4.8 The Shifting Instructions

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

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

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

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

4.9 The Odd/Even Test Instructions

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

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

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

Example:

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

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

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

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

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

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

4.11 The B-carry digit

When any one of the four addition codes

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

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

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

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

Example:

124	2	1	0
1223	3	3	Y1

Similarly each of the ten instructions

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

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

Example:

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

122	2	1	0
1223	3	3	-Y1

Chapter 5

ROUTINES AND DIRECTIVES

5.1 Routines, Subroutines and Symbolic Addresses

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

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

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

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

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

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

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

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

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

(b) Which library routines are required.

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

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

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

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

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

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

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

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

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

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

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

5.2 Routine Parameters

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

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

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

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

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

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

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

```
121    127    0    A1    causes a jump to the location whose
                          address is defined by A1.
```

Example:

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

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

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

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

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

Examples:

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

An item may be labelled more than once. Thus

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

5.3 Preset Parameters

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

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

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

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

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

5.4 Global Parameters

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

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

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

5.5 Optional Parameter Setting

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

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

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

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

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

The library routine L100 contains the directive

A24? = 160

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

5.6 Expressions

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

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

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

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

Alternative forms are:-

a:b.c

a:b

a:

b.c

b

Examples:

1:35.6 is, in octal, 00010436

2: is 00020000

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

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

Y19 is 00000023

as opposed to 19 which is 00000230.

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

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

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

Examples:

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

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

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

The operators are as follows:-

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

```
121   121   0   100D1
```

is an alternative to writing

```
121   121   0   50
```

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

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

```
121   124   0   13U12
```

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

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

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

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

```
121   2   0   J1'
```

sets b2 = J67777777

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

5.7 Separators

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

(i) Elements may be added or subtracted.

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

Examples:

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

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

Example:

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

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

Examples:

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

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

Examples:

2.4 X 2.4 = 6.2	Exact
2.1 X 1.2 = 2.6	Result rounds away from zero
J001 X J001 = 0	$2^{30} = 0$, modulo 2^{20}
Y1 X Y1 = 0.1	Result rounds away from zero

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

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

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

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

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

5.8 The Special Parameter *

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

For example

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

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

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

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

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

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

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

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

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

121 127 0 4*

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

5.9 The Ba and Bm Parts of an Instruction

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

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

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

Example:

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

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

5.10 Half-Words, Six-Bit Words and Characters

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

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

Examples:

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

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

Examples:

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

Other directives may occur mixed with half-words.

Example:

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

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

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

Example:

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

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

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

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

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

000000 000000 000000 001000	000000 000000 000001 111000
H1	H15

000000 000100 000010 000000	000011	001111	xxxxxx	
H2064	S3	S15	SA1	

000000 000000 001100 000000	000000 000000 001100 001000
H96	H97

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

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

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

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

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

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

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

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

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

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

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

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

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

For example

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

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

5.11 Floating-Point Numbers

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

The following notation is used:-

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

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

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

(i) a

Examples:

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

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

Examples:

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

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

Examples:

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

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

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

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

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

e.g. +K363.174

-K.265

-K777777

+K0.4

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

e.g. K14 -K276

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

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

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

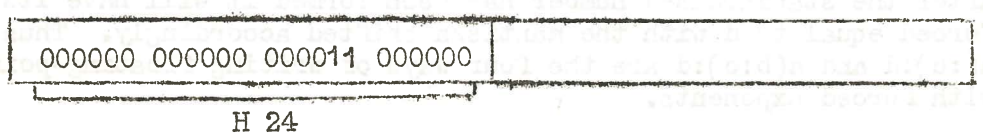
Any floating-point number can be labelled, and more than one may appear on a line. Floating-point numbers can also be mixed with half-words and six-bit words on a line, provided that the first of a group on a line is preceded by the directive F.

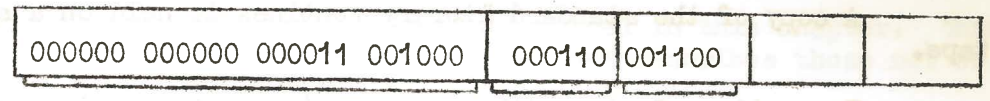
This directive, which does not need a terminator, introduces floating-point numbers. It has also the effect of increasing the value of *, if necessary, to the address of a full-word, and it can be used, for example, immediately before H to ensure that the next half-word is stored as the more-significant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

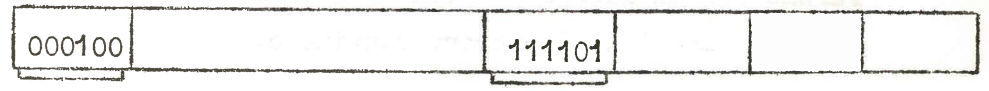
Example:

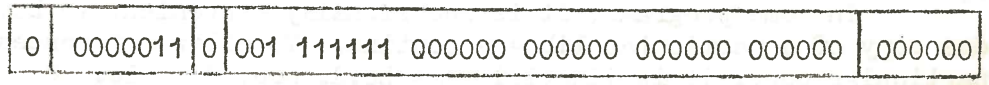
H24 F H25 S6 12 F S4 H S61 F+127

would appear in four consecutive words of the store as









F+127

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

5.12 Library Routines

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

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

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

Aa/Lc label a of library routine c.

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

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

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

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

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

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

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

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

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

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

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

5.13 Directives

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

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

Parameter = Expression

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

Parameter ? = Expression.

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

'Ignore' directives

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

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

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

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

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

Examples:

```
[a > b]
[a >[b - c]>d]
< 1 + a [1 + b [1 + c [.... >
```

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

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

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

C and CT directives

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

E directives

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

(a) It terminates the current routine.
(b) All parameters and expressions which have been used are evaluated and inserted into the program.
(c) Library routines are found and inserted at the required places or at the end of the program, depending on whether they were called for by L directives or simply referred to.

Library routines that have been referred to in the program (but not explicitly called by L-directives) will be inserted in store locations immediately after the last item before the E directive. Note that if * directives have been used this is not necessarily the highest address used by the program, and care is required to ensure that library routines do not overwrite any part of the program. The library routines may be stored in, for example, locations A3 + 9 onwards by preceding the E directive by * = A3 + 9.

(d) Fault indications are printed out for parameters which are used but not set, and for any other faults encountered. If there are any faults, the program is normally suspended and not entered. (see section 12.6 for exceptions to this.)
(e) The compiler, which has occupied store locations above $\frac{3}{4} \times 2^{20}$ (that is, J3) is deleted from the store so that storage location numbers up to $\frac{7}{8} \times 2^{20}$ (that is, J34) may now be used. (Note: the Supervisor uses store locations from J34 upwards).

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

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

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

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

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

F directive

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

H directive

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

L. Library directives

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

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

R. Routine directives

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

S directive

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

T. Title directive

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

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