7.1.1 Character Set

Basic Input Language is in characters. The characters used are

0 1 2 3 4 5 6 7 8 9

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

= . ( ) * + - , /

VB (visible space)

Dummy (always ignored)

EL (end of line)

Several VS characters are equivalent to one VS

ER character is everywhere ignored.

| (vertical bar) means "ignore rest of this line".

When a sequence of Basic Input Language is on magnetic tape or drum, the characters are represented directly (internal representation); on paper tape or cards, the characters are represented by the print-out of the tape or cards.

7.1.1.1 Magnetic Tape and Drum

For Basic Input Language programs on magnetic tape or drum, the code is the standard internal code (see 5.6.1).

Basic Input Routine reads the characters from each word, starting at the m.s. end.

Magnetic tape blocks must be less than 130 words long, with the length of each block as a binary integer in the l.s. 15 bits of the first word of the block (Word 0). The Basic Language characters begin in Word 1 unless the magnetic tape is a composite document (see 6.1).

7.1.1.2 7-track paper tape

For Basic Input Language programs on 7-track, the 7-track code is used (see 4.3..3).

Lower case and upper case letters are regarded as equivalent. BS is allowed. Crossed parentheses mean "ignore this line" and all printing characters, except cross parentheses, to the right of vertical bar are ignored. Unassigned characters are ignored.

7.1.1.3 5-track paper tape

For Basic Input Language programs on 5-track code is used (see 4.3.3). See 7.1.5.3 for 5-track conventions.

7.1.1.4 Cards

For Basic Input Language programs on cards, the standard card code (see 5.6.2) is used.

7.1.1.5 End of line

On paper tape, EL is considered to appear immediately after the last printing symbol on the line

On cards the convention is that one card represents one line.

7.1.2 Uses of Basic Input Routine

7.1.2.1 The purpose of this routine is to

(i) Assemble and store a program on the drum ready to run, with appropriate parameters, if necessary.

and (ii) Set up initial conditions for the program particularly to reserve core and drum store and peripheral devices and to ensure that the right documents are loaded.

The Basic Input Language to be read may be on 7-track , 5-track paper tape, cards, magnetic tape and the drum.

7.1.2.2 Identifiers

Two types are provided known as the L's and the V's. An identifier may be referred to in an "address—field"; when this field "comes to be stored" the current value of the identifier is used. An identifier is set either "by label" or "by equation". (See 7.1.4.4)

7.1.2.2.1 The L's

e.g. L65 or L1.1

An L is set to have a value; this is a 24-bit signed quantity.

Basic Input Routine makes a list of the L's and their values. It stores them in blocks of 64 and an L may be written as Lb.p where b is the block and p the position number. e.g. L65 and L1.1 refer to the same L.

If an L is referred to, then Basic Input Routine appropriates 64 words of working space for the L's within that block. There is a limit of 512 blocks.

An L may be in 4 states

(i) Never heard of
(ii) Referred to but not set
(iii) Partially set
(iv) Fully set (known)

References are forward references unless the L is in state (iv)

Unless an L is in state (i) or (ii), a setting of the L is an error. There are facilities for freeing an L and for nesting an L.

7.1.2.2.2 The V s

e.g. V5

The value of a V is the 48-bit content of the corresponding register and hence the value of a V is always known. A V may be re-set.

The V's are special purpose quantities used by Basic Input Routine.
V1 (stored in A1). Its value is the drum transfer address.
V2 (stored in A2). Its value is the core transfer address.

When Basic Input Routine reads a "storable" word to be stored in one word then V1 and V2 are stepped on by 1 (see 7.1.4.1)

V0 has the value 0, but when set it causes V1 and V2 to be moved in step. For example, if the old value of V1 is 2 and V2 is A64, then

V0=A74 (i.e. X)

gives new value of V1=12 (i.e. X - V2 i.e. A74-A64+2)
and V2=A74 (i.e. X)

V3 to V12 (stored in A3 to A12). These V's are not disturbed by Basic Input Routine in its purpose of storing program on the drum, unless it reads a setting of them, in which case the corresponding accumulator will contain the required value.

V13 is used for the checksum.

V62 gives the current input channel; if D0 is 1 , then the input channel is the drum.

V64 is the type of print barrel (see section 14 and 7.1.4.2.1 2).

V65 is the datum point.

V66 contains 2 constants for use by Autocode (see 7.1.5.1)

If a V is referred to in an “address-field”, then the current value is used.

7.1.2.3 Example settings (see 7.1.4.4)

L258) V5) core label settings; each identifier is given the value of the current core transfer address.

L51.3))V4)) drum label settings; each identifier is given the value of the current drum transfer address.

```
L1 = 63
L713 = A52+L3
V2 = V2+16
V11 = -4002
```

L3.2 = V5 the l.s 24 bits of the value of the V are the new value of the L.

V3=L6 the new value of the V is numerically equal to the value of the L.

7.1.2.4 A nest and equivalent facility is provided to be used in conjunction with calls. This allows the current status of all L’s (and some V’s) to be remembered and the identifiers freed for use by, for example, a called document and then later restored (nest facility) and also for specified identifiers of the new set to become identified (equivalenced) during nesting with specified identifiers of the nested set.

7.1.3 Language

7.1.3.1 Format Descriptions

A class is denoted by listing the members, separated by commas, as strings of class-expressions.

A class-expression is made up of class names and characters using the special symbols

( ) [ ] < > , * ? = %

7.1.3.2 A class name is any set of characters except * ? [ ] enclosed in square brackets. Parentheses are used for grouping and separation where required. The equal sign denotes a definition of a class name. Null is the empty class. % is used to terminate a definition.

The characters * and ? have special meanings defined as follows:-

(a) if m is any class - expression
m? = m, null % (i.e. one or nothing)
m* = m, (m*), m % (several or at least one)
m*?= (m*)? % (several or one or nothing)

(b) if [m] is any class name
[m?] = [m]? %
[m*] = [m]* %
[m*?]= [m]*? %

Associated with each * there may be an expression like 3, <6, or = 4 which indicates the range of the number of repetitions allowed.

7.1.3.3 Preliminary Definitions

Some class names, e.g. [left parenthesis], [asterisk], [directive], [sequence] are taken as understood without definition

[l] = A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z %

[d] = 0,1,2,3,4,5,6,7,8,9 %

[±] = +, - %

[n] = [d*] %

[peripheral name] = [asterisk][l][l][n] %

[style name] = [asterisk][l][l][l] %

A peripheral name is stored in a 15 bit field. Each letter is converted into a 5-bit quantity by removing the m.s. bit of the 6-bit character. The 2 truncated letters are packed into the m.s. 10 bits of the field. The number [n] is added into the field. E.g. *SR1 is stored as 20033

A style name, e.g. *SIG, is stored in a 15 bit field; the letters are truncated as described above and then stored.

7.1.3.4 Elements

(i) [element]=((V,L)*?)[n](.[n]*?), [m]%

where [m]=[peripheral name], (A?)[n], [style name]%

Examples are V4 and L20 and 2 and L1.36 and 100.3.. and *MT1 and A7.

The appearance of . (point) in an element causes the numerical part of the element already read to be shifted up 6 bits, e.g. L1.0 is an alternative way of writing L64.

V or L appearing in an element causes the value (see 7.1.2.2) of the corresponding identifier to be looked up when a terminator is found; thus forward references in an element are not allowed.

A string of V's and L's may appear, e.g. LV7 giving effectively indirect addressing of identifiers.

(ii) [gen element] = (L?)[element]%

If the first character is an L then forward references are allowed.

7.1.3.5 Quantities and Fields

(i) [quantity]=[element]([comma]([±?][element]?),[±][element]*?) %

(ii) [field]=[quantity],([quantity]([±],[comma])?)[gen element]([±][gen element]*?)%

Examples are as for elements and V2-L1 and L1.2+60 and 5,,,,*CR1 and 10,10,10+16,1+32,2,10,1+32,12

+ (plus) and - (minus) have arithmetical meaning, e.g. 10+16 is equivalent to writing 26.

The appearance of comma in a quantity causes the elements already read to be shifted up 6 bits. Thus the storable line (see 7.1.4.1)

+21,6,L7,12,,,,

will cause the 4 elements to be packed into the top half of the word.

No forward reference may appear before a comma in a field.

7.1.3.6 Components and Documents

(i) [component]=((VS?)([l],[d],.)*<9),((VS?)[asterisk]*<3?)%

Examples are ABC.12 and **

The character VS is ignored when reading a component. A component is stored in a word with the characters right-justified.

The component * causes today's date to be stored as that component. This is the date in characters as given by the 150/12 instruction (see 5.3.12), e.g. 3lMAR64.

The component ** causes the time to be stored in character form, e.g. 11.05.59

(ii) [doc name]=[component]/[component](/([component],+)*<7?)%

Example ONE/DOCUMENT/+/1.2.63/VERSION2/ABC.12

The character + (plus) as a 'component' of a document name means that the document is composite (see 6.1). It cannot be the first or second component.

(iii) [doc request name]=[component]/[component](/([component],+,-)*<7?)%

Example MY/DATA/-/XYZ/-

In a request for a document at least the first 2 components must be specified (i.e. - (minus) cannot be the first or second component). - (minus) means "don't care what this component is". If the last specified 'component' is - (minus) then this means "don't care what further components are".

(iv) [source name]=[doc request name],+[quantity],[peripheral name]%

where +[quantity] is taken as a drum address.

7.1.4 Formats

The purpose of Basic Input Routine has already been stated (see 7.1.2). Those lines of a program which cause B.I.R. to store information on the drum for the running of the program are called storable words. Those lines of a program which direct B.I.R. to for example, reserve a reader for the program are called directives.

7.1.4.1 Storable Words

The formats of storable words are defined. When B.I.R. reads such a line, one word of information is stored and the transfer addresses both stepped on by one.

Any of these formats may be preceded by one or more core-labels or drum-labels or both as defined in 7.1.4.4.

There are 4 types of storable words.

7.1.4.1.1 A number

[number]=[±][field]%

Examples are

```
-2561
+1346895
+L157
-L1.6+V3
+10+16,10,10,1+32,2,10,1+32,12
-134567,L3.1-2,,
+33.34.35.36.37.38.39.40
```

The number is stored as a 48-bit signed integer. Overflow may occur if the integer is not in the range $\pm(2^{47}-1)$.

If an L is written, then it is regarded as a 48-bit integer whose value is numerically equal to the value of the L.

7.1.4.1.2 Fraction

[fraction]=[±](0*?).[n]F%

```
e.g.     +0.333F
         -.4123F
```

The [n] must be in the range $0 \leq n \leq 10^{14}-1$

The fraction -1.0 or integer -2^{47} can be stored by using the instruction format. 00 0 0 0

7.1.4.1.3 Instruction

[instruction]=[fun][adr*<4?]%

where

[fun]=[n](.[n]?)(X?)(Y?)(S?)%

[adr]=VS[left parenthesis?][±?][field][right parenthesis?]

In the function [fun] if a mode (e.g. for a 141 instruction) is present, the Y-address will unless replaced be distributed over X- and Y-address fields.

The X, Y and S denote X- and Y- modification and signal bit respectively.

Addresses are in the order X-address, Y-address and Z-address. If fewer than three addresses are written the remaining address-fields are stored as zero. A left parenthesis denotes replacement; the right parenthesis, if present is always ignored. The Z-address may not be replaced. If the Z-address is not written the instruction is a 2—address unmodified instruction and the TX bit will be set unless Y and not X appeared in the function part.

Examples

```
04XY     (L16)     V3+L1.2-12    A4

140.1    0        *SR1
142S     L3.8+20   24

14       L3        -5
```

7.1.4.1.4 Name format

[name]=NAM[l*?]VS([doc name],[doc request name])%

```
e.g.     NAME  A/NAME/A.B.1/*
         NAME  A/REQ12/-/XYZ/-
         NAME  A/REQ12/-/XYZ/-/
```

The name is stored at the current value of the transfer addresses into 8 words. One component (see 7.1.3.6) is stored in a word. Not all 8 components and corresponding solidi need be specified. For a specified component, the corresponding word contains the characters, if less than 8, right-justified. Unless the last character on the line is - (minus), then for non-specified components the corresponding words will be clear. If the last character is - (minus) then all subsequent words will contain character - (minus) at the l.s. end. Thus in the second example the last 4 words of the 8 will all contain - (minus) and in the last example the last 3 words will be clear.

7.1.4.2 DIRECTIVES

7.1.4.2.1 Normal directive

[normal directive]=NOR[l*?]%

This has no effect except to terminate sequences (see 7.1.4.3)

7.1.4.2.2 Read directive

[read directive]=REA[l*?]VS[source name],[component]%

A line with this directive must be followed by a separator (see 7.1.5.4 and 7.1.5.5)

This directive causes Basic Input Routine to

(i) relinquish the current input peripheral, unless it is the drum or a RESERVE THIS directive has been read; (i.e. if the number in the programmer's peripheral name of the current input peripheral is 20 or above, Basic Input relinquishes it). If the current input peripheral is a tape deck and the document on it is composite, the tape is left in the load position.

and (ii) to continue reading from the specified source, if a [source name] is specified (see 7.1 3.6) or to load and enter the semi-built-in program if a [component] is specified.

(a) If [source name] is [doc request name]

e.g. REA MY/PROGRAM/MK5/-

then Basic Input Routine finds the document and continues by reading it.

The device on which the document is loaded, is reserved for the job (the number in the programmers peripheral name being 20 or above). The document may be one on a composite magnetic tape document (see 6.1) in which case Basic Input will find the document on the composite one and read it.

(b) If [source name] is +[quantity]

e.g. REA +16

then the quantity is the drum address from which Basic Input will start reading.

(c) If [source name] is [peripheral name]

e.g. REA *SR1

then Basic Input implements the reservation request for this peripheral, and then reads the document on it.

It is permitted to write for example

REA *0+V3 (0 is zero)

where 0+V3 is taken as a peripheral name.

(d) If a [component] is specified then it is the name of a semi-built-in program (see 6.3)

e.g. REA PRINT

Basic Input will load and enter Chapter 1 of this semi-built-in program.

7.1.4.2.3 Reserve directive

[reserve directive]=RES[1*?]VS(([peripheral name]VS ([doc name],[doc request name],THIS)),([asterisk]CORE, [asterisk]DRUM)VS[quantity])%

This directive is used for reserving peripherals core and drum store for the program.

(i) Peripherals

The number in the programmer's peripheral name should be below 20. The reservation request is remembered when read and is implemented when ENTER directive (see 7.1.4.2.12) or PROCESS directive (see 7.1.4.2.10) is read. The peripheral is reserved and a specific device allocated. For slow input devices and magnetic tape decks, READ [peripheral name] also causes the reservation request to be implemented. If at the time of implementation no device of the required type is free then the job is halted awaiting space.

(i) (a) Slow Input devices and magnetic tapes

e.g. RES *SR1 MY/DATA/-

The [doc request name] specifies the document to be found and the [peripheral name], the type of device required.

For documents on magnetic tape, Basic Input Routine will find a document on a composite document (see 6.1) - it will read the name block and position the tape ready to read the first block of information of the document.

USE OF THIS

e.g. RES *SR2 THIS

The current input peripheral is reserved for the job, with in general the number in the programmers' peripheral name being 20 or above. This directive asks Basic Input to change its name to that specified (it will be of the same type and the number must be below 20). When subsequently Basic Input finishes reading from this peripheral, the peripheral will still be reserved for the job; when Basic Input reads certain directives (e.g. READ, ENTER, END) it relinquishes the current input peripheral if the number in the programmer peripheral name is 20 or above.

(i) (b) Slow Output Devices

e.g. RES *LP1 MY/RESULTS/*/**

The [doc name] specifies the document name which is to be output on the allocated device.

(ii) Core Store

e.g. RES *CORE 200

The [quantity] specifies that at least this number of words of core are to be reserved. The number of words reserved is of the form 64n-16.

(ii) (a) Decreasing core reservations

In this case Basic Input will remember the request and implement it on reading ENTER (see 7.1.4.2.12) and the datum point remains unchanged.

(ii) (b) Increasing core reservations

In this case, the directive must be followed by a separator (see 7.1.5.4 and 7.1.5.5). This request is implemented on reading the directive and the datum point may be changed and so generally the request should appear before any storable words. Basic makes use of the extra core.

If the amount requested is not available, the job is halted awaiting space; the message being NO CORE.

Note that core may be reserved with JOB directive (see 5.7.2) in which case, RES *CORE directive need not be used. Only one request for core is advisable.

(iii) Drum Store

e.g. RES *DRUM 1000

The quantity specifies a number of words of drum; it is rounded up to the next multiple of 64, unless it is a multiple of 64 already.

The directive is implemented on reading it. When the program is entered, then the amount of drum reserved for the program is the greater of

The amount requested with the last directive read (this is 64 if no RES *DRUM has been read) (see 7.1.4.2.16).

or V1 (rounded to a multiple of 64).

The job may be halted awaiting space, if the drum required is not available, the message being NO DRUM.

7.1.4.2.4 New Directive

[new directive]=NEW(VS[n]?)VS[asterisk]MT[n]VS
(((P?)[n].[n].[n](P?)(L?)VS[doc name]),0)%

E.G. NEW *MT1 P1.8.1965 B/C/*/**

This directive is implemented when ENTER (see 7.1.4.2.12) or PROCESS (see 7.1.4.2.10) is read.

If the peripheral (*MT1 in the example) has not already been reserved, then it is reserved, a scratch tape (see 5.3.33 b (ii)) being allocated. If (VS[n]?) is present then this integer specifies the nominal length in hundreds of feet of the scratch tape required, otherwise any lengthed tape is requested. If the peripheral has already been reserved then no reservation takes place.

The other information given with the directive gives new Block 0 information.

The first (P?) gives the setting of the write permit bit (D0) and the second (P?) gives the date control bit (D24). P present means set the bit to 1; P not present means set the bit to 0.

[n].[n].[n] specifies the Date (in D1 to D20) required; the first [n] specifying the day, the second the month and the third the year. If the third [n] is less than 100 then 1900 is added to it and stored as the year, so that 65 is stored as 1965.

[doc name] specifies the new document name required.

(L?) - this will be used only on installations with high density decks, when a high job needs to write low density tapes. (L present causes D25 of the first word for the 150 instruction to be 1, otherwise 0)

In the case of NEW directive Basic Input Routine writes this Block 0 information onto the tape with a 150/41 instruction.

The directive NEW *MT1 0 (this is zero) gives new Block 0 information such that the tape is considered a scratch tape.

7.1.4.2.5 Compiler or Use Directive

[compiler or use directive]=(USE,COM)[1*?]VS([source name],[component])%

This directive needs a separator (see 7.1.5.4 and 7.1.5.5).

This directive differs from the READ directive (see 7.1.4.2.2) only in that the current input peripheral is not relinquished and that on reading END Basic returns.

If a [component] is specified then Chapter 1 of the named semi-built-in program is loaded and entered.

e.g. USE PUNCH

If [source name] is specified, then Basic Input Routine continues by reading from the specified source. If END directive (see 7.1.4.2.18) is read on the requested document or drum or on the document on the specified peripheral, then Basic Input Routine returns to read from the peripheral on which it read the USE directive. (Note that READ does not cause this to happen).

```
USE      MY/NEXT/TAPE/3/-
USE      +36
```

While reading the requested document etc, Basic stores the name of the peripheral to which to return in A61 (i.e. if V61 is positive, it is the programmers peripheral name; if negative the modifier is the drum address.)

7.1.4.2.6 Report Directive

[report]=REP[l*?]VS[n]VS([peripheral name]VS[doc name]),
([n]VS[doc name])?)

e.g. REP 2 *SP1 MY/MONIT/*/**

While Basic Input Routine is reading the program it may output various reports. In general a monitoring peripheral is necessary for these reports, so on reading this directive Basic Input will reserve (if not already reserved) a slow output peripheral with the specified programmer's peripheral name and on the allocated device will output the specified document name. This output peripheral is set as the monitoring peripheral for the program and the report level (the first [n]) which will be either 0,1 ,2, or 3 will be set.

e.g. REP 0 5 MY/OUTPUT/*

Instead of a programmer's peripheral name an integer (e.g. 5) may be written which means, get the first free slow output device and reserve it with appropriate programmer's peripheral name, the number being that given. (In the example if the first free slow output device were a 5-track punch, then it would be reserved for the program as *FP5 and the specified document name would be output on this device set as the monitoring peripheral). If the job already had a slow output peripheral reserved with the specified number (5 in this case,) then another peripheral would not be reserved but this peripheral would be set as the monitoring peripheral and the document name would be output.

e.g. REP 0

This makes the Flexowriter the monitoring peripheral. On the Flexowriter report level 0 is allowed only.

Report Levels

Level 0

If errors (see 7.1.6) occur then only one is given. If ENTER directive is read then only one unset, if any, identifier is given.

Level 1

If errors occur then up to 15 may be given. If ENTER directive is read, then all, if any, unset identifiers are given and a message is printed on the monitoring peripheral just before entering the program, giving the enter number, the time and the date.

Level 2

Level 1 reports are given. If ENTER directive is read, then the values of all the L identifiers set are also given; this printout also includes the block and position form of the L number and is in a form suitable for re-input.

Level 3

Level 2 reports are given. This level causes the old values of identifiers to be printed whenever they are cleared with a C directive.

If no REPORT directive is read then the Flexowriter is the monitoring peripheral, and report level 0 reports are given.

7.1.4.2.7 Abolish directive

[abolish directive]=ABO[l*?](VS[n]?)%

This causes Basic Input to stop reading and abolish the job. The number, if written will be printed (see 5.3.1l)

7.1.4.2.8 F directive

[f directive]=FVS[n]VS[n]VS[±?][field]%

e.g. F 12 24 L491

This directive is used to adjust the word in drum location whose address is V1-1. The first number (a say) and the second number (b say) specify a field in this word. (Bits in a word are D0 to D47). The field is Dj to Dk where

k = 47-b

and j = k-a+1

The value of the field expression which follows is added into this specified field (modulo the size of the field). Any number of F directives may follow a storable word.

In the above example the value of L491 is added into D12 to D23 of the word in drum location V1-1 (this usually the previous storable word).

7.1.4.2.9 Pullup directive

[pullup directive]=PUL[l*?]%

See NEST sequence 7.1.4.3.2. This causes Basic Input to lose the current set of L's and restore the nested set.

7.1.4.2.10 Process directive

[process directive]=PRO[l*?]%

This directive needs a separator (see 7.1.5.4 and 7.1.5.5)

CALL sequences (see 7.1.4.3.3) are remembered until PROCESS is read, when they are implemented. Basic Input Routine reads the "called" documents in the order in which it finds them. (Where a document is to be stored may be specified with the CALL sequence (see 7.1.4.3.3)). The "end" of a document is the END directive (see 7.1.4.2.18). After reading the last "called" document Basic Input Routine returns to read from the peripheral on which it read PROCESS.

The peripherals on which the "called" documents are loaded are reserved for the job with the number being 20 or above.

This directive will find a document on a composite magnetic tape document.

This directive insists upon reservation requests for peripherals (see 7.1.4.2.3) being implemented before returning to read from the current source.

7.1.4.2.11 Check directives

[check directives]=(X,Y)VS[±?][quantity]VS[±?][quantity]%

X-directive

X L1 4

This directive checks that the two quantities are equal. If so, the next Unit of Basic Input (see 7.1.4.5.1) is skipped, otherwise not.

Y-directive

e.g. Y 10 V3

This directive checks that the two quantities are unequal. If so the next Unit of Basic Input (see 7.1.4.5.l) is skipped, otherwise not.

7.1.4.2.12 Enter directive

[enter directive]=ENT[1*?]VS[quantity]%

e.g. ENTER 1

This directive needs a separator (see 7.1.5.4 and 7.1.5.5). It causes Basic Input to stop reading and enter the program. Before entering the program Basic Input

(i) relinquishes the current input peripheral if the number in the programmer's peripheral name is 20 or above, (see RES THIS directive see 7.1.4.2.3). If the document read is one on a composite document then the tape is left loaded.

(ii) implements the reservation requests for peripherals, the order of implementation is RES, NEW, SCR, and core store, if being decreased (see 7.1.4.2.3).

(iii) implements the monitoring conditions (see 7.1.4.2.21).

(iv) sets the timer to number of minutes specified with the TIME directive (see 7.1.4.2.17). If this directive has not been read, by default the timer is set to 1 minute.

(v) Depending on the report level, (see 7.1.4.2.6) information about the L identifiers is output. If the report level is >0 then a message on the monitoring peripheral will output to indicate when (the time and date is given) the program was entered.

e.g. ENTERED 11.05.57 1FEB65

(vi) looks at V64. If it is zero or positive Basic Input will give up its drum working space. If it is negative the lists of L's and their settings will be preserved on the drum. Unless altered by the programmer, V64 is zero or positive; this number indicates what type of print barrel this installation has (see section 14).

(vii) The core store from A66 onwards is cleared, if V64 is not negative then the program is entered. The quantity represents an integer (n say) which is the entry-point of the program. Drum locations whose address are 2n and 2n+1 should contain the chapter change pair. (i.e. a 150/50 instruction (see 5.3.50)).

A suitable chapter change pair may be written, for example as

```
150     L1     L2     50
00S     L3     L4      0
```

if it is known that the drum address L3 is less than 2^{15}.

7.1.4.2.13 Jump directive

[jump directive]=J(VS[quantity]*=4)%

e.g. J L16 L29 L2.1 L41

This causes Basic Input to stop reading and enter an interlude (see 7.1.5.2). The first quantity is the drum address, the second is the core store address, the third is the number of words and the fourth is the jump (entry) address for a chapter changing 150/50 pair.

The interlude is brought down and obeyed. The last instruction of the interlude is usually a 150/51 instruction which calls in chapter of Basic Input to continue reading more program in Basic Input Language.

Unless return is to Chapter 19 (see 7.1.5.2) this directive needs a separator (see 7.1.5.4 and 7.1.5.5)

7.1.4.2.14 Free directive

[free directive]=SVS[element]%

e.g. S 1

The [element] specifies a block of L's. The 64 L's of this block are freed (unset). No checks are made. These L's can then be reset. In the example, L64 to L127 inclusive will be freed. (i.e. these L's will be in state (i) see 7.1.2.2.l)

7.1.4.2.16 B directive

[B-directive]=B(VS[n]?)%

This causes Basic Input to stop reading and call in Binary and Map Input (see 7.3). This directive causes drum to be reserved - it is treated as though a RES *DRU directive had been read (see 7.1.4.2.3(iii)).

7.1.4.2.17 Time directive

[time directive]=TIM[l*?]VS[n]%

e.g. TIME 25

The [n] specifies the number of minutes the timer is to be set just before the program is entered (see 7.1.4.2.1 2). Number of minutes specified must be less than 547.

7.1.4.2.18 End directive

[end directive]=END%

This directive needs a separator (see 7.1.5.4 and 7.1.5.5).

In general it is used to terminate documents. Directives which may ask Basic Input to find and read these documents are USE (see 7.1.4.2.5) and PROCESS (see 7.1.4.3.3 and 7.1.4.2.10).

On reading END on a document which it has been asked to read by USE, Basic Input returns to read from the peripheral on which it read the USE directive; this is stored in A61.

On reading END on a document which it has been asked to read by PROCESS, Basic Input returns to implement any other call sequences and finally returns to read from the peripheral on which it read the PROCESS directive.

As Basic Input finishes reading one of these documents it relinquishes the peripheral on which the document is loaded (if the number in the programmers' peripheral name is 20 or above). If the document read is on a composite document (see 6.1) then when relinquishing the magnetic tape Basic Input leaves the tape loaded.

END at the highest level causes the job to be suspended awaiting rerun.

7.1.4.2.19 Q directive

[q-directive]=Q(VS?)[element]%

e.g. Q 2

The [element] specifies an L (in the example L2). Basic Input Routine skips the next Unit of Basic Input (see 7.1.4.5.1) if the L

(a) is set and its value known; i.e. if the L is in state (iv) see 7.1.2.2.1

or (b) has not been mentioned (at least not since it was last freed); i.e. if the L is in state (i).

Thus if the L is in state (ii) or (iii) this directive does not skip.

7.1.4.2.20 C directive

[c-directive]=CVS[element]VS[element]%

e.g. C 5 10

This frees (unsets) all the L's between the L specified by the first element and that specified by the second element inclusive. In the example L5 to L10 are freed. It also checks, for each L that there are no forward references to it or in its setting. If this check fails there will be an error indication (see 7.1.6). (i.e. it is an error to clear an L in state (ii) or (iii) see 7.1.2.2.1) The freed L's are then in state (i).

7.1.4.2.21 Monitor directive

[monitor directive]=MON[1*?]VS[element]VS[element]%

e.g. MON *SIG 2

The first [element] specifies an event and the second [element] a style (see 5.2). These monitoring conditions are set just before the program is entered (see 7.1.4.2.12)

7.1.4.2.22 Signal directive

[signal directive]=SIG[l*?]VS[quantity]%

e.g. SIG 30

The quantity specifies a drum address. The sign bit of the word in this drum register will be made 0.

7.1.4.2.23 Unsignal directive

[unsignal directive]=UNS[l*?]VS[quantity]%

e.g. UNS 123

The quantity specifies a drum address. The sign bit of the word in the drum register will be made 1.

7.1.4.2.24 Scratch directive

[scratch directive]=SCR[l*?](VS[n]?)VS[asterisk]MT[n]VS
(((P?)[n].[n].[n].(P?)(L?)VS[doc name]),0)%

e.g. SCR *MT1 F12.2.65 X/Y/2/*/**

This directive differs from the NEW directive (see 7.1.4.2.4) only in that Basic Input Routine writes the new Block 0 information with a 150/44 instruction.

7.1.4.3 SEQUENCES

These are terminated by the NORMAL directive (see 7.1.4.2.1)

7.1.4.3.2 Nest sequence

[nest sequence]=NES[l*?](EL L[element]VS L[element]*?)%

```
e.g.  NEST
L3.0     L20
L99      L2.1
L860     L1
NORMAL
```

This causes the current status of all the L's to be remembered (nested) and these L's to be freed forming the new set, and are regarded as being in state (i) (see 7.1.2.2.1).

Equivalences between identifiers of the 2 sets may be specified; the first L specified is one of the current set and the second L specified is one of the new set.

The nested set is remembered until a PULLUP directive (see 7.1.4.2.9) is read.

7.1.4.3.3 Call sequence

[call sequence]=CAL[l*?]VS[doc request name]
(VS+[quantity]VS[quantity]?)(EL L[element]VS L[element]*?)%

```
e.g.     CALL  A/DOCU/-    +900    A64
         L800  L0.1
         L801  L0.62
         NORMAL
```

Call sequences are remembered until PROCESS directive (see 7.1.4.2.1) is read.

[doc request name] specifies the "called" document which Basic is to find and read (a document on a composite magnetic tape document will be found).

With the CALL sequence, it is possible to specify the transfer addresses (+[quantity] specifies V1 and [quantity] specifies V2) to be used when reading this "called" document.

Succeeding lines specify pairs of L's which are to be equivalenced as in the manner of a NEST sequence (see 7.1.4.3.2) while the document is being read.

When the CALL sequence is implemented (i.e. PROCESS has been read.) Basic finds the document. If transfer addresses have been specified with the CALL sequence, Basic Stores the current (old) values of the transfer addresses, it then uses the specified transfer addresses for reading this document, and when it finishes reading this document (i.e. on reading END) it then restores the values of the transfer addresses to their former (old) values.

Basic having found the document, nests the L identifiers (i.e. the current status of all L's are remembered and then freed). L identifiers of the two sets may be identified; the first L being one of the current set and the second L being one of the new set. When the document has been read (i.e. on reading END) the nested set of L identifiers are restored.

7.1.4.4 Labels, equations

7.1.4.4.1 Core label

[core label] = (G,L,V)[element][right parenthesis]%

e.g. L1.2)
V4)
G1)

This causes the specified identifier to be set; its value equal to the core transfer address (V2). If G is written, the corresponding L is specified but the label is taken as an optional setting (see 7.1.4.4.5)

7.1.4.4.2 Drum label

[drum label]=[core label][right parenthesis]%

e.g. L3))

This causes the specified identifier to be set, its value equal to the drum transfer address (V1). If G is written the corresponding L is specified but the label is taken as an optional setting.

7.1.4.4.3 Equations

[setting] = L[element](VS?)[equals sign](VS?)[+?][field],
V[element](VS?)[equals sign](VS?)[±?](quantity]%

e.g. L26=35612+L1.3-V1
L2.3=A100
V7=V7+2

Each equation causes the identifier specified on the left-hand side to be set, its value equal to the quantity or field specified on the right-hand side.

If an L is specified, forward references in its setting are allowed. If the L specified is already set, a new setting will be treated as an error.

If a V is specified then this setting is its new value. No forward references are allowed in its setting.

7.1.4.4.4 Maximum facility

[max setting] = K[element](VS?)[equals sign](VS?)[±?][field],
W[element](VS?)[equals sign](VS?)[±?][quantity]%

e.g. K27=98+V7+V8
W6=V6+V7+1024

For K, an L is specified by the element. For W, a V is specified by the element. The equation sets the specified identifier equal to the largest element in the field or quantity specified or to zero if this is negative. In the above example L27 is set to have the value of the largest of 98, V7 or V8. V6 is set to have the value of the largest of V6, V7 or 1024.

7.1.4.4.5 Optional equation

[optional equation]=G[element](VS?)[equals sign](VS?)[±?][field]%

e.g. G99=255107-L22

If the L specified by the element (L99 in the example) is already set then this optional equation is ignored. If the L is not already set then this equation is taken as an ordinary setting of the L.

7.1.4.5 Composition of a Basic Input Program

7.1.4.5.1 Unit of Basic Input

[unit of basic input]=([any character except left parenthesis and EL] [any character except EL]*?)EL,[left parenthesis]EL[unit of basic input*?][right parenthesis]EL%

The check directives (see 7.1.4.2.11), and q-directive (see 7.1.4.2.19) if the check is satisfied skip one line unless that line is a left parenthesis in which case they skip further lines until a matching right parenthesis is found.

```
e.g.    X     L4    1
        ABOLISH

e.g.    X     V10   0
        (
              RES   *LP2   A/DOC
              V3=20
        )
```

7.1.4.5.2 Item of Basic Input Language

[item]=(VS?)[label?][item],[storable word?][comment],
([sequence][comment]*?),[directive or setting or equation]
[comment],[parenthesis][comment]%.

where

[comment]=(VS?)(|[any character except EL*?]?)EL%

```
e.g.    RES   *DRUM 2000
        150   L0    L1    50
        00S   L3    L2    0
        V2=A100

L0)L3))  14    L1.0  20          | set counter
        75    L1.1  0           | jump
        +800
        +10,10,10,10,10,10,10,10

L1.1)    00XY  A100  L2.0  A3

L10)L6=V2-L0
ENTER    0
```

Basic Input reads items until ENTER, J, B, ABOLISH or END (at highest level) is read.

7.1.5 The Basic Input Program

This program consists of several chapters all of which read Basic Input Language formats.

When JOB directive (see 5.7.2) is read by OMP, OMP loads a chapter of Basic Input into the job's reserved core-store and enters this chapter which continues by reading from the same source (on which JOB was read.). This chapter tries to get 128 more words of core if 1008 have been allocated, and then loads Chapter 1 to continue reading.

RERUN directive calls in Chapter 1.

Chapters of Basic Input are called in by obeying 150/51 with Y=A0 instruction (see 5.3.51). X=2n causes a chapter to be loaded whereas X=2n-1 causes the same chapter to be loaded and entered. X=0 is not allowed.

7.1.5.1 Chapter 1 (i.e. X=1 or 2 in 150/51)

When this chapter is called in

A64 contains an integer (zero or positive) which indicates the type of print barrel (see section 14 and 7.1.4.2.12) that this installation has.

A65 contains the datum point of the job.

A66 is used by Basic Input to store constants for use by autocodes. These constants are explained as follows.

Call $[A66]_u$ PROGLIM. Then PROGLIM is the address of the core-store register in Basic's core working space whose contents represent as an integer the amount of drum reserved for the program being stored. This is the address of the drum register of the start of Basic's drum working space. This number is initially taken from A55 and A55 cleared.

Call $[A66]_m$ BINDEX. Then BINDEX is the address of a core store register. This register is the start of Basic's index to its core working space. Each word of the index represents a buffer of 64 words of core-store; each buffer containing some words from Basic's drum working space. The format of the index words is

OOS drum address core address 0

where the drum addresses are relative to [PROGLIM]. TY=1 means that the block is to be written to the drum (it has been changed). RX=1 means that the block is "locked down" (it must not be written to the drum).

If [A64]<0 then Basic does not give up its working space and does not clear the core-store A66 onwards.

The entry-point to Chapter l is A67.

This chapter sets V1=0 and V2=A64.

Chapter 1 then

(i) assumes that the input source is in A62 (if the sign bit is 1, then the modifier half is taken as the drum address from which to read.)

(ii) assumes that the number of words of core available for Basic's use is in A56 (it must be at least 1008 otherwise reservation violation action will occur.)

(iii) uses the contents of A55 to get an initial setting of [PROGLIM] (i.e. to find where Basic's own drum working space begins - it will be at a multiple of 64). Drum is reserved in units of 512 words if possible, otherwise in units of 64. If Basic asks for more drum and none is available then the job is halted NO DRUM. (If the request is for more than would be available if there were no other programs in the machine then Basic abolishes 10.)

Unless "asked to" do so, Basic does not alter the contents of core-store registers, A3 to A12, A56, A62 and AV56 upwards nor of the drum registers D0 to D([PROGLIM]-1)

All other registers may be altered by Basic Input during input.

Chapter 1 sets monitoring on *PFN (Style 7) and *SIG (Style 0)

7.1.5.2 Interludes

An interlude is brought down from the drum and entered by a J directive (see 7.1.4.2,13).

An interlude is given one minute. There is no check that all L's so far referred to have been set. If any RESERVE or NEW or SCRATCH directives for reserving peripherals have been written the interlude must be preceded by a PROCESS directive (see 7.1.4.2.10) to implement these requests. Monitoring on *PFN (Style 7) and *SIG (Style 0) will be set; other monitoring styles are default. Restarts are set for the current input source (see 7.1.7)

When an interlude is finished it is usual to "return to" a chapter of Basic Input which then continues reading in more program in Basic Input Language (i.e. the last instruction obeyed in the interlude will probably be a 150/51 instruction with Y=A0).

What an interlude must preserve and what information will be lost depends upon which chapter of Basic Input is "returned to".

7.1.5.2.1 Return to Chapter 1

i.e. 150 1 A0 51

This instruction causes chapter 1 of Basic Input to be loaded and entered. (see 7.1.5.1)

The interlude must preserve (i.e. leave in)
A56 — the amount of core store available for Basic Input's use.
A62 - the current input source.
A55 must be reset to contain the amount of drum required by the program.

Lost will be

(i) settings of all L's and the V's except V3 to V12.

(ii) all memory of forward references not filled in.

(iii) contents of the input buffer or buffers.

On 7-track this consists of the line following the J directive.
On 5-track this consists of up to 16 characters after the CR LF terminating the J directive.

On cards this consists of the contents of the card following the card containing the J directive.

On magnetic tape this consists of the rest of the block containing the J-directive.

On drum this consists of the rest of the word containing the NL terminating the J directive.

(iv) The report level unless 0.

As this chapter switches on monitoring on *PFN (style 7) and *SIG (style 0) the interlude may switch these off if required.

7.1.5.2.3 Return to Chapter 3

i.e. 150 3 A0 51

This instruction causes Chapter 3 of Basic Input to be loaded and entered.

The interlude must preserve in the core
A44 to A65 and A500 to AV56 (except A62 - see below)

The interlude must preserve on the drum D [PROGLIM] upwards. (i.e. Basic's drum working space).

Lost will be the contents of the input buffer or buffers as for Chapter 1

A change of input source (i.e. in A62) will be noted and Basic Input will continue reading from the new source.

As this chapter switches on monitoring on *PFN (style 7) and *SIG (style 0) the interlude nay switch these off, if required.

7.1.5.2.19 Return to Chapter 19

i.e. 150 19 A0 51

This instruction causes chapter 19 of Basic Input to be loaded and entered.

The interlude must preserve in the core .A44 to A250 and A500 up to AV56.

The interlude must preserve on the drum D[PROGLIM] upwards.

The contents of the input buffer or buffers will not be lost.

This chapter does not switch on monitoring on *PFN (style 7) and *SIG (style 0) and so the interlude must not switch these off.

7.1.5.3 The GNC System (Chapter 17)

The get-next-character (GNC) routines of Basic Input provide a usable self-contained system for input from 7- or 5- track paper tape, cards, drum or magnetic tape.

The instruction

```
150   17    A0    51
```

causes Chapter 17 of Basic Input to be loaded into the core and entered to set up the GNC. Before this instruction is obeyed the name of the input source must be put into A62 and the link address into A17. The 150/51 when obeyed causes a GNC routine appropriate to that type of device to be set up and then return to the link address.

The registers used by the GNC's are A13 for checksum and
A14 to A17, A44, A48, A51 to A53, A62, A66 to A250.

All of these (except A14 to A17) must be preserved between entries to the GNC.

To set up, for example, a GNC for a 7-track reader (*SR1 say)
the sequence of instructions might be

```
14    A62   *SR1              | name of Input source
14    A17   L0                | set link
150    17   A0    51          | set up GNC
L0) an instruction            | return here
```

To use the GNC, the instruction

```
86    (A53)  A17
```

is obeyed. This enters the GNC which produces the next correct character from that source at the l.s. end of A44 in standard internal code (see 5.6.1) and returns to the instruction following the 86 instruction which caused entry.

Comments (vertical bar character and those following until NL) and ER characters are ignored within the GNC. Only one SP character will be produced for a string SP characters (i.e. VS). Spaces at the beginning and end of the line will not be produced. After the entry to chapter 17 one or two NL characters may be produced which have no counterpart on the input medium.

7-track conventions

The code accepted is the Flexowriter code (see 4.3.3). Lower and upper case letters are regarded as equivalent. Redundant shift characters are ignored, (as are the control characters, PN, PF, PT, ST.) TB is a VS character (tab positions are 16 SP characters apart). Characters are converted into the standard internal code (see 5.6.1). Unassigned characters are ignored.

A line is considered to have a limit of 118 printing positions.

Overprinting (e.g. use of BS character) at a printing position is interpreted thus:-

a line with crossed parentheses ╫ is completely ignored.

C (a character) BS ER is equivalent to ER

ER BS C is equivalent to ER

SP BS C is equivalent to C

C BS SP is equivalent to C

C BS C (the same character) is equivalent to C

C1 BS C2(different characters) gives error character (character-value 14) except ╫ (see above).

When the 118th position is reached, then further characters are treated as overprinting in this position.

BS character at the beginning of the line "leaves you" at the beginning of the line.

5-track conventions

The code accepted is given in 4.3.3. Characters are converted into the internal code (see 5.6.4). Redundant shift characters on a line of information (i.e. with other printing characters, SP is a printing character, ER is not) "don't get through", but if the job has a monitoring peripheral, then a message is printed giving error 26 and the line number, but this is not treated as an error.

≥ gives error character (character-value 14)

→ is treated as vertical bar.

CR not followed by LF gives error character. ER is not allowed between CR and LF. If since the last LF only non-printing characters are present then LF is allowed.

Card conventions

The code accepted is given in 5.6.2. One card is one line.

Magnetic Tape and Drum

The code accepted is given in 5.6.1. For magnetic tape conventions also see 6.1. The magnetic tape GNC ignores blocks written by :DUMP/ (i.e. blocks whose word 0 contains 58 in Z-address field).

7.1.5.4 Directives Losing Input Buffers

After the following directives have been processed the rest of the input buffers is lost (c.f. 7.1.5.2.1) and a line containing these directives should be followed by a separator see 7.1.5.5

(a) J (except if return is to Chapter 19)

(b) USE (or COMPILER)

(c) PROCESS

(d) RES *CORE (if reservations are increased)

(e) READ

(f) ENTER

(g) END

7.1.5.5 Separator

7-track paper tape - Leave 1 blank line

5-track paper tape - At least 13 FS followed by CR LF

Cards - 1 Blank card

Magnetic Tape - Start a new block

Drum - Start a new word

7.1.5.6

7.1.5.6.1 Before reading each line of input Basic asks for 1 minute.

7.1.6 Errors

7.1.6.1 If an error is encountered e.g. an impermissible combination of characters, then Basic outputs an error report (see 7.1.6.3) on the job's monitoring peripheral (normally this will have been specified with the REPORT directive (see 7.1.4.2.6)), or on the Flexowriter if the job has no monitoring peripheral.

7.1.6.2 If an error has been encountered and the job has no monitoring peripheral, then the job is abolished 1. If the job has a monitoring peripheral, then the job is not abolished but Basic continues reading the program keeping a count of errors encountered. If 15 of these errors are obtained then the job is abolished.

Note that if a serious error (see 7.1.6.4) is encountered the job will be abolished before this count is 15.

7.1.6.3 The error report may consist of two lines

The first line is

ERROR n1 ON g V1 n2 V2 n3

Where n1 is the error number (see 7.1.6.3)
n2 is the current drum transfer address
n3 is the current core transfer address
g is the geographical name of the current input device;
if the current source is the drum then the drum source address is given.

The second line contains characters of the line containing the error; the first character on this second line is the incorrect character and then follows the rest of the line. Not all error numbers cause printing of a second line e.g. Error 26, and in the case of Error 16, the second line is the reset L.

7.1.6.4 When Error 1, 24, 25, 27, 30 or 32 occurs the program is abolished 2

If a RES *DRUM directive or extra drum needed for working space asks for more drum than would be available if there were no other programs in the machine, Basic abolishes 10.

7.1.6.5 Error Numbers

1. Error in a J-directive
2. Wrong character at beginning or end of line
3. Error in X or Y directive
4. Error in F-directive
5. Error in an S or C directive
6. Error in right-hand side of a setting
7. Too many datum points in a field
8. Error in left-hand side of a setting
9. Forward reference in right-hand of V setting
10. Forward reference in quantity
12. Comma after a forward reference
13. Z replaced
14. Wrong character in function part, or wrong character before an address-field (e.g. no space character between fields)
15. Last digit of function 8 or 9
16. Reset label, the second line is the reset label. This error does not count as one towards the 15. Unset label is not an error.
17. Error in Q-directive
18. Overflow in some fields
19. Error in monitor directive
20. Error in document name
21. Error in TIME directive
22. Error in REPORT directive
23. Error in RES NEW or SCR directives
24. Error in READ or USE directives
25. Error in ENTER directive
26. Error in conventions of input medium (e.g. CR not followed by LF 5-track, or reading from drum outside reservations)

27. Error in B-directive

28. V1 not sensible

29. Error in element

30. Because of errors found, program cannot be entered.

31. Clearing unset label, Label is cleared - this error does not count as one towards the 15.

32. Error in GNC, e.g. asking Basic to read from device of incorrect type.

33. Error in CALL or NEST sequences, e.g. duplicated right-hand side equivalences

34. Error in SIG or UNS directives

35. Too many blocks of L's and forward references (limit is 32767)

36. Impossible document name

37. Too little core store for L's index, give Basic more core

38. Too many forward references in right-hand side of L setting. (Limit is 31)

7.1.7 Restarts

7.1.7.1 7-track Paper Tape

When a parity failure occurs on the 7 track reader from which Basic Input Routine is reading then one of the following occur.

7.1.7.1.1 If Basic is reading anything other than binary and map format or a semi-built-in program then a message on the Flexowriter requests the operator to carry out a specified restart procedure (See 6.4.1).

7.1.7.1.2 If Basic is reading binary and map format following a B Directive (see 7.1.4.2.16) then the action is to abolish the job.

7.1.7.1.3 If Basic is reading a semi-built-in program and storing it on the drum, then the action is to ask the operator to re-load the semi-built-in program. The message CHECKSUM FAIL in case of 7-track or NONSTAD TAPE in case of magnetic tape may be output.

7.1.7.2 Cards (80 or 65 column)

When a failure in reading a card occurs then one of the following occur.

7.1.7.2.1 Illegal Punching

Illegal character is given character-value 14, (see 5.6.1) which will produce an error (see 7.1.6) when the line is processed, though if to the right of vertical bar character, the error character is ignored and so illegal punching in comments is permitted.

7.1.7.2.2 Other Events

The action is to print the failure message and the standard restart message (see 6.4.2).

7.1.7.3 Magnetic Tape

If a reading failure occurs on a composite document then the message NON-STANDARD COMP.TAPE is output.

7.2 SYMBOLIC INPUT

INDEX

7.2.G Symbolic Identifiers, Labels. Drum Labels and Equations

.1 Symbolic Identifiers.
.2 Labels and Drum Labels.
.3 Equations.

7.2.H Directives

.1 List of Available Directives.
.2 START and ENTER directives. Compiled.
.3 NOSIGNAL directive. Compiled.
.4 NAME

7.2.J Numbers - Special Formats

.1 Numbers.

.1 Fractions.
.2 Integers
.3 Standard Packed Numbers.
.4 Layout and Range of Standard Packed Numbers.

.2 Directives for Special Formats, NORMAL directive.

.1 DOUBLELENGTH directive.)
.2 FLOATINGPOINT directive.)
.3 MASK directive.)
.4 MIDPOINT directive.) All implemented
.5 MIXEDNUMBERS directive.)
.6 OCTAL directive.)
.7 PACKEDNUMBERS directive.)
.8 STERLING directive.)
.9 TEXT directive.)

7.2.L Routines - Program Organisation

.1 ROUTINE and END directives. Implemented.
.2 Notation for routines.
.3 References, the identifier and prefix system.

.1 Prefixes, local and global references, references to a word in another routines.

7.2.M Library Subroutines and Assembly

.1 Library subroutines.
.2 Assembly - LIBRARY directive and 1086 macro. Implemented.

7.2.N Macro - Instructions and the + and > notations

.1 Standard Macros.
.2 Private Macros.
.3 + and > notations.

7.2.P Chapters and Chapter Changing

.1 CHAPTER directive. Implemented.
.2 TRANSFER directive. Compiled.
.3 Layout of a chapter.
.4 The & notation.
.5 Chapter - changing macros. Compiled.

7.2.Q Monitoring

MONITOR directive. Compiled.

7.2.R Messages including Error Numbers

7.2.S Dumping Facility.

7.2.A INTRODUCTION

7.2.A.1 Symbolic Input is name of the standard language for writing programs in Orion machine code. A program written in this language is called a Symbolic Program.

Symbolic programs are punched in a natural way; on 7- or 5-track paper tape or cards and are then read by a semi-built-in program called the Symbolic Compiler which translates the Symbolic program into Basic language. The resulting program may then be run as part of the same job, or the Basic version may be output on punched paper tape, cards or magnetic tape to be run as a separate job.

7.2.A.2 A run involving the symbolic compiler divides into four distinct stages. We will consider using paper—tape but the following details apply equally to cards or magnetic tape.

The stages are

(i) Reading the job-tape.

(ii) (Possibly) reading in the corrections tape,

(iii) Reading in the program tape.

(iv) Either running the program or outputting the Basic version of the program or outputting the corrected Symbolic version of the program.

The various courses of action to be followed are specified by directives on the job-tape. Section 7.2.C describes the permissible contents of the job-tape. The corrections tape is fully specified in section 7.2.D and the remainder of section 7.2. describes the facilities available to a Symbolic program.

7.2.B NOTATION AND CHARACTER EQUIVALENCES

7.2.B.1 Mark 2 - **

In this section 7.2 the symbol ** denotes Mark 2 facilities, all of which are available.

7.2.B.2 Character Equivalences

The standard Orion codes for 5-track and 7-track punched paper tape and cards are used. The following sections are written in terms of 7-track paper tape characters. Some of these are not available in the 5-track code and the following equivalences apply throughout.

7-track character	5-track character
:	≠
&	£
|	→

7.2.C. JOB TAPE LAYOUT, PERIPHERAL AND DOCUMENT NAMES

7.2.C.1. Document and Document Request Names

7.2.C.1.1 Document names must adhere to the conventions (see 6.1). A name consists of 8 components separated by solidus, though non-significant null components and corresponding solidi need not be specified. Each component is up to 8 characters; letters, digits and point are allowed. Where in a directive a document name is specified, Symbolic Input allows * to be written instead of a component and for this component it stores today's date. ** means store the time as this component. + (plus) as a component means the document is composite (magnetic tape only).

7.2.C.1.2 Document request names. When requesting a document (e.g. following a CORRECT directive) the programmer may or may not specify the complete document name - the facility of using - (minus) instead of a component to mean "don't care what this component is" is allowed. The first two components must be specified however. For example

```
ORION/ONE/-/TWO
```

is used to request any document with name ORION/ONE/any component/TWO with the last 4 components being null.

```
ORION/ONE/TWO-
```

is a request for a document whose first three components are ORION/ONE/TWO and any 5 components.

The document to be found may be on a composite magnetic tape. If Symbolic is reading from a composite tape then on finishing with the tape it relinquishes and selects.

If the required document follows on the same tape as the request name, then the word THIS may be used to specify it. THIS is allowed only on the job tape.

7.2.C.2 Layout

The job tape for a compiling run starts with JOB directive (see 5.7.2.1) followed by 2 lines

```
COMPILER (or USE) SYMBOLIC
```

and a blank line.

Symbolic needs at least 2544 words of core. If there is no request for core with JOB directive then the compiler firstly request 3056 words, but if this is not available then it reduces the request by 64 words until the request is granted unless 2480 has been reached in which case the job is halted NO CORE. If there is a request for core with JOB directive which is 2544 or more then Symbolic uses this core given to it; if the request is for less core then the action is as for no request.

The rest of this job is read by the Symbolic Compiler and may contain some or all of the following directives, each of which requires a new line.

Note that section 7.2.C.3 gives a description of peripheral names.

7.2.C.2.1 BASIC VS Peripheral Name VS Date (for magnetic tape only) VS Document Name.

e.g. BASIC *SP1 MY/BASIC/VERSION/OF/SYM/*/**

This directive causes a Basic version of the symbolic program to be given on the specified peripheral device. Several BASIC directives may be given so that several copies are obtained.

7.2.C.2.2 CORRECT VS Document Request Name

WITH VS Document Request Name

GIVING VS Peripheral Name VS Date (for magnetic only) VS document name.

These directives are only used if it is desired to modify the program.

The document names are respectively (i) the document to be corrected, (ii) the document contain the corrections, and (iii) heading of the output document, the form of which is specified by the peripheral name. The Date is only included if the specified peripheral is a magnetic tape, it is written as described in section 7.2.C.2.4.

If the directive BASIC is used then the directive GIVING is omitted. In this case the output will be the corrected program in Basic Input Language.

If the directive BASIC is omitted the output is the corrected symbolic program, and if both BASIC and GIVING are omitted the corrected program is run.

Several GIVING directive may be given, so that several copies are obtained.

7.2.C.2.3 RESERVE VS Peripheral Name VS Document Request Name

(a) Peripheral devices are reserved for the job by this directive. Each device requires a RESERVE directive on a new line. The document name specifies the name of the document that is to be input or output. For magnetic tape documents symbolic will find one on a composite tape.

To write on to magnetic tape the NEW or SCRATCH directives may be used, this is described in the following section.

(b) Reservations may be made for space in the drum store and core store as follows:

RESERVE VS * DRUM VS No. of words in drum store required. Compiled.

RESERVE VS * CORE VS No. of words in core store required. This directive is compiled only. It is advisable to have only one request for core.

In both cases a symbolic identifier may be used instead of the number, which would then be specified in the program against this identifier.

Note that on a drum load and go any RES DRUM directives are ignored. If on a load and go run no RES DRUM or CORE directives are read then the program is entered with the amount used by the Compiler.

The RESERVE directive may appear anywhere on the job tape or program tape.

7.2.C.2.4 NEW and SCRATCH directives

NEW or SCR VS Peripheral Name (magnetic tape) VS Date VS Document Name.

e.g. NEW *MT2 P1.6.66 MY/DOCUMENT/*

These directives ask for a new Block 0 to be written onto the tape. (If the peripheral has not already been reserved then a scratch tape is requested). NEW will use 150/41 instruction (see 5.3.41) so that the Block 0 is written straight away whereas SCR uses 150/44 and Block 0 is written only if, for example, the run is successful.

The above example ensures that the tape on the deck known to the program as *MT2 will have the new Block 0 information, i.e. Date meaning 1st June 1966 with write permit bit (D0) set because P before is present and with date control bit (D24) not set because P after the date is not present. The document name will have first 2 components as specified, the third as today's date and the last 5 null.

These directives may appear anywhere on the job or program tape.

7.2.C.2.5 TIME VS No. of minutes

This optional directive specifies that when the program is actually entered, the timer is to be set to the number of minutes specified and if the program uses more than this amount of mill time it has run far too long a time (perhaps having got into a loop) and the default action is to halt the program. If no time directive is given, the timer is set to 1 minute.

7.2.C.2.6 Report Directive

REPORT VS Level VS Peripheral Name or number VS Document Name

REP 2 *SP1 MY/REPORTS/*

This directive causes printing on the specified peripheral during input and reserves the peripheral if not already reserved and sets it as the job's monitoring peripheral. This directive is implemented so that reports about the Symbolic program are given while Symbolic compiler is reading it in, and it is also compiled so that reports about the Basic version are given while Basic Input is reading. Note that if the Report directive is given after a BASIC directive then the REPORT directive is implemented but not compiled.

An integer may be specified instead of peripheral name which means that a slow output device is required and that its programmer's peripheral name is to have the number given. If a slow output device has already been reserved with this number then it is used, otherwise the first available slow output device is reserved with the programmers peripheral name having this number.

```
REP   2     5     MY/REPORTS/*
```

would (if no output peripheral numbered 5 has been reserved) reserve the first available one, say a line printer as *LP5 and use it and set it as monitoring peripheral.

The levels of printing are

0 Error reports (see 7.2.R.) only.
1 All directives encountered in addition to error reports.
2 The values of all identifiers which are set or referred to and Level 1 printing with the addition of the Basic Input interpretation of the directives.

For levels 1 and 2 a peripheral must be specified and for level 0 if no peripheral is specified, the Flexowriter is used. Level 0 is assumed unless another level is specified.

7.2.C.2.7 READ VS Document Request Name

The READ directive can occur at any time; it causes the Symbolic Compiler to continue reading from the specified document. At this point usually the program will be read in. This directive needs a separator (see 7.1.5.5)

7.2.C.2.8 The directive MONITOR (section 7.2.Q) and NOSIGNAL (section 7.2.H.3) may also appear before the CORRECT directive if required.

7.2.C.2.9 End directive

This has several meanings which are:-

(i) END used to terminate the job tape. In this case it appears before any storable words and is implemented - END telling Symbolic that the job tape has been read and to continue. For example it is used on a corrections run.

```
JOB CORRECT
USE SYMBOLIC
1 Blank line
CORRECT MY/SYM/PROG/-
WITH    MY/CORS/-
END
```

Note that the following is allowed, for example

```
JOB CORA
USE SYMBOLIC
1 Blank line
CORRECT MY/SYMBOLIC/PROG/-
WITH THIS
END         |/Terminating job tape effect
1 Blank line
ALTER 3 AFTER FRED 1
60 JOE A1
END         |/ this terminates the corrections tape.
```

(ii) END used to terminate the corrections tape(see 7.2.D.1.) i.e. terminating ALTER sequence as in the last END in the above example. In this case END is implemented and Symbolic continues.

(iii) END used with ROUTINE directive (see 7.2.L.1) eg. END SORT/

In this case END is implemented and Symbolic continues.

(iv) If an END directive appears after any storable words on the program part and it is not either case (ii) or (iii), then Symbolic will compile this directive and will terminate compilation. If a basic version of the Symbolic program has been produced then this basic version may be run in the following way, for example.

```
JOB RUN
USE MY/BASIC/VERSION/OF/SYM/PROG/-
| Blank line
V1 = 593
60 A103 A5
ENTER 4
```

Where the document MY/BASIC/VERSION/OF/SYM/PROG does not have an ENTER directive and is terminated by END.

7.2.C.3 Peripheral Names

Peripherals are classified so that if a particular peripheral is not available another in the same group can be used without changes in the program.

The groups are each specified by an asterisk and two letters as follows:

*SR	seven-track tape reader.
*FR	five-track tape reader.
*CR	card reader (80 columns)
*MT	magnetic tape
*SP	seven-track tape punch
*FP	five-track tape punch
*CP	card punch
*LP	Line printer.
*VR	Card reader (65 columns)

If the program uses three magnetic tape decks, say, they may be referred to as *MT2, *MT3. The number following the letters must be less than 32. Note that the numbering need not be dense and the omission of a number implies zero i.e. *TR implies *TR0.

The two letters are stored in 5 bits each (from A=1 and B=2 to Z=26) followed by the 5 bits of the integer to give a 15 bit "number".

e.g. *SP17 is compiled by input as

$$19 \times 2^{10} + 16 \times 2^{5} + 17 = 19985$$

This number may be inserted on the job tape in place of the code specified above if desired.

Peripheral names are not ordinary identifiers. They can be used whenever an identifier can be used but they cannot be set by the user, i.e. they cannot be used as labels or appear on the left hand side of an equation. Note that MT3 is an ordinary identifier, distinct from *MT3.

During compilation Symbolic may use peripheral devices for its own purposes in which case the number of the name for this device will be 31, 30 etc. The first device being 31, the second 30 and so on. For example the tape used for storing "on TAPE" is usually *MT31. When Symbolic has finished with a device it relinquish it; if the document is on a composite magnetic tape then on finishing with it, Symbolic relinquishes and selects.

7.2.D CORRECTIONS

Corrections to a symbolic program are in the form of a text editing process e.g. "Alter the third line after the line labelled BUZZ".

A "line" is any line on the printout of the program which contains any visible printing, except that ER and any including the composite character ╫ are deemed to be invisible.

7.2.D.1 The Correction Directive ALTER.

The single correction directive "ALTER" permits the insertion, deletion and alteration of lines of the original symbolic program.

The layout is as follows:

ALTER VS Line (see 7.2.D.2) VS Number NL
Instructions etc. if any NL
Terminating directive.

'Number' gives the number of lines of the original program which are to be deleted. If it is zero, the new lines following the ALTER directive are inserted into the program immediately after the line named. For example, if the number is 5, five lines beginning with the line named are deleted from the original program before the new lines (if any) are inserted. It is permissible for an ALTER directive with a non-zero number to be followed immediately by the terminating directive (see next paragraph), thus making deletion without insertion possible.

The "terminating directive" will either be the next ALTER directive or the END directive which signifies the end of the corrections tape.

The corrections may appear in any order on the corrections tape. If two corrections attempt to alter the same line, an error is reported.

7.2.D.2 Identification of the first line to be corrected

The first line to be deleted (or the line preceding insertions) is written in the ALTER directive in the following notation.

A line on the printout is referred to as a line with a label, if it possesses one, otherwise as the line so many after a line which is labelled. Thus a line labelled BUZZ is identified as BUZZ, and the third line after BUZZ is identified as 3AFTER VS BUZZ where there may be a VS between 3 and AFTER. AFTER may be abbreviated to AFT or just A if desired. The expression 0AFTER VS BUZZ is not allowed.

Note that a line may be specified by any preceding label but not by a following one,

e.g. in the sequence of instructions

```
            FRED)   14  DATA      NUM
             JOE)   14  MOD        0
                    14  (ADDR)     0
             JIM)  00Y  (ADDR)   (DATA)  MOD

            *       81   JIM      (CT)   MOD
           JOHN)    00   DATA      CT
```

the line * may be identified as

```
                          4AFTER FRED
                       or 3AFTER JOE
                       or 1AFTER JIM
```

but not as being the line before JOHN.

It may be necessary to refer to line BUZZ of routine SORT/ as the line labelled SORT/BUZZ to distinguish it from a line labelled BUZZ in another routine. If such a line is referred to simply by the label 'BUZZ' the first line with its last component being 'BUZZ' will be corrected.

It is permissible to refer to a line not in the routine SORT/ as 10AFTER VS SORT/BUZZ if necessary. Directives of the form ROUTINE SORT/ and END SORT/ may be corrected; however, all references on the corrections tape to routine names and instructions refer to the original symbolic program. For example, if the directive ROUTINE SORT/ is corrected to ROUTINE TYPE/ any correction to the line labelled BUZZ in this routine should still refer to it as SORT/BUZZ.

7.2.D.3 Example of a Corrections Tape

It is desired to correct part of the following program:

```
ROUTINE SORT/
BUZZ)    00   A1     A2
         04   A3     A5
         14   A6     100
         75   JOE    0
```

to give a new version

```
ROUTINE SORT/
BIFF)    15   A100   1001   A63
         17   A3000  63     A30
         04   A3     A5
         14   A6     100
         110  A15    A400   A19
         75   JOE    0
```

i.e. the line BUZZ is to be deleted and replaced by the lines

```
BIFF)    15   A100   1001   A63
         17   A3000  63     A30
```

and the line

```
         110  A15    A400   A19
```

is to be inserted further down.

The following corrections will produce this result:

```
ALTER          BUZZ   1
BIFF)    15    A100   1001   A63
         17    A3000  63     A30
ALTER          2AFTER BUZZ
         110   A15    A400   A19
END
```

The line name 2AFTER VS BUZZ is correct even though the label BUZZ has been deleted and an extra line inserted. This name should not be written as 3AFTER VS BIFF because all corrections are applied to the original version of the program.

Note that a line with only comments is a line and is included in the line count.

7.2.E PUNCHING CONVENTIONS

Symbolic Input is primarily for use with 7-track paper tape (Flexowriter code) and 5-track paper tape (Pegasus teleprinter code) see 4.3.3.

7.2.E.1 7-track Orion Flexowriter Code

The punching rules for 7-track tape with Symbolic Input are chosen so that if the printout is correct then so in general also is the tape. Note 7.2.E.1.2. The following rules therefore apply primarily to the printout. It is most important that the tape should not be manually moved back in the Flexowriter punch unit.

Each tape should commence with a leader of at least 6 ins. (15cm.) of run out (repeated UC) and must terminate with a similar trailer. It is recommended that a trailer should be terminated by NL; several ER's and ST.

The last significant line on any tape should be followed by at least 3 NL's. This is also applied to lines including the directives ENTER and READ.

Run-out may also appear whenever convenient in the body of the tape. Run-out should always be preceded and terminated by NL.

7.2.E.1.1 The Flexowriter carriage can print up to 115 characters or composite characters on any one line but this is restricted to 108 by setting the left hand margin to start at character position 8. After position 8 there are six tabulating positions set at intervals of 16 print positions.

7.2.E.1.2 Overprinting is restricted. SP and ER may overprint or be overprinted by any character (C say) in which case the compound characters will be C and ER respectively. The only other overprinting allowed is crossed parentheses ╫ (fully described in the next section) and any character overprinted by itself. Hence P BS R or . BS LC, whilst appearing on the printout will be rejected.

BS is assumed to work in all positions except at the left-hand margin, hence NL BS has the effect NL. Any combination of SP, BS and TB may be used to position the carriage for overprinting.

7.2.E.1.3 If the composite character ╫ i.e.) BS (, is read anywhere on a line, the whole line will be ignored by Input. ╫ may be erased if desired and also reinstated by being printed in some other vacant position. This may be carried out as many times as proves desirable, and the line will be completely ignored so long as one ╫ remains unadulterated.

7.2.E.1.4 ER is ignored everywhere on the printout.

7.2.E.1.5 UC and LC may appear anywhere, they will be ignored if non-significant. No distinction is made between upper and lower case letters.

7.2.E.1.6 Vertical bar '|' in a line indicates that the characters following it as far as NL are comments. Comments are output as part of a corrected symbolic program. Crossed parentheses in the comments will cause the whole line to be ignored on both sides of the vertical bar.

Vertical bar may immediately follow the main part of the line or may be preceded by any number of spaces.

In this document reference to a 'line' will mean the line up to a vertical bar which if present will be regarded as NL.

7.2.E.1.7 PT, ST. PN, PF and all unassigned character values are ignored by Symbolic Input. Hence they are not copied in corrections.

7.2.E.1.8 The symbols [,], <, ?, ', ½, while quite legitimate as part of comments will be treated as error symbols in the main part of a symbolic line. Other special characters must be used in their correct context.

7.2.E.1.9 Although the characters Å, Ä, Ö are available on some Flexowriters, they may not be used in symbolic identifiers, routine names, job or document names.

7.2.E.2 ** 5-track tape - Pegasus/Mercury/Sirius Teleprinter Code

The checking of 5-track tape in Input is less effective than with 7-track tape since most of the characters will be in use.

7.2.E.2.1 Non-significant shifts or pairs of shifts are permitted, though a message will be given.

7.2.E.2.2 Runout of tape (which consists of repeated FS. i.e. "blank" tape) must be terminated by CR LF. CR must be followed by LF and LF must be preceded by CR or LF. ER is ignored in all positions except between CR and LF.

7.2.E.2.3 No facility is provided with 5-track tape Symbolic Input for abandoning a line. The only procedure apart from tape editing is to erase all characters in the line.

7.2.E.2.4 This document will deal mainly with 7-track input, any differences in procedure due to using 5-track tape will be noted.

7.2.E.2.5 The character → is used in the same way as '|' in 7-track tape.

7.2.E.3 Punched Cards

The code and punching conventions used are those of the Orion Monitor Program. This is described in section 5.6.2 of this Manual.

7.2.F ADDRESSES AND INSTRUCTIONS

The first part of this section deals with the primitive instructions i.e. instructions in which the contents of the address fields are written as numbers; basic addresses or peripheral names. 140- and 141- instructions are dealt with in section 7.2.P.3.

7.2.F.1 Layout of Primitive Instructions

An instruction is normally written:

Function VS X-address VS Y-address VS Z-address NL

7.2.F.1.1 The function is written in the usual mixed decimal octal code. VS before the function will be ignored.

7.2.F.1.2 Modification of the X and Y addresses is indicated by punching X or Y or both immediately after the function. A signalled instruction is indicated by writing S immediately after the function. X, Y and S may appear in any order.

7.2.F.1.3 The function and letters are terminated by VS, typical functions are:

TB 00SP)	
114XYSTB)	7- track
SP SP 81λSXφSP	5- track **

7.2.F.2 Addresses

The numbers in the X, Y and Z addresses may be

(i) Accumulator addresses
(ii) Register addresses
(iii) Constants
(iv) Drum addresses
(v) Names of peripherals

7.2.F.2.1 The core store locations are referred to by the programs as A0, A1,...,A63 (which are the program's accumulators) and as A64 onwards, which are the ordinary registers. Accumulator and register addresses are terminated by VS. The datum point is added, where appropriate, to the addresses on input before the program is executed.

7.2.F.2.2 X and Y addresses will be stored modulo 2^{15} and Z addresses modulo 2^{6}.

7.2.F.2.3 Replacement of X and Y addresses is denoted by including the addresses in parentheses without intervening VS. Replacement of the Z address is an error.

7.2.F.3 140- and 141-Instructions

7.2.F.3.1 The mode in these instructions is written after the function, separated from it by a full stop. Letters X, Y, S if required, are written following the mode, e.g.

140.1VS 141. 21YVS

The mode is written as one or two decimal digits, and must be in the range 0 to 31.

The addresses in these instructions require special treatment. This speciality is indicated by the decimal point and mode. If these are omitted the addresses will be treated normally.

7.2.F.3.2 The X-address in these instructions is normally written as zero.

7.2.F.3.3 The Y-address in the 140-instruction contains a peripheral name in the codes specified in section 7.2.C.2, or an address which is to be replaced, e.g.

```
140.1   0   *MT3
```

may be written, or

```
 14    A3   *MT3
140.1  0     (A3)
```

7 2.F.3.4 The Y-address in the 141-instruction must be in the range 0 to $2^{24}-1$. The least significant 15 bits will be stored in the Y-address, the least significant bit of the X-address (D23) will be zero and the remaining 9 bits of the Y-address will be added into the next 9 bits (D14-D22) of the X-address unless Y is replaced. Modification and replacement of the X-address cannot affect the 24—bit drum address.

7.2.G SYMBOLIC IDENTIFIERS, LABELS, DRUM LABELS AND EQUATIONS

7.2.G.1 Symbolic Identifiers

A Symbolic Identifier (also known as a Symbolic Address or more often just an Identifier) may appear in an address field or be used in other formats to represent an address or number whose value is not known at the time of writing. This is termed "using" the identifier.

An identifier may also appear as a label, or as the left hand side of an equation. This is termed "setting" the identifier.

7.2.G.1.1 A Symbolic Identifier is a set of up to fifteen letters and digits. The first character must be a letter, and if it is an A then the second character must also be a letter. A by itself constitutes an error, as does more than fifteen characters.

7.2.G.1.2 An identifier may be set as the sum or difference of other identifiers, numbers or addresses. The rank of an identifier is the number of datum points to be added. This will normally be 0 and 1 for numbers and addresses respectively, but provision is made for the rank to be anywhere in the range -2 to +5.

Examples of the rank of address fields are:

RANK +1 BUZZVS

RANK -1 BUZZ-JOE+30VS

RANK +3 BUZZ+JOE+30VS

RANK +5 BUZZ+JOE+A10+A11+3QVS

where BUZZ and JOE have ranks 1 and 2 respectively.

The various items, whether identifiers, addresses or numbers, are separated by + or - . VS terminates the address field but is not allowed between the separate items.

7.2.G.1.3 The Z-address must be of rank 0 or 1 and be in the range of -63 to +63 before addition of the datum point, otherwise a report will be printed. A report will be printed, optionally, if the rank of an X or Y address is not 0 or 1.

7.2.G.2 Labels and Drum Labels

An identifier may be set by an equation or by a label or drum label. An instruction or, more generally, any line, is labelled by writing the identifier before the word and terminating it by) or)). In the first case the identifier is set as the core-store address of the word it labels and in the second case as the drum address.

If there are many labels for one line it may be convenient to insert NL between the labels and the instruction or other line labelled; this is permissible.

Thus

```
                    75   JIM   0

              JOE)  14   A40   1
              JIM)  34   A40   6
```

requires identifiers set by labels as core store addresses, and less commonly

```
                BINGO))  +0.75
                         +0.77
                           .
                           .
                         +0.84
```

may be used with

```
                  141.1    0    BINGO
                  142    A30     20
```

to transfer a table of constants from the drum into the core store.

Any number of labels of either type may be attached to a word, each being terminated by its appropriate bracket(s).

A line on the printout of a program may correspond to several consecutive computer words. If a label is attached to such a line it will be interpreted as the address of the first of these words.

7.2.G.2.1 VS is not allowed between an identifier and) or)), but is ignored before a label and after the terminating) or)).

7.2.G.2.2 In the program

```
                  BUZZ)  00   A230    A30O   A4
                   JOE) 2048  A3
                  FIZZ)  85   A90     A50     7
```

where the line labelled JOE) represents, say, 4 computer words, Symbolic Input counts words to give

BUZZ + 5 = FIZZ.

7.2.G.2.3 A label will be output in basic language by Symbolic Input as Ln where n is a positive integer. If one writes L3, for example, in Symbolic Input it will still be stored as the next available Ln and hence in all probability it will not be stored as L3. However, if it is desired to store it as L3 the asterisk notation is used, i.e. *L3 in a Symbolic program will ensure that the label is stored as L3. Note that when L3 is then referred to in the program it will always have to be written *L3. L0 to L255 are left free for labels set in this way. Identifiers in Basic Input are described in section 7.1.2.

7.2.G.3 Equations

Equations are used to set identifiers which are not set by labels. They are written as shown in the following examples:

```
BUZZ = JOE-FRED+100+700
BINGO = 73219
```

The left hand side is terminated by = and the right hand side is terminated by NL. VS may appear on either side of the equality sign.

7.2.G.3.1 The right hand side may contain anything which can appear in an address field, excluding the parentheses notation.

7.2.G.3.2 The identifiers will be prefixed by the name of the routine in which they are read unless a complete prefix is written with the identifier. The system of prefixing is described in Section 7.2.L.3.

7.2.G.3.3 The rank of the right hand side may be between -2 and +5.

7.2.G.3.4 Three levels of setting an equation are available, optional or weak settings, normal equations and over—riding or strong equations for corrections.

The level of an equation is indicated by the use of parentheses, e.g.

```
(BUZZ = A30)      - optional or weak setting

 BUZZ = A30       - normal setting

)BUZZ = A30(      - strong or over-riding setting
```

No other combination of parentheses is permissible. Setting an identifier by using a label is equivalent to an equation at the "normal" level.

If an identifier is set at more than one level, it must be set only once at the strongest level used, all other settings will be ignored.

7.2.G.3.5 Equations of the form BUZZ = BUZZ+1 are not allowed. Where necessary, corrections can be carried out within the job tape or by using a correction tape (see section 7.2.D).

7.2.G.3.6 It must be possible to re-arrange the equations in a program in such a way that these equations can be solved successively (starting with the first and proceeding sequentially down the re-arranged list of equations) by nothing more than inserting previously obtained values for identifies appearing on the right hand sides.

For example, it is permissible in a program to include the following equations in any order,

```
L1 = A20+L2

L2 = L3+7

L3 = 402
```

because these may be re-arranged (by Symbolic Input) thus:

```
L3 = 402

L2 = L3+7

L1 = A20+L2
```

in which form they can be evaluated successively by simple substitution.

On the other hand, the equations

```
L10 = L20+A5

L20 = A25-L10
```

cannot be solved by the above process and therefore are impermissible.

7.2.H DIRECTIVES

A directive comprises a verb followed by a number of parameters and addresses, e.g.

START 4

The verb is set of letters; it can be terminated by VS or NL. VS before the verb is ignored. Addresses and parameters required by the directive are written after the verb, separated by VS and the final one terminated by NL.

Only the first three letters of a verb need be written as the remainder will be ignored by the compiler. Less than three letters is an error.

7.2.H.1 List of Available Directives

The section dealing with each directive is given following the full name of the directive.

ALTER	7.2.D.1	MONITOR	7.2.Q.1
		NAME	7.2.H.4
BASIC	7.2.C.2.1	NEW	7.2.C.2.4
CHAPTER	7.2.P.1.1	NORMAL	7.2.J.2
CORRECT	7.2.C.2.2	NOSIGNAL	7.2.H.3
		OCTAL	7.2.J.2.6
DOUBLELENGTH	7.2.J.2.1	PACKNUMBERS	7.2.J.2.7
END	7.2.C.2.9 and 7.2.D.1	REPORT	7.2.C.2.6
ENTER	7.2.H.2.2	PROGRAM	7.2.N.2.4
FLOATINGPOINT	7.2.J.2.2	READ	7.2.C.2.7
		RESERVE	7.2.C.2.3
		ROUTINE	7.2.C.1.1
		START	7.2.H.2.1
LIBRARY	7.2.M.2.2	STERLING	7.2.J.2.8
MACRO	7.2.N.2.4	TEXT	7.2.J.2.9
MASK	7.2.J.2.3	TIME	7.2.C.2.5
MIDPOINT	7.2.J.2.4	TRANSFER	7.2.P.2
MIXEDNUMBERS	7.2.J.2.5		

7.2.H.2 START and ENTER directives

7.2.H.2.1 The START directive is used to "label" the various starting points with a program. It is written before the first instruction to be obeyed on a separate line.

If there are numerous entry points they will be numbered from 0,

e.g. START VS 6 NL

A number must always be given and it must be preceded by VS.

7.2.H.2.2 The ENTER directive is used to enter a program at any place marked by a START directive.

Thus ENTER VS 0 NL at the end of a program tape will cause the program to be entered at the place predetermined by the directive START 0. ENTER directive needs a separator see 7.1.5.5.

Normally there will be only one entry point, but if a job is suspended the operator may wish to restart it at a different point, these points will then have to be preceded by START directives.

In this case the operator types on the Flexowriter,

e.g. BLOGGS VS ENTER VS 4 NL • .

in order to enter job BLOGGS at the instruction following the directive START VS 4 NL.

The entry points are recorded in the form of a list of chapter changing 150 instructions (section 5.3.50) each occupying two words in the drum store. When the program is entered in this way the entire chapter is brought into the core store.

7.2.H.3 NOSIGNAL

This directive may be written in the Job Tape or in the program, it causes signal instructions to be obeyed like ordinary instructions when the program is obeyed,

7.2.H.4 NAME

e.g. NAME FINS/ABC/FILE/REEL/-

This causes 8 words to contain the document name or document request name; one component in a word, the characters right justified.

e.g. NAME A/NAME/A.B.1/*

NAME A/REQ12/-/XYZ/-

NAME A/REQ12/-/XYZ/-/

The name is stored at the current value of the transfer addresses into 8 words. One component is stored in a word. Not all 8 components and corresponding solidi need be specified. For a specified component, the corresponding word contains the characters, if less than 8, right-justified. Unless the last character on the line is - (minus), then for non-specified components the corresponding words will be clear. If the last character is - (minus) then all subsequent words will contain character - (minus) at the l.s. end. Thus in the second example the last 4 words of the 8 will all contain - (minus) and in the last example the last 3 words will be clear. * and ** cause today's date and the time respectively to be stored as that component.

7.2.J NUMBERS - SPECIAL FORMATS

Symbolic Input recognises integers, fractions and sets of two 24-bit, four 12-bit or eight 6-bit packed numbers as one word. The special formats are introduced by directives, specified in section 7.2.J.2.

7.2.J.1 Numbers

An integer or fraction (or identifier) must be preceded by + or - . Each number must be separated from the next by VS or NL. Numbers are stored one per word, unless they are packed as specified in 7.2.J.1.3 or unless they follow a special directive.

Examples

```
+42VS-0.072VS-582NL
+0.0051NL
```

Identifiers may be used to represent integers, fractions or sets of packed numbers. In such cases the identifier must be preceded by + or - where it is used, but not where it is set, i.e. on the left-hand side of an equation. The sign is required to distinguish the identifier from a directive.

More generally, one may write anything which can appear on the right-hand side of an equation, provided that + or - is written first. An initial minus only affects the first item, not the whole word.

```
e.g.  -JOE+FRED+300

where JOE = 1000
      FRED = 2000
```

will be stored as 1300.

7.2.J.1.1

If x_F is a fraction then it must satisfy

$$-1.0 \leq x_F \leq 1 - \varepsilon$$

where $\varepsilon = 2^{-47}$. Fractions may be punched as -0.5, -.5, -1.0 or +00.6.

Excess digits in a fraction will be used for rounding.

7.2.J.1.2

If x_I is an integer then it must satisfy

$$-2^{47} \leq x_I \leq 2^{47}-1$$

A report will be printed if an integer or a fraction overflows.

7.2.J.1.3 It is possible to pack numbers in any of the following formats without a directive; these are:

(i) two 24-bit numbers, separated by a comma,

(ii) four 12-bit numbers, separated by 3 commas,

(iii) eight 6-bit numbers, separated by 7 commas,

any other packed number format is specified by the PACKEDNUMBERS directive described fully in section 7.2.J.2.7.

Input obtains details of the fields by counting the commas on each line. Note that these forms may change from word to word, i.e.

```
42, 4483 NL
43,484,485,44 NL
```

and will cause two 24-bit numbers and four 12-bit numbers to be stored in two words.

A comment will be printed if the number of commas is not 1, 3 or 7.

7.2.J.1.4 The following notes apply to all packed numbers.

VS may only appear immediately before or after a comma.

The field between the commas may contain anything that may be written in an address field except that the contents of any field of less than 15 bits must be independent of the D.P. and must not contain the name of a peripheral device.

A number is said to be in range in a k-bit field if its modulus can be represented in k bits, e.g. -63 is in range in a 6-bit field and is stored as +1 because the sign bit (i.e. the (k+1)st. bit) is not stored.

7.2.J.2 Directives for Special Formats

The formats described in this section will be dealt with by Input, but they must be introduced by a directive.

The introductory directive will be accepted if it complies with the notes in section 7.2.H. Input then continues to read in prescribed mode until the directive NORMAL or another directive occurs (except in the case of the TEXT directive, which is described fully in section 7.2.J.2.9).

Labels may be attached to words in special format but not to texts.

7.2.J.2.1 DOUBLELENGTH directive **

The directive is followed on the next line and lines by the numbers to be stored. Several numbers can be punched on one line, each separated by VS. Each number is stored in two consecutive words in standard double—length form, i.e. the l.s. word is non—negative. (See 2.0.2(e))

If the number is an integer, $x:_T$, then it must be in the range

$-2^{94} \leq x:_I \leq 2^{94}-1$

If the number is a fraction, $x:_F$, then it must be in the range

$-1.0 \leq x:_F \leq 1-2^{94}$ [this should be $1-2^{-94}$ Ed]

Each punched number must be preceded by a sign (plus or minus). A decimal point is not allowed in an integer. Zero and zeros before the decimal point in a fraction are optional. VS is not allowed between the sign and the last digit of the number. Several integers and/or fractions may be punched on a line, each separated by VS.

Sums and differences of numbers are not allowed nor are symbolic or basic identifiers.

Non significant zeros are allowed and excess digits in a fraction will be used for rounding.

A report will be made if the stored form overflows.

7.2.J.2.2 FLOATINGPOINT directive **

On the same line separated from the directive by VS may follow a signed integer which is used if block floating numbers are required.

The directive is followed on the next line and lines by the numbers to be stored. Only one number may be punched per line. Each number is stored in a word in standard floating-point form (see 2.0.2(d)). The permitted range is also stated in this section. These are the operands used by the Group 9 instructions.

Each number is punched as an argument and, if required, a decimal exponent. The argument must be a signed integer fraction or mixed number. If there is a decimal exponent it is punched as a signed integer on the same line as the argument and separated from the latter by a comma or by VS. If a comma is punched it can be preceded and/or followed by VS. VS is not allowed in any other position.

If block floating numbers are required then the exponent will be punched as a signed integer following the

directive and separated from it by VS. (Comma instead of VS is not allowed here.) If an exponent is also given opposite an argument the two will be added together, both exponents being signed. A block exponent can only be changed by punching a further FLO directive.

If no exponent is specified the exponent taken is zero by default.

For example, the numbers 12.34567, 345.6 and 0.0031 can thus be stored in the following ways.

```
FLO VS + 1
+1.234567
+3.456 VS +1
+3.1, -4
NOR
```

or

```
FLO
+12.34567
FLO VS -4
+3456000
+31
NOR
```

A report will be made if the stored form overflows or underflows.

7.2.J.2.3 MASK directive **

This directive is followed on the next line and lines by the masks required. Only one mask may be punched per line. The format of each line indicates the position of 0 and 1 bits required to be stored in a word. 0 or 1 is followed by -(minus) which is followed by the number of 0's or 1's required. Commas are used to separate the fields of 0's and 1's. The total number of bits specified must be 48. VS must not appear between digits or between a digit and -(minus). VS is allowed on either side of a comma.

For example, to form a mask containing 1 bits in positions D21 to D35 and 0's in the rest of the word then the following lines are used.

```
MAS
   0-21, 1-15, 0-12
NOR
```

A report will be made if the total number of bits is not 48.

7.2.J.2.4 MIDPOINT directive **

This directive is followed on the next line and lines by the mixed-numbers to be stored. Only one number may be punched per line. Each number is stored in two consecutive words in standard double-length mid-point form (see 2.0.2(e)iii), i.e. the integral part in the m.s. word and the fractional part in the l.s. word.

The number $x:_M$ must be in the range

$$-2^{47} \leq x:_M \leq 2^{47} - 2^{-47}$$

Each punched number must be preceded by a sign and in general will be a mixed number. For example +31.63 (though +2 and +2. and +.2 are allowed; +. is not allowed). VS is not allowed between the sign and the last digit of the number. Non significant zeros are allowed and excess digits in the fractional part will be used for rounding.

A report will be made if the stored form overflows.

7.2.J.2.5 MIXEDNUMBERS directive **

On the same line separated from the directive by VS follows a signed integer, n, which is used to store the number as explained below.

The directive is followed on the next line and lines by the number. Only one number may be punched per line. Each number is firstly read as a standard double-length mid-point number and is then shifted arithmetically up (47-n) places where n is specified with the directive. The m.s. half is then stored as the single-length number.

Each punched number must be preceded by a sign. VS is not allowed between the sign and the last digit of the number. Non significant zeros are allowed and excess digits in the fraction will be used for rounding.

If x_s is the number it must be in the range

$$-2^n \leq x_s \leq 2^n - 2^{(n-47)}$$

The point is considered to be (n+1) binary places from the m.s. end of the word.

If the stored number is regarded as a fraction F then the mixed number will be 2^nF.

Decimal scaling must be done by hand whilst writing the number.

For example to store $3.14159 \times 10^2 \times 2^{-12}$ the following lines are used.

```
MIX VS +12
+314.159
NOR
```

A report is made if the stored form overflows.

7.2.J.2.6 OCTAL directive **

The directive is followed on the next line and lines by the octal numbers to be stored. Only one octal number with a maximum of 16 octal digits is allowed per line.

Effectively each octal digit is stored in 3 bits. If fewer than 16 digits are punched they are stored at the l.s. end of the word, unless preceded by a point in which case they are stored at the m.s. end.

VS is ignored and thus could be punched after every fourth digit to assist checking.

A report will be made if the decimal digits 8 and 9 are present.

7.2.J.2.7 PACKEDNUMBERS directive **

This directive is used to pack anything that can be written in an address-field (see 7.2.F.2) into a word; the word being divided into at least two fields of specified lengths. The lengths of the fields in the word being

N_1 , N_2 , ...,N_i , ...,N_k bits

These field lengths are punched on the same line as the directive separated from it and each other by VS , i.e. as

PAC VS N_1 VS N_2 VS .... VS N_i VS .... VS N_k NL

where $\sum_{1}^{k} N_i$ must be 48 and at least two N_i's must be specified.

The "fields"; x_1, x_2, ..., x_i, ..., x_k (i.e. integers, symbolic identifiers, basic addresses, programmers peripheral names, drum addresses, sums and differences of these) that are to be packed into the word are punched on the next line separated by commas as

x_1, x_2, ..., x_i, ..., x_k

The resultant value of each x_i is stored in the corresponding field length (N_i) of the word.

If a field k bits long is used to hold an integer then the integer is within the range

$0 \leq n \leq 2^k-1$ if unsigned

and $-2^{k-1} \leq n \leq 2^{k-1}-1$ if signed (i.e. the m.s. bit of the field is used as a sign bit).

If the resultant value of each x_i is not in range then the integer is stored modulo the number of bits (N_i) of the specified field. For example if the field is 7 bits long then -127 will be stored as +1

Further lines containing other "fields", i.e. other x_i's, separated by commas will be stored into further words in the same length fields. The field lengths cannot be varied without a new PAC directive. (Note this differs from the rule applying to standard packed words as given in Section 7.2.J.1.3)

VS is not allowed in the "fields" between the commas.

A report will be made if the number of commas does not correspond to the number of fields specified (i.e. the number of commas will be one less than the number of fields specified).

7.2.J.2.8 STERLING directive **

This directive is followed on the next line and lines by the sterling quantities to be stored. Several sterling quantities may be punched per line, each separated by VS. Each sterling quantity is stored in a word as a binary integer which is the number of pence in this sterling quantity. The number of pence, d, must be in the range

$$-2^{47} \leq d \leq 2^{47}-1$$

i.e. the sterling quantity, S, must be in the range

-£586,406,201,480. 10. 8 ≤ S ≤ £586,406,201,480. 10. 7

Each sterling quantity A pounds B shillings and C pence must be punched in the form

sign A.B.C

The + (plus) sign may be omitted. Two points must always be punched. Non significant zeros in the pound, shilling and pence fields are allowed. The characters 10 and 11 may be used for pence. VS is not allowed between the sign (or first digit of the pounds) and the last digit of the pence.

7.2.J.2.9 TEXT directive **

On the same line as the directive and separated from it by VS may follow an integer which gives the number of new lines to be added to the end of the text.

The directive is followed on the next line and lines by the text to be stored. The text is terminated by a blank line (see 7.2.D); the NORMAL directive or another special format directive is not used as the terminator in this

case. The Symbolic Input Routine converts alpha-numeric characters including those right of a vertical bar into the standard internal code (see 5.6.1) and stores them into consecutive words, each word containing eight characters. The first word of the stored text contains in its l.s. half the total number of characters in the stored text.

Loading newline characters, excepting the first, between TEXT and the first printing character (excluding ER) are stored and do not terminate the text.

A number may be written after the directive, e.g.

TEXT VS 3

which gives the number of new line characters to be added to the end of the text. These texts are then in a form ready for output by the library text output routine.

7.2.L ROUTINES - PROGRAM ORGANIZATION

It is usual to divide a large program into routines, either to separate it into distinct functional parts or to make it possible for several programmers to work on it simultaneously or a combination of both.

Note that large programs are divided into chapters (section 7.2.P) but the division is independent of division into routines and (except in library subroutines, section 7.2.M) is not used to distinguish identifiers.

7.2.L.1 ROUTINE and END directives

7.2.L.1.1 Routines are named by writing the directive ROUTINE followed by VS and the routine name (a Symbolic identifier) which is terminated by / ,

e.g.

```
ROUTINE SORT/
```

7.2.L.1.2 The master program is an unnamed routine denoted where necessary by /.

7.2.L.1.3 A routine is terminated by the directive END, followed by VS Routine Name, e.g.

```
END SORT/
```

7.2.L.2 Notation for routines

The following notation will be used throughout section 7.2.L which is concerned solely with the way routines appear on the printout of the program tape and not with the way they are obeyed.

Letters B to K denote routines, L to Z identifiers and M.P. the master program. Underlining where necessary distinguishes that part of an expression written on the program sheet.

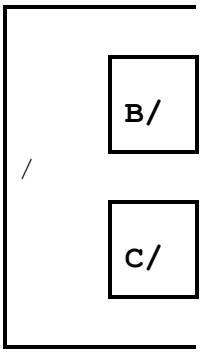
A complete routine e.g. B/ will be represented by

```
ROUTINE B/
Instructions etc. of  B/
END B/
```

or more concisely 

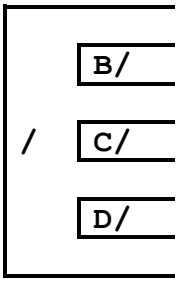
Thus a program may be denoted by

```
Instructions etc. of M.P.
ROU B/
Instructions etc. of B/
END B/
Instructions etc. of M.P.
ROU C/
Instructions etc. of C/
END C/
Instructions etc. of M.P.
```

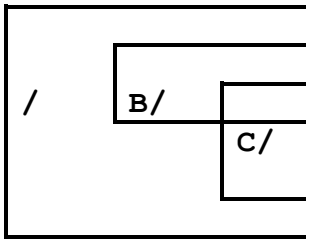


/ is the master program

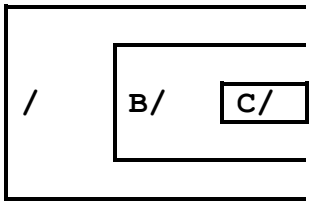
One routine must be terminated by its END directive before another ROUTINE directive is written, i.e. in a concise notation.



is permissible, but



and



are not.

7.2.L.3 References - the identifier and prefix system

The system of identifiers and prefixes provides the following facilities:

(a) simple local references (i.e. references to words in a current routine),

(b) simple reference to identifiers having the same meaning throughout the program (global identifiers),

(c) simple reference to a word in another routine,

(d) a method of incorporating a sub-program into a program which demands nothing of the author of the sub—program.

7.2.L.3.1 Prefixes

The full names of identifier P and routine G in a master program are respectively /P and /G/. The full names of P in routine J is /J/P, the expression /J/ is then called the prefix of identifier P.

It is not necessary normally to specify the complete prefix of an identifier or routine as this will be supplied by Input. A complete prefix will begin with /, hence all prefixes supplied by Input begin in this way. If the programmer begins a prefix with / Input will assume it is complete and it will never be altered in any way.

The details of these facilities are described in the following sections, all of which refer to the first example in this chapter. Note that underlining distinguishes characters that are actually written on the program sheet.

(i) "Local" references

If P is set in C/ (by label or equation), Input notes that /C/P is set. If P is then used in C/, Input notes that /C/P is used and the value of /C/P is given to P whenever it is used in C/.

For "local" references, no prefix will normally be written.

(ii) "Global" references

If P is set in the master program, Input notes that /P is set,, If then P is used in C/, Input reads this as /C/P. It then searches for a setting of /C/P and then, if unsuccessful, of /P. The first one found will be used.

For "global" references, no prefix will normally be written.

(iii) References to a word in another routine

If P is set in C/, i.e. /C/P is set, and it is required to refer to this in B/ or the master program, then C/P is written.

This system will give the required result assuming that all routine names are distinct.

7.2.M LIBRARY SUBROUTINES AND ASSEMBLY

7.2.M.1 Library Subroutines

A library subroutine will have its link L in an accumulator which is specified within the subroutine by an equation. In general, entry will be by an 86—instruction. If this is of 3—address type its Y-address may contain a parameter for the subroutine. The subroutine will exit by an instruction of the form

87 0 L

To ensure that a copy of the subroutine is in the chapter where it is to be used the LIBRARY directive (section 7.2.M.2.2(i)) or the 1086 macro-instruction (section 7.2.M.2.2(ii)) is used.

7.2.M.1.1 Accumulators

A subroutine needs accumulators for (i) finding operands and parameters, (ii) leaving results, and (iii) using as working space. All accumulators used by a library subroutine are referred to in the subroutine by identifiers, these are standardised as

:SR/X1, :SR/X2, ...

where SR is the routine name. X1 is optionally set equal to A1. If the address of an accumulator is dependent on that of another accumulator (e.g. in double-length working) the addresses are written in the form

:SR/X1, :SR/X1+1

It is open to the user to redefine the accumulators to be used if the conventional ones are inconvenient. A subroutine may need to do this with a sub-subroutine.

7.2.M.1.2 The entry point is labelled as :SR/E or, for multiple entry points, as :SR/E1, :SR/E2, ... , where SR is the routine name.

7.2.M.1.3 Preset parameters other than accumulator space are referred to within a library subroutine by standardised identifiers

:SR/P1, :SR/P2, ...

where SR is the routine name. The parameters are set optionally and can be changed by the user if desired.

7.2.M.1.4 If the library subroutine :C/ may be used independently and it is also used by library subroutine :B/ there are two cases to consider.

(i) If :C/ is used but not :B/, the preset parameters and accumulators may be reset in the usual way.

(ii) If :B/ is used, :C/ can be used independently but it is not possible to set the accumulators and parameters of :C/ independently. The combined accumulators of :B/ and :C/ may be set, since they appear to be accumulators of B/.

7.2.M.2 Assembly - the LIBRARY directive and the 1086 macro

7.2.M.2.1 The name of a library subroutine is always preceded by a colon. The colon may be preceded by a chapter name. No VS may appear between a chapter name and the colon or between the colon and the subroutine name.

7.2.M.2.2 There are two ways of ensuring that a library subroutine is included in a chapter (note that section 7.2.N deals with the concept of a macro-instruction).

(i) the LIBRARY directive which is written

LIBRARY VS :S/R Name/

This will put a copy of the named subroutine in the current chapter starting at the current transfer address and will increase the transfer address by the length of the subroutine.

(ii) the 1086 macro-instruction which is written

1086 VS :S/R Name/Entry Point

This will replace the macro-instruction with the instruction

86 VS : Name/Entry point VS :Name/L

where L is the link and will put a copy of the named subroutine at the end of the current chapter.

The colon must not be preceded by a chapter name in the LIBRARY directive or in the 1086 instruction.

Only one copy of a subroutine may be stored in one chapter. To include a parameter in the Y-address of the 86 instruction write

1086 VS :S/R Name/Entry point VS Parameter NL

and then a 3-address 86-instruction will be compiled.

7.2.M.2.3 If an identifier C3:SIN/X1 is written anywhere it means identifier X1 of the copy of the subroutine named :SIN/ stored in chapter C3.

An error is reported if no copy is loaded in the specified chapter.

7.2.M.2.4 An identifier :SIN/X1 means identifier X1 of Subroutine :SIN/ in the current chapter. An error is reported if no copy is loaded in the current chapter.

7.2.M.2.5 Preset parameters and addresses of accumulators used by a library subroutine are reset by equations of the form

```
C3 : SIN/P2 = 28

   : SIN/X1 = A4
```

Note that the first equation has the same meaning wherever it is written.

These equations can only be read once at the highest level for each copy of a library subroutine.

7.2.M.2.6 A library S/R is classified as a library S/R of a specified chapter.

7.2.M.2.7 Auxiliary chapters

Those are used with multi-chapter library subroutines. They enable the main chapter of the library subroutine to be incorporated in the chapter from which it was called and incorporate the remaining chapters of the library subroutine into the main program as secondary or auxiliary chapters.

Each chapter of a multi-chapter library subroutine must have a different name, but only the name of the main chapter of a library subroutine is available to the Object programmer. Auxiliary chapters may not be used with the chapter changing macros, as their location is only known to the main chapter of the library subroutines.

7.2.M.2.8 It is not possible to have in the same chapter, two or more library subroutines which each call the same library subroutine. If this happens then Symbolic or Basic may report reset identifiers. They may, however, be used in different chapters.

7.2.N MACRO-INSTRUCTIONS AND THE + AND MAXIMUM NOTATIONS

A macro-instruction (or 'macro') is an instruction which the programmer thinks of as one concept but which cannot necessarily be expressed in one machine instruction. Function numbers of four digits are reserved for these, and the layout of addresses may be non-standard.

Macro-instructions 1000-1999 are reserved for standard-macros, i.e. those with a fixed meaning, and numbers 2000-2999 denote programmer-defined macros**, i.e. those given a meaning by the programmer for a particular program only. The numbering of macros need not be dense.

7.2.N.1 Standard macros

The standard macros are described in their relevant sections as follows:

1000)	
1001)	Section 7.2.P.5.1
1002)	
1003)	
1086	Section 7.2.M

7.2.N.2 Private (or Programmer's) macros**

7.2.N.2.1 To clarify the description in this section we will set up an "example macro".

A macro is required which will place the greater of two numbers in A1 unless the first number is less than the number in A2, in which case it is to jump to BUZZ. The two numbers are members of a set stored in A400-A410, the exact pair required is to be specified each time the macro is used.

A suitable set of instructions (if JOE and FRED are the addresses of the numbers to be used) is

```
      64    BUZZ  JOE   A2
      04    A1    JOE
      64    NEXT  FRED  A1
      04    A1    FRED
NEXT)
```

7.2.N.2.2 There are three types of addresses in a macro; they are:

(i) fixed addresses, which are the same whenever the macro is used, e.g. A1, A2 and BUZZ in 7.2.N.2.1

(ii) addresses relative to the first order of the macro.

(iii) variable addresses, which are to be defined each time the macro is used in the program, e.g. JOE and FRED in 7.2.N.2.1.

Fixed addresses may be either primitive addresses or identifiers. Each time a macro is used the prefixes to the identifiers are set, hence each use of a macro may give rise to different prefixing.

Addresses relative to the first order of the macro are dealt with using the + notation, which is described in section 7.2.N.3.1.

Variable addresses are assigned numbers 1, 2, 3, ... (the numbering must be dense) and are referred to in the definition of a macro as =1, =2, =3 ... They are set in the program which uses the macro, as addresses following in the same line as the macro function. If our example has function 2048 and it is desired that JOE=A400 and FRED=A401 the instruction

```
2048   A400   A401
```

is used where JOE and FRED are defined in the macro-definition as =1 and =2 respectively. Equations are not allowed in a macro.

7.2.N.2.3 Labels may be attached to instructions in the definition of a macro.

A macro must be pre-defined i.e. the definition must be read before it is used by a program.

7.2.N.2.4 A macro is defined using the directive MACRO. The example in 7.2.N.2.1 would be written (with the function 2048 say) as follows:

```
MACRO 2048
64    BUZZ    =1    A2
04    A1      =1
64    2 +     =2    A1
04    A1      =2
PROGRAM
```

The terminating directive is PROGRAM, the directive NORMAL is not used as it is possible to use special formats (terminated by NORMAL) within a macro. The example in the following section shows this facility.

7.2.N.2.5 Variable addresses in a definition may be replaced and address fields in a macro definition may contain items with variable addresses or combinations thereof.

To define a one-address macro-instruction with function 2100 to put one of three specified masks into A10 we can write:

```
MACRO 2100
04    A10   =1+2+
75    4+    0
MASK
0-10,1-38
0-12,1-20,0-16
1-20,0-28
NORMAL
PROGRAM
```

This macro is used in the program by the instruction

2100 VS n

where n = 0, 1 or 2 depending on which mask is required in A10.

7.2.N.2.6 Thus a macro is written in the program in a similar manner to an ordinary instruction except that the number of addresses is limited only by the number which can be included on the same line as the function and one line on the program sheet will be turned into several instructions.

The first address will be inserted wherever =1 appears and so on.

7.2.N.2.7 The last definition read by Input will be used.

7.2.N.2.8 Standard macros cannot be defined by the user.

7.2.N.2.9 A macro-instruction may not be used in a macro-definition.

7.2.N.3 + and maximum notations

7.2.N.3.1 A single + at the end of an address will cause the transfer address to be added. This includes the D.P. For example, loops may be written

```
00    A1    A2
34    A2    4
80    -2+   A2    A3
```

In any field, +0 means zero, 0+ means the transfer address and + is an error.

Addresses terminated by + are called relative addresses, they should be used very sparingly, because

(a) a program including them cannot easily be corrected.

(b) a macro may occupy several words of storage but only one line on the program.

(c) one can easily miscount.

7.2.N.3.2 The notation >(BUZZ, A200, JOE+50) represents the greatest of the individual items separated by the commas.

It is mainly used for determining the maximum amount of core storage required for program storage.

e.g. >(&CH1, &CH2, &CH3)

represents the amount of storage required for the longest of chapters 1, 2 and 3. The &-notation is described in section 7.2.P.4.

The maximum notation may not be used within another maximum notation.

The ranks of the items to be compared must be the same.

The notation may appear wherever an identifier can be used. Combinations such as

$$>(B_1, B_2, B_3) + 20$$

and $>(B_1, B_2, B_3) - >(C_1, C_2, C_3)$

are quite permissible.

Any item within the parentheses which has a negative value counts as zero.

VS may appear only immediately preceding or following a comma.

7.2.P CHAPTERS AND CHAPTER CHANGING

Large programs are divided into chapters, each of which is brought to the working store (to be obeyed or referred to) in a single instruction.

No 'automatic' loading facilities are provided: if the user refers, in chapter C, to something not in chapter C, it is his responsibility to see that this item is in the core store.

The division of a program into chapters is independent of its division into routines, i.e. a chapter may contain several routines or a routine may contain several chapters.

7.2.P.1 CHAPTER and ENTER directives

7.2.P.1.1 Chapters are introduced by the CHAPTER directive in the format.

CHAPTER VS Name of Chapter NL

The chapter name is a set of, at the most, 15 letters and digits as specified in section 7.2.G.1.1.

If it is the first chapter, it will be stored immediately after the chapter change words (section 7.2.P.5). Other chapters will be stored immediately after the chapters preceding them on tape.

7.2.P.1.2 Chapters are terminated by a new CHAPTER directive, or ENTER directive (or END see 7.C.2.9(iv))

7.2.P.2 TRANSFER directive.

Chapters are arranged using a TRANSFER directive. The layout is carried out in the Core-Store and the completed chapter is transferred to the drum.

The directive is written

TRANSFER VS Core-Store address NL

which sets the transfer address to the given address. For example:

```
TRA  A100

TRA  BUZZ+3
```

Note that a maximum of 14 transfer directives are allowed in a chapter.

It is allowed to write, for example

```
TRA  +20+
```

which leaves 20 registers unused on both drum and core. This type does not count towards the maximum of 14.

7.2.P.2.1 The CHAPTER directive sets the transfer address to be A64.

7.2.P.3 Layout of a chapter

The layout of a chapter will be:

(i) Instructions etc. in positions specified by the TRANSFER directive. During the process the T.A.(transfer address) reaches a maximum value K say.

(ii) Library subroutines will follow (i), starting at K, unless the positions of the S/R's have been specified as described in section 7.2.M.2.

7.2.P.4 The ampersand notation

Due to 7.2.P.3(ii) above, finding the length of a chapter may be tedious. The notation

& Name of chapter

denotes the length of the named chapter.

Thus if the first word of chapter CH4 is labelled BUZZ, then BUZZ+&CH4 is the address of the first word which can be used for data.

7.2.P.4.1 VS must not appear between & and the chapter name.

7.2.P.4.2 The & notation can be used wherever an identifier can be used.

7.2.P.5 Chapter-changing macros

Chapters will normally be transferred to the core store by a standard macro since if the user loads and enters a chapter using a 141/142 combination plus a jump, the jump may be overwritten by the transfer.

7.2.P.5.1 The chapter-changing macros are written in the program as follows:

(i) 1000 VS Name of chapter VS ADDR NL
Bring the chapter down from the drum and jump to the working store address ADDR.

(ii) 1001 VS Name of chapter NL
Bring the chapter down from the drum, then continue with the next instruction.

(iii) 1002 VS Name of chapter VS ADDR1 VS ADDR2 VS ADDR3 NL
Bring down that part of the named chapter between addresses ADDR1 and ADDR2 and enter it at ADDR3. These three addresses are working store ones.

(iv) 1003 VS Name of chapter VS ADDR1 VS ADDR2 NL
Proceed was with 1002 but do not enter the chapter, instead continue with the next instruction.

Each of these macro-instructions consists of two words.

7.2.Q MONITORING

There are three levels of reporting during Input, these are introduced by the REPORT directive (see 7.2.C.2.6.)

Monitoring is available in the following events:

*0WN	Own dealing with failure
*SIG	Signals
*JUM	Jumps
*FOV	Floating point overflow
*OVR	Overflow
*DRU or peripheral name	Peripheral transfers
*IMP	Impermissible operand
*PFP or *PFN	Program failure
*TIM	Timer overflow
*UNR	Unrounded floating-point
*URQ	Urgency
*QUI	Quick jumps
*WEA	Weak reservations

Monitoring is obtained with the MONITOR directive e.g.

MONITOR VS Event VS Style

The event is written in the coded form indicated above, the style is an integer between 0 and 5, or a core-store address (this is known as Style 7) in which case action on this event is to jump to the specified location. (see 5.2 for more details).

This directive may appear anywhere on the job tape or program tape.

7.2.R MESSAGES

7.2.R.1 Flexowriter Messages

7.2.R.1.1 Initial Entry to the Compiler.

i) When the compiler is entered initially the question "STORE ON TAPE OR DRUM?" is printed. This is where the compiled program is stored during compilation„

The operator's reply is TAPE or DRUM using ANSWER directive.

7.2.R.1.2 Message during compilation

i) The compiler reserve more drum as required and if no more is available the message
NO DRUM
is printed. The job will continue when more drum is available and no operator action is required.

ii) If the Orion Library is required the normal load document message is printed. (This message will be repeated if the document reserved proves to have non-standard format for the Orion Library. The non-standard document will be automatically relinquished.) If any subroutines are required which do not appear on the library tape a message of the form
ROUTINE :SQRT NOT FOUND
is printed for each routine that is missing. This message is also output on the monitoring peripheral if one is reserved for the job. The job is then abolished.

iii) If the input medium is five-track paper tape and the punching conventions are violated the job is abolished after the message
*ERROR ON 5-TRACK TAPE. PLEASE MARK.
is output on the Flexowriter. This message corresponds to error number 66 which is specified in section 7.2.R.2.2(v). The 5-track paper tape conventions for Symbolic Input are specified in section 7.2.E.2.

7.2.R.1.3 Message when compilation is finished.

i) If no errors are found on a load-and go run the message
COMPILATION COMPLETE.
is printed immediately before Basic Input commences reading the compiled program. At this point the input peripheral is relinquished.

ii) If there are any errors on a load-and-go run the message
ERRORS FOUND. LOAD & GO ILLEGAL.
is printed. This message also appears on the monitoring peripheral if one is reserved for the job. The job is then abolished.

iii) If a BASIC directive has been read and the job should output a compiled program the message
ERRORS FOUND. NO COMPILATION.
is printed if errors are found. This message is also output or the monitoring peripheral. The job is immediately abolished and no compiled program is produced.

iv) If any identifiers have not been set then Symbolic outputs a list of these whatever the report level, on job's monitoring peripheral (output will be to the Flexowriter if the job hasn't one). Unset identifiers are not treated as errors and the program will be entered or output etc.

An example of this output is given below.

```
UNSET   IDENTIFIERS

L257    SQUARE/WS

L296

L301    TWO:SQRT/E1

L302    INITA:SQRT/E1

L345    WS

L346    (CHAPTER INITA LENGTH)

L391    (CHAPTER TWO CORE START)

L406    BLOGGS/WS

L413    (CHAPTER INITA DRUM START)
```

7.2.R.2 Monitoring Peripheral Messages

7.2.R.2.1

i) The messages
ERRORS POUND. LOAD & GO ILLEGAL
ERRORS FOUND. NO COMPILATION
on the monitoring peripheral corresponds to similar ones on the flexowriter. These are specified on section 7.2.R.1.3. They are not terminated by a full stop on the monitoring peripheral.

ii) An error in a corrections document results in a message of the form
ERROR n IN CORRECTIONS
(where n is an integer) being output on the monitoring peripheral. Depending on n (which is the error-number) further lines of information are output as follows.

n = 1 Error in an ALTER directive. A second message gives the line-image of the ALTER directive.

n = 2 Caused by one correction deleting a line which is referred to by a second correction. Both will be ignored. A second message gives the line-image of the original line of symbolic program.

n = 3 Caused by more than one correction attempting to alter the same line or group of consecutive lines. All corrections of this type will be ignored and the line-image of the original Symbolic program line is output.

n = 4 When the end of the symbolic program is reached this message is output for each unapplied correction. The second line of information gives the position of the correction as originally given in the ALTER directive as follows:

n AFTER BUZZ

where n is the line-count from symbolic identifier BUZZ.

n = 5 An attempt to apply one or more corrections after the end of the program. No further information is given. e.g. If the correction ALTER 3 After LOOP given to correct

```
LOOP) 75 BEGIN 0
      ENTER 0
```

These errors can result in further action depending on the level of the REPORT directive. This is specified in paragraph (v) of section 7.2.R.2.2.

7.2.R.2.2 REPORT directive message.

The output on each REPORT level is specified in section 7.2.C.2.6. All reports are output on the specified peripheral (i.e. the monitoring peripheral.) The form of the output message is as follows.

i) When a REPORT directive (level 1 or 2) is read a DOCUMENT directive is output on the specified peripheral with the document name requested in the REPORT directive.

ii) The line-image of each directive in the Symbolic program is output in level 1 and in level 2 the Basic Input interpretation of each directive is also given, where practical. This extra report is given for the following directives:

END, ENTER, MONITOR, NAME, NEW, NOSIGNAL, REPORT, RESERVE and TIME.

The END directive here is the directive which terminates a Program and not that which terminates a routine.

If the directive is in a programmers' macro the line-image is replaced by two lines of information is specified in paragraph (v) of this section.

iii) If the rank of a field is not 0 or 1 a message of the form

```
RANK IS -1
+A3-A4-A5
```

is output level 2. The second line gives the line-image of the field in question. If the field is in a programmer's macro the line-image of the field is replaced by two lines of information as specified in paragraph (v) of this section.

iv) The Basic Input interpretation of all the symbolic identifiers that are referred to is output in level 2 under the heading

SYMBOLIC IDENTIFIERS

The identifiers *L0 to *L255 are never output as they are not altered apart from the omission of the asterisk.

The Symbolic Input Compiler generates Basic Input identifiers, but most of these can be of no concern to the Symbolic programmer; hence they are output in the Basic Input form only. However three types of these identifiers are of some value, they refer to the core starting address, the drum starting address and the length of each chapter. In these cases the meaning is output as shown below.

Identifiers in the main program are given in the form LINK, those referring to a routine in the form ROUTINE/IDENTIFIER and those referring to a library subroutine are given in the form THREE:EXP/E1 where the library subroutine EXP is in chapter THREE.

An example of this output is given below.

```
SYMBOLIC IDENTIFIERS
L257  SQUARE/WS
L301  TWO:SQRT/E1
L302  INITA:SQRT/E1
```

v) Error Reports.

An error, other than in Corrections, is reported in the form of a message 'ERROR p' where p is an integer. It is followed normally by the line-image of the offending line of input. The error indicator is set and no further program is stored. The Compiler ignores the rest of the line, but further program is checked unless

i) the 15th error has been found

or ii) an error is found in a READ, BASIC, CORRECT, WITH, GIVING or ENTER directive.

or iii) the report level is 0,

in which cases the job is abolished. The error numbers are listed in paragraph (vi) below.

The line-image of the offending line of input is not given if the error has arisen from a line in a programmer's macro. An error in a programmer's macro is not found until the macro is used, hence the same error may be reported each time the macro is used. The error report will give a second line of information of the form

LINE m of MACRO

where m is an integer which is 1 for the first line of the macro, 2 for the second and so on. The third line gives the line image of the line which called for the macro.

Error number 65 introduces all reset identifiers. These are given in the same format as that used under the heading SYMBOLIC IDENTIFIERS described in paragraph (iv) above. Error numbers 66, 67, 68, 69 have no line-image and the job is abolished. Error 66 causes a Flexowriter message as specified in section 7.2.R.1.2(iii).

An example of error printing is given below:

```
ERROR 06

ENT) 04X (*L301) STORE WS+1

ERROR 19

LINE 3 OF MACRO

2001 (WORK) GNC/E .

ERROR 65

L402 GNC/LINK

L573 THREE:SQRT/XI

ERROR 66
```

vi) Error numbers.

The following list defines all the error-numbers excepting the errors in corrections which are defined in section 7.2.R.2(ii)

00 Impermissible first character of a line or storable word.
01 Integer overflow in any field (incl. special format directives.)
02 Wrong character in a field.
03 Wrong character terminating a field.
04 Format error in a symbolic identifier.
05 Datum point in chapter, routine or library subroutine name.
06 *Ln, where n is an integer greater than 255.
07 Peripheral name with device number greater than 31.
08 Impermissible field in > notation.
09 Field overflow.
10 Non-zero rank in a field of less than fifteen bits.
15 Wrong character in function (including mode X,Y and S) of an instruction.
16 Function or mode in an instruction out of range.
17 Impermissible address field in an instruction.
18 Wrong character terminating address field in an instruction.
19 Z-address field with rank not 0 or 1.
23 Missing field in a quantity.
24 Impermissible quantity field.
25 Number of packed quantities not 1,2,4 or 8
26 Fraction not -1.0 when integral part is non-zero.
30 Impermissible left-hand side in an equation.
31 No right-hand side in an equation.
32 Impermissible right-hand side in an equation.
35 Number of TRA directives (not +10+ type) greater than 14.
36 Format error in a directive.
37 Impermissible document name or document request name.
38 Format error in a special format sequence.
39 Unknown directive (including NORMAL if it is found other than when terminating a special format sequence).
40 Transfer-address of rank other than 1.
41 Numerical value of a transfer-address negative.
42 First CHAPTER directive securing after storable words.
43 Job-tape directive in program.
44 More than one CORRECT directive.
45 Both BASIC and GIVING directive encountered.
46 GIVING directive not preceded by CORRECT and WITH directives.
47 RESERVE Peripheral Name THIS for a device that is already reserved, as for a device of a different type from the current input peripheral.
48 LIBRARY directive duplicated in the same chapter.
49 Illegal peripheral name in BASIC, REPORT or GIVING directive.
50 No END directive at end of job tape after CORRECT directive read.
51 NO START directive.
52 Wrong character in a 1086 macro.
53 Impermissible parameter in a 1086 macro.
54 Wrong character in a chapter-change macro.
55 Wrong number of address fields in a chapter-change macro.
56 Impermissible address field in a chapter-change macro.

60 Programmer's nacre used when not pre-defined.
61 Undefined variable address used in a programmer's macro.
65 Reset identifiers.
66 Violation of 5-track paper tape punching conventions.
67 No terminating Directive in input magnetic tape.
68 Deck fail or repeated read failure on input magnetic tape.
69 Insufficient core store reserved.

If errors have occurred and the run is abolished early then the "abolish number" is the number of errors detected.

If errors have occurred then Symbolic does not implement any Assembly facilities (i.e. 1086 macro and LIBRARY directives).

7.2.5 Dumping Facility

While Symbolic is compiling it is possible to type ENTER 1 on the Flexowriter which causes a dump to be made at some stage. Thus if a machine failure occurs, it will be possible to restart the job at the last dump point. Incorporated in the Symbolic compiler is the library subroutine :DUMP/.

Symbolic dumps when it is convenient to do so. If the symbolic program is in the process of being read in when ENT 1 is typed, then Symbolic makes a note of this and dumps when a READ, END or ENTER directive is read. If the symbolic program has already been read in and is in the "winding up" stage when ENT 1 is typed then the dump takes place immediately. If a dump is taking place or the compiled program is being output then ENT 1 will have no effect except that the message ENT RES.VIOL FLX is output on the Flexowriter; Symbolic continues. Note that after the message COMPILATION COMPLETE. ENT 1 must not by typed.

If there has been a dump then Symbolic automatically dumps just before the compiled program is output, or just before Basic Input is asked to read if a load and go run.

If the mode is "storing on TAPE" the dump will be made on the magnetic tape already reserved for the job, (this is usually *MT31) whereas if the mode is "storing on DRUM" then a scratch tape will be requested (usually as *MT31) - a pre-addressed tape may not be used as a dump tape and it is advisable to use a 3600 ft. tape.

After a dump has been made, all slow peripherals will be disengaged. The operator firstly marks the reading position of any paper tape or card input with the dump number and then engages the devices.

The dump information is output on the Flexowriter only e.g.

```
DUMP NO. 8 DRUM 9472 CORE 2544 DATUM 512
DUMP ON document name of dump tape.
```

If a failure occurs, then restoring is by the utility routine ORION/SYSTEM/RESTORE/- (see Orion Library specifications). A write-permit ring must be fitted to the dump tape when the program is restored, so that Symbolic can automatically dump. A job tape for restoring for the dump point given above for example is

```
JOB REST  2544  512
V3 = 8
V4 = *MT31
RES *MT31  document req. name of dump tape
READ ORION/SYSTEM/RESTORE/-
```

7.3 Binary and Map Input

7.3.1 The Program

Binary-and-Map Input provides a means for fast input of programs. It is the nearest to binary input that is provided on Orion.

It is at present available on magnetic tape, and 7-track paper tape.

7.3.1.1 The B-directive (see 7.1.4.2.16) gives the number of words in Binary-and-Map language that follow. On seven track paper tape the B-directive is followed by two newlines, after which the binary and map information begins (this includes a checksum); on magnetic tape the binary and map information begins in the next block after that containing the B-directive. Blocks on magnetic tape follow the conventions of section 6.

When Binary-and-Map Input has completed its work, it returns to Basic Input to continue reading the tape.

7.3.2 The Language

7.3.2.1 Binary-and-Map language consists of any number of units consisting of one word of map followed by twelve words of program, and one unit consisting of one word of map and up to twelve words of program. On seven track paper tape this is followed by a checksum.

7.3.2.2 The map word in each unit specifies how each word of program in that unit is to be relativized, i.e. whether or not the datum point is to be added in to the X or Y fields.

It is laid out as follows:-

DO-D23	must be clear
D46	1 if X of 1st word of program is to be relativized
D47	1 if Y of 1st word of program is to be relativized
D44	1 if X of 2nd word of program is to be relativized
D45	1 if Y of 2nd word of program is to be relativized
.	
.	
.	
D24	1 if X of 12th word of program is to be relativized
D25	1 if Y of 12th word of program is to be relativized.

It is only possible to have addresses of rank 1 or 0 in binary-and-map language, and rank 1 addresses in the X-or Y-address positions of the word only.

7.3.2.3 The program words comprise the words that are to be stored, in binary, prior to adding in the datum point where necessary. Binary-and-Map Input begins storing words on the drum, starting at the current value of Basic Input's V1, and when it has completed its work, leaves V1 and V2 increased by the number of words stored.

7.3.3 On seven track paper tape only, there is a checksum, which is formed by adding every word of the binary and map information (program and map words) to a location initially clear, and after each addition forming the not-equivalent with OVR. I.E., for each word INFO of the information.

```
00  CHECKSUM  INFO
27  CHECKSUM  4
```

is obeyed. The number in the B-directive includes this checksum.

7.3.4 When Basic Input resumes control from Binary-and-Map Input, all settings of identifiers, forward references not yet filled in, monitor conditions, calls, etc., still apply. On 7-track paper tape Basic Input resumes reading with the next character on the tape; on magnetic tape with the first block after the last one containing binary-and-map information.

7.3.5 A library program called Mapper has been provided to convert programs from Basic Language to binary-and-map form.