconds, in digits 0-23 (One half-word)
2	Execution time estimate, (One half-word)
3	Number of store blocks required, in digits 1-11 (One half-word)
4	'Parameter' in Job Description (One-half-word)
5	Logical tape numbers defined (8 half-words). The j^{th} digit ($0 \leq j \leq 15$) of the i^{th} half-word is a 1 if tape number $16\,i + j$ is defined.
6	Inputs defined (One half-word). The i^{th} digit ($0 \leq i \leq 15$) is 1 if input stream i is defined.
7	Outputs defined (One half-word) As 6.

1141 Define Compiler

Ba = Tape Number to which the compiler is to be written. If Ba = 127, and if the compiler name specified (see below) appears in the Supervisor Directory, the compiler will be written to the current Supervisor tape. In this case there will be two loop stops with J70707070 in B120 since this extracode will use 1143, 0, 0, 0.1 and 1143, 0, 0, 0.2 (see below).

The five half-words S to S+2 contain the following parameters:

a) First four characters of name.

b) Second four characters of name.

c) Main store starting Address (of where the compiler is now).

d) Main store finishing Address (of where the compiler is now).

e) Actual main store starting Address (of where the compiler is to be placed when in use).

Notes:

i) The compiler name should be right justified within each half-word, but the first four characters should be put in the first half-word,

e.g. ABL = J00414254, 0

HARTRAN = J50416264, J00624156

ii) If the first four characters are zero, the second four will be used as the starting block address (digits 0 to 21) of the compiler on tape. This facility cannot be used when Ba = 127.

iii) The starting address and the actual starting address should have bits 12 to 23 zero, and the starting address must be greater than zero, since the block before this is used to set up the compiler title block.

iv) The following fault indications may be printed:

COMPILER NAME NOT LISTED

COMPILER NOW U/S TAPE FAIL

COMPILER TOO BIG

WRITING TO SPECIFIED BLOCKS ON SYSTEM TAPE IS PROHIBITED

The first and third can only occur if Ba = 127. The fourth implies the facility described in Note (ii) has been used with Ba = 127.

1142 End compiling

(i) If Ba $\neq$ 0, set Compile/Execute switch to Execute, and reduce store allocation to that specified in Job Description. Lose all store blocks with block labels greater than or equal to digits 1-11 of ba, unless ba less than 0, in which case lose no blocks. Transfer control to address n, unless n less than 0, in which case End Program.

(ii) If Ba = 0, do none of the above; the only effect of this extracode is then to inform the Supervisor that the copy of the compiler being used has been, or may have been, spoilt, so that a new copy must be brought from magnetic tape for any subsequent job. The Compile/Execute switch is not changed. This extracode is useful where a compiler may have been spoilt by an interlude.

1143 Reserve Supervisor Tape

Ba should be zero

n should be 0.1, 0.2 or 0, with the following meanings:

n = 0.1 The Supervisor will come to a loop-stop with J70707070 in B120 waiting for the Write Permit switch to be switched on on the Supervisor tape. When this has been done, the

program will be allowed to write to, or read from, the Supervisor Tape (logical number 127), and normal use of the tape (e.g. reading compilers) will be halted.

n = 0.2 The Supervisor will come to a loop-stop with J70707070 in B120 waiting for the Write Permit switch to be switched off on the Supervisor tape, after which normal use of the Supervisor Tape will be resumed.

n = 0 The Supervisor Tape will be reserved as with N = 0.1. but for reading purposes only. There will be no loop stop, but 1143, 0, 0, 0.2 must be obeyed after reading, in order to release the tape for normal purposes. This facility is used if it is necessary to print out part of the Supervisor tape.

An operator request will be necessary before using this extracode.

1147 Call Compiler

(i) If n is even (digit 23 = 0), then n will be interpreted as a compiler number, and the compiler in question will be called from the Supervisor Tape and entered at the address specified by ba. The numbering of the standard compilers is given in Part 1 of the Operator's Manual (CS 411). This facility is used by the Supervisor, and by programs using compilers as subroutines.

(ii) If n is odd (digit 23 = 1), the compiler will be called from block b of tape a, where a = digits 15 to 21 of n
b = digits 2 to 14 of n

It will be entered at ba.

In both cases, if ba = 0, the standard entry point will be used.

With 1150 and 1151, ba, P and K are as defined in section 12.1.

1150 Assign ba blocks, labels P to (P + ba -1) to overflow K.

This extracode enables a program or compiler to temporarily hand blocks to the Supervisor, which may write them to the system dump tape. Subsequent use of these labels in the program causes new blocks to be assigned. The block labels are retained in the 'overflow' region and additions to this region must bear distinct labels. If ba = 0, one block is transferred.

1151 Set up ba blocks, labels P onwards, from overflow K.

This extracode recalls blocks previously written to the overflow region by use of 1150. Any existing blocks having these labels are overwritten. If ba = 0, one block is recalled. If these blocks do not exist in the overflow region, the program is monitored.

1156 Enter extracode control at n if the 'In Supervisor' switch is set.

This extracode is used by various Supervisor Extracode Routines which are obeyed on main control. If the 'In Supervisor' switch is not set, the program will be monitored.

1157 Enter extracode control at n if the 'Process' switch is set.

This may only be used by Supervisor routines such as the monitor called in during the running of a main program.

Appendix A
References

AUP 4	Mercury Orion Atlas Autocode.
CS 271	Operating Instructions for the Teletype Punch.
CS 294A	Punched Tape Codes.
CS 298A	Atlas Magnetic Tape.
CS 308B	Punched Tape Codes (5- and 7-track).
CS 309A	Extracode Functions.
CS 318B	Making a Fortran II Program suitable for use with the Atlas Fortran Compiler.
CS 339	Operating Instructions for the Model 'B' Flexowriter.
CS 345	List of Atlas Basic Instructions.
CS 349A	Atlas Programming Exercises.
CS 360	Telex Data Links.
CS 361	Keyboard used with Ferranti Computers.
CS 362	Operator's Conventions for Punched Tapes.
CS 363	Specifications of Paper for Punched Tapes.
CS 364	Specifications of Dimensions for Punched Tapes.
CS 365	Creed Teleprinter with Integral Tape Reader and Punch.
CS 366	Model 'S' Flexowriter.
CS 367	Creed Keyboard Punch for 7-track tapes.
CS 368	Creed Verifier for 7-track tapes.
CS 377	Algol 60 Report.
CS 378A	Reference manual for Atlas Algol (provisional).
CS 379A	A primer of Algol 60 for Atlas.
CS 384	Summarised programming information.
CS 390	Primer of Fortran Programming.
CS 401	The Analysis of Plane Structural Frames.
CS 402/5052	Extended Mercury Autocode for Orion and Atlas.
CS 405/5055	Traffic Assignment.
CS 411	The Atlas 1 Computer System Operators' Manual.
CS 428	Atlas user's description of the L.P. input scheme.
R 40	An Assembly programme for a Phrase-Structure Language.
R 55	The Manchester University Atlas operating system and Users' description.

R 58	The Atlas supervisor.
R 67	One-level storage system.
R 68	The Atlas scheduling system.
R 70	Processing commercial data on Atlas.
R 74	Central Control Unit of the Atlas Computer.
R 76	The Compiler Compiler.

Appendix B

Notation

Most symbols are used with two completely different meanings. The interpretation to be given to a symbol depends on its context. A convenient division is whether it is used in describing the arithmetic or the basic language, so the notation is listed under these two headings.

1. ARITHMETIC

Lower-case letters are used for suffices and for the content of a location, the location being in upper-case letters. The result of an operation is denoted by a prime. Thus, s is the content of address S, and am' = am + s means the content of Am after the operation is equal to the content before plus the content of S.

(a) Suffices

x	the argument of the prefixed location
y	the exponent of the prefixed location
:	two consecutive registers, starting with the one suffixed
*	the register following that suffixed

(b) Accumulator

A	the full double-length accumulator, holding a_x 79-bit mantissa a_x and 8-bit exponent a_y
a1	the 48-bit floating-point number formed from 1, 1s and a_y
am	the 48-bit floating-point number formed from m and a_y
aq	the single-length number which is obtained by standardising, rounding and truncating the contents of the accumulator
L	the less-significant half of A_x, of 39 bits with no sign
Ls	the sign bit associated with L
M	the more-significant half of A_x, of 40 bits, the most-significant bit being the sign digit
AO	accumulator overflow
DO	division overflow
E or EO	exponent overflow
Q	standardised
R	rounded, by forcing a 1 in the least-significant digit of M if L is not clear
R+	rounded, by adding a 1 to the least-significant digit of M if the most-significant bit of L is a 1

(c) General

B	any B-register (index register)
Ba	the B-register specified by the Ba digits of an instruction
Bc	the B-carry digit
Bm	the B-register specified by the Bm digits of an instruction
Bt	the B-test register
C	the main control register (B127)
C()	the contents of the location specified within the brackets
E	the extracode control register (B126)
F	the function digits of an instruction
G	the logical accumulator (B98 and B99)
I	the interrupt control register (B125)
N	the unmodified address part of an instruction (a 24-bit number with the point one octal place from the least-significant end).
n	the modified address part of an instruction regarded as a 24-bit number with the point one octal place from the least-significant end
S	the address of a store location. A full-word address in accumulator instructions (digits 21-23 ignored), a half-word address in B-register instructions (digits 22 and 23 ignored), or a 6-bit character address (all digits relevant)
V	the V-store
Vα	register α of the V-store
X	signifies extracodes suitable for fixed-point working.

2. BASIC LANGUAGE

In practice no distinction is made between capital and small letters, though capitals are used here. However, as an aid to clarity, it is sometimes advantageous to use lower case letters for the separators m, n, v, x and q. Small Greek letters α, β, are used for 21-bit decimal integers, k for the octal number in the 3 least-significant digits of a 24-bit address, and σ for a general octal number of up to 8 octal digits.

Aα	Parameter α, $(0 \leq \alpha \leq 3999)$, of the current routine
B	An operator in an expression, causing bits 12-23 of the previous element to be set to zero i.e. it gives a "Block Address"
C	Introduces a string of characters on the next line which are translated into internal code and placed in successive character positions

Dα	An operator, causing the previous element to be logically shifted down α places
E	The enter directive, causes the compiler to evaluate any used parameters etc., insert library routines, delete the compiler and enter the program
ER	As E but the compiler is not deleted and may be used again
EX	The enter interlude directive, to enter a short program for any reason during the compiling process
F	Introduces one or more floating-point numbers on a line, after some expressions otherwise interpreted. Also, if necessary, increases * to the next full-word address
Gα	Global Parameter α ($0 \leq \alpha \leq 3999$)
H	Subsequent expressions on a line are interpreted as 24-bit words and, if necessary, * is increased to the next half-word address
Jσ	σ is octally justified to the left, i.e. the most-significant digit goes into bits 0-2, the next into 3-5 etc.
Kσ	σ is octally justified to the right, to bit 20, i.e. the least-significant digit goes into bits 18-20, the next into 15-17 etc.
Kσ.k	As Kσ, with k going into bits 21-23
Lα.k	Library routine number α, copy k. (.k is omitted if only one copy of the routine is wanted) (α = 1 to 1999, k = 1 to 1999)
M	Alternative to &
N	A separator which non-equivalences the element before it with the element after it in an expression
Pα	Preset parameter α ($0 \leq \alpha \leq 99$ normally)
Q	A separator in an expression which divides the element before it by the element after it, placing the result in digits 0-20
Rα	A directive defining the beginning of routine α ($0 \leq \alpha \leq 3999$)
S	Expressions after the directive S are interpreted as 6-bit characters, i.e. only bits 15-20 of the expression are used
Tα or Tα-β	The title is copied to output channel α or channels α to β inclusive
Uα	An operator causing the previous element to be logically shifted up α places
Uα or Uα-β	Unset preset parameter α, or α to β inclusive

V	A separator in an expression. The element before it is OR-ed with the element after it
W	An operator in an expression, causing bits 0-11 of the previous element to be set to zero, i.e. it gives an address within a block
X	A separator. The element before it is multiplied by the element after it
Yα	α is placed in bits 0-23 (instead of bits 0-20)
Z	A directive indicating the end of a routine
*	The address of the first character position in the location where the item is placed. If used in an expression on the right-hand side of a directive, then * is the address of the next character position
|	All subsequent characters up to NL are ignored (| is not a terminator)
£,π	Alternatives to |
[ ]	All characters between square brackets are ignored. Bracket nesting to any level is allowed
,	Alternative to multiple space as a terminator
&	A separator which logically ANDs the element before it with the element after
:	a special separator used in (a) an element $\alpha : \beta$. α modulo 2^{12} goes to bits 0-11 and β modulo 2^{21} to bits 0-20, added to α (b) a floating-point number N ($\alpha : \beta$) : γ where the value of the number is $N \times 10^{\alpha} \times 8^{\beta}$ and the exponent is forced to γ or standardised if γ is omitted
/	(a) in a parameter, / separates the parameter number and routine number (b) an alternative to :
' (prime)	an operator which forms the logical binary complement i.e. it replaces 1's by 0's and 0's by 1's
?	(a) in the context A α ? = expression, A α is only set to the given expression if no other definite setting of A occurs before the program is entered. Similarly for G α ? = expression (b) in the context P α ? = expression, P α is set equal to the given expression unless P α is already set, in which case the directive is ignored (c) in the context ? expression, causes the compiler to ignore the remainder of the line if the value of the expression is zero.

Appendix C

V-Store Addresses of Peripherals

Each peripheral is allocated one or more words in a part of the V-store associated with its particular type of equipment.

A V-store address is identified by having 6 as its most-significant octal digit; furthermore, since only the more significant half-words are used, the least-significant octal digit of the address is always zero.

That part of the V-store associated with the peripherals is the first 256 words of the block beginning with J6004.

To each type of equipment there corresponds 16 consecutive words, so that peripheral p of type q is allocated the V-store address

$$J6004 + 16q + p.$$

The type number q is defined in the following table:-

q	Equipment	V-store address of equipment 0 of each type
0	Card Readers	J60040000
1	Spare (London only: High Speed Data Link)	J60040200
2	TR7 Paper Tape Readers and N.E.P. Tape	J60040400
3	Graphical Outputs	J60040600
4	Anelex Line-printers	J60041000
5	I.B.M. Magnetic Tape	J60041200
6	Fast Paper Tape Punches	J60041400
7	TR5 Paper Tape Readers	J60041600
8	Teletype Punches	J60042000
9	Card Punches (and, Manchester only, X-ray Diffractometer)	J60042200
10	Spare (Manchester only: A.T. & E. On-Line Data Links)	J60042400
11	Teleprinters	J60042600

The addresses above are all for equipment 0 of the type indicated. Card Reader 1, for example, would be addressed by writing J60040010.

Appendix D

Character Codes

The following table lists all the available Atlas Internal Code characters together with their external representations in terms of punchings on 7-track tape, 5-track tape, and punched cards, using the standard Atlas character codes for these media.

The 7-track tape code is the I.C.T./Ferranti Orion/Atlas code. The 5-track code is the standard I.C.T./Ferranti code as used on Pegasus, Mercury, Sirius, Atlas and Orion. The card code is the Atlas Fortran card code.

Also in the table is an indication of which characters are available on the Anelex Line-Printer.

Some characters are designated "Unassigned". This indicates that no external printing characters have been assigned to these Internal Code characters. Most compilers and Input Routines treat these as Illegal Characters. However, these characters have had 7-track paper tape punchings assigned to them. This serves two purposes: (i) it means that, if at some later date characters are assigned to these paper tape punchings, then Internal Code characters are available and assigned to correspond to them, and (ii) it means that, since there is some internal representation for every 7-track paper tape code with odd parity, it is possible to use Internal Code representation for parity-checked "binary" information, rather than use pure "Binary" mode. However, it should be noted that, since the Supervisor treats the shift characters and the New Line characters in a special manner, the internal representation will not be an exact "image" of the external punchings. It should also be noted that some of these Internal Code characters are used by non-standard external codes to represent non-standard characters. Thus in all cases special care should be taken in using these characters.

One character (Outer Set 02) is designated "Spare". This character has no external printing assigned to it nor any 7-track paper tape code. It may however be used by non-standard external codes, and thus care should be taken over its use.

Internal Code character 00 (inner and Outer set) is designated "Not Assigned". This character is reserved for special purposes by the compilers and the Supervisor. It will never have a character assigned to it, and should never be used by normal programs.

Certain characters are not available as standard on any input medium, and may be treated as "spare" by input routines (for example, L100 treats them thus). They are

Outer Set 03 £

Outer Set 35 10

Outer Set 36 11

They have the meaning as shown when used as output characters destined for the line-printer.

Certain other characters have alternatives given in parentheses. This is because the characters are alternatives in one or more of the relevant external codes. (For example, Inner Set 13 is listed as ¤ or £; these are alternatives on 7-track tape, 5-track tape and cards.) However, in the case of the line-printer, only the first character so listed is relevant, and, except for %, the other character appears elsewhere in the table for line-printer purposes.

Note that, in the column for 5-track tape, the 5-track tape code is given with the sprocket hole after two information holes and not three. This is the reverse of the normal convention and is done because the form as printed corresponds to the internal binary representation of the character when "Binary" mode for Input or Output is in use.

The Fault character (77 Inner Set) has no external representation. It is used under certain circumstances on Input by the Supervisor as a translation of any external character which has no Internal Code representation.

Tabulate (02 Inner Set) is treated as a single space by the Supervisor on output equipment where it does not otherwise exist.

The external shift characters (06 and 07) are ignored by the Supervisor for equipments where they have no relevance (e.g. the line-printer).

The fifteen symbols

: ' [ ] < > = _ | ? , ² α β π

are on the fourth quadrant of the line-printer wheel. If none of these symbols are used in a line, the time to print the line is $\frac{1}{1000}$ minute; otherwise it is $\frac{1}{810}$ minute

Notation:

FS Figure Shift
LC Lower Case
LS Letter Shift
UC Upper Case
** A paper-tape character appearing in both shifts
.. The character concerned is not available on this peripheral.

Internal Code - Inner Set

Character	Internal code (octal)	7-track code (binary bits and case)	5-track code (binary bits and shift)	Atlas Fortran card code (holes punched)	Anelex Line-Printer (availability)
(Not Assigned)	00	..	..	..	..
Space	01	** 0010.000	FS 01.110	None	Yes
Tabulate	02	** 0000.100	..	..	..
Backspace	03	** 0010.101	..	..	..
Shift to outer set	04	..	..	..	..
Shift to inner set	05	..	..	..	..
Shift to LC/LS	06	** 0010.110	** 11.011	..	..
Shift to UC/FS	07	** 0000.111	** 00.000	..	..
(Open brackets	10	LC 0111.000	FS 10.100	0,8,4	Yes
) Close brackets	11	LC 0101.001	FS 01.100	10,8,4	Yes
, Comma	12	LC 0101.111	FS 11.110	0,8,3	Yes
π (£) Pi (Pounds)	13	LC 0111.011	LS 01.111	11,8,3	Yes
? Query	14	LC 0101.100	LS 10.111	11,8,5	Yes
& Ampersand	15	LC 0111.101	..	8,5	Yes
* Asterisk	16	LC 0111.110	FS 11.000	11,8,4	Yes
/ Oblique	17	UC 0011.111	FS 11.101	0,1	Yes
0 Zero	20	UC 0100.000	FS 00.001	0	Yes
1	21	UC 0110.001	FS 10.000	1	Yes
2	22	UC 0110.010	FS 01.000	2	Yes
3	23	UC 0100.011	FS 11.001	3	Yes
4	24	UC.0110.100	FS 00.100	4	Yes
5	25	UC 0100.101.	FS 10.101	5	Yes
6	26	UC 0100.110	FS 01.101	6	Yes
7	27	UC 0110.111	FS 11.100	7	Yes
8	30	UC 0111.000	FS 00.010	8	Yes
9	31	UC 0101.001	FS 10.011	9	Yes
< Less than	32	LC 0100.011	..	0,8,5	Yes
> Greater than	33	LC 0110.100	FS 10.001	10,8,5	Yes
= Equals	34	LC 0100.101	FS 01.010	8,3	Yes
+ Plus	35	UC 0111.101	FS 01.011	10	Yes
- Minus	36	UC 0111.110	FS 11.010	11	Yes
. Point	37	UC 0101.111	** 00.111	10,8,3	Yes

Inner set (continued)

Character	Internal code (octal)	7-track code (binary bits and case)	5-track code (binary bits and shift)	Atlas Fortran card code (holes punched)	Anelex Line-Printer (availability)
'Prime (alternative: n (letter n) on 5-track tape only)	40	LC 0100.000	FS 10.111	8,4	Yes
A	41	UC 1010.001	LS 10.000	10,1	Yes
B	42	UC 1010.010	LS 01.000	10,2	Yes
C	43	UC 1000.011	LS 11.000	10,3	Yes
D	44	UC 1010.100	LS 00.100	10,4	Yes
E	45	UC 1000.101	LS 10.100	10,5	Yes
F	46	UC 1000.110	LS 01.100	10,6	Yes
G	47	UC 1010.111	LS 11.100	10,7	Yes
H	50	UC 1011.000	LS 00.010	10,8	Yes
I	51	UC 1001,001	LS 10.010	10,9	Yes
J	52	UC 1001.010	LS 01.010	11,1	Yes
K	53	UC 1011.011	LS 11.010	11,2	Yes
L	54	UC 1001.100	LS 00.110	11,3	Yes
M	55	UC 1011.101	LS 10.110	11,4	Yes
N	56	UC 1011.110	LS 01.110	11,5	Yes
O	57	UC 1001.111	LS 11.110	11,6	Yes
P	60	UC 1110.000	LS 00.001	11,7	Yes
Q	61	UC 1100.001	LS 10.001	11,8	Yes
R	62	UC 1100.010	LS 01.001	11,9	Yes
S	63	UC 1110.011	LS 11.001	0,2	Yes
T	64	UC 1100.100	LS 00.101	0,3	Yes
U	65	UC 1110.101	LS 10.101	0,4	Yes
V	66	UC 1110.110	LS 01.101	0,5	Yes
W	67	UC 1100.111	LS 11.101	0,6	Yes
X	70	UC 1101.000	LS 00.011	0,7	Yes
Y	71	UC 1111.001	LS 10.011	0,8	Yes
Z	72	UC 1111.010	LS 01.011	0,9	Yes
(Unassigned)	73	(UC 1101.011)	..	..	..
(Unassigned)	74	(UC 1111.100)	..	..	..
(Unassigned)	75	(UC 1101.101)	..	..	..
(Unassigned)	76	(UC 1101.110)	..	..	..
Fault	77	..	..	..	..

Internal Code - Outer Set

Character	Internal code (octal)	7-track code (binary bits and case)	5-track code (binary bits and shift)	Atlas Fortran card code (holes punched)	Anelex Line-Printer (availability)
(Not Assigned)	00	..	..	..	..
Space	01	** 0010.000	FS 01.110	None	Yes
(Spare)	02	..	..	..	..
£ Pounds	03	..	..	..	Yes
Shift to outer set	04	..	..	..	..
Shift to inner set	05	..	..	..	..
Shift to LC/LS	06	** 0010.110	** 11.011	..	..
Shift to UC/FS	07	** 0000.111	** 00.000	..	..
(Unassigned)	10	(** 0001.000)	..	..	..
(Unassigned)	11	(** 0011.001)	..	..	..
(Unassigned)	12	(** 0011.010)	..	..	..
(Unassigned)	13	(** 0001.011)	..	..	..
Stop	14	** 0011.100	..	..	..
Punch On	15	** 0001.101	..	..	..
Punch Off	16	** 0001.110	..	..	..
: Colon	17	LC 0011.111	..	.6,8	Yes
ϕ (x) Phi (letter x)	20	..	FS 00.011	..	..
[Open square brackets	21	LC 0110.001	..	11,7,8	Yes
] Close square brackets	22	LC 0110.010	..	11,6,8	Yes
→ Arrow	23	..	FS 00.101	..	..
≥ Greater than or equal	24	..	FS 01.001	..	..
≠ Not equal	25	..	FS 10.010	..	..
_ Underline	26	LC 0100.110	..	10,6,8	Yes
| Vertical bar	27	LC 0110.111	..	10,7,8	Yes
^{2} (%) Superscript 2 (Percent)	30	LC 0101.010	..	..	Yes
≈ (v) Curly equal (letter v)	31	..	FS 00.110	..	..
α (10) Alpha (Ten)	32	UC 0101.010	..	..	Yes
β (11) Beta (Eleven)	33	UC 0111.011	..	..	Yes
½ Half	34	UC 0101.100	..	..	Yes
10 Ten	35	..	..	..	Yes
11 Eleven	36	..	..	..	Yes
(Unassigned)	37	(UC 1000.000)	..	..	..

Outer set (continued)

Character	Internal code (octal)	7-track code (binary bits and case)	5-track code (binary bits and shift)	Atlas Fortran card code (holes punched)	Anelex Line-Printer (availability)
(Unassigned)	40	(LC 1000.000)	..	..	..
a	41	LC 1010.001	..	..	..
b	42	LC 1010.010	..	..	..
c	43	LC 1000.011	..	..	..
d	44	LC 1010.100	..	..	..
e	45	LC 1000.101	..	..	..
f	46	LC 1000.110	..	..	..
g	47	LC 1010.111	..	..	..
h	50	LC 1011.000	..	..	..
i	51	LC 1001.001	..	..	..
j	52	LC 1001.010	..	..	..
k	53	LC 1011.011	..	..	..
l	54	LC 1001.100	..	..	..
m	55	LC 1011.101	..	..	..
n	56	LC 1011.110	..	..	..
o	57	LC 1001.111	..	..	..
p	60	LC 1110.000	..	..	..
q	61	LC 1100.001	..	..	..
r	62	LC 1100.010	..	..	..
s	63	LC 1110.011	..	..	..
t	64	LC 1100.100	..	..	..
u	65	LC 1110.101.	..	..	..
v	66	LC 1110.110	..	..	..
w	67	LC 1100.111	..	..	..
x	70	LC 1101.000	..	..	..
y	71	LC 1111.001	..	..	..
z	72	LC 1111.010	..	..	..
(Unassigned)	73	(LC 1101.011)	..	..	..
(Unassigned)	74	(LC 1111.100)	..	..	..
(Unassigned)	75	(LC 1101.101)	..	..	..
(Unassigned)	76	(LC 1101.110)	..	..	..
Erase	77	** 1111.111 **	11.111	..	..

Appendix E

Summary of Extracodes

Allocation of Function Numbers

There are 512 function numbers available for extracodes, 1000-1777. Of these, 1000-1477 are singly-modified instructions i.e. B-type, and 1500-1777 are doubly-modified i.e. A-type.

The extracodes are divided into sections as shown below:

1000 - 1077	Peripheral routines.
1100 - 1177	Organisational routines
1200 - 1277	Test instructions and character data-processing.
1300 - 1377	B-register operations.
1400 - 1477	Complex arithmetic, Vector arithmetic, and other B-type accumulator functions.
1500 - 1577	Double-length arithmetic and accumulator operations using the address as an operand.
1600 - 1677	Logical accumulator operations, trigonometric routines and half-word packing.
1700 - 1777	Logarithm, exponential, square root etc., and miscellaneous arithmetic operations.

Where possible, the last two octal function digits correspond to those of similar basic operations.

The extracode function is listed at the left of the page and followed by a reference and a description. The number of basic instructions obeyed is given at the right of the page. This number includes the extracode instruction and its entry in the jump table; where necessary a range or formula is given.

The extracodes are listed in numerical order, and are also classified by type; some extracodes are therefore given twice.

Extracode	Ref.	Description	Instructions Obeyed

Organisational and Peripheral Extracodes

E.1 Magnetic tape

Block Transfers

1001	9.3.1	Search for section n on tape B_a	
1002	9.3.1	Read next K+1 sections from tape B_a to store blocks, P, P+1...., P+K	
1003	9.3.1	Read previous K sections from B_a to P+K, ..., P.	
1004	9.3.1	Write P, P+1,.., P+K to next K+1 sections on B_a	

Extracode	Ref.	Description	Instructions Obeyed
1005	9.3.1	Move tape B_a forwards K+1 sections	
1006	9.3.1	Move tape Ba backwards K+1 sections	
		Organisational Instructions	
1007	9.5.1	Mount next reel of file Ba	
1010	9.5.1	Mount	
1011	9.5.1	Mount free	
1012	9.5.1	Mount on logical channel K	
1013	9.5.1	Mount free on logical channel K	
1014	9.5.2	Write title	
1015	9.5.2	Read title or number	
1016	9.5.2	Unload	
1017	9.5.2	Free tape	
1020	9.5.2	Release tape (pass to another program)	
1021	9.5.2	Release mechanisms	
1022	9.5.2	Re-allocate	
1023	9.5.2	How long?	
1024	9.5.2	Where am I?	
		Variable Length Organisation	
1030	9.4.2	Start reading forwards	
1031	9.4.2	Start reading backwards	
1032	9.4.2	Start writing forwards	
1033	9.4.2	Select tape Ba	
1034	9.4.2	Start reading forwards from fixed blocks	
1035	9.4.2	Start reading backwards from fixed blocks	
1036	9.4.3	ba' = selected magnetic tape	
1037	9.4.3	s' = mode of magnetic tape Ba	
		Variable Length Transfers	
1040	9.4.3	Transfer	
1041	9.4.3	Skip	
1042	9.4.3	Mark	
1043	9.4.3	Stop	
1044	9.4.3	Word search	
1046	9.7	Read next block on Orion tape	
1047	9.7	Read previous block on Orion tape	
E.2		**Input**	
1050	8.4	Select input n	
1051	8.4	Find selected input	
1052	8.14	Find peripheral equipment number	
1053	8.14	Test whether binary or internal code	
1054	8.14	Read next character to Ba. Jump to n at end of record	
1055	8.14	ba' = number of blocks read	
1056	8.14	Read ba half-words to S	
1057	8.14	Read next record to S	

Extracode	Ref.	Description	Instructions Obeyed
E.3 Output			
1060	8.4	Select output n	
1061	8.4	Find selected output	
1062	8.15	Find peripheral equipment type	
1063	8.15	Delete output n	
1064	8.15	Write character n	
1065	8.15	End this record	
1066	8.15	Write ba half-words from S	
1067	8.15	Write a record from S	
1070	8.15	Rename output n as input ba	
1071	8.15	Break output n	
1072	8.15	Define output n	
E.4 Subroutine Entry			
1100	7.7	Enter subroutine at s, ba' = c+1	6
1101	7.7	Enter subroutine at S, ba' = c+1	5
1102	7.7	Enter subroutine at bm, ba' = c+1	6
1362	7.7	Enter subroutine at n, b90' = c+1	
E.5 Branching			
1103	12.3.2	Establish Ba branches	
1104	12.3.2	Start branch Ba at S	
1105	12.3.2	Kill Ba. If Ba = 64 kill current branch	
1106	12.3.2	Halt current branch if Ba is active	
1107	12.3.2	Jump to n if Ba is active	
E.6 Monitor			
1112	11.3	Set Monitor jump to n	
1113	11.4.1	Do not restart	
1117	11.1.3	End program	
E.7 Miscellaneous Transfers			
1120	7.8	ba' = clock	
1121	7.8	ba' = date	
1122	12.4	ba' = local instruction counter	
1123	12.4	set instruction counter = n 2048	
1136	12.4	Read instruction counter	
1124	7.8	v6' = n	3
1125	7.8	ba' = v6 & n	4
1126	12.9	v7' = n (hoot)	2
1127	12.9	ba' = v7 & n (read handswitches)	3
E.8 Traps			
1131	7.5.2	See E.12 Character Data Processing	
1132	11.2	Set trap/normal mode	
1133	11.2	ba' = trap address	
1134	11.2	Trap	
1135	12.1	See E.10 Store	
1136	12.1	See E.7 Miscellaneous Transfers	

Extracode	Ref.	Description	Instructions Obeyed
E.9 Compiler and Supervisor			
1140	12.9	Read parameter Ba to s	
1141	12.9	Define Compiler	
1142	12.9	End compiling	
1143	12.9	Reserve Supervisor Tape	
1147	12.9	Call compiler n	
1150	12.9	Assign ba blocks, labels P to (P + ba-1), to overflow K	
1151	12.9	Set up blocks P onwards from overflow K	
1155	12.1	See E.10 Store	
1156	12.9	Enter extracode control at n if the "In Supervisor switch" is set	
1157	12.9	Enter extracode control at n if the "Process switch" is set	
E.10 Store			
1135	12.1	Jump to n when block ≥ ba defined	
1155	12.1	Find smallest block label defined	
1160	12.1	Read block P	
1161	12.1	Release block P	
1162	12.1	Duplicate read	
1163	12.1	Duplicate write	
1164	12.1	Rename	
1165	12.1	Write block P	
1166	12.1	Read to absolute page	
1167	12.1	Lose block P	
1170	12.1	Clear blocks	
1171	12.1	Store allocation = n blocks	
1172	12.1	ba' = number of pages available	
1173	12.1	ba' = number of blocks available	
1174	12.1	Reserve band n	
1175	12.1	Read K + 1 blocks	
1176	12.1	Write K + 1 blocks	
1177	12.1	Lose band n	

Arithmetic and Logical Extracodes

Accumulator operations are rounded floating-point unless marked X, when they are suitable for fixed-point working.

E.11 Tests			
1200	7.6.1	ba' = n if A0 set; clear A0	9
1201	7.6.1	ba' = n if A0 clear; clear A0	7
1204	7.5.3	See E.19 Logical Operations	
1206	7.6.2	ba' = n if most-significant character in g = 0	4
1216	7.6.2	ba' = n if bm > 0	5-6
1217	7.6.2	ba' = n if bm ≤ 0	4-5
1223	7.6.2	ba' = n if Bc = 1	4
1226	7.6.2	ba' = n if bt > 0	4-6
1227	7.6.2	ba' = n if bt ≤ 0	3-5
1234	7.6.1	c' = c + 2 if am approximately = s	11-12
1235	7.6.1	c' = c + 2 if am not approximately = s	11-12

Extracode	Ref.	Description	Instructions Obeyed

Approximate equality is defined by

$$\left|\frac{am-s}{am}\right| < C(ba), \text{ with am standardised}$$

If am = 0, am is not approximately = s

1236	7.6.1	ba' = n if ax > 0	4-6
1237	7.6.1	ba' = n if ax ≤ 0	3-5
1250 to 1253	7.5.2	See E.12 Character Data Processing	
1255	7.6.1	ba' = n if m is neither zero nor all ones	9
1265	7.5.3	See E.19 Logical Operations	
1727	7.6.1	c' = c + 1, c + 2, or c + 3 as am >, = or < s	7
1736	7.6.1	c' = c + 2 if |am| ≥ s	8
1737	7.6.1	c' = c + 2 if |am| < s	7

E.12 Character data processing

1131	7.5.2	Table search	8+6n
In 1250 and 1251, S is taken as a character address			
1250	7.5.2	ba' (digits 18-23) = s, ba' (digits 0-17) = 0	7-10
1251	7.5.2	s' = ba (digits 18-23)	11-18
1252	7.5.2	Unpack n characters starting from character address C(ba), to half-words from C(ba*)	16 + int.pt. ($6\frac{3}{4}$n)
1253	7.5.2	Pack n characters starting from half-word address C(ba*), to character address C(ba)	18 + 5n

E.13 B-register operations

1300	7.5.1	ba' = integral part of s, am' = fractional part of s	10
1301	7.5.1	ba' = integral part of am, am' = fractional part of am	9
1302	7.5.1	ba' = ba.n, rounded away from zero	23-24
1303	7.5.1	ba' = -ba.n rounded away from zero	22-23
1304	7.5.1	ba' = integral part of (ba/n), b97' = remainder	25-28

In 1302-1304, ba and n are 21-bit integers in digits 0-20

1312	7.5.1	ba' = ba.n	23-24
1313	7.5.1	ba' = -ba.n	22-23
1314	7.5.1	ba' = integral part of (ba/n) b97' = remainder	25-28

In 1312-1314, ba and n are 24-bit integers

Extracode	Ref.	Description	Instructions Obeyed
1340	7.5.1	ba' = ba. 2^{-n}; unrounded arithmetic shift right	10-22
1341	7.5.1	ba' = ba. 2^{n} ; unrounded arithmetic shift left	9-21
1342	7.5.1	ba' = ba circularly shifted right n places	10-19
1343	7.5.1	ba' = ba circularly shifted left n places	9-18
1344	7.5.1	ba' = ba logically shifted right n places	10-21
1345	7.5.1	ba' = ba logically shifted left n places	9-20
1347	7.5.1	s' = s v ba	5
1353	7.5.1	ba' = position of most-significant 1 in bits 16-23 of n (as B123)	7
1356	7.5.1	bt' = ba ≠ s	7
1357	7.5.1	bt' = ba ≠ n	5
1362	7.7	See E.4 Subroutine Entry	
1364	7.5.1	ba' = (ba & n) v (bm & n); b119' = (ba ≠ bm) & n	4
1371	7.5.1	b121' = Ba, b119' = N + bm	2
1376	7.5.1	bt' = ba & s	5
1377	7.5.1	bt' = ba & n	4

E.14 Complex arithmetic

The complex accumulator, Ca, is a pair of consecutive registers, the first register having address ba. If Ba = 0, Ca is locations 0,1. s: is a number pair. Ca may coincide with S: but not otherwise overlap with it. A is spoiled.

1400	7.4.6	ca' = log s:	
1402	7.4.6	ca' = exp s:	140
1403	7.4.6	ca' = conj s:	5
1407	7.4.2	See E.16. Miscellaneous B-type accumulator operations	
1410	7.4.6	ca' = √ s:	Max.117
1411	7.4.6	am' = arg s: radians	
1412	7.4.6	ca' = mod s:	Max. 53
1413	7.4.6	ca' = s cos s*, s sin s*	95
1414	7.4.6	ca' = 1/s:	15
1415	7.4.2	See E.16. Miscellaneous B-type accumulator operations	
1420	7.4.6	ca' = ca + s:	8
1421	7.4.6	ca' = ca - s:	8
1424	7.4.6	ca' = s:	6
1425	7.4.6	ca' = -s:	6
1456	7.4.6	s:' = ca	5
1462	7.4.6	ca' = ca. s:	18

Extracode Ref.		Description	Instructions Obeyed	
E.15	**Vector Operations**			
		The vectors are of order n. s_1 is stored in consecutive locations from ba, and s_2 from ba*. A is spoiled.		
1430	7.4.7	$s_1' = s_1 + s_2$	9 + 4n	
1431	7.4.7	$s_1' = s_1 - s_2$	9 + 4n	
1432	7.4.7	$s_1' = am.\ s_2$	10 + 4n	
1433	7.4.7	$s_1' = s_1 + am.\ s_2$	10 + 5n	
1434	7.4.7	$s_1' = s_2$ (forwards or backwards)	13 + 3n	
1436	7.4.7	$am' = \sum_{i=0}^{n-1} s_{1i} . s_{2i}$	10 + 5n	
1437	7.4.7	$a' = \sum_{i=0}^{n-1} s_{1i} . s_{2i}$	10 + 13n	
E.16	**Miscellaneous B-type accumulator operations**			
1407	7.4.2	Remainder and adjusted integral quotient	14-31	
1415	7.4.2	Generate pseudo-random number		
1441	7.4.5	See E.18. Arithmetic using address as Operand.		
1452	7.4.3	$m' = m.s_x.\ 8^{ay+sy-bay}$, $ay' = bay$	19-23	(X)
1456) 1462)	7.4.6	See E.14. Complex arithmetic		
1466	7.4.2	$a' = C(N+bm+ba) \times C(N+bm) + a$	18	
1467	7.4.2	$am' = \sum_{i=0}^{r} s_i x^i$ where $x = am$, $S_i = S+i$, $r = ba$	6 + 3n	
1473	7.4.3	$m' = (ax/s\,x).8^{ay-sy-bay}$, $ay' = bay$	24-28	(X)
1474	7.4.3	C(ba)' = quotient (am/s), am' = remainder	20-29	(X)
1475	7.4.3	C(ba)' = quotient (a/s), am' = remainder	19-28	(X)
1476	7.4.3	C(ba)' = quotient ([integral part of am]/s), am' = remainder	28-37	
E.17	**Double-length arithmetic**			
		The double-length number is s: = s + s* where $sy - 13 \geq s^*y$. s* and al are assumed to be positive numbers.		
1500	7.4.4.	a' = a + s:	10	
1501	7.4.4.	a' = a - s:	10	
1502	7.4.4.	a' = a + s:	14	
1504	7.4.4	a' = s:	4	
1505	7.4.4	a' = -s:	3	
1520) to 1535)	7.4.5	See E.18. Arithmetic using address as Operand		
1542	7.4.4	a' = a.s:	15	
1543	7.4.4	a' = -a.s:	19	
1556	7.4.4	a:' = a	5	
1562	7.4.5	See E.18. Arithmetic using address as Operand.		

Extracode	Ref	Description	Instructions Obeyed	
1565	7.4.4	a' = -a	5	
1566	7.4.4	a' = | a |	4-6	
1567	7.4.4	a' = | s: |	5	
1574) 1575)	7.4.5	See E.18 Arithmetic using address as Operand		
1576	7.4.4	a' = a/s:	19	

E.18 Arithmetic using address as Operand

The address is taken as a 21-bit integer with one octal fractional place. Fixed-point operations imply an exponent of 12.

1441	7.4.5	sx' = ba, sy' = 12	5	
1520	7.4.5	am' = am + n	10	
1521	7.4.5	am' = am - n	9	
1524	7.4.5	am' = n, 1' = 0	8	
1525	7.4.5	am' = -n, 1' = 0	7	
1534	7.4.5	am' = n, 1' = 0	10	(X)
1535	7.4.5	am' = -n, 1' = 0	9	(X)
1562	7.4.5	am' = am.n	8	
1574	7.4.5	am' = am/n	16	
1575	7.4.5	am' = aq/n	15	

E.19 Logical accumulator operations

The logical accumulator G is B98 and B99

1204	7.5.3	ba' = number of 6-bit characters from most-significant end identical in g and s	10-31
1265	7.5.3	g' = 2^6g + n, ba' = 6-bits shifted out of g	11
1601	7.5.3	g' = s	3
1604	7.5.3	g' = g + s	7
1605	7.5.3	g' = g + s with end around carry	12
1606	7.5.3	g' = g ≠ s	4
1607	7.5.3	g' = g & s	3
1611	7.5.3	g' = $\overline{g}$	3
1613	7.5.3	s' = g	3
1615	7.5.3	am' = g	4
1624) 1626)	7.4.8	See E.20 Half-word Packing	
1630	7.5.3	g' = g & $\overline{s}$	5
1635	7.5.3	g' = am	4
1646	7.5.3	g' = g v s	3
1652	7.5.3	bt' = g - s	7-9

E.20 Half-word packing

s has an 8-bit exponent and a 16-bit mantissa.

1624	7.4.8	am' = s	6
1626	7.4.8	s'= am, with s rounded	8

Extracode	Ref.	Description	Instructions Obeyed	
E.21 Functions and miscellaneous routines				
1700	7.4.1	am' = log s		
1701	7.4.1	am' = log aq		
1702	7.4.1	am' = exp s	43	
1703	7.4.1	am' = exp aq	42	
1704	7.4.2	am' = integral part of s	5	
1705	7.4.2	am' = integral part of am	4	
1706	7.4.2	am' = sign s	5-6	
1707	7.4.2	am' = sign am	4-5	
1710	7.4.1	am' = $\sqrt{s}$	Max 42	
1711	7.4.1	am' = $\sqrt{aq}$	Max 41	
1712	7.4.1	am' = $\sqrt{aq^2 + s^2}$	Max 50	
1713	7.4.2	am' = aq^s		
1714	7.4.2	am' = 1/s	4	
1715	7.4.2	am' = 1/am	4	
1720	7.4.1	am' = arc sin s $(-\pi/2 \leq s \leq \pi/2)$		
1721	7.4.1	am' = arc sin aq		
1722	7.4.1	am' = arc cos s $(0 \leq s \leq \pi)$		
1723	7.4.1	am' = arc cos aq		
1724	7.4.1	am' = arc tan s $(-\pi/2 < s < \pi/2)$		
1725	7.4.1	am' = arc tan aq		
1726	7.4.1	am' = arc tan (aq/s) $(-\pi \leq aq \leq \pi)$		
1727	7.6.1	See E.11 Test Instructions		
1730	7.4.1	am' = sin s	41	
1731	7.4.1	am' = sin aq	40	
1732	7.4.1	am' = cos s	42	
1733	7.4.1	am' = cos aq	41	
1734	7.4.1	am' = tan s	34	
1735	7.4.1	am' = tan aq	33	
1736) 1737)	7.6.1	See E.11 Test Instructions		
1752	7.4.3	m' = ax. 8^{12}; ay' = ay - 12	10	(X)
1753	7.4.3	ax' = m. 8^{-12}, ay' = ay + 12	6	(X)
1754	7.4.2	Round am by adding; standardise	6	
1755	7.4.3	ax' = ax. 8^{ay-ny}; ay' = ny	17	(X)
1756	7.4.2	s' = am, am' = s	8	
1757	7.4.2	am' = s/am	4	
1760	7.4.2	am' = am^2	3	
1762	7.4.3	m' = ax. 8^{12}	9	(X)
1763	7.4.3	ax' = m. 8^{-12}	5	(X)
1764	7.4.3	ax' = ax. 8^{n}	17	(X)
1765	7.4.3	ax' = ax. 8^{-n}	12	(X)
1766	7.4.3	am' = |s|	4	(X)
1767	7.4.3	am' = |am|	3	(X)
1771	7.5.1	b121' = Ba, b119' = N + ba + bm	2	
1772	7.4.3	m' = (m.sx) 8^{12}; ay' = ay + s y-12	11	(X)
1773	7.4.3	m' = (ax/sx) $8^{ay-sy-12}$; ay' = 12	27	(X)
1774	7.4.2	am' = am/s	10	
1775	7.4.2	am' = aq/s	9	
1776	7.4.2	Remainder	13	

Appendix F

Summary of Basic Instructions by Function

B-Line Operations

Logic Operations

106 ba′ = ba ≠ s	147 ba′ = ba v s	107 ba′ = ba & s
116 s′ = ba ≠ s	167 ba′ = ba v n	117 s′ = ba & s
126 ba′ = ba ≠ n		127 ba′ = ba & n
		164 ba′ = ba + (bm & n)
		165 ba′ = bm & n

Cyclic Shifts

143 ba′ = (1/2) ba - s	105 ba′ = (64) ba + s
163 ba′ = (1/2) ba - n	125 ba′ = (64) ba + n

Index Arithmetic

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

Set B-Test

150 bt′ = s - ba
152 bt′ = ba - s
170 bt′ = n - ba
172 bt′ = ba - n

Test Instructions

Count

200 If bm≠0, then ba′=n and bm′=bm+0.4	220 If bt≠0, then ba′=n and bm′=bm+0.4
201 If bm≠0, then ba′=n and bm′=bm+1.0	221 If bt≠0, then ba′=n and bm′=bm+1.0
202 If bm≠0, then ba′=n and bm′=bm-0.4	222 If bt≠0, then ba′=n and bm′=bm-0.4
203 If bm≠0, then ba′=n and bm′=bm-1.0	223 If bt≠0, then ba′=n and bm′=bm-1.0

Test

234 If ax=0, ba′=n	214 If bm=0, ba′=n	210 If bm odd, ba′=n	224 If bt=0, ba′=n
235 If ax≠0, ba′=n	215 If bm≠0, ba′=n	211 If bm even, ba′=n	225 If bt≠0, ba′=n
236 If ax≥0, ba′=n	216 If bm≥0, ba′=n		226 If bt≥0, ba′=n
237 If ax<0, ba′=n	217 If bm<0, ba′=n		227 If bt<0, ba′=n

Accumulator Operations

	Unstandardised (Pseudo Fixed-Point)	Standardised Floating-Point		Pseudo Double-Length
		Rounded	Unrounded	
Addition and Subtraction	330 a'=am+s AO 331 a'=am-s AO 332 a'=s-am AO	320 am'=am+s QRE 321 am'=am-s QRE 322 am'=s-am QRE	300 a'=am+s QRE 301 a'=am-s QE 302 a'=s-am QE	310 a'=am+s NQE 311 a'=am-s NQE
Transfers In	334 a'=s 335 a'=-s AO		324 a'=s Q 325 a'=-s QE	314 am'=s N 315 am'=-s NAO
				344 l'=s x 345 l'=s x, m'=ss
Transfers Out	346 s'=am, a'=0 356 s'=am			347 s'=al, l'=0 357 s'=al
Multi-plication	352 l'=m.s EAO 353 l'=-m.s EAO	362 am'=am.s QRE 363 am'=-am.s QRE	342 a'=am.s QE 343 a'=-am.s QE	372 a'=am.s EAO 373 a'=-am.s EAO
Division	375 al'=+a/|s| m'=rem E	374 am'=am/s l'=0 QRE DO	376 al'=+a/|s| m'=rem EDO 377 al'=|am|/|s| m'=rem EDO	
Standardi-sation and Rounding	361 am'=a RE 354 am'=a R+AO	360 am'=a QRE	340 a'=a QE	355 a'=al.8^{-1} Q
Octal Shifts & Moduli	364 ax'=8ax 365 ax'=ax/8		366 a'=|am| QE 367 a'=|s| QE	
Check Exponent Overflow	341 a'=a E			

AO	Accumulator may overflow	Q	Accumulator standardised
N	L not cleared	R	Accumulator rounded
R+	Rounded by adding	E	Exponent overflow may occur
◯	Cyclic Shift	DO	Division overflow may occur

Summary of Instructions by numbers

		0	1	2	3	4	5	6	7	
Index Instructions	10	ba′=s-ba Bc	ba′ = s	ba′=ba-s Bc	ba′= -s	ba′=ba+s Bc	ba′= (64b) +s	ba′ = ba∓s	ba′ = ba & s	10
	11	s′=s-ba Bc	s′ = -ba	s′=ba-s Bc	s′ = ba	s′=ba+s Bc	(s′=ba+s) No Bc	s′ = ba∓s	s′ = ba & s	11
	12	ba′=n-ba Bc	ba′ = n	ba′=ba-n Bc	ba′ = -n	ba′=ba+n Bc	ba′= (64b) +n	ba′ = ba∓n	ba′ = ba & n	12
	13	(NA n-type Bc)	(NA n-type)	(NA n-type Bc)	(NA n-type)	(NA n-type Bc)	(NA n-type)	(NA n-type)	(NA n-type)	13
	14	(As 100)	(As 101)	(As 102)	ba′= (½ba) -s	(As 104)	(As 105)	(As 147)	ba′ = ba v s	14
	15	bt′=s-ba Bc	(NA s-type)	bt′=ba-s Bc	(NA s-type)	(NA s-type Bc)	(NA s-type)	(NA s-type)	(NA s-type)	15
	16	(As 120)	(As 121)	(As 122)	ba′= (½ba) -n	ba′=ba+(bm&n) Bc	ba′= bm & n	(As 167)	ba′= ba v n	16
	17	bt′=n-ba Bc	(NA n-type)	bt′=ba-n Bc	(NA n-type)	(NA n-type Bc)	(NA n-type)	(NA n-type)	(NA n-type)	17
Test Instructions	20 (24)	If bm≠0, ba′=n and bm′=bm+0.4	If bm≠0, ba′=n and bm′=bm+1.0	If bm≠0, ba′=n and bm′=bm-0.4	If bm≠0, ba′=n and bm′=bm-1.0	(As 200)	(As 201)	(As 202)	(As 203)	20 (24)
	21 (25)	If bm odd, ba′=n	If bm even, ba′=n	(As 210)	(As 211)	If bm=0, ba′=n	If bm≠0, ba′=n	If bm≥ 0, ba′=n	If bm < 0, ba′=n	21 (25)
	22 (26)	If bt≠0, ba′=n and bm′=bm+0.4	If bt≠0, ba′=n and bm′=bm+1.0	If bt≠0, ba′=n and bm′=bm-0.4	If bt≠0, ba′=n and bm′=bm-1.0	If bt=0, ba′=n	If bt≠0, ba′=n	If bm ≥ 0, ba′=n	If bt < 0, ba′=n	22 (26)
	23 (27)	(As 234)	(As 245)	(As 236)	(As 237)	If ax=0, ba′=n	If ax≠0, ba′=n	If ax ≥ 0, ba′=n	If ax < 0, ba′=n	23 (27)
Accumulator Instructions	30	a′=am+s QE	a′=am-s QE	a′=am-s QE	(As 302)	(As 324)	(As 325)	(As 324)	(As 324)	30
	31	a′=am+s N QE	a′=am-s N QE	(As 302)	(As 302)	am′=s N	am′=s N AO	(As 324)	(As 324)	31
	32	am′=am+s QRE	am′=am-s QRE	am′=am+s QRE	(As 322)	a′=s Q	a′=s QE	(As 324)	(As 324)	32
	33	a′=am+s AO	a′=am-s AO	a′=am+s AO	(As 332)	a′=s	a′=s AO	(As 334)	(As 334)	33
	34	a′=a QE	a′=a E	a′=am.s QE	a′=-am.s QE	l′=sx	l′=sx, m′=ss	s′=am, a′=0	s′=al, l′=0	34
	35	(As 340)	(As 341)	a′=am.s AO E sgn l′=sgn m′	a′=-am.s AO E sgn l′=sgn m′	am′=a R+ AO	l′=a1.8^{-13}Q	s′=am	s′=al	35
	36	am′=a QRE	am′=a RE	am′=am.s QRE	am′=-am.s QRE	ax′=8ax	ax′=ax/8	a′= |am| QE	a′= |s| QE	36
	37	(As 340)	(As 341)	a′=am.s AO E	a′=-am.s AO E	am′=am/s QRE DO l′=0	al′=a/|s| m′=rem E*	al′=a/|s| m′=rem E DO*	al′=|am|/|s| m′=rem E DO	37

Legend

- ◯ Circle denotes circular shift
- N L not cleared (including Ls)
- ss sign digit of sx
- Q Accumulator standardised
- R Accumulator rounded by Forcing
- R+ Rounded by adding
- E Exponent overflow may occur
- DO Division overflow may occur
- AO Accumulator overflow may occur

Instructions given in brackets are non-standard and should not normally be used. They are given here only for the sake of completeness.

NA means that no registers are altered as a result of these operations. However instructions designated "s-type" do make a store-reference and thus SVO or Non-equivalence may occur

The "n-type" instructions do not make any store-reference, and are thus in effect dummy instructions.

* These division instructions are not fully described by the summary. Reference should be made to Chapter 6 before use.

Bc These instructions set B-carry

ADDENDUM TO CS348A

HALF INCH TAPE OPERATIONS

Some Atlas installations are equipped with half inch tape mechanisms as well as the standard one inch tape. The half inch tape operations are described in the following sections. At present there is an upper limit of 4096 characters in a record, if no information is to be lost during a transfer, and this is described herein; this limit may later be removed.

Section		Date of Issue
9.8	Atlas Half Inch Tape	5.65
9.9	Half Inch Tape Instructions	
	9.9.1 The Selection Instructions	5.65
	9.9.2 The Transfer Instructions	5.65
	9.9.3 The Skip Instructions	5.65
	9.9.4 Other Instructions	5.65
	9.9.5 Summary of Half Inch Tape Instructions	5.65
9.10	Specification of the Atlas Half Inch Tape	
	9.10.1 Control	5.65
	9.10.2 The Tape Layout	5.65
	9.10.3 Performance	5.65
	9.10.4 Safeguards	5.65
9.11	Half Inch Tape Faults	5.65
(Job Description).		
	10.6.4 Half Inch Tapes	5.65

9.8 Atlas Half Inch Tape

Information on each magnetic tape is split up into sections or records, consisting of up to 4096 six bit characters each. Groups of records may be formed into a file, separated from other files by file marks - short, standard records recognised by hardware. When the tape is first mounted it is positioned ready move forward and use the first record. As the records are of variable length, and the tape is not pre-addressed, it is very difficult to use particular records, unless these occur sequentially on the tape. The file marks enable unwanted files to be neglected without having to examine every record within them.

Searching for file marks is a relatively slow process compared with the computing speed of Atlas, and the time taken is proportional to the length of tape traversed. Any search instructions should be given as early as possible in the program.

The characters stored on the tape each have a parity bit associated with them - the programmer can chose whether this should give odd or even parity. The character themselves may be in one of two forms: Binary Coded Decimal, (BCD) usually for input or output data, and normally with even parity, or pure Binary, usually with odd parity, when the tapes are used as an extension of the computer store. The characters may be packed with one of up to three densities on the tape.

9.9 Half Inch Tape Instructions

9.9.1 The Selection Instructions

The tape which is required for processing is initially selected by obeying a 1260 instruction, which also defines the maximum record length, and the mode of transfer. All succeeding half inch tape instructions apply to this tape, with the exception of the 1262 instruction. The tape number is that appearing in the job description, and is specified by the Ba digits of the instruction ($0 \leq Ba \leq 126$). The maximum record length is specified by bits 12-20 of the singly-modified operand n; hence the longest record allowable is 512 Atlas words, or 4096 characters. The octal fraction of n determines the mode of transfer as follows:-

bit 21 = 0 for odd parity
= 1 for even parity
bit 22 = 0 if no code conversion is required (binary)
= 1 if code conversion is required (BCD)
bit 23 must be zero

Thus, the instruction

```
1260    42    0    48
```

would select tape 42 to provide backing store, transfers being unconverted with odd parity, with a maximum of 384 characters in any record.

```
1260    2    0    4.6
```

would select tape 2 for B.C.D. even parity working, maximum record length 32 characters.

It may often be necessary to find which tape is being used, for instance on entering a subroutine. Extracode 1261 will find the selected tape, mode, and record length. The contents of Ba will be set to the tape selected, stored in the half word position, bits 15-21. ba will be set to J4 if no half inch tapes are selected. The record length and mode will be given in the same way as for the 1260 instruction, and will appear in the B-line specified in the half-word position (bits 15-21) of the singly modified operand n. Thus the instruction

```
1261    10    0    11D1
```

at the beginning of a routine, and

```
 121    121    10    0
1260    122    11    0
```

at the end would preserve the half inch tape selected on entry, together with the maximum record length and mode of transfer.

Before any transfers take place, the instruction 1256 must be obeyed to select the density of character packing on the currently chosen tape, using operand n as follows:-

n = 0 for low density, 200 characters per inch
n = 0.4 for medium density, 556 characters per inch
n = 1 for high density, 800 characters per inch

This density applies to the selected channel and should not be changed between tape transfers. If an attempt is made to read a tape in a higher density than it has been written in, the situation may be irretrievable. Hence if a tape's density is unknown, it should be read first with low density selected. If the wrong density is selected, there will be a read failure, which may be trapped (see sections 9.11 and 11.2).

9.9.2 The Transfer Instructions

Records are written to the selected tape using the 1270 extracode. The number of characters, specified by bits 12-23 of ba, must not exceed the number specified by the 1260 extracode. The first character transferred is taken from the address S.

The 1274 instruction will read the next record from the tape. Ba must be 1; S specifies the character position to which the first character will be transferred. Thus

```
1260    1    0    63.4
```

will transfer the next record from tape to store, starting at address location 63.4. If S < 0 there will be no storage of the record, providing a means of skipping records. If the number of words in the record is greater than the maximum specified by the 1260 instruction, the tape will continue to move to the end of the record, but the excess words will be lost.

The 1276 instruction sets ba to the number of characters transferred from the tape by the previous 1274 instruction reading from the currently selected half inch tape. This may be greater than the maximum length specified by the 1260 instruction, in which case characters will have been lost.

9.9.3 The Skip Instructions

Besides reading and writing records, it is also possible to move the tape along without either operation occurring. Skipping one record forward is achieved by 'reading' to a negative store address using 1274. The instruction

```
1275    1    0    S
```

where S is negative, will skip one record backwards.

The tape is moved more quickly when searching for a file mark than when simply scanning each record, and so the 1257 instruction allows this faster search either backward or forward along the tape; 1257 is also used to write a file mark at the current position of the selected tape. The operation performed depends on the value of Ba as follows:-

Ba = 0, write a file mark
Ba = 1, search forwards for a file mark, positioning the tape to read the record following it.
Ba = 2, search backwards for a file mark, positioning the tape to read the file mark. The short record forming the file mark must be skipped before reading the first record of the file; alternatively searching forward for a file mark will have the same effect.

The tape may be rewound and positioned ready to read the first record by obeying the instruction

```
1267    0    0    0
```

9.9.4 Other Instructions

When a program deals with data that may be either on one inch tape or on half inch tape it is necessary to know which is being used in a particular run. Obeying a 1262 instruction will set ba according to the type of device that is listed as number n (singly modified operand) in the job description. Thus

```
1262    10    0    6
```

will set b10' = 0 if device 6 in the job description is a one inch tape mechanism
= 0.4 if device 6 is a half inch tape mechanism
= J4 if no device 6 is defined.

The following instructions have a similar use for both one and half inch mechanisms.

1010 Mount tape. For half inch tape, although it will be called for by name, no check can be made that the correct tape is mounted.

1016 Unload and Store
1021 Release Mechanisms

9.9.5 Summary of Half Inch Tape Instructions

1256 Select density n on selected mechanism
n = 0 low density, 200ch/in
n = 0.4 medium density, 556ch/in
n = 1 high density, 800ch/in
Reading in a higher density than the tape is written in has an unpredictable effect. Once the correct density is found, no further changes should be made.

1257 Write file mark/Skip to file mark, using selected device.
Ba = 0, write a file mark
Ba = 1, position the tape to read the record following the next file mark, moving the tape forwards.
Ba = 2, move the tape backwards to the previous file mark, positioning the tape to read the file mark.

1260 Select deck Ba; define record length and mode of transfer n. ($0 \leq Ba \leq 126$)
Bits 12-20 of n define the maximum record length
Bit 21 of n = 0 for odd parity
= 1 otherwise
Bit 22 of n = 0 if no code conversion required
= 1 otherwise
Bit 23 of n = 0 always

1261 Set ba to selected device; set b(n) to record length and mode defined.
ba' = number of selected deck in the half word position
= J4 if no half inch tape machamism selected.
The B-register given in bits 15-21 of n is set to the record length and mode as defined in 1260.

1262 Set ba to type of device n (bits 0-20)
ba' = 0 if device n is one inch tape
= 0.4 if device n is half inch tape
= J4 if no device n defined

1267 0 0 0 Rewind selected device. Position tape to read first record.

1270 Write a record of ba characters from core store starting at S to selected device
ba must not exceed the number of characters defined by 1260

1274 Read a record to core store starting at S/Skip forward one record, using selected device.
Ba must be 1.
If the record is longer than defined by 1260, the extra characters will be lost. If S < 0, there will be no transfer, only skipping.

1275 Skip backward one record on selected device. Ba must be 1, and S negative.

1276 Set ba to length of previous record read from selected device.
If ba' is greater than the record length defined in 1260, characters have been lost.

9.10 Specification of the Atlas Half Inch Tape System

9.10.1 Control

An Atlas installation may have up to 6 half inch magnetic tape mechanisms, all connected to the computer through one channel, which may be used by one inch mechanisms in the usual way. This channel controls a read, write or backspace operation on one deck at any one time. Rewind operations are autonomous, and need the channel only to initiate them. A search for a file mark is autonomous with any operations involving one inch mechanisms connected to the same channel.

9.10.2 The Tape Layout

The tape mechanism is the IBM 729 Mark IV or the Potter MTS 120X - 41306, using half inch wide magnetic tape. There are seven tracks across the tape; six are used for data, and one for parity. There is no clock track provided; characters are recognised by the presence of a bit in at least one of the tracks.

Information is stored on tape in blocks or records, which can be of any length, and are unaddressed. Because records are of variable length, selective overwriting is virtually impossible. At the end of each record is one character generated from the parities of each track in the record; these parity characters are used to check the accuracy of all reading and writing operations. Records may be grouped into files separated by file marks - short records recognised by the hardware.

Data is stored in two different character representations.

a) Binary Mode

When it is required to use the tape as a backing store to the computer, the data will be transferred to tape directly from the main store without any code conversion, i.e. in binary form. The parity track bits then ensure odd parity.

b) B.C.D. Mode

When it is required to use the tape as an input or output device, then alpha-numeric information will be required. The data is converted into Binary Coded Decimal (BCD) by the hardware; the parity track bits ensure even parity.

Three densities of recording are possible: 200, 556 and 800 characters per inch. Each record is separated from its neighbour by a 3/4" gap in which nothing is recorded. Tapes are up to 2400 feet long.

9.10.3 Performance

For the IBM deck, the normal tape speed is 112.5 inches per second, allowing instantaneous transfer rates of 22,500, 62,500 or 90,000 characters per second for the low, medium and high densities respectively. There is a fast wind and rewind speed of about 500 inches per second; this speed is also used on long searches for file marks. For the Potter decks the normal tape speed is 120 inches per second giving instantaneous transfer rates of 24,000, 66,600 and 96,000 characters per second. The fast wind speed is 240 inches per second.

There are independent read and write heads, separated by a gap of about 0.3 inches. It is possible to read or write when the tape is moving forwards only, although the tape may be backspaced a record at a time.

9.10.4 Safeguards

A program is held up if it attempts to read from or write to a block of store which is involved in a magnetic tape transfer. The Supervisor may then enter another program until the transfer is completed.

If a transfer cannot be initiated when it is requested, it is placed in queue; if this is already full, the program is held up.

A write permit ring must be fitted to a reel of tape before that reel can be written on. Tapes containing permanent information will not have such a ring.

Because the tape's title is not written on the tape itself, the operator must ensure that the correct tape is mounted on the deck allocated; the Supervisor will assume that any tape so mounted will be the one requested.

9.11 Half Inch Tape Faults

Trap entry 7 is now defined as

IBM TAPE ERROR

This message will be printed by the Supervisor if the fault is not trapped (see Chapter 11). If the fault is trapped, B119 contains a number giving the reason for monitoring; this number is printed in decimal after 'IBM TAPE ERROR', if not trapped.

b119	Reason for monitoring
0.0	Filemark read by 1274
0.4	Read failure
1.0	End of tape detected either following a 'skip backwards' or a 'write' instruction
1.4	Write failure
2.0	Failure to detect a filemark
(2.4	No IBM LAM within 2 minutes).

10.6.4 Half Inch

At those
magnetic tape, each
means of a TAPE hea
of the letters IBM,
tape reference let

TAPE

IBM 4

TAPE

IBM C

The tape
same way as for one
programmer's numbe

68 NEWMAN STREET,
LONDON, W.1.

LIST NO. CS348A
M.F. CARDINAL