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.

	EXPONENT		MANTISSA
0	1 7	8	9 47

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

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

$$x.8^y$$

The exponent is an integer and lies in the range

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

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

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

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

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

Exponent	Mantissa
0 0000010	0.001000000............. 000

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

3.2 The Accumulator

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

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

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

Am consists of A_y and M

Al consists of A_y and L with Ls

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

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

Diagram of the whole accumulator

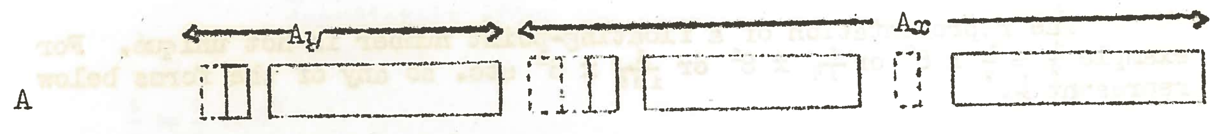
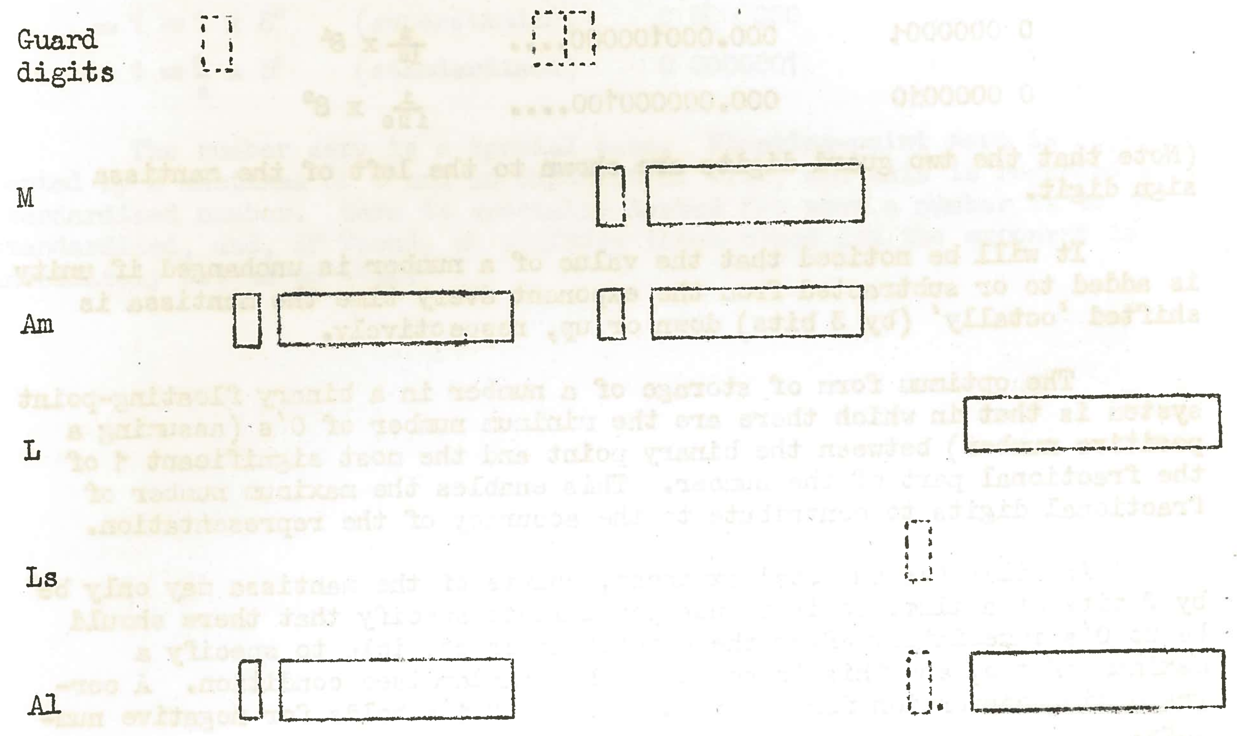


Diagram showing component parts of the accumulator



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

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

3.3 Standardised numbers

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

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

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

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

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

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

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

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

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

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

Example: the addition of $\frac{5}{8} + \frac{3}{8}$ with a standardising instruction gives the correct result of $+1$ standardised; though immediately after the addition the number is 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 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 halfway case of L being a binary one followed by a string of zeros.

3.6 Floating-point operations

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

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

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 $sy < ay$, then s would be shifted down into L. overwriting the original contents of L.

3.7 Standardising and rounding accumulator instructions

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

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

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

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

```
324  0  0  3   put d into Am
362  0  0  3   form d²
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 d²
356  0  0  0   store answer in word 0
```

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

3.8 The timing of instructions

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

3.9 Some fixed-point Instructions

Fixed-point working has been introduced in section 3.4

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

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

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

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

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

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

Chapter 4

THE B-REGISTERS

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

4.1 General Purpose B-registers

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

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

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

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

4.2 Arithmetic Operations

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

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

<u>Example</u>: 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 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

<u>Examples</u>:

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	-n + 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 ≠ gives a 1 in the positions in which the corresponding digits of the two numbers differ, and 0's elsewhere.

The result of non-equivalencing

```
     00010110
with 01110100
is   01100010
```

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

The result of ORing

```
     00010110
with 01110100
is   01110110
```

Function	Description	Notation
107	Collate the digits of Ba with the digits of s placing the result in Ba	$\overline{ba'} = ba \,\&\, s$
106	Non-equivalence ba with s, placing the result in Ba	$\overline{ba'} = ba \neq s$
147	OR ba with s, placing the result in Ba	$\overline{ba'} = ba \vee s$
117	As 107, but placing the result in S	$\overline{s'} = s \,\&\, ba$
116	As 106, but placing the result in S	$\overline{s'} = s \neq ba$
127	Collate ba with n, placing the result in Ba	$\overline{ba'} = ba \,\&\, n$

Function	Description	Notation
126	Non-equivalence ba with n, placing the result in Ba	$\overline{ba'} = ba \neq n$
167	OR ba with n, placing the result in Ba	$\overline{ba'} = ba \vee n$

Examples:

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

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

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

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

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

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

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

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

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

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

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

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

```
126     62     0     0'
```

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

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

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

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

```
164     2     1     J0077
```

4.4 Test Instructions

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

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

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

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

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

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

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

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

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

```
Register                Contents

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

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

4.5 Special Purpose B-registers B120-B127

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

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

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

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

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

Example:

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

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

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

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

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

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

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

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

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

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

A few examples will make this clearer.

Example 1

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

Example 2

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

This could be done by the sequence of instructions

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

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

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

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

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

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

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

(a) Bm specified as B122

```
121    1    122    0     always puts zero in B1
```

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

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

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

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

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

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

4.6 Modification/Counting Instructions

The technique of modification has already been introduced.

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

```
324     0     3     100
```

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

Suppose we have 20 unstandardised floating-point numbers stored in locations 100-119, and it is required to standardise these numbers and re-store 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 ≥ 0
```

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

The most important of these are:-

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

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

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

```
203    127    3    11    If b3 ≠ 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 $\neq 0$, and the other to show whether it is ≥ 0 or < 0.

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

Function	Description	Notation
152	Set the B-test register by writing to it the contents of Ba minus the contents of S. ba and s are unchanged.	$bt' = ba - s$
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 $ba' = n$

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

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

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

Example:

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

```
0   121   1   0   -3      set required number in B1
1   121   2   0   100     first address in B2
2   152   1   2   0       bt' = ba - s
3   224   127 0   7       jump if bt = 0, i.e. s = -3
4   170   2   0   199.4   bt' = 199.4 - ba
5   220   127 2   2       if bt ≠ 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
```