

Ferranti

**THE ATLAS
PROVISIONAL
PROGRAMMING MANUAL**

LIST CS 348
JANUARY 1963

PREFACE

The Atlas Computer is the latest result of a long-standing collaboration between Manchester University and Ferranti Limited; this machine is now known as Atlas 1, the name Atlas 2 being used for a later version developed jointly by Cambridge University and Ferranti Limited. The information in this manual is intended to refer to Atlas 1 only, but as the two machines are substantially compatible much of the manual applies equally well to Atlas 2.

The pages which follow describe the machine-language programming facilities of Atlas: they are not concerned with problem-oriented languages such as Mercury Autocode or Fortran. There are incorporated, however, details of the Atlas Basic Language, ABL; this is a symbolic input language enabling the programmer to utilize a variety of parameters and symbolic expressions, and it includes a comprehensive system of directives to control the assembly of a complete program from several input documents and other sources.

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 engineers' test programs, in which only the accumulator is numbered as here and in all other cases bit 0 is the least-significant bit.

At the outset it should be understood that this manual is necessarily provisional: it has been written concurrently with the commissioning of many of the facilities which it describes, and some of the details have been subject to revision. Much of the material is therefore new and is presented here for the first time; it is hoped that a more permanent manual will soon become available.

CONTENTS

| | Page |
|--|-----------|
| PREFACE | |
| CHAPTER 1 AUTOMATIC DIGITAL COMPUTERS | 1 |
| 1. Introduction | 1 |
| 1.1 Electronic Computers | 1 |
| 1.1.1 Digital Computers | 1 |
| 1.2 Addresses | 1 |
| 1.3 Instruction Code | 2 |
| 1.4 Jump Instructions | 2 |
| 1.4.1 Looping | 2 |
| 1.4.2 Modification | 2 |
| 1.5 Binary Numbers | 2 |
| 1.5.1 Negative Numbers | 2 |
| CHAPTER 2 THE ATLAS COMPUTER | 3 |
| 2.1 A General Description of Atlas | 3 |
| 2.1.1 The Control Unit | 3 |
| 2.1.2 The Arithmetic Units | 3 |
| 2.1.3 The Supervisor | 3 |
| 2.1.4 Storage | 3 |
| 2.1.5 Input and Output | 4 |
| 2.2 The Main Store | 4 |
| 2.3 Storage of information in Atlas | 4 |
| 2.4 Instructions in the machine | 5 |
| 2.5 The written form of Instructions | 6 |
| CHAPTER 3 THE ACCUMULATOR | 7 |
| 3.1 Floating-point numbers | 7 |
| 3.2 The Accumulator | 7 |
| 3.3 Standardised numbers | 8 |
| 3.4 Fixed-point numbers | 9 |
| 3.5 Rounding | 9 |
| 3.6 Floating-point operations | 9 |
| 3.7 Standardising and Rounding accumulator instructions | 10 |
| 3.8 The timing of instructions | 10 |
| 3.9 Some fixed-point instructions | 11 |

| | Page |
|--|-----------|
| CHAPTER 4 THE B-REGISTERS | 12 |
| 4.1 General Purpose B-registers | 12 |
| 4.2 Arithmetic Operations | 12 |
| 4.3 Logical Operations | 13 |
| 4.4 Test Instructions | 14 |
| 4.5 Special Purpose B-registers | 15 |
| 4.6 Modification/Counting Instructions | 17 |
| 4.7 The B-test register | 18 |
| 4.8 The Shifting Instructions | 19 |
| 4.9 The Odd/Even Test Instructions | 19 |
| 4.10 Restrictions on the use of B81-B119 | 20 |
| | |
| CHAPTER 5 ROUTINES AND DIRECTIVES | 21 |
| 5.1 Routines, Subroutines and Symbolic Addresses | 21 |
| 5.2 Parameters | 21 |
| 5.3 Preset Parameters | 22 |
| 5.4 Global Parameters | 23 |
| 5.5 Optional Parameter Setting | 23 |
| 5.6 Expressions | 23 |
| 5.7 Separators | 24 |
| 5.8 The Special Parameter * | 25 |
| 5.9 The Ba and Bm Parts of an Instruction | 25 |
| 5.10 Half-Words, Six-Bit Words and Characters | 26 |
| 5.11 Floating-Point Numbers | 27 |
| 5.12 Library Routines | 28 |
| 5.13 Directives | 28 |
| | |
| CHAPTER 6 THE REMAINING ACCUMULATOR INSTRUCTIONS | 31 |
| 6.1 Standardised Unrounded Operations | 31 |
| 6.2 Unstandardised Instructions | 32 |
| 6.2.1 Storing and Loading the Accumulator | 32 |
| 6.2.2 Unstandardised Multiplication | 33 |
| 6.2.3 Division with Remainder | 33 |
| 6.2.4 Miscellaneous | 35 |
| 6.3 Test Instructions | 35 |
| | |
| CHAPTER 7 EXTRACODE INSTRUCTIONS | 37 |
| 7.1 Introduction | 37 |
| 7.1.1 Uses of the Extracode Instructions | 37 |
| 7.1.2 A-type and B-type Extracodes | 37 |
| 7.2 The Logical Interpretation of Extracode Instructions | 37 |
| 7.3 Allocation of Functions | 38 |

| | Page |
|-----------|--|
| 7.4 | The Accumulator Extracodes 39 |
| 7.4.1 | The Most Used Arithmetic Functions 39 |
| 7.4.2 | Other Floating-Point Arithmetic Functions 39 |
| 7.4.3 | Accumulator functions suitable for Fixed-Point Working .. 41 |
| 7.4.4 | Double-Length Arithmetic 42 |
| 7.4.5 | Arithmetic Using the Address as an Operand 42 |
| 7.4.6 | Complex Arithmetic 43 |
| 7.4.7 | Vector Arithmetic 43 |
| 7.4.8 | Half-Word Packing 43 |
| 7.5 | B-register Arithmetic 44 |
| 7.5.1 | General B-register Operations 44 |
| 7.5.2 | Character Data Processing 45 |
| 7.5.3 | Logical Accumulator Instructions 45 |
| 7.6 | Test Instructions 46 |
| 7.6.1 | Accumulator Test Instructions 46 |
| 7.6.2 | B-register Test Instructions 46 |
| 7.7 | Subroutine Entry 47 |
| | |
| CHAPTER 8 | INPUT AND OUTPUT 48 |
| 8.1 | Introduction 48 |
| 8.2 | Selecting Input and Output 48 |
| 8.3 | Input and Output Routines 48 |
| 8.3.1 | Input 48 |
| 8.3.2 | Output 49 |
| 8.3.3 | Further details of L1 and L100 50 |
| 8.4 | Internal Code 51 |
| 8.5 | Page Layout 52 |
| 8.6 | Character Input and Output Routines 53 |
| 8.7 | Input and Output Extracodes 54 |
| | |
| CHAPTER 9 | MAGNETIC TAPE 57 |
| 9.1 | Introduction 57 |
| 9.2 | Variable-Length Working 57 |
| 9.3 | Variable-Length Instructions 59 |
| 9.3.1 | Start and Select Instructions 59 |
| 9.3.2 | Transfer and Organizational Instructions 60 |
| 9.3.3 | Example 62 |
| 9.3.4 | Efficiency of Variable-Length Working 62 |
| 9.4 | Block Transfers 62 |
| 9.4.1 | Block Transfer Instructions 63 |
| 9.4.2 | Use of Block Transfers 63 |
| 9.4.3 | Reading Orion Tapes 64 |
| 9.5 | Specification of the Atlas Tape System 64 |
| 9.5.1 | Control 64 |
| 9.5.2 | The Tape Layout 65 |
| 9.5.3 | Performance 65 |
| 9.5.4 | Safeguards 65 |

| | Page |
|---|---------|
| 9.6 Organizational Instructions | 65 |
| 9.6.1 Mount Instructions | 66 |
| 9.6.2 Other Organizational Instructions | 66 |
| CHAPTER 10 DOCUMENTS AND JOB DESCRIPTIONS | 68 |
| 10.1 Introduction | 68 |
| 10.2 Headings and Documents | 68 |
| 10.3 Job Descriptions | 68 |
| 10.3.1 Job Description - Input | 69 |
| 10.3.2 Job Description - Output | 69 |
| 10.3.3 Job Description Combined with Program Document | 71 |
| 10.3.4 Job Description Combined with Data Document | 71 |
| 10.3.5 Job Description - Tapes | 72 |
| 10.3.6 Job Description - Further Facilities | 74 |
| 10.4 End of Tape Markers | 75 |
| CHAPTER 11 ADVANCED PROGRAMMING | 77 |
| 11.1 Programmed Drum - Transfers | 77 |
| 11.2 Optimization of Program Loops | 79 |
| 11.2.1 Store Access | 79 |
| 11.2.2 The Overlapping of Instructions | 80 |
| 11.3 Branching | 81 |
| 11.3.1 Existing Parallel Operations | 81 |
| 11.3.2 The Branch Instructions | 81 |
| 11.3.3 The Use of Branching | 82 |
| 11.3.4 An Example of Branching | 82 |
| 11.3.5 Store Requirements | 85 |
| 11.4 Monitoring and Trapping | 85 |
| 11.4.1 Types of Program Fault | 85 |
| 11.4.2 The Trapping Vector | 87 |
| 11.4.3 Restart Procedures | 87 |
| 11.4.4 Private Monitor | 88 |
| 11.4.5 Standard Post-Mortem | 88 |
| 11.4.6 Example | 89 |
| 11.4.7 Monitor Extracodes | 92 |
| APPENDIX A REFERENCES | 95 |
| APPENDIX B NOTATION | 96 |
| APPENDIX C V-STORE ADDRESSES OF PERIPHERALS | 99 |
| APPENDIX D CHARACTER CODES | 100 |
| APPENDIX E SUMMARY OF EXTRACODES | 104 |
| APPENDIX F SUMMARY OF BASIC INSTRUCTIONS | 110 |

Chapter 1

Automatic Digital Computers

1. INTRODUCTION

Atlas 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.

1.1 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.

1.1.1 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.

1.2 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 .

1.3 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.

1.4 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.

1.4.1 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.

1.4.2 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 sum or average.

1.5 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".

1.5.1 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 Computer ✓

2.1 A General Description of Atlas

The main parts of Atlas 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 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 (μ s); 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 30,000 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 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.

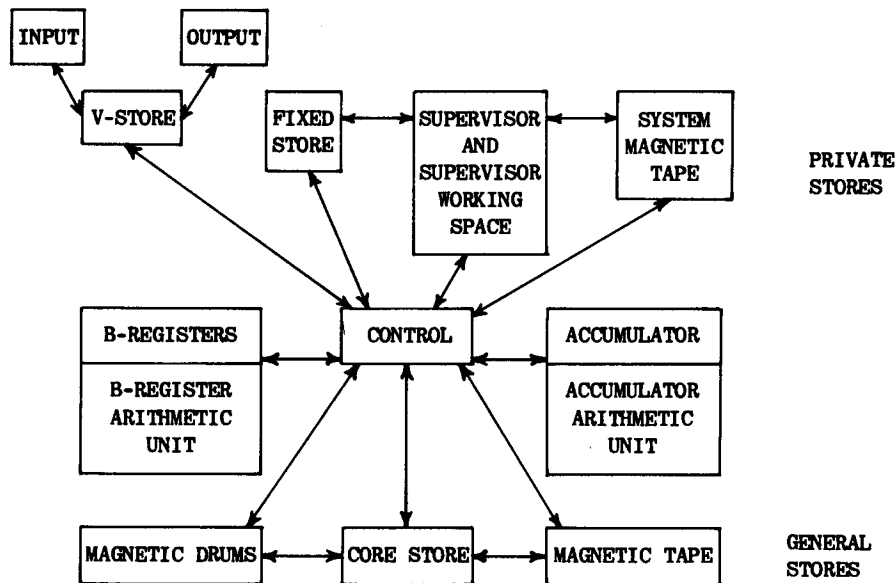
2.1.4 The main store on Atlas consists of core store in units of 8192 words and magnetic drum store in units of 24,576 words. A total of about one million words are directly addressable. Up to 32 magnetic tapes can also be used as a backing store. The main store can consist of core store and drums in 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 "extracodes". 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. 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.

| | |
|---------------------------------|----------------|
| Ferranti TR5 paper tape readers | 300 ch/sec |
| Ferranti 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 |
| Rank-Xerox Xeronic printers | 3000 lines/min |
| Graphical outputs | |
| Clock | |

The following diagram shows the component parts of Atlas:-



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.

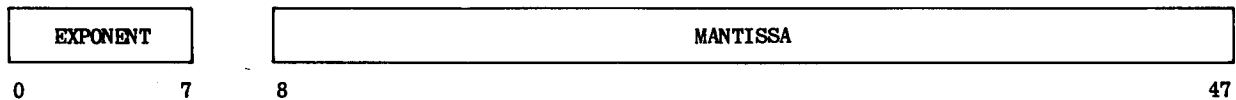
Wherever it is safe to do so, as soon as the control unit has decoded the odd address instruction it sends for the next pair. Thus, there is overlap between the execution of instructions and it is in fact possible for the computer to be in different stages of execution of up to five instructions. As 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

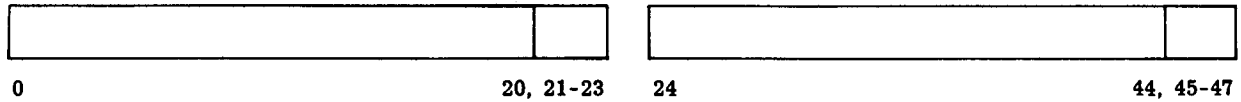
An Atlas register, or word, contains 48 binary digits. These are conventionally numbered from 0 (most-significant) to 47 (least-significant).

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

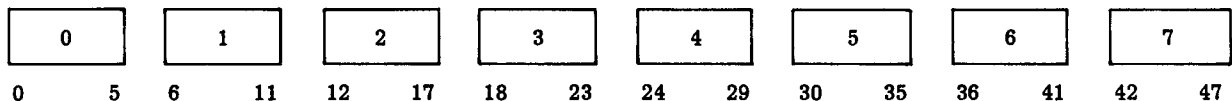
(a) A 48-bit floating or fixed point number, with the 8 most-significant bits representing the exponent and the other 40 the mantissa.



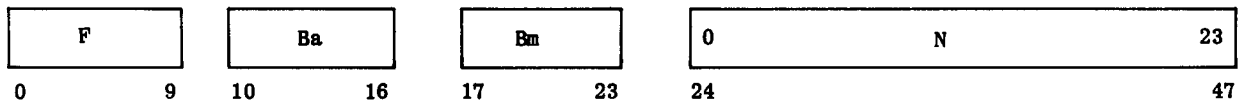
(b) Two 24-bit half-word numbers. These are taken usually as 21-bit signed integers in digits 0-20, with an octal fraction in digits 21-23



(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).

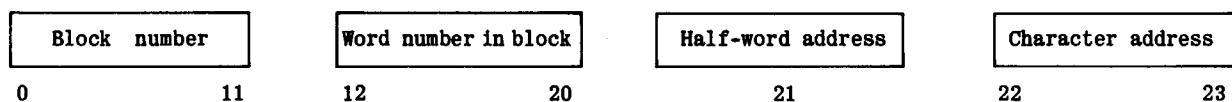


Throughout this manual the binary digits of a word are numbered from the most-significant end, starting with bit 0. The engineers' numbering system is the reverse of this, and is used in some documents describing basic programs such as the Supervisor and Engineers' programs: in these documents bit 0 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 0-20 specify a word in the 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 which require one type of number 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 0-11 specify the block.

Thus the address consists of



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 |

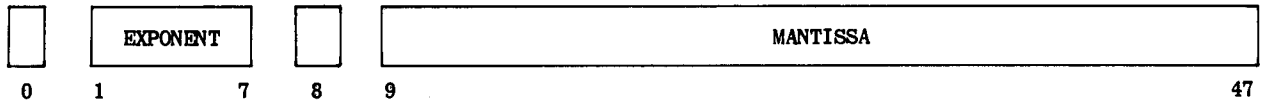
Instructions are read into Atlas through the media of punched paper tape, either of 5-track or 7-track width, and 80-column punched cards. 7-track paper tape (the most common input medium) is prepared on a Flexowriter, 5-track paper tape on a Creed teleprinter and cards on a card punch. These three equipments 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.

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 000010 | 0.00100000..... 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 ay and a double-length mantissa ax , of 78-bits and a sign bit. The mantissa Ax 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 A_m and A_l .

A_m consists of A_y and M
 A_l consists of A_y and L with Ls

There are a further two digits to the left of the sign of A_m . 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.

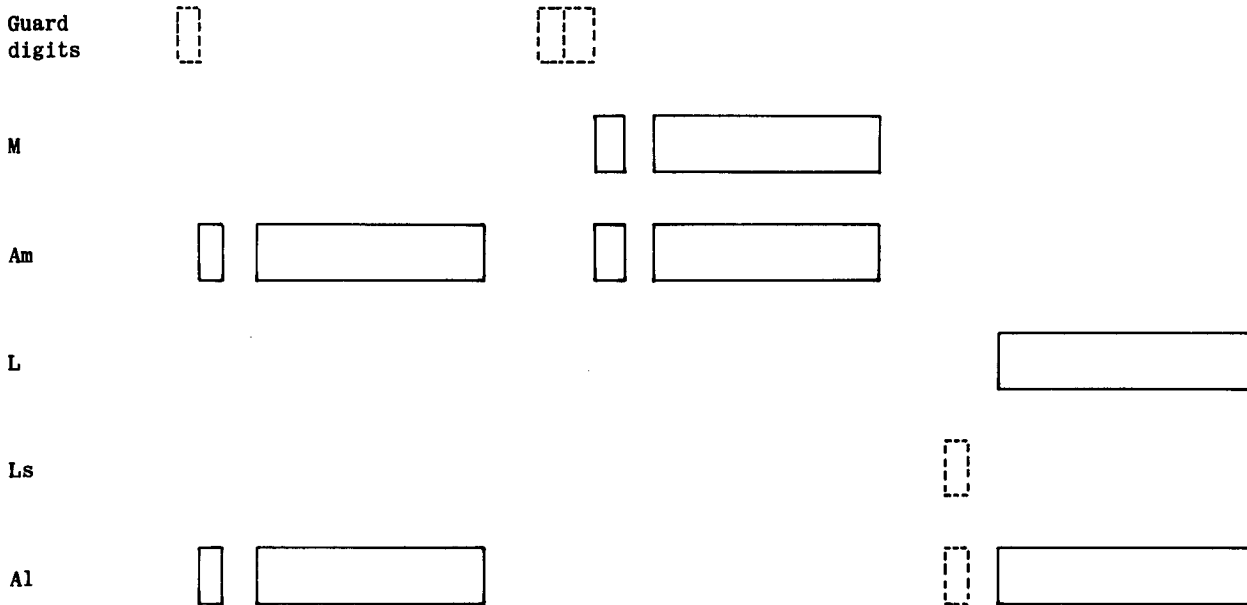
The exponent ay is held in the special B-register B124, which consists only of nine digits at its most-significant end, the other digits 9-23 being zero. The exponent is held in digits 1-8. Digit 0 is used as a guard digit, to detect if the number departs from the range $-128 \leq ay \leq 127$. Digit 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 of its permitted range. If $d_1 = 1$ and $d_0 = 0$, then $a_y > 127$; this is called exponent overflow. If $d_1 = 0$ and $d_0 = 1$, then $a_y < -128$; this is called exponent underflow. Special action is taken if either of these cases occurs (see section 4.6).

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 superstandard.

| | | |
|--|-----------|--------------|
| $\frac{5}{8} = \frac{5}{8} \times 8^0$ | 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 -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 (AO) 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 AO.

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 half-way 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.

ay 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 ay, 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. | $am' = am + s \text{ QRE}$ |
| 321 | As 320 but subtract s | $am' = am - s \text{ QRE}$ |
| 322 | As 320 but first negate the contents of A | $am' = -am + s \text{ QRE}$ |
| 324 | Transfer the floating point number in S to Am and standardise it | $am' = s \text{ Q}$ |
| 325 | As 324 but transfer negatively and check for exponent overflow | $am' = -s \text{ QE}$ |
| 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 | $am' = am.s \text{ QRE}$ |
| 363 | As 362 but multiply negatively | $am' = -am.s \text{ QRE}$ |
| 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 instruction takes about $1.6 \mu s$. If modified by bm, the time is $1.9 \mu s$, and if modified by bm and ba the time is $2.6 \mu s$. The times for multiplication and division are $5.0 \mu s$ and about $20 \mu s$ 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 $2 \mu s$.

As alternate addresses are in the even and odd stacks, 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 μ s 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 μ s. 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 H of the operand. The contents of this are known as h.

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

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 \bar{n} 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 \neq 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 \vee , 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 h placing the result in Ba | $ba' = ba \& h$ |
| 106 | Non-equivalence ba with h, placing the result in Ba | $ba' = ba \neq h$ |
| 147 | OR ba with h, placing the result in Ba | $ba' = ba \vee h$ |
| 117 | As 107, but placing the result in H | $h' = h \& ba$ |
| 116 | As 106, but placing the result in H | $h' = h \neq ba$ |
| 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 | $ba' = ba \neq n$ |
| 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 7 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 J771₄ 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 J7777777 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 | $ba' = ba + (bm \& n)$ |

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 $bm \neq 0$, $ba' = n$ |
| 216 | If bm is greater than or equal to zero, place n in Ba | If $bm \geq 0$, $ba' = n$ |
| 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 | <u>121</u> | <u>127</u> | <u>0</u> | <u>100</u> | if not, a < 0, b ≥ 0 so jump to 100 |
| 5 | 216 | 127 | 2 | 102 | if a ≥ 0, b ≥ 0, jump to 102 |
| 6 | <u>121</u> | <u>127</u> | <u>0</u> | <u>103</u> | 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.

| Digits 0-15 | Input to B123 | | | | | | | Output from B123 | | | | | | |
|-------------|---------------|----|----|----|----|----|----|------------------|------|----|----|----|----|-------|
| | 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 |

When bits 16 to 23 of the input to B123 are zero, the output is 8.0 i.e. a single digit 1 in bit 17

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, and indeed to all the B-registers after B80.

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 only consists of 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 ≠ 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 \equiv 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~~ ^(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 |

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 ≠ 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 ≠ 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 | $bt' = ba - s$ |
| 150 | Set Bt by writing s minus ba to it. s and ba are unchanged | $bt' = s - ba$ |
| 172 | Set Bt by writing ba minus n to it. | $bt' = ba - n$ |
| 170 | Set Bt by writing n minus ba to it. | $bt' = n - ba$ |
| 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 $ba' = n$ |

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.

Examples

1. 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 ≠ 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 | $ba' = 2^6ba + s$, (circular shift) |
| 125 | Shift ba as in 105, then add n | $ba' = 2^6ba + n$, (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 bm is odd, $ba' = n$ |
| 211 | If the least-significant bit in Bm is a zero, place n in Ba | If 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.

Examples

1. 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 c3 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 B81-B119

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 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 | Links to library |
| B91-97 | Extracodes |
| B98-99 | The Logical Accumulator |
| B100-110 | Supervisor |
| B111-118 | Interrupt routines |
| B119 | Extracode operand address |

Provided no reference is made to library routines, B81-90 may be freely used, and similarly B91-99 may be used whenever extracodes are not required; it is never safe for an ordinary program to use B100-119, since an interrupt can occur at any time and cause control to be transferred to the Supervisor.

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.

5.2 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

Preset parameters are distinct from routine parameters, and are intended for use by the program as a whole.

One hundred 'preset parameters' (numbered 0-99) are allowed and these are not associated with any particular routine. They can only be set by directive equations, must be set before they are referred to, and can be unset by further directives.

Preset parameters are identified by the letter P followed by a number.

P3 = 7 set P3 to +7

The directive U can be used to unset one or more preset parameters, which may thereafter not be referred to until they are set once more.

```
UP3 unsets P3
UP3-6 unsets P3, P4, P5 and P6.
```

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, probably 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.

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 taken modulo 2^{12} , and 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 octal digits assembles these from the most-significant octal position and right-hand zeros may be omitted.

K followed by a string of 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.

(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 0'

sets b2 = -0.1.

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 no 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^{20} and the octal fraction is rounded towards 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 = 2.5 Result rounds down
 J001 X J001 = 0 $2^{30} = 0$, modulo 2^{20}
 ~~$Y_i \times Y_j = 0$~~
 ~~$G_i \times G_j = 0$~~ Result rounds down to zero
 17 Q 3 = 5 Result rounds down
 14 Q 3.4 = 4 Exact

X and Q use the extracodes 1302 and 1304 which are described later.

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.

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 section 5.9.)

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 instruction 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

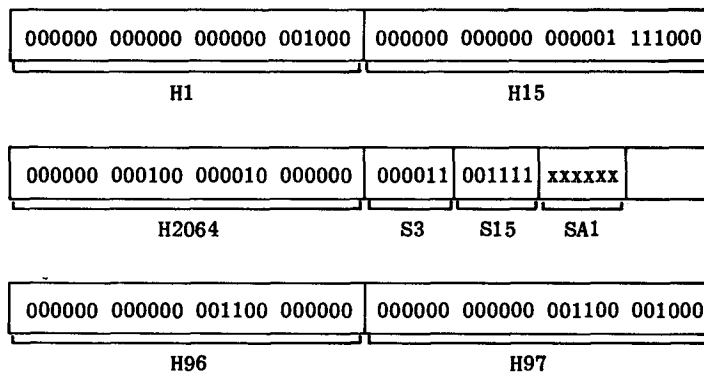
```

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 successive words of the store as:-



However, for clarity numbers should not be mixed in this way.

A facility also exists to read in characters from the input media and store them in a form ready for output, in fact in the machine internal code which is described later. This allows sub-titles, headings, comments etc. to be output at suitable times in the execution of the program. The string of characters is introduced by the letter C which has the following effect.

All subsequent characters up to and including new-line are ignored, then all following characters up to but not including the next new-line are stored in the 6-bit internal code in the next available character locations. The first character only may be labelled, by writing a label before the C. The last character can be labelled by a directive on the line after the characters.

See amendments (page 4)

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 ..

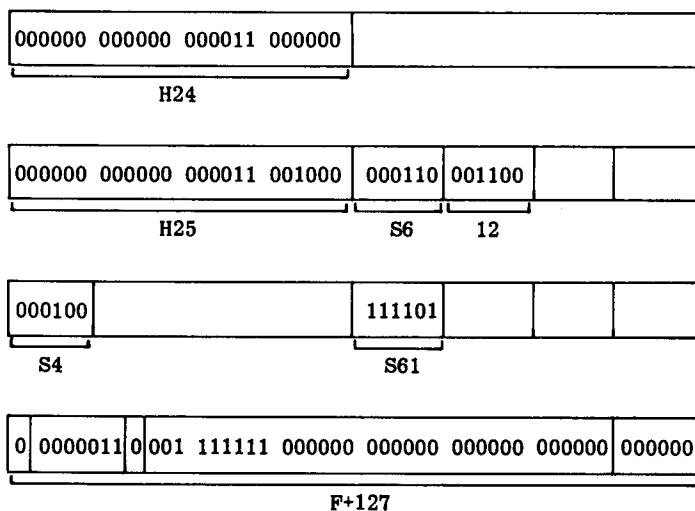
Any floating-point number can be labelled, and more than one may appear on a line. Floating-point numbers can also be mixed with half-words and six-bit words on a line, provided that the first of a group on a line is preceded by the directive F.

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



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

5.12 Library Routines

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

To avoid confusion with programmers' routines, library routines are referred to by the letter L followed by a decimal number.

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 the L.

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

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 an * directive and then writing an L-directive, for example

* = A2 } Read L10 into addresses
L10 } from A2 onwards.

In a large program it is very occasionally convenient to have more than one copy of a particular library routine. To allow this and to allow references to any particular copy the directive is written as La.b where b is also a decimal number. Then the copy of library routine a is identified as copy b.

Thus also Aa/Lc.d label a of copy d of library routine c.

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 *, and of preset parameters (0-99). They are of the form

Parameter = Expression

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

Parameter ? = Expression.

'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 [and], 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. (Note that 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).

Examples

[a > b]

[a > [b - c] > d]

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

Note: | π $\&$ [] and < > are directives and not terminators. They must not occur other than after correctly terminated items.

C directive

The directive C on one line introduces 6-bit characters on the following line.

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, location $A3 + 9$ 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 suspended and not entered.
- (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.

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

L_n , where n is a decimal number, calls for a copy of library routine n to be inserted at the program location indicated.

R. Routine directives

R_n , 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 T_a or T_{a-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 , T_a or T_{a-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.

UP. Unset Parameters directive

UP_n , where n is a decimal integer, causes the preset parameter P_n to be unset. Further, UP_{n-m} unsets from P_n to P_m 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 $am' = a$ QRE
and check for exponent
overflow.

6.1 Standardised Unrounded Operations

In these instructions, L is first cleared and 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 $a' = a + s$ QE
(pseudo double-length) (if $a_y \leq s_y$)
311 Subtract s from a $a' = a - s$ QE
(pseudo double-length) (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 $am' = am + s$, l spoiled. QE
311 $am' = am - s$, l 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

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

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

The 355 instruction is provided to position a_l correctly.

355 Copy the special sign bit of L $a' = a_l.8^{-13}$ Q
(ls) into all bits of M, then $\left\{ \begin{array}{l} a' = a - am \\ \text{when } ls = 0 \end{array} \right\}$ Q
standardise.

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 al *
356  0    0    101  store al.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 al
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      a' = a   QE
      exponent overflow
342  Multiply am by s, leaving the  a' = am.s  QE
      double-length product
      standardised in a
343  As 342 but multiply negatively a' = -am.s  QE
366  Clear L, complement am if     a' = |am|  QE
      negative, and standardise
367  Clear L, copy the modulus of  a' = |s|  QE
      s to Am, and standardise

```

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 instructions which store and load a are described below:-

```

356  Copy am into S                s' = am
357  Copy al (that is l, ls and ay)
      into S                       s' = al
346  Transfer am to S and clear A   s' = am, a' = 0
      (a' = floating-point zero)
347  Transfer al to S and clear L   s' = al, l' = 0
      and Ls

```

- 344 Copy the argument and sign from S into L and Ls, leaving Am, including the exponent, unchanged $l' = sx$
- 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. $l' = sx, n' = \text{sign of } s, ay' = ay$
- 314 Copy s into Am leaving L and Ls unchanged $am' = s, l' = 1$
- 315 Copy s negatively into Am leaving L and Ls unchanged (AO will be set for $s = -1.0$) $am' = -s, l' = 1$ 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 $a' = am.s$
 $ls' = 0$ EAO
- 373 As 372 but multiply negatively $a' = -am.s$
 $ls' = 0$ 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 $a' = am.s$
 $ls' = \text{sign of } m$ EAO
- 353 As 352 but multiply negatively $a' = am.s$
 $ls' = \text{sign of } m$ 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 $s_x = c \times 8^{-n}$, where n is normally 12 or 13, and an exponent s_y , usually +0 or +n. Provided that they are positive and that the divisor has $s_x > \frac{1}{2}$, 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 $ax < |s_x|$ provided that $s_x \neq 0$. 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 $0 \leq ax < |s_x| < \frac{1}{2}$; the remainder will then lie in the range $0 \leq m' < |s_x|$.

$l' = a/|s|$ E
 $m' = \text{remainder}$
 $(0 \leq ax < |s_x| < \frac{1}{2})$
 $0 \leq m' < |s_x|$
 $ay' = ay - sy$

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 A5. 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 | $ax' = 8^{-13} (8^{-12} c)$ (i.e. $m' = 0$ and $l' = 8^{-12} c$) |
| 375 | 0 | 0 | 1A5 | $l' = 8^{-13} c/d$ $m' = 8^{-12} \times \text{remainder}$ |
| 121 | 124 | 0 | 12U12 | $ay' = 12$ |
| 356 | 0 | 0 | 8 | Remainder am to location 8 |
| 364 | 0 | 0 | J4 | $l' = 8^{-12} c/d$ |
| 357 | 0 | 0 | 7 | Quotient al 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 $m'' = m' + 1.0$. | $al' = a/ s $ E $m' = \text{"remainder"}$ $(0 \leq ax < sx < 1)$ True remainder $m'' = m'$ if $m' \geq 0$ $= m' + 1.0$ if $m' < 0$ $0 \leq m'' < sx $ |
|---|--|

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 |
|---------------------------------------|-----------------------|
| $ sx \leq ax < 8 sx $ | $ay - 12$ |
| $0 \leq ax < sx < 1.0$ | $ay - 13$ |
| $sx = -1.0$ 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 A1 and the "remainder" in M. The dividend a must not be negative and the divisor s must be standardised before the 376 instruction is obeyed. The "remainder" is such that: 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 $ | $al' = a/ s $ ($a \geq 0$). EDO Remainder $m'' = m'$ if $m' \geq 0$ $= m' + 1.0$ if $m' < 0$ |
|-----|---|--|

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.a.l' 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.a.l' 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 A1 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. | $al' = 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 ax (including the sign of L) is zero | $ba' = n$ if $ax = 0$ |
| 235 | Place n in Ba if ax (including the sign of L) is non-zero | $ba' = n$ if $ax \neq 0$ |
| 236 | Place n in Ba if ax is greater than or equal to zero | $ba' = n$ if $ax \geq 0$ |
| 237 | Place n in Ba if ax is less than zero | $ba' = n$ if $ax < 0$ |

Note: 236 and 237 test the most-significant guard digit and not the sign digit. With standardised numbers this is immaterial as the guard digits will be copies of the sign bit, and with fixed-point working the correct result might still be obtained even if accumulator overflow had occurred.

Examples

- 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 |

- B1 and B2 contain positive integers n1 and n2. Form $n1 \times n2$ 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 $m' = \text{sign}$ of $n1 = 0$. $ax' = n1 \times 8^{-25}$ |
| 375 | 0 | 0 | 7 | $l' = (n1/n2) \times 8^{-13}$, $m' =$ remainder $\times 8^{-12}$ |
| 356 | 0 | 0 | 6 | store remainder am |
| 364 | 0 | 0 | J4 | shift up quotient to integer position. $l' = (n1/n2) \times 8^{-12}$ |
| 357 | 0 | 0 | 7 | store quotient $l = (n1/n2) \times 8^{-12}$ |
| 101 | 2 | 0 | 6.4 | set $b2' = \text{remainder}$ |
| 101 | 1 | 0 | 7.4 | set $b1' = \text{quotient}$ |

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 of the four octal digits in the 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 8 and 10).

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 $ba' = ba \cdot 2^n$ (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 $bt \neq 0$, $b121' = b121 + 0.4$ 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.

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

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.

Section 7.4 describes the accumulator arithmetic extracodes and section 7.5 describes the B-register arithmetic extracodes. The actual number of basic instructions obeyed in each extracode routine is given in the right-hand column.

The non-arithmetic extracodes will be described in later chapters.

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

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 |
| 1720 | Place the arc sine of s in Am. | $am' = \arcsin s$ | |
| 1721 | Place the arc sine of aq in Am. am' is in radians in the range $-\pi/2 \leq am' \leq +\pi/2$. | $am' = \arcsin aq$ | |
| 1722 | Place the arc cosine of s in Am. | $am' = \arccos s$ | |
| 1723 | Place the arc cosine of aq in Am. am' is in radians, in the range $0 \leq am' \leq \pi$. | $am' = \arccos aq$ | |
| 1724 | Place the arc tangent of s in Am. | $am' = \arctan s$ | |
| 1725 | Place the arc tangent of aq in Am. am' is in radians and such that $-\pi/2 < am' < \pi/2$. | $am' = \arctan aq$ | |
| 1726 | Divide aq by s and place the arc tangent of this number in Am. am' is in radians and such that $-\pi \leq am' \leq \pi$. | $am' = \arctan (aq/s)$ | |
| 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. | $a' = \text{int pt } s$ | QE | 5 |
| 1705 | Place the integer part of a in A. | $a' = \text{int pt } a$ | QE | 4 |
| | See also 1300 and 1301. | | | |
| 1706 | Set $a' = +1, 0$ or -1 as $s >, =,$ or $<$ zero. | $a' = \text{sign } s$ | Q | 5-6 |

| | | | | |
|------|--|-----------------------------|-----|-----|
| 1707 | Set $a' = +1, 0$ or -1 as $a >, =,$ or $<$ zero. | $a' = \text{sign } a$ | 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 . | $am' = 1/s$ | QRE | 4 |
| 1715 | Place the reciprocal of am in Am . | $am' = 1/am$ | QRE | 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 . | $am' = s/am$ | QRE | 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. | $am' = am/s$ | QRE | 10 |
| 1775 | Divide aq by s and place the result in Am . The original numbers need not be standardised. | $am' = aq/s$ | QRE | 9 |

1774 and 1775, besides providing a division instruction which operates on unstandardised numbers, store information which enables extracodes 1776 and 1477 to calculate an ~~integral~~ 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: | $s' = \text{quotient}$ $am' = \text{remainder}$ | QE | 13 |
|------|--|--|----|----|

Place the ~~integral~~ quotient of the
previous division in s and the
remainder in Am , where the remainder
has the sign of the divisor.

| | | | | |
|------|---|------------------------------------|----|--|
| 1477 | As 1776 except that the ^{the quotient is integral} in the instruction ^{adjusted integral} specifies the sign of the remainder ^{quotient} as follows: ^{protein} | $s' =$ $am' = \text{remainder}$ | QE | |
|------|---|------------------------------------|----|--|

and is adjusted according to the sign of the remainder, which is specified by ba as follows:

- 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 $s_0 + s_1 \cdot am + s_2 \cdot am^2 + \dots s_{ba} \cdot am^{ba}$ 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 $s^*y = 0$, $s_x > 0$ and $s^*x > 0$, then s' will be a PRN in the range 0 to 8^{s_y} , rectangularly distributed and fixed-point (i.e. s_x' is a fixed-point PRN and $s_y' = s_y$). a' will be a PRN in the range 0 to $s^*x \cdot 8^{s_y}$ (with $a_1' = s'$).
 - (b) If $s^*y = 0$, $s_x < 0$ and $s^*x > 0$, then as (a) except that the ranges become -8^{s_y} to 0 and $-s^*x \cdot 8^{s_y-1}$ 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 $s_y = s^*y = 0$, 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 $-\log x \cdot \delta x$ of being in the neighbourhood δx of x .

In all cases the generation process must be started with S_x 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 . | $m' = ax \cdot 8^{12}$ $ay' = ay - 12$ | A0 | 10 |
| 1753 | Shift m down 12 octal places in ax and increase ay by 12. | $ax' = m \cdot 8^{-12}$ $ay' = ay + 12$ | A0 | 6 |
| 1755 | Force ay to the number ny given in bits 0-8 of n , shifting ax up or down accordingly. | $ax' = ax \cdot 8^{ay-ny}$ $ay' = ny$ | A0 | 17 |
| 1762 | Shift ax up 12 octal places leaving ay unchanged. | $m' = ax \cdot 8^{12}$ $ay' = ay$ | A0 | 9 |
| 1763 | Shift m down 12 places in ax , leaving ay unchanged. | $ax' = m \cdot 8^{-12}$ $ay' = ay$ | A0 | 5 |
| 1764 | Shift ax up n octal places, leaving ay unchanged. If n is negative, shift ax in the opposite direction. | $ax' = ax \cdot 8^n$ $ay' = ay$ | A0 | 17 |
| 1765 | Shift ax down n octal places, leaving ay unchanged. If n is negative shift ax in the opposite direction. | $ax' = ax \cdot 8^{-n}$ $ay' = ay$ | 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 . | $m' = (m \cdot sx) \cdot 8^{12}$ $ay' = ay + sy - 12$ | A0 | 11 |
| 1773 | Divide a by s , and force ay equal to 12, shifting the result, which is in M , if necessary. | $m' = (ax/sx) \cdot 8^{ay-sy-12}$ $ay' = 12$ | 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. | $ax' = m \cdot sx \cdot 8^{ay+sy-bay}$ $ay' = bay$ | 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. | $ax' = (ax/sx) \cdot 8^{ay-sy-bay}$ $ay' = bay$ | 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 $\frac{am}{sx} = 0$ 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 $ry = ay - 13$. | $C(ba)' = \text{quotient } (am/s)$ $am' = \text{remainder } (am/s)$ | DO AO E | 20-29 |
| 1475 | Divide a by s. The exponents of q and r are given by $qy = ay - sy$ and $ry = ay - 13$. | $C(ba)' = \text{quotient } (a/s)$ $am' = \text{remainder } (a/s)$ | 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 $ am < 8^{24} sx $ 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 true integral part is used). | $C(ba)' = q \left(\frac{\text{int pt } am}{s} \right)$ $am' = r \left(\frac{\text{int pt } am}{s} \right)$ | 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.

| | | | | |
|------|---|----------------|-----|----|
| 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 |

(Note that putting $Ba=0$ will avoid the need to set ba , since ba is always zero)

| | | | | |
|------|--------------------|--------------|-----|----|
| 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 1477 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. 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 . | $ca' = +\sqrt{s}$: | ≤ 117 |
| 1411 | Place the argument of s : (radians) in Am . | $am' = \arg s$: | |
| 1412 | Place the modulus of s : in Am . | $am' = \text{mod } s$: | ≤ 53 |
| 1413 | Form the numbers $s \cos s^*$, $s \sin s^*$ and place these in Ca . (s^* is in radians). | $ca' = s \cdot \cos s^*$, $s \cdot \sin s^*$ | 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 : | $ca' = ca \cdot s$: | 18 |

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 .

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 . | $s_1' = am \cdot s_2$ | $10 + 4n$ |
| 1433 | Multiply s_2 by am and add this to s_1 . | $s_1' = s_1 + am \cdot s_2$ | $10 + 5n$ |
| 1434 | Copy s_2 to s_1 | $s_1' = s_2$ | $13 + 3n$ |
| 1436 | Form in Am the scalar product: $s_{10} \cdot s_{20} + \dots + s_{1(n-1)} \cdot s_{2(n-1)}$, where $s_{10}, s_{11}, s_{12}, \dots,$ $s_{1(n-1)}$ are the numbers in s_1 , and s_{20}, s_{21}, \dots are the numbers in s_2 . | $am' = \sum_{i=0}^{n-1} s_{1i} \cdot s_{2i}$ | $10 + 5n$ |
| 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 H into the accumulator, without standardising. | $a' = h$ | | 6 |
| 1626 | Copy a_y and the 16 most-significant digits of a_x into H after rounding this number in A_m by forcing a one in its lowest bit if the rest of a_x is non-zero. | $h' = a_m$ | R | 8 |

7.5 B-Register Arithmetic

7.5.1 General B-register Operations. 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 towards zero, and overflow of the answer is not detected. The accumulator is used in the calculation, but a_m is preserved.

| | | | | |
|------|---|---|--|-------|
| 1302 | Multiply ba by n and place the result in B_a . | $ba' = ba \times n$ | | 23-24 |
| 1303 | Multiply ba negatively by n and place the result in B_a . | $ba' = -ba \times n$ | | 22-23 |
| 1304 | Divide ba by n . Place the integer quotient in B_a and the remainder, which has the sign of the dividend, in B_{97} . | $ba' = \text{int pt } (ba/n)$ $b_{97}' = \text{remainder}$ | | 25-28 |

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

| | | | | |
|------|---|---|--|-------|
| 1312 | Multiply ba by n and place the result in B_a . | $ba' = ba \times n$ | | 23-24 |
| 1313 | Multiply ba negatively by n and place the result in B_a . | $ba' = -ba \times n$ | | 22-23 |
| 1314 | Divide ba by n . Place the integer quotient at the least-significant end of B_a and the remainder, which has the sign of the dividend, as a 21-bit <i>24-bit</i> integer with an octal fraction in B_{97} . | $ba' = \text{int pt } (ba/n)$ $b_{97}' = \text{remainder}$ | | |

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. | $ba' = ba \cdot 2^{-n}$ | | 10-22 |
| 1341 | Shift ba arithmetically to the left by n places. | $ba' = ba \cdot 2^n$ | | 9-21 |
| 1342 | Shift ba circularly to the right by n places. | $ba' = ba \cdot 2^{-n}$, circular shift | | 10-19 |
| 1343 | Shift ba circularly to the left by n places. | $ba' = ba \cdot 2^n$, circular shift | | 10-18 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 h , and place the result in H. | $h' = ba \vee h$ | | 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 3). | $b123' = n,$ then $ba' = b123$ | |
| 1356 | Set the B-test register as the result of non-equivalencing ba and h. | $bt' = ba \neq h$ | 7 |
| 1357 | Set Bt as the result of non-equivalencing ba and n. | $bt' = ba \neq n$ | 5 |
| 1376 | Set Bt as the result of collating ba and h. | $bt' = ba \& h$ | 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. | $ba' = (ba \& \bar{n}) \vee (bm \& n)$ [also $b119' = (ba \neq bm) \& n$] | 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. 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. | $ba' = \text{m.s. character of } g.$ $g' = 2^6g + n$ | 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 | $g' = g \neq s$ | 4 |
| 1607 | Collate s with g | $g' = g \& s$ | 3 |
| 1611 | Replace g by its logical binary complement. | $g' = \bar{g}$ | 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. | $g' = g \& \bar{s}$ | 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 (A0) is set. Clear A0. | $ba' = n$ if A0 is set. | 9 |
| 1201 | Place n in Ba if A0 is not set. Clear A0. | $ba' = n$ if A0 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 |
| 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 $ am \geq s$ | |
| 1737 | Increase main control by 2 if the modulus of am is less than s. | $c' = c + 2$ if $ am < s$ | |

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

7.5.2 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 ≤ 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.

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 |

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

121 127 B \bar{l} 0

where B \bar{l} is the address of the B register (Ba) in which the link was set. It is conventional to use B90 for this purpose.

Chapter 8

Input and Output

See "Input + Output"

8.1 Introduction

Atlas can have a great variety of peripheral devices, such as paper-tape readers for reading information and line printers for printing results. These peripheral devices are quite slow compared with the computing speed of Atlas, and a great deal of time would be wasted in waiting for these devices if programs were to control them directly. Instead, input information is read from peripheral devices, such as paper-tape readers and card readers, and stored in a standard form on magnetic tape. Only after all the information for a given job has been assembled on magnetic tape is the program for that job initiated. Similarly the program is not held up waiting for output devices: its output is stored in a standard form on magnetic tape and printed or punched when an appropriate output device is available.

During the first stage of input, described above, each paper-tape character or card column is normally converted to one six-bit character in Atlas Internal Code. Each character for output is also represented in this code. Equivalent characters on punched cards, line-printer output, and 5- or 7-track paper tape are all represented by the same internal-code character. Thus one input routine reading internal code can deal with data from punched cards or paper tape, provided that the printed form of the information is the same in each case. Similarly, one output routine can deal with punched-card, paper-tape or line-printer output.

8.2 Selecting Input and Output

The data for a given job may arise from several different sources and each such set of data may be prepared as a separate unit, henceforth to be referred to as a document, on a separate paper tape or pack of punched cards. Each document is allocated a distinct input number by a process to be described in Chapter 10. When the program requires information from a given document it must first identify that document by selecting the appropriate input number: this must be done by obeying extracode 1050 as described below. Similarly, before forming a document for output the program must obey extracode 1060 to select the appropriate output number. The Select extracodes are all singly-modified and are defined as follows:-

1050 Select Input n

All succeeding input operations (until the next 1050 instruction) refer to Input n. If no input document number n has been defined there will be an exit to the monitor routine (Chapter 10). If input instructions are obeyed without previously selecting an input, then Input 0 is used.

1051 Find Selected Input

ba' = number of currently selected input. This instruction may be used, for example, in a subroutine when it is required to read data from a different input and then restore the previous input selection.

1060 Select Output n

All succeeding output operations (until the next 1060 instruction) refer to Output n. Normally n will be written without an octal fraction (or with an even octal fraction), but if it is required to output in binary (Section 8.7) n should have an octal fraction .1 (or some other odd octal fraction). 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, then Output 0 is used.

1061 Find Selected Output

ba' = number of currently selected output (and octal fraction as in 1060).

8.3 Input and Output Routines

As input and output require fairly complicated programs, it is usually convenient to use subroutines for this purpose.

8.3.1 Input. L100 is a library routine which will read data one line at a time and then, on each entry, present the program with one number or character from the currently selected input. Its principal entry points are as follows, each referring to the currently selected input.

A1/L100 Read the next number to Am (standardised and rounded) leaving L clear. am' = Input QR+

| | | |
|---------|---|------------------------|
| A2/L100 | Read the next integer to B81, storing it as an integer in the most-significant 21 bits, with the 3 least-significant bits zero. | b81' = Input |
| A3/L100 | Read the next character to B81. | b81' = Input Character |
| A4/L100 | Lose the remainder of the current line. | |

In each case the link must be set in B90.

Example

Read ten numbers from Input 2 and store them in floating 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 next number to Am. |
| 356 | 0 | 3 | 49 | Store Number |
| 201 | 127 | 3 | A5 | Count Numbers |

8.3.2 Output. L1 is a library routine which will, on each entry, prepare one number or character to be printed on the selected output. Its principal entry points are as follows; each refers to the currently selected output.

| | | |
|-------|---|---|
| A1/L1 | Print the number a in style b89. | Output a |
| A2/L1 | Print the integer b81 in style b88. | Output b81 |
| A3/L1 | Print the character in B81. | Output Character b81 |
| A4/L1 | Terminate the current line, if any, and move on to the beginning of the next line. | End Line (One New Line) |
| A5/L1 | Terminate the current line, if any, and use b87 to determine the paper movement to follow it (Section 8.5). | End Record (Carriage Control in B87) |

It is of the more convenient and efficient to output character by using extracode 1064, to be described on page 56.

It is usually more convenient and efficient to terminate a line by using extracode 1065, (p. 56).

In each case the link must be set in B90.

Before entering L1 at A1 (or A2) the style of output must be set in B89 (or B88) by an instruction of the form

121 89 0 p:q.k

where p = Number of decimal digits before the point

q = Number of digits after the point

k = Style = 0 for standard style

When k = 0 the .k may be omitted.

q must be 0 when printing integers using entry A2, but may be 1 if it is desired to print the octal fraction.

Examples

1. Print the number in the accumulator correct to four decimal places, allowing up to five digits before the decimal point. Assume that the correct output has already been selected.

| | | | | |
|------|----|---|-------|-----------------------------|
| 121 | 89 | 0 | 5:4 | Set Print parameters in B89 |
| 1101 | 90 | 0 | A1/L1 | Enter L1 to print a |

2. Print the numbers x1 to x12 from locations 60 to 71 on Output 3. Print each number on a new line, and precede each by the appropriate suffix, 1 to 12. Allow four digits before the decimal point and five after for each x; two before the point for each suffix.

| | | | | |
|------|----|---|-----|-----------------------------|
| 1060 | 0 | 0 | 3 | Select Output 3 |
| 121 | 10 | 0 | 1 | Initial value of suffix = 1 |
| 121 | 89 | 0 | 4:5 | Set style for number print |
| 121 | 88 | 0 | 2:0 | Set style for integer print |

| | | | | | |
|----|------|-----|----|-------|----------------------------|
| 6) | 1101 | 90 | 0 | A4/L1 | Start New Line |
| | 121 | 81 | 10 | 0 | Copy suffix to B81 |
| | 1101 | 90 | 0 | A2/L1 | Print suffix |
| | 324 | 0 | 10 | 59 | Read number to A |
| | 1101 | 90 | 0 | A1/L1 | Print number |
| | 172 | 10 | 0 | 12 | bt' = b10 - 12 |
| | 221 | 127 | 10 | A6 | If bt ≠ 0, add 1 to suffix |

8.3.3 Further Details of L1 and L100

On output by L1, positive numbers are preceded by a space and negative numbers by a minus sign. Non-significant zeros on the left are replaced by spaces. Thus a typical line of output from the above program would be spaced as indicated in the diagram below, where each square represents one printing position on the line.

| | | | | | | | | | | | | | | |
|--|--|--|---|--|--|---|---|---|---|---|---|---|---|---|
| | | | 4 | | | - | 2 | 3 | . | 6 | 2 | 5 | 0 | 0 |
|--|--|--|---|--|--|---|---|---|---|---|---|---|---|---|

Before the integer, one space is left for a second digit and one for a sign. Before the number two spaces are left for two more digits. In order to space out the printing it is usual to specify more digits in front of the decimal point than are expected to be needed.

The style number k specified in B89 or B88 before using entry A1 or A2 of L1 is not always zero. The significance of different values of k can best be indicated by the following table.

- k = 0 Same Line, Signed, Fixed
- 1 Same Line, Signed, Floating
- 2 New Line, Signed Fixed
- 3 New Line, Signed, Floating
- 4 Same Line, Unsigned, Fixed
- 5 Same Line, Unsigned, Floating
- 6 New Line, Unsigned, Fixed
- 7 New Line, Unsigned, Floating

When a New Line is indicated, the previous line, if any, is terminated and one NL is output before the number.

When the number is to be signed it is preceded by SP or -; when it is to be unsigned the SP or - is omitted altogether, for example -2.5 would be printed as 2.5.

On entry A1/L1 with k odd, indicated by the word 'Floating' above, the number will be output as a floating-point number. The argument will be between 1 and 10, and will be printed with one digit before the decimal point and q digits after; it will be immediately followed by a signed decimal exponent. There will be p - 1 (≥ 0) spaces, and a sign when specified, before the first digit of the argument.

On entry A2/L1 with k odd, indicated by the word 'Floating' in the table above, b81 will be treated as a 24-bit integer instead of a 21-bit integer. In this case the octal point and digit will always be omitted whatever value of q is specified.

Numbers to be read by L100 are normally punched in the form:

| | | | | | |
|--------|------|---------|---|----------|------------|
| Layout | Sign | Integer | . | Fraction | Terminator |
|--------|------|---------|---|----------|------------|

The Layout, if any, may consist of Spaces, Tabs, Newlines, Paper Throws, Upper and Lower Cases, Carriage Returns, Line Feeds, Figure Shifts.

The + sign may be omitted for positive numbers.

The Integer may be omitted if zero.

The . Fraction may be omitted if zero.

The Terminator may be Space, Tab, Newline, Carriage Return Line Feed, Comma, Paper Throw or Line Feed.

Erases will be ignored throughout and back spaces will be correctly treated.

If desired a decimal exponent enclosed in round brackets may be inserted between the fraction and the terminator.

Other characters will be treated as faults, and dealt with in a manner to be described later (8.6). Decimal point and round brackets will also be treated as faults when reading integers using entry A2. (One octal digit may be read using entry A8, described below.)

L1 and L100 have several parameters which may be changed by the user if desired; details of these will be given in their specifications. The number of Inputs is of more general interest.

With L100 it is not permitted to have more than two Inputs active at any one time. An Input is active if part of a line remains to be read. If it is desired to read $n (> 2)$ Inputs at one time the program should include a directive $A23/L100 = n$.

A maximum line-length of 160 characters is assumed in L100, but this should be sufficient for most purposes.

Only the principal entries to L100 and L1 were given in section 8.3.1 and 8.3.2. For completeness other entries are listed below, but for full details the specifications should be consulted.

- A6/L100 **Read Text**
Read the next line and store it in half-words b89 onwards in a form suitable for text output (A6/L1).
- A7/L100 **Read the next integer to B81, storing it as a 24-bit integer.**
- A8/L100 **Read the next number to B81, storing it as a 21-bit integer with one octal digit after the point.**
- A6/L1 **Print Text**
Print the text stored in half-words b89 onwards. Location b89 must contain the count of the number of characters in the line.

8.4 Internal Code

As mentioned in section 8.1, information read into Atlas is stored in a special 6-bit internal code.

The number of characters available on paper tape is extended by having two sets (Upper Case and Lower Case on 7-track tape): the meaning of a paper-tape character depends on which set it is in. To represent the same number of characters in internal code it is again necessary to have two sets, called inner set and outer set, but the distribution of characters between the two internal code sets differs from that on paper tape. Most of the commonly used characters, including all punched-card characters, are included in the inner set. Every line commences in inner set and the characters "shift to outer set" and "shift to inner set" are used to change from one set to the other. An alternative representation used by the library routines L1 and L100 is to add an extra bit to the 6-bit internal code: in this 7-bit code the most-significant bit is 0 for inner set and 1 for outer set.

When a single 6-bit character is held in a 24-bit index-register it is usually stored in the 6 least-significant bits. When written in the address of an instruction it is therefore represented by two octal digits separated by a point, and this notation will be used throughout this chapter to represent 6-bit characters. The meaning of the characters in the six-bit internal code is given in the table on the following page.

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

- u All peripheral devices
- a Anelex Printer
- 7 7-track paper tape
- c Punched cards
- 5 5-track paper tape

Note: Characters in parentheses are alternatives available in the Commercial 7-track or 5-track paper-tape codes, as used on Orion and Pegasus.

Abbreviations:

- | | |
|-----------------------|-------------------------|
| BS = Back Space | PF = Punch off |
| *CR = Carriage Return | PM = Punch on |
| ER = ERase | *PT = Paper Throw |
| FS = Figure Shift | SI = Shift to Inner Set |
| FT = Fault | SO = Shift to Outer Set |
| LC = Lower Case | SP = SPace |
| *LF = Line Feed | ST = STop |
| LS = Letter Shift | TB = TaBulate |
| *NL = New Line | UC = Upper Case |
| | UL = UnderLine |

* Abbreviations marked with an asterisk are for page-layout characters, which are not represented in the internal code.

UC, LC, FS and LS need not be used when preparing output, nor are they stored on input when they are used to change from one case or shift to the other: the change of case merely alters the meaning and hence the internal code representation of succeeding characters. Redundant shift characters, such as FS when already in FS, are stored on input as they may indicate a fault.

On Output, an internal code character which cannot be represented by one character will be replaced by a full-stop on the Anelex printer or the card punch. This rule extends to non-printing characters, such as Back Space and Tabulate, which are only available on 7-track tape.

The Internal Code

| Internal Code | Character | | Internal Code | Character | | 75 |
|---------------|-----------|-----------|---------------|---------------|-----------|-----|
| | Inner Set | Outer Set | | Inner Set | Outer Set | |
| 0.0 | | | 4.0 | '(n) | u | |
| 0.1 | | SP | 4.1 | A | u | a 7 |
| 0.2 | TB | 7 | 4.2 | B | u | b 7 |
| 0.3 | BS | 7 | 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 | 4.6 | F | u | f 7 |
| 0.7 | | UC/FS | 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 | 5.4 | L | u | l 7 |
| 1.5 | & | a7 | 5.5 | M | u | m 7 |
| 1.6 | * | u | 5.6 | N | u | n 7 |
| 1.7 | / | u | 5.7 | O | u | o 7 |
| 2.0 | 0 | ú | 6.0 | P | u | p 7 |
| 2.1 | 1 | u | 6.1 | Q | u | q 7 |
| 2.2 | 2 | u | 6.2 | R | u | r 7 |
| 2.3 | 3 | u | 6.3 | S | u | s 7 |
| 2.4 | 4 | u | 6.4 | T | u | t 7 |
| 2.5 | 5 | u | 6.5 | U | u | u 7 |
| 2.6 | 6 | u | 6.6 | V | u | v 7 |
| 2.7 | 7 | u | 6.7 | W | u | w 7 |
| 3.0 | 8 | u | 7.0 | X | u | x 7 |
| 3.1 | 9 | u | 7.1 | Y | u | y 7 |
| 3.2 | < | a7c | 7.2 | Z | u | z 7 |
| 3.3 | > | u | 7.3 | 10 | a | |
| 3.4 | = | u | 7.4 | 11 | a | |
| 3.5 | + | u | 7.5 | 12 | a | |
| 3.6 | - | u | 7.6 | | | |
| 3.7 | . | u | 7.7 | FT | 7c | ER |

8.5 Page Layout

Page-layout characters are not included in the ordinary internal 6-bit code. Instead, each line of information is stored as a separate record and the last character of each record is the page-layout or carriage-control character. Input from paper tape or punched cards can give rise to the following four carriage-control characters:

- 2.1 NL (7-track tape) or End-of-Card
- 4.0 PT (7-track tape)
- 2.0 CR (5-track tape)
- 0.1 LF (5-track tape)

On output the carriage-control character is interpreted according to the following table:

| Character | 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 |

The carriage-control facilities, and hence the interpretations of the 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 the other facilities.

On the Anelex Printer

Line Feed and Paper Throw are always accompanied by Carriage Return.

On the Card Punch

Paper Throw is replaced by one Line Feed.

Line Feed means Next Card and is always accompanied by Carriage Return.

Carriage Return without Line Feed is ignored.

On 7-track Tape

Line Feed is always accompanied by Carriage Return.

Paper Throw is not accompanied by Carriage Return.

Carriage Return without Line Feed is ignored.

The Paper-Throw character on tape will only take effect if the Flexowriter on which it is printed has a paper-throw facility.

On 5-track Tape

Paper Throw is replaced by one Line Feed.

8.6 Character Input and Output Routines

It is often necessary to insert identifying characters between numbers in the data or in the results of a program. Such characters may be dealt with by Library Routines L1 and L100. These routines use the standard six-bit internal code supplemented by one extra bit to indicate inner set or outer set: in this seven-bit code the most-significant bit is 0 for inner-set characters and 1 for outer-set characters. Thus in this code 8 is added to the value of all outer-set characters, and the character [, for example, is represented by 10.1 instead of 2.1 in outer set.

Example:

Print the character > on the selected Output

```
121   81   0   3.3   Set > in B81
1101  90   0   A3/L1  Print >
```

On character input, using L100, the following characters have special values which differ from the standard internal code:

```
ER (Erase)      J4
FT (Fault)      J3
UL (Underline)  J2
FS (Figure Shift) J1
```

On output L1 will accept either these special values or the 7-bit code values (15.7, 7.7, 10.6 and 0.7).

An entry, A3/L100, is available for reading one character at a time. This reads the next character on the line into B81. If the end of the line has been reached b81 will be set zero and the carriage-control character will be in B82; the next entry A3/L100 will then proceed to the next line.

Reading characters one at a time is not a very convenient or efficient method of searching for special characters that may be punched with the data. This is best done by providing a special routine for L100 to enter if, when asked to read a number or an integer, it first encounters a character which is not normally acceptable during number input. The entry point of this special routine must be labelled 21/L100.

Example

Read the numbers on Input 2 until the character D is encountered, then calculate the sum of the numbers read and print it on Output 0.

```
1050  0   0   2   Select Input 2
324   0   0  1A5  am' = 0
```

| | | | | | |
|----------|------|-----|---|---------|------------------------------|
| 4) | 356 | 0 | 0 | A5 | C(A5) = Partial Sum |
| | 1101 | 90 | 0 | A1/L100 | am' = Next Number |
| | 320 | 0 | 0 | A5 | am' = Partial Sum |
| | 121 | 127 | 0 | A4 | Go to read next number |
| 5) | +0 | | | | |
| | +0 | | | | |
| 21/L100) | 172 | 81 | 0 | 4.4 | bt' = b81 - D |
| | 225 | 127 | 0 | A6 | Go to Fault Routine if not D |
| | 1060 | 0 | 0 | 0 | Select Output 0 |
| | 324 | 0 | 0 | A5 | a' = Sum |
| | 121 | 89 | 0 | 5:4 | |
| | 1101 | 90 | 0 | A1/L1 | Print Sum |

The routine labelled 21/L100 above will only be entered if the unacceptable character is separated by an acceptable terminator from the previous number. If the unacceptable character occurs in the middle of a number it will normally be treated as a fault; a routine labelled 22/L100 may be provided to deal with this case, but it will have to deal with a partially assembled number.

When data is prepared on 7-track tape it is possible to form compound characters, such as \neq by punching = BS /. When L100 encounters such a compound character it will store the two component 7-bit characters in the same half-word, with the larger of the two shifted up seven binary places. If the character is underlined then J2 will be added in to the half-word.

Example:

0 011 100 or 3.4 represents =

0 001 111 or 1.7 represents /

Shifting up =, which is larger in value, by seven binary places and adding /, the compound character obtained from = BS / or / BS = is

00 111 000 001 111

which may be written as K701.7 or J00007017.

Thus if the numbers in the previous example had been punched on 7-track tape and terminated by \neq , the instruction labelled 21/L100 would have read

172 81 0 K701.7

Note, however, that \neq is a single character (2.5 in outer set) when read from 5-track tape. Therefore the program should check for both K701.7 and 10.5 before treating the character as a fault.

Compound characters which include ER will always be stored as code J4. Compound characters comprising three characters other than underline will be packed as 7-bit characters in one half-word, but only the first two will be in order. A fault is indicated whenever more than three characters, other than UL or ER, occur in a compound character.

The output routine L1 will accept compound characters packed as described above, but they can be correctly output only on 7-track tape.

8.7 Input and Output Extracodes

L100 and L1 do not cope with the input and output requirements of all programs, and for some purposes it is necessary to have special input and output routines. These must use extracodes to read 6-bit characters from the selected Input or write them to the selected Output. It must be noted that as soon as L100 starts to work on a new line it reads in the complete line of Input. Therefore it is not permissible to use both L100 and the input extracodes to work on the same line of Input. However, one line of Output can be formed partly by L1 and partly by output extracodes. Normally, each character on the input or output device is represented by one 6-bit Internal Code character, but it may alternatively be represented by two 6-bit characters in Binary Code. In this Binary Code each binary digit of the paper-tape or card character is represented by the corresponding binary digit in the pair of 6-bit characters.

When reading data from 7-track tape using the input extracodes it must be remembered that the line reconstruction process, by which L100 analyses each line of input to form an image of the printed page, has not been done. Thus a character in one part of the record may be erased or altered by another character later in the record.

The 6-bit characters are grouped into records and, when using Internal Code, each record corresponds to one line of print or one punched card. Four 6-bit characters are packed into each half-word of the record, with the first character in the most-significant 6 bits of the first half-word. The last 6-bit character in each record is the carriage-control character, defining the page movement to be performed after the line has been printed. When using binary code, the whole binary section, or the whole of the one paper tape, is stored as one record with a zero carriage-control character.

On input, a data tape is read and stored in internal code until one of the directives ***B, ***F or ***G is read; thereafter it is read in binary. These directives will be described in more detail in Chapter 10. On output, as described in section 8.2, the least-significant digit in the address of the 1060 extracode selects between Internal and Binary Code.

The Select extracodes are defined in section 8.2; the remaining input and output extracodes are as follows; each is singly-modified and refers to the currently selected Input or Output as appropriate.

1052 Find Input Device Number.

ba' = V-store address of the peripheral equipment used for this Input,
= 0 if this Input originated as output from another program.
For input from 5-track tape the least-significant bit of ba' is 1,
otherwise it is 0.
The V-Store addresses are described in Appendix C.

1053 ~~1052~~ Test Binary.

ba' = n if the next input character is binary. Otherwise ba is unaltered.

1054 Read one Character/Test end of Record.

ba' = Next 6-bit input character.
b127' = n if the end of the record has been reached; ba' will then be the carriage control character. With binary input two uses of the 1054 extracode are required for each 12-bit binary character: the first reads the more-significant 6 bits and the second reads the less-significant 6 bits.

1055 ba' = Number of Blocks Read.

This sets in Ba the number of 512-word blocks of the selected Input. In internal code each block holds 4096 characters, but some of these are carriage-control characters and record counts. In binary code each block holds 4096 6-bit characters, but these represent only 2048 characters actually read from the Input Device.

1056 Read ^{characters!} ~~ba~~ half-words to S.

This reads up to ba half-words from the selected Input and places them in half-words S to S + ba - 0.4, there being 4 characters packed in each half-word. The sign bit of ba and the two least-significant bits of ba and of S are ignored. If the record is not more than ba characters in length (character count in bits 1 to 23 of Ba) then the end of the record will be reached. The sign bit of ba' will then be zero and bits 1 to 23 will be a count of the number of characters read. The last character in this case will be the carriage-control character.
If the end of the record is not reached, ba half-words (half-word count in bits 1 to 21 of Ba) will be read. The sign bit of ba' will be one and bits 1 to 23 will be a count of the number of characters read, which will always be a multiple of four.
The 1056 extracode transfers 24-bit half-words only. If the record has been partly read by use of extracode 1054 the first half-word is the one containing the next character, but that character is not necessarily at the most-significant end of the half-word.

See Amendments.

1057 Read next record to S.

This reads the next record and places it in the store starting at the half-word address specified in S.

On exit ba' will contain, in bits 1 to 23, a count of the number of characters read; the sign bit of ba' will be 0.

If the record has been partly read (by use of 1054 or 1056) the remaining part of the record is read. As with the 1056 extracode, transfers are of half words only and the same rule applies.

will be very inefficient if the no. of characters of the current record previously transferred is not a multiple of 4.

If any of the extracodes 1053, 1054, 1056 or 1057 is obeyed when there are no further characters to be read on the selected Input, the Monitor Routine is entered (Chapter 11).

Extracodes 1056 and 1057 use far fewer instructions per character than does 1054, and they should therefore be preferred for large amounts of input, *provided the transfer begins at a multiple of 4 characters from the beginning of the record.*

1062 Find Output Device Type.

ba' = V-store address of equipment number 0 of the peripheral type currently selected for output.
= 0 if the currently selected output is to any peripheral device.
The V-store addresses are given in Appendix C.

1063 Delete Output n.

This deletes any information previously sent to Output n, and prevents it being printed, provided that it has not been printed already.

1064 Write Character n.
This writes the character in the 6 least-significant address bits to the selected Output. As with the 1054 extracode, two uses of 1064 are required to write each 12-bit binary character.

1065 End this Output Record.
This writes the carriage control character in the 6 least-significant address bits to the 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 ignored at the time of printing and any character could be used.

1066 Write ^{Character} ~~ba half-words~~ from S.
This writes ba half-words starting at half-word S to the currently selected Output. If the sign bit of ba is 1 the record is not ended, the two least-significant bits of ba are ignored and the number of half-words specified in bits 1 to 21 of ba are transferred. If the sign bit of ba is 0 the record is terminated, the number of characters specified in bits 1 to 23 of ba are transferred and the last character, in address $S + ba - 0.1$, is interpreted as the carriage-control character. In both cases the two least-significant bits of S are ignored. If the record has already been started using extracode 1064 then the first transfer is to whichever half-word in the record 1064 would have used next. The previous 1, 2 or 3 characters may therefore be overwritten if the number of characters already written is not a multiple of four.

1067 Write a Record of ba characters from S.
This writes the record starting at half-word S to the currently selected Output. The number of characters in the record is specified in bits 1 to 23 of ba. The last character, in address $S + ba - 0.1$, is interpreted as the carriage-control character. The sign bit of ba and the two least-significant bits of S are ignored. ~~If the record has already been started using extracode 1064 then 1, 2 or 3 characters may be overwritten if the number of characters already written is not a multiple of four.~~

If output of the record has already been started, then the subsequent transfer by 1067 will be very inefficient if the number of characters previously transferred is not a multiple of 4.
Extracodes 1066 and 1067 use far fewer instructions per character than 1064 and they are therefore to be preferred for large amounts of output, *provided the transfer begins at a multiple of 4 characters from the beginning of the record.*

1070 Rename Output n as Input ba.
This enables information sent to Output to be read back by the same program as Input.

1071 Break Output n.
This extracode indicates that the information so far recorded on Output n may, if convenient, be treated as separate from any subsequent information sent to that Output. The preceding information ceases to be part of Output n after the Break and may be printed before the subsequent document on Output n is completed.

1072 Define Output n.
Normally, output documents should be defined by listing them in the job description (Chapter 10), but they may alternatively be defined using extracode 1072. Before obeying a 1072 instruction, ba must be set equal to the maximum number of 512-word blocks to be allowed on Output n, and ba* must define the type of output device to be used, in the same code as is used in 1062.

Special Note on Punched Cards

Note that when using Atlas internal code an 80-column punched card is represented by 81 six-bit characters, the last of which is the carriage control character. Thus, to punch all 80 columns using extracodes 1066 or 1067, it is necessary to specify 81 characters in Ba, which may be done by an instruction

121 Ba 0 81D3

Similarly, if it is desired to read a card using extracode 1056, ba should specify at least 81 characters. In binary, a punched card is represented by 161 six-bit characters.

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.

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 $1 \leq B \leq 99$. The tape number, B , is normally allocated to the appropriate tape by the Job Description, which will be described in Chapter 10.

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 . Thus, to search for section 8 on tape 4, we would write

```
1001  4    0    8
```

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 11, 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 $b121$.

9.2 Variable-Length Working

To simplify the writing of magnetic-tape programs, extracodes are provided which execute the transfer of variable-length records between magnetic tape and the main store. These extracodes 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:O.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.

```
1001  4  0  8      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:O.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:

```
1001  4  0  8      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.

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.

| | | | | | |
|----|------|-----|---|--------|---|
| | 1001 | 2 | 0 | 1 | Search for section 1, tape 2 |
| | 1001 | 3 | 0 | 1 | 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 A6 onwards |
| | 1040 | 4 | 0 | A6 | |
| | 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. |

9.3 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.

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 seven orders of marker, numbered 1 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.

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 operation: 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.

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 a search instruction; when starting to read variable-length records, this starting address must be the address of a marker at one end of a record. When using the start instruction merely to change from one mode of transfer to another, it is not essential to stop at a marker.

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 $K+1$ sections, or forwards from any of the last $K+1$ sections, will lead to an end-of-tape interrupt.

9.3.1 Start and Select Instructions

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.

- 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.
- 1033 **Select**
Select tape Ba for succeeding variable-length operations, in the mode specified by the preceding start instruction for that tape.

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.

9.3.2 Transfer and Organizational Instructions

Notation: b_w = Integral part of b_a (bits 1 to 20)
($0 \leq b_w < 2^{19}$)

b_k = Octal fraction of b_a .
($0 \leq b_k \leq 7$)

- 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 came immediately after word b_w .
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

$$S + b_w - 1, S + b_w - 2, \dots, S + b_w - b_w'$$

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

$$S + b_w' - 1, S + b_w' - 2, \dots, S.$$

- 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.

$b_k' = 0$ if no marker $\geq b_k$ was encountered.
 $= m$ if a marker $m (\geq b_k)$ terminated the transfer or came immediately after word b_w .
 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 ($1 \leq K \leq 7$) 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 B_a .

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.

1024 Where am I?

After variable-length transfers

$$s' = A/W$$

where W = Address within the section of the current marker on tape B_a or, if not on a marker, of the next word going forwards.

A = Address of the section containing word W .

1044 Word Search

Search for word W , section A of tape B_a , where $s = A/W$. Also, stop variable-length operations on tape B_a .

1001 Block Search

Search for word O , section n of tape B_a . Also, stop variable-length operations on tape B_a . Note that a search instruction, when required, must precede the start instruction for the tape to which it refers.

1036 Set ba' = number of selected tape

This places in B_a the program number of the tape currently selected for variable-length operations. If there is currently no tape selected, a negative number will be placed in B_a . One application is to enable a sub-routine to select a different tape for variable-length transfers, and the main program to then re-select the original tape.

1037 Store 'mode of tape B_a ' in H

The transfer mode at present selected for tape B_a will be indicated by placing in half-word H 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.

9.3.3 Example: 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.

| | | | | | |
|----|------|-----|----|--------|--|
| | 1001 | 1 | 0 | 1 | } Initiate variable-length operations on tapes, 1, 2 and 3 |
| | 1030 | 1 | 0 | 20:0.1 | |
| | 1001 | 2 | 0 | 1 | |
| | 1030 | 2 | 0 | 22:0.1 | |
| | 1001 | 3 | 0 | 1 | |
| | 1032 | 3 | 0 | 24:0.1 | |
| 4) | 1033 | 2 | 0 | 0 | } Read Amendment Record |
| | 121 | 12 | 0 | 50 | |
| | 1040 | 12 | 0 | 100 | |
| 5) | 1033 | 1 | 0 | 0 | } Read Main-File Record |
| | 121 | 11 | 0 | 50 | |
| | 1040 | 11 | 0 | 150 | |
| | 1033 | 3 | 0 | 0 | Select Updated File |
| | 101 | 10 | 0 | 150 | } Go to A7 if Amendment Key equals Main-File Key |
| | 102 | 10 | 0 | 100 | |
| | 214 | 127 | 10 | A7 | |
| | 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 |

9.3.4 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.

9.4 Block Transfers

The variable-length tape instructions are not appropriate for all purposes, and there is some loss of efficiency in using them. Block-transfer instructions are available which 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.

9.4.1 Block-Transfer Instructions. The block-search instruction, 1001, described in section 9.1, 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.

Examples:

1. Read section 19 of tape 3 to main-store block 6.

1001 3 0 19 Search for section 19
 1002 3 0 6: Read to block 6

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 10
 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

Note that 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.

Note also that 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.

9.4.2 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 11, 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. With block transfers, overlap can be obtained by transferring alternately to two different blocks, computing on one whilst transferring to the other.

For 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
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 sections
      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

```

It should be noted that, because it is reading two sections in advance, this program reads one section more than it requires, but it does not attempt to process the extra section. This extra read operation could be avoided by an extra test in this case, 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.....

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.

*See
Amendments*

9.4.3 Reading Orion Tapes. It is possible to read, into Atlas, tapes prepared on the Ferranti Orion Computer. This is accomplished by the use of the following two extracodes:-

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. $K + 1$ pages 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.

1047 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.

9.5 Specification of the Atlas Tape System

9.5.1 Control. An Atlas installation may have as few as 8 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.

9.5.2 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. Check-sums 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 2½ million 48-bit words.

9.5.3 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 mid-way 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.

9.5.4. 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 section 0 of the tape. Before using such a tape, a program or its job description must specify the correct title and allocate a number B ($1 \leq B \leq 99$) to the tape. A "Free" tape is one which is available for general use: it has no useful information on it and no special title.

9.6 Organisational Instructions

A number of organisational 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 of the tapes 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. This can be achieved by asking for a free tape and specifying that the title "FREE" should be written on it. This method suffers from the disadvantage that the tape cannot so easily be identified if a fault arises during the run and a restart is necessary. It is better to give the tape a unique title at the start and then to free it just before the 1117 (end program) instruction. Tapes may also be freed by means of a steering tape, but this requires an extra run on the machine.

Many of the organisational instructions require to refer to the title stored on section 0 of a magnetic tape. To do so they require to keep a copy of the title and refer to it by its main store address. The title must be stored as one record in a similar form to that described in Chapter 8: the six-bit characters are packed eight to a word and the last character is a zero character.

9.6.1 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 titles of 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 will be given in Chapter 10.

If further magnetic tapes are required after a program has been entered, they may be called in by obeying "mount" instructions, but 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 two 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 mount instructions are as follows:-

1010 Mount

Allocate the number Ba to the tape whose title is stored in locations S onwards. If this tape is not already available, instruct the operator to mount it on any available tape mechanism.

1011 Mount Free

Allocate the number Ba to a free tape and write on section 0 of that tape the title stored in locations S onwards. If no free tape is available, instruct the operator to mount one on any available tape mechanism.

1007 Mount Next Reel of File

Allocate the number n to the next reel of file Ba. If this tape is not already available, instruct the operator to mount it on any available tape mechanism.

The following mount instructions refer to logical channel number K of a given program. These logical channel numbers do not refer to a fixed magnetic-tape channel: they are merely a device to enable the program to separate onto 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 groups.

When a program refers to logical channel number K, it has the following effect. If K has been previously defined by the program, the new tape is mounted on the same channel, if possible. If K has not been previously defined, the new tape is mounted, if possible, on a channel different to any which has previously been defined; that channel is then designated logical channel K of the given program.

1012 Mount on Channel K

Allocate the number Ba to the tape whose title is stored in locations S onwards. If this tape is not already available, instruct the operator to mount it, if possible, on channel K of this program.

1013 Mount Free on Channel K

Allocate the number Ba to a free tape and write on section 0 of that tape the title stored in locations S onwards. If no free tape is available, instruct the operator to mount one, if possible, on channel K of this program.

9.6.2 Other Organizational Instructions

1014 Write Title

Assign to tape Ba the title stored in locations S onwards. This title will be written on section 0 of the tape when it is dismounted.

1015 Read Title

Store in locations S onwards the title of tape Ba. This title will be that read from Section 0 when the tape was mounted - unless a 1014 extracode has since been used, in which case the newly assigned title will be stored.

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?**
 h' = 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.
- 1024 **Where am I?**
 $h' = A$; $h'' = W$ ($0 \leq W \leq 511$).
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.

Thus, the job description will begin:-

JOB
Title
e.g. JOB
F6479/2, J. SMITH, FERRANTI, LONDON: 1962 BEETLE SURVEY

which will be followed by the sections described in detail below.

10.3.1 Job Description - Input. This section begins with the word

INPUT

which is followed by a list of titles of data tapes used in this job, each preceded by the number by which they are referred to in the program. For example, if a program operates on two data documents which it refers to as inputs 1 and 2 respectively, the job description would contain

INPUT
1 (Title of data 1)
2 (Title of data 2)

With the examples of data given above this becomes:-

INPUT
1 F6479/2, BEETLE SURVEY DATA 1962
2 F6479, BEETLE POPULATION 1961

The document headed "F6479/2, BEETLE SURVEY DATA 1962" could then be selected by the programmer by means of the order

1050 0 0 1

The program document itself would appear in this list as data 0

e.g. 0 F5, SURVEY ANALYSIS Mk 2

and the complete input section would be:-

INPUT
0 F5, SURVEY ANALYSIS Mk 2
1 F6479/2, BEETLE SURVEY DATA 1962
2 F6479, BEETLE POPULATION 1961

10.3.2 Job Description - Output. This section of the job description specifies the type of peripherals to be used for output. It begins with the word

OUTPUT

which is followed by a list of output mechanisms each preceded by the programmer's number. Types of equipment are:-

LINE PRINTER
TELETYPE
CARDS
FIVE-HOLE TELETYPE
ANY

where "teletype" means a 7-track paper-tape punch
"cards" means a card punch, and
"any" means output on a line printer or teletype punch.

For example, we might have

```

OUTPUT
0  LINE PRINTER
1  CARDS
2  ANY

```

To output on punched cards, for example, the programmer would then use the instruction

```
1060  0    0    1
```

When printed, the output information is preceded by

```

OUTPUT n
(Title of job)

```

where n is the programmer's number.

It should be noted that output 0 is used by the Supervisor and by the compilers for monitor output, but it is still available for normal output from the program. Also, the organisation of Atlas is such that it can readily accept two or more streams of output from a program destined for the same equipment, even though only one such device may exist. The streams are accumulated independently within the computer and eventually output one after another.

The specification of the type of equipment should normally be followed by a limit on the amount of output thus:-

```

OUTPUT
0  LINE PRINTER m BLOCKS

```

where m is the limit on the number of blocks, each of about 4000 characters, which will be produced for that output stream. If this limit is exceeded the program is stopped, in order to prevent peripherals being tied up, for longer than necessary, by useless information produced by a faulty program.

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.

Here then is an example of a complete job description.

```

JOB
F6480, STATISTICAL ANALYSIS OF METAL SAMPLES           - Job Title
INPUT
0  F6480, ANALYSIS PROGRAM                             - Program Title
1  F6480, IRON CONTENT                                 }
2  F6480, COPPER CONTENT                               } - Data Tapes
OUTPUT
0  LINE PRINTER 3 BLOCKS                               }
1  ANY                                                  } - Output Streams
***Z                                                    - End of tape marker
                                                    (see below)

```

The program would then be headed

```

COMPILER EMA                                           - (EMA = Extended
F6480, ANALYSIS PROGRAM                               Mercury Autocode)

```

supposing it to be written in Mercury Autocode, and the data tapes by

```

DATA
F6480, IRON CONTENT
and DATA
F6480, COPPER CONTENT

```

10.3.3 Job Description Combined with Program Document. A job description may be combined with a program to form one composite document. In this case the last item of the description will be followed by the heading

COMPILER (Program Language)

and then the program itself. No further title will be required. This will be the usual form for small programs. If this procedure is adopted, the item "input 0" of the job description may be omitted and the computer will then compile and execute the program immediately following the job description. If there are no separate data documents either, then the input section of the job description may be omitted completely.

Example 1

JOB
F916, J.BROWN, FERRANTI, LONDON: NUMBER FREQUENCY

INPUT
1 F916, TABULATED DATA

OUTPUT
0 LINE PRINTER

COMPILER ABL
followed by program written in ABL.

The data document would then be headed

DATA
F916, TABULATED DATA

Example 2

JOB
F916/2, J.BROWN, FERRANTI, LONDON: PRIME NUMBERS

OUTPUT
0 CARDS 2 BLOCKS

COMPILER ABL
followed by program.

10.3.4 Job Description Combined with Data Document. It is also possible to combine the job description with a data tape. This is particularly useful when the same program is to be run more than once, each time with different data. In this case the input section of the job description would include

SELF = n

where n is the programmer's number of the data following on the same document. The program itself is specified as input 0, in the same way as when the job description is a separate document.

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 the 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 tape.

The program would be headed by, say,

COMPILER ABL
F900, MONTHLY WAGE CALCULATIONS

and the fixed data document by

DATA
F900, PAYE CODES

The job description could then be combined with the second data document thus:-

```
JOB
F900, ABC, FERRANTI, LONDON: WAGES OCTOBER 1962

INPUT
0 F900, MONTHLY WAGE CALCULATIONS
1 F900, PAYE CODES

SELF = 2

DATA
```

followed immediately by the list of hours worked.

10.3.5 Job Description - Tapes. A programmer may use magnetic tapes directly in his program, by use of tape instructions as distinct from using tapes in connection with input or output. Information is stored on magnetic tape in blocks of 512 words. The title of the information on the tape is stored in block 0.

The word "FREE" indicates that the tape has no useful information on it and is available for general use.

Each tape required should normally be specified in the job description by two lines of printing

```
TAPE
n (The title stored on block 0 of the tape)
```

where n is the programmer's number of the tape

e.g. TAPE
1 AMENDMENTS F236/NPB

When a new tape is required by the program, the appropriate two lines in the job description are

```
TAPE FREE
n (Title)
```

In this case the title specified is written into Block 0 of a free tape by the computer

e.g. TAPE FREE
2 FILE F236/NPB

If the program requires more tape-decks than the number of distinct tapes listed, the total number of tape-decks used must be specified in the following way:

```
DECKS d
```

where d is the number of "private" tapes required by the programmer.

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

```
TAPE/m
n (Title)
```

where m is the number of the continuation, counting from 1 upwards. The programmer's number n will be the same for all m. The final tape of the file is entered as

```
TAPE/m END
n (Title)
```

For example

Suppose a file of information extends over three magnetic tapes which have respectively as titles in block 0:

```
F101 Birmingham Sales
F101 London Sales
F101 Manchester Sales
```

These would be referred to in the Job Description as

```

TAPE/1
1 F101 BIRMINGHAM SALES

TAPE/2
1 F101 LONDON SALES

TAPE/3 END
1 F101 MANCHESTER SALES

```

These tapes, regarded as a continuous file, will be referred to by the program as tape 1; only TAPE/1 will be mounted before the job begins.

If a programmer requires extensive input, he may wish to copy all input to a private magnetic-tape before starting his program. To initiate this copying process, the input is headed

```

COPY TAPE FREE
(Title)

```

and the title specified will be written on block 0 of a free tape followed by all input from the slow peripherals. If it is necessary to employ a previously used tape, the heading is

```

COPY TAPE (b)
(The title already on block 0 of this tape)

```

where b is the number of the tape block at which it is intended to begin copying input.

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

Thus, to use an input document from a tape, write

```

INPUT
TAPE a/b/c
(Title of Input Document)

```

To refer to a system tape, a = Sn, where n is the number of the system tape required.

To refer to a private tape, titled (ABC), a = n, the number of the private tape, ~~and the tape section must include~~

```

TAPE
n (ABC)

```

This private tape is created by means of a directive to the Supervisor, of the form

```

COPY TAPE FREE
(ABC) - Title of tape
DATA - Document
(Title of Input Document)

```

For example

If the data had been entered under the heading

```

COPY TAPE FREE
AMENDMENTS FCS/N6 1/1/62
DATA
F900 AMENDMENTS
(The data)

```

When this copying process is carried out the address of the document on magnetic tape will be printed out
The job description for a program to amend a file would read

```

JOB
F900, RN, FERRANTI, LONDON: UPDATE FCS/N6 1/1/62

```

in the form TAPE NUMBER/FIRST SECTION/FIRST WORD
 Eg. TAPE 306/1/0
 - The Title of the job

INPUT

TAPE 306/1/0

- Tape 306, section 1, word 0

1 AMENDMENTS FCS/N6 1/1/62

- specifies input tape, allots number 1

TAPE

2 FSC/N6 1/12/61

- names an existing file, allots number 2

TAPE FREE

3 FSC/N6 1/1/62

- allots number 3 to a free tape to be used for amended file

Note that the Amendments file is on tape no 306, is referred to as input doc. no. 1, whereas the other two files are tapes 2 & 3.
Similarly if a program involves extensive output this may be written to a private magnetic-tape. The output section of the job description is then

OUTPUT

n TAPE FREE/(type of equipment) m BLOCKS

(The title to be written on block 0)

where n, "The type of equipment", and m are as for direct output. To use a previously employed tape we substitute:

n TAPE b/(type of equipment) m BLOCKS

(The title on block 0)

where b is the number of the tape block at which the output document should start. The private tape can then be printed by a steering tape consisting of

PRINT TAPE

(The title on block 0)

if the whole tape is to be printed

or PRINT TAPE a/b/c

(title of output)

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

10.3.6 Job Description - Further Facilities

Further information may be given in the job description to indicate:

- (a) The time for which the program is expected to compute.
- (b) The amount of store used.

All these apply to the execution stage of the program, i.e. excluding input from slow peripherals, compiling and output to slow peripherals.

1. The computing time is specified in the job description by

COMPUTING p.q (HOURS
(MINUTES
(SECONDS

where p.q is a fixed-point decimal number as in the example below:

COMPUTING 7.5 SECONDS

The time stated is an outside estimate of the expected computing time of the program concerned. If the computing time exceeds this limit, the program is stopped. This is to prevent a faulty program wasting computing time. If the total execution time is significantly different from the actual computing time, because there is considerable tape waiting time, the actual execution time should also be specified.

e.g. EXECUTION 5 MINUTES
COMPUTING 30 SECONDS

2. The amount of core and drum store used is specified by

STORE S

where S is the maximum number of 512-word blocks of store in use within the program.

If the information is not supplied in the job description then

4 SECONDS computing time
20 store blocks

are reserved for the program. Estimates of computing and execution times are taken as being equal unless both are specified explicitly.

10.4 End of Tape Markers

The end of a section of tape is indicated by

* * * followed by one of Z, A, B, C, T, E, F, P.

The marker

- * * * Z indicates the genuine end of a tape or stack of cards.
- * * * A indicates "abandon previous incomplete documents if any"
(This may be required by a machine operator if part of a document is damaged before input is complete.)
- * * * B }
* * * E } all indicate that a binary tape follows (see below)
* * * F }
- * * * P Suppress Parity Checking
- * * * C indicates the end of a document and that another document follows on the same tape or in the same card stack.
- * * * T indicates a temporary stop within a document.

When the marker * * * Z is read, the peripheral equipment is disengaged by the computer. When the operator next engages this equipment, a new document, with the appropriate heading and title, is read.

When the marker * * * B is encountered, the computer reads the information following on the same document in binary, without testing for further tape markers.

On reading

* * * E

information in binary is read until one of the tape markers

* * * A, Z or C

is encountered, when the appropriate action will be taken.

* * * F causes the input mechanism to be disengaged and, when re-engaged, the reading of binary input continues to the physical end of the tape.

* * * P suppresses parity checking until one of the markers

* * * A, Z or C

is encountered. If the input is in internal code, wrong-parity characters are replaced by the fault code, 77 (octal).

The number of characters, n, on the binary document may be indicated in the case of B, E and F by

* n * * * B
* n * * * E
* n * * * P

The computer ^{then} ~~when~~ reverts to reading in internal code after n characters.

On reading

* * * C the computer notes the end of the document and continues reading the next section of the tape.

Finally on reading

* * * T

the equipment is disengaged as for * * * Z. However, when the operator next engages this equipment, a continuation of the current document, without a new heading, is read. This is useful if a document consists of two tapes or stacks of cards.

A better method of dealing with such a case if the document is a data list is by use of a modified "data" heading

DATA/n

where n is the number of the continuation of the section of data followed by the same title for all n. The last number of such a series must have the heading

DATA/n END

For example, if the data called F6480, IRON CONTENT is on two distinct paper tapes, the data may be headed

DATA/1

F6480, IRON CONTENT

and DATA/2 END

F6480, IRON CONTENT

each tape ending with *** Z. These tapes may be read into the computer in any order.

On punched cards, the three asterisks may be replaced by holes 7 and 8 punched in the first column of the card.

The standard method of reading cards uses the first column as an indication of the type of card which follows:-

if the first column is a standard code, then the contents of the card are converted to internal code.

if the first column is a non-standard code, then the card is assumed to be in binary.

Chapter 11

Advanced Programming

11.1 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. There therefore 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 can take up to 1.5 milliseconds to switch to another program and a further 1.5 milliseconds 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. The Supervisor endeavours always 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.

For the purposes we have discussed, sixteen extracodes are available 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 |

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.

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.

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.

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.

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.
- 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.**
Here K will be taken modulo 5 (so that $1 \leq K + 1 \leq 6$) since there are 6 sectors on a band of a drum. From the program logical band d, $K + 1$ blocks are transferred to the store blocks $P, P + 1, \dots, P + K$. This extracode may only be used after logical band d has been defined by the use of extracode 1174.
- 1176 **Write $K + 1$ blocks to band d.**
The $K + 1$ store blocks $P, P + 1, \dots, P + K$ are copied to logical band d. The store blocks involved are not made free. This extracode may only be used after logical band d has been defined by the use of extracode 1174.
- 1177 **Lose band D.**
The band of the drum store previously reserved as logical band D is freed and made available for general use.

11.2 Optimization of Program Loops

The following table gives the approximate times in microseconds currently being achieved on the Manchester University 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 | 2.6 |
| $am' = am \times s$ | 0, 1 or 2 | 5.0 |
| $am' = am/s$ | 0, 1 or 2 | 20.0 |
| $ba' = ba + h$ | 0 | 1.5 |
| | 1 | 2.3 |

- It is not possible to time a single instruction because, in general, this is dependent on
- the exact location of the instruction and operand in the store;
 - the instructions preceding and following; for wherever possible one instruction is overlapped in time with some part of three other instructions,
 - whether the operand address has to be modified,
 - for floating-point instructions, the numbers themselves.

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.

11.2.1 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 $2\mu s.$, that is, the time after reading or writing a number before another number can be read from or written to the same stack is $2\mu s.$

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 $2\mu s$ 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.

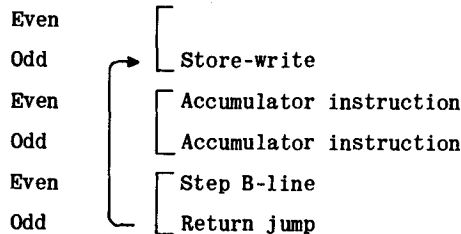
The Supervisor attempts to organise the store so that instructions and operands are placed in different pairs of stacks. On the Manchester University Atlas, wherever possible, instructions are kept in pages 0-15 and operands in pages 16-31. The programmer can assist the Supervisor to do this by using the extracodes described in 11.1. 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.

11.2.2 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 25% of the time, three for 56% and two for 19%.

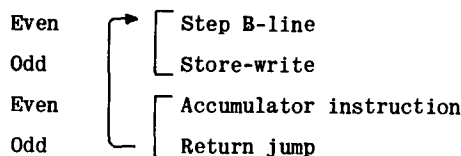
This overlapping is possible because the accumulator arithmetic, the B-register arithmetic, the function decoding, the B-store, and the main core-store are independent of each other to a large extent. A number of rules which enable the programmer to gain as much advantage as possible from the overlapping 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 established 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 accumulator 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.

Example 1 Odd number of instructions



Example 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.

11.3 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.

11.3.1 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.

11.3.2 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.

11.3.3 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 also 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.

11.3.4 An Example of Branching. 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.

Suppose, for example, that 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 |
| | 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 |
| 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 |
| | . | . | . | . | |
| | . | . | . | . | |

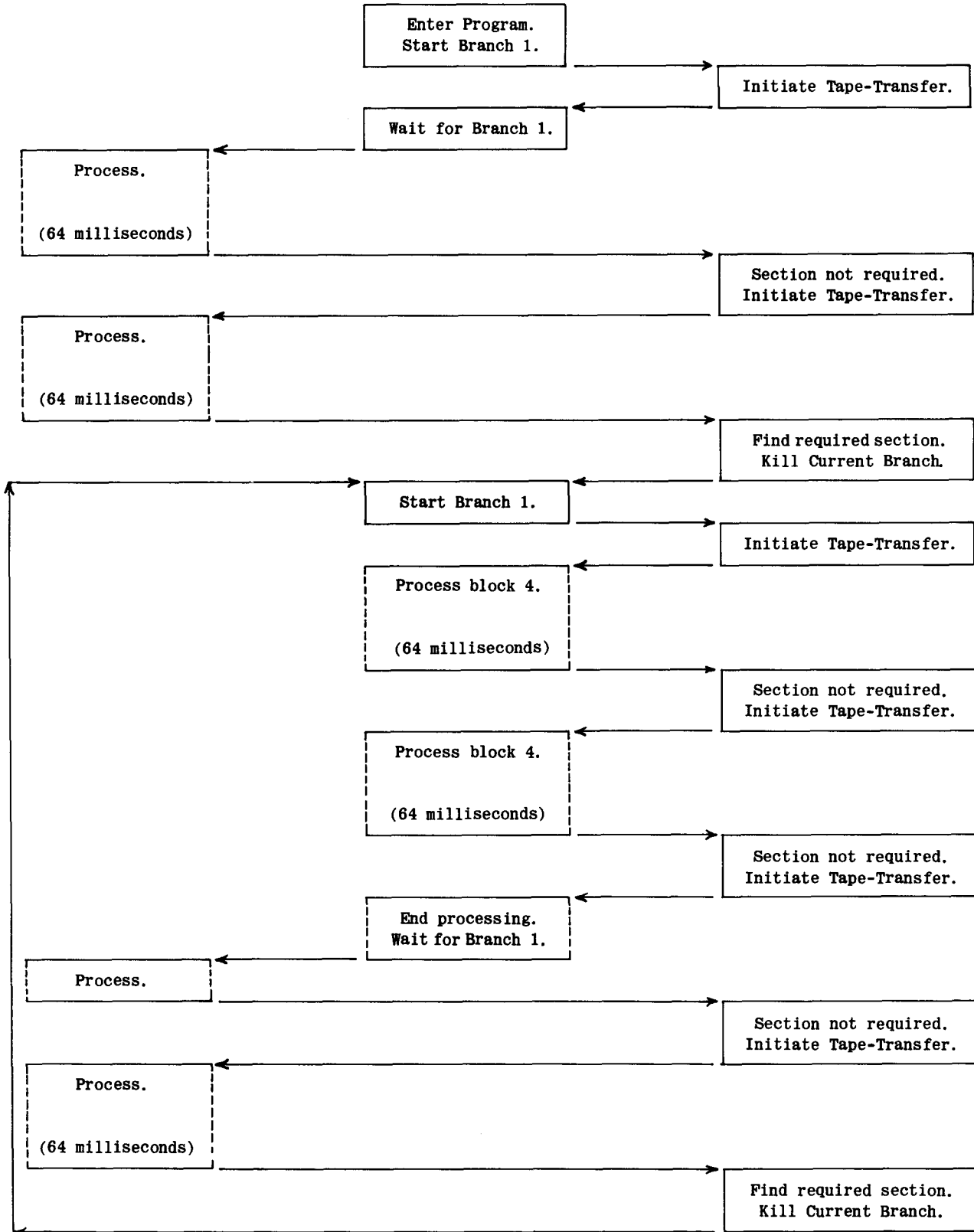
The chart below shows how control would pass from one branch to the other in a typical sequence of operations when the program is obeyed. Interruptions from the Supervisor and higher-priority programs have been excluded because they would complicate the chart without altering the sequence significantly. 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. The chart is not to scale, and it should be remembered that each entry to branch 1 takes only a few microseconds, whereas 64 milliseconds (the time required to read one tape section) must elapse between successive entries to branch 1.

See Chart on following page.

LOWER-PRIORITY
PROGRAM

BRANCH 0

BRANCH 1



11.3.5 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.

11.4 Monitoring and Trapping

When a program is run on the computer, its successful execution may be prevented by the occurrence of errors, which can be due either to mistakes in the program itself or to faults arising in the computer or the peripheral equipments. In Atlas, provision is made for the automatic detection of a variety of clearly defined fault conditions; these 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.

11.4.1 Types of Program Faults. We shall here concern ourselves 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 restart 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. Amongst these faults there are some which lead to the setting of an appropriate digit of B91, just as if they had been detected by hardware, as in (a) above; these include over-running of computing time or of tape waiting time, and attempts by extracode 1166 to transfer information from a drum to a locked-down page of the core store. 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 | Detected by | Mark or count in B91 | Trap number (if any) |
|---|-------------|----------------------|----------------------|
| Local time expired | S | Bit 5 | 0 |
| Division Overflow | I | Bit 6 | 1 |
| Exponent Overflow | I | Bit 14 | 2 |
| Page locked down | S | Bit 1 | 3 |
| Number of blocks exceeded | S | 2.0 | 4 |
| Square root argument < 0 | E | 2.4 | 5 |
| Logarithm argument ≤ 0 | E | 3.0 | 6 |
| SPARE | | 3.4 | 7 |
| Inverse trig. function | E | 4.0 | 8 |
| Reading after Input ended | S | 4.4 | 9 |
| End of magnetic tape | S | 5.0 | 10 |
| Variable-length record error | E | 5.4 | 11 |
| Magnetic Tape failures | S | 6.0 | 12 |
| Computer failures | S | 6.4 | 13 |
| Unassigned function | I | Bit 4 | |
| Sacred Violation Instruction | I | Bit 8 | |
| Sacred Violation Operand | I | Bit 10 | |
| Illegal block number | S | 9.6 | |
| Band not reserved | S | 10.2 | |
| Computing time expired | S | Bit 2 | |
| Execution time expired | S | Bit 3 | |
| Input not defined | S | 11.6 | |
| Output not defined | S | 12.2 | |
| Output exceeded | S | 12.6 | |
| Tape not defined | S | 13.2 | |
| Illegal search | S | 13.6 | |
| No selected tape | S | 14.2 | |
| No mode defined, or attempt to write when not permitted | S | 14.6 | |

I = Interrupt

E = Extracode

S = S.E.R.

Some of the entries in the table need further elucidation:-

Local time means that the local instruction counter has run out. (This may be set by extracode 1123; if not required it should be set to zero. Note that the Supervisor will override any attempt to set the local count to a figure in excess of the amount of allotted time still left.)

Page locked down refers to an attempt by extracode 1166 to transfer information to a locked-down page of the core store.

Number of blocks exceeded implies that the store assignment would be exceeded if the program were allowed to continue.

Reading after Input ended implies that the program is attempting to read from an input stream whose end has been reached.

End of tape refers to an attempt to read or write beyond the last addressed section, or into block 0.

Variable-Length Record Error means an attempt to transfer when not aligned on a tape marker, or an encounter with an incorrectly made marker.

Sacred Violation Instruction refers to a transfer of control to an 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.

Illegal search means a search for Section 0 of a tape, or for a non-existent section, i.e. a search beyond the last addressed section (section 5000 on a 3600 ft. tape).

11.4.2 The Trapping Vector. Upon entry, 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 previous value of main control and therefore the point in the main program at which the fault occurred.

Only some of the faults listed in the table on page 86 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: for example, a convenient way of ending an iterative loop may be the overflow of a local timer. 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 typed as part of the program, and the unwanted entries may be typed as +0 or merely omitted, as shown in the example in section 11.4.6. (Note that floating-point zero produces a negative first half-word, because its exponent is -128.)

It will be seen from the table that computer failures and magnetic tape failures can each be trapped: after entering such a trap routine, the extracode 1114 ("Exit from trap") must be used. The effect of this extracode is determined by the singly-modified value of n, as follows:

| | |
|------------|---|
| n < 0 | Enter monitor and end program. |
| n = 0 | Re-enter program. |
| n odd > 0 | Resume program at b127 = n. |
| n even > 0 | Recover extracode working registers, set b127 = n, and resume at current extracode. |

It is to be noted that re-entry (n = 0) is not permissible after an 'End of Magnetic Tape' fault. If the program is not in trap, the extracode 1114 will result in exit to the monitor.

11.4.3 Restart Procedures. Following a failure in the computer or in an on-line peripheral, there are various possible courses which events may take.

Firstly, it must be pointed out that if a disastrous failure occurs in the central computer, any further useful action may be precluded. However, assuming that the computer is still active after the failure, the system will, in the absence of contrary instructions by the program, do one of two things, depending upon the type of object program involved. If the program does not use magnetic tape and has

not been computing for very long, it will be recompiled and started again; otherwise, a complete dump will be made of the whole store allocation of the program, together with the associated Supervisor working registers, the accumulator and all the B-registers; details will be printed out on output stream 0 to enable the programmer to gain access to this information if required.

However, computer and tape failures are trappable faults; if the appropriate trap is set, control will be transferred to the private fault-routine, as described in section 11.4.2. Alternatively, if the trap is not set, control may be transferred to the private monitor routine, if one is provided, or otherwise to the standard monitor program.

Finally, there is available the 'restart' facility: in the case of a long program, or one using magnetic tape, it is advisable to arrange the program in such a way that it can be divided up into 'recoverable' sections - each of which avoids overwriting any of the data on which it operates; then, if a computer failure occurs, the program can be restarted at the beginning of the current section since all the data necessary will be preserved: normally it will be necessary to obey a short restart routine to set up the starting conditions required by that section. Before starting each recoverable section of the program, a 1113 instruction should be obeyed to specify the entry point for the appropriate restart routine.

1113 Set restart.

Preserve the Supervisor working registers associated with this program. In the event of a future restart, recover these working registers and re-enter this program at $b127 = n$. If n is negative, cancel the previous restart point without supplying a new one.

If a computer failure occurs after a 1113 instruction has been obeyed, the specified restart routine will be entered. Before entry to the restart routine, the input streams will be moved back to the positions they were in at the time the 1113 instruction was obeyed, and so also will the time counters. It should be noted, however, that if the input is being read by a routine such as L100, which reads one line at a time, the restart routine must arrange to dispose of any partially processed line: for example, it must enter at A4/L100 for each input being dealt with by L100. The output streams will not be moved back, and if output is involved, the restart routine should print some indication of the fact that a restart and duplication of output has occurred. The restart routine must select the appropriate input, output and magnetic-tape if required, and also search for the appropriate starting addresses on magnetic tapes.

To enable the restart routine to set up the appropriate starting conditions, some of the required information may have to be stored by the program just before obeying the 1113 instruction. Similarly, if a program cannot conveniently be divided into sections which preserve their data, the restart points may be established by storing the data just before the 1113 instruction and then restoring it in the restart routine. Care must be taken to ensure that the restart information for one restart point is not overwritten before the next restart point is established: thus consecutive restart points may not normally dump information in the same addresses. It should be noted that the Restart facility only applies if the main store is intact: to provide a major Restart facility which will work when a store parity failure has occurred would involve dumping the required parts of the store on magnetic tape before the 1113 instruction.

Final recommendations as to how the Restart facility should be used are not available at the time of writing. Further information will be published as soon as possible.

11.4.4 Private Monitor. When the monitor program encounters a fault which is not trapped, it prepares to terminate the program and proceeds to diagnostic printing; this consists of:

- (a) one line describing the fault,
- (b) standard post-mortem printing.

If he so desires, the programmer may supply a private monitor sequence, whose starting address must be specified by using the extracode 1112. The last octal digit of the starting address determines the time of entry to the private monitor as follows:

| Octal Digit | Entry |
|-------------|-------------------------|
| 1 | Before (a) |
| 0 | After (a) (normal case) |
| 2 | After (b) |

When the entry is before (a), B91 contains the record of faults and B92 the current value of main control; only B93 and B121 are 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.

11.4.5 Standard Post-Mortem. As stated above, the standard diagnostic print-out consists of one line describing the fault, followed by the standard post-mortem printing. This is normally printed on Output 0 and contains the following information:

LINE 1 The heading **INSTR**, followed by the last instruction but three. The address and contents of location b127-3 are printed in the form

Address F Ba Bm N

with F in octal, Ba and Bm as three digit decimal integers, and N as a signed decimal integer followed by a "point" and a single octal digit (which are omitted if zero).

Then, on the same line, the contents of Ba and Bm are printed as signed decimal integers followed by a single octal digit; a blank is left in the case of B0, or for any empty register. If b127-3 is either a location in the private store or an undefined location, the heading **INSTR** is followed by the word **UNALLOCATED**.

LINES 2 Repeats of line 1 for b127-2, b127-1, and b127 instead of b127-3. Thus the current instruction
3 AND 4 and the preceding three instructions in the store are printed, these being the ones most likely to have caused faults. Note, however, that these are not necessarily the last four instructions obeyed, and in some cases a jump may have occurred since the instruction causing the fault. Extracode faults will appear in line 3; block-addressing faults resulting in non-equivalence may appear in line 4.

LINES 5 Contents of B1 to B99, omitting those with zero contents. These are printed four to a line in
ONWARDS the form

B3 = signed decimal integer (. octal digit)

with the point and octal digit omitted if zero.

NEXT The heading **ACCUMULATOR**, followed by the single-length contents of the accumulator as an
LINE unstandardised, signed decimal fraction separated by an oblique stroke from the octal exponent. If exponent overflow has occurred, the overflow digit is ignored, but the sign digit is preserved for printing.

FINAL The positions of any magnetic tapes in use are listed on separate lines as
LINES

TAPE n AT (block-number)/(word-number)

If variable-length tape operations are not in use, the word-number is omitted.

If no private monitor printing is called for, the standard-post-mortem is concluded by the "END PROGRAM" sequence. This prints the number of instructions obeyed and the accumulated time of use of magnetic tapes. The quantity of output on each stream is printed at the end of the stream; the quantity and location of input is printed at the start of the program on output stream 0.

11.4.6 Example. To illustrate the various monitoring facilities, we give here an example of their use in a simple program for the updating of a magnetic-tape file. It is assumed that the file tape 2 and a free tape 3 have been mounted from the job description, and also that the amendments have been loaded on input 1.

The program itself has been made as simple as possible, so that attention may be concentrated on the monitoring routines; because of this, the program is somewhat artificial, but it serves to demonstrate a number of useful points of technique.

The amendments to the file are assumed to be restricted to direct substitutions; the first half-word of each variable-length record is used as a key, and as the file is copied to tape 3 the successive amendment records are used to replace those file records having corresponding keys.

A trap is set for the end of the input, the trap routine being used to set a zero key so that the remainder of the file can be copied merely by returning to the program. The end of the file is detected as a dummy record having a zero key, and control is then transferred to the titling sequence to conclude the program.

No trap is set for end-of-tape, and should this occur the program will be terminated by the monitor.

Restart points are established immediately following each successful substitution of an amendment record, although in practice they would not be spaced at such frequent intervals. Just prior to each restart point, the positions reached on the tapes are stored in one of two private dumps, used alternately; to ensure that recovery will always be made from the correct dump, no matter when a restart occurs, the choice between two alternative restart addresses is preselected and actually made only when the new restart point is established.

In case the program cannot be brought to a successful conclusion, a private monitor routine is provided to print out the contents of both the private dumps; the programmer should then be able to deduce the position of both tapes at the last established restart point, from reference also to the standard diagnostic print-out which will precede the private monitor output on channel 0.

UPDATING ROUTINE

| | | | | | |
|--------------|------|-----|----|--------|-------------------------------|
| 9) | 1050 | 0 | 0 | 1 | Select input 1 |
| | 1001 | 2 | 0 | 1 | Search for section 1, tape 2 |
| | 1001 | 3 | 0 | 1 | Search for section 1, tape 3 |
| | 1112 | 0 | 0 | 0.2A7 | Set private monitor |
| | 1132 | 0 | 0 | A4 | Set trapping vector |
| | 121 | 4 | 0 | A1 | Set jump in B4 |
| | 121 | 6 | 0 | 0 | Select first private dump |
| | 121 | 15 | 0 | 2 | Preselect first restart point |
| 13) | 1030 | 2 | 0 | 20:0.1 | Start reading tape 2 |
| | 1032 | 3 | 0 | 30:0.1 | Start writing tape 3 |
| 3) | 1024 | 2 | 6 | A12 | Where am I on tape 2? |
| | 1024 | 3 | 6 | 2A12 | Where am I on tape 3? |
| | 1113 | 0 | 15 | A10 | Establish new restart point |
| | 126 | 6 | 0 | 1 | Alternate private dumps |
| | 1057 | 11 | 0 | 100 | Read next input record |
| | 101 | 10 | 0 | 100 | Copy key to B10 |
| 2) | 1033 | 2 | 0 | 0 | Select tape 2 |
| | 121 | 12 | 0 | 25 | Define maximum record length |
| | 1040 | 12 | 0 | 200 | Read next file record |
| | 150 | 10 | 0 | 200 | Set B-test |
| | 1033 | 3 | 0 | 0 | Select tape 3 |
| | 224 | 127 | 4 | 0 | Jump if keys correspond |
| | 1040 | 12 | 0 | 200 | Write file record to tape 3 |
| | 121 | 127 | 0 | A2 | Return for next file record |
| 1) | 124 | 11 | 0 | 0.7 | Round up |
| | 127 | 11 | 0 | -1.0 | Clear octal fraction |
| | 164 | 11 | 12 | 0.7 | Copy marker |
| | 126 | 15 | 0 | 2 | Preselect next restart point |
| | 1040 | 11 | 0 | 100 | Write input record to tape 3 |
| | 121 | 127 | 0 | A3 | Return for next input record |
| TRAP ROUTINE | | | | | |
| 5) | 121 | 10 | 0 | 0 | Place zero key in B10 |
| | 121 | 4 | 0 | A15 | Change jump in B4 |
| | 121 | 127 | 0 | A2 | Return for rest of file |

RESTART SEQUENCE

| | | | | | |
|-----|------------|------------|----------|------------|---------------------------|
| 10) | 121 | 6 | 0 | 0 | Select private dump |
| | <u>121</u> | <u>127</u> | <u>0</u> | <u>A11</u> | Jump |
| | 121 | 6 | 0 | 1 | Select private dump |
| 11) | → 1044 | 2 | 6 | A12 | Search for word on tape 2 |
| | 1044 | 3 | 6 | 2A12 | Search for word on tape 3 |
| | <u>121</u> | <u>127</u> | <u>0</u> | <u>A13</u> | Re-enter program |

PRIVATE MONITOR

| | | | | | |
|-----|-------------|----------|----------|----------|------------------------|
| 7) | 1101 | 90 | 0 | A4/L1 | Start new line |
| | 121 | 5 | 0 | A12-A16 | Character count |
| | 1067 | 5 | 0 | A16 | Output text |
| | 121 | 16 | 0 | -3.4 | Set counter |
| | 121 | 88 | 0 | 6:0.4 | Select style of output |
| 19) | → 101 | 81 | 16 | 3.4A12 | Copy next integer |
| | 1101 | 90 | 0 | A2/L1 | Output integer |
| | → 200 | 127 | 16 | A19 | Test count and return |
| | <u>1117</u> | <u>0</u> | <u>0</u> | <u>0</u> | End program |

TITLING SEQUENCE

| | | | | | |
|-----|-------------|----------|----------|----------|------------------------|
| 15) | 1121 | 8 | 0 | 0 | Form date in B8 |
| | 1015 | 2 | 0 | 300 | Store title |
| | 113 | 8 | 0 | 300 | Change date in title |
| | 1014 | 3 | 0 | 300 | Assign title to tape 3 |
| | 1017 | 2 | 0 | 0 | Free tape 2 |
| | <u>1117</u> | <u>0</u> | <u>0</u> | <u>0</u> | End program |

| | | | | | |
|-----|-----------------|--|--|--|------------------|
| 16) | C | | | | Store text |
| | PRIVATE MONITOR | | | | |
| | SK21 | | | | Carriage control |
| | A 12 = * | | | | Private dump |
| | A4 = * + 4 | | | | |
| | * = 9A4 | | | | |
| | H A5 3.4 | | | | Trap information |
| | * = * + 3 | | | | |
| | + 0 | | | | |
| | E A9 | | | | Enter program |

11.4.7 Monitor Extracodes. Extracodes 1112, 1114 and 1132 have already been introduced; the complete list of instructions for monitoring and trapping will now be given.

- 1122 **ba' = local instruction count.**
Set ba to the number of units of 2048 instructions still to be obeyed before the local instruction counter runs out.
- 1123 **Reset local instructions counter to $n \times 2048$.**
- 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. Used to enable library routines etc. to set up a private trapping-vector.
- 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. The 1134 instruction can be used to enter the standard traps, numbered 0 to 13, or to enter fault routines listed in the trapping vector as numbers 14 onwards. It may also be used as a means of entry to subroutines which are not necessarily fault routines.
- 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.
- 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. |

- 1113 **Set restart.**
Preserve the Supervisor working registers associated with this program. In the event of a future restart, recover these working registers and re-enter this program with b127 = n. If n is negative, cancel the previous restart point without supplying a new one.
- 1114 **Exit from trap.**
This extracode must be used after a computer failure or a magnetic-tape failure has been trapped, and it may also be used to exit from any other trap routine; it may only be used when in trap, and the program will be monitored if it attempts to use this extracode without having set up a trapping vector. The action taken by 1114 is determined by the sign of the singly-modified address S and the parity of its octal digit, as follows:

| S | Action |
|-----------|---|
| -ve | Print "Monitor Entered" and enter Monitor. |
| 0 | Re-enter program. |
| +ve, odd | Resume program at S. |
| +ve, even | Recover extracode working-registers, set b127 to S, and resume current extracode. |

Note that the use of re-entry (S = 0) is not permitted after an 'End of Magnetic Tape' fault.

- 1115 **If monitored, dump on to tape Ba from section n, instead of onto system tape.**
- 1116 **Do not dump if monitored.**

✓ 1117 End program.

This prints out the number of instructions obeyed and the accumulated time of use of magnetic tapes, and instructs the operator to disengage and rewind all tapes used. At the end of each output stream, the amount of output on it is recorded. The program is then cleared from the store.

Appendix A

References

- AUP4 Mercury Orion Atlas Autocode.
- CS 271 Operating Instructions for the Teletype Punch.
- CS 272B Features of the Ferranti Atlas Computer.
- CS 294 Punched Tape Codes.
- CS 298 Atlas Magnetic Tape - Provisional Instruction Code MK4.
- CS 308A Punched Tape Codes (5- and 7-track).
- CS 309A Extracode Functions.
- CS 318 A Fortran Compiler for Atlas.
- CS 339 Operating Instructions for the Model 'B' Flexowriter.
- CS 345 List of Atlas Basic Instructions.
- CS 349 Atlas Programming Exercises.
- CS 350 Extended Mercury Autocode for Orion and Atlas.
- CS 360 Telex Data Links.
- CS 361 Keyboard used with Ferranti Computers.
- CS 362 Operator's Conventions 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 369 Tape Handling Accessories.
- R 40 An Assembly programme for a Phrase-Structure Language.
- R 55 The Manchester University Atlas Operating System.
- R 58 The Atlas Supervisor.
- R 67 Atlas One-Level Storage System.
- R 68 The Atlas Scheduler.

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 79-bit mantissa ax and 8-bit exponent ay
- al the 48-bit floating-point number formed from l , ls and ay
- am the 48-bit floating-point number formed from m and ay
- aq the single-length number which is obtained by standardising, rounding and truncating the contents of the accumulator
- L the less-significant half of Ax , of 39 bits with no sign
- Ls the sign bit associated with L
- M the more-significant half of Ax , of 40 bits, the most-significant bit being the sign digit
- AO accumulator overflow
- DO division overflow
- 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)
- H a half-word register of 24 bits
- 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
- V α 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.

- A α 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
- D α 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
- G α 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.
- J σ σ is octally justified to the left, i.e. the most-significant digit goes into bits 0-2, the next into 3-5 etc.
- K σ 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.
- K σ .k As K σ , with k going into bits 21-23
- L α .k Library routine number α , copy k. (.k is omitted if only one copy of the routine is wanted)
- M Alternative to &
- N A separator which non-equivalences the element before it with the element after it in an expression
- P α Preset parameter α ($0 \leq \alpha \leq 99$)
- Q A separator in an expression which divides the element before it by the element after it, placing the result in digits 0-20
- R α 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
- T The title directive. The line following T is copied to the program output
- T α or }
T α - β } The title is copied to output channel α or channels α to β inclusive
- U α An operator causing the previous element to be logically shifted up α places

Up α or } Unset preset parameter α , or α to β inclusive
UP α - β }

- 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
- Y α α , modulo 2^{24} , 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 $N (\alpha : \beta) : \gamma$ 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 $A \alpha ? = \text{expression}$, $A \alpha$ is only set to the given expression if no other definite setting of A occurs before the program is entered. Similarly for $G \alpha ? = \text{expression}$
 - (b) in the context $P \alpha ? = \text{expression}$, $P \alpha$ is set equal to the given expression unless $P \alpha$ is already set, in which case the directive is ignored

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 | Rank Xeronic Printers <i>Spare</i> | J60040200 |
| 2 | TR7 Paper-Tape Readers | J60040400 |
| 3 | Graphical Outputs | J60040600 |
| 4 | I.C.T. Hammer Printers <i>Amplex Line Printers</i> | 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 | J60042200 |

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 Internal Character Code

Inner set

| Character | Internal code (octal) | Flexowriter code (binary bits & case) | Five-hole code (binary bits & shift) | IBM Fortran card code (holes punched) | Anelex Line-Printer (availability) |
|--------------------|-----------------------|---------------------------------------|--------------------------------------|---------------------------------------|------------------------------------|
| (Unassigned) | 00 | ** 0000.001 | .. | .. | .. |
| Space | 01 | ** 0010.000 | FS 011.10 | 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 | ** 110.11 | .. | .. |
| Shift to UC/FS | 07 | ** 0000.111 | ** 000.00 | .. | .. |
| (Open brackets | 10 | LC 0111.000 | FS 001.01 | 0, 8, 4 | Yes |
|) Close brackets | 11 | LC 0101.001 | FS 001.10 | +, 8, 4 | Yes |
| , Comma | 12 | LC 0101.111 | FS 011.11 | 0, 8, 3 | Yes |
| π (£) Pi | 13 | LC 0111.011 | LS 111.10 | -, 8, 3 | Yes |
| ? Query | 14 | LC 0101.100 | LS 111.01 | -, 8, 5 | Yes |
| & Ampersand | 15 | LC 0111.101 | .. | .. | Yes |
| * Asterisk | 16 | LC 0111.110 | FS 000.11 | -, 8, 4 | Yes |
| / Oblique | 17 | UC 0011.111 | FS 101.11 | 0, 1 | Yes |
| 0 Zero | 20 | UC 0100.000 | FS 100.00 | 0 | Yes |
| 1 | 21 | UC 0110.001 | FS 000.01 | 1 | Yes |
| 2 | 22 | UC 0110.010 | FS 000.10 | 2 | Yes |
| 3 | 23 | UC 0100.011 | FS 100.11 | 3 | Yes |
| 4 | 24 | UC 0110.100 | FS 001.00 | 4 | Yes |
| 5 | 25 | UC 0100.101 | FS 101.01 | 5 | Yes |
| 6 | 26 | UC 0100.110 | FS 101.10 | 6 | Yes |
| 7 | 27 | UC 0110.111 | FS 001.11 | 7 | Yes |
| 8 | 30 | UC 0111.000 | FS 010.00 | 8 | Yes |
| 9 | 31 | UC 0101.001 | FS 110.01 | 9 | Yes |
| < Less than | 32 | LC 0100.011 | .. | 0, 8, 5 | Yes |
| > Greater than | 33 | LC 0110.100 | FS 100.01 | +, 8, 5 | Yes |
| = Equals | 34 | LC 0100.101 | FS 010.10 | 8, 3 | Yes |
| + Plus | 35 | UC 0111.101 | FS 110.10 | + | Yes |
| - Minus | 36 | UC 0111.110 | FS 010.11 | - | Yes |
| . Point | 37 | UC 0101.111 | ** 111.00 | +, 8, 3 | Yes |

✓
Inner set (continued)

| Character | Internal code (octal) | Flexowriter code (binary bits & case) | Five-hole code (binary bits & shift) | IBM Fortran card code (holes punched) | Anelox Line-Printer (availability) |
|------------------|-----------------------|---------------------------------------|--------------------------------------|---------------------------------------|------------------------------------|
| ' (n) Apostrophe | 40 | LC 0100.000 | FS 111.01 | 8,4 | Yes |
| A | 41 | UC 1010.001 | LS 000.01 | +,1 | Yes |
| B | 42 | UC 1010.010 | LS 000.10 | +,2 | Yes |
| C | 43 | UC 1000.011 | LS 000.11 | +,3 | Yes |
| D | 44 | UC 1010.100 | LS 001.00 | +,4 | Yes |
| E | 45 | UC 1000.101 | LS 001.01 | +,5 | Yes |
| F | 46 | UC 1000.110 | LS 001.10 | +,6 | Yes |
| G | 47 | UC 1010.111 | LS 001.11 | +,7 | Yes |
| H | 50 | UC 1011.000 | LS 010.00 | +,8 | Yes |
| I | 51 | UC 1001.001 | LS 010.01 | +,9 | Yes |
| J | 52 | UC 1001.010 | LS 010.10 | -,1 | Yes |
| K | 53 | UC 1011.011 | LS 011.11 | -,2 | Yes |
| L | 54 | UC 1001.100 | LS 011.00 | -,3 | Yes |
| M | 55 | UC 1011.101 | LS 011.01 | -,4 | Yes |
| N | 56 | UC 1011.110 | LS 011.10 | -,5 | Yes |
| O | 57 | UC 1001.111 | LS 011.11 | -,6 | Yes |
| P | 60 | UC 1110.000 | LS 100.00 | -,7 | Yes |
| Q | 61 | UC 1100.001 | LS 100.01 | -,8 | Yes |
| R | 62 | UC 1100.010 | LS 100.10 | -,9 | Yes |
| S | 63 | UC 1110.011 | LS 100.11 | 0,2 | Yes |
| T | 64 | UC 1100.100 | LS 101.00 | 0,3 | Yes |
| U | 65 | UC 1110.101 | LS 101.01 | 0,4 | Yes |
| V | 66 | UC 1110.110 | LS 101.10 | 0,5 | Yes |
| W | 67 | UC 1100.111 | LS 101.11 | 0,6 | Yes |
| X | 70 | UC 1101.000 | LS 110.00 | 0,7 | Yes |
| Y | 71 | UC 1111.001 | LS 110.01 | 0,8 | Yes |
| Z | 72 | UC 1111.010 | LS 110.10 | 0,9 | Yes |
| (Unassigned) | 73 | UC 1101.011 | .. | .. | Yes (10) |
| (Unassigned) | 74 | UC 1111.100 | .. | .. | Yes (11) |
| (Unassigned) | 75 | UC 1101.101 | .. | .. | Yes (8) |
| (Unassigned) | 76 | UC 1101.110 | .. | .. | .. |
| Fault | 77 | .. | .. | .. | .. |

Outer Set

| Character | ✓ | Internal code (octal) | Flexowriter code (binary bits & case) | Five-hole code (binary bits & shift) | Anelex Line-Printer (availability) |
|--------------------------------|---|-----------------------|---------------------------------------|--------------------------------------|------------------------------------|
| (Unassigned) | | 00 | ** 0000.001 | .. | .. |
| Space | | 01 | ** 0010.000 | FS 011.10 | .. |
| (Spare) | | 02 | .. | .. | .. |
| (Spare) £ | | 03 | .. | .. | Yes |
| Shift to outer set | | 04 | .. | .. | .. |
| Shift to inner set | | 05 | .. | .. | .. |
| Shift to LC/LS | | 06 | ** 0010.110 | ** 110.11 | .. |
| Shift to UC/FS | | 07 | ** 0000.111 | ** 000.00 | .. |
| (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 | .. | Yes |
| φ (x) Phi | | 20 | .. | FS 110.00 | .. |
| [Open square brackets | | 21 | LC 0110.001 | .. | Yes |
|] Close square brackets | | 22 | LC 0110.010 | .. | Yes |
| → Arrow | | 23 | .. | FS 101.00 | .. |
| ≥ Greater or equal | | 24 | .. | FS 100.10 | .. |
| ≠ Not equal | | 25 | .. | FS 010.01 | .. |
| _ Underline | | 26 | LC 0100.110 | .. | Yes |
| Vertical bar | | 27 | LC 0110.111 | .. | Yes |
| ² (%) Superscript 2 | | 30 | LC 0101.010 | .. | Yes |
| ≈ (v) Curly equal | | 31 | .. | FS 011.00 | .. |
| α (IO) Alpha | | 32 | UC 0101.010 | .. | Yes |
| β (II) Beta | | 33 | UC 0111.011 | .. | Yes |
| ½ Half | | 34 | UC 0101.100 | .. | Yes |
| (Spare) 10 | | 35 | .. | .. | Yes |
| (Spare) 11 | | 36 | .. | .. | Yes |
| (Unassigned) | | 37 | UC 1000.000 | .. | .. |